UCAM

UNIVERSIDAD CATÓLICA DE MURCIA

ESCUELA INTERNACIONAL DE DOCTORADO

Programa de Doctorado en Tecnologías de la Computación e Ingeniería Ambiental

Optimización de algoritmos bioinspirados en sistemas heterogéneos CPU-GPU.

Autor:
Antonio Llanes Castro

Directores:
Dr. D. José Mª Cecilia Canales
Dr. D. Horacio E. Pérez Sánchez
Dra. Dña. Antonia Mª Sánchez Pérez

Murcia, septiembre de 2016

UCAM
UNIVERSIDAD CATÓLICA DE MURCIA

ESCUELA INTERNACIONAL DE DOCTORADO

Programa de Doctorado en Tecnologías de la Computación e Ingeniería Ambiental

Optimización de algoritmos bioinspirados en sistemas heterogéneos CPU-GPU.

Autor:
Antonio Llanes Castro

Directores:
Dr. D. José Mª Cecilia Canales
Dr. D. Horacio E. Pérez Sánchez
Dra. Dña. Antonia Mª Sánchez Pérez

Murcia, septiembre de 2016

UCAM

UNIVERSIDAD CATÓLICA DE MURCIA

AUTORIZACIÓN DEL DIRECTOR DE LA TESIS PARA SU PRESENTACIÓN

El Dr. D. José María Cecilia Canales, el Dr. D. Horacio Emilio Pérez Sánchez, y la Dra. Dña. Antonia María Sánchez Pérez como Directores(1) de la Tesis Doctoral titulada Optimización de algoritmos bioinspirados en sistemas heterogéneos CPU-GPU, realizada por D. Antonio Llanes Castro en la Escuela Internacional de Doctorado (EIDUCAM), **autoriza su presentación a trámite en su modalidad de compendio** dado que reúne las condiciones necesarias para su defensa.

Lo que firmo, para dar cumplimiento a los Reales Decretos 99/2011, 1393/2007, 56/2005 y 778/98, en Murcia a 30 de septiembre de 2016.

(1) Si la Tesis está dirigida por más de un Director tienen que constar y firmar ambos.

UCAM EIDUCAM Escuela Internacional de Doctorado

A mi familia,
mi motivación para todo.

Agradecimientos

Me es imposible comenzar el trabajo sin antes agradecer a todos los que han hecho posible que mi dedicación no afectase en demasía a mi razón:

A Delia, a la que amo y me ama,
sin pedir nada o casi nada,
que no es lo mismo, pero es igual.

A mis padres, y a Sole y Robertos,
a los que amo y me aman
sin pedir nada o casi nada,
que no es lo mismo, pero es igual.

A Jesús y Raúl, a los que amo y me aman,
pidiendo mucho o pidiendo un poco,
que no es lo mismo, pero da igual.

A todos los que por estar cerca de mí, contribuyeron de alguna manera a aliviar mi carga. A mis compañeros, que soportaron durante muchas horas mi dedicación a la tesis, dejando incluso ser inmiscuidos en estos temas. A Bueno, por su compañía, buenos consejos y risas. A Jesús, por sus clases sobre matemáticas, estadística, hijos, historia, etc. A Mada, porque su compañía en el camino hizo más llevadera la travesía. A Muñoz, por haber sido un compañero excepcional, un director más en la sombra. A Pereñíguez, si de alguien se puede aprender, es de tí. A FArcas, gracias por hacer que cale en mí tu espíritu crítico. A Alberto, gracias por haber sabido perdonar mis idas y venidas, sin pedir ni siquiera un porqué. A Miguel Ángel, por ser un espejo en el que mirarse. A Jose Luís, que siempre da sosiego y razón donde es necesario. A Raquel, una persona que puede llevar ella sola un grado completo. A Mayte, mi único apoyo en el balcón durante dos años. A Baldo, un aire fresco en CUDA. Y muy especialmente a Belén, por creer que somos capaces, casi más que nosotros mismos, y a Luz, porque soportarnos a todos no es tarea baladí.

A Horacio, tu rigor científico y dedicación es un ejemplo a seguir, alguna vez espero llegar a la mitad de la mitad de tu camino en mi carrera de investigación. A Onia, mayor sacrificio en el trabajo, dedicación y predisposición es insuperable, un ideal que se encuentra justo al lado de mí. Con gente como tú, ir a trabajar se hace mucho más fácil. A Chema, aunque sus promesas fueron derivando en muchos quebraderos de cabeza no planteados en un principio, no me queda otro remedio que darle la razón. Además, su implicación, sus consejos y su insistencia han posibilitado que este trabajo sea mucho más llevadero. De un tiempo a esta parte, para mí el nombre de José M.Cecilia no es únicamente un referente en investigación, sino también en muchos otros aspectos de la vida que transmite con su dedicación y trabajo. Siento que me llevo un amigo, no un director.

A todos, no tengo otra palabra que defina mejor lo que siento;
Gracias.

Índice

Esta tesis es un compendio de publicaciones científicas, a continuación se muestran las referencias completas de los artículos que conforman la tesis.

Título	*Comparative evaluation of platforms for parallel Ant Colony Optimization*
Revista	The Journal of Supercomputing
Páginas	318–329
Año	2014
Estado	Publicado

Detalles de la revista *The Journal of Supercomputing*
Redactor jefe: Hamid Arabnia
ISSN: 0920-8542 (versión impresa)
ISSN: 1573-0484 (versión electrónica)
Editorial: Springer
Factor de impacto (2015): 1.088
Categoría: *Computer Science, Hardware and Architecture*
Página Web: `http://www.springerlink.com/openurl.asp?genre=journal&issn=0920-8542`

Autores – Afiliación	
Nombre	**Dr. Ginés David Guerrero**
Centro	Director de Tecnología del Laboratorio Nacional de Computación de Alto Rendimiento de Chile
Nombre	**Dr. José M. Cecilia**
Universidad	Universidad Católica de Murcia
Nombre	**Antonio Llanes Castro**
Universidad	Universidad Católica de Murcia
Nombre	**Dr. José Manuel García**
Universidad	Universidad de Murcia
Nombre	**Dr. Martyn Amos**
Universidad	Manchester Metropolitan University
Nombre	**Dr. Manuel Ujaldón**
Universidad	Universidad de Málaga

Título	*Dynamic load balancing on heterogeneous clusters for parallel ant colony optimization*
Revista	Cluster Computing-The Journal of Networks Software Tools and Applications
Páginas	1–11
Año	2016
Estado	Publicado

Detalles de la revista *Cluster Computing*
Redactor jefe: Salim Hariri
ISSN: 1386-7857 (versión impresa)
ISSN: 1573-7543 (versión electrónica)
Editorial: Springer
Factor de impacto (2015): 1.514
Categoría: *Computer Science, Theory and Methods*
Página Web: `http://www.springer.com/computer/communication+networks/journal/10586`

Autores – Afiliación	
Nombre	**Antonio Llanes Castro**
Universidad	Universidad Católica de Murcia
Nombre	**Dr. José M. Cecilia**
Universidad	Universidad Católica de Murcia
Nombre	**Antonia Sánchez**
Universidad	Universidad Católica de Murcia
Nombre	**Dr. José Manuel García**
Universidad	Universidad de Murcia
Nombre	**Dr. Martyn Amos**
Universidad	Manchester Metropolitan University
Nombre	**Dr. Manuel Ujaldón**
Universidad	Universidad de Málaga

Título	*Soft Computing Techniques for the Protein Folding Problem on High Performance Computing Architectures*
Revista	Current Drug Targets
Año	2016
Estado	*Aceptado*

Detalles de la revista *Current Drug Targets*
Redactor jefe: Francis J. Castellino
ISSN: 1389-4501 (versión impresa)
ISSN: 1873-5592(versión electrónica)
Editorial: BenthamScience
Factor de impacto (2015): 3.029
Categoría: *Pharmacology and Pharmacy*
Página Web: `http://benthamscience.com/journals/current-drug-targets/`

Autores – Afiliación	
Nombre	**Antonio Llanes Castro**
Universidad	Universidad Católica de Murcia
Nombre	**Dr. Andrés Muñoz**
Universidad	Universidad Católica de Murcia
Nombre	**Dr. Andrés Bueno-Crespo**
Universidad	Universidad Católica de Murcia
Nombre	**Dra. Teresa García Valverde**
Universidad	Universidad Católica de Murcia
Nombre	**Dra. Antonia Sánchez**
Universidad	Universidad Católica de Murcia
Nombre	**Dr. Francisco Arcas Túnez**
Universidad	Universidad Católica de Murcia
Nombre	**Dr. Horacio Pérez-Sánchez**
Universidad	Universidad Católica de Murcia
Nombre	**Dr. José M. Cecilia**
Universidad	Universidad Católica de Murcia

El último de los artículos se encuentra en estado de *Aceptado*, a la espera de su publicación, se adjunta la carta de aceptación por parte de la revista.

From: Current Drug Targets <cdt@benthamscience.org>
To: "Dr. José María Cecilia Canales" <jmcecilia@ucam.edu>
Cc: "Horacio Pérez-Sánchez" <hperez@ucam.edu>, Francis Castellino <seemee.shabana@gmail.com>, Current Drug Targets <cdt@benthamscience.org>, Ambreen <ambreenirshad@benthamscience.org>

Date: Wed, 11 May 2016 06:45:36 +0500
Subject: [CDT] Manuscript Acceptance letter | BSP-CDT-2015-HT54-7
Reference#: BSP-CDT-2015-HT54-7

Submission Title: Metaheuristics for the Protein Folding Problem on High Performance Computing Architectures

Dear Dr. José María Cecilia Canales,

I am pleased to inform you that your article entitled Metaheuristics for the Protein Folding Problem on High Performance Computing Architectures has been accepted for publication in "Current Drug Targets" after independent peer review.

You may wish to request your Librarian to subscribe to the journal so that your work gets maximum exposure among your colleagues, researchers and readers in the field. I am pleased to bring to your attention a special limited time offer in this connection: If your Librarian decides to subscribe to the Bentham journal, you will become eligible to an optional offer which will allow free Open Access to this article as well as any other articles that you submit and which are accepted after peer review during the next 2 years.

We are also pleased to offer your institutions/library a FREE three months on-line trial of all Bentham Science Journals at no obligation to subscribe thereafter. The journals trial would allow free access to all the members of your institution/Library. For an on-line trial request, please click here (http://benthamscience.com/free-online-trials-request-main.php). If you are interested in this special limited time offer then you or your Librarian may contact either the Subscription Department directly at <subscriptions@benthamscience.org> Bentham Science or alternatively orders may be placed via your librarian's journal acquisition agency.

You will be pleased to know that Bentham Science Publishers has collaborated with Kudos to increase the portfolio of its services for Bentham authors. Kudos is among the preferred media for the researchers. It is a web-based service that helps researchers to maximize the visibility, usage of and citations to your published articles (www.growkudos.com.) Kudos will be contacting you to register to use this new service, that they are offering to a selected group of authors to help increase the readership and citations of their articles.

We wish to thank you for submission of the manuscript to Current Drug Targets and look forward to a continued collaboration in the future.

With warm regards,

Editorial Office
Bentham Science Publishers
Current Drug Targets
http://bsp-cms.eurekaselect.com/index.php/CDT

Capítulo 1

Introducción

1.1 Definición

Los retos científicos del siglo XXI precisan del tratamiento y análisis de una ingente cantidad de información en la conocida como la era del *Big Data* [31]. Los futuros avances en distintos sectores de la sociedad como la medicina, la ingeniería o la producción eficiente de energía, por mencionar sólo unos ejemplos, están supeditados al crecimiento continuo en la potencia computacional de los computadores modernos. Sin embargo, la estela de este crecimiento computacional, guiado tradicionalmente por la conocida *"Ley de Moore"* [32], se ha visto comprometido en las últimas décadas debido, principalmente, a las limitaciones físicas del silicio [30]. Los arquitectos de computadores han desarrollado numerosas contribuciones (*multicore*, *manycore*, heterogeneidad, *dark silicon*, etc) para tratar de paliar esta ralentización computacional, dejando en segundo plano otros factores fundamentales en la resolución de problemas como la programabilidad, la fiabilidad, la precisión, etc.

El desarrollo de *software*, sin embargo, ha seguido un camino totalmente opuesto, donde la facilidad de programación a través de modelos de abstracción, la depuración automática de código para evitar efectos no deseados y la puesta en producción son claves para una viabilidad económica y eficiencia del sector empresarial digital. Esta vía compromete, en muchas ocasiones, el rendimiento de las propias aplicaciones; consecuencia totalmente inadmisible en el contexto científico.

En esta tesis doctoral tiene como hipótesis de partida reducir las distancias entre los campos *hardware* y *software* para contribuir a solucionar los retos científicos del siglo XXI. El desarrollo de *hardware* está marcado por la consolidación de los procesadores orientados al paralelismo masivo de datos, principalmente GPUs (*Graphic Processing Unit*) [23] y procesadores vectoriales [12], que se combinan entre sí para construir procesadores o computadores heterogéneos [2].

En concreto, nos centramos en la utilización de GPUs para acelerar aplicaciones científicas [25]. Las GPUs se han situado como una de las plataformas con mayor proyección para la implementación de algoritmos que simulan problemas científicos complejos. Desde su nacimiento, la trayectoria y la historia de las tarjetas gráficas ha estado marcada por el mundo de los videojuegos, alcanzando altísimas cotas de popularidad según se conseguía más realismo en este área. Un hito importante ocurrió en 2006, cuando NVIDIA (empresa líder en la fabricación de tarjetas gráficas) lograba hacerse con un hueco en el mundo de la computación de altas prestaciones y en el mundo de la investigación con el desarrollo de CUDA [33] (*Compute Unified Device Arquitecture*). Esta arquitectura posibilita el uso de la GPU para el desarrollo de aplicaciones científicas de manera versatil. Desde entonces, la producción científica que se apoya en las tarjetas gráficas como plataformas para el cómputo no ha cesado de crecer; ya son más de cincuenta mil artículos científicos que se apoyan en CUDA, tal y como NVIDIA publicita, así como catálogos completos de aplicaciones basadas en CUDA [34]. A pesar de la importancia de la GPU, es interesante la mejora que se puede producir mediante su utilización conjunta con la CPU, lo que nos lleva a introducir los *sistemas heterogéneos* tal y como detalla el título de este trabajo. Es en entornos heterogéneos CPU-GPU donde estos rendimientos alcanzan sus cotas máximas, ya que no sólo las GPUs soportan el cómputo científico de los investigadores, sino que es en un sistema heterogéneo combinando diferentes tipos de procesadores donde podemos alcanzar mayor rendimiento. En este entorno no se pretende competir entre procesadores, sino al contrario, cada arquitectura se especializa en aquella parte donde puede explotar mejor sus capacidades.

Esta combinación de CPU-GPU está muy presente en los ordenadores más potentes del mundo [1]. En la última versión (Junio de 2016), las tarjetas gráficas de NVIDIA están presentes en el 58.1 % de los supercomputadores de dicha lista. Donde mayor rendimiento se alcanza es en estos clústeres heterogéneos, donde múltiples nodos son interconectados entre sí, pudiendo dichos nodos diferenciarse no sólo entre arquitecturas CPU-GPU, sino también en las capacidades computacionales dentro de estas arquitecturas. Con este tipo de escenarios en mente, se presentan nuevos retos en los que lograr que el *software* que hemos elegido como candidato se ejecuten de la manera más eficiente y obteniendo los mejores resultados posibles.

Estas nuevas plataformas hacen necesario un rediseño del *software* para aprovechar al máximo los recursos computacionales disponibles. Se debe por tanto rediseñar y optimizar los algoritmos existentes para conseguir que las aportaciones en este campo sean relevantes, y encontrar algoritmos que, por su propia naturaleza sean candidatos para que su ejecución en dichas plataformas de alto rendimiento sea óptima. Encontramos en este punto una familia de algotirmos denominados bioinspirados, que utilizan la inteligencia colectiva como núcleo para la resolución de problemas. Precisamente esta inteligencia colectiva es la que les hace candidatos perfectos para su implementación en estas plataformas bajo el nuevo paradigma de computación paralela, puesto que las soluciones pueden ser construidas en base a individuos que mediante alguna forma de comunicación son capaces de construir conjuntamente una solución común.

Esta tesis se centrará especialmente en uno de estos algoritmos bioinspirados que se engloba dentro del término metaheurísticas bajo el paradigma del *Soft Computing* [9], [26], [42], [44], el Ant Colony Optimization *ACO*, [14]. Se realizará una contextualización, estudio y análisis del algoritmo. Se detectarán las partes más críticas y serán rediseñadas buscando su optimización y paralelización, manteniendo o mejorando la calidad de sus soluciones. Posteriormente se pasará a implementar y testear las posibles alternativas sobre diversas plataformas de alto rendimiento. Se utilizará el conocimiento adquirido en el estudio teórico-práctico anterior para su aplicación a casos reales, más en concreto se mostrará su aplicación sobre el plegado de proteínas.

Desde un punto de vista algorítmico, las técnicas de computación tradicionalmente se han basado en tres objetivos principalmente: La precisión, la certidumbre, y el rigor. Estos principios hacen que el coste computacional de dichos algoritmos sea habitualmente muy alto, particularmente cuando tratamos con problemas reales en los que los tamaños de los datos crecen exponencialmente. Este es el punto de partida del *Soft Computing* que intenta superar las mencionadas dificultades, bajo la hipótesis de que a veces la precisión y la exactitud son inalcanzables, por lo que puede incluir la tolerancia a la imprecisión y la incertidumbre. En otras palabras, el término *Soft Computing* hace referencia a una colección de metodologías que tienen por objetivo explorar la tolerancia a la imprecisión e incertidumbre para lograr la manejabilidad, la robustez y las soluciones a bajo costo. Muchas veces el modelo a seguir por el *Soft Computing* ha sido considerado como la mente humana [42].

Dentro del *Soft Computing* encontramos las técnicas de optimización, y dentro de éstas, se localizan las metaheurísticas, las cuales siguiendo los principios del *Soft Computing*, permiten abordar tamaños de problemas inusualmente grandes ofreciendo soluciones satisfactorias en tiempos razonables. Sin embargo, las metaheurísticas no aseguran la obtención de las soluciones óptimas. Los problemas de optimización se encuentran en un amplio número de áreas de conocimiento [41], y éstos suelen ser muy complejos, por lo que las metaheurísticas se han incorporado a las resoluciones de todo tipo de problemas. Algunos ejemplos son el diseño de ingeniería, optimización topológica [43], aerodinámica [21], dinámica de fluidos [20], telecomunicaciones [35], aprendizaje máquina [6], minería de datos [36], modelado de sistemas [4], simulaciones químicas [24], físicas y biológicas [37] [45], problemas de planificación de rutas [22], problemas de planificación logística [18] y transporte [3], etc.

Bajo este término de metaheurísticas es donde se ubica al algoritmo Ant Colony Optimization, que atendiendo a la clasificación de las metaheurísticas según Blum [8], se englobaría dentro de los algoritmos *nature-inspired.* Dicha clasificación tiene en cuenta cinco diferentes características de los algoritmos; su origen (*nature-inspired o non nature-inspired*), el número de agentes en la búsqueda de soluciones al mismo tiempo, el tipo de funciones objetivo que usa, la estructura del vecindario y el uso que le dan al histórico de búsqueda. Como se ha detallado, el algoritmo Ant Colony Optimization se enmarca dentro de los algoritmos bioinspirados que intentan imitar algún proceso biológico, tal como el comportamiento de ciertas especies. Estos algoritmos utilizan la inteligencia de enjambre (*Swarm Intelligence*), la cual consiste en el principio de que un único individuo de un enjambre no posee la inteligencia suficiente como para resolver un problema, pero cooperando entre ellos son capaces de resolver eficientemente dichos problemas. Estos comportamientos pueden localizarse en la naturaleza en las colonias de hormigas, enjambres de abejas, bancos de peces, etc. En el caso concreto de las hormigas, que es el caso de interés en este trabajo, el Ant Colony Optimization fue introducido originalmente por Marco Dorigo en su tesis doctoral [14], y posteriormente analizado y estudiado por varios grupos [7], [40], [15], [39], por citar sólo algunos de los trabajos más interesantes al respecto. Las hormigas en la naturaleza viven en colonias y, aunque no existe un comportamiento que se defina como inteligente en un único individuo de la colonia, sí pueden mostrar comportamientos complejos cuando colaboran entre muchos individuos en realizar tareas difíciles. Precisamente este comportamiento en el que la cooperación entre multitud de agentes individuales conforman la solución final, es el principio que le hace candidato perfecto para ser rediseñado para su cómputo en las nuevas plataformas masivamente paralelas.

Todo este análisis es trasladado a su aplicación a un caso concreto. En este trabajo, aunamos las nuevas plataformas *hardware* de alto rendimiento junto al rediseño e implementación *software* de un algoritmo bioinspirado aplicado a un problema científico de gran complejidad como es el caso del plegado de proteínas. Es necesario cuando se implementa una solución a un problema real, realizar un estudio previo que permita la comprensión del problema en profundidad, ya que se encontrará nueva terminología y problemática para cualquier neófito en la materia, en este caso, se hablará de aminoácidos, moléculas o modelos de simulación que son desconocidos para los individuos que no sean de un perfil biomédico.

Una de las características que definen a un ser vivo es la capacidad de poder ensamblar todos y cada uno de sus componentes moleculares. Averiguar el mecanismo detrás de este ensamblaje es todavía uno de los mayores misterios que persisten en la biología molecular. Este problema cuando nos centramos en el caso de biomoléculas tales como las proteínas, se denomina plegado de proteínas o *Protein Folding* [11]. Debido a la ingente cantidad de posibilidades en la que pueda plegarse una secuencia de aminoácidos, encontrar el estado nativo de una proteína mediante algoritmo tradicionales de búsqueda que encuentren la solución óptima entre todas las configuraciones posibles se convierte en un problema NP-completo, irresoluble incluso por los ordenadores más potentes del mundo [38]. Para minimizar la complejidad del problema, el grupo de investigación de Ken A. Dill desarrollo una técnica denominada modelo HP basada en un alfabeto reducido donde los aminoácidos se dividen en dos grupos; polares (P) o hidrofóbicos (H) según su solubilidad en el agua. Las secuencias se pliegan en una malla bidimensional o tridimensional y mediante una optimización de una función matemática relacionada con las características HP de los diferentes aminoácidos y como se ensamblan estos entre sí, se busca la menor energía sirviendo como una aproximación al plegado de una forma simplificada [5]. Tomando como base esta técnica simplificada, se han usado metaheurísticas [17] tales como búsqueda tabú, Monte Carlo, algoritmos evolutivos y algoritmos genéticos.

Con el ánimo de estructurar y agilizar la lectura, se detalla en este punto la estructura del documento:

Capítulo 1: Introducción

En este capítulo se introduce las principales temáticas de investigación de esta tesis doctoral. Además se contextualiza el problema mostrando los principales objetivos a conseguir y a fin de servir de línea conductora que muestre como los artículos han servido en la consecución de dichos objetivos.

Dentro de esta sección se trata los tres apartados siguientes:

- ✓ *Definición.* Es el apartado actual donde se contextualiza la tesis y se presentan las partes más relevantes del texto.
- ✓ *Objetivos.* Muestra los objetivos a conseguir en la realización del trabajo.
- ✓ *Fundamentación del compendio de trabajo.* Fundamenta los artículos de los que consta este compendio mostrando un hilo conductor entre los mismos, y demostrando que los objetivos han sido conseguidos mediante la implementación de dichos artículos.

Capítulo 2: Artículos que componen la tesis doctoral.

Se incluirán los artículos de los que consta este trabajo, para que el lector pueda constatar en ellos, tanto los objetivos perseguidos en este trabajo, como la calidad de los mismos.

Capítulo 3: Resultados

Constará de un resumen de los resultados que han sido obtenidos en los artículos, de las conclusiones de los mismos y posibles vías futuras de ampliación.

En él se mostrarán los datos relativos a la calidad de las revistas en los que los trabajos de esta tesis han sido publicados, para dar indicios sobre la calidad y relevancia del presente trabajo.

Capítulo 4: Bibliografía Por último se realizará una relación de trabajos citados que completan la información necesaria para el estudio más profundo de los temas tratados.

1.2 Objetivos

En esta sección se detallan los objetivos establecidos que sientan las bases del presente trabajo. Se ha realizado una descripción de cada objetivo y se detallan las tareas que ayudan en la consecución del mismo.

Objetivo 1: Análisis, diseño y optimización del ACO en GPUs.

- Implementación del ACO en OpenCL para portarlo a distintas plataformas.

- Se analiza el comportamiento del ACO en base a *micro-benchmarks* específicamente diseñados, aislándolos del resto del algoritmo.

Objetivo 2: Evaluación de diferentes arquitecturas para la ejecución del ACO.

- Se analiza de forma exhaustiva el algoritmo con el fin de detectar cuellos de botella en el algoritmo candidatos a ser rediseñados.
- Analizar el rendimiento del ACO en distintas plataformas.

Objetivo 3: Estrategias de computación para la optimización del ACO en clústeres heterogéneos CPU-GPU para abordar problemas científicos de gran calado.

- Detectar posibles optimizaciones para la ejecución en clústeres heterogéneos en la ejecución del ACO.
- Implementar posibles alternativas de ejecución para el ACO en clústeres heterogéneos, haciendo especial hincapié en el consumo energético.

Objetivo 4: Estado del arte de técnicas de *Soft Computing* en plataformas de alto rendimiento para un problema científico concreto, *el plegado de proteínas.*

- Realizar un estudio sobre técnicas de *Soft Computing* que se aplican a problemas científicos en las publicaciones de los últimos años.
- Analizar cómo el uso de las arquitecturas de alto rendimiento sirven de apoyo a los científicos en problemas científicos concretos tales como el plegado de proteínas.

Objetivo 5: Implementación del algoritmo ACO aplicado a un problema científico concreto.

- Realizar un estudio sobre técnicas de *Soft Computing* que se aplican a problemas científicos en las publicaciones de los últimos años, aplicadas al plegado de proteínas.
- Implementar el algoritmo ACO aplicado a un problema científico concreto, en este caso al plegado de proteínas, mediante CUDA.

1.3 Fundamentación del compendio de trabajos

Este último punto de la introducción justifica cómo los diferentes artículos de los que consta este trabajo, cubren por completo los objetivos planteados, llevando una unidad temática que precisa cualquier trabajo científico y explicando los orígenes que motivaron la realización de la tesis.

La inquietud por la optimización de algoritmos bioinspirados parte del área de investigación de uno de los directores del trabajo, más concretamente del trabajo realizado por Cecilia y otros en [10], que establece la línea de inicio que marca el trabajo de esta tesis. Partiendo de las mejoras introducidas en [10], se plantea la migración del algoritmo a OpenCL para que pueda ser ejecutado en diversas plataformas, analizar las diferenecias, observando que los mejores resultados se alcanzan en las plataformas de NVIDIA. El trabajo que presenta esta migración y comparativa de plataformas es el titulado *Comparative evaluation of platforms for parallel Ant Colony Optimization* [19], que corresponde con el primero cronológicamente de los que consta esta tesis. Con el trabajo planteado en [19] se cubren dos de los objetivos planteados, concretamente los objetivos 1 y 2. Para la realización de la nueva implementación de ACO basada en OpenCL es necesario el análisis, diseño y optimización del algoritmo, tal y como el objetivo 1 plantea. La consecución apropiada del objetivo 2 queda demostrada en la comparación de los resultados obtenidos frente a otras implementaciones bajo diferentes plataformas *hardware*.

El siguiente trabajo que complementa los objetivos de la tesis es el titulado *Dynamic load balancing on heterogeneous clusters for parallel ant colony optimization* [27]. Este artículo aborda en su totalidad el objetivo 3. En [27], una vez que se tiene un algoritmo, en este caso en su versión CUDA, logramos escalar la ejecución de dicho algoritmo a un clúster heterogéneo mediante OpemMP y MPI, ofreciendo diversas estrategias para su cómputo en los casos en los que tarjetas con diferentes capacidades computacionales son las que están presentes en los nodos del clúster. Las diferentes estrategias, evidentemente, son detalladas en el artículo así como en el apartado de resultados y son ideas que se pueden portar a otros trabajos.

El tercer artículo que completa el listado de artículos en cuanto al compendio se refiere, es el titulado *Soft Computing Techniques for the Protein Folding Problem on High Performance Computing Architectures*, y se trata de una revisión de los artículos publicados durante los últimos años sobre técnicas de *Soft Computing* señalando cuáles de ellas utilizan plataformas de alto rendimiento como soporte a sus investigaciones. El Objetivo 4 se cubre en su totalidad por este trabajo, tanto realizando un estudio sobre el paradigma bajo el que se encuentra ACO, el *Soft Computing*, como analizando las plataformas y arquitecturas de alto rendimiento que sirven de apoyo en problemas de alta complejidad y gran calado científico, como por ejemplo el plegado de proteínas.

Como último punto, y fuera de los que son los artículos que sostienen la viabilidad de esta tesis, se ofrece también un artículo de un congreso internacional en el que se contribuyó con una comunicación oral en el mismo, dando lugar a su publicación como *lecture notes* [29]. Dicho artículo, es titulado *Parallel Ant Colony Optimization for the HP Protein Folding Problem*, y es precisamente el que desarrolla todo el trabajo que sustenta el objetivo 5, implementando el ACO ya no sólo para un *benchmark* clásico, sino aplicando esa implementación a un problema real como es el plegado de proteínas. Se consigue una implementación del ACO para el plegado de proteínas basada en CUDA que es susceptible a su vez de escalar a clústeres heterogéneos con las ideas propuestas en [27].

Capítulo 2

Artículos que componen la tesis doctoral

2.1 Comparative evaluation of platforms for parallel Ant Colony Optimization

Título	*Comparative evaluation of platforms for parallel Ant Colony Optimization*
Autores	Ginés.D. Guerrero, José M. Cecilia, Antonio Llanes, José M. García, Martyn Amos y Manuel Ujaldón
Revista	The Journal of Supercomputing
Páginas	318–329
Año	2014
Estado	Publicado

Contribución del Doctorando
Antonio Llanes Castro, declara ser el principal autor y el principal contribuidor del artículo *Comparative evaluation of platforms for parallel Ant Colony Optimization.*

J Supercomput (2014) 69:318–329
DOI 10.1007/s11227-014-1154-5

Comparative evaluation of platforms for parallel Ant Colony Optimization

Ginés D. Guerrero · José M. Cecilia · Antonio Llanes · José M. García · Martyn Amos · Manuel Ujaldón

Published online: 18 March 2014

Abstract The rapidly growing field of *nature-inspired computing* concerns the development and application of algorithms and methods based on biological or physical principles. This approach is particularly compelling for practitioners in high-performance computing, as natural algorithms are often *inherently parallel* in nature (for example, they may be based on a "swarm"-like model that uses a population of agents to optimize a function). Coupled with rising interest in nature-based algorithms is the growth in *heterogenous computing*; systems that use more than one kind of processor. We are therefore interested in the performance characteristics of nature-

G. D. Guerrero
National Laboratory for High Performance Computing, University of Chile, Santiago, Chile
e-mail: gguerrero@nlhpc.cl

J. M. Cecilia ( ) · A. Llanes
Computer Science Department, Universidad Católica San Antonio de Murcia, Murcia, Spain
e-mail: jmcecilia@ucam.edu

A. Llanes
e-mail: allanes@ucam.edu

J. M. García
Computer Engineering Department, University of Murcia, Murcia, Spain
e-mail: jmgarcia@ditec.um.es

M. Amos
School of Computing, Mathematics and Digital Technology, Manchester Metropolitan University, Manchester, UK
e-mail: m.amos@mmu.ac.uk

M. Ujaldón
Computer Architecture Department, University of Malaga, Malaga, Spain
e-mail: ujaldon@uma.es

inspired algorithms on a *number* of different platforms. To this end, we present a new OpenCL-based implementation of the Ant Colony Optimization algorithm, and use it as the basis of extensive experimental tests. We benchmark the algorithm against existing implementations, on a wide variety of hardware platforms, and offer extensive analysis. This work provides rigorous foundations for future investigations of Ant Colony Optimization on high-performance platforms.

Keywords Heterogeneous computing · Ant Colony Optimization · CUDA · OpenCL · APU · GPU

1 Introduction

Algorithms inspired by *natural* processes are gaining increasing acceptance, and are now used in a wide variety of application domains [28]. Many nature-inspired methods (such as the genetic algorithm [16], or particle swarm optimization [20]) are *population-based*, meaning that they maintain a *collection* of individual solutions which evolves or is modified as the computation proceeds. This structure naturally lends itself to *parallelisation*, and many parallel versions of such algorithms now exist [1].

One nature-based method that is proving to be increasingly popular is *Ant Colony Optimization* (ACO) [8,10,13]. This algorithm is based on foraging behaviour observed in colonies of real ants and has been applied to a wide variety of problems, including vehicle routing [32], feature selection [6] and autonomous robot navigation [15]. The method generally uses simulated "ants" (i.e., mobile agents), which first construct tours or paths on a network structure (corresponding to solutions to a problem), and then deposit "pheromone" (i.e., signalling chemicals) according to the quality of the solution generated. The algorithm takes advantage of emergent properties of the multi-agent system, in that positive feedback (facilitated by pheromone deposition) quickly drives the population to high-quality solutions.

The original ACO method (called the *Ant System* [11]) was developed by Dorigo in the 1990s, and this version (or slight variants thereof, such as the MAX-MIN Ant System (MMAS) [31]) is still in regular use [5,19,21]. Parallel versions of the Ant System have been developed [7,23,30,33] (see also [26] for a survey), and, in recent work, we present a graphics processor unit (GPU)-based version of ACO that, for the first time, parallelizes *both* main phases of the algorithm (that is, tour construction *and* pheromone deposition) [3,4].

The original version of our algorithm was developed for the CUDA (Compute Unified Device Architecture) platform,[1] which offers easy access to the parallel processing capabilities of GPUs (thus facilitating so-called "GPGPU" or "general purpose GPU" computation). Although it laid the foundations for general GPU-based computing, CUDA is proprietary to Nvidia, one of the dominant manufacturers in the GPU market. With that in mind, an alternative open-standard was developed, which became known as *OpenCL* (Open Computing Language) [29]. This standard provides a com-

[1] Full technical details at http://docs.nvidia.com/cuda/index.html.

mon language, programming interfaces and hardware abstractions over a *wide range* of devices (CPUs, GPUs and other accelerators), and has contributed significantly to the growth of *heterogeneous computing* [2]. Importantly, OpenCL offers *portability* across combinations of operating system, GPU and other processors, which, in turn, have their own hardware costs and performance characteristics. It is therefore possible to write a portable, parallel algorithm for a specific problem, which may run on a hardware/software combination that meets multiple constraints (cost, performance, and so on).

With that in mind, we present a new OpenCL-based version of our ACO algorithm, which may run on a variety of platforms (from laptops to high-end servers). Our aim is to demonstrate how such an implementation may be used as the foundation for *high-performance, portable* ACO-based solutions. We benchmark our algorithm on a range of platforms and give an analysis about its scalability on high-end platforms.

The paper is organized as follows: in Sect. 2 we briefly describe our ACO-based algorithm and the process of migrating it to OpenCL. We then present the results of experimental investigations in Sect. 3, offer some analysis in Sect. 4, and then conclude in Sect. 5 with a brief discussion of our findings.

2 ACO algorithm

Our ACO-based solution to the Travelling Salesman Problem (TSP) is described in detail in [3,4], so here we simply give a brief overview to highlight specific issues arising from the migration to OpenCL.

The TSP is a well-known NP-hard optimization problem, and is often used as a standard benchmark for heuristic algorithms [18]. Indeed, it was the first problem to be solved using ACO [11], and our own work is a natural development of this. Briefly, the TSP involves finding the shortest ("cheapest") round-trip route that visits each of a number of "cities" exactly once. In what follows, we address the symmetric TSP on n cities, which may be represented as a complete weighted graph, G, of n nodes, with each weighted edge, $e_{i,j}$, representing the inter-city distance $d_{i,j} = d_{j,i}$ between cities i and j. The ACO algorithm for TSP uses a number of simulated "ants" (or *agents*), which perform distributed search on a graph. Each ant moves on the graph until it completes a tour, and then offers this tour as its suggested solution. To achieve this latter step, each ant deposits "pheromone" on the edges that it visits during its tour. The quantity of pheromone deposited, if any, is determined by the *quality* of the solution relative to those obtained by the other ants. Pheromone levels on each edge "evaporate" over time (i.e., they are gradually reduced), to prevent the algorithm from being locked into sub-optimal solutions.

While building a tour, each ant probabilistically chooses the next city to visit based on two different sources of information: (1) heuristic information, obtained from inter-city distances, and (2) the pheromone trail, which facilitates indirect communication between ants via their *environment* (a process known as *stigmergy* [9]). The combination of local search and global signalling enables a process of directed positive feedback, by which the population quickly converges to a high-quality solution to the problem. The main body of the algorithm, therefore, has two main phases: (1) *tour construction*, and (2) *pheromone deposition*.

During tour construction, a number of ants build tours in parallel. Ants are initially placed at random, and they then repeatedly apply a probabilistic action choice rule to decide which city to visit next. Pheromone deposition occurs once all ants have constructed their tours; first, the pheromone levels on all edges are reduced by a constant factor (to simulate evaporation), and then pheromone is deposited on edges that ants have included in their tours (the precise amount for each edge in a particular tour being inversely proportional to the tour's length). In this way, edges that are used by many ants (and which are part of short tours) receive more pheromone, and are therefore more likely to be selected by ants in subsequent rounds (thus implementing the positive feedback process that we have already described).

2.1 Original CUDA implementation

We first briefly review the main characteristics of CUDA [24], for the benefit of readers who are unfamiliar with the programming model. CUDA is based on a hierarchy of abstraction layers; the *thread* is the basic execution unit; threads are grouped into *blocks*, each of which run on a single multiprocessor, where they can share data on a small but extremely fast memory. A *grid* is composed of blocks, which are equally distributed and scheduled among all multiprocessors. The parallel sections of an application are executed as *kernels* in a SIMD (Single Instruction Multiple Data) fashion, that is, with all threads running the same code. A kernel is therefore executed by a grid of thread blocks, where threads run simultaneously grouped in batches called *warps*, which are the scheduling units.

We now consider the implementation of each phase of the algorithm. The "traditional" task-based parallelism approach is based on the observation that ants run in parallel while searching for the best tour [4] (that is, parallelism is expressed at the level of individual ants). Within the basic model, each ant is associated with an individual thread, but this approach has three main drawbacks:

1. **Low degree of parallelism**. Because the number of ants used is generally a (linear) function of the problem size, the number of threads required is generally too low to fully exploit the resources of the GPU.
2. **Control dependencies**. *Warp divergences* (a situation where threads take different control-flow paths) can often arise when ants check the so-called *tabu list*—the record of cities already visited. Put simply, different threads in a warp may need to do different things, depending on which cities the different ants have visited, and this is expensive.
3. **Irregular memory access**. Because the ACO algorithm is inherently stochastic, this can produce an unpredictable *memory access pattern*. This prevents the GPU from taking advantage of *caching* schemes and other techniques for reducing memory access latency.

In previous work, we developed an alternative approach that places more emphasis on *data parallelism* [3]. We now briefly describe this algorithm, to establish the differences between the CUDA and OpenCL implementations.

When an ant makes a decision on which city to visit next, it must calculate heuristic information, as previously described. The heuristic information available to any one ant at a given time is *the same*, regardless of which ant is making the query, so it makes sense to separate out the computation of heuristic values into a separate *heuristic info kernel*, which is then executed prior to tour construction. Transition probabilities are stored in a two-dimensional *choice matrix*, which is used to inform "roulette wheel" (Monte Carlo) selection by each ant.

In the *tour construction* kernel, each ant is associated with a *thread block*, such that each thread represents a city (or cities) that the ant may visit. This avoids the problem of warp divergences, and enhances data parallelism, as all threads within a block may *co-operate*. The degree of parallelism improves by a factor of $1 : w$, where w is the number of CUDA threads per block.

Finally, the *pheromone kernel* performs evaporation and deposition, as described earlier. Evaporation is straightforward, as a single thread can independently lower each entry in the pheromone matrix by a constant factor. Deposition is more problematic, as each ant generates its own private tour in parallel, and will eventually visit the same edge as another ant. Therefore, to prevent race conditions, we require the use of CUDA atomic operations when accessing the pheromone matrix.

The efficiency of a parallel implementation is also affected by the *types* of operation on which it relies; in our code, scatter/gather operations [17] predominate (i.e., those which either write or read a large number of data items). As Table 1 reflects, the vast majority of operations are of the "gather" type; algorithms of this type are memory-bounded and amenable to optimization via methods such as *coalescing* (Nvidia GPUs) and the use of SSE *vector instructions* (Intel CPUs). A comparative study [22] of these optimisations reveals similar impact on performance across platforms, which suggests that the experimental sections of the current paper will not suffer too much from platform-specific biases.

2.2 OpenCL migration

In this section, we briefly describe various issues that arose during the migration from CUDA to OpenCL. The foundations of OpenCL are based on the CUDA threading model, but with differences in terms of naming schemes and identifiers. We therefore used source-to-source translation to migrate our CUDA-based kernels to OpenCL. This mapping requires in-depth knowledge of both application programming interface (API) models, as it is considerably more complex than simple instruction conversion. Also, OpenCL is still relatively young compared to CUDA, and does not provide the same functionality offered by its more mature partner.

The process of setting up a device for kernel execution differs *substantially* between CUDA and OpenCL. The APIs for context creation and data copying use different conventions for mapping the kernel onto the device processing elements, which may substantially affect the programming effort required to code and debug a parallel application. CUDA provides several libraries to enhance the functionality of its API. For example, our ACO algorithm uses the CURAND library [25] to generate pseudo-random numbers. This library is not directly implemented in OpenCL, where the main

Table 1 Characterization of the stages involved in our ACO implementation on GPUs

Algorithm stage	Operator	Key features	CUDA kernel
Generation of `choice_info` array	Gather	Data parallelism fully exploited	`choice_info`
Tour construction	Gather	Optimized via `choice_info` array	`Next_tour`
Tabu list update	Scatter	Optimized via an array in register file	`Next_tour`
Pheromone evaporation	Scatter	Concurrent updates, no queries	`Pheromone`
Pheromone deposit	Gather	Single update using atomic operations	`Pheromone`

alternative is an implementation of the RANLUX pseudo-random number generator, called RANLUXCL.[2] Unfortunately, we found this library to be fairly wasteful in terms of memory, so we decided to implement our own, taking a C counterpart as a departure point [14].

3 Experimental results

In this section we give the results of extensive comparative evaluations of ACO-based solutions to the TSP on different CPU, APU and GPU platforms. The underlying hardware platforms we tested are specified in Table 2.

For validation purposes, we use a baseline comparison with the sequential ANSI C code provided in [12]. The experimental setup (in terms of hardware/software) is listed in Table 3. We run our three ACO implementations (ANSI C, CUDA and OpenCL) on selected benchmark TSP instances from the well-known TSPLIB library [27]. All instances are defined on a complete graph, and distances are given as integers. Table 4 specifies the instances used; they were selected to ensure a representative sample, from "small" to "medium" and "large" (for reasons of practicality, we test only the high-end platforms on $pr2392$; these results are used for the later scalability analysis). Importantly, we note that our methods solve all instances *to optimality*; for the purposes of this paper, we are less interested in the *quality* of solutions produced, so to ensure a fair comparison we use instances that are solvable to optimality by our implementations as described in [3].

For all runs, we set the ACO parameters according to the values recommended in [12]; $\alpha = 1, \beta = 2, \rho = 0.5$, and $m = n$, meaning that the number of ants, m, is equal to the number of cities, n. We run each algorithm for 1,000 iterations, and average timings over 1,000 runs. CUDA times are obtained with a block size of 128 threads, and OpenCL local size is also set to 128.

Before discussing the results of our experiments, we consider several issues with respect to performance. First, APUs are much more limited in terms of thermal design power, as they must also include the CPU. This means that execution units will need to be removed to keep power consumption down. Second, because the APU is a cost-effective solution, it does not have its own dedicated global memory, but instead it

[2] See https://bitbucket.org/ivarun/ranluxcl/.

Table 2 Summary of hardware features for the CPUs, APUs and GPUs used during our experimental survey

	CPU	GPU	GPU
(a) Processors found in high-end servers			
Release date	Q4 2009	Q4 2009	Q1 2010
Codename	Intel Westmere	Nvidia Fermi	ATI Cypress
Commercial model	Xeon E5620	Tesla C2050	FirePro V8800
No. cores @ speed	4 @ 2.4 GHz	–	–
No. stream processors	–	448 @ 1.15 GHz	1,600 @ 925 MHz
L2 cache size	12 MB	768 KB	512 KB
DRAM memory size	16 GB	3 GB	2 GB
DRAM type	DDR3	GDDR5	GDDR5
Memory bus width	128 bits	384 bits	256 bits
Memory clock	1,066 MHz	2×1.5 GHz	4×1.15 GHz
Memory bandwidth	17 GB/s	144 GB/s	147.2 GB/s
	CPU on APU	GPU on APU	
(b) Processors found in desktop PCs			
Release date	Q1 2010	Q1 2010	
Codename	AMD Llano	ATI Redwood	
Commercial model	E-350	ATI HD 6310	
No. cores @ speed	2 @ 1.6 GHz	–	
No. stream processors	–	80 @ 492 MHz	
L2 cache size	2×512 KB	–	
DRAM memory size	4 GB (shared)	4 GB (shared)	
DRAM type	DDR3	DDR3	
Memory bus width	64 bits	64 bits	
Memory clock	1,066 MHz	1,066 MHz	
Memory bandwidth	8.5 GB/s	8.5 GB/s	
	CPU on APU	GPU on APU	GPU
(c) Processors found in laptops			
Release date	Q2 2011	Q2 2011	Q1 2011
Codename	AMD Llano	ATI Redwood	ATI Redwood
Commercial model	A6-3420	Radeon HD 6520	Radeon HD 6650M
No. cores @ speed	4 @ 1.4 GHz	–	–
No. stream processors	–	320 @ 400 MHz	480 @ 600 MHz
L2 cache size	4 MB	–	–
DRAM memory size	4 GB (shared)	4 GB (shared)	1 GB (exclusive)
DRAM type	DDR3	DDR3	DDR3
Memory bus width	64 bits	64 bits	128 bits
Memory clock	1,333 MHz	1,333 MHz	900 MHz
Memory bandwidth	10.6 GB/s	10.6 GB/s	14.4 GB/s

Table 3 Software resources used for each hardware platform in our experimental study

Target hardware	Software tools
Intel Xeon CPU	gcc compiler, 4.3.4 version with the -O3 flag set
Nvidia Tesla GPU	CUDA compilation tools, release 4.0
ATI FirePro GPU	Software Suite 8.85.7.2 and OpenCL runtime v831.4
AMD APUs and dedicated GPUs	AMD's APP SDK 2.6, Catalyst driver 11.12, OpenCL runtime version 793.1

Table 4 TSP instances used in our study

	Small dataset				Medium/large dataset				
Graph name	d198	a280	lin318	pcb442	rat783	pr1002	pcb1173	d1291	pr2392
Number of cities	198	280	318	442	783	1,002	1,173	1,291	2,392
Best tour length	15,780	2,579	42,029	50,778	8,806	259,045	56,892	50,801	378,032

relies on an emulated global memory located in system memory. While this is good for performance when transferring data directly between the CPU and GPU, it means that it will also suffer in terms of overall bandwidth, as even low-end GPUs have more memory bandwidth.

We present a summary of our results in Fig. 1. For each row (i.e., each platform, or hardware/software combination), we show execution times averaged over the small (top bar) and medium/large (bottom bar) instances. Note that times are measured in milliseconds (ms), and represent the elapsed time for a *single iteration* of the platform-specific algorithm, averaged over 100 runs of 1,000 iterations each (as opposed to the average run time for the whole algorithm). We focus on the average time for a single iteration precisely because we are interested in the *overall kernel performance* on each platform, so this fine-grained approach gives us the insights that we require.

4 Analysis

We now give an analysis of the performance of each category of hardware platform.

4.1 Desktop PCs

Beginning with the E-350 APU, we see that the CPU does not perform particularly well. This is expected, based on the architecture's emphasis on power consumption over performance for this consumer market. However, when moving to the GPU we see that, for small problem instances, it actually scales better in terms of overall computational time than the FirePro V8800 for the same base architecture.

Looking closely at the numbers, the E-350 APU, which is outclassed by factors of 37 and 17 for computational power (*execution resources* $\times$ *clock speed*) and memory

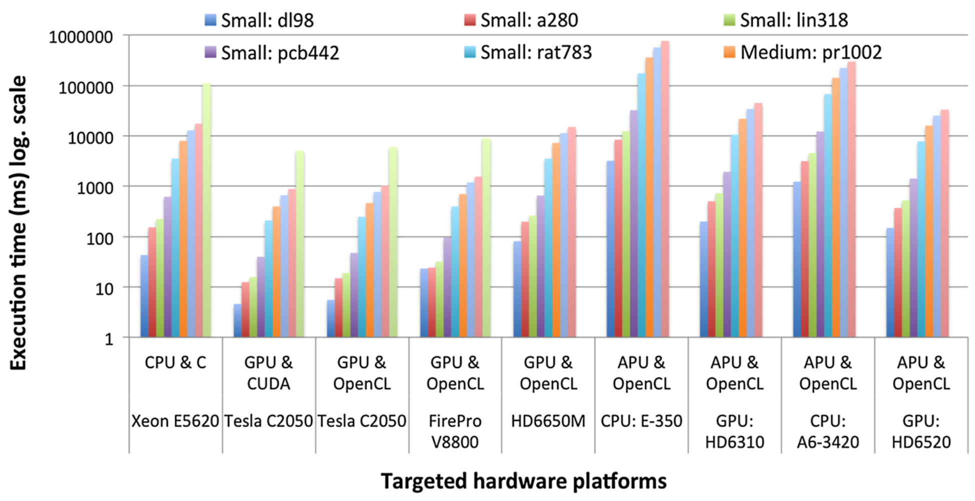


Fig. 1 Summary of experimental results. *X* axis shows each platform, *Y* axis (logarithmic plot) shows execution time (ms) for one iteration. *Bars* are ordered from the smallest (*left*) to the largest (*right*) instances

bandwidth, respectively, manages to only perform at roughly 1/10th the speed. We attribute this to the APU's ability to quickly transfer data to and from the CPU to the GPU. However, as the input size increases this advantage disappears, as raw computational throughput and bandwidth become more important than latency. Comparing these results to the Tesla C2050 GPU, the APU is at an even greater disadvantage, due to its VLIW architecture (compared to the scalar and compute-oriented architecture of the C2050). This should change, however, with AMD future generations of APUs, which consider a GPU based on their newly released Graphics Core Next (GCN) architecture. GCN greatly improves computational throughput, by moving scheduling from the compiler to the hardware.

4.2 Laptop computers

Moving to the A6-3420M APU, we see very similar results as with the E-350 APU. Here, our integrated GPU (iGPU) has roughly 3 times the amount of computational resources, but only 1/4 more bandwidth. This is evident in the scaling of the algorithm, as we go from 200 ms. with the E-350 APU to 148 ms. with the iGPU, a near exact scaling of the bandwidth advantage that the iGPU possesses.

As we increase the size of the problem instance, we then see that performance becomes constrained more by computational resources than by bandwidth. Our simplest comparison is to the A6-3420M's dedicated GPU (which has 2.25 times the computational power), where, as the input size increases, the difference between the two solutions approaches this limitation. This shows that, while the memory system influences ACO performance, computational resources become the dominating factor in overall performance. For PCI-express 2.0, the maximum bandwidth (unidirectional) is 8 GB/s, while using zero copy the APU is able to reach nearly 16 GB/s. If this had been taken into account, the results for the APU and dedicated GPUs would in fact be much closer, as this type of workload/data transfer is playing to the APU's strength.

Table 5 Scalability on high-end-platforms depending on hardware and programming methods

Scalability →		Short range	Mid range	Long range
Language/API	HW platform	Time(pr2392)/ Time(rat783)	Time(rat783)/ Time(d198)	Time(pr2392)/ Time(d198)
C	CPU Xeon	31.24*x*	82.30*x*	2,571.46*x*
CUDA	GPU Tesla	24.43*x*	45.74*x*	1,117.88*x*
OpenCL	GPU Tesla	24.16*x*	44.83*x*	1,083.52*x*
OpenCL	GPU FirePro	22.70*x*	17.09*x*	388.12*x*

FirePro behaves better on larger problem instances, followed by Tesla using OpenCL (with CUDA very close), and finally Xeon using C

Ending with the dedicated GPU (dGPU), we see a similar speedup increase, just as we did for the E-350 and A6-3420M APUs. Again, for small input sizes, latency and bandwidth are much more important than the computational abilities of the device, as there are fewer threads to interleave to hide memory accesses. This is visible in the dGPU, which has 7 and 2 times the amount of computational power and memory bandwidth as the E-350 APU, while performing just over twice as quickly for the d198 dataset.

As we increase the complexity of the workload, we again see that memory bandwidth becomes a less important issue, and computational power becomes the main contributing factor for overall performance. Comparing once again to the Tesla C2050, the APU solution does not perform as well as we had hoped.

4.3 High-end platforms

High-end processors usually cover large-scale applications, and our performance analysis emphasises *scalability*. Table 5 shows the behaviour of the execution time when the problem size increases. We compare execution times on small, medium and large instances, and obtain the coefficient or multiplier which separates them. The larger this coefficient is for a given processor, the poorer the degree of scalability.

Looking at those numbers, we see that when comparing Tesla versus FirePro (GPUs running the same OpenCL code), Tesla is 1.5*x*–2*x* *faster*, but FirePro *scales* better. Also, comparing languages on the same Tesla hardware, CUDA is 1.15*x*–1.20*x* faster, but OpenCL scales slightly better. Finally, comparing GPU results with numbers on the CPU, the GPU is faster and scales better: The speed-up factor ranges 9*x*–15*x* on four small data sets, 17*x*–20*x* on four medium data sets, and finally 21.5*x* on the large data set.

5 Conclusions

In this paper we presented a comprehensive performance review of different platforms for Ant Colony Optimization, an emerging and fast-growing nature-inspired algorithm.

We discussed the translation of our previous algorithm from CUDA to OpenCL, and highlighted certain issues that may be faced by other practitioners in future. We then performed a performance analysis of three variants of the ACO algorithm, using the Travelling Salesman Problem as a benchmark, and focussed on issues of scalability.

In general, GPUs are superior to CPUs on the high-end segment: they yield twenty times faster execution on large problem instances. The GPU–CPU difference is similar on desktops and laptops, 10–20x in favor of GPUs. At an early stage of its evolution, the APU offers a low-cost platform, without powerful computational units nor swift memory data paths. Our results demonstrate that these two issues have a severe impact on performance.

The growth of heterogeneous systems represents a solid trend in modern systems, and we believe that future work on Ant Colony Optimization in this domain can benefit from the promising insights into scalability demonstrated by our experimental study.

Acknowledgments This work is jointly supported by the Fundación Séneca (Agencia Regional de Ciencia y Tecnología, Región de Murcia) under grant 15290/PI/2010, by the Spanish MEC and European Commission FEDER under grant TIN2012-31345, by the UCAM under grant PMAFI/26/12, by the Junta de Andalucía under Project of Excellence P12-TIC-1741 and by the supercomputing infrastructure of the NLHPC (ECM-02). We also thank NVIDIA for hardware donation under CUDA Teaching Center 2011-14, CUDA Research Center 2012-14 and CUDA Fellow 2012-14 Awards.

References

1. Alba E, Luque G, Nesmachnow S (2013) Parallel metaheuristics: recent advances and new trends. Int Trans Oper Res 20(1):1–48. doi:10.1111/j.1475-3995.2012.00862.x
2. Brodtkorb AR, Dyken C, Hagen TR, Hjelmervik JM, Storaasli OO (2010) State-of-the-art in heterogeneous computing. Sci Progr 18(1):1–33
3. Cecilia JM, Garcia JM, Nisbet A, Amos M, Ujaldón M (2013) Enhancing data parallelism for ant colony optimization on GPUs. J Parallel Distrib Comput 73(1):42–51
4. Cecilia JM, Garcia JM, Ujaldon M, Nisbet A, Amos M (2011) Parallelization strategies for ant colony optimisation on GPUs. In: Proceedings of the 2011 IEEE international symposium on parallel and distributed processing. IEEE, pp 339–346
5. Chang RSS, Chang JSS, Lin PSS (2009) An ant algorithm for balanced job scheduling in grids. Future Gener Comput Syst 25(1):20–27. doi:10.1016/j.future.2008.06.004
6. Chen Y, Miao D, Wang R (2010) A rough set approach to feature selection based on ant colony optimization. Pattern Recognit Lett 31(3):226–233. doi:10.1016/j.patrec.2009.10.013
7. Delévacq A, Delisle P, Gravel M, Krajecki M (2013) Parallel ant colony optimization on graphics processing units. J Parallel Distrib Comput 73(1):52–61. doi:10.1016/j.jpdc.2012.01.003
8. Dorigo M, Birattari M, Stutzle T (2006) Ant colony optimization. Comput Intell Mag IEEE 1(4):28–39
9. Dorigo M, Bonabeau E, Theraulaz G (2000) Ant algorithms and stigmergy. Future Gener Comput Syst 16(8):851–871
10. Dorigo M, Di Caro G (1999) Ant colony optimization: a new meta-heuristic. In: Proceedings of the 1999 congress on evolutionary computation (CEC'99). IEEE Press, pp 1470–1477
11. Dorigo M, Maniezzo V, Colorni A (1996) Ant system: optimization by a colony of cooperating agents. IEEE Trans Syst Man Cybern B 26(1):29–41
12. Dorigo M, Stutzle T (2004) Ant Colony Optimization. Bradford Company
13. Dorigo M, Stützle T (2010) Ant colony optimization: overview and recent advances. In: Handbook of metaheuristics. Springer, Berlin, pp 227–263
14. Flannery BP, Press WH, Teukolsky SA, Vetterling W (1992) Numerical recipes in c. Press Syndicate of the University of Cambridge, New York

15. Garcia MP, Montiel O, Castillo O, Sepúlveda R, Melin P (2009) Path planning for autonomous mobile robot navigation with ant colony optimization and fuzzy cost function evaluation. Appl Soft Comput 9(3):1102–1110. doi:10.1016/j.asoc.2009.02.014
16. Goldberg DE (1989) Genetic algorithms in search, optimization, and machine learning. Addison-Wesley Professional, Reading
17. He B, Govindaraju NK, Luo Q, Smith B (2007) Efficient gather and scatter operations on graphics processors. In: Proceedings of the 2007 ACM/IEEE conference on supercomputing. ACM, New York, pp 46–57
18. Johnson DS, McGeoch LA (1997) The traveling salesman problem: a case study in local optimization. In: Lenstra J, Aarts E (eds) Local search in combinatorial optimization. Wiley, New York, pp 215–310
19. Ke BR, Chen MC, Lin CL (2009) Block-layout design using max-min ant system for saving energy on mass rapid transit systems. IEEE Trans Intell Transp Syst 10(2):226–235. doi:10.1109/TITS.2009.2018324
20. Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proceedings of IEEE international conference on neural networks, vol 4. IEEE, pp 1942–1948
21. Komarudin Wong KY (2010) Applying ant system for solving unequal area facility layout problems. Eur J Oper Res 202(3):730–746. doi:10.1016/j.ejor.2009.06.016
22. Lee VW, Kim C, Chhugani J, Deisher M, Kim D, Nguyen AD, Satish N, Smelyanskiy M, Chennupaty S, Hammarlund P (2010) Debunking the 100x gpu vs. cpu myth: an evaluation of throughput computing on cpu and gpu. In: ACM international symposium on computer architecture. ACM, pp 451–460
23. Manfrin M, Birattari M, Stützle T, Dorigo M (2006) Parallel ant colony optimization for the traveling salesman problem. In: Ant colony optimization and swarm intelligence. Springer, Berlin, pp 224–234
24. Nickolls J, Buck I, Garland M, Skadron K (2008) Scalable parallel programming with cuda. Queue 6(2):40–53
25. Nvidia (2011) CUDA Toolkit 4.0 CURAND Guide. http://developer.download.nvidia.com/compute/DevZone/docs/html/CUDALibraries/doc/CURAND_Library.pdf
26. Pedemonte M, Nesmachnow S, Cancela H (2011) A survey on parallel ant colony optimization. Appl Soft Comput 11(8):5181–5197. doi:10.1016/j.asoc.2011.05.042
27. Reinelt G (1991) TSPLIB—a traveling salesman problem library. ORSA J Comput 3(4):376–384. Library available at http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/
28. Rozenberg G, Bäck T, Kok JN (2011) Handbook of natural computing. Springer, Berlin
29. Stone JE, Gohara D, Shi G (2010) OpenCL: a parallel programming standard for heterogeneous computing systems. Comput Sci Eng 12(3):66–72. doi:10.1109/MCSE.2010.69
30. Stützle T (1998) Parallelization strategies for ant colony optimization. In: Parallel Problem Solving from Nature (PPSN V). Springer, Berlin, pp 722–731
31. Stutzle T, Hoos HH (2000) MAX-MIN ant system. Future Gener Comput Syst 16(8):889–914
32. Yu B, Yang ZZ, Yao B (2009) An improved ant colony optimization for vehicle routing problem. Eur J Oper Res 196(1):171–176. doi:10.1016/j.ejor.2008.02.028
33. Zhu W, Curry J (2009) Parallel ant colony for nonlinear function optimization with graphics hardware acceleration. In: IEEE international conference on systems, man and cybernetics, 2009, SMC 2009. IEEE, pp 1803–1808

2.2 Dynamic load balancing on heterogeneous clusters for parallel ant colony optimization

Título	*Dynamic load balancing on heterogeneous clusters for parallel ant colony optimization*
Autores	Antonio. Llanes, José M. Cecilia, Antonia Sánchez, Martyn Amos y Manuel Ujaldón
Revista	Cluster Computing
Páginas	1-11
Año	2016
Estado	Publicado

Contribución del Doctorando
Antonio Llanes Castro, declara ser el principal autor y el principal contribuidor del artículo *Comparative evaluation of platforms for parallel Ant Colony Optimization.*

Cluster Comput
DOI 10.1007/s10586-016-0534-4

Dynamic load balancing on heterogeneous clusters for parallel ant colony optimization

Antonio Llanes[1] · **José M. Cecilia**[1] · **Antonia Sánchez**[1] · **José M. García**[2] · **Martyn Amos**[3] · **Manuel Ujaldón**[4]

Received: 30 March 2015 / Revised: 4 January 2016 / Accepted: 5 January 2016

Abstract Ant colony optimisation (ACO) is a nature-inspired, population-based metaheuristic that has been used to solve a wide variety of computationally hard problems. In order to take full advantage of the inherently stochastic and distributed nature of the method, we describe a parallelization strategy that leverages these features on heterogeneous and large-scale, massively-parallel hardware systems. Our approach balances workload effectively, by dynamically assigning jobs to heterogeneous resources which then run ACO implementations using different search strategies. Our experimental results confirm that we can obtain significant improvements in terms of both solution quality and energy expenditure, thus opening up new possibilities for the development of metaheuristic-based solutions to "real world" problems on high-performance, energy-efficient contemporary heterogeneous computing platforms.

Keywords Heterogeneous computing · Ant colony optimization · CUDA · Power-aware systems

✉ Manuel Ujaldón
ujaldon@uma.es

1 Department of Computer Science, Universidad Católica San Antonio de Murcia (UCAM), 30107 Murcia, Spain

2 Department of Computer Engineering, University of Murcia, 30080 Murcia, Spain

3 School of Computing, Mathematics and Digital Technology, Manchester Metropolitan University, Manchester, UK

4 Department of Computer Architecture, University of Málaga, 29071 Málaga, Spain

1 Introduction

Heterogeneous systems combine different types of processor, and computing nodes may use a combination of traditional multicore architectures (CPUs) and accelerators (mostly Nvidia GPUs [31] or Intel Xeon Phi cards [35]). Although such systems are becoming more common [42], they present a new set of specific challenges, such as scalability, energy efficiency, data management, programmability and reliability [2].

The role of the software developer will be increasingly important as such systems grow in popularity. They will be expected to manage the inherent tension between performance and power consumption, exploit the most useful feature of each component type, and be able to handle the complexity implied by combinations of hardware, instruction sets and programming models. So far, the efficient mapping of system components to computations within heterogeneous systems is largely the responsibility of the programmer (that is, the ability of the run-time system to achieve this is relatively immature).

The *hardware/software co-design* methodology has emerged since the 1990s as an approach to providing both *analysis* methods (which allow developers to assess whether or not a system meets its goals in terms of performance, power usage, etc.), and *synthesis* methods (which allow developers and researchers to rapidly explore the space of design methodologies) [8,44].

This approach has facilitated significant advances in high-performance computing, which has, in turn, allowed for developments in computational modelling, image analysis, and many other areas [25,38].

A particular application domain of interest to us is *metaheuristics*; specifically, algorithms inspired by *natural* processes or phenomena [37]. Many of these methods (such as the genetic algorithm [18], or particle swarm optimization [23]) are *population-based*: they maintain a *collection* of individual solutions which "evolves" in some way as the computation proceeds. These algorithms are generally stochastic, as they tend to rely on randomized search techniques. Additionally, they are inherently parallel, and many such variants have been described [1].

One nature-based method of particular interest is *Ant Colony Optimization* (ACO) [10,14,16]. This algorithm is based on foraging behavior observed in colonies of ants, and has been applied to a wide variety of problems, including vehicle routing [45], feature selection [7] and autonomous robot navigation [17]. The method relies on "ants" (i.e., mobile agents) constructing paths on a graph representing a particular problem, where the paths represent a given solution. Paths are assessed according to the quality of the solution that they represent, and ants then deposit "pheromone" (i.e., signalling chemicals) accordingly (the better the solution, the higher the pheromone concentration). The algorithm takes advantage of positive feedback behaviour that emerges from the multi-agent system, where distributed selection quickly drives the population to high quality solutions.

The original ACO method (called the *Ant System* [12]) was developed by Dorigo in the 1990s, and this version (or slight variants thereof, such as the MAX-MIN Ant System (MMAS) [41]) is still in regular use [6,22,24]. Parallel versions of the Ant System have been developed [9,27,40,46] (see also [33] for a survey), and, in recent work, we have presented a GPU-based version of ACO that, for the first time, parallelizes *both* main phases of the algorithm (that is, tour construction *and* pheromone deposition) [3,4].

The initial version of our ACO algorithm [3,4] was implemented in CUDA (Compute Unified Device Architecture) and written in C, which gave access to the parallel processing capabilities of the GPU. This paper extends our framework to encompass large-scale supercomputers, thus enabling its implementation in MPI and OpenMP (in addition to CUDA), and also incorporating different generations of Nvidia GPUs.

Since the advent of CUDA in 2006, at least four different generations of GPUs have been released: Tesla, Fermi, Kepler and Maxwell. Our algorithmic design investigates the potential to deploy a load-balancing strategy across several generations of Nvidia GPUs, for maximum performance and minimum power consumption. In what follows, we use our well-established ACO based metaheuristic as a both a benchmarking application and an illustration of the long-term potential for this method. Our experimental study covers a wide range of computing systems, from consumer-market devices to high-end servers.

This paper is organized as follows. Section 2 reviews the ACO method, the CUDA programming model and our ACO-based algorithm. Section 3 describes our parallelization techniques to enhance ACO simulation on GPU-based heterogeneous clusters, which form the main contribution of this work. Section 4 focuses on the experimental results, Sect. 5 gives a performance analysis, and we conclude in Sect. 6 with an overall assessment and suggestions for future work.

2 Background

2.1 Ant colony optimisation for the traveling salesman problem

In what follows, we reprise our description of the algorithm, which was first given in [5]. The Traveling Salesman Problem (TSP) [26] involves finding the shortest (or "cheapest") round-trip route that visits each of a number of "cities" exactly once. The symmetric TSP on n cities may be represented as a complete weighted graph, G, with n nodes, with each weighted edge, $e_{i,j}$, representing the inter-city distance $d_{i,j} = d_{j,i}$ between cities i and j. The TSP is a well-known NP-hard optimisation problem, and is used as a standard benchmark for many heuristic algorithms [21].

The TSP was the first problem solved by Ant Colony Optimisation (ACO) [11,13]. This method uses a number of simulated "ants" (or *agents*), which perform distributed search on a graph. Each ant moves through on the graph until it completes a tour, and then offers this tour as its suggested solution. In order to do this, each ant may drop "pheromone" on the edges contained in its proposed solution. The amount of pheromone dropped, if any, is determined by the *quality* of the ant's solution relative to those obtained by the other ants. The ants probabilistically choose the next city to visit, based on *heuristic information* obtained from inter-city distances and the net pheromone trail. Although such heuristic information drives the ants towards an optimal solution, a process of "evaporation" is also applied in order to prevent the process stalling in a local minimum.

The Ant System (AS) is an early variant of ACO, first proposed by Dorigo [11]. The AS algorithm is divided into two main stages: *Tour construction* and *Pheromone update*. Tour construction is based on m ants building tours in parallel. Initially, ants are randomly placed. At each construction step, each ant applies a probabilistic action choice rule, called the *random proportional rule*, in order to decide which city to visit next. The probability for ant k, placed at city i, of visiting city j is given by the Eq. 1

$$p^k_{i,j} = \frac{[\tau_{i,j}]^\alpha [\eta_{i,j}]^\beta}{\sum_{l \in N^k_i} [\tau_{i,l}]^\alpha [\eta_{i,l}]^\beta}, \qquad if \ j \in N^k_i, \tag{1}$$

where $\eta_{i,j} = 1/d_{i,j}$ is a heuristic value that is available a priori, α and β are two parameters which determine the relative *influences* of the pheromone trail and the heuristic information respectively, and N_i^k is the feasible neighbourhood of ant k when at city i. This latter set represents the set of cities that ant k has not yet visited; the probability of choosing a city outside N_i^k is zero (this prevents an ant returning to a city, which is not allowed in the TSP). By this probabilistic rule, the probability of choosing a particular edge (i, j) increases with the value of the associated pheromone trail $\tau_{i,j}$ and of the heuristic information value $\eta_{i,j}$. The numerator of the Eq. 1 is pretty much the same for every ant in a single run, thus, computation times can be saved by storing this information in additional matrix, called *choice_info matrix* as showed in [15]. The random proportional rule ends with a selection procedure, which is done analogously to the *roulette wheel* selection procedure of evolutionary computation (for more detail see [15,19]). Each value $choice\_info[current\_city][j]$ of a city j that ant k has not visited yet determines a slice on a circular roulette wheel, the size of the slice being proportional to the weight of the associated choice. Next, the wheel is spun and the city to which the marker points is chosen as the next city for ant k. Furthermore, each ant k maintains a memory, M^k, called the *tabu list*, which contains the cities already visited, in the order they were visited. This memory is used to define the feasible neighbourhood, and also allows an ant to both to compute the length of the tour T^k it generated, and to retrace the path to deposit pheromone.

After all ants have constructed their tours, the pheromone trails are updated. This is achieved by first lowering the pheromone value on all edges by a constant factor, and then adding pheromone on edges that ants have crossed in their tours. Pheromone evaporation is implemented by

$$\tau_{i,j} \leftarrow (1-\rho)\tau_{i,j}, \qquad \forall (i,j) \in L, \tag{2}$$

where $0 < \rho \leq 1$ is the pheromone evaporation rate. After evaporation, all ants deposit pheromone on their visited edges:

$$\tau_{i,j} \leftarrow \tau_{i,j} + \sum_{k=1}^{m} \Delta\tau_{i,j}^k, \qquad \forall (i,j) \in L, \tag{3}$$

where $\Delta\tau_{ij}$ is the amount of pheromone ant k deposits. This is defined as follows:

$$\Delta\tau_{i,j}^k = \begin{cases} 1/C^k \text{ if } e(i,j)^k & \text{belongs to } T^k \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

where C^k, the length of the tour T^k built by the k-th ant, is computed as the sum of the lengths of the edges belonging to T^k. According to Eq. 4, the better an ant's tour, the more pheromone the edges belonging to this tour receive. In general, edges that are used by many ants (and which are part of short tours), receive more pheromone, and are therefore more likely to be chosen by ants in future iterations of the algorithm.

2.2 The CUDA programming model

Compute Unified Device Architecture (CUDA) [29] is a platform for Graphics Processing Units (GPUs), covering both hardware and software. On the hardware side, the GPU consists of N multiprocessors which are replicated within the silicon area, each endowed with M cores sharing the control unit, and a shared memory (a small cache explicitly managed by the programmer). Each GPU generation has increased CUDA Compute Capabilities (CCC), as well as increasing the number of cores and shared memory size (see Table 1). In conjunction with these developments, power consumption has been reduced by a factor of 2 at each new generation.

The CUDA software paradigm is based on a hierarchy of abstraction layers: the *thread* is the basic execution unit; threads are grouped into *blocks*, and blocks are mapped to multiprocessors. C language procedures to be ported to GPUs are transformed into CUDA *kernels*, mapped to many-cores in a SIMD (Single Instruction Multiple Data) fashion (that is, with all threads running the same code but having different IDs). The programmer deploys parallelism by declaring a

Table 1 CUDA summary by hardware generation since its inception (four generations up to 2015)

Hardware generation and starting year	Tesla 2007	Fermi 2010	Kepler 2012	Maxwell 2014
Multiprocessors per die (up to)	30	16	15	16
Cores per multiprocessor	8	32	192	128
Total number of cores (up to)	240	512	2880	2048
Shared memory size (maximum in Kbytes, per multiprocessor)	16	48	48	96
CUDA Compute Capabilities (CCC)	1.3	2.1	3.5	5.2
Peak single-precision performance (GFLOPS)	672	1178	4290	4980
Performance per watt (approximated and normalized)	1	2	6	12

grid composed of blocks equally distributed among all multiprocessors. A kernel is therefore executed by a grid of thread blocks, where threads run simultaneously grouped in batches called *warps*, which are the main scheduling units.

2.3 Our initial CUDA implementation

In previous work, we developed a CUDA-based ACO implementation, with an emphasis on *data parallelism* [4]. We now summarize this algorithm, as it provides the foundation of the current work.

Recall that our ACO implementation involves ants moving on a graph, deciding where to move next based on simulated pheromone concentrations. When an ant makes a decision on which city/node to visit next, it must calculate heuristic values which are the same for all ants at any one time step (that is, the heuristic information constitutes information on nodes, which must be consistent and accessible to all ants). It makes sense, therefore, to split the computation of heuristic values into a separate *heuristic info kernel*, which is then executed prior to tour construction. Transition probabilities are stored in a two-dimensional *choice matrix*, which is used to inform “roulette wheel” (Monte Carlo) selection by each ant.

In the *tour construction* kernel, each ant is associated with a *thread block*, such that each thread represents a city (or cities) that the ant may visit. This avoids the problem of warp divergences, and enhances data parallelism, as all threads within a block may *cooperate*. The degree of parallelism improves by a factor of $1 : w$, where w is the number of CUDA threads per block.

Finally, the *pheromone kernel* performs evaporation and deposition. Evaporation is straightforward, as a single thread can independently lower each entry in the pheromone matrix by a constant factor. Deposition is more challenging, since each ant generates its own private tour in parallel, and will eventually visit the same edge as another ant. In order to prevent race conditions, we require the use of CUDA atomic operations when accessing the pheromone matrix in this stage.

3 Scaling to heterogeneous clusters

Traditional parallel implementations are not always efficient when ported to heterogeneus systems. They are often inherited from scalable supercomputers, where all nodes in the cluster have the same compute capabilities, and they therefore lack the ability to distinguish computational devices with assymmetric computational power and energy consumption. Differences are not limited to fundamental hardware design (CPUs vs. GPUs), but also occur within the same family of processors. For example, the Kepler family (see Table 1)

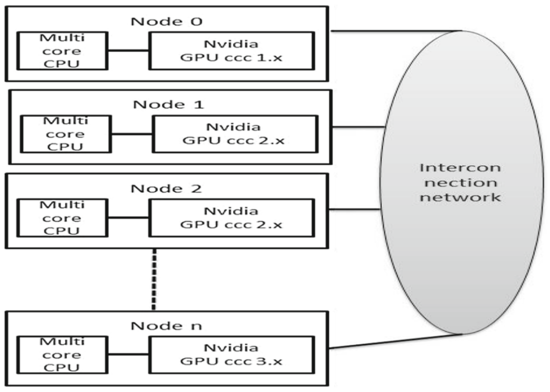


Fig. 1 Heterogeneous system based on different Nvidia GPU generations

includes Tesla K20, K20X and K40 models, endowed with 13, 14 and 15 multiprocessors, respectively (the K80 model even reaches 30 multiprocessors split into two chips). Figure 1 shows a heterogeneous cluster which, nowadays, may include different Nvidia GPU generations, even within the same node.

With this scenario in mind, we introduce a heterogeneity-aware parallelization of ACO applied to the Travelling Salesman Problem as introduced in Sect. 2.1. Our departure point is (1) the CUDA-based implementation of ACO described in Sect. 2.3, and (2) the parallelization strategy proposed by Stützle [39], where independent instances of the ACO algorithm are run on different processors (GPUs in our case, having assorted CUDA Compute Capabilities).

Parallel runs do not incur any communication overhead, and the final solution is chosen across all independent executions, taking advantage of the stochastic nature of ACO algorithms. The execution time of each independent execution may differ, as it depends on (1) the underlying GPU each ACO instance runs on, which is actually unknown at compile-time, and (2) the TSP instance size (the same in principle for all processors, but affected by GPU heterogeneity). Given that the slowest GPU will determine the overall execution time, our mission is to make use of the idle time offered by the most powerful GPUs. Performance and energy differences shown in the last two rows of Table 1 lead us to believe that there is ample room for improvement here.

We have designed an implementation with three main focuses: (1) Resources accounting through MPI processes, (2) performance monitoring via OpenMP threads, and (3) power consumption balance using GPU Boost. We now expand on each of these in the following subsections.

3.1 Resources accounting

First, our algorithm defines a MPI thread for each existing node in the cluster where we run our simulation. Heuristic information about inter-city distances is sent to each node, where supporting data structures are also created to avoid communication overhead. Then each MPI thread creates as many OpenMP threads as GPUs are available on a node, which is easily attained by querying the GPU properties at runtime (using `cudaGetDeviceCount` from the CUDA API) and NVML (Nvidia Management Library).

3.2 Performance monitoring

Secondly, a *warm-up* phase is performed to establish performance differences among all targeted GPUs running the particular TSP instance to be solved. This phase measures, at run-time, the execution time of a small number of iterations of the ACO algorithm (five to ten) in order to detect these differences. Importantly, at this stage, the algorithm is not trying to *solve* the TSP problem in any meaningful sense (five to ten iterations is not enough to do so) but these runs allow us to calculate the performance differences between GPUs. The execution times spent at this *warm-up* phase on all GPUs are reduced to obtain the maximum value using `MPI_Allreduce`. Thus, the *Percent* parameter is eventually determined according to Eq. 5. The slowest GPU will have $Percent = 1$, a GPU two times faster than slowest GPU would have $Percent = 0.5$, and so on.

$$Percent = \frac{Ex.time_{actualGPU}}{Ex.time_{slowestGPU}} \tag{5}$$

We then establish the *time-budget*, which is a threshold that determines the maximum completion time for that ACO algorithm on every GPU. It corresponds to the execution time required to perform *a* number of iterations of ACO on the slowest GPU available. This number of iterations (referred to as δ from now on) is a configuration parameter of our algorithm, and is known by all nodes in the simulation. It is empirically determined to be good enough to find out a good solution to the TSP on our CUDA implementation of ACO. For instance, in our experimental section δ is set to 1000 iterations.

Each OpenMP thread then calculates the slot that it can use for the simulation (γ, with $\gamma > \delta$). This slot can be used for a deeper search (thus computing additional iterations of ACO), or for reducing the power consumption (by relaxing the clock rate in GPU cores). In addition, when $\gamma \geq \delta/2$, the algorithm can even do a restart to avoid becoming "trapped" in a local minimum.

Additional iterations (γ) are obtained by Eq. 6.

$$\gamma = \delta * (1/percent) \tag{6}$$

where "percent" is the performance difference identified among GPUs at warm-up stage, which we have previously explained.

The number of restarts or additional iterations that each GPU may perform is calculated by Eq. 7

$$\gamma = 1/percent \tag{7}$$

as the numerator represents the percent for the slowest GPU, which is always set to 1.

Finally, if we wish to reduce the overall *power consumption* of our simulation, we may use GPU Boost™, which is a new hardware feature introduced by Nvidia from the K40 Kepler GPU onwards. GPU Boost manipulates the clock rate of the GPU cores to trade performance by energy. The idea is to sacrifice time in favour of power consumption when the latter is more critical. Developers can use the `nvidia-smi` shell command to set up the frequency in the GPU, usually exceeding/reducing the nominal value around 20 %. To prevent excessive thermal stress, Nvidia does not allow developers to change this parameter at run-time or within an application, as the Intel SpeedStep™does. Moreover, the GPU is required to work in *Persistence Mode*, which ensures that driver stays loaded even when the GPU has no work to run on it. The range of clocks supported can be queried by the `nvidia-smi -d SUPPORTED_CLOCKS` command, and changed with the `-ac` option (see [32] for more details and a full list of commands). Clock changes require superuser privileges, or developers can use the NVIDIA Management Library (NVML) [30] instead. NVML is a C-based API for monitoring and managing diverse states of NVIDIA GPU devices (including clock settings), without requiring the user to run `nvidia-smi` prior to launching the application on the GPU. The real-time power consumption measurement of individual GPU components using a software approach is only supported by the Nvidia Kepler architecture GPU. This is also done by using NVML, which reports the GPU power usage at real-time. We use `nvmlDeviceGetPowerUsage` command to obtain power usage.

4 Experimental setup

4.1 Hardware environment

For this experimental study, we used the following platforms:

- **On the CPU side:** Four Intel Xeon X7550 processors running at 2 GHz and plugged into a quad-channel motherboard endowed with 128 Gigabytes of DDR3 memory.
- **On the GPU side:** Four GPUs, starting with anTesla C2050 (Fermi generation, approximately 4 years old) and ending with a brand new GeForce GTX 980 (Maxwell generation), with two Kepler models in between (K20

Table 2 Hardware resources and experimental setup used during our executions

	Vendor and type	Intel CPU	Nvidia GPUs			
	Family	Haswell	Fermi	Kepler	Kepler	Maxwell
	Class	Xeon	Tesla	Tesla	Tesla	GeForce
	Model	X7550	C2050	K20c	K40c	GTX 980
	Year	2015	2012	2013	2014	2015
Processing elements	Cores per multiprocessor	(does not apply)	32	192	192	128
	Number of multiprocessors		14	13	15	16
	Total number of cores	8	448	2496	2880	2048
	Clock frequency (MHz)	2000	1147	706	745	1216
Maximum number of GPU threads	Per multiprocessor	(does not apply)	1536	2048	2048	2048
	Per block		1024	1024	1024	1024
	Per warp		32	32	32	32
Register file	32-bit registers (per multiprocessor)		32768	65536	65536	65536
SRAM memory (per multiproc.on GPUs)	Shared (only GPUs)	(32 KB L1D and 32 KB L1I)	16 or 48 KB	16 or 48 KB	16 or 48 KB	96 KB
	L1 cache		48 or 16 KB	48 or 16 KB	48 or 16 KB	(48 KB per block)
	(Shared + L1)		64 KB	64 KB	64 KB	
L2 cache	(shared by all cores)	256 KB	768 KB	1280 KB	1536 KB	2048 KB
L3 cache		16 MB	(does not apply)			
DRAM memory	Size (Megabytes)	131072	2687	4800	11520	4096
	Speed (MHz)	2x666	2x1546	2x2600	2x3004	2x3505
	Width (bits)	256	384	320	384	256
	Bandwidth (Gbytes/s)	42.66	148.41	208	288.38	224.32
	Technology	DDR3	GDDR5	GDDR5	GDDR5	GDDR5
CUDA Compute Capabilities		(d.n.a.)	2.0	3.5	3.5	5.2

and K40), all sharing the motherboard space with PCI-e 3.0 slots to communicate with the CPUs.

Table 2 gives a detailed description of all these platforms. We use gcc 4.8.2 with the -O3 flag to compile on the CPU, and the CUDA compiler/driver/runtime version 6.5 to compile and run on the GPU.

4.2 Benchmarking

We test our designs using a set of benchmark instances from the well-known TSPLIB library [36,43]. All benchmark instances are defined on a complete graph, and all distances are defined as integer numbers. Table 3 shows a list of all targeted benchmark instances with information on the number of cities, the type of distance and the length of optimal tours.

ACO parameters such as the number of ants (m), and those values to set up their behaviour, like α, β, ρ, and so on, are set according to the values recommended in [15]. In particular, $m = n$ (being n the number of cities), $\alpha = 1$, $\beta = 2$ and $\rho = 0.5$.

Table 3 Description of benchmark instances from TSPLIB library (EUC_2D stands for 2D euclidean distance)

Name	Cities	Type	Best tour length
d198	198	EUC_2D	15,780
a280	280	EUC_2D	2579
lin318	318	EUC_2D	42,029
pcb442	442	EUC_2D	50,778
rat783	783	EUC_2D	8806
pr1002	1002	EUC_2D	259,045

5 Experimental results

Given the fact that our techniques establish the experimental setup dynamically, results shown below are platform dependent.

5.1 Performance and workload balance

Figure 2 shows performance differences across different GPU generations when they run several TSP instances.

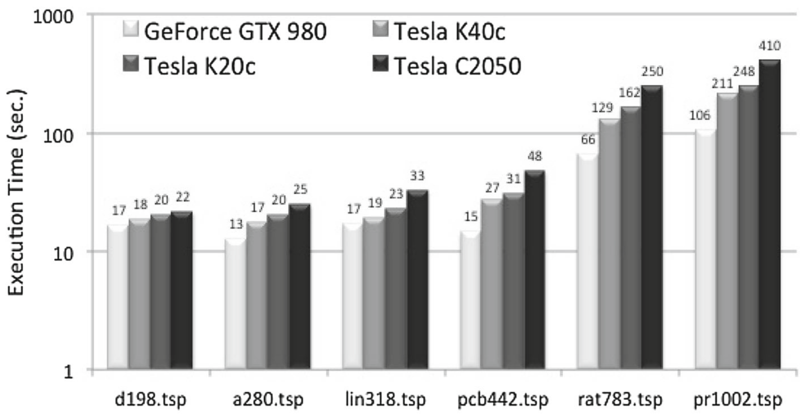


Fig. 2 Execution times in seconds on different Nvidia GPU generations for several TSP instances. Although we have used a Tesla s2050 in our experiments, the figure only shows the performance of a single GPU of the S2050 server (i.e. Tesla C2050)

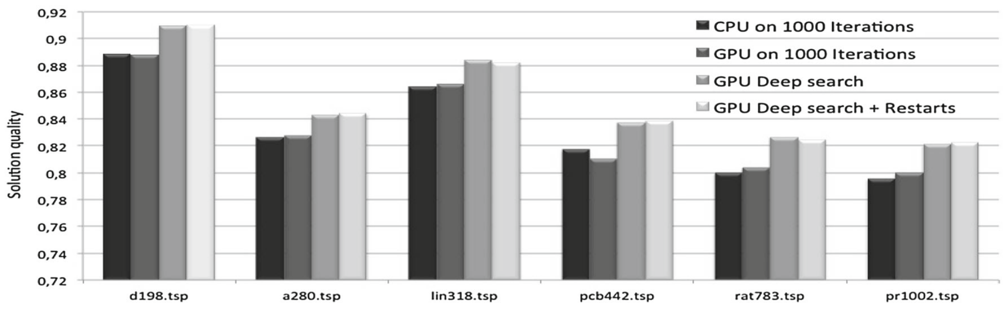


Fig. 3 Quality of the results obtained for different TSP Lib instances, normalized to the optimal solution

Results are recorded for 1000 iterations, and averaged over 10 different runs. The fastest GPU belongs to the latest generation (Maxwell-based GeForce GTX 980), outperforming the slowest GPU by up to a 4.2× factor. This slowest GPU is the Tesla C2050, which determines the *time-budget* for the entire execution. Tesla K20c, the Kepler model, obtains intermediate results, with up to 1.6× gain versus the Tesla C2050.

Results are measured statically for the sake of showing performance differences in a real scenario. However, as described, our methodology includes a *warm-up stage* to calculate these differences at run-time. In previous work [4], more details about performance analysis are given; in particular, we reported up to 20× speed-up factor on average for a Tesla C2050 versus a single-threaded CPU.

We now enhance our parallelization strategy to take advantage of the time that Kepler and Maxwell GPUs are idle, in order to improve the quality of the results. One idea, which we call Deep Search, is to increase the *number of iterations* in order to perform a deeper search within the same time budget. For instance, GeForce GTX 980 carries out 4102 iterations, Tesla K40 carries out 1946 iterations, Tesla K20c carries out 1654 iterations, and Tesla C2050 just 1000 iterations (the time-budget established for this simulation).

Another possibility is to include a restart to avoid being trapped in a local minimum. That is possible if and only if the performance gap is at least twice the slowest GPU performance. These two goals can be merged to create a hybrid approach which we call Deep Search + Restart. Driven by this combination, GeForce GTX 980 may perform up to four restarts of 1000 iterations each (as its percent value is 0.24 on pr1002 TSP instance), whereas Tesla K40 and Tesla K20c only perform a single phase with a deeper search involving 1946 and 1657 iterations, respectively (0.51 and 0.60 % values are not enough to complete two restarts).

Figure 3 shows a tour quality comparison across the sequential run and all parallel strategies for a variety of benchmarks normalized by the optimal solution. The first bar represents the sequential code, written in ANSI C, provided by Stuzle in [15]. This code runs for 1000 ACO iterations on a single-threaded CPU. The second bar is the result quality for our GPU version over 1000 ACO iterations. Figures show that the quality of solutions obtained for these two versions are relatively similar to each other.

The third bar shows our GPU Deep Search strategy, and the fourth bar represents Deep Search + Restart. These two last versions improve results by significant margin within the same time-budget, with a small advantage for Deep Search on

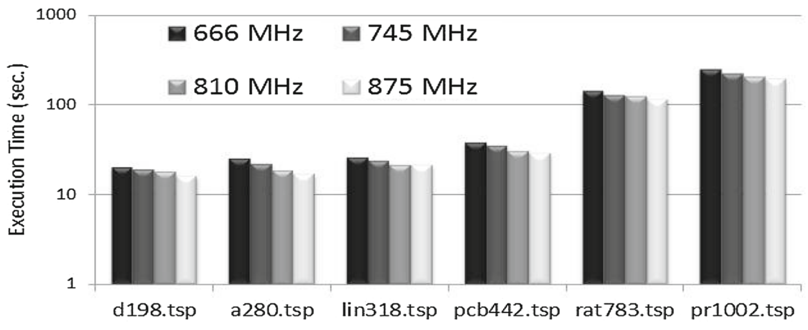


Fig. 4 Execution times in seconds on a Tesla K40 GPU for several TSP instances using different clock frequencies

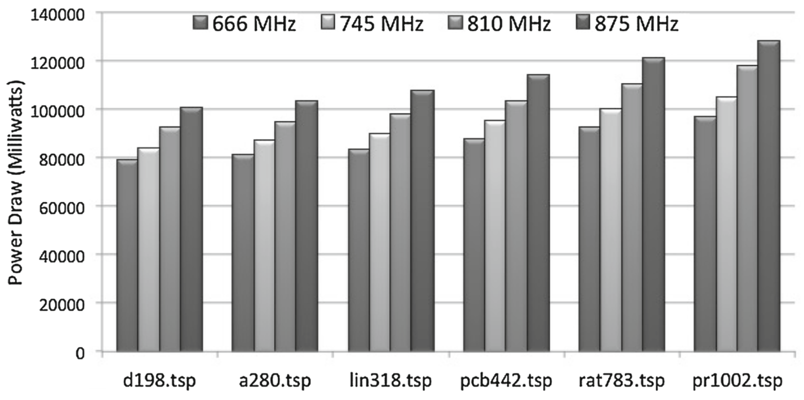


Fig. 5 Power consumption (in milliwatts) measured for the Tesla K40 GPU on different clock frequencies and TSP instances

average. Note that Deep Search performs restarts implicitly, as different searches are executed on different GPUs, whereas Deep Search + Restarts includes restarts explicitly on the same GPU.

5.2 Power consumption

Figure 4 shows the power budget for our simulation under different clock settings. Performance gains reflect up to 1.3 × speed-up factor, in line with the 31 % increment in the clock rate (frequency raises from 666 to 875 MHz).

Figure 5 outlines power consumption in milliwatts for different clock rates. As expected, power consumption raises with higher clock frequencies.

The overall power budget is correlated to the total execution time of the application (see Fig. 6a). However, the 745 MHz clock setting—which is actually set by default on Nvidia's driver for the Tesla K40—is the most energy efficient.

5.3 Power-aware performance metrics

Researchers have proposed metrics combining performance and power measures into a single index. The most popular in low-power circuit design is in the form of ED^n [34], where E is the energy, D is the circuit delay, and n is a nonnegative integer. The power-delay product (PDP), the energy-delay product (EDP) [20] and the energy-delay-squared product (ED^{2}P) [28] are all special cases of ED^n with $n = 0, 1, 2$, respectively.

Intuitively, ED^n captures the energy usage per operation, with a lower value reflecting the fact that power is more efficiently translated into the speed of operation. The parameter n implies that a 1 % reduction in circuit delay is worth paying an n % increase in energy usage; thus, different n values represent varying degrees of emphasis on deliverable performance over power consumption.

Figure 6b shows the Energy Delay Product (EDP) for our ACO simulation, and Fig. 6c the Energy Delay Square Product (triple weight on performance). These couple of metrics prioritize performance over energy. Figure 4 shows that performance differences among different clock frequencies are remarkable, to benefit fastest settings.

6 Conclusions and future work

We present a parallelization strategy tailored to heterogeneous and massively parallel systems. Heterogeneity may limit acceleration and waste energy unless programmers

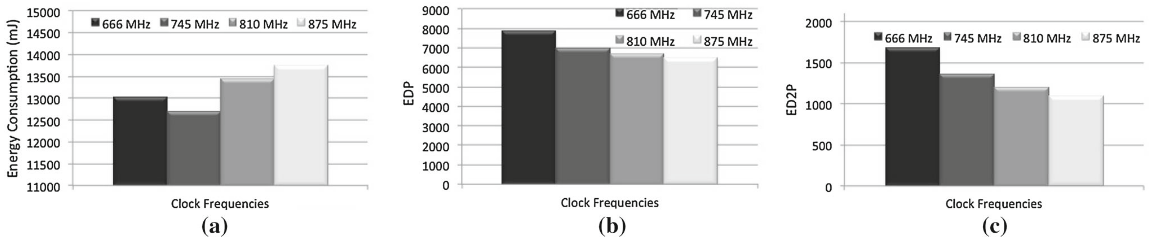


Fig. 6 Energy consumption in $Joules/1000$ (mJ) measured on different clock frequencies for the Tesla K40 GPU. Measurements are taken for the execution on all targeted TSP instances, and averaged over 10 launches. **a** Total energy, **b** Energy delay product (EDP), and **c** Energy delay square product

develop smarter applications to wisely control those features on the road towards an optimal performance/watt ratio. Our proposal cares about accuracy, joules and time equally, deploying those magnitudes on an equilateral triangle managed by a cooperative scheduling of jobs to attain an optimal balance among them at run-time. This makes our strategy particularly useful for non-deterministic algorithms and stochastics behaviours where real-time and/or energy contraints must be fulfilled. With the user setting up those constraints properly, our method may even grant priority to any of the goals composing the metaheuristic.

In a preliminary stage of development, we have illustrated our ideas using Ant Colony Optimization as case study. Given the scalability demonstrated along our experimental study, we foresee an immense potential to extend and refine our methods in future heterogeneous systems. In particular, queries to measure energies and temperatures within the GPU are weak and almost non-existing on low-power devices like Tegra heterogeneous plaforms. Given the long way ahead for improvement and how vendors are enthusiastically endorsing low-power devices, we believe the ideas presented here will greatly benefit from incoming sensors, hardware counters, middleware, libraries and tools, to provide the research community solid pillars to face the expected growth of heterogeneous systems in a much better power-aware manner.

Acknowledgments This work is jointly supported by the Fundación Séneca (Agencia Regional de Ciencia y Tecnología, Región de Murcia) under Grants 15290/PI/2010 and 18946/JLI/13, by the Spanish MEC under grants TIN2012-31345 and TIN2013-42253-P, by the Nils Coordinated Mobility under Grant 012-ABEL-CM-2014A, in part financed by the European Regional Development Fund (ERDF), and by the Junta de Andalucía under Project of Excellence P12-TIC-1741. We also thank Nvidia for hardware donations within UCAM and UMA CUDA Teaching and Research Centers awards.

References

1. Alba, E., Luque, G., Nesmachnow, S.: Parallel metaheuristics: recent advances and new trends. Int. Trans. Oper. Res. **20**(1), 1–48 (2013). doi:10.1111/j.1475-3995.2012.00862.x
2. Carretero, J., Garcia-Blas, J., Singh, D.E., Isaila, F., Fahringer, T., Prodan, R., Bosilca, G., Lastovetsky, A., Symeonidou, C., Perez-Sanchez, H., et al.: Optimizations to enhance sustainability of mpi applications. In: Proceedings of the 21st European MPI Users' Group Meeting, p. 145. ACM (2014)
3. Cecilia, J.M., Garcia, J.M., Ujaldon, M., Nisbet, A., Amos, M.: Parallelization strategies for ant colony optimisation on GPUs. In: Proceedings of the 2011 IEEE International Symposium on Parallel and Distributed Processing, pp. 339–346. IEEE (2011)
4. Cecilia, J.M., Garcia, J.M., Nisbet, A., Amos, M., Ujaldón, M.: Enhancing data parallelism for ant colony optimization on GPUs. J. Parallel Distrib. Comput. **73**(1), 42–51 (2013)
5. Cecilia, J.M., Nisbet, A., Amos, M., Garcia, J.M., Ujaldón, M.: Enhancing GPU parallelism in nature-inspired algorithms. J. Supercomput. **63**(3), 773–789 (2013)
6. Chang, R.S.S., Chang, J.S.S., Lin, P.S.S.: An ant algorithm for balanced job scheduling in grids. Future Gener. Comput. Syst. **25**(1), 20–27 (2009). doi:10.1016/j.future.2008.06.004
7. Chen, Y., Miao, D., Wang, R.: A rough set approach to feature selection based on ant colony optimization. Pattern Recognit. Lett. **31**(3), 226–233 (2010). doi:10.1016/j.patrec.2009.10.013
8. De Michell, G., Gupta, R.K.: Hardware/software co-design. Proc. IEEE **85**(3), 349–365 (1997)
9. Delévacq, A., Delisle, P., Gravel, M., Krajecki, M.: Parallel ant colony optimization on graphics processing units. J. Parallel Distrib. Comput. **73**, 52–61 (2013). doi:10.1016/j.jpdc.2012.01.003
10. Dorigo, M., Di Caro, G.: Ant colony optimization: a new meta-heuristic. In: Proceedings of the 1999 Congress on Evolutionary Computation (CEC'99), pp. 1470–1477. IEEE Press (1999)
11. Dorigo, M.: Optimization, learning and natural algorithms. Ph.D. thesis, Politecnico di Milano, Italy (1992)
12. Dorigo, M., Maniezzo, V., Colorni, A.: Ant system: optimization by a colony of cooperating agents. IEEE Trans. Syst. Man Cybernet. B **26**(1), 29–41 (1996)
13. Dorigo, M., Maniezzo, V., Colorni, A.: The ant system: optimization by a colony of cooperating agents. IEEE Trans. Syst. Man Cybernet. B **26**, 29–41 (1996)
14. Dorigo, M., Birattari, M., Stutzle, T.: Ant colony optimization. IEEE Comput. Intell. Mag. **1**(4), 28–39 (2006)
15. Dorigo, M., Stutzle, T.: Ant Colony Optimization. Bradford Company, Scituate (2004)
16. Dorigo, M., Stützle, T.: Ant colony optimization: overview and recent advances. Handbook of Metaheuristics, pp. 227–263. Springer, Berlin (2010)
17. Garcia, M.P., Montiel, O., Castillo, O., Sepúlveda, R., Melin, P.: Path planning for autonomous mobile robot navigation with ant colony optimization and fuzzy cost function evaluation. Appl. Soft Comput. **9**(3), 1102–1110 (2009). doi:10.1016/j.asoc.2009.02.014

18. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Professional, New York (1989)
19. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning, 1st edn. Addison-Wesley Longman Publishing Co. Inc, Boston (1989)
20. González, R., Horowitz, M.: Energy dissipation in general purpose microprocessors. IEEE J. Solid-State Circuits **31**(9), 1277–1284 (1996)
21. Johnson, D.S., Mcgeoch, L.A.: The Traveling Salesman Problem: A Case Study in Local Optimization. Wiley, New York (1997)
22. Ke, B.R., Chen, M.C., Lin, C.L.: Block-layout design using max-min ant system for saving energy on mass rapid transit systems. IEEE Trans. Intell. Transp. Syst. **10**(2), 226–235 (2009). doi:10.1109/TITS.2009.2018324
23. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of IEEE International Conference on Neural Networks, vol. 4, pp. 1942–1948. IEEE (1995)
24. Komarudin, Wong, K.Y.: Applying ant system for solving unequal area facility layout problems. Eur. J. Oper. Res. **202**(3), 730–746 (2010). doi:10.1016/j.ejor.2009.06.016
25. Krueger, J., Donofrio, D., Shalf, J., Mohiyuddin, M., Williams, S., Oliker, L., Pfreund, F.J.: Hardware/software co-design for energy-efficient seismic modeling. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, p. 73. ACM (2011)
26. Lawler, E., Lenstra, J., Kan, A., Shmoys, D.: The Traveling Salesman Problem. Wiley, New York (1987)
27. Manfrin, M., Manfrin, M., Stützle, T., Dorigo, M.: Parallel ant colony optimization for the traveling salesman problem. Ant Colony Optimization and Swarm Intelligence, pp. 224–234. Springer, Berlin (2006)
28. Martin, A.: Towards an energy complexity of computations. Inf. Process. Lett. **77**, 181–187 (2001)
29. Nickolls, J., Buck, I., Garland, M., Skadron, K.: Scalable parallel programming with cuda. Queue **6**(2), 40–53 (2008)
30. Nvidia Corporation. NVML API Reference ([last accessed 15 November 2014]). http://developer.download.Nvidia.com/assets/cuda/files/CUDADownloads/NVML/nvml.pdf
31. NVIDIA: NVIDIA CUDA C Programming Guide 6.5 (2014)
32. Parallel forall blog. Nvidia CUDA Zone. http://devblogs.nvidia.com/parallelforall/increase-performance-gpu-boost-k80-autoboost/ [11 March 2015]
33. Pedemonte, M., Nesmachnow, S., Cancela, H.: A survey on parallel ant colony optimization. Appl. Soft Comput. **11**(8), 5181–5197 (2011). doi:10.1016/j.asoc.2011.05.042
34. Pénzes, P., Martin, A.: Energy-delay efficiency of vlsi computations. In: Proceedings of the ACM Great Lakes Symposium on VLSI (GLSVLSI). IEEE (2002)
35. Rahman, R.: Xeon phi system software. Intel® Xeon Phi Coprocessor Architecture and Tools, pp. 97–112. Springer, Berlin (2013)
36. Reinelt, G.: TSPLIB—a traveling salesman problem library. ORSA J. Comput. **3**(4), 376–384 (1991)
37. Rozenberg, G., Bäck, T., Kok, J.N.: Handbook of Natural Computing. Springer, Berlin (2011)
38. Shalf, J., Quinlan, D., Janssen, C.: Rethinking hardware-software codesign for exascale systems. Computer **44**(11), 22–30 (2011)
39. Stützle, T.: Parallelization strategies for ant colony optimization. In: PPSN V: Proceedings of the 5th International Conference on Parallel Problem Solving from Nature, pp. 722–731. Springer, London (1998)
40. Stützle, T.: Parallelization strategies for ant colony optimization. Parallel Problem Solving from Nature (PPSN V), pp. 722–731. Springer, Berlin (1998)
41. Stutzle, T., Hoos, H.H.: MAX-MIN ant system. Future Gener. Comput. Syst. **16**(8), 889–914 (2000)
42. Top 500 supercomputer site ([last accessed 15 November 2014]). http://www.top500.org/
43. TSPLIB Webpage (2011). http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/
44. Wolf, W.: A decade of hardware/software codesign. Computer **36**(4), 38–43 (2003)
45. Yu, B., Yang, Z.Z., Yao, B.: An improved ant colony optimization for vehicle routing problem. Eur. J. Oper. Res. **196**(1), 171–176 (2009). doi:10.1016/j.ejor.2008.02.028
46. Zhu, W., Curry, J.: Parallel ant colony for nonlinear function optimization with graphics hardware acceleration. In: IEEE International Conference on Systems, Man and Cybernetics, SMC, pp. 1803–1808. IEEE (2009)

Antonio Llanes obtained his B.S. degree in Computer Science in Univ. of Murcia (Spain, 2006), he also received his M.S. in Univ. of Murcia (Spain, 2010). He is Lecturer at Catholic University of Murcia (Spain) from 2006. Nowadays, he is working in his Ph.D. at Catholic University of Murcia (Spain, 2014–). He has been involved in several regional and international projects, like SENECA and NILS mobility. His main research interests are parallel computing, AI, and bioinformatics applications.

José M. Cecilia received his B.S. degree in Computer Science from the University of Murcia (Spain, 2005), his M.S. degree in Computer Science from the University of Cranfield (United Kingdom, 2007), and his Ph.D. degree in Computer Science from the University of Murcia (Spain, 2011). Dr. Cecilia was predoctoral researcher at Manchester Metropolitan University (United Kingdom, 2010), supported by a collaboration grant from the European Network of Excellence on High Performance and Embedded Architecture and Compilation (HiPEAC) and visiting professor at the Impact group leaded by Professor Wen-Mei Hwu at University of Illinois (Urbana, IL, USA). He has published several papers in international peer-reviewed journals and conferences. His research interest includes heterogeneous architecture as well as bio-inspired algorithms for evaluating the newest frontiers of computing. He is also working in applying these techniques to challenging problems in the fields of Science and Engineering. Now, he is working as Assistant Professor at the Computer Science Department in the Catholic University of Murcia. He is teaching several lectures such as Introduction to Parallel Computing, Object-Oriented Programming, Operative System, Computer Architecture, Computer Graphics; all of them are part of the Computer Science degree.

Antonia Sánchez was born in Cartagena, Spain. She graduated in Computer Engineering at the University of Murcia. She studied a master's degree in Mathematics and Computer science Applied in Sciences and Engineering. During 1998 and 1999 she was a Research Assistant in the Computer and System Dept. at the University of Murcia. Since 1999 is working at Department of Computer Science, Universidad Católica San Antonio de Murcia (UCAM), Spain, where she is Assistant Professor. She recovers different positions of management in her university as Subdirector of the Degree in IT Engineering, Responsible for planning of schedules and spaces. Her main research interests are in Supervisory Control. Nowadays, she is researching in topics such as bioinformatics, and high performance computer among others, focused on databases. She has published papers in international journals and conferences and participated in different research projects.

José M. García is professor of Computer Architecture at the Department of Computer Engineering at the University of Murcia (Spain), and also the Head of the Research Group on Parallel Computer Architecture. He served as the Dean of the School of Computer Science from 2006 to 2012. Prof. García has developed several courses on Computer Structure, Computer Architecture, Parallel Computer Architecture, Peripheral Devices, and Multicomputer Design. He was involved in the "EA-Grid: Euro-Asia United Establishment of Double Degree Master Programme in Grid Computing", which was an Education and Research Network in Grid Computing between the EU and Asia funded by the European Commission. He specializes in Computer Architecture, Parallel Application Processing and Interconnection Networks. He has supervised fifteen doctoral Theses and has published more than 140 refereed papers in different journals and conferences in these fields. Prof. García is a member of several international associations such as HiPEAC, the European Network of Excellence on High Performance and Embedded Architecture and Compilation, and also IEEE and ACM. His current research interests lie in the design of power-efficient heterogeneous systems, and the development of data-intensive applications for those systems (especially bioinspired evolutionary algorithms, and bioinformatics applications).

Martyn Amos is Professor of Novel Computation and Director of the Informatics Research Centre, Manchester Metropolitan University, UK. His research interests include nature-inspired computing, synthetic biology, complex systems and crowd science, and he is the author of "Genesis Machines: The New Science of Biocomputing".

Manuel Ujaldón received his B.S. degree in Computer Science from the Univ. of Granada (Spain, 1991) and his M.S. and Ph.D. degrees in Computer Science from the Univ. of Malaga (Spain, 1993 and 1996). During 1994 and 1995 he was a Research Assistant in the Computer Architecture Dept. at the University of Malaga, where he became Assistant Professor in 1996, Associate Professor in 1999 and credited by ANECA as Full Professor in 2013. Dr. Ujaldon was a predoctoral and postdoctoral researcher at the Computer Science Dept. of the University of Maryland (USA, 1994, 1996–97) and visiting researcher at Biomedical Informatics Dept. of the Ohio State University (USA, 2003–08). He was also Conjoint Senior Lecturer at the University of Newcastle (Australia, 2012–2015). He has published 8 books on computer architecture and around 100 papers in international peer-reviewed journals and conferences. He was awarded CUDA Fellow by Nvidia in 2012, and over the last five years he has been involved in more than 100 activities about GPU computing worldwide, including 20 invited talks and 17 tutorials in ACM/IEEE conferences. His main research interest are GPGPU computing for image processing, biomedical applications and evolutionary computation.

2.3 Soft Computing Techniques for the Protein Folding Problem on High Performance Computing Architectures

Título	*Soft Computing Techniques for the Protein Folding Problem on High Performance Computing Architectures*
Autores	Antonio. Llanes, Andrés Muñoz, Andrés Bueno-Crespo, Teresa García-Valverde, Antonia Sánchez, Francisco Arcas-Túnez, Horacio Pérez-Sánchez y José M. Cecilia
Revista	Current Drug Targets
Páginas	.
Año	2016
Estado	*Aceptado*

Contribución del Doctorando
Antonio Llanes Castro, declara ser el principal autor y el principal contribuidor del artículo *Comparative evaluation of platforms for parallel Ant Colony Optimization.*

Current Drug Targets, 2016 1

Soft Computing Techniques for the Protein Folding Problem on High Performance Computing Architectures.

Antonio Llanes[a], Andrés Muñoz[b], Andrés Bueno-Crespo[a], Teresa García-Valverde[b], Antonia Sánchez[a], Francisco Arcas-Túnez[b], Horacio Pérez-Sánchez[a], José M. Cecilia*[a]

[a]*Computer Science Department, Bioinformatics and High Performance Computing Research Group (BIOHPC), Universidad Católica San Antonio de Murcia (UCAM). Campus de los Jerónimos, s/n Guadalupe 30107 (Murcia) - Spain.*
[b]*Computer Science Department, Universal Knowledge Enhancement by Multidisciplinary Implementation (UKEIM), Universidad Católica San Antonio de Murcia (UCAM). Campus de los Jerónimos, s/n Guadalupe 30107 (Murcia) - Spain.*

Abstract: The protein-folding problem has been extensively studied during the last fifty years. The understanding of the dynamics of global shape of a protein and the influence on its biological function can help us to discover new and more effective drugs to deal with diseases of pharmacological relevance. Different computational approaches have been developed by different researchers in order to foresee the three-dimensional arrangement of atoms of proteins from their sequences. However, the computational complexity of this problem makes mandatory the search for new models, novel algorithmic strategies and hardware platforms that provide solutions in a reasonable time frame. We present in this revision work the past and last tendencies regarding protein folding simulations from both perspectives;hardware and software. Of particular interest to us are both the use of inexact solutions to this computationally hard problem as well as which hardware platforms have been used for running this kind of *Soft Computing* techniques.

Keywords: Soft Computing, Protein FoldingProblem, Protein Structure Prediction, Parallel Computing, Distributed Computing, Metaheuristics, High Performance Computing.

1. INTRODUCTION

1.1. Protein folding problem

A noteworthy interrelation exists at the molecular level between the structure of a protein and its biological function, and in biochemistry we can find a diversity of such functionalities. It is well known that the mechanism by which a protein exerts its biological function is directly related to its native three-dimensional structure, which is precisely codified on its sequence of aminoacids[1].

Being able to solve this problem is of outstanding importance since having access to the information related to the structure of these biomolecules, allows for being able to explain how bioactive compounds can modulate their biological activity and therefore paves the way to the drug discovery process.

In addition, one can find many more sequences than structural information, mainly due to the last advances in high-throughput sequencing and personalized medicine efforts [1-2]. Thus, a noticeable interest exists in the development of methodologies that, exploiting only information extracted from sequences, can predict in detail the structure of proteins.

1.2. The simulation problem

Finding accurate solutions of the PSP problem is very challenging, and researchers have developed many different approaches in order to solve it by means of computer simulation. These simulation methods receive as input a protein sequence and outputtheir predictions for the protein structures.

Existing computer simulation methods for the PSP problem can be classified depending on:

a) the degree of details used in the protein model that undergoes the computer simulation:there are detailed all-atom models that try to accurately represent and describe bonded and non-bonded interactions present in the folded protein structure. From the other side, coarse grain models can also be considered. In the last decades, the first theoretical hypotheses concerning protein folding, such as those stated by Dill et al.[2] were proposed. Main underlying ideas indicated that forces implied in the protein folding process were related with the intercommunication between their aminoacids. But recently, a theory that states that non-bonded interactions significantly contribute to the dynamics of this mechanism, is being accepted, and researchers are showing interest to the use of very simple models of proteinsand other biological macromolecules. In this context, the study of these coarse grain models through computer simulation techniques can yield interesting results when their predictions are contrasted with empirical measurements.

b) the scoring function used for the estimation of the interactions between the elements of the protein model:thismathematical function will mainly depend on the type of protein model used, and for a given model, it might contain different sets of parameters that describe the relative intensity of the interactions between the different elements of the protein model. Its derivation or construction depends usually on physical theories or statistical analyses performed on previously available protein structures.

c) the algorithm used for the global optimization problem of the scoring function: once a given protein model and scoring function have been chosen, a optimization methodis selected for working on the global optimization problem. It concerns the search of the most optimal value of the scoring function, since we assume that this value will correspond to the native protein fold [1]. Here it is possible to use methods that take into account the dynamics of the system, such as Molecular Dynamics [3], or stochastic methods that try to solve the optimization problem not taking into account the dynamics of the system [4]. The former is more realistic, but at the same time it is more computationally demanding, whereas the latter is much faster, but by using it we lose information about the evolution of the system.

Once we have chosen a model, scoring function and optimization algorithm, we can still consider what is the fastest way to carry out the required simulations depending on the available hardware architectures.

1.3. Combination of models, algorithms and HPC.

The choice of model and its associated algorithm is mainly motivated by the required objectives, but it is also constrained by the computer hardware characteristics attainable in the relevant time frame. One of the most widely studied models of protein folding is the hydrophobic-hydrophilic (HP) model introduced by Dill [2]. In the description of the HP model, the different amino acids that form the macromolecular chain can be seen as a discretized conformation in a three-dimensional grid or lattice. Here, one of the most relevant underlying assumptions is that hydrophobic forces contribute considerably to the folding process, and the protein chain is modeled as an array of hydrophobic or hydrophilic chains (H or P for nonpolar and polar, respectively). Then, the most optimal protein conformation is the one that augments that number of nonpolar residues that are contiguous. In this case the folding process can be described as a minimization of the free-energy of the system, and it can be considered as NP-hard problems [5]. This implies that such problems can not be efficiently processed by a computer (for insights we refer the reader to [6,7]).

Models and their associated algorithms should not be selected in isolation though. They must be evaluated in the context of the computer hardware environment they are going to run on. Algorithms that are designed to leverage maximum performance on a particular hardware architecture could become less effective on a different hardware. Therefore, the selection must be made carefully, and may change over time [8]. This issue even grows exponentially nowadays as we are witnessing the consolidation of heterogeneous systems (i.e., systems that use more than one kind of processors), mainly motivated for the exacerbated power consumption in current microprocessors, and trying to follow the wake of Moore's law. Such heterogeneity is found at different levels from laptops to large-scale computers like supercomputers, clouds, etc, and also where it emerges naturally is in the low-power devices market such as smartphones, tablet and so on. [9]. This emergent landscape of computation in the high performance computing market offers new opportunities in the simulation of protein structure prediction. However, the recent 2014 United States Department of the Energy (DoE) report on top ten exascale research challenges [8] shows as one of the main challenges for next years the design of *Exascale algorithms*. It will require redesigning, or even reinventing the algorithms used in current scientific and engineering codes, and potentially reformulating the science problems to leverage billion-way parallel architectures.

In this sense, S*oft Computing* techniques are designed to deal with the difficulties which arise in real problems by including several factors like several levels of imprecision into the calculation and taking this into account to even change the granularity of the problem or somehow relaxing the goal of optimization at some point[10]. The source of inspiration of *Soft Computing* is based on the natural processes, trying to formalize such processes to solve a particular task. Techniques within this field include neural networks, genetic algorithms (GA), evolutionary algorithms, etc., having many of them a common ingredient in their definition: parallelism as the way of speeding-up simulations and providing practical implementations for a feasible search of a single, unified and parameterized solution.

This review article shows the last tendencies on the prediction of protein structure by computer simulation and our perspectives for the forthcoming years. We focus on both the *Soft Computing* techniques that have been applied to coarse-grain protein models, such as the HP-model since it is one of the most widely used coarse-grain models in the literature, and also the underlying hardware and programming models that have been used to execute those algorithms. The paper is structured as follows: Section 2 briefly introduces the reader into the main concepts underlying this review. Section 3 shows the *Soft Computing* techniques applied to protein folding methods before discussingin Section 4 about new trends in novel algorithms and architectures related to this problem. The paper finishes with some conclusions on the current state of the art for this topic.

2. BACKGROUND

2.1. Benchmarks in protein structure prediction.

In order to test the accuracy and convenience of PSP methods it is necessary to have control data (benchmarks) so that we can check whether our predictions are reliable or not. If our particular PSP model, scoring function and algorithm can reproduce the structure of proteins for which experimental structural data is available, we can continue forward and start to make predictions for sequences for which structures are still unknown. It is therefore of outstanding importance to test our PSP methods against all possible available benchmarks.

The field of PSP benchmarks can be usually divided into experimental and synthetic ones. When working with detailed atomic models, we will be able to compare them with structural data from online public databases such as Protein Data Bank (PDB) [11]. In order to test the accuracy of protein structure prediction methods, the current "gold standard" rule is to compare the predicted structure with the experimental one, and calculate the RMSD (Root Mean

Structure Deviation) between them. This is only possible when protein structures have been obtained by experimental methods such as X-ray crystallography, nuclear magnetic resonance, or cryo electron microscopy, and deposited in public access databases such as PDB.

In the case of coarse grain models we have two options. The first one is to convert them to all-atom models and then compare with experimental structures from PDB, and the second one is (when the first possibility does not exist) to compare them with synthetic data obtained previously from other researchers who have performed an exhaustive search of the solution space of the problem.

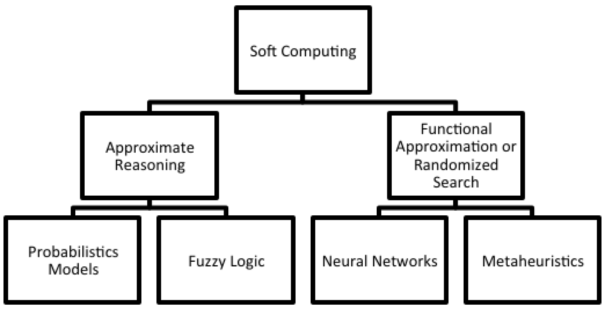


Figure 1 - Classification of Soft Computing techniques

Lastly, and independently of the detail of the method used, we might be also interested in benchmarking the computational speed of our PSP method, depending on its hardware implementation, programming language used, etc. This is also very relevant since the computational performance of the method, and the availability of computational resources the researchers have access to, will dictate the size of the systems we want to study.

2.2. Soft Computing techniques

From the algorithmic point of view, traditional hard computing techniques are based on three main objectives: precision, certainty and rigor. These requirements make the computational cost of such algorithms very costly, particularly to deal with real problems where the input size grows exponentially. Actually, this is the departure point of Soft Computing that tries to overcome the main difficulties in real problems, with the thesis that precision and certainty are sometimes unapproachable, and thus it may include the tolerance for imprecision and uncertainty [12,13]. Therefore, Soft Computingcan be defined as the antithesis of what we have called Hard Computing. We refer the reader to [10,14] for a more detailed definition of Soft Computing.

Although several classification of *Soft Computing* techniques have been proposed in the literature [12,13], Figure 1 shows a consensus among all of them. Since the fuzzy boom at the beginning of 90's, many methodologies based on these techniques have been proposed in the literature [15,16]. Although *Soft Computing*is a term introduced by Zadeh in 1994 [17], previous work was done by the definition of fuzzy sets [18]. Fuzzy sets are the pioneer paradigm in *Soft Computing*.They have been included in many other *Soft Computing*methods to provide hybrid methods. Among these new methods we may highlight Neural Networks [19], Support Vector Machines [20], Fuzzy Logic [12], Metaheuristics [21] (including techniques such as Evolutionary Computation [22, 23] or Swarm Intelligence [24]), to name just a few. There are a large number of algorithms within the umbrella of *Soft Computing*. They are applied to different fields such as symbol and pattern representation to enrich knowledge representation, machine learning for flexible knowledge acquisition, and inference by flexible knowledge processing. Moreover, *Soft Computing*techniques can be offered as a tool to interact with or they can be integrated in a larger framework where they provide unified and hybrid architectures.*Soft Computing* has been successfully applied to solve problems within the field of bioinformatics [25-27]. However, the large data sets generated from biological experiments and new high-throughput technologies make mandatory that modern *Soft Computing* approaches will be scalable across large-scale problems. In Section 3, we briefly introduce the *Soft Computing*techniques that have been applied to the protein folding problem.With that in mind, this paper focuses on the functional approximation or randomized search part of *Soft Computing*(see Figure 1) as it is gaining popularity during the last few years.

2.3 HPC platforms and programming models

In what follows, we reprise and update our vision of the High Performance Computing (HPC) arena, which was first given in [28]. HPC techniques and platforms are being applied for addressing many scientific challenges that would be otherwise very difficult to solve. The number of calculation required for this kind of scientific applications requires large computing resources. Just to mention an example, Anton is a supercomputer specially designed to simulate protein movements that could aid the drug design process [29].

However, we are witnessing a revolution in this areaas the Moore's law that has driven the development of new microprocessors in the last years [30,31], which is based on the idea that the number of transistors in an microprocessor would be doubled every two years, is running up against the laws of physics [32,33]. While a new microprocessor technology come up into the market, the industry has taken the steady transition to heterogeneous computing systems [34], with heterogeneity representing systems where nodes combine traditional multicore architectures (CPUs) and accelerators (mostly represented by GPU computing movement [35]or Intel Xeon Phi cards [36]).Heterogeneity limits system growing as it cannot be performed in an incremental way anymore. In particular, concepts like energy consumption, programmability, scalability, data location, and reliability become challenges for tomorrow's cyberinfrastructure [37]. This Section summarizes current trends in HPC platforms that are commonly used within the field of Bioinformatics. Of particular interest to us are,manycore architectures like Graphics Processing Units (GPUs), clusters of computers also known as

Supercomputers and cloud and distributed computing architectures.

2.3.1 GPU computing

Motivated by the computational demand of the videogame industry, Nvidia introduced in 2006 a graphics processing unit (GPU), codenamed CUDA (Compute Unified Device Architecture), which made available the computational power of those novel computing architectures to the scientific community. Nowadays, they have become a compelling alternative to the traditional architectures as they deliver high rates of floating point performance and massively parallelism at a very low cost, and thus democratizing the high performance computing (HPC) arena [38, 39].This movement was termed “GPGPU” which stands for *General-Purpose computation on Graphics Processing Units*. The GPGPU has promoted the use of this novel and massively parallel architecture in a wide range of applications, particularly in Bioinformatics, where parallelism and arithmetic intensity are common denominators in almost every application (we refer the reader to GPU application catalog provided by Nvidia[40]).

Following this trend almost all microprocessor company(e.g. ATI/AMD, Intel, etc) have developed their own hardware alternatives designed specifically foraccelerating general purpose applications.Among them, we may highlight Tesla-based GPUs from Nvidia, Firestream is ATI/AMD alternative and finally the new Intel Xeon Phicoprocessor which is based on Many Integrated Core (MIC) architecture. Along with these hardware components, those companies have also provided new programming models to easily leverage the horsepower of these emergent technologies.The first programming model for GPGPU was CUDA [35] (Compute Unified Device Architecture) provided by Nvidia that is specifically developed for programming Nvidia’s GPUs. Nvidia has a wide scientific community behind CUDA, and it offers several educational and research communities to promote the development of scientific applications with CUDA on Nvidia's GPUs. ATI/AMDfirst offered a programming model called Stream Computing which is not supported anymore and Intel relies on vectorization instructions based on X86programming. In 2008 the Khronos Group developed an open standard for parallel programming on cross-platform heterogeneous systems, called OpenCL[41]. OpenCL is an attempt to provide a standard programming language that allows multiplatform development on different devices like GPUs, accelerators, multicore systems, etc.

All of those novel programming models provide an easier way to leverage massively parallel architectures. However, programmers still have to deal with a new programming paradigm, which is rather different to the traditional sequential-basedarchitectures [42]. Moreover, those computing architectures are nowadays plugged into the motherboard through PCI Express bus. This fact provides heterogeneous computers that may have a traditional CPU and other computing devices like GPUs or accelerators. Each of these processors have their own memory spaces, different instruction set architectures and communication latencies. Therefore, programmability here is not an easy task.

Currently, the scientific community is looking for new programming models and tools that hide those inherently hardware particularities and provide an easier and faster way to develop application on this new landscape of computation. There are two different trends to provide such abstraction layer. First, the execution of a given program efficiently on different devices from a single source code [43,44]. Second, the API development to extent traditional programming languages like OMPSs for OpenMP [45],or OpenACCAPI [46], which establishes several directives to specify loops and regions of code in standard programming language such as FORTRAN, C++, C.

2.3.2 Supercomputers

High performance computer (also known as Supercomputers) are those computers that are developed to deal with great challenges within the industry and academia. Statistics on supercomputers are provided in the TOP500 list [34], where information about the number of systems installed, the performance of each system or their location among others is provided to manufactures and (potential) users. Supercomputers within TOP500 are highly involved in Bioinformatics research. For instanceTianhe-II and Titan, two top supercomputers in this list, are heavily involved in developing bioinformatics domain problems. Tianhe-II is addressing the needs of genetic engineering and biopharmaceutical simulations. Moreover, Titan is being used for molecular similarity to provide a description of membrane fusion. This is actually one of the main ways for molecules to enter or exit from living cells. Other leading examples are the supercomputer installed at the Leibniz Supercomputer Center in Monaco (SuperMUC) and the Piz Daint the CSCS/Swiss Bioinformatics Institute. The former supercomputer is commonly used for running bioinformatics applications like analysis of linkage disequilibrium in genotyping. The later has been successfully applied to run a challenge of evolutionary genomics based on calculating selection events in genes achieving several orders of acceleration.

Supercomputers are adopting the use of accelerators to speedup arithmetic intensive parts of the applications. Actually, five of the ten fastest supercomputers in Top 500 list [34] include accelerators in their designs. Those accelerators are basically limited to Intel Xeon Phi and Nvidia GPUs architectures. However, these accelerators increase the overall power consumption of the system which is actually a big issue, particularly for large-scale datacenters where Total Cost of Ownership is mainly influenced by the power supply [47]. Indeed, the inclusion of these accelerators can increase the power consumption of a cluster node up to 30%.

However, the total cost of ownership is not the only concern to reduce overall power consumption in supercomputers. Actually, this is now becoming mandatory as the carbon footprint of those systems is actually very high, and the reduction of carbon emissions is one of the main challenges in the last 2015 United Nations Climate Change Conference where the International Trade Union Confederation has called for the goal to be "zero carbon, zero poverty".For instance, the power consumption of TI supercomputers companies such as Google or Facebook, consumed about 0.5% of the overall power consumption in the world during 2005. If the cooling and power distribution were also taken into account then the power consumption increases up to 1% [48]. The high performance computing community is trying to develop supercomputers and infrastructures that reduce power consumption. Actually, the GREEN500 list [49]shows the 500 most power efficient supercomputers in the world. Indeed, we are envisioning a shift from the traditional metrics like FLOPS (FLoating point Operations Per Second) to FLOPS per watt.

Virtualization techniques are placed as the main way to reduce the overall power consumption in supercomputers, as they enable to have several virtual machines running at the same time in the same real hardware. Actually, datacenters are adopting this new trend for several applications.Of course, virtualization may have a performance impact. For instance, Amazon Elastic Cloud Computing EC2offers a virtual infrastructure of 26496 cores, achieving484,2TeraFLOPS for the High Performance Linpack benchmark, placing the cluster at position 101 in the November 2014 Top500 list but this is actually a tradeoff the scientific community has to deal with.

2.3.3 Cloud and distributed computing

As previously explained, the TCO of having an in-housesupercomputer is very high and it is not affordable for small institutions [50]. Cloud computing is ubiquitous and energy-efficient computer organization by its definition [51], in which virtualization is the main ingredient to obtain great energy reduction. In cloud computing platforms, services run remotely in a ubiquitous and distributed computing set of computers (a.k.a cloud) that may provide scalable and virtualized resources. In this way, heavy workloads can be migrated to other virtual nodes of the cloud, providing higher levels of hardware utilization [52]. Cloud providers offer their resources in a pay as you go fashion. Actually, it can be seen as an alternative to physical infrastructures but this is only useful for a specific amount of data and target execution time.

Cloud computing propose an on-demand scenario where users only pay for the computational time usersutilize for running their applications. There are several cloud computing models: infrastructure as a service (IaaS), platform as a service (PaaS), software as a service (SaaS), and Data as a service (DaaS). Among them, IaaS is the most commonly used model while the other may provide other level of abstraction [53]. In the cloud, developers may use several instances and thus they can create a parallel cluster on demand. Like real hardware scenarios, those clusters can be programmed using libraries such as the Message Passing Interface (MPI). Those instances can be also used in a batchprocessing mode, launching several instances of a program and so on.

Cloud computing platforms are very interesting for bioinformatics practitioners mainly for the flexibility and the cost-effectiveness. Truth be told, this actually depends on the workloads they expect to run on the cloud but, in general, small-medium bioinformatics laboratories, which may perform bioinformatics analysis are moving to this technology as they avoid cost and issues of having an in-house computer infrastructure [54]. An alternative solution is represented by Hybrid Clouds that have both the scalability offered by cloud computing and the control and ad-hoc customizations supplied by in-house computers [55].

Those distributed solutions are evolving in the era of Big Data to frameworks like Hadoopthat allows distributed access to files. These frameworks are well suited for distributed algorithms such as MapReduce [56]. MapReduce is a programming environment to manage large data sets with a parallel, distributed algorithm on a cluster. For example, the PSIPRED [57] protein analysis workbench leverages the Hadoop implementation of MapReduce to launch several services to perform the execution of prediction methods in a large-scale system. Moreover, MapReduce has been also applied to provide an enhanced framework where parallel genetic algorithms target the protein folding in distributed environment [58].

Finally, some efforts have been done in the volunteer computing arena that is noteworthy to remark. Among them, we may highlight Folding@Home [59] which is a volunteer computing project that tries to solve the protein folding problem by means of collective human knowledge. Folding@Homehas been used in several medical researcheslike to cure Alzheimer's disease, Huntington's disease, and many forms of cancer, among other diseases. This project is pioneer in the use of many novel computing platforms such as Graphics Processing Units, CellBe processor, multi and many core systems through MPI and OpenMP language, as well as some smartphones for distributed computing and scientific research [60].

Kondow and Berlich [61] runs particle swarm optimization (PSO) on cloud for the simulation of proteins three-dimensional structure. They simulate all-atom force field using ArFlock library, aimed at finding the folded state of two proteins of different sizes starting from completely extended conformations.

2.3.4. Multiagent systems

Multiagent systems (MAS) can be also considered as a platform to tackle Bioinformatics problems such as protein folding. As defined in [62], they combine a flexible and high-level paradigm with a technology developed at the intersection between artificial intelligence and distributed computing. A typical MAS is composed of several autonomous entities –agents— that can communicate and interact among them in a competitive or cooperative manner. MAS are especially useful for simulation tasks, including the behavior of biological systems [63], where the different parts of the system have some individual features that distinguish it from the rest.

There are several works in the literature that have adopted MAS to address the protein folding problem. For example, a MAS using an independent energy model where every amino acid is identifying with an agent is presented in [64]. These amino-agents lay at the bottom level of the MAS architecture, their positions being coordinated by a set of cooperative agents in a higher level. Amino-agents movements are based on Monte Carlo-like criterion and hill-climbing strategy (to avoid local minimum). Coordination agents act as orchestra director suspending amino-agents movements when they are not improving a global strategy. These coordination agents offer the possibility of designing complex heuristics depending on external information and on the search history. Thus, external knowledge from databases can be injected to coordination agents to force amino-agents to make movements oriented to improve the energy results. Experimental tests performed in this MAS show that the proposed coordination level always introduces a better performance, but the energy function used is too coarse to provide good biological model.

Moreover, a MAS based on reinforcement learning for solving bidimensional protein folding is showed in [65]. In this case there are several basic agents trying to solve the problem using the Q-learning algorithm [66] based on their local knowledge and a reduced set of supervisor agents that synchronize and coordinate the basic agents according to the current best solution. Basic agents are distributed across multiple processes/machines and they use a blackboard to communicate with their supervisor agents. Authors claim that this distributed proposal greatly reduces the computational time employed in the training phase of the Q-learning algorithm with respect to a non-distributed approach. However, it must be further investigated how to preserve the accuracy of the results using a MAS. Finally, in [67] a competitive approach among agents is taken to implement an architecture named Discovery Bus aimed at modeling molecular design workflows. This MAS follows the quantitative structure–property relationships (QSPR) model to predict the properties of novel proteins.

An excellent discussion of pros and cons when using MAS in protein folding is given in [68]. The main advantage of this approach resides in its flexibility: addition and removal of agents could be done at run-time and therefore it is possible to change the structure of the experiment (e.g., the protein’s structure). In practice, not only may the limit conditions and the simulation constraints be changed dynamically, but also elements from the structure could be added and removed during the simulation. This fact augments the potentialities of simulated experiments, enabling a virtual manipulation of the system simulating the protein folding, even when this is not possible in reality. This property extends in silico experiments to in virtuo experiments, i.e., not only enabling the change of values of the parameters characterizing simulations, but also the structure of the experiment during run-time in an easy manner thank to MAS features. As for the main disadvantage of the use of MAS in this topic, it has been criticized that simulations performed by means of multiagent systems are not totally validated against real data, diminishing their credibility. Thus far, works in this area have focused on the reliability of MAS proposals from a qualitative point of view, showing that multiagent-based simulations are tantamount to other approaches. However, a quantitative validation must be performed to take MAS as a prominent alternative to protein folding.

3. Implementation of protein folding methods.

This section summarizes main contributions on the field of *Soft Computing*applied to the protein folding simulation. Particularly, we focus on the functional approximation or randomized search part of *Soft Computing*; i.e. Artificial Neural Networks and Metaheuristics, applied to the protein folding problem.

3.1. Artificial Neural Networks and SVM.

Artificial Neural Networks (ANNs) have been widely used in the protein folding field. Specifically, the most relevant types of ANNs are the feedforward neural networks [69] and recurrent neural networks [70]. ANN can learn tasks without needing much prior knowledge, and moreover they are tolerant to errors and noisy data. While the most common use of ANNs in protein folding has been devoted to detect secondary structures [71-73], they have been also employed in other tasks such as predicting the posttranslational modifications [74-76]; to identify disordered regions [77]; to predict metal binding sites [78,79]; to assign sub-cellular localization [80-82];classification of proteins into functional classes [83]; reconstructing protein structures [84] and protein class prediction [73,85], among others.

Regarding the prediction task, classifying secondary structure is an easy job for a neural network, as for example to learn to distinguish between alpha-helices and beta-strands models. This classification allows detecting the most three-dimensional structures as they are based on secondary ones. Although the alpha-helices and beta-strands is the main approach in the prediction task, there are some other papers that propose classifications among more than two classes [70,86,87]. Regarding the databases used by neural network, the most popular are the Protein Data Bank (PDB) [11] and the Structural Classification of Proteins dataset (SCOP) [88].

A major advance in the way in which the datasets are treated is to add sequences that are homogeneous to those that are being studying [89]. For example, given the same family of proteins, they share similar structural and functional features. For ANNs, this fact provides additional information in the inductive learning process that improves the task learning. This method is known as Evolutionary Information, however to find these homogeneous sequences is not trivial. For this research line, it is very popular the PSI-BLAST program [90].

Support Vector Machine (SVM) can be focused on the same field of work than ANNs for protein folding [86,88,91], although SVM presents a much better performance for regression against classification in protein folding recognition [92]. Furthermore, they have been used to estimate the significance of the sequence-template alignments [93] and protein secondary structure prediction [94].

It is worth mentioning that although neural networks have been widely used for protein folding, they have not been combined with high performance computing because the prediction of secondary structure do not imply a large computational complexity. However, new trends in neural networks such as Deep Learning [95] have called for reconsidering high performance in the field of neural networks due to its computational complexity. In this sense, Deep Learning has been proposed to make use of graphical processing units (GPUs) and CUDA parallel computing. Hence, Deep Learning has been used for sequence-based residue–residue contact prediction [96] and later for protein secondary structure prediction [97]. These proposals have been implemented using CUDAMat [98], a Python library that provides methods of fast matrix calculations on CUDA-enabled GPUs, providing high-level access to computing cores of graphics processing units.

3.2 Metaheuristics.

There are different approaches to classify metaheuristic algorithms in the literature. A good review of metaheuristic classification can be found in[99], depicted here inFigure 2. This classification takes into account five different features of such algorithms, namely their origins;the number of solutions used at the same time; the way the objective function is used; the neighborhood structure; and the use they make of the search history.

Depending on their origins, a new trend in designing metaheuristics concerns nature-inspired methods. These methods take as a source of inspiration biological or physical principles. Nature-inspired methods are very attractive for practitioners in high performance computing, as they are inherently parallel in definition (e.g.they may be inspired by a “swarm”-like schema that uses several agents to optimize a function). Ants, bees and fireflies are only some examples of populations that inspired algorithms based on their social behavior. Those algorithms rely on swarm to deal with complex problems [100,101]. Despite of this trend, in the last part of this sectionare introduced the most important non-nature inspired algorithms applied to the PSP problem, such as local search methods.

Regarding the number of solutions used at the same time, we can find algorithms working with a single solution or trajectory (e.g.,Tabu Search) or with the evolution of a set of solutions (e.g., Genetic algorithms). On the other hand, some metaheuristics define static objective functions that do not change during the algorithm execution (e.g., Genetic algorithms), whereas others may be modified during the search trying to escape from local minima (e.g., Guided Local Search).

Metaheuristics may be also classified depending on their neighborhood structure. The one-neighborhood structure does not change the fitness landscape topology during the execution, while in the various neighborhood search it is possible to expand the search among different fitness landscapes. Finally, the use of memory in the metaheuristic is another discriminative feature, separating into algorithms that take into account previous states to perform the next action orthose that use a Markov process to decide the next action only based upon the current state.

In this paper we have adopteda classification of metaheuristics based on origins as it is one of the most used and easy to understand.

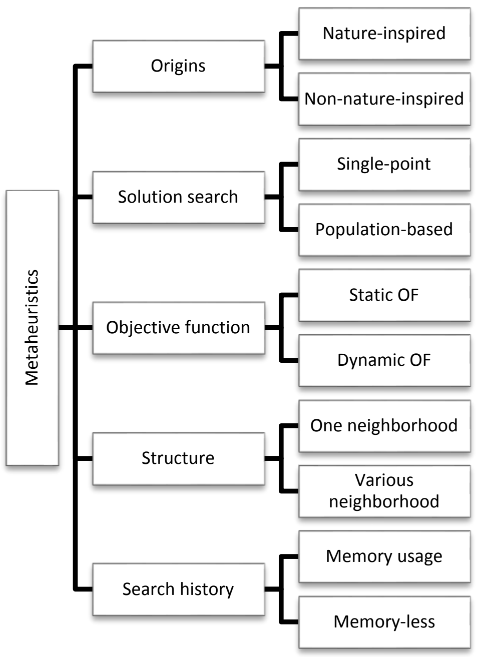


Figure 2Classification of metaheuristics techniques

Next sections review the main metaheuristics employed in protein folding.

3.2.1.Nature-inspired metaheuristics

Ant Colony Optimization

One nature-based method that is proving to be increasingly popular is *ant colony optimization* (ACO) [102,103].This algorithm is based on foraging behavior observed in colonies of real ants, and it has been applied to a wide variety of combinatorial problems [104, 105], including vehicle routing [106], feature selection [107] and protein function prediction [108]. The method generally uses simulated “ants” (i.e., mobile agents), which first construct tours or paths on a network structure (corresponding to solutions to a problem), and then deposit “pheromone” (i.e., signaling chemicals) according to the quality of the solution generated. The algorithm takes advantage of emergent properties of the multi-agent system, in that positive feedback (facilitated by pheromone deposition) quickly drives the population to high-quality solutions.

ACO algorithms have been extensively applied to the protein folding although most of them are based on the coarse-grain HP model. For instance, Shmygelska and Hoos [109] applied ACO to optimize the protein folding based on the HP model in both 2 and 3 dimensions. There are also other ACO-based implementations that have been applied to this problem in the literature. Song et al [110] provides a rapid transfer pheromone matrix method, a scheme to avoid deadlock folding problems, adynamic method of pheromone updating and also three different local search methods. This work uses the tortilla 3D benchmark [111] for the experimental evaluation.

Thalheim et al [112] combine the ACO with a branch and bound algorithm to enhance the protein folding simulation. For the experimental evaluation, they use proteins that are based on the bibliography and some of them come from PDB. Hu et al. [113] develop four different mechanismsto improve ACO algorithm, concretelyincludinga path retrieval method, the path construction, some folding heuristics and the pheromone attraction. These new mechanisms provide interesting results for solving protein folding problems with the HP square lattice model. Other hybrid approaches can be found at Chen et al. [114], where an ACO with genetic ideas was developed.

Some parallelization strategies have been applied to ACO solving the protein folding. [115] uses MPI to implement the parallel version of ACO.And in [116,117] OpenMPis used. It is noteworthy to highlight that only these few versions of parallelism have been implemented to solve the protein folding problem with ACO. From the High Performance Computing point of view, these parallel implementations use hardware clusters to evaluate their results.In [115]an IBM Blade center composed of 9 nodes, each node comprised of 2 2.4 Ghz Intel processors with 1 Gbyte of shared RAM is used. In [116]authors use a single PC to evaluate the sequential algorithm results, and an IBM pServer with eight 1.6GHz Power(gr) CPUs and 6GB RAM to run the parallel ones, which it seems not too fair. In [117], authors run the CASP8 benchmark on a multicore PC, specifically an IBM p550 server with an 8-core 64-bit 1.6-GHz PowerPC CPU, and the CASP9 benchmark is run on a cluster with 20 nodes of 16-core 1.6-GHz AMD CPU per node.

Although it has been demonstrated that this algorithm can take advantage of the GPU massively parallelism [118], to the best of our knowledge we could not find any work in this direction for the protein structure prediction using coarse-grain models.

ArtificialBeeColony

Artificial bee colony (ABC) algorithm is an optimization algorithm based on the behavior of honeybee swarms [119]. It provides a population-based search procedure in which the communication between bees is emulated to discover the best places with high nectar amount. Contrary to ACO, where only the HP model was targeted, ABC has been applied to different protein models such as HP, HP-SC, AB or ECEPP/3. There are several implementations of ABC applied to the protein folding problem. Zhang and Wu [120] use the HP-2D model to simulate the protein folding.However, authors use four Fibonacci sequences simulating proteins to test the algorithm instead of using a well-known benchmark like PDB or CASP.Another example of this algorithm can be found in [121], where synthetic sequences are created using Fibonacci sequences. Authors obtain experimental results with some PDB structures, though.

There are also parallel implementations of ABC that could be found in the literature. For example, in Benítez et al. [122-124], a complete study of different algorithm implementations can be found. Firstly, authors start implementing two parallel approximations of ABC algorithm in [122]: a master-slave implementation and a hybrid-hierarchical one, both of them implemented using ANSI C with MPI.They continue with the same two parallel approximations with genetic algorithm in [123],and finally authors conclude with the same parallel implementations of a hybrid algorithm merging an ABC with a Genetic algorithm (ABC-GA algorithm) in [124]. These authors remark that in future work they will consider the use of alternative computing technologies, such as reconfigurable computing and General-Purpose Graphics Processing Units, to accelerate processing. Nonetheless, no further papers in this sense have been found, at least applied to the protein folding problem with these algorithms. Finally, Bahamish et al. [125] develop a modified ABC that optimizes the Marriage in Honey Bee Optimization algorithm.

All the experimental environments in [120], [121]and [125]are based on single or multicores PCs.Benítez et al. [12-124] run their implementations on a 124 processing cores cluster.

Other papers considered in this area are Wang et al [126], where the Chaotic Artificial Bee Colony (CABC) algorithm, which combines the ABC algorithm with the chaotic search algorithm, is applied to 3D protein structure prediction; Li et al [127], where a balance-evolution artificial bee colony (BE-ABC) is presented and an AB off-lattice model is adopted, testedby Fibonacci sequences and proteins from the PDB as well; and [128], whereanother version of ABC is presented. These papers do not include any kind of HPC environment, and all the experiments run on a single PC.

Particle swarm optimization.

The third kind of algorithm that is shown in this section is Particle Swarm Optimization (PSO).PSO is a stochastic population-based optimization technique that is based on the social behavior of fish schooling or bird flocking. Applied to the protein folding problem, in [129] the authors implement PSO with an algorithm to avoid local minimums named levy flight. Like other algorithms, a parallel approach is performed by authors in [130] implemented using MPI, which is the most common way to parallelize the algorithms reviewed in this field. None of these papers, neither Chen et al. [129] nor Hernández et al.[130,131],give details about the environment for running the experiments on. Solely in [130] authors say that experiments are implemented in a "*dual-core PC and a Cluster*".

Other PSO algorithmscan be found in Liu et al.[132] and Mansour et al.[133]. The latterhave also developed a genetic algorithm for protein structure prediction. Both papers adopt the HP model with no HPC environments.

Genetic Algorithms

Genetic algorithms have been very used to address a broad range of combinatorial optimization problems that are NP-complete [134,135]. Genetic algorithms start from an initial randomly generated population of individuals. Over this initial population different selection, recombination and mutation operators are applied in order to evolve toward better solutions. In each iteration (generation), a function evaluates each individual, namely fitness function. On the one hand, the selection operator removes those individuals with worse fitness from a probabilistic point of view. On the other hand, the recombination and mutation operators generate variations of the individuals in order to produce new individuals. [136].

One of the first proposals of evolutionary algorithms to the PSP problem was presented by Unger and Moult [137]. In this work, a genetic algorithm is applied as an extension of a traditional Monte Carlo method to include information exchange between a set of parallel simulations. This method proves to find better solutions in the bidimensional HP lattice model than the traditional Monte Carlo methods. Some years later, an improved version of the basic GA [138] was presented using a new crossover operator and a new search strategy to avoid the homogenization of the population. Since then, several works following this idea has been proposed using different operators and strategies [139-145].

Genetic algorithms constitute a good alternative in several optimization problems. Nevertheless, one of the disadvantages of the genetic algorithm in optimization problems is the slow convergence. Concretely, in problems like PSP, they can suffer from excessively slow convergence rate due to the high number of needed calculations.

In order to avoid such problem, there is an opportunity in the hybridization of evolutionary algorithms with other heuristics, machine learning techniques, etc. The hybrid genetic algorithms can improve the performance of the basic algorithm and the quality of the solutions. For instance, the algorithms proposed in [146-148] combine a GA with tabu search algorithm, showing better results for the PSP than a genetic algorithm alone. Other works have proposed GA combined with other techniques, like backtracking [149], hill-climbing [150] and simulated annealing [151] or Particle Swarm Optimization [130].

As a result, since the PSP problem presents a large and complex search space, algorithms that combine local search methods with GA show significant improvements. In this sense, the combination of GA and local search using domain-specific knowledge, i.e. memetic algorithms [152] can help to find better solutions. Memetic algorithms (MA) use the concept of meme. A meme can be defined as a unit of cultural evolution which is able to local refinements. Some works have explored this mechanism for the PSP, resulting in that MAs are robust for finding structures across a range of models and difficulty [153-159].

The described proposals define the PSP as a single-objective optimization problem. This approach gets good results when one of the objective should be optimized or when all the objectives are not in conflict among them. Nevertheless, if several objectives should be optimized, a better approach is to consider the objectives separately, i.e., as a multiobjective optimization problem (MOP). A common problem in MOPs is the fact that usually there is no solution able to optimize all objectives at the same time. Therefore, the idea of optimum should be redefined and it is searched a solution that satisfies all the objectives in an acceptable manner. Some of the best well-known multiobjective evolutionary algorithms (MOEAs) are PAES-II, NSGA-II and MOEA/D. [160].

In this sense, some works propose the formulation of the PSP problem as an MOP to be solved by an MOEA. For example, [160] considers the PSP problem as the problem of minimizing free Potential Energy (PE) and minimizing Solvent Accessible Surface area (SAS). Authors solve this MOP using a modified version of the popular NSGA-II. In a similar way, the work of Day et al. [161] proposes a multiobjectivization for the HP model which scores better results in most of the cases than using a single-objective. Another example of this approach is the work of Brasil et al. [162,163]. In this work a new MOEA based on tables, called MEAMT, is presented. MEAMT is able to use four objectives based on tables to solve the PSP problem. In MEAMT, each table stores a subset of solutions with the best found solutions according to one of the objectives. More recently, several works have been proposed following this line of research. Some examples can be found in [164-167].

A great deal of the GA's popularity lies in its parallel nature and the inherent efficiency of parallel processing. MOEAs are a clear example of this parallelization, since their different objectives can be processed in parallel in an easy way. Despite the parallelization of MOEAs has been studied in several real-world problems, less work has been done in the parallel multiobjective approaches to PSP.

One of the works in this field has been developed by Calvo et al. [168-171]. They propose different parallel MOEAs approaches to the PSP problem reducing the complexity of the problem by the minimization of the set of variables involved in the process. Authors use 14 processors to execute parallel algorithms. They show that, although the quality of the solutions is not significantly improved, the process requires less time and presents a better parallel efficiency.

Tantar et al. [172] also propose a solution for the PSP using multiobjective parallel hybrid GAs (Hill Climbing local search [173] and simulated annealing [174] combined with GA) using computational grid. They use the ParadisEO-CMW framework, which combine the PAradisEO framework and the Condor-MW middleware. ParadisEO [175] is an open source framework dedicated to distributed and parallel models and the design of a broad range of metaheuristics. The Condor3 system [176] provides mechanisms that support High Throughput Computing (HTC). The underlying support the experiments was GRID5000 (2500 processors, 2.5TB of cumulated memory and 100 TB of non-volatile storage capacity). The tests were addressed using the tryptophan-cage (Protein Data Bank ID 1L2Y) and α-cyclodextrin proteins. Their studies show that, although the multiobjective GA increases the complexity, it provides more accurate solutions.

A different approximation for the PSP is proposed by Benítez et al. [177]. They present a parallel GA using the 3DHP-Side Chain model. In their approach the parallelism is reached by the division of the load into several processors (slaves) that are coordinated by a master processor. While the slaves have to compute the individual's fitness function, the master is in charge of the initialization the population and performing the rest of the GA operators. Since there is not dataset for the used model, the proposal was tested with a benchmark of synthetic sequences. Authors show that, although the results obtained are not the optimal, they are the best results found for the 3DHP-SC model. Finally, authors show that parallel processing accelerates significantly the process, but they propose other hardware-based approaches in order to get a better performing.

Unfortunately, this technique can suffer from a bottleneck in the master processor. In order to avoid this problem, in [178] it is proposed a mesh NoC-based multicore architecture in which the single-master multi-slave design is partitioned in small islands where an island has slaves and a master processor. In order to avoid GA falling in local minimal within each island, authors define a GA which is able to migrate between the islands. The experiments are performed using 9 proteins from a benchmark of synthetic sequences for the lattice protein model. Results show an overall 310X speedup gain compared to the design of the single-master /slave.

Others works have proposed modified GAs in order to parallelize the problem. For example, Narayanan et al. [58] propose a simple GA in which the mutation and selection strategies are parallelized using the MapReduce [179] architecture. Authors pursue to obtain the optimal conformation of a protein using the two dimensional square HP model. The proposal is validated against benchmarks of synthetic sequences, showing that the convergence of the algorithm to the optimal is faster than the obtained with traditional techniques.

Another modified version of an evolutive algorithm inspired by the biological immune systems, namely the clonal selection algorithm (CSA), is presented in [180] for PSP on AB Off-Lattice model. Experiments are performed using sequences of Fibonacci for simulating the AB model. The interesting aspect of this work is that the algorithm is parallelized using the CUDA platform and GPUs. In fact, authors show that the speed can be improved effectively, but they do not measure the quality of obtained solutions. There are also other hybrid GAs with bioinspired algorithms like Scalabrin et al. [181], but no more discussion is necessary because this paper has been also considered in the bioinspired algorithms section.

To summarize, although more works should be done in this direction, in the last years the parallelization of MOEAs is getting more attention and several works are including it as their future works [182,183].

Other nature-inspired algorithms

Other bioinspired algorithms also worth mentioning are gathered in this section.Firstly, a **Firefly Algorithm** (FA) [184] has been tested in the protein folding problem. Firefly Algorithm is a new algorithm that is based on the flashing behaviors of firefly swarms. The main purpose of the flash of fireflies is to attract other fireflies. The FA's assumptions consist in three basic rules: (1) sex of fireflies does not mind at all as all fireflies are unisex. Each firefly flashes in order to attract other fireflies regardless their sex; (2) the intensity of the flash is mainly due to attract a prey and to share food; (3) the more a firefly shines, the more attractive it is to others. Therefore, each firefly firstly moves toward a neighbor whom glow is brighter. In this paper, two dimension HP lattice model is tested in a single PC, a P4 IBM with 3.1 GHz processor and 2 GB of RAM.

Only one approach to GP-GPU implementation has been found for bioinspired algorithms. Scalabrin et al. [181] (same authors of [122-124])have implemented a new algorithm named **Population-Based Harmony Search**, (PBHS). The Harmony Search is inspired by the improvisation process of a musician searching for the best harmony. The solution is represented by a harmony and the method of improvisation guides the balance between deep search and wide exploration. The results of this paper show that the implementation in CPU could be better when few data are used, but the GP-GPU is clearly better when data grow. The hardware experimental environment in this paper is an Intel processor (Core2-Quad at 2.8 GHz) and a NVIDIA GeForce GTX280.

Another bioinspired algorithm is the one developed by Cai et al. [185], where authors proposea new algorithm inspired by the plant growth process called **Artificial Plant Optimization Algorithm** (APOA). Photosynthesis operator, phototropism operator and apical dominance operator are designed in this paper.Another version of this algorithm can be found in [186], where authors implement the gravitropism mechanism that is neglected in the standard version. In this paper, authors employ this phenomenon to enhance the performance. To test the efficiency, they apply this new variant to solve protein structure prediction problem, including short sequences, Fibonacci sequences and real protein sequences, showing effective simulation results. The authors of these papers also present another bioinspired algorithm in [187] called **Social Emotional Optimization Algorithm** (SEOA). It is a new swarm intelligent methodology by simulating the human social behaviors. In this algorithm, each individual represents one virtual person in the searching space, all of them trying to promote to a high society position by collaboration and competition. In this paper, it is applied to predict the structure of toy model proteins. To test the performance, short sequence, Fibonacci sequence and real protein sequences are selected to compare. Simulation results show that this approach is valid. Authors do not use HPC environments in any of these papers commented in this paragraph.

Several hybrid approximations have been implemented, as for example in Benitez et al. [122], discussed above. Other papers with this point of view areNemati et al. [108], showing an implementation that combines a hybrid genetic and ACO algorithm; and also in Lin and Su [188], where authors implement a hybrid genetic and PSO algorithm. Moreover, although several modifications in algorithms have been tested, no improvements in hardware environments are found, since in [108] authors run the algorithm in a 3.0 GHz CPU and 512 MB of RAM, and no specification was found about hardware in [188].

To summarize, these papers give us the idea that several implementations of different algorithms have been tested during last years. Perhaps the more common algorithms at this point are ACO and ABC, although some other algorithms with different implementations have been found, for example hybrids algorithms. On the other hand, too little parallel implementations have been developed for these algorithms, and the exploitation of High Performance Architectures is reduced to the executions of parallel implementations based on MPI and OPENMP. Other types of more intensive data parallelism, like GP-GPU implementations, are expected to be widely developed, but unfortunately, the implementation in[181] byScalabrin et al. has been the only one found in this direction.

It is worth mentioning other reviews on this area, such as [189,190], that show the same point of view of different algorithms applied to the protein folding problem, although none of them elaborate a review from the High Performance Computing view.

3.2.2. Non-nature-inspired metaheuristics

Non-nature-inspired algorithms are mainly based on local search methods. They are a family of metaheuristic algorithms aimed at solving NP-hard optimization problems. Applied to protein folding, they try to obtain the minimum energy structure in polynomial time from a set of candidate solutions sampled from the search space. The main idea is to start from a folded protein deemed as a potential solution and then modify it (i.e., move to a neighbor solution in the search space) trying to obtain a slight improvement in the energy structure. Local search methods possess the main advantage of rapid convergence to better quality solutions, if not optimal, when efficient neighborhood functions are employed. However, an optimal solution cannot be guaranteed since the candidate solutions are randomly selected and the optimal one could not be included nor reached from the selected ones. Another drawback to take into account is that these methods get locked in a local optimum very often and may revisit the same set of solutions repeatedly.

Among the local search methods for protein folding simulation, Tabu search[191] is the most frequently found in the literature. The basic feature of this method is the use of memory structures to save solutions already explored. Then, if a potential solution is explored again in a specific period of time, it is considered tabu (i.e., forbidden) and therefore it is not expanded in order to promote the exploration of new regions in the search space. Tabu search algorithms applied to protein folding are also based on this feature, and they differ in the moves definition and how to avoid local optima.

Apart from Tabu search, hill climbing[192] and simulated annealing (SA)[193,194] are other two local search algorithms applied to protein folding. Hill climbing consists in starting with a random solution and changing a single element of the protein structure iteratively and incrementally while each change produces a better solution, until no further improvements can be made. On the other hand, simulated annealing uses a probabilistic heuristic to change from one random solution to another random solution with the aim of moving to a state of lower energy, but it still possible to change to a worse solution, i.e., a state of higher energy (and in this manner avoid local optima). The probability to move from a state *s* to a state'*s* depends on the energy of each state and on a global dynamic variable called temperature (T), which is initiated to a high value. As usual, if s' is considered better than s, then the movement is performed. However, if s' is considered worse than s, it is still possible to make that movement depending on T. For higher values of T, the probability of making this "worse" movement is higher. As T decreases through iterations, this probability also decreases, simulating the annealing process in metal. In this manner, it is possible during the initial phase of the process to move towards less promising solutions so as to avoid local optima, but at the end of the process --when T has values next to 0-- the probability of selecting worse solutions is almost inexistent. It is worth mentioning that both algorithms are normally used in combination with genetic or stochastic algorithms, as an alternative to improve the efficiency in the latter.

In the next paragraphs we review some of the most relevant works on protein folding for each local search algorithm.

Tabu Search.

[195] describes a generic tabu search plus a set of new moves for named "pull moves", that modifies the basic Tabu search by moving one aminoacid a small distance and then pull the chain along, stopping as soon as possible. These moves are complete (all existing configurations can be reached from the initial one), reversible and local (displace as few vertices as possible). As a result, authors propose small adjustments to a given configuration in order to improve the effectiveness of Tabu search in protein folding for HP-2D models. [196] also addresses HP-2D models. Moves are defined as changes of single angles of consecutive positions in the vector representing the protein, whereas the tabu list consists of forbidden angle moves to avoid reverse moves in a specific number of iterations. Authors claim to find optimal conformations for all short sequences from 5 to 12 aminoacids.

[197] explores on HP energy models on 3D FCC lattices. The Tabu method is composed of a function to initialize the model in a randomized, structured manner; a fitness function to guide the search; and efficient data structures to avoid cycles.. Authors obtain the first foldings in the well-known "Harvard instances"[198], 10 different proteins on the cubic lattice. This work has been revisited in [199], where the tabu algorithm is combined with constraint programming. Results show to be promising and reliable for proteins consisting in less than 100 aminoacids. Eventually, all the previous results on HP energy models on 3D FCC lattices have been outperformed by the work in[200]. This paper defines a hydrophobic-core centric local search algorithm named SS-Tabu. Movements are defined as a coil spinning around a dynamic hydrophobic-core center (HCC) by means of a diagonal move to build the cores. In order to avoid local minima, two different techniques named random-walk (based on the pull moves defined in[195] and relay-restart are defined. Another appealing approach on 3D HP lattices is proposed in[201] where authors develop an hybrid search algorithm that combines an enhanced particle swarm algorithm with an enhanced tabu search algorithm. The former appends the operation of crossover (single-point and two-point crossover) whereas the latter adds the operation of mutation. The main idea resides in using the tabu search algorithm to "help" the swarm algorithm to avoid local minimum. This hybrid algorithm has been implemented by MATLAB R2009b under a Windows XP system and tested through Fibonacci sequences and some PDB real proteins. Results show that it is superior to other 3D HP algorithms up

to sequences no longer than 48 aminoacids. A different approach for obtaining minimum energy in oligopeptides is presented in[202]. Moves are based on the dihedral angles in the protein's skeleton and the cost function is the empirical energy function ECEPP/3. It is aimed at working in angle space while keeping bond length and bond angle values constant. The algorithm is parallelized by executing several moves simultaneously. Hence, it is created a partitioning of the set of possible movements on p subsets of approximately the same size, and every partition is evaluated in p different processors. In this manner every processor finds its best move, and the best between these is eventually selected. The main drawback in this approach is the extensive communication requirement among processors. It has been tested using the Met-enkephalinpentapetide, showing a real speed-up compared to related techniques due to the parallelization process. As a result, Tabu search is considered valid for conformational searches of peptides when an optimal combination of tabu parameter values can be found.

Xiaolong et al. proposes a tabu algorithm whose main feature is the generation of the initial solution for 3D AB off-lattice models [203]. Instead of using a random function, a better-informed method is defined by locating hydrophobic residues at the center of three-dimension space and locating hydrophilic residues surrounding hydrophobic ones. In[204] a similar heuristic for the initial solution is employed and a new one is defined for conformation updating in 2D AB off-lattice models. The conformation updating heuristic consists in picking out hydrophilic monomers squeezed among hydrophobic monomers and placing them in certain spots in 2D space to speed up the search for lower-energy states.

Hill Climbing.

Regarding hill climbing works in the protein folding area, we have found that this technique is usually combined with genetic algorithms to improve the results of the latter. Thus, in [205] a hybrid of hill-climbing and ERS-GA (genetic algorithm with elite-based reproduction strategy), named HHGA, is proposed for protein structure prediction on the HP-2D triangular lattice. Two hill climbing strategies are proposed: In the first one, the algorithm selects its neighbour residues from the current solution. These residues are generated as in mutation operations, i.e., randomly changing its direction. In the second one, the neighbour residues are generated following a method similar to the crossover operation. Hence, five neighbours are generated by changing the direction of the second segment after the crossover point, where rotation angles are 60°, 120°, 180°, 240° and 300°, respectively. If any of the five folding directions leads to a superior fitness to the original direction, this neighbour will replace the current solution. A benchmark composed of eight HP-2D protein sequences up to 64 aminoacids is evaluated and compared to simple genetic algorithms [206] and tabu search [207], demonstrating that HHGA produces a similar outcome to the those algorithms, but at the cost of incrementing the running time. Another work adopting hill climbing along with a genetic algorithm can be found in [208], which relies on hill-climbing recombination and mutation to support the search process of the evolutive algorithm for HP proteins. Here, the crossover operation is dynamically performed, allowing offspring to be added in the population during the same generation in an asynchronous manner. In this model, the proposed mutation operator is problem-specific and it is applied in a steepest-ascent hill-climbing manner. Moreover, to avoid local optima, redundant individuals may be replaced with new genetic material thanks to an explicit diversification stage which is carried out periodically during the population evolution. Standard S1-S8 HP proteins are employed as a benchmark and they are evaluated by the hybrid model presented in the paper and compared to other three simpler models, namely a simple evolutionary algorithm, an evolutionary algorithm with diversification stage and an evolutionary algorithm with hill climbing but without diversification. Results show that using hill climbing to support evolutive algorithm is clearly beneficial with respect to other models neglecting its use and it could compete with other algorithms such as memetics. Another hybrid GA-hill climbing algorithm, this time to fold proteins from knowledge of the primary sequence and predictions of its secondary structure, can be found in[209]. Dihedral angles are used to represent the protein's structure augmented with a four-helix bundle to improve the folding simulation conditions. According to the obtained results, the inclusion of a hill climbing algorithm to execute local searches in the GA outperforms 20% and 50% the execution of the pure, original GA in [210]. In conclusion, it can be stated that hill climbing algorithms are not practical by their own in protein folding, but they are rather combined with genetic algorithms to improve the latter.

Simulated annealing.

Like hill climbing, SA is mainly adopted for improving other global search algorithms. For example, [211] introduces a protein folding simulation procedure on FCC lattice that employs a constraint satisfaction problem (CSP) solver to generate neighbourhood states for a simulated annealing-based local search method. This proposal has been evaluated using three basic proteins for tuning (namely, 4RXN, 1ENH, 4PTI) and then several proteins selected from PDB, with length varying from 54 to 74 aminoacids. Results show that the hybrid approach outperforms CSP alone both in accuracy and efficiency, and outperforms local search alone in accuracy but not in time.

Another approach consisting in a combination of Bayesian and SA functions is described in [212]. It uses Bayesian scoring functions to assemble native-like structures from fragments of unrelated protein structure with similar local sequences. The simulated annealing contributes to generate native-like structures for small helical proteins in a rapid manner. Finally, it is worth mentioning the approach in[213] based on a pure SA algorithm in 3D HP protein folding simulations aimed at experimentally determining upper bounds for the maximum depth of local minima of the underlying energy and for the stopping criterion. Tests on the well-known ten benchmarks

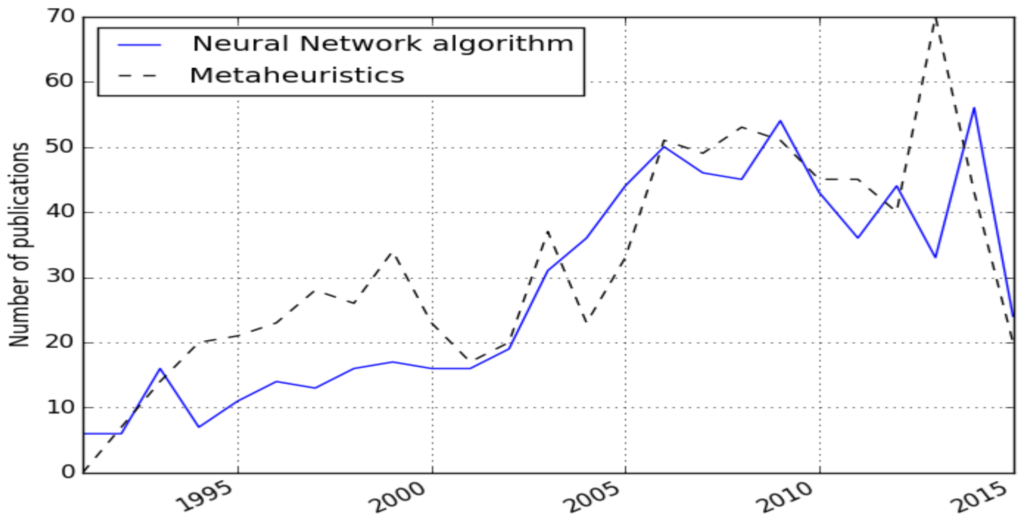


Figure 3 - Number of publications in protein folding, protein structure prediction or HP model

given by [214] show that the maximum escape height from local minima can be upper bounded by n^(⅔) whereas the stopping criterion complies with the number of Markov chain transitions that lead to minimum conformations. Further tests must be carried out on real foldings of short protein sequences to validate these results, which could serve as appropriate starting conformations for folding simulations of real protein sequences and realistic energy functions.

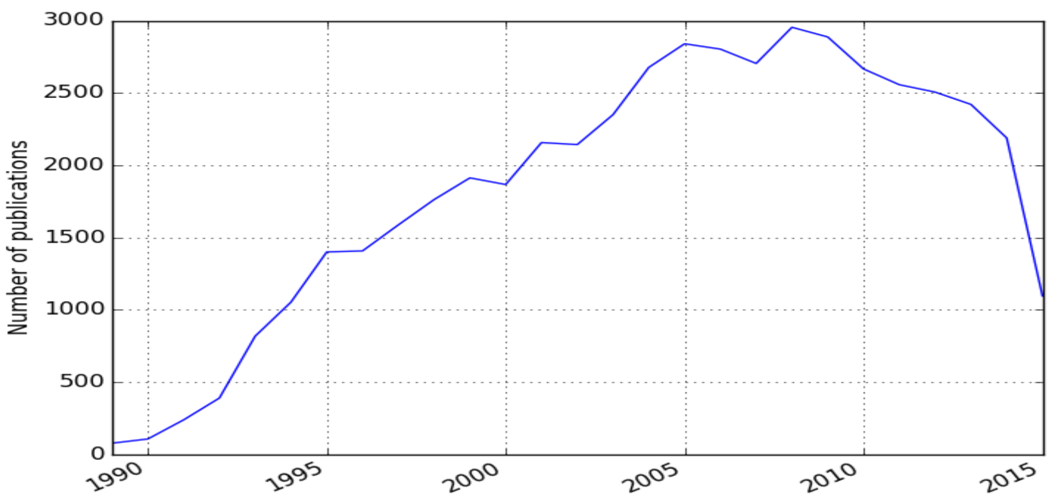


Figure 4 - Number of publications for Neural Networks and Metaheuristics techniques applied to protein folding

4. TRENDS IN DESIGNING NOVEL ALGORITHMS AND ARCHITECTURES

This section provides quantitative information about the main contributions in the field of Metaheuristics applied to PSP, mainly based on coarse-grain models. Moreover, we show what kind of hardware architectures have been used to run these novel algorithms on. Our deep search literature review follows a methodology that is firstly described to let the reader reproduce the experiments.

4.1. Experimental methodology

For this experimental study, we have used the Web of Knowledge (WOK, formerly known as ISI Web of Knowledge) [215]. WOK belongs to Thomson Reuters Corporation and it is an academic citation indexing and search service to provide bibliographic content and tools to access, analyze and manage multiple research information.

A particular interest to us is the WOK advanced search tool. This tool offers a very powerful search tool to look for different research articles using formal rules based on field tags, Boolean operators, parentheses, and query sets to create your own query. Booleans operators include AND, OR, NOR, SAME and NEAR. The following field tags are the most interesting for our searches purposes:

- TS = Topic. Searches the Topic fields in all databases in your institution subscription. Topic fields include Titles, Abstracts, Keywords and Indexing fields such as Systematics, Taxonomic Terms and Descriptors.
- SU = Research Area. Searches the Research Areas field within a Full Record.
- GP=Group Author. Searches the Group Author(s) and Book Group Author(s) fields within of a record.
- AU=Author. Searches for author names of journal articles and books in the Author(s) field and the Corporate Author(s) field.

The most interesting filed tag for our data mining purpose is TS as we are looking for articles related to protein folding, different Metaheurtistics techniques and particular hardware implementations. For instance, the following pattern searches for articles in which either “Protein folding” or “Protein Structure Prediction” or “HP model” are included in the article’s Title, Abstract or Keywords.

$$\text{TS} = (\text{"protein folding"} \\ \text{OR "protein structure prediction"} \\ \text{OR "HP model"})$$

However, the information obtained from this tool may have some inaccuracies as we are dealing with unstructured data. For instance, the terms “Neural Networks” and “Protein Folding” may be included in chemistry research articles about the brain, which clearly is not our scope. Therefore, after searching for some keywords we dida carefully review on ambiguous papers and checked whether they were related to the topics we are really looking for. Moreover, the WOK does not have very up-to-date information. Some recent papers are not included in their databases, and therefore, the quantitative information of the last couple of years may be incomplete. This issue mayaffect our conclusions regarding to the hardware trends as hardware platforms have evolved very rapidly in the last five years. As a result, we have also included articles from other databases such as Google Scholar, arXiv, CiteSeer(X), DBLP and IEEEXplore, to name just a few.

4.2. Trends in Soft Computing for the protein folding

In the first place, Figure 3shows the number of publications within the field of protein folding, protein structure prediction or coarse-grain HP model available in the WOK. The rule to perform this search is the following:

$$TS = (\text{"}proteinfolding\text{"} \\ OR\ \text{"}proteinstructureprediction\text{"} \\ OR\ \text{"}HPmodel\text{"})$$

From Figure 3 we can state that the Protein Structure Prediction is a very active field of research that began in eighties and it is still an object of continuous research with approximately 2.500 published papers per year.

Next, Figure 4shows the number of publications related to the protein folding that use *Soft Computing* techniques. Here we have grouped *Soft Computing* techniques into two different categories: Neural Networks and Metaheuristics. According to this figure, Neural Networks have been the most active research topic from the nineties. However, Metaheuristics has recently attracted interest in the protein folding community. In the last few years the number of articles published in Metaheuristics is at the top of the *Soft Computing* techniques applied to the protein folding. Local Search techniques, however, are almost always combined with other global techniques such as genetic algorithms, swarm intelligence or ant colony optimization to provide hybrid *Soft Computing* techniques. They improve the optimization process of those global search techniques to avoid stalling in local optimum, as noted in Section 3.2.2. The following rule is only an example of how we have obtained the number of publications for Neural Networks:

TS = ("protein folding" OR "protein structure prediction" OR "HP model")

$$and\ TS = (\text{"}Neural\ Network * \text{"}\ or\ \text{"}deep\ learning * \text{"}$$

$$or\ \text{"}support\ vector\ machine * \text{"}\ or\ \text{"}random\ forest * \text{"}$$

or "extreme learning machine" or "multilayer perceptron*")*

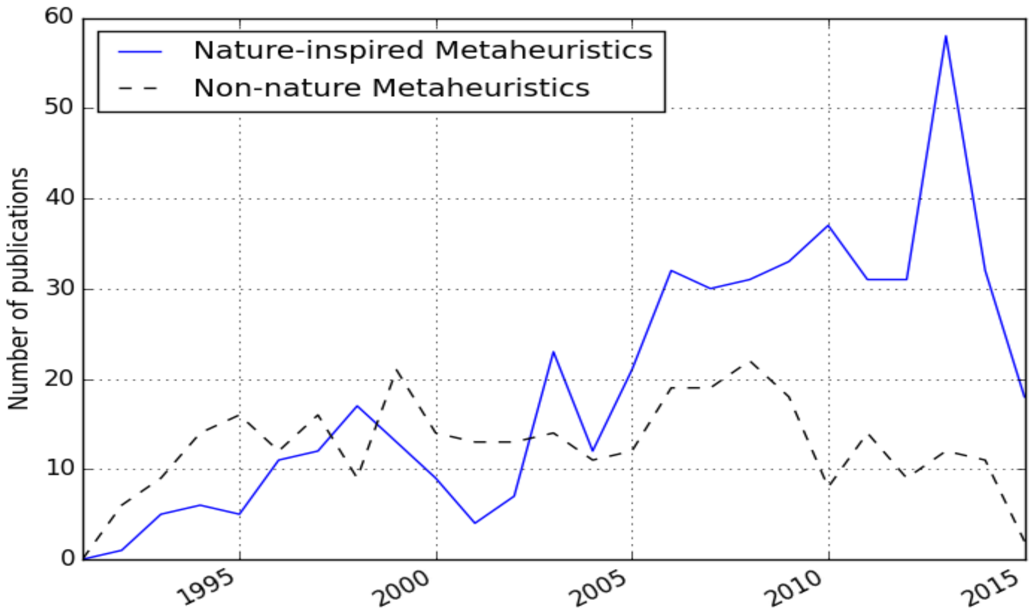


Figure 5 Number of publications that use Metaheuristics according to their origin.

Figure 5shows the number of publications in WOK related to both:the protein folding and different kind of Metaheuristics that area classified depending on their origins.The keywords used to do this search include for the nature-inspired metaheuristics:Genetic algorithm, Ant Colony Optimization, Artificial Bee Colony, Particle Swarm Optimization, Firefly Algorithm, Population-Based Harmony Search, memetic algorithm, Artificial Plant Optimization Algorithm and Social Emotional Optimization Algorithm. For non-nature-inspired metaheuristics the keywords are hill climbing, simulated annealing and tabu search. Some issues come up with this search as these keywords may belong to the same algorithmic family. For instance, ACO and ABC are population based methods which is also another keyword in Figure 5. Therefore, the number of publications depends on what keywords have been included in the article. Finally, those Metaheuristics that we could not find any work related to protein folding have not been included in Figure 5.

Figure 5places Genetic Algorithms are widely used in this area as they are one of the pioneer in Metaheuristic research. Particle Swarm and Ant Colony Optimization techniques are at the second place of the techniques used for protein structure prediction. Some variations of these Metaheuristics like memetics, firefly or Artifical Bee Colony are also applied in the literature but their use is marginal.

Figure 5shows the number of publication for different kind of non-nature metaheuristics that are mainly local search techniques. As previously described, local search techniques are used along with other global techniques to provide hybrid search method that improve simulation's quality and performance. The methods used in the protein folding arena are Tabu search, simulated annealing and hill climbing. The latter is widely used to improve the search provided by Metaheuristics. Although Tabu search is very close to hill climbing, the computational cost of tabu is higher than hill climbing, and thus it is not so convenient to integrate it in a hybrid method. Simulated Annealing is, however, a very powerful local search and it is actually the most studied in the literature.

4.3. Trends in hardware architectures for Soft Computing techniques applied to the protein folding

A common computational feature shared by many *Soft Computing* methods is their inherent massive parallelism. Most of them are population-based, that is, a collection of agents "collaborate" to find an optimal (or at least a satisfactory) solution. Because of this inherently parallel nature, these methods are well-suited to leverage parallel, distributed or even GPU architectures. Table 1summarizes the hardware platforms that have been used to improve the execution of different *Soft Computing* techniques.Neural Networks are basically executed on single core processors. Although there are some efforts in parallelizing neural networks applied to other problems, to the best of our knowledge there is only a work that cares about performancein this kind of algorithms applied to coarse-grain protein folding. Moreover, this algorithm is based on deep learning, which has many layers and thus the computational requirements increase drastically. Genetic algorithms are, however, very tied to parallel architectures. They are based on a population of entities where the island-model is very attractive to improve the solution.

In the parallel island-model of genetic programming, the population for a given run is divided into semi-isolated subpopulations. Each subpopulation is assigned to a separate processor or node of computing system and it proceeds independently to each other. Once each instance of the genetic algorithm finishes (or other interval), a relatively small percentage of the individuals in each subpopulation are probabilistically selected (based on fitness) for migration from each processor to various neighboring processors. This idea has been implemented on different platforms from clusters of computer nodes to grid computing environments. There are also other different parallel algorithms based on data approach that are better suited to GPUs. ACO, ABC and PSO also use the island model to leverage cluster computing architectures. Population Based Harmony Search has been implemented on GPUs as well. Finally, local search techniques have been also improve with some ways of parallelism in different architectures. Nonetheless, as previously mentioned, these methods are always combined to other methods, and therefore, they are also involved in other rows of the Table 1.

4.4. Summary

This section briefly summarizes the strengths and weaknesses of the reviewed algorithms grouped into main categories we have used throughout the paper. First of all, Artificial Neural Networks (ANNs) have been successfully applied to the protein's secondary structure prediction. The ANN computational cost of learning, applied to this problem, is affordable for sequential architectures, and thus it does not require the use of high performance computing.

Moreover, the ANNs offer an abstraction layer that provides solutions without having deep-knowledge of the problem domain that is very appreciated for non-domain experts within this area. However, we have only found few works using ANN that target more complex protein structure. This actually limits the successful of these techniques. Indeed, new trends in neural networks, such as deep learning, are demonstrating very good results in other domain fields [216]. They demand the use of high performance computing. The search for the ANN optimal architecture; i.e. the number of neurons within the hidden layer or even the number of layers, can be a very time consuming process.

This paper divides Metaheuristics for their origins into two main groups; nature and non-nature-inspired. Nature-inspired metaheuristics provide very good solutions in a reduced time-frame but they do not guarantee optimal solutions. Algorithms like ACO, ABC, PSO and so on, are based on swarm intelligence to solve problems. They are inherently parallel, and therefore, theoretically well-suited for parallelization on emergent architectures. This feature has been explored in few papers, but indeed, we still see many remaining work in this area.Moreover, genetic algorithms have the advantage that they could escape from suboptimal local maximum/minimum. They are population-based and they use stochastic operators that allow searching in different regions, thus if the population finds a better fitness value can move away from the suboptimal solutions. Genetic algorithms are also inherently parallel as population-based algorithms and therefore they are also well-suited for parallelization. Nevertheless, genetic algorithms also have some disadvantages whenever they target problems like protein folding. Sometimes genetic algorithms may converge very slowly, especially near an optimum. Some hybrid approximations have been presented for the protein folding problem in order to solve such problem. In that sense, genetic algorithms could suffer of the opposite problem and they can converge prematurely to the suboptimal solutions if the operators are not efficient enough. Finally, another disadvantage inherently associated to genetic algorithms is finding the algorithm parameters; it is not straightforward at all and very problem-dependent.

Non-nature-inspired metaheuristics, which in this paper are basically focused onlocal search techniques, provide appealing solutions for 2D/3D HP models. They can be easily combined with other global algorithms such as Genetic Algorithms or ACO to improve their solutions. They can quickly converge to better quality solutions, even optimal, when efficient neighborhood functions are employed and they could serve as appropriate starting conformations for folding simulations of real protein sequences and realistic energy functions.However, local search algorithms by themselves cannot guarantee an optimal solution. The candidate solutions are randomly selected and the optimal one could not be included nor reached from the selected ones. Also they may get locked in a local optimum very often and may revisit the same set of solutions repeatedly.

SoftComputing Technique	Algorithm	Hardware Platform	Data Set	Model	Ref
Neural Networks	Deep Learning	CUDA	D329, SVMCON_TEST and CASP9	HP 2D	[96,97]
	NNPIF (Neural Network Pairwise Interaction Fields)	Single core	PDB	HP 2D	[84]
	MLP (Multilayer Perceptron)	Single core	PDB, SCOP	HP 2D	[72,73]
	MLP + tailored early-stopping	Single core	PDB	HP 2D	[85]
	MLP + Evolutionary information	Single core	PDB	HP 2D	[71]
	SVM (Support Vector Machine)	Single core	SCOP	HP 2D	[86,91,94]
Genetic algorithms	Multiobjective GA	14 processors	1CRN protein	Atomic model based on the dihedrals angle base between the Cα	[168-171]
	Hybrid Multiobjective GA (Simulated Annealing and Hill Climbing)	ParadisEO-CMW framework. GRID5000	tryptophan-cage (Protein Data Bank ID 1L2Y) and α-cyclodextrin proteins	Atomic model based on the dihedrals angle base between the Cα	[173,174]
	Simple GA	MapReduce architecture (cluster)	Benchmarks of synthetic sequences	HP model	[179]
	Parallel GA (single-master multi-slave)	Master-slaves processors	Benchmarks of synthetic sequences	3DHP-Side Chain model	[177]
	Parallel GA (multi-master multi-slave)	Mesh NoC-based multicore architecture	Benchmarks of synthetic sequences	Lattice protein model	[178]
	Clonal selection algorithm (CSA)	GPUs and CUDA platform	Fibonacci based sequences	AB Off-Lattice model	[180]
ACO	Parallel ACO	Cluster	http://www.cs.sandia.gov/tech reports/compbio/tortilla-hp-benchmarks.html	HP 3D	[115]
	Parallel ACO	Single PC and Cluster	-	HP 2D	[116]
	Parallel ACO - packBackbone	CASP 8 Multicore PC. CASP 9 run on a Cluster.	CASP 8/9	HP 3D	[117]
ABC	Parallel ABC. MPI	Cluster Networked computers with 124 processing cores	Sequences from bibliography	HP 3D Side-Chain	[122]
	Modified ABC. IF-ABC	Multicore PC (Matlab)	Fibonacci based sequences. PDB sequences.	AB	[121]
	Modified ABC. MHBO	Multicore PC Visual C++	Met-enkphaline	Atomic model based on the dihedrals angle base between the	[125]

				Cα	
PSO	Parallel PSO	Multicore PC and a cluster	Sequences from bibliography	Atomic model based on the dihedrals angle base between the Cα	[130]
PBHS	Population Based Harmony Search	Multicore PC. NVIDIA GeForce GTX280.	Benchmarks of synthetic sequences.	AB 2D	[181]
Tabu Search	Pull moves similar to de Gennesreptation model.	HuGS middleware (Human-Guided Search)	Sequences from bibliography	HP-2D	[195]
	Protein's angles-based moves	Single core PC AMD Duron 700Mhz Linux	Sequences from bibliography	HP-2D	[196]
	Tabu search + Constraint programming	A cluster of Dell Power Edge 1950 4-core IntelE5430 processor with 2.66GHz and 16Gb RAM (no parallelism)	Sequences from bibliography	HP 3D FCC lattice	[199]
	Spiral Search Tabu	-	Sequences from bibliography CASP 8/9	HP 3D FCC lattice	[200]
	Particle Swarm Optimizer + Tabu Search	MATLAB R2009b under a Windows XP system.	Fibonacci based sequences PDB proteins: IBXL, IEDP, IAGT	HP 3D FCC lattice	[201]
	Empirical energy function ECEPP/3	SGI Origin 2000 computers parallelized (32 processors) Distributed memory MPI for interprocessor communication	Met-enkephalinpentapetide	Atomic model based on the dihedrals angle base between the Cα	[202]
	Well-informed initial solution	-	Fibonacci based sequences (13, 21, 34) PDB (1BXL, 1EDP, 1AGT)	3D AB off-lattice	[203]
	Heuristic for conformation updating	Intel Core2 Duo, 2.66 GHz processor and 2.0 GB of RAM	Fibonacci based sequences (13, 21, 34,55) PDB (1AGT, 1AHO)	2D AB off-lattice	[204]
Hill Climbing	Montecarlo + hill climbing	Linda Tuple Spaces (Agents) Multithread C Two Opteron dual core CPU at 2 GHz	Several proteins from PDB	1. coarse grained structures based on previous bibliography 2. Own model	[64]
	Genetic algorithm + hill climbing	Single core Intel i7-920	Sequences from bibliography	2D HP Triangular lattice	[205]

		machines			
	Genetic algorithm + hill climbing	-	S1-S8 standard HP proteins	2D HP	[208]
	Genetic algorithm + hill climbing	SGI Onyx2 12 × R10000 supercomputer	Folding of the alpha carbon atoms of 100 non-redundant test proteins	Dihedral angles to augmented with a four-helix bundle	[209]
Simulated annealing	Time-dependent cooling schedule	Gentoo Linux on a 2.4 GHz Intel Pentium IV processor	Sequences from bibliography	HP 3D	[213]

Table 1 Summary of the hardware platforms used to improve different Soft Computing techniques.

CONCLUSIONS AND FUTURE WORK

The protein folding problem is a very well-known topic that has been widely studied during the last fifty years. Indeed, this review article showsthat the protein structure prediction problemis still a very active field of research nowadays, where many novel techniques and algorithms have been applied by means of computer simulation, mainly due to their high computational requirements. Our review focuses on both computational aspects:

1.-From the *algorithmic point of view*, we center on novel algorithms within the *Soft Computing* fieldthat have been applied mainly to the coarse-grain protein-folding problem, and focusing mostly on the HP-model.A particular interest to us are Neural Networks and Metaheuristics, as they are increasing in popularity during the last decade.The combination of these methods with local search techniques produces very powerful search strategies that providesome remarkable and interesting solutions to this problem. In this sense, and to the best of our knowledge, we have not found any work that design a hyper-heuristic or parametrized metaheuristic schema for the problem of the prediction of protein structure. These techniques provides a high-level of abstraction to look for the best metaheuristic to be applied to a concrete problem. Basically, metaheuristics search solutions within the problem domain and hyper-heuristics do the search within the search space of heuristics. Future designs should not only consider a metaheuristic, they should design a hyper-heuristic to provide a wide search within the space solution though. Besides, new trends in neural networks, such as deep learning, are gaining popularity, and we envision them as a good alternative for the protein structure prediction problem. However, fruitful works in this area should be designed taking care of computational requirements they intrinsically have by its definition, and thus they should designed on massively parallel architectures.

2.-From the *hardware point of view*, there are also some relevant contributions in the literature. Most of *Soft Computing* techniques are inspired bynature and they are massively parallel by their definition.Therefore they are well suited for implementation on parallel or even massively parallel architectures. After a deep literature review, we concludethat the gap between hardware and software in the simulation of protein folding is still very wide. There are some works that combine novel hardware and software techniques but they representjust an incipient research line. We are witnessing a revolution in hardware platforms where massive and heterogeneous platforms are dominating the marketsuch as GPUs.There are many applications already working right on the scientific and engineering fields. Changing them to run with billion-way parallelism will require redesigning or even reinventing the algorithms used in them, and potentially reformulating the science problems.

The protein folding simulation is a multidisciplinary field of research where scientists from different areas work together in order to solve challenges of the next century. Although many success cases have been reported in this review, there are still many aspects on the scientific side that need improvement. Just to name a few, the focus of application of these techniques relies on the study of single systems such as isolated proteins, but an “out of the box” approach should be followed in order to exploit them in more complex systems such as the ones in study by systems biology, as the cell as a whole. Also, techniques reviewed in this paper for the PSP problem might be directly applied to other biological macromolecules such as disordered proteins, nucleic acids, polymers, and systems with relevant nanotechnological interest. However, solving the problem of the prediction of protein structure, it is not an easy task. The workflow in Bioinformatics to create efficient tools is a long pipeline where each stage may take several years. Once theoretical models have been defined by experts from fundamental research fields such as physics, biology and chemistry, computer scientists need to define algorithms to simulate such models in computers. Moreover, as we move to a sustainable world, there are also other important concerns to take into accountas performance and energy efficiency of such algorithms on particular hardware architectures. Understanding how to bridging the gaps between hardware and software will be the key to solve mission-critical science problems at exascale.

From our point of view, future developments in this area should be aware of this landscape of computation.First of all, the physical limitations of silicon-based architectures are threatening the evolution of processors. Heterogeneous computing including GPUs, multiprocessors, or low-power processors come to the rescue when no answer looms on the horizon.Particularly, GPUs are showing great benefits in terms of performance and power consumption. The ratios compared with CPUs are expected toincrease even more as long as the problem size keeps growing and GPU microarchitectures take the next step forward. Moreover, the novel interest of governments in green computing makes mandatory developsscientific power-aware applications that use all hardware resources at minimum power-budget.

ACKNOWLEDGEMENTS

This work is jointly supported by the FundaciónSéneca (Agencia Regional de Ciencia y Tecnología, Región de Murcia) under grants 15290/PI/2010 and 18946/JLI/13, by the Spanish MEC and European Commission FEDER under grant with reference TEC2012-37945-C02-02 and TIN2012-31345, by the Nils Coordinated Mobility under grant 012-ABEL-CM-2014A, in part financed by the European Regional Development Fund (ERDF). We also thank NVIDIA for hardware donation within UCAM GPU educational and research centers.Finally, we thank anonymous reviewers for their constructive and valuable comments.

REFERENCES

[1] Dill KA, MacCallum JL. The protein-folding problem, 50 years on. Science 2012; 338(6110): 1042-6.

[2] Dill KA, Bromberg S, Yue K, *et al.* Principles of protein folding a perspective from simple exact models. Protein Sci 1995; 4(4): 561-602.

[3] Shaw DE, Dror RO, Salmon JK, *et al.* In:Millisecond-scale molecular dynamics simulations on Anton. Proceedings of the Conference on: High Performance Computing Networking, Storage and Analysis. Portland, IEEE 2009; pp 1-11.

[4] Merlitz H, Wenzel W. Comparison of stochastic optimization methods for receptor–ligand docking. Chem Phys Lett 2002; 362(3): 271-7.

[5] Baker D. A surprising simplicity to protein folding. Nature. 2000; 405(6782): 39-42.

[6] Berger B, Leighton T. Protein folding in the hydrophobic-hydrophilic (HP) model is NP-complete. J Comput Biol 1998; 5(1): 27-40.

[7] Fraenkel AS. Complexity of protein folding. Bull Math Biol 1993; 55(6): 1199-210.

[8] U.S. Department of Energy. Report on Top Ten Exascale Research Challenges.2014. Available at: http://science.energy.gov/~/media/ascr/ascac/pdf/meetings/20140210/Top10reportFEB14.pdf

[9] Asanovic K, Bodik R, Catanzaro BC, *et al*. The landscape of parallel computing research: A view from Berkeley. California; 2006. Report No.: UCB/EECS-2006-183.

[10] Li X, Ruan D, van der Wal AJ. Discussion on soft computing at FLINS'96. Int J Intell Syst 1998; 13(2-3): 28-300.

[11] Berman H, Westbrook J, Feng Z, *et al.* The protein data bank. Nucleic Acids Res 2000; 28(1): 235-42.

[12] Zadeh LA. Soft computing and fuzzy logic. IEEE Software 1994; 11(6): 48-56.

[13] Verdegay JL., Yager RR, Bonissone PP. On heuristics as a fundamental constituent of soft computing. Fuzzy Set Syst 2008; 159(7): 846-55.

[14] Bonissone PP. Soft computing: the convergence of emerging reasoning technologies. Soft Comput 1997; 1(1): 6-18.

[15] Bonissone PP. In:Hybrid Soft Computing for Classification and Prediction Applications. Proceedings of the First International Conference on Computing in an Imperfect World. London: Springer-Verlag 2002; pp 352-3.

[16] Mitra S, Pal SK, Mitra P. Data mining in soft computing framework: a survey. IEEE T Neural Networ 2002; 13(1): 3-14.

[17] Zadeh LA. Fuzzy logic, neural networks, and soft computing. Commun ACM 1994; 37(3): 77-84.

[18] Zadeh LA. Fuzzy sets. Inform Control 1965; 8(3): 338-53.

[19] Haykin S, Network N. A comprehensive foundation. Neural Networks 2004; 2(2004).

[20] Hearst MA, Dumais ST, Osman E, Platt J, Scholkopf B. Support vector machines. IEEE Intell Syst App 1998; 13(4): 18-28.

[21] Glover F, Kochenberger GA. Handbook of metaheuristics. Springer Science & Business Media 2003.

[22] Back T, Fogel DB, Michalewicz Z. Handbook of Evolutionary Computation Bristol, UK: IOP Publishing Ltd. 1997.

[23] Davis L. Handbook of genetic algorithms, New York: Van Nostrand Reinhold 1991.

[24] Kennedy J, Kennedy JF, Eberhart RC. Swarm intelligence, Morgan Kaufmann 2001.

[25] Sánchez-Linares I, Pérez-Sánchez H, Cecilia JM, García JM. High-throughput parallel blind virtual screening using BINDSURF. Bioinformatics. 2012; 13.

[26] Jena RK, Aqel MM, Srivastava P, Mahanti PK. Soft computing methodologies in bioinformatics. Eur J Sci Res 2009; 26(2): 189-203.

[27] Mitra S, Hayashi Y. Bioinformatics with soft computing. IEEE T Syst Man Cy C 2006; 36(5): 616-35.

[28] Peréz-Sánchez, H, Cecilia, J M, Merelli, I. In:The role of High Performance Computing in Bioinformatics. Proceedings of International Work-Conference on Bioinformatics and Biomedical Engineering. Granada, Spain: 2014; pp 494-506.

[29] Shaw DE, Maragakis P, Lindorff-Larsen, K, *et al.* Atomic-level characterization of the structural dynamics of proteins. Science 2010; 330(6002): 341-6.

[30] Schaller RR. Moore's law: past, present and future. IEEE Spectr 1997; 34(6): 52-9.

[31] Schulz M. The end of the road for silicon? Nature 1999; 399(6738): 729-30.

[32] Pavlus J. The Search for a New Machine. Sci Am 2015; 312(5): 58-63.

[33] Huang A. Moore's Law is Dying (and that could be good). IEEE Spectr 2015; 52(4): 43-7.

[34] Top500 The List. Available at: http://www.top500.org/ [accesed June 24,2015].

[35] Kirk DB, Wen-mei WH. Programming massively parallel processors: a hands-on approach, MA, USA: Elsevier 2013.

[36] Jeffers J, Reinders J. Intel Xeon Phi coprocessor high-performance programming, MA, USA: Elsevier 2013.

[37] Carretero J, García-Blas J, Singh DE, *et al.* In:Optimizations to enhance sustainability of MPI applications. Proceedings of the 21st European MPI Users' Group Meeting. Kyoto, Japan: ACM New York 2014; pp 145.

[38] Garland M, Le Grand S, Nickolls J, *et al.* Parallel computing experiences with CUDA. IEEE micro 2008; (4): 13-27.

[39] Nickolls J, Buck I, Garland M, Skadron K. Scalable parallel programming with CUDA. ACM Queue 2008; 6(2): 40-53.

[40] nVIDIA. GPU Accelerated. Available at: http://www.nvidia.com/content/gpu-applications/PDF/GPU-apps-catalog-mar2015.pdf [accessed May 30,2015].

[41] Tsuchiyama R, Nakamura T, Iizuka T, Asahara A, Miki S, Tagawa S. The OpenCL programming book. Fixstars Corporation 2010.

[42] Garland M, Kirk DB. Understanding throughput-oriented architectures. Commun ACM 2010; 58-66.

[43] Kim HS, El Hajj I, Stratton J, Lumetta S, Hwu WM. In:Locality-centric thread scheduling for bulk-synchronous programming models on CPU architectures. Proceedings of the 13th Annual IEEE/ACM International Symposium on Code Generation and Optimization. San Francisco, CA, USA: IEEE Computer Society Washington 2015; pp 257-68.

[44] Chang LW, Dakkak A, Rodrigues CI, Hwu WM. In: Tangram: a High-level Language for Performance Portable Code Synthesis. Proceedings of Programmability Issues for Heterogeneous Multicores. Amsterdam, Netherlands, 2015.

[45] The OpenMP API specification for parallel programming. Available at: http://openmp.org/wp/ [accessed May 30,2015].

[46] OpenACC, Directives for accelerators. Available at: http://www.openacc-standard.org/ [accessed May 30, 2015].

[47] Fan X, Weber WD, Barroso LA. In: Power provisioning for a warehouse-sized computer. Proceedings of the 34th annual international symposium on Computer architecture. San Diego, CA, USA: ACM New York 2007; pp 13-23.

[48] Koomey JG. Worldwide electricity used in data centers. Environ Res 2008; 3(3).

[49] The Green 500 List. Available at: http://www.green500.org/ [accessed June 24, 2015].

[50] Guerrero GD, Wallace RM, Vázquez Poletti JL, *et al*. A performance/cost model for a CUDA drug discovery application on physical and public cloud infrastructures. Concurrency-Pract Ex 2014; 26(10): 1787-98.

[51] Hewwit C. ORGs for Scalable, Robust, Privacy-Friendly Client Cloud Computing/Carl Hewitt. IEEE Internet Comput 2008; 12(5): 96-9.

[52] Berl A, Gelenbe E, Di Girolamo M, *et al.* Energy-efficient cloud computing. Comput J 2010; 53(7): 1045-51.

[53] Armbrust M, Fox A, Griffith R, *et al.* A view of cloud computing. Commun ACM 2010; 53(4): 50-8.

[54] D'Agostino D, Clematis A, Quarati A, *et al.* Cloud infrastructures for in silico drug discovery: economic and practical aspects. Biomed Res Int 2013; 19.

[55] D'Agostino D, Galizia A, Clematis A, Mangini M, Porro I, Quarati A. A QoS-aware broker for hybrid clouds. Computing 2013; 95(1): 89-109.

[56] Shvachko K, Kuang H, Radia S, Chansler R. In: The Hadoop distributed file system. Proceedings of the 26th Symposium on Mass Storage Systems and Technologies (MSST). Incline Village, NV: IEEE 2013; pp 89-109.

[57] Buchan DW, Minneci F, Nugent TC, Bryson K, Jones DT. Scalable web services for the PSIPRED Protein Analysis Workbench. Nucleic Acids Res 2013; 41(1): 349-57.

[58] Narayanan AH, Krishnakumar U, Judy MV. In: An Enhanced MapReduce Framework for Solving Protein Folding Problem Using a Parallel Genetic Algorithm. Proceedings of the 48th Annual Convention of Computer Society of India. Springer International Publishing 2014; pp 241-50.

[59] Beberg AL, Ensign DL, Jayachandran G, Khaliq S, Pande VS. In: Folding@ home: Lessons from eight years of volunteer distributed computing. Proceedings of Parallel & Distributed Processing. Rome: IEEE 2009; pp 1-8.

[60] Folding@Home. Available at: http://folding.stanford.edu/home/the-science [accessed June 15, 2015].

[61] Kondov I, Berlich R. In: Protein structure prediction using particle swarm optimization and a distributed parallel approach. Proceedings of the 3rd workshop on Biologically inspired algorithms for distributed systems. Karlsruhe, Germany: ACM New York 2011; pp 35-42.

[62] Wooldridge M. An Introduction to Multiagent Systems Chichester, UK: John Wiley and Sons 2002.

[63] Cannata N, Corradini F, Merelli E, Omicini A, Ricci A. In: An agent-oriented conceptual framework for systems biology. Proceedings of Transactions on computational systems biology III. Berlin, Springer 2005 pp. 105-22.

[64] Bortolussi L, Dovier A, Fogolari F. Agent-based protein structure prediction. Multiagent and Grid Systems 2007; 3(2): 183-97.

[65] Czibula G, Bocicor MI, Czibula IG. Solving the protein folding problem using a distributed q-learning approach. Int J Comput Inf Sci 2011;(5): 404-13.

[66] Czibula G, Bocicor M, Czibula I. A reinforcement learning model for solving the folding problem. Int J Comput Appl T 2011; 2: 171–82.

[67] Cartmell J, Enoch S, Krstajic D, Leahy DE. Automated QSPR through competitive workflow. J Comput Aid Mol Des 2005; 19(11): 821-33.

[68] Amigoni F, Schiaffonati V. In: Multiagent-based simulation in biology. Proceeding of the Model-Based Reasoning in Science, Technology, and Medicine. Berlin, Springer 2007; pp 179-91.

[69] Hornik K, Stinchcombe M, White H. Multilayer feedforward networks are universal approximators. Neural networks 1989; 2(5): 359-66.

[70] Pollastri G, Przybylski D, Rost B, Baldi P. Improving the prediction of protein secondary structure in three and eight classes using recurrent neural networks and profiles. Proteins 2002; 47: 228–35.

[71] Rost B, Sander C. Combining evolutionary information and neural networks to predict protein secondary structure. Proteins 1994; 19(1): 55-72.

[72] Chandonia JM, Karplus M. Neural networks for secondary structure and structural class predictions. Protein Sci 1995; 4(2): 275-85.

[73] Chandonia JM, Karplus M. The importance of larger data sets for protein secondary structure prediction with neural networks. Protein Sci 1996; 5(4): 768-74.

[74] Blom N, Hansen J, Blaas D, Brunak S. Cleavage site analysis in picornaviralpolyproteins: discovering cellular targets by neural networks. Protein Sci 1996; 5(11): 2203-16.

[75] Nielsen H, Engelbrecht J, Brunak S, von Heijne G. A neural network method for identification of prokaryotic and eukaryotic signal peptides and prediction of their cleavage sites. Int J Neural Syst 1997; 8: 581-99.

[76] Nielsen H, Brunak S, von Heijne G. Machine learning approaches for the prediction of signal peptides and other protein sorting signals. Protein Eng 1999; 12(1): 3-9.

[77] Li X, Romero P, Rani M, Dunker A, Obradovic Z. Predicting protein disorder for N-, C-and internal regions. Genome Inform 1999; 10: 30-40.

[78] Sodhi JS, Bryson K, McGuffin LJ, Ward JJ, Wernisch L, Jones DT. Predicting metal-binding site residues in low-resolution structural models. J Mol Biol 2004; 342(1): 307-20.

[79] Passerini A, Punta M, Ceroni A, Rost B, Frasconi P. Identifying cysteines and histidines in transitionmetalbinding sites using support vector machines and neural networks. Proteins 2006; 65(2): 305-16.

[80] Nair R, Rost B. Better prediction of subcellular localization by combining evolutionary and structural information. Proteins 2003; 53(4): 917-30.

[81] Emanuelsson O, Nielsen H, Brunak S, von Heijne G. Predicting subcellular localization of proteins based on their N-terminal amino acid sequence. J Mol Biol 2000; 300(4): 1005-16.

[82] Reinhardt A, Hubbard T. Using neural networks for prediction of the subcellular location of proteins. Nucleic Acids Res 1998; 26(9): 2230-6.

[83] Jensen LJ, Gupta R, Blom N, *et al.* Prediction of human protein function from post-translational modifications and localization features. J Mol Biol 2002; 319(5): 1257-65.

[84] Mirabello C, Adelfio A, Pollastri G. Reconstructing Protein Structures by Neural Network Pairwise Interaction Fields and Iterative Decoy Set Construction. Biomolecules 2014; 4(1): 160-80.

[85] Igel C, Gebert J, Wiebringhaus T. In: Protein fold class prediction using neural networks with tailored early-stopping. Proceedings of Neural Networks. IEEE International Joint Conference 2004; pp 1693-7.

[86] Ding CH, Dubchak I. Multi-class protein fold recognition using support vector machines and neural networks. Bioinformatics 2001; 17(4): 349-58.

[87] Karchin R, Cline M, Mandel-Gutfreund Y, Karplus K. Hidden Markov models that use predicted local structure for fold recognition: alphabets of backbone geometry. Proteins 2003; 51(4): 504-14.

[88] Andreeva A, Howorth D, Brenner SE, Hubbard TJ, Chothia C, Murzin AG. SCOP database in 2004: refinements integrate structure and sequence family data. Nucleic Acids Res 2004; 32(1): 226-9.

[89] Rost B, Sander C. Prediction of protein secondary structure at better than 70% accuracy. J Mol Biol 1993; 232(2): 584-99.

[90] Altschul SF, Madden TL, Schaffer AA, *et al.* Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Res 1997; 25(17): 3389-402.

[91] Khan MA, Jan Z, Ali H, Mirza AM. In: Performance of Machine Learning Techniques in Protein Fold Recognition Problem. Proceedings of Information Science and Applications IEEE 2010; pp 1-6.

[92] Sangjo H, Byung-chul L, Seung TY, Chan-seok J, Soyoung L, Dongsup K. Fold recognition by combining profile–profile alignment and support vector machine. Bioinformatics 2005; 21(11): 2667-73.

[93] Xu J. Fold recognition by predicted alignment accuracy. IEEE ACM T Comput Bi 2005; 2(2): 157-65.

[94] Hua S, Sun Z. A novel method of protein secondary structure prediction with high segment overlap measure: support vector machine approach. J Mol Biol 2001; 308(2): 397-407.

[95] Schmidhuber J. Deep learning in neural networks: An overview. Neural Networks 2015; 61: 85-117.

[96] Eickholt J, Cheng J. Predicting protein residue–residue contacts using deep networks and boosting. Bioinformatics 2012; 28(23): 3066-72.

[97] Spencer M, Eickholt J, Cheng J. A Deep Learning Network Approach to ab initio Protein Secondary Structure Prediction. IEEE ACM T Comput Bi 2014: 103-12.

[98] Mnih V. Cudamat: a CUDA-based matrix class for python. Department of Computer Science, University of Toronto, Tech Rep UTML TR, 4: 2009.

[99] Blum C, Roli A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Comput Surv 2003; 35(3): 268-308.

[100] Das S, Abraham A, Konar A. In: Swarm intelligence algorithms in bioinformatics. Proceedings of Computational Intelligence in Bioinformatics. Springer Berlin Heidelberg 2008; pp 113-47.

[101] Bergholt MS, Zheng W, Lin K, *et al*. In vivo diagnosis of gastric cancer using Raman endoscopy and ant colony optimization techniques. Int J Cancer 2011; 128(11): 2673-80.

[102] Dorigo M, Birattari M, Stutzle T. Ant colony optimization. IEEE Comput Intell Mag 2006; 1(4): 28-39.

[103] Blum C. Ant colony optimization: Introduction and recent trends. Phys Life Rev 2005; 2(4): 353-73.

[104] Dorigo M, Maniezzo V, Colorni A. The ant system: optimization by a colony of cooperation agents. IEEE Trans Syst Man Cybern 1996; 29-41.

[105] Dorigo M, Caro G, Gambardella L. Ant algorithms for discrete optimization. Artif Life 1999; 5(2): 137-72.

[106] Dorigo M, Stützle T. In: Handbook of Metaheuristics; Springer US: 2003; pp 250-85.

[107] Sivagaminathan RK, Ramakrishnan S. A hybrid approach for feature subset selection using neural networks and ant colony optimization. Expert Syst Appl 2007; 33(1): 49-60.

[108] Nemati S, Basiri ME, Ghasem-Aghaee N, Aghdam MH. A novel ACO–GA hybrid algorithm for feature selection in protein function prediction. Expert Syst Appl 2009; 36(10): 12086-94.

[109] Shmygelska A, Hoos HH. An ant colony optimisation algorithm for the 2D and 3D hydrophobic polar protein folding problem. BMC bioinformatics 2005; 6(1): 30.

[110] Song J, Cheng J, Zheng T. In: Protein 3D HP model folding simulation based on ACO. Proceedings of Intelligent Systems Design and Applications. IEEE 2006; pp 410-5.

[111] Tortilla HP benchmark. Available at: http://www.cs.sandia.gov/tech_reports/compbio/tortilla-hp-benchmarks.html [accessed Jun 15, 2015]

[112] Thalheim T, Merkle D, Middendorf M. Protein folding in the HP-model solved with a hybrid population based ACO algorithm. Int J Comp Sci 2008; 35(3): 291-300.

[113] Hu XM, Zhang J, Li Y. In: Flexible protein folding by ant colony optimization. Proceedings of Computational Intelligence in Biomedicine and Bioinformatics. Berlin: Springer 2008; pp 317-36.

[114] Chen C, Tian YX, Zou XY, Cai PX, Mo JY. A hybrid ant colony optimization for the prediction of protein secondary structure. Chinese Chem Lett 2005; 16(11): 1551-4.

[115] Chu D, Zomaya A. In: Parallel Ant Colony Optimization for 3D Protein Structure Prediction using the HP Lattice Model. Nedjah N, de MacedoMourelle L, Alba E. Springer Berlin Heidelberg 2006; pp 177-198.

[116] Guo H, Lu Q, Wu J, Huang X, Qian P. In: Solving 2D HP Protein Folding Problem by Parallel Ant Colonies. Proceedings of 2nd International Conference Biomedical Engineering and Informatics. Tianjin: IEEE. 2009; pp 1 - 5.

[117] Lv Q, Wu H, Wu J, Huang X, Luo X, Qian P. A parallel ant colonies approach to de novo prediction of protein backbone in CASP8/9. Sci China Inform Sci 2013; 56(10): 1-13.

[118] Cecilia JM, García JM, Nisbet A, Amos M, Ujaldón M. Enhancing data parallelism for ant colony optimization on GPUs. J Parallel Distr Com 2013; 73(1): 42-51.

[119] Karaboga D, Basturk B. On the performance of artificial bee colony (ABC) algorithm. Appl Soft Comput 2008; 8(1): 687-97.

[120] Zhang Y, Wu L. Artificial bee colony for two dimensional protein folding. AEES 2012; 1(1): 19-23.

[121] Li B, Li Y, Gong L. Protein secondary structure optimization using an improved artificial bee colony algorithm based on AB off-lattice model. Eng Appl Artif Intel 2014; 27: 70-9.

[122] Benítez CMV, Lopes HS. In: Parallel artificial bee colony algorithm approaches for protein structure prediction using the 3DHP-SC model. Proceedings of Intelligent Distributed Computing IV. Springer Berlin Heidelberg, 2010; pp 255-64.

[123] Benitez CMV, Lopes HS. In: Hierarchical Parallel Genetic Algorithm applied to the three-dimensional HP Side-chain Protein Folding Problem. Proceedings of Intenrantional Conference on Systems Man and Cybernetics. Istanbul: IEEE 2010; pp 2669-76.

[124] Benitez CMV, Parpinelli RS, Lopes HS. Parallelism, hybridism and coevolution in a multi-level ABC-GA approach for the protein structure prediction problem. Concurr Comput 2012; 24(6): 635-46.

[125] Bahamish HAA, Abdullah R, Abu-Hashem MA. In: A modified Marriage in Honey Bee Optimisation (MBO) algorithm for protein structure prediction. Proceedings of 2nd International Conference on Computer Technology and Development. Cairo: IEEE 2010; pp 65-9.

[126] Wang Y, Guo GD, Chen LF. Chaotic Artificial Bee Colony algorithm: A new approach to the problem of minimization of energy of the 3D protein structure. Mol Biol+ 2013; 47(6): 894-900.

[127] Li B, Chiong R, Lin M. A balance-evolution artificial bee colony algorithm for protein structure optimization based on a three-dimensional AB off-lattice model. Comput Biol Chem 2015; 54: 1-12.

[128] Li Y, Zhou C, Zheng X. Artificial Bee Colony Algorithm for the Protein Structure Prediction Based on the Toy Model. Fundam Inform 2015; 136(3): 241-52.

[129] Chen X, Lv M, Zhao L, Zhang X. An Improved Particle Swarm Optimization for Protein Folding Prediction. IJIEEB 2011; 3(1): 1-8.

[130] Pérez-Hernández LG, Rodríguez-Vázquez K, Garduño-Juárez R. In: Parallel particle swarm optimization applied to the protein folding problem. Proceedings of the 11th Annual conference on Genetic and evolutionary computation. New York: ACM 2009; pp 1791-2.

[131] Pérez-Hernández LG, Rodríguez-Vázquez K, Gorduño-Juárez R. In: Estimation of 3D protein structure by means of parallel particle swarm optimization. Proceedings of Evolutionary Computation. Barcelona: IEEE 2010; pp 1-8.

[132] Liu J, Wang L, He L, Shi F. In: Analysis of toy model for protein folding based on particle swarm optimization algorithm. Proceedings of First International Conference. Changsha, China: Springer Berlin Heidelberg 2005; pp 636-45.

[133] Mansour N, Kanj F, Khachfe H. Particle swarm optimization approach for protein structure prediction in the 3D HP model. Interdiscip Sci 2012; 4(3): 190-200.

[134] Goldberg DE. Genetic algorithms in search, optimization and machine learning. Addison-Wesley 1989.

[135] De Jong KA, Spears WM. In: Using genetic algorithms to solve NP-complete problems. Proceedings of International Conference on Genetic Algorithms. California: Morgan Kaufmann Publishers, 1989; pp 124-32.

[136] Holland JH. Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence. Oxford, England: U Michigan Press 1975.

[137] Unger R, Moult J. Genetic algorithms for protein folding simulations. J Mol Biol 1993; 231(1): 75-81.

[138] König R, Dandekar T. Improving genetic algorithms for protein folding simulations by systematic crossover. BioSystems 1999; 50(1): 17-25.

[139] Patton AL, Punch III WF, Goodman ED. In: A Standard GA Approach to Native Protein Conformation Prediction. Proceedings of International Conference of Genetic Algorithms; 1995. pp 574-81.

[140] Pedersen JT, Moult J. Protein folding simulations with genetic algorithms and a detailed molecular description. J Mol Biol 1997; 269(2): 240-59.

[141] Lopes HS, Scapin MP. In: An enhanced genetic algorithm for protein structure prediction using the 2D hydrophobic-polar model. Talbi EG, Liardet P, Collet P, Lutton E, Schoenauer M. Artificial Evolution: Springer Berlin Heidelberg 2006. pp 238-46.

[142] Hoque MT, Chetty M, Dooley LS. In: A new guided genetic algorithm for 2D hydrophobic-hydrophilic model to predict protein folding. Proceedings of Evolutionary Computation. Edinburgh, Scotland: IEEE 2005; pp 259-66.

[143] Song J, Cheng J, Zheng T, Mao J. In: A novel genetic algorithm for HP model protein folding. Proceedings of Sixth International Conference on Parallel and Distributed Computing, Applications and Technologies. IEEE 2005; pp. 935-37.

[144] Sun S. Reduced representation model of protein structure prediction: statistical potential and genetic algorithms. Protein Sci 1993; 2(5): 762-85.

[145] Dandekar T, Argos P. Folding the main chain of small proteins with the genetic algorithm. J Mol Biol 1994; 236(6): 844-61.

[146] Zhang X, Wang T, Luo H, *et al.* In: 3D Protein structure prediction with genetic tabu search algorithm. Proceedings of The ISIBM International Joint Conferences on Bioinformatics, Systems Biology and Intelligent Computing (IJCBS). Shanghai, China: 2009;

[147] Jiang T, Cui Q, Shi G, Ma S. Protein folding simulations of the hydrophobic–hydrophilic model by combining tabu search with genetic algorithms. J Chem Phys 2003; 119(8): 4592-6.

[148] Rashid MA, Hoque MT, Newton MH, Pham DN, Sattar A. In: A New Genetic Algorithm for Simplified Protein Structure Prediction. Proceedings of 25th Australasian Joint Conference. Sydney, Australia: Springer Berlin Heidelberg 2012; pp 107-19.

[149] Cotta C. In: Protein structure prediction using evolutionary algorithms hybridized with backtracking. Proceedings Artificial Neural Nets Problem Solving Methods. Springer Berlin Heidelberg, 2003; pp 321-8.

[150] Chira C. Hill-Climbing search in evolutionary models for protein folding simulations. Stud Univ Babe\c s-Bolyai Inform 2010; 55: 29-40.

[151] Zhang X, Lin X, Wan C, Li T. In: Genetic-annealing algorithm for 3D off-lattice protein folding model. Proceedings of Emerging Technologies in Knowledge Discovery and Data Mining. Springer Berlin Heidelberg 2007; pp 186-93.

[152] Moscato P, Cotta C. In: A gentle introduction to memetic algorithms. Glove F, Kochenberger GA. Handbook of Metaheuristics: Springer US 2003. pp. 105-44.

[153] Islam MK, Chetty M. In: Novel memetic algorithm for protein structure prediction. Proceedings of Advances in Artificial Intelligence. Springer Berlin Heidelberg 2009; pp 412-21.

[154] Krasnogor N, Blackburne BP, Burke EK, Hirst JD. In: Multimeme Algorithms for Protein Structure Prediction. Proceedings of 7th International Conference of Parallel Problem Solving from Nature. Granada, Spain: Springer Berlin Heidelberg 2002; pp 769-78.

[155] Smith JE. In: The co-evolution of memetic algorithms for protein structure prediction. Hart WE, Smith JE, Krasnogor N. Recent Advances in Memetic Algorithms. Springer Berlin Heidelberg 2005. pp 105-28.

[156] Bazzoli A, Tettamanzi AG. In: A memetic algorithm for protein structure prediction in a 3D-lattice HP model. Proceedings of Applications of Evolutionary Computing. Springer Berlin Heidelberg 2004; pp 1-10.

[157] Islam MK, Chetty M. Clustered memetic algorithm with local heuristics for ab initio protein structure prediction. IEEE T Evolut Comput 2013; 17(4): 558-76.

[158] Islam MK, Chetty M, Murshed M. In: Novel local improvement techniques in clustered memetic algorithm for protein structure prediction. Proceedings of Evolutionary Computation. New Orleans, LA: IEEE 2011; pp 1003-11.

[159] Smith JE. In: Protein structure prediction with co-evolving memetic algorithms. Proceedings of the congress on Evolutionary Computation. IEEE 2003; pp 2346-53.

[160] Coello CAC, Van Veldhuizen DA, Lamont GB. Evolutionary algorithms for solving multi-objective problems New York: Kluwer Academic 2007.

[161] Day RO, Zydallis JB, Lamont GB, Pachter R. Solving the protein structure prediction problem through a multiobjective genetic algorithm. Nanotechnology 2002; 2: 32-5.

[162] Brasil CRS, Delbem ACB, da Silva FLB. Multiobjective evolutionary algorithm with many tables for purely ab initio protein structure prediction. J Comput Chem 2013; 34(20): 1719-34.

[163] SoaresBrasil CR, BotazzoDelbem AC, FerrazBonetti DR. In: Investigating relevant aspects of MOEAs for protein structures prediction. Proceedings of the 13th annual conference on Genetic and evolutionary computation. Dubin, Ireland: ACM 2011; pp 705-12.

[164] Cutello V, Narzisi G, Nicosia G. A multi-objective evolutionary approach to the protein structure prediction problem. J R Soc Interface 2006; 3(6): 139-51.

[165] Garza-Fabre M, Rodriguez-Tello E, Toscano-Pulido G. In: Multiobjectivizing the HP model for protein structure prediction. Proceedings of 12th European Conference Evolutionary Computation in Combinatorial Optimization. Málaga, Spain: Springer Berlin Heidelberg 2012; pp 182-93.

[166] Handl J, Lovell SC, Knowles J. In: Investigations into the effect of multiobjectivization in protein structure prediction. Proceedings of 10th International Conference on Parallel Problem Solving from Nature. Dortmund, Germany: Springer Berlin Heidelberg 2008; pp 702-11.

[167] Garza-Fabre M, Toscano-Pulido G, Rodriguez-Tello E. In: Locality-based multiobjectivization for the HP model of protein structure prediction. Proceedings of the 14th annual conference on Genetic and evolutionary computation. New York: ACM 2012; pp 473-80.

[168] Calvo JC, Ortega J. In: Parallel protein structure prediction by multiobjective optimization. Proceedings of 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing. Weimar, Germany: IEEE 2009; pp 268-75.

[169] Calvo JC, Ortega J, Anguita M, Urquiza JM, Florido JP. In: Protein structure prediction by evolutionary multi-objective optimization: search space reduction by using rotamers. Proceedings of Bio-Inspired Systems: Computational and Ambient Intelligence. Springer Berlin Heidelberg 2009; pp 861-8.

[170] Calvo JC, Ortega J, Anguita M. Comparison of parallel multi-objective approaches to protein structure prediction. J Supercomput 2011; 58(2): 253-60.

[171] Calvo JC, Ortega J, Anguita M. PITAGORAS-PSP: Including domain knowledge in a multi-objective approach for protein structure prediction. Neurocomputing 2011; 2675-82.

[172] Tantar A, Melab N, Talbi EG. In: A comparative study of parallel metaheuristics for protein structure prediction on the computational grid. Proceedings of Parallel and Distributed Processing Symposium. Long Beach, CA: IEEE 2007; pp 1-10.

[173] Tantar AA, Melab N, Talbi EG, Parent B, Horvath D. A parallel hybrid genetic algorithm for protein structure prediction on the computational grid. Future Gener Comp Sy 2007; 23(3): 398-409.

[174] Tantar AA, Melab N, Talbi EG. A grid-based genetic algorithm combined with an adaptive simulated annealing for protein structure prediction. Soft Comput. 2008; 12(12): 1185-1198.

[175] Cahon S, Melab N, Talbi EG. ParadisEO: A framework for the reusable design of parallel and distributed metaheuristics. J Heuristics 2004; 10(3): 357-80.

[176] Thain D, Tannenbaum T, Livny M. Distributed computing in practice: The Condor experience. Concurr Comp-Pract E 2005; 17(2-4): 323-56.

[177] Benítez CMV, Lopes HS. Protein structure prediction with the 3D-HP side-chain model using a master–slave parallel genetic algorithm. J Braz Comp Soc 2010; 16(1): 69-78.

[178] Xue Y, Qian Z, Bogdan P, Ye F, Tsui CY. In: Disease Diagnosis-on-a-Chip: Large Scale Networks-on-Chip based Multicore Platform for Protein Folding Analysis. Proceedings of 51st ACM/EDAC/IEEE Design Automation Conference. San Francisco, CA: IEEE 2014; pp 1-6.

[179] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. Commun ACM 2008; 107-13.

[180] Zhu H, Xiao H, Gu J. In: Parallelism of Clonal Selection for PSP on CUDA. Proceedings of 3rd International Conference on Intelligent Networks and Intelligent Systems Shenyang: IEEE 2010; pp 467-70.

[181] Scalabrin MH, Parpinelli RS, Benítez CM, Lopes HS. Population–based harmony search using GPU applied to protein structure prediction. gbs-ijcse 2014; 9(1): 106-18.

[182] Mansour N, Kanj F, Khachfe H. Enhanced genetic algorithm for protein structure prediction based on the HP model: intech 2011;

[183] Garcia-Martinez JM, Garzón EM, Cecilia JM, Perez-Sanchez H, Ortigosa PM. An efficient approach for solving the HP Protein Folding Problem based on UEGO. J Math Chem 2015; 794-806.

[184] Zhang Y, Wu L, Wang S. Solving two-dimensional HP model by firefly algorithm and simplified energy function. Math Probl Eng 2013;

[185] Cai X, Wu X, Wang L, Kang Q, Wu Q. Hydrophobic-polar model structure prediction with binary-coded artificial plant optimization algorithm. J Comput Theor Nanos 2013; 10(6): 1550-4.

[186] Cui Z, Liu X, Liu D, Zeng J, Shi Z. Using Gravitropism Artificial Plant Optimization Algorithm to Solve Toy Model of Protein Folding. J Comput Theor Nanos 2013; 10(6): 1540-4.

[187] Cai X, Liu D, Wang L, Kang Q, Wu Q. Using Social Emotional Optimization Algorithm to Solve Toy Model of Protein Folding. J Comput Theor Nanos 2013; 10(6): 1545-9.

[188] Lin CJ, Su SC. Protein 3 D HP Model Folding Simulation Using a Hybrid of Genetic Algorithm and Particle Swarm Optimization. Int J Fuzzy Syst 2011; 13(2): 140-7.

[189] Zhao X. Advances on protein folding simulations based on the lattice HP models with natural computing. Appl Soft Comput 2008; 8(2): 1029-40.

[190] Karami Y, Khakzad H, Arab S, Fathy M, Shirazi H. In: Protein structure prediction using bio-inspired algorithm: A review. Proceedings of 16th CSI International Symposium on Artificial Intelligence and Signal Processing. Shiraz, Fars: IEEE 2012; pp 201-6.

[191] Glover F, Laguna M. Tabu search. In Du DZ, Pardalos PM. Handbook of Combinatorial Optimization. Springer US 1999. pp 2093-229.

[192] Russell S, Norvig P. A modern approach. Artificial Intelligence: Prentice-Hall 1995.

[193] Kirkpatrick S, Gelatt JCD, Vecchi MP. Optimization by simulated annealing. Science 1983; 220(4598): 671-80.

[194] Eglese RW. Simulated annealing: a tool for operational research. Eur J Oper Res 1990; 46(3): 271-81.

[195] Neal L, Mitzenmacher M, Whitesides S. In: A complete and effective move set for simplified protein folding. Proceedings of the seventh annual international conference on Research in computational molecular biology. New York: ACM 2003; pp 188-95.

[196] Błażewicz J, Łukasiak P, Miłostan M. Application of tabu search strategy for finding low energy structure of protein. Artif Intell Med 2005; 35(1): 135-45.

[197] Cebrián M, Dotú I, Van Hentenryck P, Clote P. In: Protein structure prediction on the face centered cubic lattice by local search. Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence. 2008; pp 241-6.

[198] Yue K, Fiebig K, Thomas P, Chan H, Shakhinovich E, and Dill K. In: A test of lattice protein folding

algorithms. Proceedings of the National Academy of Sciences. 1995. pp 325-9.

[199] Dotu I, Cebrián M, Van Hentenryck P, Clote P. On lattice protein structure prediction revisited. IEEE/ACM Tans Comput Biol Bioinf 2011; 8(6): 1620-32.

[200] Rashid MA, Newton MAH, Hoque MT, Shatabda S, Pham D, Sattar A. Spiral search: a hydrophobic-core directed local search for simplified PSP on 3D FCC lattice. BMC Bioinformatics 2013; 14.

[201] Zhou C, Hou C, Zhang Q, Wei X. Enhanced hybrid search algorithm for protein structure prediction using the 3D-HP lattice model. J Mol Model 2013; 19(9): 3883-91.

[202] Morales LB, Garduño–Juárez R, Aguilar–Alvarado JM, Riveros–Castro FJ. A parallel tabu search for conformational energy optimization of oligopeptides. J Comput Chem 2000; 21(2): 147-56.

[203] Xiaolong Z, Cheng W. In: An improved tabu search algorithm for 3D protein folding problem. Proceedings of 10th Pacific Rim International Conference on Artificial Intelligence: Trends in Artificial Intelligence. Hanoi, Vietnam: Springer Berlin Heidelberg 2008; pp 1104-9.

[204] Liu J, Sun Y, Li G, Song B, Huang W. Heuristic-based tabu search algorithm for folding two-dimensional AB off-lattice model proteins. Comp Biol Chem 2013; 47: 142-8.

[205] Su SC, Lin CJ, Ting CK. An effective hybrid of hill climbing and genetic algorithm for 2D triangular protein structure prediction. Proteome Sci 2011; 9: 19.

[206] Hoque MT, Chetty M, Dooley LS. In: A hybrid genetic algorithm for 2D FCC hydrophobic-hydrophilic lattice model to predict protein folding. Proceedings of 19th Australian Joint Conference on Artificial Intelligence. Hobart, Australia: Springer Berlin Heidelberg 2006; pp 867-76.

[207] Böckenhauer HJ, Ullah AZMD, Kapsokalivas L, Steinhöfel K. In: A local move set for protein folding in triangular lattice models. Proceedings of 8th International Workshop. Karlsruhe, Germany: Springer Berlin Heidelberg 2008; pp 369-81.

[208] Chira C, Horvath D, Dumitrescu D. Hill-Climbing search and diversification within an evolutionary approach to protein structure prediction. BioData Min 2011; 4(1): 23.

[209] Cooper LR, Corne DW, Crabbe MJC. Use of a novel Hill-climbing genetic algorithm in protein folding simulations. Comp Biol Chem 2003; 27(6): 575-80.

[210] Dandekar T, Argos P. Identifying the tertiary fold of small proteins with different topologies from sequence and secondary structure using the genetic algorithm and extended criteria specific for strand regions. J Mol Biol 1996; 256(3): 645-60.

[211] Ullah AD, Steinhöfel K. In: A hybrid approach to protein folding problem integrating constraint programming with local search. Proceedings of the Eighth Asia Pacific Bioinformatics Conference. Bangalore, India: LaxmiParida and Gene Myers 2010;

[212] Simons KT, Kooperberg C, Huang E, Baker D. Assembly of protein tertiary structures from fragments with similar local sequences using simulated annealing and Bayesian scoring functions. J Mol Biol 1997; 268(1): 209-25.

[213] Albrecht AAM, Skaliotis A, Steinhöfel K. Stochastic protein folding simulation in the three-dimensional HP-model. Comp Biol Chem 2008; 32(4): 248-55.

[214] Beutler TC, Dill KA. A fast conformational search strategy for finding low energy structures of model proteins. Protein Sci 1996; 5(10): 2037-43.

[215] Web of Knowledge. Available at:www.webofknowledge.com [accessed Jun 25, 2015].

[216] Deng, L., Yu, D. Deep learning: methods and applications. Fond T Sign Proc 2014; 7(3–4): 197-387.

2.4 Parallel Ant Colony Optimization for the HP Protein Folding Problem

Título	*Parallel Ant Colony Optimization for the HP Protein Folding Problem*
Autores	Antonio. Llanes, Carlos Vélez, Antonia Sánchez, Horacio Pérez-Sánchez y José M. Cecilia
Congreso	4th International Work-Conference on Bioinformatics and Biomedical Engineering (IWBBIO)
Lugar	Granada, España
Año	2016
Páginas	615–626
Estado	Publicado

Contribución del Doctorando
Antonio Llanes Castro, declara ser el principal autor y el principal contribuidor del artículo *Comparative evaluation of platforms for parallel Ant Colony Optimization.*

Parallel Ant Colony Optimization for the HP Protein Folding Problem

Antonio Llanes, Carlos Vélez, Antonia M. Sánchez, Horacio Pérez-Sánchez, and José M. Cecilia

Bioinformatics and High Performance Computing Research Group (BIO-HPC),
Computer Science Department,
Universidad Católica San Antonio de Murcia (UCAM), Spain
{allanes, asanchez, hperez, jmcecilia}@ucam.edu
cvelez@alu.ucam.edu

Abstract. Ant Colony Optimisation (ACO) is a bio-inspired population-based metaheuristic which emulates the ant colony's behavior to solve problems computationally. Indeed, it is a *swarm*-based algorithm as it needs the interactions among all ants to provide good solutions to a particular problem. This collective computation is theoretically well-suited for parallelisation as several ants run in parallel looking for solutions, sharing their findings among them. In this paper, we design an ACO metaheuristic to solve the Protein Folding Problem using a simplified model (HP) that identifies amino acids like Hydrophobic (H) or Polar(P), attending to the attraction or the rejection that the amino acid present against water. We also propose a parallel ACO version applied to the HP model on Graphics Processing Units (GPUs) using Compute Unified Device Architecture (CUDA). Our results reveal up to 7x speed-up factor compared to a sequential counterpart version. Results and conclusions about this parallel version suggests a broader area of inquiry, where researchers within the fields of Bioinformatics may learn to adapt similar problems to the tupla of an optimization method and GPU architecture.

Keywords: ACO, HP, GPUs, HPC, CUDA

1 Introduction

Ant Colony Optimization (ACO) was originally introduced by Dorigo et al. [12]. This is a stochastic algorithm used to solve several computational problems by simulating the behavior of an ant colony. As in the real life, an ant does not have enough intelligence to solve a particular problem by itself, but all ants within the colony can cooperate to solve problems efficiently. This algorithm can be classified as *Swarm Intelligence* [20], or *Metaheuristics* [17], both of them belong to a large number of algorithms within the umbrella of *Soft Computing* [27, 3]. ACO has been probed in a variety of problems, including vehicle routing [28], feature selection [6], or autonomous robot [15].

Of particular interest to us is hydrophobichydrophilic (HP) model introduced by Dill [8] to reduce the protein folding complexity. This model assumes that the hydrophobic interactions make an important contribution to the free energy of the folding process, so a protein is modeled as an specific sequence of hydrophobic (H for nonpolar) or hydrophilic (P for polar) monomers. The optimal solution to this model is the conformation with more number of adjacencies between H's that originally were not contiguous. This problem is a NP-complete optimization problem according to [2].

Due to the relevance of Protein Folding Problem and the effectiveness of the HP model, intensive research work in this line has been recently developed. Therefore, several approaches based on the application of different optimization methods are described in literature including Monte Carlo methods [25], evolutionary algorithms [24, 16], and particle swarm optimization [21], just to name a few.

The choice of model and its associated algorithm is mainly motivated by the required objectives, but it is also constrained by the computer hardware characteristics attainable in the relevant time frame. The role of the software developer is increasingly important as their algorithms are expected to handle a soft balance between performance, power consumption and the quality obtained in the results. About performance, developers have to test different implementations, considering which code fits perfectly with the hardware platform where they run their codes. About power consumption, this issue is increasingly importance specially in large clusters [23]. Finally, developers have to take care about quality due to the inherit stochastic nature of ACO, so the goal of ACO is to reduce drastically the computation time maintaining the quality of the results.

This paper shows the parallelisation of ACO metaheuristic on Graphics Processing Units to solve the Protein Folding Problem using HP model. Our implementation leads to factor gains exceeding 7x as applied to the protein folding when compared to its sequential counterpart version running on a similar single-threaded high-end CPU. Moreover, an extensive discussion focused on different implementation paths on GPUs shows the way to deal with parallel graph connected components. The rest of the paper is structured as follows: First, a description of ACO and HP are introduced, next we describe the process used to implement the parallel version, giving details of our design. Then, preliminary experimental results are shown to finish with some conclusions and directions for future work.

2 Methodology

This section describes HP model for the protein folding as well as the way we adapt the ACO algorithm to optimize this problem. Moreover, we briefly review the main characteristics of CUDA for those who are not familiar with this programming model.

2.1 Description of HP model

HP model for the protein folding problem was introduced by Dill et. al [8], and it has been widely used to predict protein structures, like [1, 4, 22]. In HP model each protein sequence is represented as a string $A = a_1, a_2, .., a_n$, where $a_i \in H, P$ and $1 \leq i \leq n$. A conformation of A is defined by a sequence of fold directions starting from the lattice site occupied by the first amino acid a_1. The different protein conformations are restricted to self-avoiding paths on two or three dimensions. Most protein structure prediction methodologies assume that the native state of the protein is defined by the lowest value of the Gibbs free energy what is estimated by a specific scoring function, which strongly depends on the coarse-grained or all-atom model used for representing the protein structure [9].

In HP model, the energy of a conformation is defined by a scoring function that assumes that the hydrophobic interactions make an important contribution to the free energy of the folding process. The optimal solution to this model will be the conformation with more number of adjacencies between H's (topological contacts) that originally weren't contiguous in the given sequence. Thus, the Protein Folding Problem is translated into an optimization problem as follows: Given an amino acid sequence $A = a_1, a_2, .., a_n$, find an energy minimizing conformation $\mathcal{C}^o$.

$$E\mathcal{C}^o = min\{E(\mathcal{C})\forall\mathcal{C}\} \tag{1}$$

Where $\mathcal{C}$ is a valid conformation of the string A and $E(\mathcal{C})$ is defined by

$$E\mathcal{C} = \sum_{i,j} e(a_i, a_j) \tag{2}$$

where
$e(a_i, a_j) = \begin{cases} -1 \text{ if } a_i, a_j = HH \text{ and they form a topological contact} \\ 0 \text{ otherwise} \end{cases}$

Therefore, the objective function for the HP protein folding problem is defined by Eq. 2, and its values will be referred as scoring values. The goal of our parallel version of ACO is the computation of the conformations which achieve minimum values of the energy function according to Eq. 2.

2.2 Ant Colony Optimization for the protein folding based on HP model

Ant Colony Optimization (ACO) [11, 7, 13] is based on foraging behavior observed in colonies of real ants. The method generally uses simulated "ants" (i.e., mobile agents), which first construct tours or paths on a network structure (corresponding to solutions for a problem), and then deposit "pheromone" (i.e., signaling chemicals) according to the quality of the solution generated. The algorithm takes advantage of emergent properties of the multi-agent system, where positive feedback (facilitated by pheromone deposition) quickly drives the population to high quality solutions.

The original ACO method (called the *Ant System* [12]) was developed by Dorigo in the 1990s, and this version (or slight variants thereof, such as the MAX-MIN Ant System (MMAS) [26]) is still in regular use [5, 19, 14]. The Ant System (AS) algorithm is divided into two main stages: *Conformation construction* and *Pheromone update*. Conformation construction is based on m ants building protein conformations in parallel. Those protein conformations are constructed based on a probabilistic action choice rule, called the *random proportional rule* in order to decide which position to place the next amino acid (restricted to H or P). The probability for ant k, placed at position i, of amino acid j is given by the equation 3

$$p_{i,j}^{k} = \frac{[\tau_{i,j}]^{\alpha}\,[\eta_{i,j}]^{\beta}}{\sum_{l \in N_i^k} [\tau_{i,l}]^{\alpha}\,[\eta_{i,l}]^{\beta}}, \qquad if\ j \in N_i^k, \tag{3}$$

where $\eta_{i,j} = C_{i,j}$ is a heuristic value that represents the number of non-contiguous H-H contacts, α and β are two parameters which determine the relative *influences* of the pheromone trail and the heuristic information respectively, and N_i^k is the feasible neighbourhood of ant k when at position i. This latter set represents the set of positions that ant k has not yet visited; the probability of choosing a position outside N_i^k is zero (this prevents an ant returning to a position, which is not allowed in the HP model). By this probabilistic rule, the probability of choosing a particular edge (i, j) increases with the value of the associated pheromone trail $\tau_{i,j}$ and of the heuristic information value $\eta_{i,j}$. The random proportional rule ends with a selection procedure, which is done analogously to the *roulette wheel* selection procedure of evolutionary computation (for more detail see [10], [18]). Each value $\tau_{i,j}^{\alpha}\eta_{i,j}^{\beta}$ of a position j that ant k has not visited yet determines a slice on a circular roulette wheel, the size of the slice being proportional to the weight of the associated choice. Next, the wheel is spun and the position to which the marker points is chosen as the next position for ant k. Furthermore, each ant k maintains a memory, M^k, called the *tabu list*, which contains the positions already visited. This memory is used to define the feasible neighbourhood, and also allows an ant to both to compute the conformation's score T^k it generated, and to retrace the path to deposit pheromone.

After all ants have constructed their protein conformations, the pheromone trails are updated. This is achieved by first lowering the pheromone value on all edges by a constant factor, and then adding pheromone on edges that ants have crossed in their conformations. Pheromone evaporation is implemented by

$$\tau_{i,j} \leftarrow (1-\rho)\tau_{i,j}, \qquad \forall (i,j) \in L, \tag{4}$$

where $0 < \rho \leq 1$ is the pheromone evaporation rate. After evaporation, all ants deposit pheromone on their visited edges:

$$\tau_{i,j} \leftarrow \tau_{i,j} + \sum_{k=1}^{m} \Delta\tau_{i,j}^{k}, \qquad \forall (i,j) \in L, \tag{5}$$

where $\Delta\tau_{ij}$ is the amount of pheromone ant k deposits. This is defined as follows:

$$\Delta\tau_{i,j}^{k} = \begin{cases} 1/C^k \text{ if } & e(i,j)^k \text{ belongs to } T^k \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

where C^k, the score of the conformation T^k built by the k-th ant, is computed as the sum of number of non-contiguous H-H contacts belonging to T^k. According to equation 6, the better an ant's conformation, the more pheromone the edges belonging to this conformation receive. In general, edges that are used by many ants, receive more pheromone, and are therefore more likely to be chosen by ants in future iterations of the algorithm.

2.3 CUDA programming model

Before we discuss our parallel versions, we briefly review the main characteristics of CUDA, for the benefit of readers who are unfamiliar with the programming model. CUDA is based on a hierarchy of abstraction layers; the *thread* is the basic execution unit; threads are grouped into *blocks*, each of which runs on a single multiprocessor, where they can share data on a small but extremely fast memory. A *grid* is composed of blocks, which are equally distributed and scheduled among all multiprocessors. The parallel sections of an application are executed as *kernels* in a SIMD (Single Instruction Multiple Data) fashion, that is, with all threads running the same code. A kernel is therefore executed by a grid of thread blocks, where threads run simultaneously grouped in batches called *warps*, which are the scheduling units.

3 Parallel Ant Colony Optimization on GPUs for the HP protein folding

This Section summarizes the parallelization process of the Ant Colony Optimization as applied to the HP protein folding using CUDA. Algorithm 1 shows Single Program Multiple Data (SPMD) pseudocode for the AS. Firstly, all AS structures for the HP protein folding problem (Conformation matrix, number of amino acids,...) are initialized. Next, the conformation construction and pheromone update stages are performed until the convergence criterion is reached (number of iterations in our case). Both stages are executed by each ant and it can be repeated several times from the beginning to restart the computation. This restarts are implemented to avoid stalling in local optimum.

Algorithm 1 Sequential pseudocode of ACO-HP

```
1: for each epoch do
2:    for each ant do
3:       conformationConstruction()
4:       updatePheromoneMatrix()
5:    end for
6: end for
```

In what follows, we describe the parallelization approach for both main kernels of ACO algorithm: Conformation construction and Pheromone Update.

3.1 Conformation construction parallelization

The ”traditional” task parallelism approach to conformation construction is based on the observation that ants run in parallel looking for the best protein conformation they can find. Therefore, any inherent parallelism exists at the level of individual ants. To implement this idea of parallelism using CUDA, each ant is identified as a CUDA thread, and threads are equally distributed among CUDA thread blocks. Each thread deals with the task assigned to each ant; i.e., maintenance of an ants memory (list of all visited positions, and so on) and movement which is mainly based on the *random proportional rule* previously explained.

3.2 Pheromone update parallelization

The final stage in the ACO algorithm is pheromone update, which comprises two main tasks: pheromone evaporation and pheromone deposit. The first step is quite straightforward to implement in CUDA, as a single thread can independently calculate the Eq. 4 for each entry of the pheromone matrix, thus lowering the pheromone value on all edges by a constant factor.

Ants then deposit different quantities of pheromone on the edges that they have crossed to create their conformations. As stated previously, the quantity of pheromone deposited by each ant depends on the quality of the protein conformation found by that ant. This kernel allocates a thread per each amino acid, which is placed at different positions. Each ant generates its own private conformation in parallel, and they may place an amino acid into the same position as another ant. This fact forces us to use atomic instructions for accessing the pheromone matrix.

4 Experimental Results

This section briefly shows the experimental results obtained with our implementation. First of all we review our experimental set up. Then we proceed with an evaluation of both sequential and parallel versions before we present the experimental results.

4.1 Hardware environment

During our experimental study, we have used the following platforms:

– **On the CPU side:** Four Intel Xeon X7550 processors running at 2 GHz and plugged into a quad-channel motherboard endowed with 128 Gigabytes of DDR3 memory.
– **On the GPU side:** GPU NVIDIA Tesla Kepler K40c with 2.880 cores, (15 multiprocessors with 192 cores each), running at 880 MHz, offering a processing power up to 5.068 GFLOPS. It also have 12 GB of GDDR5 RAM with ECC capability and a buswidth of 384 bits, giving a bandwidth of 288 GB per second.

Table 1. Hardware Description

	Vendor and type	**Intel CPU**	**NVIDIA**
	Family	**Haswell**	**Kepler**
	Class	**Xeon**	**Tesla**
	Model	**X7750**	**K40c**
	Year	**2015**	**2014**
Processing elements	Cores per multiprocessor	(does not	192
	Number of multiprocessors	apply)	15
	Total number of cores	8	2880
	Clock Frequency (MHz)	2000	880
Maximum number of GPU threads	Per multiprocessor	(does	2048
	Per block	not	1024
	Per warp	apply)	32
SRAM Memory (per multiprocessor in GPU)	Shared (only GPU)	32 KB L1D	16 or 48 KB
	L1 Cache	and	48 or 16 KB
	(Shared + L1)	32 KB L1I	64 KB
L2 Cache	(shared by all cores)	256 KB	1536 KB
L3 Cache		16 MB	(d.n.a)
DRAM Memory	Size (MB)	131072	11520
	Speed (MHz)	2x666	2x3004
	Width (bits)	256	384
	Bandwidth (GB/s)	42.66	288.34
	Tecnology	DDR3	GDDR5
CUDA Compute Capabilities		(d.n.a.)	3.5

Table 1 shows a detailed descriptions of all these platforms. Moreover, we use gcc 4.8.2 with the -O3 flag to compile on the CPU, and the CUDA compiler/driver-/runtime version 6.5 to compile and run on the GPU.

4.2 Profiling

This section briefly shows the performance analysis for both GPUs and CPU codes. The profiling is performed with Microsoft Visual Profiler for the sequential code, and with NVIDIA Visual Profiler for the CUDA code. Figure 1 shows the sequential code profiling where the Conformation Construction stage takes more than half of total computation time, this function is the responsible to create a valid conformation for each ant. In second position is the pheromone stage which is actually parallelized as well.

The table 2 summarizes all the kernels implemented in the new parallel code, with theirs occupancy rate. All kernels have a high occupancy rate.

4.3 Execution results

This section shows our experimental results from several points of view. Firstly, we evaluate the scalability of our parallel implementation, varying the number of ants from 256 to 8192 ants (see Table 3). In the conformation construction

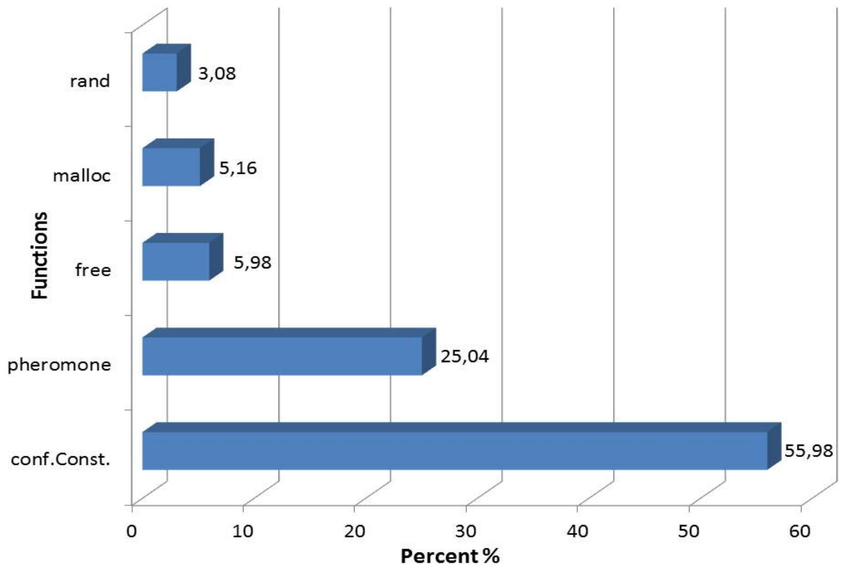


Fig. 1. Main functions for our ACO implementation

Table 2. Kernel occupancy

Function Name	**Duration (μs)**	**Occupancy**
conformationConstruction_Cuda	1.888.756,607	62.50%
startPheromoneMatrix_Cuda	65.182,431	100%
pheromoneVaporize_Cuda	51.420,896	100%
stateVectorGenerator	1.198,016	75%
updatePheromoneMatrix_Cuda	51,904	100%
iamax_kernel	8,479	100%
updatePheromoneMatrix_BestSolution_Cuda	5,216	100%

stage, each ant is identify to a CUDA thread and those threads are equally divided into blocks. Therefore, the number of threads per block depends on the number of ants set. Our algorithm prevents this situation by setting our empirically demonstrated optimum thread block layout for each case. Whenever the number of ants is large enough, the best configuration is for 256 threads per block.

Table 3 shows a great scalability along with the number of ants. However, it also shows that with a low number of ants, sequential code run even faster than the parallel version. It is due to the extra charge of transferring of data between host and device, in these cases, CPU can support the charge of the complexity of the problem in order to maintain advantage from the parallel implementation. But, in the other hand, as soon as we increment the number of ants, the sequential code experiments an exponential increment of time, unlike the CUDA version. This is exactly what we expect as the data parallelism in this problem is not very high, besides, we have implemented the kernel with more time consumption "conformationConstruction()", and we established the association from one ant to one thread, our benefits become higher when we increase the number of ants.

Table 3. Execution time (in seconds) for both: sequential and CUDA implementations of HP protein folding problem by varying the number of ants.

	Execution Time (sec.)	
Number of Ants	**Sequential Version**	**CUDA Version**
256	*16.188*	53.2158
512	*30.114*	53.4919
1024	59.035	*53.891*
2048	117.434	*53.2006*
4096	234.533	*57.8629*
8192	483.835	*67.3611*

Table 3 shows the execution times in seconds for both implementations we have developed in this work. These experimental results are obtained by using as a benchmark the "1tuk" protein which contains up to 67 amino acids in a 3-D fashion. Moreover, ACO parameters include the following: 1000-independent ACO runs, $\alpha = 1$ and $\beta = 3$.

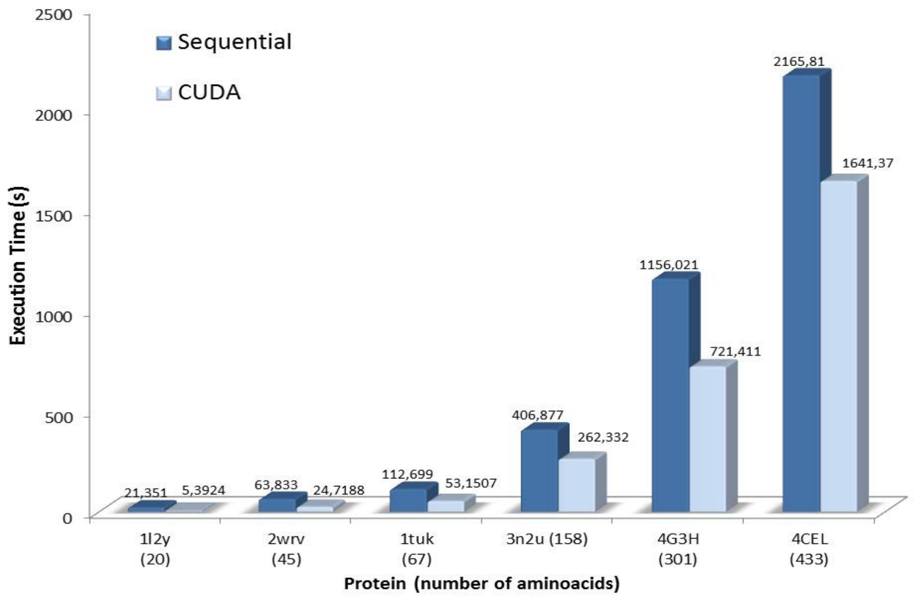


Fig. 2. Execution time in seconds for the execution of sequential and CUDA ACO implementations for different proteins that have different number of amino acids.

Figure 2 shows the execution time fixing the number of ants (up to 2048), and varying proteins in order to test the algorithm with several number of amino acids. It is noteworthy to point out that the parallel version is faster than the sequential counterpart version in all the proteins tested. In this case, when the number of amino acids is increased, the differences between sequential and parallel version get closer, this is for the same reason that previously was presented, since we established the association of one thread to one ant, our benefits become higher when the number of ants is increased, beating the sequential code by a wide margin.

5 Conclusions and outlook

Ant Colony Optimization (ACO) belongs to the family of population-based metaheuristics that has been successfully applied to many NP-complete problems. In this work, we present a parallel version of ACO algorithm as applied to the protein folding problem on Graphics Processing Units. We use a well-known coarse grained HP model that classifies amino acids into Hydrophobic (H) or Polar(P), attending to the attraction or the rejection that the amino acid present against water. We identify the natural parallelism of ACO; i.e. ants running in parallel to find out a solution with threads in the cuda programming model. Our experimental results leads performance gains up to 7x speedup factor compared to its sequential counterpart version.

The tupla ACO and protein folding on GPUs is still at a relatively early stage, and we acknowledge that we have tested a relatively simple variant of the algorithm and the protein model. But, with many other types of combinations still to be explored, this field seems to offer a promising and potentially fruitful area of research. Especially due to the nature of the problem, this is not a problem with a lot of data to compute, only 4 (in 2 dimensions) or 6 (in 3 dimensions) operations we can do simultaneously. However, some algorithmic improvements may be introduced to enhance performance. Among them, we may highlight to provide a data-parallelism design that takes advantage of vector-fashion execution in current processor architectures.

Acknowledgements

This work has been funded by grants from the Fundación Séneca of the Región of Murcia (18946/JLI/13) and by the Nils Coordinated Mobility under grant 012-ABEL-CM-2014A, in part financed by the European Regional Development Fund (ERDF). We also thank Nvidia for the hardware donation under GPU Research and Educational Center Program.

References

1. Rolf Backofen and Sebastian Will. A constraint-based approach to fast and exact structure prediction in three-dimensional protein models. *Constraints*, 11(1):5–30, 2006.

2. Bonnie Berger and Tom Leighton. Protein folding in the hydrophobic-hydrophilic (hp) model is np-complete. *Journal of Computational Biology*, 5(1):27–40, 1998.
3. Piero P Bonissone. Soft computing: the convergence of emerging reasoning technologies. *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, 1(1):6–18, 1997.
4. Thang N Bui and Gnanasekaran Sundarraj. An efficient genetic algorithm for predicting protein tertiary structures in the 2d hp model. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pages 385–392. ACM, 2005.
5. Ruay-Shiung Chang, Jih-Sheng Chang, and Po-Sheng Lin. An ant algorithm for balanced job scheduling in grids. *Future Generation Computer Systems*, 25(1):20–27, 2009.
6. Yumin Chen, Duoqian Miao, and Ruizhi Wang. A rough set approach to feature selection based on ant colony optimization. *Pattern Recognition Letters*, 31(3):226–233, 2010.
7. G Di Caro and M Dorigo. Ant colony optimization: a new meta-heuristic. In *Proceedings of the 1999 Congress on Evolutionary Computation*, 1999.
8. Ken A Dill, Sarina Bromberg, Kaizhi Yue, Klaus M Fiebig, David P Yee, Paul D Thomas, and Hue Sun Chan. Principles of protein folding–a perspective from simple exact models. *Protein science: a publication of the Protein Society*, 4(4):561, 1995.
9. Ken A Dill and Justin L MacCallum. The protein-folding problem, 50 years on. *Science*, 338(6110):1042–1046, 2012.
10. M Dorigo. St ützle t (2004) ant colony optimization. *Bradford Company, Scituate, MA, USA*, 2004.
11. Marco Dorigo, Mauro Birattari, and Thomas Stützle. Ant colony optimization. *Computational Intelligence Magazine, IEEE*, 1(4):28–39, 2006.
12. Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Ant system: optimization by a colony of cooperating agents. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 26(1):29–41, 1996.
13. Marco Dorigo and Thomas Stützle. Ant colony optimization: overview and recent advances. In *Handbook of metaheuristics*, pages 227–263. Springer, 2010.
14. Russ C Eberhart and James Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the sixth international symposium on micro machine and human science*, volume 1, pages 39–43. New York, NY, 1995.
15. MA Porta Garcia, Oscar Montiel, Oscar Castillo, Roberto Sepúlveda, and Patricia Melin. Path planning for autonomous mobile robot navigation with ant colony optimization and fuzzy cost function evaluation. *Applied Soft Computing*, 9(3):1102–1110, 2009.
16. JM García-Martínez, EM Garzón, JM Cecilia, H Pérez-Sánchez, and PM Ortigosa. An efficient approach for solving the hp protein folding problem based on uego. *Journal of Mathematical Chemistry*, 53(3):794–806, 2015.
17. Fred Glover and Gary A Kochenberger. *Handbook of metaheuristics.* Springer Science & Business Media, 2003.
18. David E Golberg. Genetic algorithms in search, optimization, and machine learning. *Addion wesley*, 1989, 1989.
19. Bwo-Ren Ke, Meng-Chieh Chen, and Chun-Liang Lin. Block-layout design using max–min ant system for saving energy on mass rapid transit systems. *Intelligent Transportation Systems, IEEE Transactions on*, 10(2):226–235, 2009.
20. James Kennedy, James F Kennedy, Russell C Eberhart, and Yuhui Shi. *Swarm intelligence.* Morgan Kaufmann, 2001.

21. Ivan Kondov. Protein structure prediction using distributed parallel particle swarm optimization. *Natural Computing*, 12(1):29–41, 2013.
22. Jingfa Liu, Gang Li, Jun Yu, and Yonglei Yao. Heuristic energy landscape paving for protein folding problem in the three-dimensional hp lattice model. *Computational biology and chemistry*, 38:17–26, 2012.
23. Paul I Pénzes and Alain J Martin. Energy-delay efficiency of vlsi computations. In *Proceedings of the 12th ACM Great Lakes symposium on VLSI*, pages 104–111. ACM, 2002.
24. A Schug and W Wenzel. An evolutionary strategy for all-atom folding of the 60-amino-acid bacterial ribosomal protein l20. *Biophysical journal*, 90(12):4273–4280, 2006.
25. Timo Strunk, Moritz Wolf, and Wolfgang Wenzel. Peptide structure prediction using distributed volunteer computing networks. *Journal of Mathematical Chemistry*, 50(2):421–428, 2012.
26. Thomas Stützle and Holger H Hoos. Max–min ant system. *Future generation computer systems*, 16(8):889–914, 2000.
27. José L Verdegay, Ronald R Yager, and Piero P Bonissone. On heuristics as a fundamental constituent of soft computing. *Fuzzy Sets and Systems*, 159(7):846–855, 2008.
28. Bin Yu, Zhong-Zhen Yang, and Baozhen Yao. An improved ant colony optimization for vehicle routing problem. *European journal of operational research*, 196(1):171–176, 2009.

Capítulo 3

Resultados

Este capítulo resume los resultados más relevantes obtenidos durante la elaboración de la tesis, así como las conclusiones derivadas de la misma y caminos futuros a explorar. También se proporcionan los datos relativos a la calidad de las publicaciones que componen esta tesis.

3.1 Resultados

En primer lugar, y por comenzar cronológicamente con los artículos incluidos en la tesis, se destacan los resultados obtenidos del artículo [19]. Este artículo es el que posibilita la primera toma de contacto con el *Ant Colony Optimization*, la cual sirve para conocer el algoritmo en profundidad y aprender cuáles son sus etapas más críticas que concentran la mayor carga computacional. Este conocimiento es fundamental para el desarrollo de la tesis, y además, sirve para contrastar la ejecución de las distintas versiones del algoritmo en diferentes plataformas, comparando entre plataformas de AMD, NVIDIA e INTEL. Se obtienen resultados que concluyen que la implementación que obtiene mejores resultados en los tests es la de CUDA, frente a la migrada a OpenCL y frente a la secuencial ofrecida por M.Dorigo y T.Stützle en [13].

Una vez analizado y diseñado la paralelización del algoritmo ACO en aceleradores, y demostrado que las GPUs de Nvidia es la alternativa más eficiente de las estudiadas, nos planteamos cómo escalar a un clúster heterogéneo basado en CPUs y GPUs de Nvidia para abordar problemas que requieran un ingente número de computaciones. En este artículo [27], se propone un escenario de supercomputación donde el clúster heterogéneo puede tener GPUs de diferentes capacidades computacionales (diferencias substanciales llegando hasta 4x). En este entorno, se plantean estrategias de balanceo de carga para analizar el impacto que puede alcanzar en el tiempo de ejecución y en la calidad de los resultados. Es importante destacar que esto no es únicamente un resultado teórico, y en el artículo se muestran los resultados que sustentan esta teoría, así como la implementación para llevar a cabo estas estrategias, basadas en MPI y OpenMP.

Si bien en [19] se manifiesta una inquietud por el consumo energétio, que se define como uno de los nuevos cuellos de botella existentes, con la inclusión de plataformas más orientadas al consumo que al rendimiento como la APU frente a los servidores de alta gama, es en el artículo [27] donde esta inquietud por el consumo energético llega a cotas más altas, realizando mediciones de consumo en la ejecución en clústeres heterogéneos. Estos resultados también se muestran en el artículo, realizando cambios en la frecuencia de reloj a la que trabajan las tarjetas medianto el mecanimso hardware GPU Boost$^{\text{TM}}$, *(sólo para tarjetas NVIDIA a partir de la K40 de la familia Kepler, motivo por el que nos centramos en clústeres basados en NVIDIA)*, dichos cambios de frecuencia se analizan para contrastar el consumo energético de las distintas tarjetas a las diferentes frecuencias soportadas.

Se puede considerar que una de las aportaciones de esta tesis es el planteamiento de diferentes estrategias para las ejecuciones en clústeres heterogéneos, (*reinicios* y *búsqueda_intensiva*), con las que se mantienen todos los recursos computacionales trabajando en su totalidad, y con las que se mantienen o incluso mejoran la calidad final de las soluciones aportadas por los algoritmos, tal y como se muestra en el artículo.

El último de los artículos de los que consta esta tesis [28], muestra la conciencia que tienen los investigadores en el uso de recursos computacionales de alto rendimiento en sus investigaciones, usándolos en problemas reales y complejos, no sólo para *benchmarks* sintéticos. Las conclusiones de esta investigación, es que sólo aquellos investigadores con un perfil tecnológico más marcado, son los que muestran mayor inquietud en los desarrollos paralelos y en el uso de plataformas de alto rendimiento, a pesar de estar más que contrastado los beneficios que estos entornos ofrecen. Por otro lado, este hecho abre las puertas a futuros desarrollos en este campo para cualquier otro problema diferente al planteado en el artículo del plegamiento de proteínas.

Fuera de lo que son los artículos del compendio, como ya se destacó con anterioridad, también se incluye una comunicación en un congreso internacional [29], y la aportación con la que este artículo contribuye es la de contrastar todos los conocimientos teóricos adquiridos sobre el algoritmo, a un problema real como el plegado de proteínas. Concretamente, esta motivación es plasmada en una implementación en CUDA del algoritmo ACO, aplicada al plegado de proteínas por medio de un modelo simplificado denominado H-P y en el que se consiguen aceleraciones de hasta un factor de 7x.

3.2 Conclusión y vías futuras

En esta sección del documento se presentan las conclusiones de esta tesis doctoral. Es importante tener en cuenta que el desarrollo de una tesis doctoral no debe ser un fin en sí mismo, sino una puerta que se abre al área de la investigación, y con la que se pueden sentar las bases de conocimiento necesario para iniciar una carrera investigadora fructífera, por lo que, igual de importante deben ser los resultados conseguidos, que las conclusiones y vías futuras del trabajo para seguir investigando. La primera toma de contacto con el algoritmo *Ant Colony Optimization* en [19] se considera fundamental, no sólo para el desarrollo de la tesis, sino como la obligada adquisición del conocimiento necesario por parte del doctorando del algoritmo en profundidad, que debe acompañarle durante el resto de su carrera investigadora.

Uno de los puntos más motivantes para futuros desarrollos, es que los resultados obtenidos son alentadores. Un claro ejemplo son las estrategias de ejecución planteadas en [27], las cuales son portables a cualquier ejecución que se plantee en entornos de ejecución similares. De hecho, una de las posibles vías de ampliación para continuar las investigaciones desarrolladas durante esta tesis, es trasladar estas estrategias a la implementación paralela de ACO sobre el modelo H-P realizada en [29]. Si los resultados son como los conseguidos en [27], podremos mejorar tanto en rendimiento como en calidad en los resultados. Siguiendo en esta vía, aprovechando las nuevas implementaciones de las que disponemos tras la realización de esta tesis, se plantea la posibilidad de escalar los algortimos diseñados a clústeres heterogéneos no sólo de GPUs de NVIDIA, sino a clústeres que cuentan con tarjetas NVIDIA y AMD.

A continuación se muestra una serie de líneas de investigación que se encuentran en vías de desarrollo, manteniendo una línea continuísta con la presentada en este trabajo, concretamente con el ***objetivo 1:****Análisis, diseño y optimización del ACO en GPUs*, estas líneas son:

Se plantea la optimización algorítmica del ACO en sus fases más costosas, la construcción del tour y la actualización de feromonas.

- Realizar un estudio de la fase de *actualización de feromona*, en la que se realiza tanto el depósito como la evaporización de feromona, haciéndose prioritario el uso de instrucciones atómicas para mantener el rigor en las actualizaciones debido a las condiciones de carrera que pueden plantearse en el patrón de acceso. Dicho estudio se plantea en base a diferentes plataformas, en las cuales las unidades atómicas son implementadas con distinto acierto, y con latencias diferentes.

- Otra de las vías futuras que se encuentran en un avanzado estado de desarrollo es una nueva propuesta para la fase de selección de la siguiente ciudad a visitar por una hormiga, *Selection Procedure*. Tradicionalmente, el método de selección utilizado ha sido el de la ruleta, *Roulette Wheel* o *Método de MonteCarlo*, que reproduce la información heurística que se mantiene en la estrucutra *choice_info*, y ampliamente utilizado en muchos de los algoritmos evolutivos. El primer acercamiento en la literatura para la paralelización de este método puramente secuencial es en [10], donde se propone un método denominado *I-Roulette* que es totalmente paralelo. La nueva propuesta que se ha desarrollado, es un algoritmo al que se le llama *SS-Roulette*, en mención a los patrones Scan-Stencil, que son los que usa en su implementación, y que, además de ser del orden de 19x-45x veces más rápido, reproduce la información heurística contenida en *choice_info* de una manera mucho más exacta.

- Tambien se está desarrollando un estudio en cuanto a la granularidad en el diseño del algoritmo. Esto es, si obtenemos mejores resultados mapeando en el algoritmo una hormiga a un bloque, o una hormiga a un warp, *(Unidad de planifiación en cuda)*, o, lo que es incluso más interesante, intentar diseñar un cambio en el tamaño del warp mediante las nuevas funciones que nos ofrece CUDA 8.0, para ver si un tamaño mayor de warp al que está diseñado en la arquitectura de NVIDIA puede ofrecernos alguna ventaja en este tipo de algoritmos.

Otra de las vías futuras para esta trabajo es la implementación del algoritmo para otros procesadores vectoriales, dado que todas las aportaciones que se realizan son en el sentido de vectorizar el algoritmo para aprovechar al máximo los recursos *hardware* de estas nuevas plataformas. Concretamente, se considera el nuevo procesador Intel Xeon Phi como un candidato a ejecutar el algoritmo y contrastar los rendimientos que se obtienen en esta plataforma. De igual manera, existen actualmente entornos virtualizados de GPUs, tales como rCUDA [16], con los que se pretende paliar tanto el consumo energético como reducir el coste para estos entornos heterogéneos de altas prestaciones, puede resultar muy interesante comparar los resultados de rCUDA, con otros clústeres como con los que se han trabajado en esta tesis.

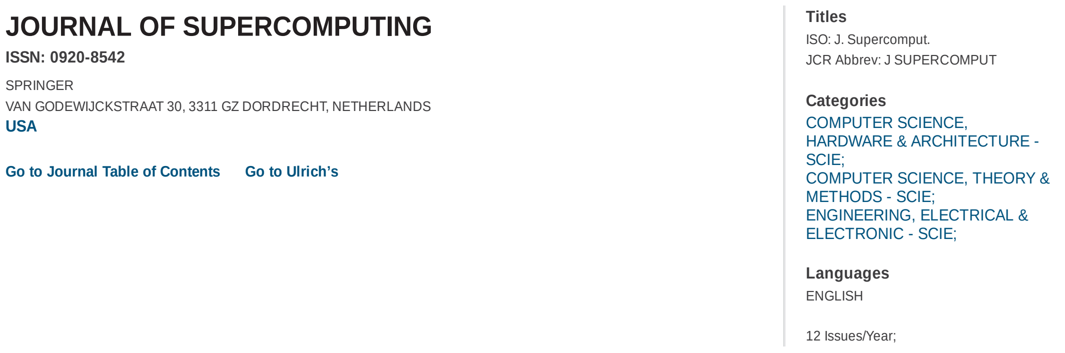
Para plantear nuevos retos en la aplicación real del algoritmo al campo científico, una vez que se estudia un problema como el del plegamiento de proteínas, se presentan varias posibilidades de actuación, existen multitud de modelos de entre los cuales, se ha elegido el modelo H-P, pero se pueden trasladar las ideas a otras implementaciones mediante otro tipo de modelos que simulen de manera más precisa el comportamiento del plegado de proteínas. Otro de los objetivos implícitos de una tesis doctoral, aunque no en su desarrollo, sino como un fin último, es la divulgación del conocimiento científico al público en general, quizá por el marcado perfil docente del doctorando, este punto se considera de especial interés, y se plantea el objetivo de la divulgación del área científico que ha formado parte del estudio en el trabajo, el plegado de proteínas, que sigue siendo un reto en la actualidad para la biología molecular, y con el que se tiene la certeza de poder conseguir logros de relevancia acercando el problema a la sociedad y a la comunidad docente, por ejemplo por medio de la implementación de una aplicación móvil que, partiendo de una gamificación en este área, acerque al público a un mayor conocimiento del problema en cuestión. Este es un punto en el que ya se han realizado los primeros pasos, participando en concurrencia de proyectos nacionales para obtener los recursos necesarios para su desarrollo, partiendo de un prototipo funcional con un nivel de madurez tecnológica TRL 6.

3.3 Datos relativos a la calidad de las publicaciones

En este apartado se aportan todos los índices principales de calidad de las revistas en las que han sido publicados los artículos que componen el compendio de la presente tesis doctoral.

3.3.1 Comparative evaluation of platforms for parallel Ant Colony Optimization - The Journal of Supercomputing

Con las siguientes figuras se aportan los datos relativos a la calidad de la revista *The journal of Supercomputing*, en la que fue publicado el artículo *Comparative evaluation of platforms for parallel Ant Colony Optimization.*, primero de los artículos que conforman la presente tesis doctoral.



JOURNAL OF SUPERCOMPUTING

ISSN: 0920-8542

SPRINGER

VAN GODEWIJCKSTRAAT 30, 3311 GZ DORDRECHT, NETHERLANDS

USA

Go to Journal Table of Contents **Go to Ulrich's**

Titles

ISO: J. Supercomput.

JCR Abbrev: J SUPERCOMPUT

Categories

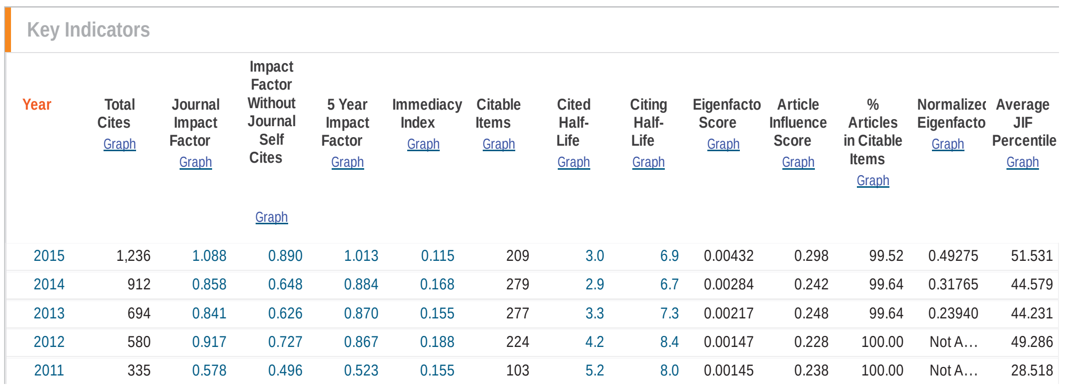
COMPUTER SCIENCE, HARDWARE & ARCHITECTURE - SCIE;
COMPUTER SCIENCE, THEORY & METHODS - SCIE;
ENGINEERING, ELECTRICAL & ELECTRONIC - SCIE;

Languages

ENGLISH

12 Issues/Year;

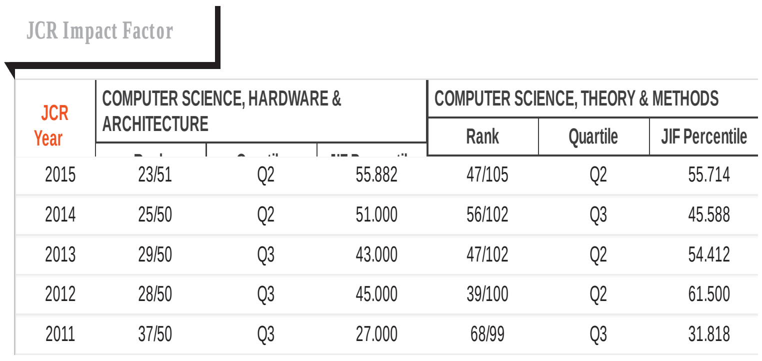
Figura 3.1: Título y datos de la revista de publicación.



Key Indicators

Year	Total Cites Graph	Journal Impact Factor Graph	Impact Factor Without Journal Self Cites Graph	5 Year Impact Factor Graph	Immediacy Index Graph	Citable Items Graph	Cited Half-Life Graph	Citing Half-Life Graph	Eigenfacto Score Graph	Article Influence Score Graph	% Articles in Citable Items Graph	Normalizec Eigenfacto Graph	Average JIF Percentile Graph
2015	1,236	1.088	0.890	1.013	0.115	209	3.0	6.9	0.00432	0.298	99.52	0.49275	51.531
2014	912	0.858	0.648	0.884	0.168	279	2.9	6.7	0.00284	0.242	99.64	0.31765	44.579
2013	694	0.841	0.626	0.870	0.155	277	3.3	7.3	0.00217	0.248	99.64	0.23940	44.231
2012	580	0.917	0.727	0.867	0.188	224	4.2	8.4	0.00147	0.228	100.00	Not A…	49.286
2011	335	0.578	0.496	0.523	0.155	103	5.2	8.0	0.00145	0.238	100.00	Not A…	28.518

Figura 3.2: Indicadores clave de los últimos cinco años.



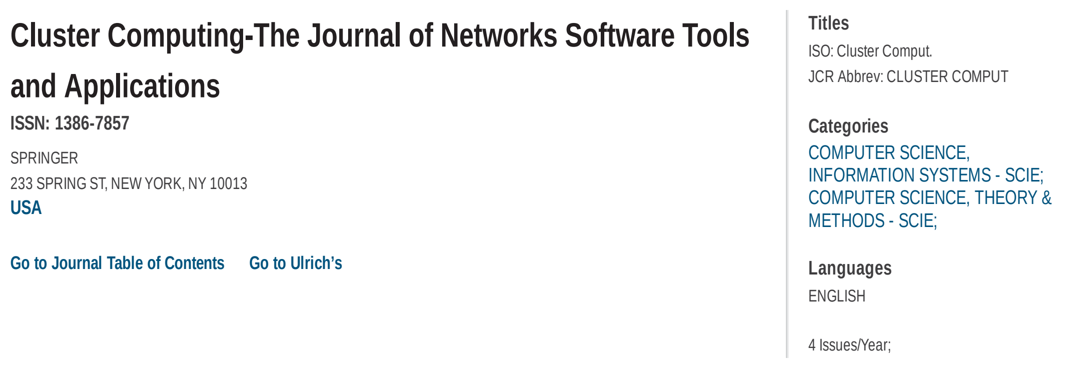
JCR Impact Factor

JCR Year	COMPUTER SCIENCE, HARDWARE & ARCHITECTURE			COMPUTER SCIENCE, THEORY & METHODS		
	[illegible]	[illegible]	[illegible]	Rank	Quartile	JIF Percentile
2015	23/51	Q2	55.882	47/105	Q2	55.714
2014	25/50	Q2	51.000	56/102	Q3	45.588
2013	29/50	Q3	43.000	47/102	Q2	54.412
2012	28/50	Q3	45.000	39/100	Q2	61.500
2011	37/50	Q3	27.000	68/99	Q3	31.818

Figura 3.3: Factor de impacto de los últimos cinco años

3.3.2 Dynamic load balancing on heterogeneous clusters for parallel ant colony optimization - Cluster Computing

En las figuras 3.4-3.6 se reflejan los datos relativos a la calidad de la revista *Cluster Computing-The Journal of Networks Software Tools and Applications*, en la que fue publicado el artículo *Dynamic load balancing on heterogeneous clusters for parallel ant colony optimization.*, otro de los artículos que soportan este trabajo.



Cluster Computing-The Journal of Networks Software Tools and Applications

ISSN: 1386-7857

SPRINGER
233 SPRING ST, NEW YORK, NY 10013
USA

Go to Journal Table of Contents   Go to Ulrich's

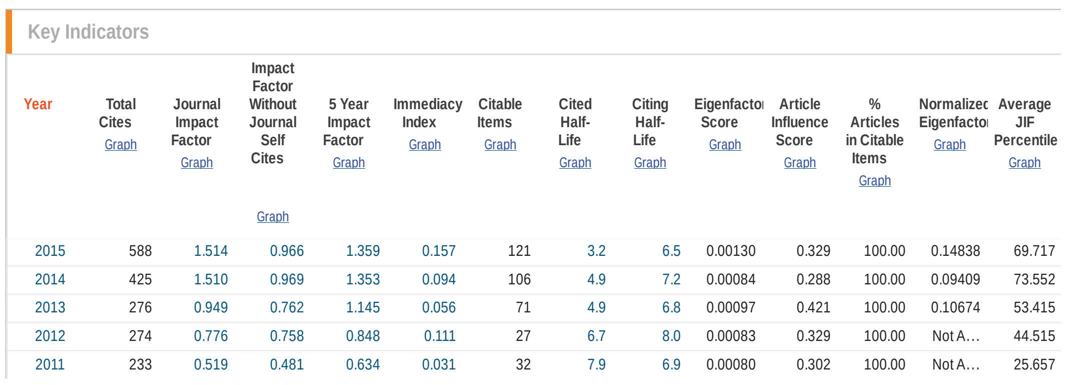
Titles
ISO: Cluster Comput.
JCR Abbrev: CLUSTER COMPUT

Categories
COMPUTER SCIENCE, INFORMATION SYSTEMS - SCIE;
COMPUTER SCIENCE, THEORY & METHODS - SCIE;

Languages
ENGLISH

4 Issues/Year;

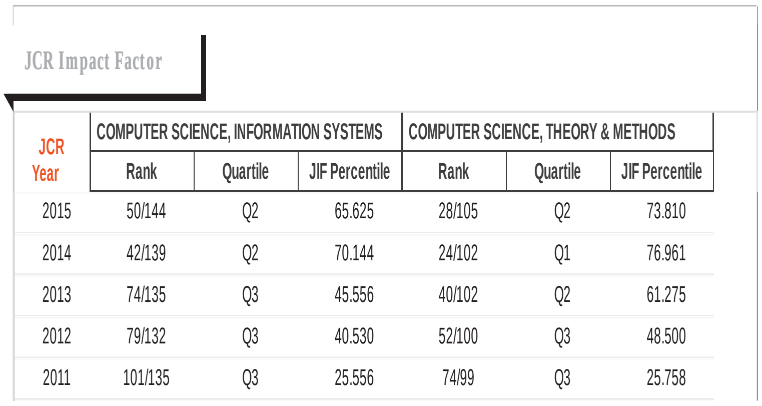
Figura 3.4: Título y datos de la revista de publicación.



Key Indicators

Year	Total Cites Graph	Journal Impact Factor Graph	Impact Factor Without Journal Self Cites Graph	5 Year Impact Factor Graph	Immediacy Index Graph	Citable Items Graph	Cited Half-Life Graph	Citing Half-Life Graph	Eigenfacto Score Graph	Article Influence Score Graph	% Articles in Citable Items Graph	Normalizec Eigenfacto Graph	Average JIF Percentile Graph
2015	588	1.514	0.966	1.359	0.157	121	3.2	6.5	0.00130	0.329	100.00	0.14838	69.717
2014	425	1.510	0.969	1.353	0.094	106	4.9	7.2	0.00084	0.288	100.00	0.09409	73.552
2013	276	0.949	0.762	1.145	0.056	71	4.9	6.8	0.00097	0.421	100.00	0.10674	53.415
2012	274	0.776	0.758	0.848	0.111	27	6.7	8.0	0.00083	0.329	100.00	Not A...	44.515
2011	233	0.519	0.481	0.634	0.031	32	7.9	6.9	0.00080	0.302	100.00	Not A...	25.657

Figura 3.5: Indicadores clave de los últimos cinco años.

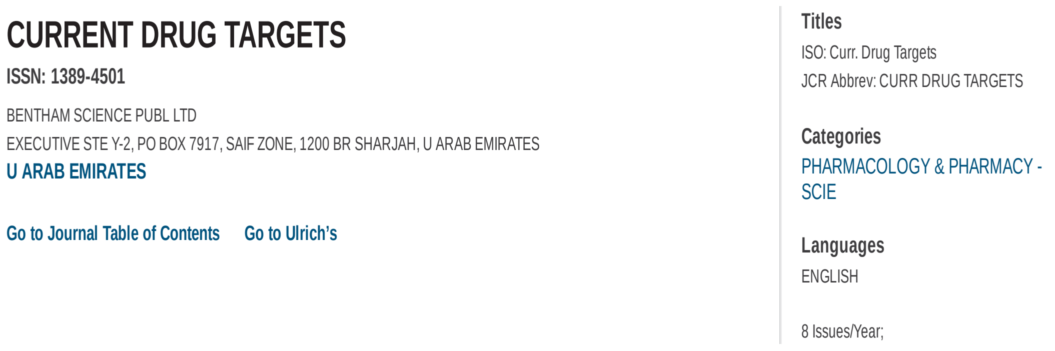

JCR Impact Factor

JCR Year	COMPUTER SCIENCE, INFORMATION SYSTEMS			COMPUTER SCIENCE, THEORY & METHODS		
	Rank	Quartile	JIF Percentile	Rank	Quartile	JIF Percentile
2015	50/144	Q2	65.625	28/105	Q2	73.810
2014	42/139	Q2	70.144	24/102	Q1	76.961
2013	74/135	Q3	45.556	40/102	Q2	61.275
2012	79/132	Q3	40.530	52/100	Q3	48.500
2011	101/135	Q3	25.556	74/99	Q3	25.758

Figura 3.6: Factor de impacto de los últimos cinco años

3.3.3 Soft Computing Techniques for the Protein Folding Problem on High Performance Computing Architectures - Current Drug Targets

En las figuras 3.7-3.9 se muestran los datos relativos a la calidad de la revista *Current Drug Targets*, en la que fue publicado el artículo *Soft Computing Techniques for the Protein Folding Problem on High Performance Computing Architectures.*, el último de los artículos que conforman este compendio.



CURRENT DRUG TARGETS

ISSN: 1389-4501

BENTHAM SCIENCE PUBL LTD

EXECUTIVE STE Y-2, PO BOX 7917, SAIF ZONE, 1200 BR SHARJAH, U ARAB EMIRATES

U ARAB EMIRATES

Go to Journal Table of Contents   Go to Ulrich's

Titles

ISO: Curr. Drug Targets

JCR Abbrev: CURR DRUG TARGETS

Categories

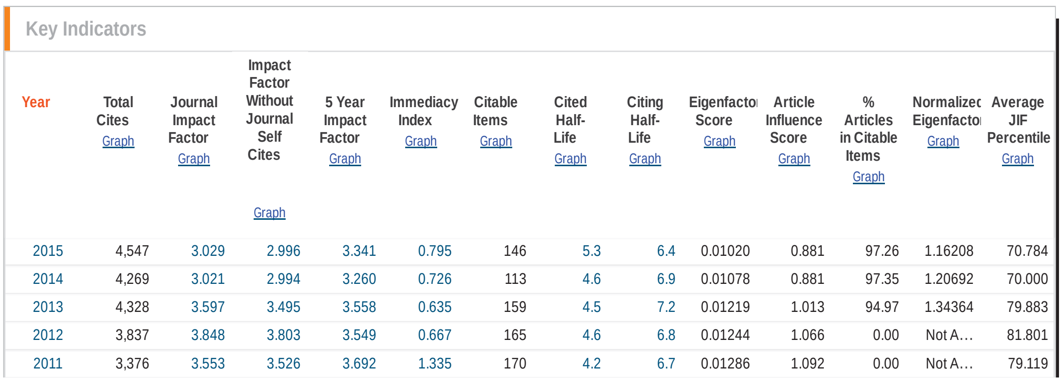
PHARMACOLOGY & PHARMACY - SCIE

Languages

ENGLISH

8 Issues/Year;

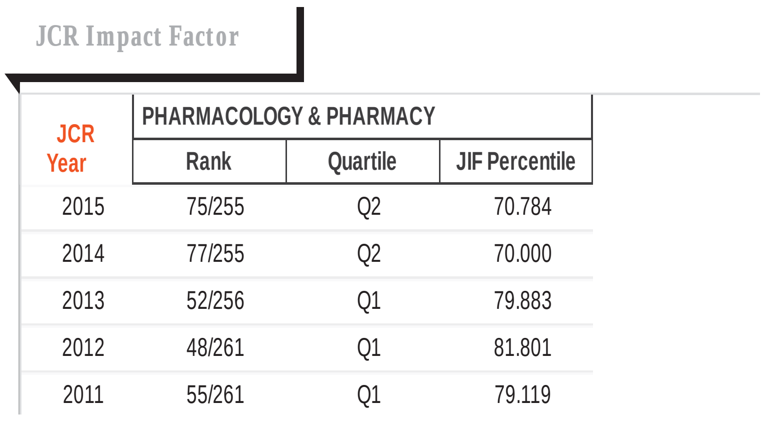
Figura 3.7: Título y datos de la revista de publicación.



Key Indicators

Year	Total Cites Graph	Journal Impact Factor Graph	Impact Factor Without Journal Self Cites Graph	5 Year Impact Factor Graph	Immediacy Index Graph	Citable Items Graph	Cited Half-Life Graph	Citing Half-Life Graph	Eigenfactor Score Graph	Article Influence Score Graph	% Articles in Citable Items Graph	Normalized Eigenfactor Graph	Average JIF Percentile Graph
2015	4,547	3.029	2.996	3.341	0.795	146	5.3	6.4	0.01020	0.881	97.26	1.16208	70.784
2014	4,269	3.021	2.994	3.260	0.726	113	4.6	6.9	0.01078	0.881	97.35	1.20692	70.000
2013	4,328	3.597	3.495	3.558	0.635	159	4.5	7.2	0.01219	1.013	94.97	1.34364	79.883
2012	3,837	3.848	3.803	3.549	0.667	165	4.6	6.8	0.01244	1.066	0.00	Not A…	81.801
2011	3,376	3.553	3.526	3.692	1.335	170	4.2	6.7	0.01286	1.092	0.00	Not A…	79.119

Figura 3.8: Indicadores clave de los últimos cinco años.



JCR Impact Factor

JCR Year	PHARMACOLOGY & PHARMACY		
	Rank	Quartile	JIF Percentile
2015	75/255	Q2	70.784
2014	77/255	Q2	70.000
2013	52/256	Q1	79.883
2012	48/261	Q1	81.801
2011	55/261	Q1	79.119

Figura 3.9: Factor de impacto de los últimos cinco años

Capítulo 4

Bibliografía

4.1 References

[1] Top 500 supercomputer site, [last accesed 12 Julio 2016]. `http://www.top500.org/`.

[2] HSA Foundation, [last accesed September, 29th 2016]. `http://www.hsafoundation.com/`.

[3] González Álvarez and David Lesmes. Metaheurísticas, optimización multiobjetivo y paralelismo para descubrir motifs en secuencias de adn. 2013.

[4] Ahmad Taher Azar and Sundarapandian Vaidyanathan. *Chaos modeling and control systems design*. Springer, 2015.

[5] Bonnie Berger and Tom Leighton. Protein folding in the hydrophobic-hydrophilic (hp) model is np-complete. *Journal of Computational Biology*, 5(1):27–40, 1998.

[6] Mauro Birattari and Marco Dorigo. The problem of tuning metaheuristics as seen from a machine learning perspective. 2004.

[7] Christian Blum. Ant colony optimization: Introduction and recent trends. *Physics of Life reviews*, 2(4):353–373, 2005.

[8] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, 35(3):268–308, 2003.

[9] Piero P Bonissone. Soft computing: the convergence of emerging reasoning technologies. *Soft computing*, 1(1):6–18, 1997.

[10] José M Cecilia, José M García, Andy Nisbet, Martyn Amos, and Manuel Ujaldón. Enhancing data parallelism for ant colony optimization on gpus. *Journal of Parallel and Distributed Computing*, 73(1):42–51, 2013.

[11] Hue Sun Chan and Ken A Dill. The protein folding problem. *Physics today*, 46(2):24–32, 1993.

[12] Daniel Dabbelt, Colin Schmidt, Eric Love, Howard Mao, Sagar Karandikar, and Krste Asanovic. Vector processors for energy-efficient embedded systems. In *Proceedings of the Third ACM International Workshop on Many-core Embedded Systems*, pages 10–16. ACM, 2016.

[13] M Dorigo and MDT Stützle. Ant colony optimization. bradford bks, 2004.

[14] Marco Dorigo. Optimization, learning and natural algorithms. *Ph. D. Thesis, Politecnico di Milano, Italy*, 1992.

[15] Marco Dorigo and Luca Maria Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation*, 1(1):53–66, 1997.

[16] José Duato, Antonio J Pena, Federico Silla, Rafael Mayo, and Enrique S Quintana-Ortí. rcuda: Reducing the number of gpu-based accelerators in high performance clusters. In *High Performance Computing and Simulation (HPCS), 2010 International Conference on*, pages 224–231. IEEE, 2010.

[17] Fred W Glover and Gary A Kochenberger. *Handbook of metaheuristics*, volume 57. Springer Science & Business Media, 2006.

[18] Stanley E Griffis, John E Bell, and David J Closs. Metaheuristics in logistics and supply chain management. *Journal of Business Logistics*, 33(2):90–106, 2012.

[19] Ginés D Guerrero, José M Cecilia, Antonio Llanes, José M García, Martyn Amos, and Manuel Ujaldón. Comparative evaluation of platforms for parallel ant colony optimization. *The Journal of Supercomputing*, 69(1):318–329, 2014.

[20] John F Hall and Anil K Chopra. Dynamic analysis of arch dams including hydrodynamic effects. *Journal of Engineering Mechanics*, 109(1):149–167, 1983.

[21] José F Herbert-Acero, Jaime Martínez-Lauranchet, Oliver Probst, Santos Méndez-Díaz, Krystel K Castillo-Villar, Manuel Valenzuela-Rendón, and Pierre-Elouan Réthoré. A hybrid metaheuristic-based approach for the aerodynamic optimization of small hybrid wind turbine rotors. *Mathematical Problems in Engineering*, 2014, 2014.

[22] Jörg Homberger and Hermann Gehring. Two evolutionary metaheuristics for the vehicle routing problem with time windows. *INFOR: Information Systems and Operational Research*, 37(3):297–318, 1999.

[23] Stephen W Keckler, William J Dally, Brucek Khailany, Michael Garland, and David Glasco. Gpus and the future of parallel computing. *IEEE Micro*, 31(5):7–17, 2011.

[24] Albert YS Lam and Victor OK Li. Chemical-reaction-inspired metaheuristic for optimization. *IEEE Transactions on Evolutionary Computation*, 14(3):381–399, 2010.

[25] E Scott Larsen and David McAllister. Fast matrix multiplies using graphics hardware. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing*, pages 55–55. ACM, 2001.

[26] X Li. Discussion on soft computing at flins'96. *International Journal of Intelligent Systems*, 13:287–300, 1998.

[27] Antonio Llanes, José M Cecilia, Antonia Sánchez, José M García, Martyn Amos, and Manuel Ujaldón. Dynamic load balancing on heterogeneous clusters for parallel ant colony optimization. *Cluster Computing*, 19(1):1–11, 2016.

[28] Antonio Llanes, A Muñoz, A Bueno-Crespo, T García-Valverde, A Sánchez, F Arcas-Túnez, H Pérez-Sánchez, and JM Cecilia. Soft computing techiniques for the protein folding problem on high performance computing architectures. *Current drug targets*, 2016.

[29] Antonio Llanes, Carlos Vélez, Antonia M Sánchez, Horacio Pérez-Sánchez, and José M Cecilia. Parallel ant colony optimization for the hp protein folding problem. In *International Conference on Bioinformatics and Biomedical Engineering*, pages 615–626. Springer, 2016.

[30] Mark Lundstrom. Moore's law forever? *Science*, 299(5604):210–211, 2003.

[31] Andrew McAfee, Erik Brynjolfsson, Thomas H Davenport, DJ Patil, and Dominic Barton. Big data. *The management revolution. Harvard Bus Rev*, 90(10):61–67, 2012.

[32] Gordon Moore. Moore's law. *Electronics Magazine*, 38(8), 1965.

[33] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda. *Queue*, 6(2):40–53, 2008.

[34] NVIDIA. `www.nvidia.es`, 18 de Julio de 2016.

[35] Ibrahim H Osman and Gilbert Laporte. Metaheuristics: A bibliography. *Annals of Operations research*, 63(5):511–623, 1996.

[36] Rafael S Parpinelli, Heitor S Lopes, and Alex Alves Freitas. Data mining with an ant colony optimization algorithm. *IEEE transactions on evolutionary computation*, 6(4):321–332, 2002.

[37] Maria Rodriguez-Fernandez, Jose A Egea, and Julio R Banga. Novel metaheuristic for parameter estimation in nonlinear dynamic biological systems. *BMC bioinformatics*, 7(1):1, 2006.

[38] David E Shaw, Paul Maragakis, Kresten Lindorff-Larsen, Stefano Piana, Ron O Dror, Michael P Eastwood, Joseph A Bank, John M Jumper, John K Salmon, Yibing Shan, et al. Atomic-level characterization of the structural dynamics of proteins. *Science*, 330(6002):341–346, 2010.

[39] Alena Shmygelska and Holger H Hoos. An ant colony optimisation algorithm for the 2d and 3d hydrophobic polar protein folding problem. *BMC bioinformatics*, 6(1):1, 2005.

[40] Thomas Stützle and Holger Hoos. Improvements on the ant-system: Introducing the max-min ant system. In *Artificial Neural Nets and Genetic Algorithms*, pages 245–249. Springer, 1998.

[41] El-Ghazali Talbi. *Metaheuristics: from design to implementation*, volume 74. John Wiley & Sons, 2009.

[42] José L Verdegay, Ronald R Yager, and Piero P Bonissone. On heuristics as a fundamental constituent of soft computing. *Fuzzy sets and systems*, 159(7):846–855, 2008.

[43] Xin-She Yang, Mehmet Karamanoglu, and Simon Fong. Bat algorithm for topology optimization in microelectronic applications. In *The First International Conference on Future Generation Communication Technologies*, pages 150–155. IEEE, 2012.

[44] Lotfi A Zadeh. Soft computing and fuzzy logic. *IEEE software*, 11(6):48, 1994.

[45] Yan Zhao, Liping Chen, Gang Xie, Jianjun Zhao, and Jianwan Ding. Gpu implementation of a cellular genetic algorithm for scheduling dependent tasks of physical system simulation programs. *Journal of Combinatorial Optimization*, pages 1–25, 2016.