UCAM
UNIVERSIDAD CATÓLICA DE MURCIA

ESCUELA INTERNACIONAL DE DOCTORADO

Programa de Doctorado en Tecnologías de la Computación e Ingeniería Ambiental

TESIS DOCTORAL

Estrategias de paralelización para la optimización de métodos computacionales en el descubrimiento de nuevos fármacos

Autor:
D. Baldomero Imbernón Tudela

Directores:
Dr. D. José María Cecilia Canales
Dr. D. Horacio Emilio Pérez Sánchez
Dr. D. Domingo Giménez Cánovas

MURCIA, DICIEMBRE 2017

UCAM
UNIVERSIDAD CATÓLICA DE MURCIA

ESCUELA INTERNACIONAL DE DOCTORADO

Programa de Doctorado en Tecnologías de la Computación e Ingeniería Ambiental

TESIS DOCTORAL

Estrategias de paralelización para la optimización de métodos computacionales en el descubrimiento de nuevos fármacos

Autor:
D. Baldomero Imbernón Tudela

Directores:
Dr. D. José María Cecilia Canales
Dr. D. Horacio Emilio Pérez Sánchez
Dr. D. Domingo Giménez Cánovas

MURCIA, DICIEMBRE 2017

AUTORIZACION DE LOS DIRECTORES DE LA TESIS DOCTORAL PARA SU PRESENTACIÓN Y DEFENSA

El Dr. D. José María Cecilia Canales, el Dr. D. Horacio Emilio Pérez Sánchez y el Dr. D. Domingo Giménez Cánovas como DIRECTORES[1] de la Tesis Doctoral titulada "Estrategias de paralelización para la optimización de métodos computacionales en el descubrimiento de nuevos fármacos y su aplicación a problemas reales", realizada por D. Baldomero Imbernón Tudela en la Escuela Internacional de Doctorado (EIDUCAM), autorizan su presentación a trámite en su modalidad de compendio dado que reúne las condiciones necesarias para su defensa.

Lo que firmamos, para dar cumplimiento a los Reales Decretos 99/2011, 1393/2007, 56/2005 y 778/98, en Murcia a 30 de Noviembre de 2017.

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

A Isa y Ángel José, mi mujer y mi hijo con los que comparto mi vida
A mis padres, por darme todo, sin nada a cambio

Agradecimientos

Cuando terminas de redactar un trabajo como este, comienzas a darte cuenta de las personas que han aportado su esfuerzo y dedicación, tanto académica como personal, para ayudar a conseguir este logro. A todos ellos, les dedico estas palabras de agradecimiento, y les reservo un hueco privilegiado en esta tesis doctoral para poder demostrárselo.

Comenzar dando las gracias a mi esposa, Isa, que decidió compartir su vida conmigo hace más de cinco años, y que le estaré eternamente agradecido. A mi hijo Ángel José, mi pequeño, que con sus ganas de vivir, llena de alegría mi vida. A mis padres, que con su esfuerzo diario me dieron una educación y unos recursos para poder salir adelante. A estos tres pilares de mi vida les dedico esta tesis doctoral.

Mis tres directores han jugado un papel fundamental en el desarrollo de esta tesis doctoral. Su gran experiencia investigadora y su tutorización continua, sin lugar a dudas han sido determinantes para realizar este trabajo.

- A Domingo, que con su gran experiencia investigadora y excelentes conocimientos de algorítmica, me ha guiado a plantear la fundamentación de esta tesis, haciendo mucho más fácil el resto del camino.

- A Horacio, que con su rigor científico, dedicación y experiencia, no me han hecho desviarme del objetivo final, siguiendo una línea de investigación que me ha permitido conseguir los objetivos propuestos.

- A Chema, que desde la finalización de mi Grado hace más de cinco años, ha confiado en mí y ha seguido apoyándome y asesorándome hasta llegar a este punto. Ha tenido una paciencia infinita conmigo, realizando grandes esfuerzos en revisar y perfeccionar trabajos y publicaciones, que sin su aportación y empuje, no hubiesen sido realidad. Es muy difícil expresar con palabras mi agradecimiento hacia su persona, que ya lo considero una presencia fundamental en mi día a día en la Universidad.

Desde que me incorporé al Departamento del Grado de Informática, fui ganando compañeros, que se han convertido en amigos. A todos ellos, mi agradecimiento por el apoyo que he recibido desde el principio y por todas sus aportaciones.

Comenzar por Luz, que con ella es mucho más fácil abordar el día a día, escuchando y ofreciendo soluciones para todo. A Miguel Ángel, mi nuevo compañero de mesa, que aporta experiencia y equilibrio en muchos ámbitos de la institución, cualidades que admiro profundamente. A Paco Arcas, que transmite seguridad y sinceridad, dos cualidades fundamentales en docencia y en la vida misma. A Muñoz, un todocamino, que la investigación y las ganas de trabajar le transforman en un *big data* de la vida. A José Luis, que con su profesionalidad y templanza, aporta seguridad para abordar junto a él cualquier tarea. A Mada, que me aporta confianza a la hora de pedir consejo, siendo

para mí un ejemplo de compromiso con la docencia y dedicación al alumno. A Bueno, que está dispuesto para abordar cualquier proyecto, aportando una versatilidad difícil de encontrar en un investigador, junto a una manera de ser que te hace siempre ir a sus reuniones con una sonrisa. A Onia, incansable trabajadora por y para la titulación, asumiendo responsabilidades que por conocimiento de la Universidad, sería muy difícil asumir por otra persona. A Raquel, primera compañera de mesa, y a la que debo muchos consejos del hacer diario en el departamento, siendo un ejemplo de constancia con el alumno y con la investigación. A Jesús, cuyos aportes, datos históricos, y consejos sobre del día a día en la Universidad, hacen sentirte seguro en colaborar en cualquier proyecto que forme parte. Al Dr. Llanes, una mano amiga que tengo a mi espalda, que siempre está ahí para cualquier balcón, cualquier código, o consejo que necesite. A todos vosotros, muchas gracias por todo.

Quiero dedicar estas últimas líneas a Belén, nuestra recién nombrada vicerrectora, y además, profesora y directora del grado de Informática. Toda esta responsabilidad que ha ido adquiriendo con los años, la convierten en un espejo para cualquier docente que quiera ver en una misma persona, la misión y valores de esta Universidad. Además, agradecerle de todo corazón la confianza que ha depositado en mí, al darme la oportunidad de formar parte del grupo de profesores del Grado. Lo recordaré siempre.

Índice

1 Introducción **1**

1.1 La supercomputación al servicio del descubrimiento de fármacos 2

1.2 Métodos computacionales actuales para el descubrimiento de fármacos . 3

1.3 Esquemas metaheurísticos parametrizados para la optimización de métodos de cribado virtual 4

1.4 Objetivos 6

1.5 Estructura del documento de Tesis 7

2 Artículos que componen la tesis doctoral **9**

2.1 Fundamentación de la tesis doctoral 9

2.2 METADOCK: A parallel metaheuristic schema for virtual screening methods 11

2.3 Exploiting multilevel parallelism on a many-core system for the application of hyperheuristics to a molecular docking problem 27

2.4 Enhancing large-scale docking simulation on heterogeneous systems: an MPI vs rCUDA study 40

3 Conclusiones y vías futuras **53**

3.1 Conclusiones 53

3.2 Vías futuras 55

4 Publicaciones, colaboraciones y calidad de las revistas **57**

4.1 Colaboraciones 60

4.2 Otras publicaciones 61

4.3 Datos relativos a la calidad de las publicaciones 62

4.3.1 METADOCK: A parallel metaheuristic schema for virtual screening methods - The International Journal of High Performance Computing Application 63

4.3.2 Exploiting multilevel parallelism on a many-core system for the application of hyperheuristics to a molecular docking problem - The Journal of Supercomputing 64

4.3.3 Enhancing large-scale docking simulation on heterogeneous systems: an MPI vs rCUDA study - Future Generation Computer System 65

Bibliografía **67**

Capítulo 1

Introducción

El desarrollo de nuevos fármacos es un proceso largo y complejo que implica tres fases principales. La primera y segunda fase de ***descubrimiento*** y ***exploración***, donde se descubre un candidato a fármaco y se establece un prueba de concepto mediante ensayos en modelos animales y, eventualmente, en un conjunto de pacientes humanos reducidos [48]. La tercera fase de ***confirmación***, donde el candidato a fármaco se desarrolla de manera completa y se extiende su aplicación a un gran número de pacientes. Aunque es difícil de cuantificar, este proceso en su conjunto puede alcanzar entre 12 y 15 años de duración, como se muestra en la figura 1.1, con importantes costes económicos [17], limitando los tiempos de respuesta ante situaciones de crisis en la sociedad. Esta tesis doctoral se centra en la ***aceleración*** de la fase de descubrimiento, utilizando la bioinformática estructural, y en concreto el ***cribado virtual*** [29, 32], como herramienta para el descubrimiento de nuevos compuestos bioactivos.

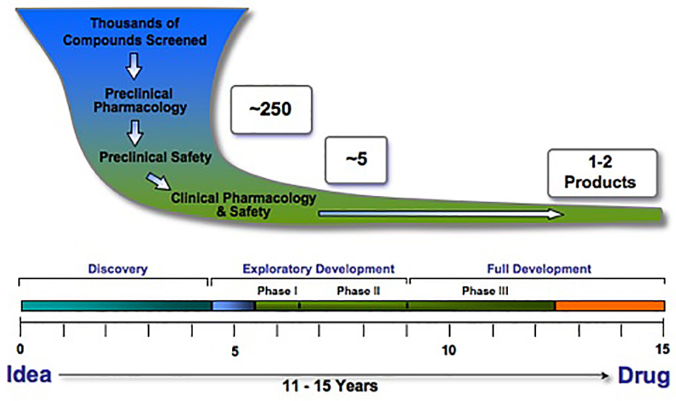


Figura 1.1: El proceso de descubrimiento de fármacos, sus diferentes fases y la duración aproximada de cada una de ellas.

El cribado virtual consiste en la simulación en computadoras ("*in-silico*") del proceso de ***interacción molecular***. Estas herramientas están siendo muy utilizadas, entre otras, para el descubrimiento de nuevos fármacos [46], ya que permiten simular, una ingente cantidad de interacciones entre proteínas (también conocidos como receptores) y candidatos a fármacos (***ligandos***) a una gran velocidad y bajo coste.

Sin embargo, el éxito de estas herramientas radica en dos puntos principales. En primer lugar, el número de ***ligandos*** a procesar; cuántos más ligandos se procesen más probable es encontrar un ligando candidato a fármaco. En segundo lugar, el ***modelo*** físico-químico que representa la interacción molecular; cuánto más preciso más posibilidades de reproducir fielmente la realidad. Por tanto el éxito de estas técnicas radica en un proceso computacionalmente intensivo que hace obligatorio el uso de las técnicas ***hardware*** y ***software*** más avanzadas.

1.1 La supercomputación al servicio del descubrimiento de fármacos

El paradigma de supercomputación se ha fundamentado, entre otras, en las últimas innovaciones en el desarrollo de microprocesadores. Tradicionalmente, este desarrollo ha estado guiado por la ley de Moore [52] que establece que el número de transistores dentro del mismo área de silicio se duplica aproximadamente cada 18 meses [53]. Esta ley empírica establecida en el año 1965 por Gordon E. Moore se ha cumplido hasta el día de hoy, alcanzando un tamaño de integración de 14 nm en la actualidad en los procesadores de Intel Kaby Lake y planificando una escala de integración de 5 nm con la familia Canonlake [66]. Este continuo decremento de la escala de integración se ha visto frenado por la leyes físicas del silicio, declarándose por primera vez en las últimas décadas el fin de la ley de Moore [73].

El aumento en el rendimiento de los sistemas de cómputo sigue siendo demandado por las nuevas aplicaciones de consumo. Dominios como la Bioinformática, el Big Data y la Realidad Virtual son sólo ejemplos de aplicaciones con una alta demanda de cómputo que requieren de un continuo incremento del poder computacional de los procesadores. Actualmente, este aumento está viniendo de la mano de dos componentes principales, (1) la ***especialización*** y (2) el ***paralelismo masivo***. Promovido por el lucroso mundo de los videojuegos, las unidades de procesamiento gráfico (*Graphics Processing Units*, GPUs) se han situado como un dispositivo de altas prestaciones para la ejecución de partes de códigos intensivas en cómputo que puedan beneficiarse de un paralelismo masivo de datos. Este es sólo un ejemplo donde la especialización y paralelismo masivo, van de la mano para ofrecer rendimientos pico de hasta 10.6 TeraFLOPS [55].

La especialización acompaña a la generalidad de procesamiento en los nuevos sistemas heterogéneos [3], donde los nodos combinan las arquitecturas multicore tradicionales (CPUs) con aceleradores como las GPUs, ofreciendo rendimientos muy elevados en comparación con arquitecturas multicore tradicionales [47]. Sin embargo, la heterogeneidad puede limitar el crecimiento del sistema en la ruta hacia el *Exascale* [62], ya que no se puede abordar el desarrollo de grandes computadores de forma incremental. Entre los desafíos existentes, se destaca la eficiencia energética [11] donde el consumo está previsto que crezca exponencialmente. Pero ésta no es la única limitación, la escalabilidad de las aplicaciones, la facilidad de su programación o la gestión de datos, por mencionar sólo algunos, pueden comprometer también el crecimiento del sistema.

Uno de los enfoques tomados por la comunidad para aprovechar todos los recursos de un clúster de computadores, de manera transparente al usuario, ha sido la virtualización del sistema. Se han desarrollados diferentes técnicas de virtualización capaces de gestionar los recursos disponibles aumentando el *throughput* del sistema [12]. Soluciones *software* como VMware [64] o Xen [5] están cada vez más presentes en los centros de datos. La idea de estos sistemas es virtualizar todo el sistema y presentar a cada usuario una máquina totalmente funcional y transparente, mostrando todos los recursos como si fuesen propios. Sin embargo, aunque el uso de entornos virtuales es atractivo en muchos casos, incluso para computación de alto rendimiento, cuando el objetivo es hacer uso de aceleradores, estas soluciones introducen una sobrecarga inaceptable debido a las fuertes limitaciones que presentan con respecto a su uso compartido. En este sentido, los enfoques actuales de máquinas virtuales no pueden compartir simultáneamente una GPU entre varias instancias de máquina virtual [33].

En lugar de virtualizar todo el sistema, un enfoque alternativo sería virtualizar recursos específicos, como la GPU. rCUDA [61] es un software del sistema que permite el uso simultáneo y remoto de GPUs compatibles con CUDA (*Compute Unified Device Arquitecture*) [45, 56]. Para habilitar la aceleración remota de GPUs, este software del sistema crea dispositivos virtuales compatibles con CUDA en máquinas sin GPUs locales, habilitando servicios GPGPU (*General-Purpose computing on Graphics Processing Unit*) en ese nodo del clúster. De este modo, todos los nodos de un clúster son CUDA-compatibles. Además, rCUDA aporta una reducción de la complejidad algorítmica, evitando utilizar técnicas basadas en paso de mensajes (MPI) [18, 76], comúnmente utilizadas en estos entornos.

1.2 Métodos computacionales actuales para el descubrimiento de fármacos

La evolución del poder computacional de los computadores ha hecho proliferar el desarrollo de nuevas aplicaciones bioinformáticas. En el campo del cribado virtual se destaca *AutoDock* [54] y *AutoDock VINA* [74] debido al gran impacto de sus publicaciones (más de 10.000 citas según *Google Scholar* en Noviembre de 2017). Estos métodos son estándares de facto en la comunidad. Sin embargo, su alta carga computacional se trata desde una perspectiva algorítmica, utilizando aproximaciones basadas en mallas precalculadas [51, 58]. En estas aplicaciones, la explotación del sistema paralelo queda limitado a la explotación con OpenMP [57] de sistemas basados en *multicores* de memoria compartida.

Existen otras aplicaciones de cribado virtual como *Glide* [23], *LeadFinder* [71], *SurFlex* [40], *ICM+* [70], *FlexScreen* [58] o *DOCK* [20]. La gran mayoría de estas herramientas se limitan también a arquitecturas de memoria compartida, y algunas de ellas como *FMD* [19] añade además tecnologías de paso de mensajes (MPI) [76]. Software como *BUDE* [49], *BINDSURF* [68], *GROMACS* [34] o *AMBER* [67] utilizan además aceleradores (principalmente GPUs) para trabajar con problemas de cribado virtual y dinámica molecular.

Finalmente destacar que, esta tesis doctoral tiene como fundamentos el trabajo previo realizado por mis directores de tesis en BINDSURF. BINDSURF desarrollado desde cero en GPUs, se caracteriza por ser una novedosa metodología de cribado virtual que explora toda la superficie de las proteínas para encontrar nuevos puntos de unión (*hotspots*), donde los ligandos podrían interactuar. Este nuevo enfoque llamado (*Blind docking*) aumenta la calidad predictiva de la aplicación, sin embargo incrementa exponencialmente las necesidades computacionales de dichos métodos. Además destacar que BINDSURF está basado en el método de Monte Carlo [63] cuya paralelización es bastante compleja y no adecuada a la arquitectura de la GPU. En esta tesis doctoral se exploran nuevas estrategias de búsqueda que se adaptan mejor al nuevo escenario de computación.

1.3 Esquemas metaheurísticos parametrizados para la optimización de métodos de cribado virtual

Las aplicaciones de *docking* o acoplamiento molecular predicen la bondad del enlace entre dos moléculas [46], en nuestro caso, entre la proteína y el ligando en una determinada posición, orientación y forma. Estas aplicaciones están basadas en la ***optimización*** de funciones de ***scoring*** [41]. Las funciones de *scoring* están formadas por un conjunto de modelos matemáticos que representan los comportamientos físicos de las moléculas, permitiendo trasladar la interacción molecular a una simulación en plataformas computacionales. Entre ellas, destacamos el valor de las fuerzas electrostáticas [28], el potencial de Lennard-Jones [30] como modelo matemático para resolver las fuerzas de Van der Waals, el cálculo de los enlaces por puente de hidrógeno [21], y el término de desolvatación o de torsión [77].

Como se puede intuir, la optimización de este tipo de funciones es muy costosa computacionalmente. De hecho, se considera un problema ***NP-completo*** [69]; por lo que la resolución de dicho problema requiere de un tiempo de ejecución o espacio de memoria que crece de manera no polinómica [15, 25]. Para trabajar con este tipo de problemas se suelen utilizar algoritmos ***heurísticos*** [7]. Las heurísticas buscan un compromiso entre el tiempo de ejecución de los algoritmos y la calidad de la solución encontrada. En muchas ocasiones esta solución no es óptima.Un nivel de abstracción superior al de las técnicas heurísticas son las técnicas ***metaheurísticas*** [50]. Estas técnicas son estrategias heurísticas empleadas para resolver cualquier tipo de problema computacional de manera abstracta.

Las metaheurísticas utilizan una serie de parámetros dados por el usuario en procedimientos abstractos y genéricos buscando la eficiencia y una solución cercana a la óptima. Existen una gran variedad de algoritmos metaheurísticos en la bibliografía [43]: optimización aleatoria, búsquedas locales, algoritmos genéticos, etc. Existen diferentes taxonomías para clasificar metaheurísticas. En nuestro caso vamos a distinguir dos tipos, las basadas en ***vecindad*** y en ***poblaciones***. Las metaheurísticas basadas en vecindad usan el concepto de *k-vecinos*, donde k es el número de vecinos que se va a generar desde una posible solución, actualizando progresivamente esta solución mediante la exploración de sus vecinos. La vecindad es el número de posibles soluciones alcanzables desde la solución actual. La determinación de la vecindad es muy importante en estas metaheurísticas, pudiendo construirla por ejemplo aplicando un operador de movimiento sobre el individuo. Como ejemplos de metaheurísticas basadas en búsqueda local tenemos: VNS o *Variable Neighborhood Search* (Búsqueda en Vecindario Variable) [31], ILS o *Iterated Local Search* (Búsqueda Local Iterada) [72] y TS o *Tabu Search* (Búsqueda Tabú) [26]. En las metaheurísticas basadas en poblaciones [6] se genera un conjunto de individuos iniciales y se va trabajando sobre ellos. Destacan los algoritmos Genéticos [35] o las búsquedas dispersas [27]. Ambos tipos de metaheurísticas son muy utilizados en procesos de cribado virtual [46].

Existe por tanto un conjunto muy amplio de estrategias que se pueden aplicar a un mismo problema. De hecho, el problema de buscar la mejor metaheurística para un problema determinado es, en sí mismo, un problema de optimización. Algunos autores han planteado ***esquemas*** algorítmicos unificados [60, 75] para representar un amplio abanico de metaheurísticas comunes. Estos esquemas definen un conjunto de funciones básicas (*Inicializar, Fin, Seleccionar, Combinar, Mutar, Mejorar e Incluir*) para generar una plantilla que, dependiendo de su instanciación, pudieran generar diferentes tipos de metaheurísticas. Como ejemplo, se podría definir un algoritmo genético a partir del esquema omitiendo la mejora, quedando una combinación-mutación. También podemos explorar una búsqueda local no combinando-mutando los elementos y explorar la vecindad a partir de una inicialización junto con la función de mejora. De este modo se pueden crear fácilmente metaheurísticas híbridas a partir de heurísticas tradicionales.

Esta versatilidad a la hora de adaptarse a una metaheurística u otra cambiando la implementación de sus funciones, nos permite ir un poco más allá e introducir una serie de parámetros adicionales a las funciones [4]. Estos parámetros adicionales convierten el esquema metaheurístico general en un esquema metaheurístico ***parametrizado***, donde cada una de las funciones que lo componen reciben un conjunto de parámetros de entrada, a fin de configurar un tipo de metaheurística concreta a instanciar en cada ejecución de la aplicación. Este conjunto de parámetros de entrada permite no sólo determinar la optimización del problema a resolver, sino también el algoritmo empleado para su resolución.

La selección de los parámetros utilizados también puede verse como un problema de optimización. Sus diferentes valores dan lugar a distintas metaheurísticas, no pudiendo determinar a priori una combinación apropiada para un problema concreto. Este problema se puede abordar a través de un esquema superior a la metaheurística, denominado ***hiperheurística*** [8, 65], que optimice sus parámetros en función del valor de *fitness* de la metaheurística [16]. La hiperheurística busca la mejor de entre un conjunto de metaheurísticas aplicadas a un mismo problema. Al igual que pasa con las metaheurísticas, existen distintas taxonomías para su clasificación. En nuestro caso, vamos a clasificar las hiperheurísticas desde dos puntos de vista [9], (1) el primero basado en la selección de heurísticas en función de combinación de heurísticas conocidas, y (2) generación de nuevas heurísticas a partir de patrones computacionales de partes de varias heurísticas.

Las hiperheurísticas basadas en selección de heurísticas, tienen un conjunto de heurísticas definidas, y de forma progresiva se aplica cada heurística al problema actual. A finalizar se elige la heurística que obtenga la mejor solución. Diversos problemas se han abordado con hiperheurísticas de esta categoría, tales como implementación de una lista tabú de heurísticas para resolver problemas de optimización [10] o problemas de rutas de vehículos [59]. En las hiperheurísticas basadas en patrones de computo de varias heurísticas, se generan automáticamente nuevas heurísticas con bloques de otras, obteniendo una solución para problemas NP, como el problema del viajante de comercio [44], problemas de satisfacción booleana [24] o el problema del embalaje [1].

Existen por tanto un conjunto de estrategias algorítmicas basadas en heurísticas y metaheurísticas que podemos explorar para encontrar soluciones para el problema del cribado virtual.

1.4 Objetivos

Esta tesis doctoral se fundamenta sobre un conjunto de objetivos que se enumeran y se detallan a continuación. Estos objetivos son los pilares sobre los que se asienta la investigación presentada en esta tesis doctoral, y trabajan en dos líneas distintas. La primera línea se orienta a conseguir logros desde el punto de vista computacional, en cuanto a diseño algorítmico y rendimiento. Una segunda línea de trabajo se encamina a mejorar la calidad predictiva del método, validando las simulaciones con bases de datos de referencia en bioinformática estructural.

- **Objetivo 1. Diseño de un método de búsqueda para problemas de cribado virtual.** *Implementar un método de búsqueda para problemas de cribado virtual basado en un esquema metaheurístico parametrizado que permita la generación de un gran abanico de metaheurísticas*

  El esquema metaheurístico parametrizado ofrece una visión novedosa y mejorada respecto a las metodologías actuales de búsqueda, pudiendo combinar y elaborar diferentes estrategias de búsqueda variando determinados parámetros. Estos parámetros dependen de (1) el problema de descubrimiento de fármacos, como puedan ser el número de átomos o número de enlaces rotables del ligando, y (2) de la metaheurísticas en sí, como tamaño de la población, número de iteraciones o porcentajes de selección, combinación o mejora.

- **Objetivo 2. Optimización del método de búsqueda mediante hiperheurísticas.** *Aumentar un nivel de abstracción el esquema metaheurístico a uno* *hiperheurístico pudiendo elegir un esquema de cálculo diferente cada vez que se quiera o necesite, adaptado al problema concreto.* Se pretende el desarrollo de hiperheurísticas basadas en esquemas metaheurísticos parametrizados, cuya finalidad es la selección automática de la mejor metaheurística para un problema o conjunto de problemas dados, en función del *fitness*. Como la estructura de las hiperheurísticas se basa en el esquema parametrizado de metaheurísticas, se puede considerar en este caso la misma metodología de modelado y optimización, pero a un nivel superior de abstracción.

- **Objetivo 3. Análisis, diseño y optimización del nuevo método de cribado virtual en clústers de GPUs.** *Ejecutar los desarrollos en un clúster heterogéneo aprovechando los recursos de la manera más óptima posible.*

  Actualmente son muy comunes los clústers de cómputo, que junto a las CPUs, incorporan GPUs con soporte CUDA de diferentes familias tecnológicas y capacidades distintas en uno o varios nodos del mismo clúster, tanto físicos como virtualizados. Las técnicas algorítmicas que se van a desarrollar, van a posibilitar la elección de las diferentes plataformas de cómputo disponibles optimizando el entorno de ejecución y, balanceando la carga de trabajo con los parámetros de configuración más idóneos. Además, se extenderá a la exploración de clústers con técnicas de virtualización de GPUs, para evaluar parámetros de escalabilidad y rendimiento.

1.5 Estructura del documento de Tesis

Por último, para guiar al lector y estructurar mejor el contenido de la tesis doctoral, a continuación se describe de forma resumida el contenido de cada uno de los capítulos que la componen:

- *Capítulo 1: Introducción.* En este primer capítulo se contextualiza

  la tesis doctoral, describiendo el campo de estudio, el problema al que nos enfrentamos, y soluciones planteadas por otros investigadores para resolverlo. También se enumeran los objetivos a alcanzar a lo largo de esta tesis.

- *Capítulo 2: Publicaciones que componen la tesis doctoral.* En este capítulo se incluyen las publicaciones que forman parte del compendio junto con la fundamentación de la tesis. Esta fundamentación consiste en demostrar que con el conjunto de publicaciones científicas presentadas, se obtienen los resultados que cumplen los objetivos propuestos en el capítulo 1.

- *Capítulo 3: Conclusiones y Vías futuras.* Cada una de las publicaciones científicas que se presentan conllevan asociados una serie de resultados y conclusiones que nos permiten constatar la consecución de los objetivos propuestos. En este capítulo se comentan las conclusiones que se derivan de las publicaciones y se plantean posibles vías de continuación de las investigaciones.

- *Capítulo 4: Calidad de las publicaciones.* Las revistas en las que se han publicado los artículos que forman el compendio, tienen asociadas una serie de indicadores de calidad. En este capítulo se resumen estos indicadores para constatar la calidad de las publicaciones aportadas en esta tesis doctoral.

- *Bibliografía.* Se aportan referencias donde profundizar y conseguir más información de los estudios planteados.

Capítulo 2

Artículos que componen la tesis doctoral

2.1 Fundamentación de la tesis doctoral

La consecución de los objetivos planteados en el capítulo 1 a través de publicaciones científicas en revistas de reconocido prestigio, aporta solidez a la hora de justificar su cumplimiento. A continuación se va a identificar cada una de las tres publicaciones aportadas en esta tesis por compendio con el objetivo que aborda, desde el diseño algorítmico del método, hasta los diferentes estudios desarrollados de rendimiento y exploración paramétrica.

La primera de las publicaciones sobre la aplicación de metaheurísticas a procesos de cribado virtual, marca el inicio de la línea de investigación por la que se guía esta tesis. En esta primera publicación, se diseña un nuevo método basado en un esquema metaheurístico parametrizado para abordar los problemas de cribado virtual en entornos de computación heterogéneos (CPU-GPU). El uso de computación heterogénea mejora el rendimiento con respecto al uso exclusivo de arquitecturas tradicionales basadas en *multicore* CPU. Además, el método proporciona buenos resultados en calidad predictiva, comprobando sus predicciones con resultados experimentales de bases de datos de referencia. Este nuevo método, METADOCK [38], es el primer artículo que forma parte del compendio, cumpliendo el primer objetivo de la tesis doctoral, que consiste en diseñar de un método de búsqueda para problemas de cribado virtual.

Como se ha explicado anteriormente, METADOCK esta basado en un esquema parametrizado que contiene un amplio conjunto de parámetros metaheurísticos. Este número elevado de posibles combinaciones paramétricas, hace muy extenso el número de metaheurísticas que se pueden generar. Para ayudar a trabajar en la búsqueda de la mejor combinación paramétrica, en la segunda publicación del compendio, se puso en marcha una estrategia algorítmica con un nivel de abstracción superior a la metaheurística, denominada hiperheurística. Esta hiperheurística busca la mejor combinación de parámetros metaheurísticos, que dará como resultado una metaheurística más eficiente para abordar el problema. Para acelerar la computación, se utiliza un esquema basado en una arquitectura *multicore* de memoria compartida en CPU con varios niveles de paralelismo, distribuidos entre los esquemas hiperheurístico y metaheurístico. Las ejecuciones se realizan en un sistema *many-core* como es el Intel Xeon Phi [14], demostrando que esquema algorítmico es totalmente escalable, obteniendo resultados cercanos al óptimo con una importante reducción del tiempo de ejecución. Este aumento de rendimiento es comparado al uso exclusivo de *multicore* con una sola CPU. Este nuevo nivel de abstracción plasmado en [13] cumple el objetivo 2 de esta tesis, que consiste en la optimización del método de búsqueda a través de hiperheurísticas. Las dos contribuciones anteriores que forman parte del compendio ponen de manifiesto el alto coste computacional de este tipo de técnicas, y la necesidad de utilizar aceleradores para mitigar este gasto. Para abordar la extensión a un clúster heterogéneo de cómputo, se hace necesario el uso de técnicas basadas en paso de mensajes (MPI), añadiendo una mayor complejidad al binomio actual de OpenMP + CUDA en un sólo nodo.

En el tercer artículo que forma parte del compendio se analiza la escalibilidad de METADOCK en un clúster heterogéneo. Para ello se compara el uso de MPI + OpenMP + CUDA con el uso de técnicas de virtualización, donde el acceso a las GPUs remotas de otros nodos del clúster se realiza a través del middleware *rCUDA*. El estudio realizado ejecutando METADOCK con *rCUDA* [39] comprueba la escalabilidad del problema de cribado virtual usando METADOCK, a un clúster heterogéneo, cumpliendo el objetivo 3 de la tesis doctoral, analizando, diseñando y optimizando del nuevo método de cribado virtual en clústers de GPUs. Los resultados obtenidos muestran que *rCUDA*, además de reducir la complejidad algorítmica, mejora el rendimiento en clústers heterogéneos y ofrece mejores prestaciones que arquitecturas tradicionales de paso de mensajes, utilizadas comúnmente en estos entornos.

2.2 METADOCK: A parallel metaheuristic schema for virtual screening methods

Título	*METADOCK: A parallel metaheuristic schema for virtual screening methods*
Autores	Baldomero Imbernón, José M. Cecilia, Horacio Pérez y Domingo Giménez
Revista	The International Journal of High Performance Computing Application
Año	2017
Estado	Publicado

Contribución del Doctorando

Baldomero Imbernón Tudela, declara ser el principal autor y el principal contribuidor del artículo *METADOCK: A parallel metaheuristic schema for virtual screening methods*.

International Journal of
HIGH PERFORMANCE
COMPUTING APPLICATIONS

Original Article

The International Journal of High Performance Computing Applications
1–15
© The Author(s) 2017
Reprints and permissions:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/1094342017697471
journals.sagepub.com/home/hpc
SAGE

METADOCK: A parallel metaheuristic schema for virtual screening methods

Baldomero Imbernón[1], José M Cecilia[1], Horacio Pérez-Sánchez[1] and Domingo Giménez[2]

Abstract
Virtual screening through molecular docking can be translated into an optimization problem, which can be tackled with metaheuristic methods. The interaction between two chemical compounds (typically a protein, *enzyme* or *receptor*, and a small molecule, or *ligand*) is calculated by using highly computationally demanding scoring functions that are computed at several binding spots located throughout the protein surface. This paper introduces *METADOCK*, a novel molecular docking methodology based on parameterized and parallel metaheuristics and designed to leverage heterogeneous computers based on heterogeneous architectures. The application decides the optimization technique at running time by setting a configuration schema. Our proposed solution finds a good workload balance via dynamic assignment of jobs to heterogeneous resources which perform independent metaheuristic executions when computing different molecular interactions required by the scoring functions in use. A cooperative scheduling of jobs optimizes the quality of the solution and the overall performance of the simulation, so opening a new path for further developments of virtual screening methods on high-performance contemporary heterogeneous platforms.

Keywords
Drug discovery, virtual screening, molecular docking, high performance computing, metaheuristics, heterogeneous computing

1 Introduction

The discovery of new drugs may benefit from using virtual screening (VS) methods (Jorgensen, 2004). These are computational techniques that analyze large libraries of small molecules (*ligands*) to search for compounds which are most likely to bind to a drug target, typically a protein receptor or enzyme (e.g. Rester, 2008; Rollinger et al., 2008). These libraries of chemical compounds may contain millions of ligands (Irwin and Shoichet, 2005). Indeed, the analysis of larger databases increases exponentially the chances of generating hits.

The computational complexity of VS methods is determined by two main parameters: The size of the database to be analyzed and the accuracy of the chosen VS method. Fast VS methods with atomic resolution require some minutes per ligand (Zhou et al., 2007). In contrast, molecular dynamic approaches can require up to thousands of hours per ligand (Wang et al., 2006). Therefore, the main bottleneck for the success of VS methods is the lack of computational resources or, in other words, there is a need for highly efficient applications that leverage emergent, high performance computing architectures (Asanovic et al., 2006).

We are witnessing the steady transition to heterogeneous computing systems (Top500, 2016), where nodes combine traditional central processing units (CPUs), which may have multiple cores, and many-core systems or accelerators (like NVIDIA graphics processing units, GPUs, NVIDIA Corporation, 2017 or Intel Xeon Phi, Sodani et al., 2016), that enable us to speed-up computationally demanding parts of the code. Heterogeneity limits system growth, which can not now be addressed in an incremental way. In particular, concepts like scalability, energy barrier, data management, programmability and reliability are becoming challenges for tomorrow's cyberinfrastructures (Song et al., 2016). Run-time systems are still too immature to map processors and computations efficiently. In the meantime,

[1]Bioinformatics and High Performance Computing Research Group (BIO-HPC), Universidad Católica San Antonio of Murcia (UCAM), Spain
[2]Department of Computing and Systems, University of Murcia, Spain

Corresponding author:
Baldomero Imbernón, Bioinformatics and High Performance Computing Research Group (BIO-HPC), Polytechnic School, Universidad Católica San Antonio of Murcia (UCAM), 30107, Murcia, Spain.
Email: bimbernon@ucam.edu

researchers are focusing on the latest breakthroughs in high performance computing together with specific fields (metaheuristics, image processing, computational modeling, etc.). This results in a vertical approach enabling remarkable advances in computer-driven scientific simulations, the so-called hardware-software codesign (De Michell and Gupta, 1997).

Programmers play a fundamental role in this emerging scenario. They have to redesign, and even rethink, applications to leverage the best features of all the sides in a joint execution to maximize performance, with parallelism as a mandatory ingredient. Of particular interest to us are *metaheuristic* algorithms, especially those inspired by *natural* processes, whose computations are intrinsically massively parallel, and therefore well-suited for implementation in this area of computation (Cecilia et al., 2013). There are a wide variety of metaheuristic algorithms, each with its own characteristics (Blum and Roli, 2003), and sometimes it is necessary to develop and experiment with various soft computing methods, and tune them for each particular problem in order to obtain satisfactory results.

Metaheuristics are frequently used to solve Non Polynomial (NP)-hard problems (Rozenberg et al., 2011). Some of these problems are in the field of *bioinformatics*, e.g. DNA analysis (Minetti et al., 2008) or molecular docking (López-Camacho et al., 2015). Many metaheuristic methods are available, including *distributed metaheuristics* (e.g. genetic algorithms, scatter search, ant colony, particle swarm optimization, etc.) and *neighborhood metaheuristics* (e.g. tabu search, hill climbing, simulated annealing, etc.). The best metaheuristic to deal with a particular problem is not clear. Many experiments with different metaheuristics and hybridations of basic metaheuristics are needed to discover the optimum solution for a problem. Additionally, for any particular metaheuristic, a tuning process is traditionally conducted to select appropriate values of some parameters in the metaheuristic, and experimentation with several metaheuristics and their tuning process will drastically increase the computational cost.

In this paper, we introduce *METADOCK*, a virtual screening technique that uses a unified schema to generate a wide range of metaheuristics. *METADOCK* is designed to leverage massively parallel and heterogeneous architectures, including chip multiprocessors and NVIDIA's GPUs. We use a molecular *docking* computational methodology that seeks to predict non-covalent binding of molecules or, more frequently, of a macromolecule (receptor) and a small molecule (ligand). The goal is to predict the bound conformations and the binding affinity; i.e. the strength of association, which is usually measured with a scoring function. These functions compute the score by calculating different conformations of the ligand at several binding spots throughout the protein surface, including computations between pairs of atoms in the protein and the ligand. For detailed reviews on recent and widely used scoring functions see the wokr by Yuriev and Ramsland (2013); Li et al. (2014b,a) and Lionta et al. (2014).

This paper is an extension of a previous work developed by the same authors (Imbernón et al., 2016). Major contributions include the following.

1. A new methodology called *METADOCK* for virtual screening, widely applied in the field of computational drug discovery, is introduced. It is based on a parameterized schema that is able to generate a branch of different search algorithms (metaheuristics) by setting different input parameters.
2. *METADOCK* leverages heterogeneous computers based on CPUs and NVIDIA GPUs to provide an agile framework to run real experiments. We assume the nodes may have NVIDIA GPUs with different compute capabilities. A load balance strategy is introduced to efficiently distribute different workloads at runtime among all GPUs in the system. This load balancing strategy is based on the application performance.
3. We analyze *METADOCK*'s prediction quality by calculating its performance on binary classification (active or not active compounds), for which we generate receiver operating characteristic (ROC) curve analysis, after processing benchmarks containing experimental information about protein-ligand datasets, such as the directory of useful decoys (DUDs). Our results place *METADOCK* as a very competitive docking method, reporting an area under curve (AUC) value of up to 0.84, in the benchmarks reported.
4. We provide an extensive performance analysis to validate our parallelization strategy for such heterogeneous environments. Our results reveal that our techniques achieve up to a $50\times$ speed-up factor compared to a multicore counterpart version. Moreover, they also provide the following conclusions:
   (a) homogeneous distribution is not a good load balancing strategy for systems with GPUs with different compute capabilities;
   (b) technical specifications are not enough to achieve peak performance, and load balancing at runtime is needed, with the workload depending on application performance;
   (c) all these parallelization strategies give the opportunity to improve the solution quality in our problem.

The rest of the paper is structured as follows. The next section includes the background of virtual screening and some relevant knowledge about metaheuristics and GPU computing. Next, our metaheuristic-based virtual screening technique and its design for heterogeneous computers are introduced before showing the experimental set-up and results. A final section summarizes the conclusions and gives some directions for future work.

2 Background

This section briefly shows the main characteristics of virtual S screening methods, metaheuristics and GPU computing for a better understanding of the rest of the paper.

2.1 GPU computing

Almost from the outset, computer architects have relied on technology scaling to provide sustainable performance. Heterogeneous architecture design is now seen as the only solution to continue scaling Moore's law through innovation (Austin, 2015), with systems where nodes combine traditional multicore architectures (CPUs) and accelerators (mostly GPUs, Kirk and Wen-mei, 2013 or Intel Xeon Phi, Sodani et al., 2016). In a few years, the GPGPU (developing general purpose application on GPUs) field expanded and became one of the best ways of achieving high performance computing from emergent heterogeneous computers. We briefly introduce the compute unified device architecture (CUDA) programming model and refer the reader to the NVIDIA Corporation (2017) for insights. CUDA is a platform for GPUs that covers both hardware and software. On the hardware side, the GPU consists of N multiprocessors which are replicated within the silicon area. Each is endowed with M cores sharing the control unit, and with a shared memory (a small cache explicitly managed by the programmer). Moreover, all of these multiprocessors are connected to an off-chip memory (GPU device memory) that acts as an interface to the host CPU. Each GPU generation has increased the CUDA compute capabilities (CCCs), as well as the number of cores, and the shared and device memory sizes (see Table 1). In conjunction with these developments, the power efficiency has been reduced by a factor of two in 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 an SIMD (single instruction multiple data) fashion (that is, with all threads running the same code but with different threads number). The programmer deploys parallelism by declaring a *grid* composed of blocks equally distributed among all the 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.

2.2 Metaheuristics

There are many optimization problems of high computational cost which can not be solved by evaluating all the possible solutions. Due to the high computational cost of exact methods, the optimum solutions for the NP-hard problems can be found for only very small instances, and so they are traditionally approached through heuristics and metaheuristics (general information on metaheuristics can be found, for example, in Blum et al., 2011; Dréo et al., 2005; Glover and Kochenberger, 2003; Hromkovič, 2003; Michalewicz and Fogel, 2002), which are tuned for the problem in question.

Metaheuristics include an abstraction layer that may provide a sufficiently good solution for an optimization problem, especially with limited computation capacity or inexact information (Bianchi et al., 2009). They reduce the search space, focusing only on the most promising areas, and thus, cannot guarantee the analysis of all possible solutions, which means that they do not guarantee they will find the optimal solution. There are many metaheuristic algorithms of different characteristics (Blum and Roli, 2003) that can provide several good solutions to the same problem. Among them, we highlight the following.

1. *Distributed metaheuristics*, which search for solutions within the whole solution space. These work

Table 1. CUDA summary by generation, with Maxwell to increase the number of cores soon.

Hardware generation and starting year	Fermi 2010	Kepler 2012
Multiprocessors per die (up to)	16	15
Cores per multiprocessor	32	192
Total number of cores (up to)	512	2880
Shared memory size (maximum in KB)	48	48
Device memory size (maximum in GB)	6	12
CUDA Compute capabilities	2.x	3.x
Peak single-precision performance (GFLOPS)	1178	4290
Performance per watt (approx. and normalized)	2	6

with populations or sets of elements that are combined in order to generate better solutions progressively. Some examples include *scatter search, genetic algorithms, ant colony* and *particle swarm optimization.*

2. *Neighborhood metaheuristics*, which work with an element in the solution space and search for better elements in its neighborhood. Examples include *hill climbing, tabu search, guided local search, variable neighborhood search, simulated annealing* and *greedy randomized adaptive search procedure (GRASP).*

2.3 Virtual screening

We draw on our description of the virtual screening (VS), which was first given by Guerrero et al. (2014) and Sánchez-Linares et al. (2012). VS methods are computational techniques used in several scientific areas, such as catalysts and energy materials (Franco, 2013), and mainly drug discovery (Kitchen et al., 2004), where experimental techniques can benefit from the predictions provided by simulation methods.

VS methods search for libraries of small molecules that can potentially bind to a drug target, typically a protein receptor or enzyme, with high affinity. Within VS methods, molecular docking techniques simulate the docking process of small molecules into the structures of macromolecular targets (see Figure 1). Moreover, they look for (i.e. score) the optimal binding sites by providing a ranking of chemical compounds according to the estimated affinity or *scoring* (Schneider, 2002). In general, VS methods optimize *scoring functions*, which are mathematical models used to predict the strength of the non-covalent interaction between two molecules after docking (Jain, 2006). In fact, these candidate molecules will continue the drug discovery process roadmap that goes from in-vitro studies, to animal investigations and, eventually, to human trials (Drews, 2000).

Although VS methods have been researched for many years and several compounds can be identified that evolve into drugs, the impact of VS has not yet fulfilled all expectations. Neither the VS methods nor the scoring functions used are sufficiently accurate to identify high-affinity ligands reliably. To deal with a large number of potential candidates (many databases comprise of hundreds of thousands of ligands), VS methods must be very fast and still be able to identify "the needles in the haystacks". These techniques require hundreds of CPU hours for each ligand, and, according to Wang et al. (2006), even thousands of CPU hours for each ligand when simulation strategies are used to compute absolute binding affinities.

The relevant non-bonded potentials used in VS calculations are the Coulomb, or electrostatic, and the Lennard–Jones potentials, since they describe the most important short and long-range interactions between atoms of the protein-ligand system very accurately. The calculation of non-bonded potentials is usually the most time-consuming calculation in VS methods. For example, in molecular dynamics simulations, the calculation of these kernels can take up to 80% of the total execution time (Kuntz et al., 2001).

Within these VS methods, of particular interest to us are protein–ligand docking techniques. Examples are given in the work by Huang and Zou (2010) and Yuriev et al. (2011). Docking simulations are typically carried out on the protein surface using known methods, like Autodock (Morris et al., 1998); Glide (Friesner et al., 2004); DOCK (Ewing et al., 2001); FMD (Dolezal et al., 2015), which combines message passing interface (MPI) with multithreading; or BUDE McIntosh-Smith et al. (2014), a molecular docking program for hybrid computing architectures that exploits the heterogeneity with OpenCL for portability to different computer architectures. The surface region is commonly derived from the position of a particular ligand in the protein–ligand complex, or from the crystal structure of the protein without any ligands. The main problem of many docking methods is that they assume, once the binding site is specified, that all ligands will interact with the protein in the same region, and completely discard the other areas of the protein. In *BINDSURF*, Sánchez-Linares et al. (2012) use GPUs to overcome this problem by dividing the whole protein surface into arbitrary independent regions (or spots). Using GPU parallelism, a large ligand database is screened against the target protein simultaneously over its whole surface, and docking simulations for each ligand are performed simultaneously in all the specified protein spots, resulting in new spots found after the examination of the distribution of scoring function values over the entire protein surface.

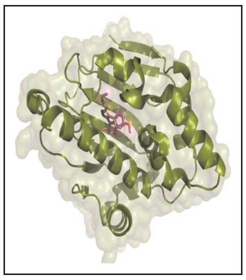


Figure 1. Crystallographic structure of 5-(5-chloro-2,4-dihydroxyphenyl)-N-ehtyl- 4-(4-methoxyphenyl)-1H-pyrazole-3-carboxamide (pink color) bound to HSP90 protein (green color). Details available on the PDB database with accession code 2BSM.

3 Metaheuristics for virtual screening on heterogeneous systems

Traditional parallel implementations are not always efficient when ported to heterogeneous systems. They are often inherited from scalable supercomputers, where all nodes in the cluster have the same compute capabilities, and therefore they lack the ability to distinguish computational devices with asymmetric computational power. 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) 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). Here, we distinguish two different aspects; the system itself, which may be heterogeneous or homogeneous, and the parallel distribution of the workload which can also be heterogeneous or homogeneous. This section shows our proposal, *METADOCK*, for metaheuristic-based virtual screening applications that leverage massively parallel and heterogeneous computers. We introduce the reader to the design of our virtual screening approach before showing the parallelization strategy for heterogeneous distribution of the workload.

3.1 METADOCK: Metaheuristics for VS methods

Our VS technique divides the whole protein surface into arbitrary and independent regions (or spots). Spots are specified around alpha-carbons of the protein backbone, so that we can ensure a full scanning of the protein surface. All these spots are independent of each other and, thus, offer great opportunities for data-based parallelization. Docking simulations for each ligand are then performed simultaneously at every protein spot. Actually, the computation places copies of the same ligand at each of those spots. These copies (also known as individuals or conformations) are different from each other as they have a different position and orientation with respect to each spot. Docking simulations search for an optimized *conformation* for both the protein and ligand and the relative orientation between them, such that the free energy (given by the *scoring function*) of the overall system is minimized. Therefore, *METADOCK* uses an optimization procedure where the scoring function, that models the non-bonded interactions between protein and ligand, is minimized throughout the execution. With that in mind, we first introduce the optimization procedure used in *METADOCK* before briefly describing the GPU implementation of the underlying scoring function. The scoring function computation represents more than 95% of the *METADOCK* overall computation time and thus it is offloaded to the GPU to increase overall application performance.

3.1.1 Search method based on a parameterized metaheuristic schema. Algorithm 1 shows the *METADOCK* generic template that we use to generate several metaheuristics for the VS problem. Several authors agree (Vaessens et al., 1998; Raidl, 2006) that many metaheuristics, particularly those based on populations, share six basic functions (see Algorithm 1): *Initialize*, *End condition*, *Select*, *Combine*, *Improve* and *Include*. These functions are like algorithmic templates in which the programmer can provide different implementations, so obtaining different metaheuristics. Population-based metaheuristics maintain and improve multiple candidate solutions (S), and often use population characteristics to guide the search. Local search metaheuristics are also included in the schema, with $|S| = 1$.

Algorithm 1: METADOCK's parameterized metaheuristic schema

```
Initialize(S,ParamIni)
while no End condition(S)do
  Select(S,Ssel,ParamSel)
  Combine(Ssel,Scom,ParamCom)
  Improve(Scom,ParamImp)
  Include(Scom,S,ParamInc)
end while
```

Each of the functions in the algorithm works with various sets or populations (*S, Ssel* and *Scom*). *S* represents the whole population of candidate solutions. In our case, a candidate solution (or individual) is a conformation. Thus, several individuals are selected (*Ssel*) to be combined, so generating a new set of elements, *Scom*. Candidate solutions can also be improved by applying a local search; i.e. moving, translating and/or rotating with respect to each spot.

A further step in developing unified metaheuristics schemes is the introduction of several parameters, i.e. *metaheuristic parameters* (ParamXXX in Algorithm 1), in each of these functions to provide a wider range of metaheuristics. Cutillas-Lozano et al. (2012); Almeida et al. (2013) and Cutillas-Lozano and Giménez (2013) show that the use of a parameterized schema of metaheuristics helps to find satisfactory metaheuristics and to tune them for a particular problem. Several metaheuristics could be evaluated for the problem (each with its corresponding tuning process), and hybrid metaheuristic schemes can also be considered. As a consequence, the selection and tuning for a satisfactory metaheuristic or hybridation for a problem is a complex process, which can require large execution times.

METADOCK is based on that unified parameterized metaheuristic schema and is used for docking simulations. As mentioned in the 'Metaheuristics' subsection and as shown in Algorithm 1, the schema is like a template that defines a set of functions to be implemented for a particular problem. Those functions use several parameters to provide different metaheuristic implementations. Particularly, *METADOCK* uses up to 15 metaheuristic parameters (see Table 2).

Table 2. The fifteen metaheuristic parameters used in the unified parameterized metaheuristic schema for *METADOCK*.

Metaheuristic parameters	Description
INEIni	Number of initial ligand conformations.
PEIIni	Percentage of the best conformations that are improved in the function Initialize.
IIEIni	The intensification of the improvement in the function Initialize.
PBEIni	Percentage of best conformations to be included in the initial set for the next iterations.
PWEIni	Percentage of worst conformations to be included in the initial set for the next iterations.
PBESel	Percentage of the best conformations to be selected for combination.
PWESel	Percentage of the worst conformations to be selected for combination.
PBBCom	Percentage of best-best conformations to be combined.
PWWCom	Percentage of worst-worst conformations to be combined.
PBWCom	Percentage of best-worst conformations to be combined between them.
PEIImp	Percentage of best conformations of the combination to be improved.
IIEImp	The intensification of the improvement of elements generated by combination.
PBEInc	Percentage of best conformations to be included in the reference set.
NIREnd	Maximum number of steps without improvement.
MNIEnd	Maximum number of iterations with or without improvement.

The schema is applied at each spot, with the same metaheuristic parameters in Table 2 (the same metaheuristic) for each spot, and with the basic functions working on different subsets. Below, we briefly summarize details about the implementation of the basic functions used in *METADOCK*.

1. **Initialize** returns an initial set of solutions. *INEIni* conformations are generated randomly for each of the *m* spots. Once they have been generated, a percentage (*PEIIni*) of the initial conformations of each spot is improved. The intensification of the improvement is indicated by the parameter *IIEIni*. Finally, $(PBEIni + PWEIni) * INEIni$ conformations from each spot are selected for the execution of the following functions. *PBEIni* and *PWEIni* represent the percentage of best and worst conformations to be included. The best conformations are those with the best value of the scoring function, and the "worst" conformations are selected from the remaining ones. Indeed, *METADOCK* does not select only the best conformations, so as to diversify the search and avoid falling into local optima.
2. **End condition** determines the stop criteria for *METADOCK*. Either, the maximum number of steps without improvement of the best solution from all the spots, *NIREnd*, or the maximum number of iterations, *MNIEnd*, is used to finish the execution.
3. **Select** chooses some conformations to work with for the next phases. A percentage of the best and worst conformations relative to each spot are selected, i.e. *PBESel* and *PWESel*. Again, to diversify the search and avoid local minima, not only "good" conformations are selected.
4. **Combine** mixes conformations in pairs, depending on their scoring. Three parameters represent the percentage of best–best, worst–worst and best–worst conformations to be combined: *PBBCom, PWWCom* and *PBWCom*. Combinations are performed among conformations at the same spot.
5. **Improve** performs a local search within the neighborhood of some of the conformations previously generated by *Combine*. Two metaheuristic parameters are considered for each spot. First, *PEIImp* defines the percentage of conformations that the local search will be applied to to improve these conformations. Second, *IIEImp* establishes the number of trials for the local search. Hence, *METADOCK* can generate hybrid metaheuristics with different degrees of intensification.
6. **Include** updates the reference set for the next iteration of the schema. The parameter *PBEInc* establishes the percentage of best conformations associated to each spot to be included in its reference set. The rest of the conformations to be included in this set are randomly selected from the remaining conformations at the spot. The inclusion of non-promising conformations contributes to diversify the search, so avoiding stalling in local minima.

3.1.2 GPU implementation of the scoring function. The *METADOCK* scoring function is based on the relevant non-bonded potentials typically used in VS calculations previously described in the 'Background' section. They are the Coulomb, or electrostatic, the Lennard–Jones potentials and the hydrogen-bounds interactions kernel. A discussion about the main terms included in the scoring function is beyond the scope of this paper, but we refer the reader to the work by Wang et al. (2004) for insights.

Algorithm 2 shows the sequential baselines for the Lennard–Jones potential interactions between a receptor and all conformations for a particular ligand (i.e. the set S in algorithm 1) to briefly illustrate the GPU implementation of the scoring function.

Algorithm 2 Sequential baselines for the Lennard–Jones interactions between receptor and ligand.

```
for i=1 to N_CONFORMATION do
  for j=1 to N_ATOMS_RECEPTOR do
    for k=1 to N_ATOMS_LIGAND do
      Energy = 4*epsilon*(term12(j,k) - term6(j,k))

      Scoring + = Energy
    end for
  end for
  S_energy[i] = Scoring
  Scoring = 0
end for
```

Algorithm 3 Method to calculate scoring on GPU.

```
pos = atom_position
individual = get_individual()
for i=1 to r do
  Energy = 4*epsilon*(term12(i,pos) - term6(i,pos))

  Scoring + = Energy
end for
synchronize_threads()
S_energy[individual] = Reduction_atoms_individual()
```

The scoring function of *METADOCK* is implemented in a single kernel where all terms are calculated at the same time. We identify each candidate solution (i.e. conformation) to a CUDA warp, and warps are grouped into blocks depending on the CUDA thread block granularity. Algorithm 3 shows the CUDA kernel for the Lennard–Jones potential. Here, some performance strategies that we have applied to our codes to leverage NVIDIA GPU architectures are introduced.

1. The use of shared memory facilitates the reusability of data by threads of the same block. In our case, the compound is loaded into the shared memory so that threads within the same block can share this information, so saving costly device memory accesses. Thus, each thread calculates the scoring function corresponding to the elements that are associated to each of them, thus increasing the overall application bandwidth.
2. The possibility of using shuffle instructions is available in devices with 3.X or higher compute capability, and their use can improve application performance substantially. These instructions enable information sharing within threads that belong to the same warp without using either shared or device memory.

In addition to the scoring function CUDA kernels, other kernels are also included in *METADOCK* to implement different actions required by main schema functions shown in Algorithm 1. Among them, we may highlight the modification of ligand conformations in *Initialize* and *Improve* functions. The main objective of these kernels is to keep the information in the GPU device memory, so avoiding data-movement through Peripheral Component Interconnect (PCI-Express) bus.

3.2 Scaling to a heterogeneous node

Algorithm 4 shows the parallelization scheme we use to leverage heterogeneous nodes with shared-memory multiprocessors and multiple GPUs. OpenMP is used to manage several CPU threads, where each thread is responsible for controlling a GPU context (we refer the reader to Chapman et al., 2008 for insights). Then, each GPU calculates the scoring function for a set of candidate solutions. In a *homogeneous distribution*, these candidate solutions are equally distributed among GPUs in the form of CUDA thread blocks. However, a high-performance computing (HPC) cluster may have different kinds of devices or even devices within the same family with different compute capabilities or improved instruction set architecture. Thus, the programmer plays a fundamental role in deciding where the code will run on each of these different architectures as long as performance is the main objective.

With this scenario in mind, we introduce a heterogeneity distribution of the workload for our VS methodology. The execution time of each independent execution can differ, as it depends on:

Algorithm 4 Scoring computation on a parameterized metaheuristic for multicore + multiGPU.

```
omp_set_num_threads(number_GPUs)
#pragma omp parallel for
for i=1 to number_GPUs do
  Select_device(Devices[i].id)
  Host_To_GPU(Scom,Stmp)
  Conformations=Devices[i].conformations
  threads=Devices[i].Threadsblock
  stride=Devices[i].stride
  Calculate_scoring<Conformations/threads,threads>
  (Stmp + Devices[i].stride)
  GPU_To_Host(Scom,Stmp)
end for
```

(a) the underlying GPU each metaheuristic instance runs on, which is actually unknown at compile-time;
(b) the number of candidate solutions (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. The peak performance differences shown in the last two rows of Table 1 lead us to believe that there is ample room for improvement. Nevertheless, it is worth noting that these are theoretical peak performances provided by the manufacturer, but only the different speed of the targeted GPUs is the factor for heterogeneity that should be considered to distribute the workload.

We have designed an implementation with two main focuses:

(a) resources accounting through OpenMP processes;
(b) performance monitoring via OpenMP threads.

First, our algorithm defines a master thread which creates as many OpenMP threads as GPUs available on a node, which is easily attained by querying the GPU properties at runtime (using `cudaGetDeviceCount` from the CUDA Application Programming Interface [API]) and NVML (NVIDIA management library). Secondly, a *warm-up* phase is performed to establish performance differences among all targeted GPUs, running the scoring function for a few candidate solutions. This phase measures the execution time of a small number of iterations of the metaheuristic at run-time (once) in order to detect these differences. Importantly, at this stage, the algorithm is not trying to *solve* the docking problem in any meaningful sense, but these runs allow us to calculate the performance differences between GPUs. The execution times in this *warm-up* phase on all GPUs are reduced to obtain the maximum value using `omp reduction`. Thus, the *Percent* parameter is eventually determined, as shown in equation (1)

$$Percent = \frac{Ex.time_{\text{actualGPU}}}{Ex.time_{\text{slowestGPU}}} \quad (1)$$

The slowest GPU will have $Percent = 1$; a GPU two times faster than the slowest GPU would have $Percent = 0.5$, and so on. Each OpenMP thread then calculates the number of conformations it is in charge of for the simulation. The optimum number of threads is also experimentally obtained at this stage to increase the level of parallelism and to maximize processor occupancy.

4 Experimental set-up

Experiments have been conducted in two different heterogeneous systems based on multicore + multiGPU configurations. Below, we show the main characteristics of these computational systems along with the metaheuristic scheme parameters we have used to generate different kinds of metaheuristics to test our implementations. Finally, a description of the target datasets is provided.

4.1 Hardware environment

Two different computational systems are used to run our experiments.

1. *Jupiter*. This is a system with two hexa-cores (12 cores) Intel Xeon E5-2620 at 2 GHz and 32 GB of RAM. The node has up to six GPUs. Two of them are GPUs NVIDIA Fermi Tesla C2075 with 448 CUDA cores (14 streaming multiprocessors and 32 streaming processors per multiprocessor) running at a boost clock of 1.15 GHz, giving a raw processing power of up to 1030 GFLOPS. The memory size is 5.3 GB of GDDR5. The other four GPUs are actually two NVIDIA Fermi GPUs GeForce GTX 590 that each contain two chips in turn. Both have 512 CUDA cores (16 streaming multiprocessors and 32 streaming processors per multiprocessor) running at boost clock of 1.21 GHz, giving a raw processing power of up to 2488.3 GFLOPS.
2. *Hertz*. This has four Intel Xeon X7550 processors running at 2 GHz and plugged into a quad-channel motherboard endowed with 128 GB of DDR3 memory. It has two NVIDIA GPUs. A GPU NVIDIA Tesla Kepler K40c with 2880 CUDA cores (15 streaming multiprocessors and 192 streaming processors per multiprocessor) running at a boost clock of 0.88 GHz, giving a raw processing power of up to 5068 GFLOPS. The memory size is 12 GB of GDDR5. A GPU NVDIA Fermi GeForce GTX 580 with 512 CUDA cores (16 streaming multiprocessors and 32 streaming processors per multiprocessor) running at boost clock of 1.54 GHz, giving a raw processing power of up to 1581 GFLOPS.

In both platforms, gcc 4.8.2 with the -O3 flag was used for compilation on the CPU, and the CUDA toolkit version 6.5 was used for compilation on the GPU.

4.2 Benchmarking

4.2.1 Metaheuristics. The template shown in Algorithm 1 allows experimentation with several basic metaheuristics and combinations/hybridations of them. So, it can be used for the selection and tuning of satisfactory metaheuristics for the problem we are working with, and, furthermore, the parallelization of the schema facilitates the parallelization and the determination of the best parallelism configurations for different metaheuristics and combinations. In the experiments, we consider up to six metaheuristics of different characteristics for comparison purposes. The performance, efficiency and quality are evaluated on various hardware configurations.

Table 3 shows the values of the metaheuristic parameters for the six metaheuristics considered in the experimental section. The first metaheuristic (M1) is a *genetic algorithm* with a population of 500 individuals (*INEIni* = 500) for each spot in the receptor at the

Table 3. Parameter setting used for experimentation.

	COMBINATIONS					
	M1	M2	M3	M4	M5	M6
INEIni	500	50	300	20	100	10
PEIIni	0	50	100	100	50	100
IIEIni	0	20	100	100	50	200
PBEIni	8	20	0	100	20	100
PWEIni	0	20	0	100	20	100
PBESel	100	100	0	50	100	100
PWESel	0	100	0	50	100	0
PBBCom	100	100	0	100	50	100
PWWCom	0	25	0	50	10	0
PBWCom	0	50	0	10	25	0
PEIImp	0	50	0	50	100	100
IIEImp	0	20	0	20	10	200
PBEInc	100	100	0	80	50	80

initialization stage. Only the best 40 individuals keeps on going with the computation ($PBEIni = 8$) after initialization. After that, all of the best candidates are selected, combined and included for the next iteration ($\{PBESel, PBBCom, PBEInc\} = 100$), and no local search is included to improve the conformations. The second metaheuristic (M2) is also an evolutionary method but, in this case, its computation is similar to a *scatter search* algorithm. It works with a reference set of 50 individuals, many elements are improved after they have been generated, initially or by combination, through local search in the neighborhood of each element to obtain better solutions, and combinations between the worst or best and worst elements are included. After the initialization phase, all the selected elements are combined with each other, and a further improvement is applied to half of them. The metaheuristic M3 is a neighborhood-based metaheuristic, where local searches are applied to candidate solutions for a large initial set, thus the initialization stage is the only stage executed in this metaheuristic, and it can be seen as a GRASP method. The metaheuristics M4, M5 and M6 are combinations of the above metaheuristics, changing the parameter values of combination and improvement, to try to get better results. For example, for larger sets, fewer elements are improved or the intensification of the improvements is lower. The unified schema allows us to experiment with several configurations to determine the best metaheuristic for our problem, but this study is beyond the scope of the paper.

4.2.2 Accuracy of the predictions. It is necessary to measure the performance of VS methods in terms of accuracy of the predictions they yield. One common approach in this research area is to process datasets of known compounds and to check the ability of VS methods in classifying them. A set of benchmark instances from the DUDs was used for this (DUD, 2006). The DUDs dataset contains, for 40 sets of protein targets, a set of active ligands, decoys (ligands known to be not active) and the structural information (X-ray crystallographic studies) about a ligand co-crystallized with each respective protein. The decoy compounds have similar physical properties to the active ligands but dissimilar topology, and were designed in order to make the classification task difficult. In this work, three different targets are selected from the DUDs (see Table 4). These targets have different numbers of atoms (13,261, 7158 and 3419 respectively) that require different amount of memory in the simulation.

4.2.3 Performance datasets. Data sets in Table 4 are also used to evaluate computational performance of our parallelization strategies. The docking simulations, in this case, are performed with the different receptors (GPB, SRC and COMT) and their corresponding crystallography ligands. They have different sizes to test scalability on the different platforms targeted. Our scoring function calculates the interaction between all atoms from the protein (*nrec*) and all atoms from the ligand (*nlig*). This is performed at each spot where a number of individuals (the same ligand with different spatial locations) runs in parallel. Therefore, a *METADOCK* simulation performs $Spots * Individuals$ simulations at the same time, and each calculates $nrec * nlig$ interactions. For instance, M1 works with 500 conformations at the same time in the initialization stage. The number of interactions for GPB would be $13,261 * 52 * 500 = 344,786,000$ at each spot, having up to 813 spots. The number of spots, receptor atoms and crystallography ligand atoms are listed in Table 4. Indeed, memory requirements for our simulations are directly related to this formula. The number of bytes for these benchmarks is shown.

Table 4. Number of decoys and active ligands of the benchmark targets from the DUDs database. We also include the size of the receptor and crystallography ligand (number of atoms) used for performance comparison, and the number of interactions for each individual.

Targets	Number of spots	Decoy ligands	Active ligands	Receptor size (number of atoms)	Crystallography ligand size (number of atoms, KB)	Target memory size (KB)
GPB	813	2139	52	13,261	(52, 1.21)	190
SRC	452	6319	159	7158	(67, 1.57)	293
COMT	214	468	11	3419	(29, 0.67)	543

Table 5. Execution time (s) and speed-up for the metaheuristics, executing the targets COMT, SRC, GPB in Hertz.

Metaheuristics	Multicore CPU (s)	Speed-up GPU K40c vs. Multicore CPU	Speed-up homogeneous distribution vs. Multicore CPU	Speed-up hardware-feature distribution vs. multicore CPU	Speed-up heterogeneous distribution vs. multicore CPU
DUD:COMT target					
M1	180.19	24.32	27.35	29,90	37.91
M2	223.11	23.88	28.23	29.51	36.92
M3	2942.99	28.21	35.08	36.94	46.56
M4	284.31	23.35	26.56	29.88	35.56
M5	402.25	24.33	29.35	30.67	38.49
M6	1758.24	24.64	30.61	31.79	39.75
DUD:SRC target					
M1	1025.19	23.21	26.12	28.82	37.64
M2	1269.99	22.66	27.97	28.89	36.77
M3	15,042.33	23.03	26.66	29.46	37.98
M4	1616.11	22.26	27.62	30.16	38.05
M5	2287.21	22.81	23.25	28.11	37.16
M6	10,015.06	22.94	24.51	29.27	37.57
DUD:GPB target					
M1	1479.51	28,56	31,71	36.19	45.61
M2	1831.24	28.94	35.78	36.88	47.29
M3	21,758.21	29.47	37.42	37.75	48.57
M4	2335.44	29.07	36.43	37.06	47.39
M5	3303.33	29.11	36.51	37.32	47.96
M6	14,439.52	29.34	37.07	38.26	48.45

5 Experimental results

This section shows the experimental results for *METADOCK* on multicore and multiGPU systems. The main objective of these experiments is two-fold. First, we analyze our load distribution strategies to improve performance on heterogeneous nodes based on CPU and MultiGPU. Second, we study the quality of the results with several chemical compounds to discuss the effectiveness of our approach.

5.1 Performance results

Given that our technique establishes the experimental set-up dynamically, the results shown below are platform-dependent. Therefore, we provide an exhaustive analysis on the two heterogeneous systems previously described. Tables 5 and 6 show the execution time (single-point precision execution) and relative speed-up factor among different implementations and metaheuristic configurations for each target dataset in both systems (see Table 4 for the dataset description and Table 3 for the schema configuration). They show the execution times for OpenMP implementation on multicore CPUs as a reference for the improvements.

Table 5 shows performance numbers on the Hertz system, in which the computational capability of the GPUs available are quite different (Fermi and Kepler architecture). First, it shows the speed-up factor for a single GPU (Tesla K40c) versus multicore CPU. This is in the range of 22–29 $\times$. The implicit data parallelism in this problem benefits greatly from the GPU horsepower. Our CUDA implementations take advantage of data-locality through tilling implementation via shared memory, which benefits the receptor scalability. Then, computing with several GPUs come into the scene.

Table 6. Execution time (s) and speed-up for the metaheuristics, executing the targets COMT, SRC, GPB in Jupiter.

Metaheuristics	multicore CPU (s)	Speed-up GPU Tesla C2075 vs. multicore CPU	Speed-up homogeneous distribution vs. multicore CPU	Speed-up heterogeneous distribution vs. multicore CPU
DUD:COMT target				
M1	140.48	10.42	38.72	39.35
M2	193.67	13.81	39.88	43.55
M3	1911.52	9.62	53.25	54.16
M4	209.56	9.59	33.86	34.43
M5	262.65	8.55	34.81	35.51
M6	1379.93	10.39	53.61	53.89
DUD:SRC target				
M1	639.32	7.45	36.43	39.35
M2	678.41	7.86	37.89	40.97
M3	10,670.57	8.55	49.34	49.41
M4	1150.81	8.49	40.21	43.75
M5	1574.79	8.27	43.08	44.62
M6	7422.66	8.93	50.08	50.27
DUD:GPB target				
M1	910.42	9.24	45.19	46.21
M2	964.82	9.44	49.01	53.01
M3	15,050.81	10.75	60.98	62.28
M4	1654.85	10.94	52.88	57.58
M5	2449.27	11.47	58.71	62.47
M6	10,186.09	10.94	61.57	62.41

First, our version using a homogeneous distribution of the workload (i.e. assigning the same amount of workload to each GPU) is in the range 26–35× speed-up factor compared to multicore CPU for small-medium molecules. This speed-up for large molecules is up to 37×. This means an additional 1.27× speed-up factor by adding a GPU in some cases. Furthermore, these data show that metaheuristic parameters are important for the performance, that is enhanced when the number of combined and improved individuals increases. This is clearly shown in metaheuristic M3 for small-medium compounds where all initial population generated is improved, and metaheuristics M3 and M6 with the largest compound. In this case, in M6 another large improve phase for all combined elements produces a significant increase in computation.

A further step to increase performance is to figure out a new distribution strategy for load balancing. A straightforward approach is to distribute the workload according to the hardware features (i.e. peak performance). The theoretical peak performance for Tesla K40c and GeForce GTX 580 are 5068 and 1581 GFLOPS respectively. This means a 3.2× speed-up factor in favor of Tesla K40c. The strategy called hardware-feature distribution in Table 5 schedules the workload based on this idea. Some performance gains are reported compared to the homogeneous approach although they are not remarkable. Peak performance reported in technical specifications is reported under ideal conditions (e.g. the execution of only FMA instructions, not memory latencies, etc) and thus application performance will be determined at runtime. The last column in Table 5 shows the speed-up factor of our heterogeneous distribution strategy compared to multicore. The results are better than with the hardware-feature distribution, reaching up to 1.43× speed-up factor on average compared to a homogeneous approach.

Table 6 shows performance numbers in Jupiter. The GPUs available in Jupiter (up to six) are based on the same architecture (code-named Fermi) and, thus, the overall GPU heterogeneity in this system is very low. Performance numbers in this system raise similar conclusions to those in Hertz. Our heterogeneous distribution strategy of the workload here shows fewer benefits. Indeed, this means that GPU heterogeneity on a computing node makes load balancing strategy mandatory to get peak performance. Finally, Tables 5 and 6 show that the speed-up increases with the problem size, thus proving the scalability of the multiGPU versions.

We report higher speed-up ratios whenever we increase either the level of intensification in a local search or the size of the reference set. Metaheuristics M2 and M3 contain different values for local search in the neighborhood of each conformation with the same number of initial elements. In all the executions with the three compounds, more intensive searches provide higher speed-up ratios, and they are even higher in multiGPU environments. The M4 metaheuristic studies the extreme case in which only a local search is applied on a very large number of elements, achieving the best speed-up ratios in comparison with the distributed metaheuristics.

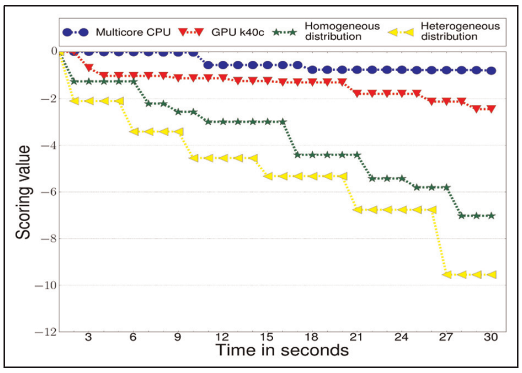


Figure 2. Scoring function evolution within 30 s time-frame. Metaheuristic *M6* and target COMT on Hertz.

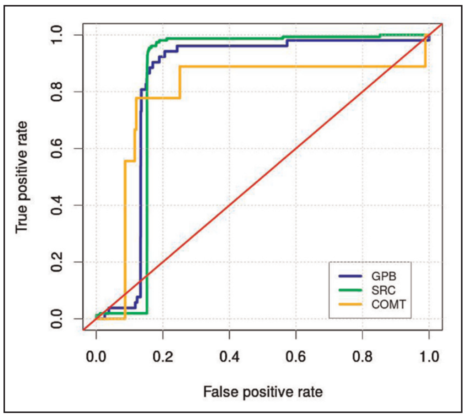


Figure 3. ROC plots for three targets of the DUDs database: GPB (blue), SRC (green) and COMT (orange).

Finally, Figure 2 shows the scoring function evolution during a simulation of 30 s for a docking simulation with *METADOCK* with *M*6 parameter configuration and the target COMT. The techniques with the highest performance obtains better quality results as it performs more computations within the same time-frame. So, the efficient exploitation of parallelism facilitates obtaining satisfactory results at shorter times.

5.2 Quality results: Accuracy of the predictions

As mentioned in the benchmark section, the quality of a docking program is usually measured through its ability to differentiate between active ligands (which could evolve into drugs) and decoys.

The ROC (receiver operating characteristic) curve is a binary classification model which is frequently applied for the analysis of the accuracy of virtual screening methods. This model allows us to compare how good a method is when selecting active ligands and discarding decoys. In order to validate our algorithm, Figure 3 shows the ROC curves obtained for three different targets conveniently selected from the DUDs database. The R language package ROCR (published by Sing et al., 2005) was used to perform the analyses over generated docking results. The y-axis shows the fraction of true positives (TPF), and the fraction of false positives (FPF) is shown on the x-axis. A diagonal line would indicate that the classifier works randomly. The value of AUC (area under curve) is 1.0 when true positive rate (TPR) is always 1 and false positive rate (FPR) equal to 0 (the ideal case). The results shown in Figure 3 have been obtained with *METADOCK* with the combination of metaheuristic parameters M4 in Table 2, and they are representative of those obtained with the other configurations (results not shown). The AUC values obtained for the DUDs datasets GPB, SRC and COMT are 0.838, 0.842 and 0.747, respectively. AUC values greater than 0.65 can be considered to be adequate, with values closer to 1 indicating better specificity and sensitivity of the method in detecting decoys. The values obtained with *METADOCK* are satisfactory, and the efficient exploitation of heterogeneous computing systems facilitates experimentation with moderate execution times.

6 Conclusions and future work

VS methods are computational techniques that aid or complement the experimental drug discovery process but they are very computationally demanding applications. This paper introduces a VS technique, called *METADOCK*, based on a unified parameterized metaheuristic schema that is able to generate a wide variety of metaheuristics, and so provides a fully flexible framework for drug discovery, and thus facilitates enhanced performance and prediction accuracy. *METADOCK* is tailored for heterogeneous computers based on CPU and multiple GPUs. This heterogeneity may limit acceleration which is not acceptable in such challenging applications. In this work the heterogeneity of the system is exploited at two levels;

1. CPU-GPU heterogeneity; with an implementation in which some parts of the computation are carried out on the CPU side while the most costly parts are assigned to the GPUs;
2. GPU-GPU heterogeneity, i.e. GPUs with different characteristics, including different architectures, numbers of cores and compute capabilities, providing a load balancing technique based on the application performance on the targeted GPUs.

The efficient exploitation of the heterogeneous system provides good speed-ups, which increase with the problem size. Our strategy is particularly useful for non-deterministic algorithms and stochastic behaviors, where real-time constraints must be fulfilled. Performance gains are translated into quality improvements that are a decisive factor in virtual screening. AUC results obtained with *METADOCK* support that its parallel, metaheuristic-based schema makes it a useful tool in the early stages of drug discovery.

For future work and to tackle larger problems or for better solutions with limited execution times, it could be convenient to adapt our virtual screening method to more complex hardware systems, comprising of several computational nodes working together with the message-passing paradigm, and each node with several computational components, e.g. multicore, heterogeneous GPUs and many integrated cores (MICs). In this scenario, multicore processors could also be used to compute part of the simulation instead of only monitoring the GPU. Regarding the quality of the results, many other types of scoring functions are still to be explored, including metal and aromatic interactions, and inclusion of implicit solvation effects. This field seems to offer a promising and potentially fruitful area of research.

Acknowledgements

We thank NVIDIA for hardware donation under GPU Educational Center 2014-2016 and Research Center 2015-2016.

Funding

This work was supported by the Fundación Séneca, Agencia Regional de Ciencia y Tecnología, Región de Murcia, (grant number 18946/JLI/13) and by the Spanish MEC and European Commission FEDER (grant numbers TIN2015-66972-C5-3-R and TIN2016-78799-P, AEI/FEDER, UE).

References

Almeida F, Giménez D, López-Espín JJ, et al. (2013) Parameterised schemes of metaheuristics: Basic ideas and applications with genetic algorithms, scatter search and GRASP. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans* 43(3): 570–586.

Asanovic K, Bodik R, Catanzaro BC, et al. (2006) The landscape of parallel computing research: A view from Berkeley. Report, Report no. UCB/EECS-2006-183, University of California at Berkeley, USA, Electrical Engineering and Computer Sciences.

Austin T (2015) Bridging the Moore's law performance gap with innovation scaling. In: *Proceedings of the 6th ACM/SPEC international conference on performance engineering*, Austin, Texas, 31 January–4 February, pp.1. ACM.

Bianchi L, Dorigo M, Gambardella LM, et al. (2009) A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing: An International Journal* 8(2): 239–287.

Blum C, Puchinger J, Raidl GR, et al. (2011) Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing* 11(6): 4135–4151.

Blum C and Roli A (2003) Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)* 35(3): 268–308.

Cecilia JM, García JM, Nisbet A, et al.(2013) Enhancing data parallelism for ant colony optimization on GPUs. *Journal of Parallel and Distributed Computing* 73(1): 42–51.

Chapman B, Jost G and Van Der Pas R (2008) *Using OpenMP: Portable Shared Memory Parallel Programming*, vol. 10. Massachusetts, United States: *MIT* press.

Cutillas-Lozano JM and Giménez D (2013) Determination of the kinetic constants of a chemical reaction in heterogeneous phase using parameterized metaheuristics. In: *7th Workshop on Computational Chemistry and Its Applications on International Conference on Computational Science (ICCS 2013)*, Barcelona, Spain, 5–7 June.

Cutillas-Lozano LG, Cutillas-Lozano JM and Giménez D (2012) Modeling shared-memory metaheuristic schemes for electricity consumption. In: *Distributed computing and artificial intelligence – 9th international conference*, Salamanca, Spain, 28–30 March, pp.33–40.

De Michell G and Gupta RK (1997) Hardware-software co-design. *Proceedings of the IEEE* 85(3): 349–365.

Dolezal R, Ramalho TC, França TC, et al. (2015) Parallel flexible molecular docking in computational chemistry on high performance computing clusters. In: *Computational Collective Intelligence*, vol 9330. Berlin, Heidelberg: Springer, pp.418–427.

Dréo J, Pétrowski A, Siarry P, et al. (2005) *Metaheuristics for Hard Optimization*. New York, United States: Springer Science & Business Media.

Drews J (2000) Drug discovery: A historical perspective. *Science* 287(5460): 1960–1964.

DUD (2006) Directory of Useful Decoys. Available at: http://dud.docking.org/ (accessed 4 October 2016).

Ewing TJA, Makino S, Skillman AG, et al. (2001) DOCK 4.0: Search strategies for automated molecular docking of flexible molecule databases. *Journal of Computer-Aided Molecular Design* 15(5): 411–428.

Franco AA (2013) Multiscale modelling and numerical simulation of rechargeable lithium ion batteries: Concepts, methods and challenges. *RSC Advances* 3(32): 13027–13058.

Friesner RA, Banks JL, Murphy RB, et al. (2004) Glide: A new approach for rapid, accurate docking and scoring: Method and assessment of docking accuracy. *Journal of Medicinal Chemistry* 47(7): 1739–1749.

Glover F and Kochenberger GA (2003) *Handbook of Metaheuristics*. New York, United States: Kluwer Academic Publishers.

Guerrero GD, Cebrián JM, Pérez-Sánchez H, et al. (2014) Toward energy efficiency in heterogeneous processors: Findings on virtual screening methods. *Concurrency and Computation: Practice and Experience* 26(10): 1832–1846.

Hromkovič J (2003) *Algorithmics for Hard Problems*. 2nd ed. Berlin: Springer.

Huang SY and Zou X (2010) Advances and challenges in protein-ligand docking. *International Journal of Molecular Sciences* 11(8): 3016–3034.

Imbernón B, Cecilia JM and Giménez D (2016) Enhancing metaheuristic-based virtual screening methods on massively parallel and heterogeneous systems. In: *Proceedings* *of the 7th international workshop on programming models and applications for multicores and manycores*, Barcelona, Spain, 12–16 March, pp.50–58. ACM.

Irwin JJ and Shoichet BK (2005) ZINC–a free database of commercially available compounds for virtual screening. *Journal of Chemical Information and Modeling* 45(1): 177–182.

Jain AN (2006) Scoring functions for protein-ligand docking. *Current Protein and Peptide Science* 7(5): 407–420.

Jorgensen WL (2004) The many roles of computation in drug discovery. *Science* 303: 1813–1818.

Kirk DB and Wen-Mei WH (2013) *Programming Massively Parallel Processors: A Hands-On Approach*. Boston, MA, USA: Morgan Kaufmann Publishers Inc.

Kitchen DB, Decornez H, Furr JR, et al. (2004) Docking and scoring in virtual screening for drug discovery: Methods and applications. *Nature Reviews Drug Discovery* 3(11): 935–949.

Kuntz SK, Murphy RC, Niemier MT, et al. (2001) Petaflop computing for protein folding. In: *Proceedings of the tenth SIAM conference on parallel processing for scientific computing*, Porstmouths, Virginia, USA, 12–14 March, pp. 12–14.

Li Y, Han L, Liu Z, et al. (2014a) Comparative assessment of scoring functions on an updated benchmark: 2. evaluation methods and general results. *Journal of chemical information and modeling* 54(6): 1717–1736.

Li Y, Liu Z, Li J, et al. (2014b) Comparative assessment of scoring functions on an updated benchmark: 1. Compilation of the test set. *Journal of Chemical Information and Modeling* 54(6): 1700–1716.

Lionta E, Spyrou G K, Vassilatis D, et al. (2014) Structure-based virtual screening for drug discovery: Principles, applications and recent advances. *Current Topics in Medicinal Chemistry* 14(16): 1923–1938.

López-Camacho E, García-Godoy MJ, García-Nieto J, et al. (2015) Solving molecular flexible docking problems with metaheuristics: A comparative study. *Applied Soft Computing* 28: 379–393.

McIntosh-Smith S, Price J, Sessions RB, et al. (2014) High performance in silico virtual drug screening on many-core processors. *International Journal of High Performance Computing Applications* 29(2): 119–134.

Michalewicz Z and Fogel DB (2002) *How to Solve It: Modern Heuristics*. Berlin: Springer.

Minetti G, Alba E and Luque G (2008) Seeding strategies and recombination operators for solving the DNA fragment assembly problem. *Information Processing Letters* 108(3): 94–100.

Morris GM, Goodsell DS, Halliday RS, et al. (1998) Automated docking using a Lamarckian genetic algorithm and an empirical binding free energy function. *Journal of Computational Chemistry* 19(14): 1639–1662.

NVIDIA Corporation (2017) *NVIDIA CUDA C Programming Guide 8.0*. Available at: http://docs.nvidia.com/cuda/cuda-c-programming-guide/

Raidl GR (2006) A unified view on hybrid metaheuristics. In: *Hybrid Metaheuristics*. Heidelberg, Berlin: Springer, pp.1–12.

Rester U (2008) From virtuality to reality–virtual screening in lead discovery and lead optimization: A medicinal chemistry perspective. *Current Opinion in Drug Discovery & Development* 11(4): 559–568.

Rollinger JM, Stuppner H and Langer T (2008) Virtual screening for the discovery of bioactive natural products. In: *Natural Compounds as Drugs Volume I*. Basel: Birkhäuser Verlag, pp.211–249.

Rozenberg G, Bäck T and Kok JN (2011) *Handbook of Natural Computing*. Heidelberg, Berlin: Springer.

Sánchez-Linares I, Pérez-Sánchez H, Cecilia JM, et al. (2012) High-throughput parallel blind virtual screening using BINDSURF. *BMC Bioinformatics* 13(Suppl 14): S13.

Schneider G (2002) Virtual screening and fast automated docking methods. *Drug Discovery Today* 7: 64–70.

Sing T, Sander O, Beerenwinkel N, et al. (2005) Rocr: Visualizing classifier performance in r. *Bioinformatics* 21(20): 3940–3941.

Sodani A, Gramunt R, Corbal J, et al. (2016) Knights landing: Second-generation intel xeon phi product. *IEEE Micro* 36(2): 34–46.

Song WJ, Mukhopadhyay S and Yalamanchili S (2016) Amdahl's law for lifetime reliability scaling in heterogeneous multicore processors. In: *2016 IEEE international symposium on high performance computer architecture (HPCA)*, pp.594–605. IEEE.

Top500 (2016) Top500 supercomputer site. Available at: `http://www.top500.org/` (accessed 4 October 2016).

Vaessens RJ, Aarts EH and Lenstra JK (1998) A local search template. *Computers & Operations Research* 25(11): 969–979.

Wang J, Deng Y and Roux B (2006) Absolute binding free energy calculations using molecular dynamics simulations with restraining potentials. *Biophysical Journal* 91(8): 2798–2814.

Wang R, Lu Y, Fang X, et al. (2004) An extensive test of 14 scoring functions using the PDBbind refined set of 800 protein-ligand complexes. *Journal of Chemical Information and Computer Sciences* 44(6): 2114–2125.

Yuriev E and Ramsland PA (2013) Latest developments in molecular docking: 2010–2011 in review. *Journal of Molecular Recognition* 26(5): 215–239.

Yuriev E, Agostino M and Ramsland PA (2011) Challenges and advances in computational docking: 2009 in review. *Journal of Molecular Recognition* 24(2): 149–164.

Zhou Z, Felts AK, Friesner RA, et al. (2007) Comparative performance of several flexible docking programs and scoring functions: Enrichment studies for a diverse set of pharmaceutically relevant targets. *Journal of Chemical Information and Modeling* 47(4): 1599–1608.

Author Biographies

Baldomero Imbernón received his BS degree in Computer Science from Catholic University of Murcia (Spain, 2013) and MS degree in New Technologies in Computer Science in University of Murcia (Spain, 2015) specializing in high performance architectures and supercomputing. In the last two years, he has authored several journal papers in in the areas of bioinformatics and high performance computing. He is a predoctoral researcher at the Catholic University of Murcia (Spain).

José M Cecilia received his BS degree in Computer Science from the University of Murcia (Spain, 2005), his MS degree in Computer Science from the University of Cranfield (United Kingdom, 2007), and his PhD 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 the 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 on applying these techniques to challenging problems in the fields of Science and Engineering. Now, he is working as an 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 program.

Horacio Pérez-Sánchez has contributed to computational and physical chemistry with several methodological developments, implementing them in high performance computing (HPC) architectures (supercomputers, GPUs, cell processor) so that they can be accessed by other researchers either as standalone software packages or web tools. Some of them have been commercialized directly with industry. He has applied these developments directly in drug discovery projects (anticoagulants, Parkinson, Alzheimer, cancer, Fabry disease, anti-inflammatories, etc.) or for the analysis and interpretation of experimental data (encapsulation processes, ion channels, nutraceuticals, etc.). Main results have been published in 56 ISI indexed scientific articles (eight as first author and 20 as corresponding author), two international (and licensed) patents, 50 contributions to international conferences with 45 peer reviewed computer engineering and bioinformatics conference proceedings (six as first author and 21 as corresponding author) and 17 invited talks. He has been awarded with the 2016 HiPEAC technology transfer award and participated in 30 research projects attracting around € 725,000 (being Principal Investigator in 16 of them, with total funding € 327,000), and established four contracts with industry and academy.

Domingo Giménez is a professor in the Computer Science Department at the University of Murcia, Spain. He has been a faculty member of the university since 1988, where he teaches algorithms and parallel computing. He received his degree in Mathematics from the University of Murcia in 1982, and his PhD in Computer Science from the Polytechnic University of Valencia in 1995. His research interests include scientific applications of parallel computing, matrix computation, scheduling and software auto-tuning techniques.

2.3 Exploiting multilevel parallelism on a many-core system for the application of hyperheuristics to a molecular docking problem

Título	*Exploiting multilevel parallelism on a many-core system for the application of hyperheuristics to a molecular docking problem*
Autores	(Orden Alfabético) José M. Cecilia, José-Matías Cutillas, Domingo Giménez y Baldomero Imbernón
Revista	The Journal of Supercomputing
Año	2017
Estado	Publicado

Contribución del Doctorando
Baldomero Imbernón Tudela, declara ser el principal autor y el principal contribuidor del artículo *Exploiting multilevel parallelism on a many-core system for the application of hyperheuristics to a molecular docking problem.*

J Supercomput
DOI 10.1007/s11227-017-1989-7

Exploiting multilevel parallelism on a many-core system for the application of hyperheuristics to a molecular docking problem

José M. Cecilia[1] · José-Matías Cutillas-Lozano[2] · Domingo Giménez[2] · Baldomero Imbernón[1]

Abstract The solution of Protein–Ligand Docking Problems can be approached through metaheuristics, and satisfactory metaheuristics can be obtained with hyperheuristics searching in the space of metaheuristics implemented inside a parameterized schema. These hyperheuristics apply several metaheuristics, resulting in high computational costs. To reduce execution times, a shared-memory schema of hyperheuristics is used with four levels of parallelism, two for the hyperheuristic and two for the metaheuristics. The parallel schema is executed in a many-core system in "native mode," and the four-level parallelism allows us to take full advantage of the massive parallelism offered by this architecture and obtain satisfactory fitness and an important reduction in the execution time.

Keywords Parameterized metaheuristic schemas · Parallel metaheuristics · Hyperheuristics · Many-core systems · Protein–Ligand Docking Problem

Domingo Giménez
domingo@um.es

José M. Cecilia
jmcecilia@ucam.edu

José-Matías Cutillas-Lozano
josematias.cutillas@um.es

Baldomero Imbernón
bimbernon@ucam.edu

1 Polytechnic School, Catholic University of San Antonio of Murcia (UCAM), Murcia, Spain

2 Department of Computing and Systems, University of Murcia, Murcia, Spain

1 Introduction

Bioinformatics and computational biology problems are generally highly computationally demanding, which propitiates the application of parallel techniques in these fields [16,18–20] in various types of computational systems [11,15]. In particular, virtual screening (VS) methods [9] are applied to the discovery of new drugs and they can benefit from the efficient exploitation of parallel systems.

VS methods analyze large libraries of small molecules (*ligands*) to search for those structures which are most likely to bind to a drug target, typically a protein receptor or enzyme [14]. These libraries of chemical compounds can have millions of ligands [8]. Actually, the analysis of larger databases exponentially increases the chances of generating hits.

The computational complexity of VS methods is determined by two main parameters: the size of the database to be analyzed and the accuracy of the chosen VS method. Fast VS methods with atomic resolution require some minutes per ligand [22]. In contrast, molecular dynamics approaches can require up to thousands of hours per ligand [17]. Therefore, the main bottleneck for the success of VS methods is the lack of computational resources or, in other words, there is a need for highly efficient applications that leverage emergent, high-performance computing architectures [4].

We use a molecular *docking*-based computational methodology that attempts to predict non-covalent binding of molecules or, more frequently, of a macromolecule (receptor) and a small molecule (ligand). The main goal is to predict the bound conformations and the binding affinity, i.e., the strength of association. In this Protein–Ligand Docking Problem (PLDP), a scoring function is optimized to obtain the position at which a ligand best matches a given protein (Fig. 1). Different scoring functions can be used [21]. In this work, the scoring function is computed through the Lennard-Jones potential, obtained as the sum of the interactions of each atom, i, of an active site of the protein with each atom, j, of the ligand:

$$V(i,j) = 4\epsilon\left(\left(\frac{\sigma}{r(i,j)}\right)^{12} - \left(\frac{\sigma}{r(i,j)}\right)^{6}\right) \tag{1}$$

where σ and ϵ are empirical constants of the model, and $r(i,j)$ is the distance between atoms i and j.

The PLDP problem is an NP-hard problem [1], and therefore, metaheuristics are well-suited to provide good results in a reasonable time-frame. It can be seen as a search problem with up to six degrees of freedom that globally minimize the scoring function. These degrees of freedom are based on the ligand movements (translation and rotation) to look for the best conformation to fit with highest fitness into the receptor. The values of the movements and rotations of the ligand can be approached with metaheuristics, and hyperheuristics can be used for the selection of satisfactory metaheuristics for the PLDP, so facilitating and automating their application and improving the fitness. In [7], we analyzed the application of metaheuristics to the PLDP in heterogeneous GPUs. The GPU parallelism was exploited only for the computation of the fitness (Eq. 1) of the elements in the reference set. Here, the parallelism is exploited at a higher level, combining parallelism in the application of hyperheuristics and metaheuristics.

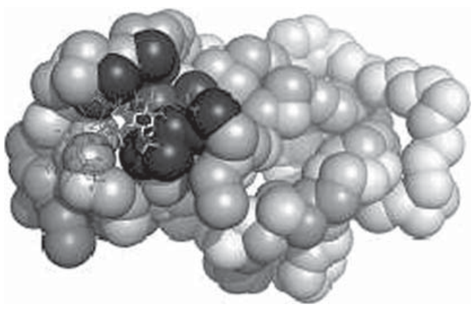


Fig. 1 Binding a ligand and a receptor. The *gray tones* represent different spots where binding can take place [10]

Hyperheuristics select satisfactory metaheuristics or build new ones by combining basic metaheuristics for a particular problem [5,12]. In our approach, we use a parameterized schema of metaheuristics [3], which facilitates the selection of metaheuristics or their combinations by selecting numerical values for metaheuristic parameters in the schema. It considers a set of basic functions whose instantiation determines the particular metaheuristic being implemented. When hyperheuristics are used to select satisfactory metaheuristics, the execution time increases significantly, so high-performance computing strategies are compulsory.

This paper analyzes the use of massive parallelism in a MIC (Xeon Phi) for exploiting multilevel parallelism in hyperheuristics working on top of parameterized metaheuristics as applied to the PLDP. The multilevel approach allows the application of parallelism at hyperheuristic and metaheuristic levels, and the optimal combination of the number of threads to use at each level should be selected. The major contributions are:

1. We design a hyperheuristic schema for the PLDP to find the best metaheuristic configuration for this problem.
2. Metaheuristic search through hyperheuristic schema is computationally demanding, so we parallelize this process on Intel Xeon Phi architecture.
3. An extensive evaluation is provided in terms of both performance and fitness prediction.
4. To the best of our knowledge, this is the first time that PLDP is tackled with hyperheuristics, so offering multilevel parallelism.

The rest of the paper is organized as follows. Section 2 gives the basis of the shared-memory, parameterized metaheuristic schema which is used for the development of hyperheuristics in a MIC. Section 3 discuses the experimental results obtained with the application of hyperheuristics to an instance of the PLDP. Section 4 concludes and outlines possible research directions.

2 A parallel, parameterized metaheuristic schema

Our approach for parallelizing metaheuristics consists on the parallelization of a unified parameterized metaheuristic schema (Algorithm 1). The hyperheuristics and the metaheuristics they search for are implemented with the same schema, and so they are parallelized in the same way. In the schema, $ParamX$ represents the metaheuristic parameters and *ThreadsX* the parallelism parameters for each basic function.

Algorithm 1 Parameterized shared-memory metaheuristic schema

```
Initialize(S,ParamIni,ThreadsIni) //Generate initial set and improve some elements
while (not EndCondition(S,ParamEnd)) do
  SS=Select(S,ParamSel) //Select elements for combination
  SS1=Combine(SS,ParamCom,ThreadsCom) //Combine pairs of elements selected
  SS2=Improve(SS1,ParamImp,ThreadsImp) //Improve and diversify some elements
  S=Include(SS2,ParamInc,ThreadsInc) //Update the reference set
end while
```

The basic functions in the schema can be implemented in different ways, and the number of parameters and their meanings can change. We are not interested here in an in-depth discussion of the metaheuristic and the parallelism parameters. We refer the reader to [2] for insights on parallelism parameters and [3] for insights on metaheuristic parameters.

Our implementation considers eighteen metaheuristic parameters (five in the initialization, two in the selection, three for combination, six for the improvement and two for the inclusion). Some elements are initially generated (parameter INEIni), and some are selected for the successive steps (FNEIni). A certain percentage of the generated elements (PEIIni) is improved with a given intensity (IIEIni) and with a short tabu memory (STMIni). The corresponding parameters are used for the improvement of elements after combination (PEIImp, IIEImp and SMIImp) and diversification (PEDImp, IDEImp and SMDImp). In each iteration of the algorithm, some elements are selected from the best (NBESel) and the worst (NWWSel) ones, and combinations between pairs of best, worst and best-worst elements (NBBCom, NWWCom and NBWCom) are made. Some of the best elements are selected for the next iteration (NBEInc), with the use of a long-term tabu memory (LTMInc).

Some of the routines are parallelized with only one level of parallelism, while in other functions two levels are used. So, the parallelism parameters of each routine correspond to the number of threads at each level. In our implementation, the one-level parallel routines are in the initial generation of the reference set (number of threads, TGEIni), the combination of pairs (TCPCom threads) and the inclusion of elements in the reference set (TIEInc threads); and the two-level routines are those improving elements: after the initial generation (pair of threads, TI_Ini), elements of the reference set (TR_Imp), those obtained by combination (TC_Imp), and those selected to be diversified from the reference set (TR_Div) or from those obtained by combination (TC_Div).

Hyperheuristics can be developed on top of the parameterized schema, and they search automatically in the space of metaheuristics for a satisfactory metaheuristic for a particular problem. The elements for the hyperheuristic are vectors of metaheuristic parameters (metaheuristics), and the fitness for an element is obtained through the application of the metaheuristic it represents to the problem on hand (Fig. 2). There

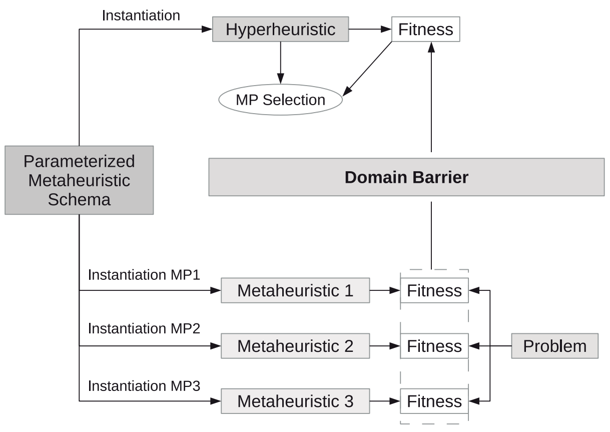


Fig. 2 Graphic representation of a hyperheuristic implemented on top of the parameterized metaheuristic schema

are different possibilities for combining elements (metaheuristics) and for computing the fitness [6].

The parallelization based on the shared-memory paradigm [2] is adapted here to a many-core system for the massive parallelization of the schema, which allows up to four levels of parallelism, including parallelism in the hyperheuristic and in the metaheuristics being selected. The schema is executed directly on the coprocessor without offloading from a host system, which is known as running in “native mode.”

3 Experimental results

The PLDP can be seen as the problem of searching for the values of the degrees of freedom (six) that globally minimize the scoring function. The values of the movements and rotations of the ligand can be approached with metaheuristics. Experiments are carried out in two Many Integrated Core (MIC) Intel Xeon Phi, one is the model 3120 with 57 cores at 1.1 GHz and 6 GBytes GDDR5, and the other is a 7120 with 61 cores at 1.2 GHz and 16 GDDR5. Both are based on Pentium (x86), each core supports four hardware threads, and both have a bidirectional ring bus. We call them XP57 and XP61. The results obtained in MIC are compared with those in two multicore nodes, the first comprises two Intel hexa-cores Xeon Haswell ES-2620 V3, 3.40 GHz, and 64 GB of RAM, and the second is an AMD Phenom II X6 1075 (hexa-core), 3.00 GHz, with 16 GB. We call them Xeon12 and AMD6.

Two instances of the PLDP from the well-known Protein Data Bank [13] are used in the experiments. They correspond to proteins PDB:2BSM and PDB:2BXD.

3.1 Exploitation of four-level parallelism

Experiments are conduced to analyze the influence on the execution times of the division of threads between the four levels of parallelism in the MIC architecture in order

Table 1 Values of the metaheuristic parameters for the Reduced Hyperheuristic (Hre) used in the experiments

INEIni	FNEIni	PEIIni	IIEIni	STMIni	NBESel
5	5	50	3	0	3
NWESel	NBBCom	NBWCom	NWWCom	PEIImp	IIEImp
2	2	3	2	50	3
SMIImp	PEDImp	IDEImp	SMDImp	NBEInc	LTMInc
0	10	5	0	3	5

to enhance the performance of our hyperheuristic schema when massive parallelism is used. Due to the high computational cost of applying hyperheuristics to the PLDP and because the study was not focused on optimizing fitness, a Reduced Hyperheuristic (Hre) with small values for the metaheuristic parameters (Table 1) is used for the analysis of the execution time. Due to the high computational cost of the hyperheuristic, low values were fixed for most of the hyperheuristic parameters, and more threads are devoted to the computations with the metaheuristics the hyperheuristic experiments with.

Different values of the parallelism parameters for the Reduced Hyperheuristic in Table 1 are considered for the executions in the range between 20 and 250 threads (ThT). The threads are spread over the hyperheuristic (ThH) and the metaheuristics (ThM). The product of the number of threads in the hyperheuristic and the metaheuristics is equal to the total (ThM · ThH = ThT). The ThH combinations for ThT = 20 are shown in Table 2. For example, for the fifth series (ThH_20_5), with a total of ThT = 20 threads, when the initial generation of the reference set is executed for the hyperheuristic with ThH = 5 threads, the metaheuristics are executed with ThM = 20/5 = 4 threads. The values of the metaheuristic parameters for the metaheuristics are higher than for the hyperheuristic, which leads to coarse-grained parallelism, for which a large number of threads is preferred at the first level of the two-level routines, and the number of threads at the second level of the metaheuristics is fixed to 1.

The Reduced Hyperheuristic in Table 1 was applied to search for satisfactory metaheuristics. The protein PDB:2BSM was used to train the hyperheuristic, and Table 3 shows the experimental times (in seconds) obtained in XP57. Due to the small parameter values managed by the Hre (NFEIni = 5 among others), values of ThT greater than 20 have the same thread combinations as ThH_ 20 for all the series in Table 2, so the influence of the threads of the metaheuristic is given for a fixed configuration of the Hre threads (ThH_20 to ThH_250 series). Figure 3 depicts the times in Table 3. The advantage of using parallelism is clear, with a maximum speed-up of approximately 30 with respect to the sequential execution in Xeon Phi and with the total number of threads close to the maximum available. The advantage of the exploitation of parallelism seems to reach saturation for a not very large number of threads (flat shape of the surface in Fig. 3). For this computationally demanding problem, the massive parallelism of MIC is well exploited with the parallel, parameterized metaheuristic schema.

Table 2 Number of threads of one and two levels of parallelism for the execution of the Reduced Hyperheuristic in Table 1, with the total number of threads set to 20

	One-level parallel routines			Two-level parallel routines				
	TGEIni	TCPCom	TIEInc	TI_Ini	TR_Imp	TC_Imp	TR_Div	TC_Div
ThH_20_1								
p_1	1	1	1	1	1	1	1	1
p_2	–	–	–	2	2	2	1	1
ThH_20_2								
p_1	2	2	2	1	1	1	2	2
p_2	–	–	–	2	2	2	1	1
ThH_20_3								
p_1	5	5	5	2	2	2	5	5
p_2	–	–	–	2	2	2	1	1
ThH_20_4								
p_1	5	5	5	1	1	1	1	1
p_2	–	–	–	5	5	5	2	2
ThH_20_5								
p_1	5	5	5	1	1	1	2	2
p_2	–	–	–	5	5	5	2	2
ThH_20_6								
p_1	10	10	10	2	2	2	5	5
p_2	–	–	–	5	5	5	2	2
ThH_20_7								
p_1	10	10	10	1	1	1	1	1
p_2	–	–	–	10	10	10	5	5
ThH_20_8								
p_1	10	10	10	1	1	1	2	2
p_2	–	–	–	10	10	10	5	5
ThH_20_9								
p_1	20	20	20	2	2	2	4	4
p_2	–	–	–	10	10	10	5	5

ThH_ThT_series represents *threads used in the hyperheuristic_total number of threads_series of the experiment*, and the remaining threads are assigned to the metaheuristics at low level (ThM), with ThM·ThH = ThT

3.2 Comparison of metaheuristics

The application of a hyperheuristic of the type considered here generates a satisfactory metaheuristic for the problem in hand. The metaheuristics so generated are parallelized with the same schema, but now with all the threads available. We compare the time and fitness results of four metaheuristics generated by hand (M1–M4 in Table 4) considering low execution time and goodness of fitness. The results are compared with those with the metaheuristics obtained by two simple hyperheuristics:

Table 3 Execution time in seconds for the Hre in Table 1 and various thread combinations in Table 2 applied to PDB:2BSM in XP61

Thread combination	Total number of threads (ThT)					
	20	50	100	150	200	250
ThH_ThT_1	795	428	346	367	366	361
ThH_ThT_2	782	508	388	296	327	422
ThH_ThT_3	940	495	354	283	282	338
ThH_ThT_4	765	519	359	339	324	393
ThH_ThT_5	950	469	343	375	390	397
ThH_ThT_6	1021	530	396	401	345	365
ThH_ThT_7	1449	696	398	331	477	357
ThH_ThT_8	1436	734	407	407	308	442
ThH_ThT_9	1512	910	459	416	318	265

The results are the mean of five executions. The sequential time in Xeon Phi was 7983 s

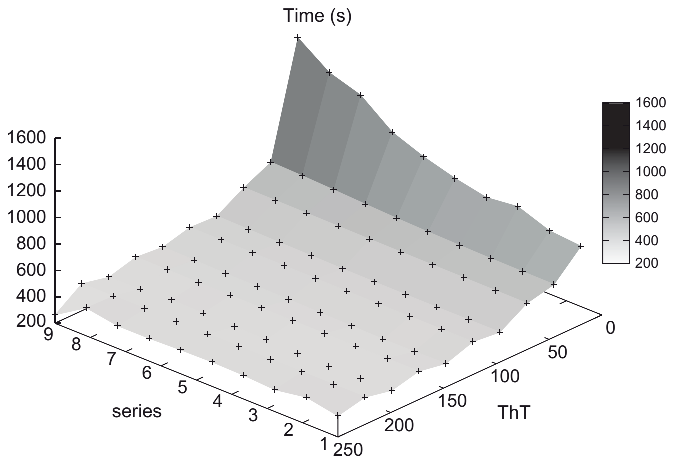


Fig. 3 Evolution of the execution time for the Hre in Table 1 with various thread combinations in Table 2

- A GRASP-based Hyperheuristic (Hgr) with low values of the parameters, consisting of a set of INEIni = 5 individuals to be improved with an intensification of IIEIni = 3.
- A basic Random Search Hyperheuristic (Hrs) with medium size, INEIni = 100.

The values of the metaheuristic parameters obtained with the hyperheuristics are labeled M-Hgr and M-Hrs in Table 4. The parameters PEIImp and IIEImp are fixed to 0 for all combinations to reduce the computation time of the metaheuristics, but other values are also possible. As mentioned, the methodology to work with the parameterized metaheuristic schema is valid for different implementations of the schema, so the implementation of the metaheuristics for this experiment differs slightly from the previous implementation: After the initialization, the NFBEIni best elements are

Table 4 Values of the parameters for four metaheuristics not selected automatically (M1–M4), and those selected automatically by the random search (M-Hrs) and the GRASP-based (M-Hgr) hyperheuristics; and lower and upper limits of the metaheuristic parameters for the selection of elements by the hyperheuristics

	INEIni	NEIIni	IIEIni	NFBEIni	NFWEIni	NBESel	NWESel
M1	64	32	10	16	16	20	12
M2	64	64	10	32	32	40	24
M3	96	96	10	50	46	46	50
M4	128	128	10	64	128	64	64
M-Hrs	86	24	2	7	17	14	10
M-Hgr	96	62	4	20	35	28	27
Lower	10	10	1	5	10	10	10
Upper	100	100	5	100	100	100	100

	NBBCom	NWWCom	NBWCom	NEIImp	IIEImp	NBEInc
M1	10	5	10	0	0	16
M2	20	15	20	0	0	32
M3	40	25	40	0	0	50
M4	20	15	20	0	0	64
M-Hrs	42	14	45	0	0	6
M-Hgr	36	49	6	0	0	7
Lower	5	5	5	0	0	5
Upper	50	50	50	0	0	100

included in the reference set, together with NFWEIni elements selected from the rest, with FNEIni = NFBEIni + NFWEIni. The interval of the metaheuristic parameters where the hyperheuristics search is also given.

Table 5 shows the execution time (in seconds) and the speed-up obtained when applying the metaheuristics in Table 4 to PDB:2BSM and PDB:2BXD. The sequential times obtained with one core from each multicore system are compared with parallel times in the multicores and the Xeon Phi cards. The XP61, with a maximum of 244 threads, was the fastest for all the metaheuristic and problem configurations studied. For the two instances of PLDP, parallel executions on MIC cards are faster than the parallel ones in the multicores. AMD6 is approximately three times slower than Xeon12, and the speed-up achieved with the MICs with respect to Xeon12 is between 4 and 6, with higher speed-ups for protein-metaheuristic combinations with larger times.

Finally, Table 6 shows the fitness obtained when applying the metaheuristics in Table 4 to the two proteins, sequentially in a core of the multicore systems, and in parallel with the multicores and with XP57 and XP61. Executions on the MICs were carried out with the maximum number of threads available (228 and 244) in the first level of parallelism and with one thread for the second level. A limit of 1000 s was established for each execution. In general, the parallel metaheuristics show better values of fitness (lower) than the sequential ones. This is especially observable in the

Table 5 Execution times (in seconds) and speed-ups obtained when applying the metaheuristics in Table 4 to two instances of the PLDP

	M1	M2	M3	M4	M-Hrs	M-Hgr
PDB:2BSM						
AMD6 sequen	126.2	361.3	1032.4	513.5	867.7	860.3
Xeon12 sequen	84.3	241.5	689.9	343.1	579.6	575.7
AMD6 paral	24.7	70.5	201.0	100.3	169.4	167.8
Xeon12 paral	8.8	25.1	86.3	35.7	60.9	70.2
XP57	2.1	5.4	14.5	7.5	12.2	12.2
XP61	1.8	4.5	12.1	6.3	10.3	10.2
Sp Xeon12/XP57	4.1	4.7	5.9	4.8	5.0	5.8
Sp Xeon12/XP61	4.8	5.6	7.1	5.6	5.9	6.9
PDB:2BXD						
AMD6 sequen	832.7	2387.2	6801.0	3390.8	5730.1	5691.8
Xeon12 sequen	558.0	1597.3	4564.6	2273.2	3840.4	3809.2
AMD6 paral	162.4	465.2	1327.0	660.9	1116.4	1109.0
Xeon12 paral	57.7	164.8	469.8	234.1	395.6	392.8
XP57	11.6	32.3	91.3	45.8	77.0	76.4
XP61	9.7	27.0	75.9	38.1	64.0	63.6
Sp Xeon12/XP57	5.0	5.1	5.1	5.1	5.1	5.1
Sp Xeon12/XP61	6.0	6.1	6.2	6.1	6.2	6.2

The sequential times were obtained in one core of the multicores, and the parallel ones in the two multicore systems and the two Xeon Phi cards. Executions on XP57 and XP61 were carried out with 228 and 244 threads, respectively, in the first level of parallelism and with one in the second level

results of PDB:2BXD, with a problem size approximately twice that of PDB:2BSM. The last column shows the mean of the fitness for each protein and computational system. Better results are obtained in Xeon Phi due to their higher speed, which results in more evaluations for the same execution time. These results confirm the suitability of the many-core architecture to improve the performance of the metaheuristics applied to the PLDP, with two possibilities of exploitation of parallelism: faster solutions with similar fitness, or better fitness with similar execution times.

4 Conclusions and future work

A shared-memory schema of hyperheuristics is used to select satisfactory metaheuristics to be applied to a molecular docking problem. Due to the high computational cost of the hyperheuristic, the parallel schema is executed in a many-core system in "native mode" with four levels of parallelism, which allows us to take full advantage of the massive parallelism offered by this architecture, obtaining an important reduction in execution times. The best results are obtained with a relatively low number of threads assigned to the hyperheuristic, and the rest is allocated to the low-level metaheuristic, with the total number of threads close to the maximum available.

Table 6 Fitness obtained when applying the metaheuristics in Table 4 to two instances of the PLDP, sequentially in one core of the multicore systems and in parallel with the two multicore and the two MICs

	M1	M2	M3	M4	M-Hrs	M-Hgr	Mean
PDB:2BSM							
AMD6 sequen	−122.2	−119.7	−97.0	−106.2	−100.4	−109.1	−109.1
Xeon12 sequen	−122.2	−119.7	−97.0	−113.6	−100.4	−109.1	−110.3
AMD6 paral	−116.6	−121.6	−127.5	−122.9	−123.7	−127.1	−123.2
Xeon12 paral	−108.8	−129.1	−136.3	−130.5	−125.9	−130.3	−126.8
XP57	−126.2	−134.9	−131.2	−125.5	−118.1	−137.1	−128.8
XP61	−121.0	−134.3	−132.6	−127.7	−122.1	−136.6	−129.1
PDB:2BXD							
AMD6 sequen	−155.1	−127.3	−135.2	−126.9	−136.1	−140.9	−136.9
Xeon12 sequen	−155.1	−127.3	−135.2	−126.9	−136.1	−140.9	−136.9
AMD6 paral	−158.2	−168.6	−143.7	−160.1	−156.8	−155.4	−157.1
Xeon12 paral	−167.8	−175.2	−188.2	−183.1	−152.7	−165.2	−172.0
XP57	−183.5	−202.2	−198.1	−196.1	−182.3	−207.1	−194.9
XP61	−174.5	−202.1	−208.8	−193.4	−191.4	−189.8	−193.3

A limit of 1000 s was established for each execution

For higher reductions of the execution time, it will be necessary to combine parallelism in the multicore host and the MIC coprocessor, and to exploit the vectorization capacity of MIC. The metaheuristics obtained from simple hyperheuristics are competitive with hand-generated metaheuristics in terms of fitness and have similar speed-ups. Bi-objective hyperheuristics searching for metaheuristics with low execution times and satisfactory fitness should be considered.

The parameterized schema can be applied to other optimization problems. We have applied it to the Maximum Diversity Problem, and its application to optimize other computationally demanding optimization problems in Xeon Phi is being analyzed. The inclusion of new basic metaheuristics, for example, ant colony optimization or particle swarm optimization, is also contemplated. Similar parameterized, parallel metaheuristic schemas should be developed for GPU and in heterogeneous clusters comprising nodes of multicores + multiple GPU or MIC. The use of large, heterogeneous clusters would be of special interest for the application of hyperheuristics with large reference sets or with a high fitness function cost, as in the case of the molecular docking problems.

Acknowledgements We thank the Service of Support to Technological Research of the Technical University of Cartagena for allowing us to run some experiments in their systems and to Luis Pedro García for his guidance in using them. This work was supported by the Spanish MINECO, as well as European Commission FEDER funds, under grants TIN2015-66972-C5-3-R and TIN2016-78799-P (AEI/FEDER, UE).

References

1. Andrusier N, Mashiach E, Nussinov R, Wolfson HJ (2008) Principles of flexible protein-protein docking. Proteins 73(2):271–289
2. Almeida F, Giménez D, López-Espín JJ (2011) A parameterized shared-memory scheme for parameterized metaheuristics. J Supercomput 58(3):292–301
3. Almeida F, Giménez D, López-Espín JJ, Pérez-Pérez M (2013) Parameterised schemes of metaheuristics: basic ideas and applications with Genetic Algorithms, Scatter Search and GRASP. IEEE Trans Syst Man Cybern Part A Syst Humans 43(3):570–586
4. Asanovic K, Bodik R, Catanzaro BC, Gebis JJ, Husbands P, Keutzer K, Patterson DA, Plishker WL, Shalf J, Williams SW, Yelick KA (2006) The landscape of parallel computing research: a view from Berkeley. Tech. rep., UCB/EECS-2006-183, EECS Department, University of California, Berkeley
5. Burke EK, Hyde M, Kendall G, Ochoa G, Özcan E, Woodward J (2010) A classification of hyper-heuristic approaches. In: Gendreau M, Potvin J-Y (eds) Handbook of Meta-heuristics. Springer, Berlin, pp 449–468
6. Cutillas-Lozano J-M, Giménez D, Almeida F (2015) Hyperheuristics based on parametrized meta-heuristic schemes. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 361–368
7. Imbernón B, Cecilia JM, Giménez D (2016) Enhancing metaheuristic-based virtual screening methods on massively parallel and heterogeneous systems. In: Proceedings of the 7th International Workshop on Programming Models and Applications for Multicores and Manycores, pp 50–58
8. Irwin JJ, Shoichet BK (2005) ZINC-a free database of commercially available compounds for virtual screening. J Chem Inf Model 45(1):177–182
9. Jorgensen WL (2004) The many roles of computation in drug discovery. Science 303:1813–1818
10. Navarro-Fernández J, Pérez-Sánchez H, Martínez-Martínez I, Meliciani I, Guerrero JA, Vicente V, Corral J, Wenzel W (2012) In silico discovery of a compound with nanomolar affinity to antithrombin causing partial activation and increased heparin affinity. J Med Chem 55(14):6403–6412
11. Nobile MS, Cazzaniga P, Tangherloni A, Besozzi D (2016) Graphics processing units in bioinformatics, computational biology and systems biology. Brief Bioinform. doi:10.1093/bib/bbw058
12. Özcan E, Bilgin B, Korkmaz E (2008) A comprehensive analysis of hyper-heuristics. Intell Data Anal 12(1):3–23
13. Protein Data Bank (1971) Nature New Biol 233:223
14. Rester U (2008) From virtuality to reality-virtual screening in lead discovery and lead optimization: a medicinal chemistry perspective. Curr Opin Drug Discov Dev 11(4):559–568
15. Talbi E-G, Zomaya AL (2006) Grids in bioinformatics and computational biology. J Parallel Distrib Comput 66(12):1481
16. Vega-Rodríguez MA, González-Álvarez DL (2015) Parallelism in bioinformatics: a view from different parallelism-based technologies. Parallel Comput 42:1–3
17. Wang J, Deng Y, Roux B (2006) Absolute binding free energy calculations using molecular dynamics simulations with restraining potentials. Biophys J 91(8):2798–2814
18. Yang MQ, Athey BD, Arabnia HR, Sung AH, Liu Q, Yang JY, Mao J, Deng Y (2009) High-throughput next-generation sequencing technologies foster new cutting-edge computing techniques in bioinformatics. BMC Genom 10(S–1):I1
19. Yang JY, Yang MQ, Zhu MM, Arabnia HR, Deng Y (2008) Promoting synergistic research and education in genomics and bioinformatics. BMC Genom 9(S–1):I1
20. Yang W, Yoshigoe K, Qin X, Liu JS, Yang JY, Niemierko A, Deng Y, Liu Y, Dunker AK, Chen Z, Wang L, Xu D, Arabnia HR, Tong W, Yang MQ (2014) Identification of genes and pathways involved in kidney renal clear cell carcinoma. BMC Bioinform 15(S–17):S2
21. Yuriev E, Agostino M, Ramsland PA (2011) Challenges and advances in computational docking: 2009 in review. J Mol Recognit 24(2):149–164
22. Zhou Z, Felts AK, Friesner RA, Levy RM (2007) Comparative performance of several flexible docking programs and scoring functions: enrichment studies for a diverse set of pharmaceutically relevant targets. J Chem Inf Model 47(4):1599–1608

2.4 Enhancing large-scale docking simulation on heterogeneous systems: an MPI vs rCUDA study

Título	*Enhancing large-scale docking simulation on heterogeneous systems: an MPI vs rCUDA study*
Autores	Baldomero Imbernón, Javier Prades, Domingo Giménez, José M. Cecilia y Federico Silla
Revista	Future Generation Computer Systems (FGCS)
Año	2017
Estado	Publicado

Contribución del Doctorando
Baldomero Imbernón Tudela, declara ser el principal autor y el principal contribuidor del artículo *Enhancing large-scale docking simulation on heterogeneous systems: an MPI vs rCUDA study.*

Future Generation Computer Systems 79 (2018) 26–37

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Enhancing large-scale docking simulation on heterogeneous systems: An MPI vs rCUDA study

Baldomero Imbernón[a,*], Javier Prades[c], Domingo Giménez[b], José M. Cecilia[a,*], Federico Silla[c]

[a] Polytechnic School, Universidad Católica de Murcia (UCAM), 30107, Murcia, Spain
[b] Department of Computing and Systems, University of Murcia, 30071, Murcia, Spain
[c] Departament d'Informàtica de Sistemes i Computadors, Universitat Politècnica de València, 46002 Valencia, Spain

HIGHLIGHTS

- We develop a docking application to leverage heterogeneous clusters.
- GPU virtualization is under-study as an alternative for intensive computing apps.
- Several load balancing strategies are proposed on those systems.
- The evaluation is two-fold performance and quality.

ARTICLE INFO

Article history:
Received 12 May 2017
Received in revised form 25 July 2017
Accepted 27 August 2017
Available online 14 September 2017

Keywords:
Virtual screening
HPC
rCUDA
Metaheuristics
Heterogeneous computing

ABSTRACT

Virtual Screening (VS) methods can considerably aid clinical research by predicting how ligands interact with pharmacological targets, thus accelerating the slow and critical process of finding new drugs. VS methods screen large databases of chemical compounds to find a candidate that interacts with a given target. The computational requirements of VS models, along with the size of the databases, containing up to millions of biological macromolecular structures, means computer clusters are a must. However, programming current clusters of computers is no easy task, as they have become heterogeneous and distributed systems where various programming models need to be used together to fully leverage their resources. This paper evaluates several strategies to provide peak performance to a GPU-based molecular docking application called *METADOCK* in heterogeneous clusters of computers based on CPU and NVIDIA Graphics Processing Units (GPUs). Our developments start with an OpenMP, MPI and CUDA *METADOCK* version as a baseline case of cluster utilization. Next, we explore the virtualized GPUs provided by the *rCUDA* framework in order to facilitate the programming process. rCUDA allows us to use remote GPUs, i.e. installed in other nodes of the cluster, as if they were installed in the local node, so enabling access to them using only OpenMP and CUDA. Finally, several load balancing strategies are analyzed in a search to enhance performance. Our results reveal that the use of middleware like rCUDA is a convincing alternative to leveraging heterogeneous clusters, as it offers even better performance than traditional approaches and also makes it easier to program these emerging clusters.

1. Introduction

Drug discovery and development may take more than a decade from discovery of a candidate drug to patient treatment [1]. There are several stages that a candidate drug must successfully go through. Among them, we would highlight the basic research of drug discovery, pre-clinical stages, clinical trials, and final review by associations like FDA (Food and Drug Administration) in the USA. The use of Virtual Screening (VS) methods can tremendously improve the drug discovery process, saving time, money and computational resources [2]. VS methods are computational techniques that analyze large libraries of small molecules (a.k.a. *ligands*) to search for structures most likely to bind to a target drug, typically a protein receptor or enzyme [3]. These libraries of chemical compounds may contain up to millions of ligands [4], given that analyzing larger databases exponentially increases the chances of generating hits. However, current VS methods, such as docking [5],

* Corresponding authors.
E-mail addresses: bimbernon@ucam.edu (B. Imbernón), japraga@gap.upv.es (J. Prades), domingo@um.es (D. Giménez), jmcecilia@ucam.edu (J.M. Cecilia), fsilla@disca.upv.es (F. Silla).

http://dx.doi.org/10.1016/j.future.2017.08.050
0167-739X/© 2017 Elsevier B.V. All rights reserved.

fail to make good toxicity and activity predictions, since they are constrained by their access to computational resources; indeed, the fastest VS methods cannot process large biological databases in reasonable times.

The use of high performance computing in order to enhance virtual screening methods is therefore necessary to fulfill pharmaceutical industry expectations, and a lot of research is been carried out in this regard. Methods like Autodock [6], Autodock VINA [7], Glide [8], LeadFinder [9], SurFlex [10], ICM+ [11], FMD [12] or DOCK [13] use multithreading programming at the node level in order to leverage multicore architectures, and some of them even distribute their computations among the CPUs of several nodes by means of the Message Passing Interface (MPI) library. However, we are currently witnessing a steady transition to heterogeneous computing systems [14], with heterogeneity representing systems where nodes combine traditional multicore architectures (CPUs) with accelerators such as Graphics Processing Units (GPUs). Programs such as BUDE [15], AMBER [16] or BINDSURF [17] use GPUs to overcome this problem by dividing the whole protein surface into independent regions (or spots). However, heterogeneity may limit system growth as it can no longer be addressed in an incremental way. Indeed, several computational challenges come up with such heterogeneous systems [18], like scalability, programmability or data management, to mention just a few.

In addition to the use of heterogeneous systems, virtualization techniques may provide significant improvements, as they enable a larger resource utilization by sharing a given hardware among several users, thus reducing the required amount of instances of that particular device. As a result, virtualization is being increasingly adopted in data centers. Some of the most extended virtualization techniques are based on software solutions, such as the VMware [19] (by VMware Inc.) or Xen [20] hypervisors. These solutions virtualize the entire system, providing a whole virtual computer to each user. However, although using virtual machines is appealing in many cases, even for high performance computing, when the goal is to make use of GPUs, these solutions introduce an unacceptable overhead due to the strong limitations they present with respect to the shared use of accelerators. In this regard, current virtual machine approaches are unable to concurrently share a GPU among several virtual machine instances.[1] Therefore, instead of virtualizing the entire computer, an alternative approach would be to virtualize specific resources, such as the GPU.

rCUDA [21] is a framework that enables remote concurrent use of CUDA-compatible GPUs. To enable remote GPU-based acceleration, this framework creates virtual CUDA-compatible devices on machines without local GPUs. These virtual devices represent physical GPUs located in a remote host offering GPGPU (General-Purpose Computing on Graphics Processing Units) services. Thus, all nodes in a cluster are able to access the whole set of CUDA accelerators concurrently. Moreover, a single-node shared-memory application could access all the GPUs in the cluster without using the MPI library, which potentially reduces the programming complexity. Additionally, given that real GPUs are concurrently shared among several applications, energy would be saved at the same time that a lower hardware investment is required. Furthermore, this approach would still deliver an acceptable performance, as shown in [22].

In this paper, we analyze the current computational landscape by applying heterogeneous clusters based on NVIDIA GPUs and CPUs to a challenging problem such as molecular *docking* computational methodology, called *METADOCK* [23], where the interaction between two molecules (a macromolecule known as receptor and a small molecule referred to as ligand) is simulated by minimizing a scoring function (affinity between the two molecules) that models the chemical process behind molecular interaction. The METADOCK methodology has two main characteristics: (1) the user can configure the *optimization procedure* at compile time from among a wide set of metaheuristics (i.e, algorithms like Genetic Algorithm, Scatter Search or local search methods), and (2) the calculation of the computationally expensive *scoring function* is offloaded to GPUs. With this in mind, major contributions of this paper include the following:

1. We develop a new version of *METADOCK* to perform large-scale simulations on heterogeneous computer clusters based on CPUs and NVIDIA GPUs. The implementation is developed using a traditional approach based on MPI, OpenMP and CUDA.
2. We evaluate *rCUDA* as a framework to leverage virtualized GPUs and also to facilitate the programming. This implementation only requires us to deal with OpenMP and CUDA APIs.
3. Several load balancing strategies are also evaluated in both configurations (virtualized and non-virtualized GPUs) to pursue the performance into unprecedented levels.
4. Finally, we check whether the search for performance is translated into an actual benefit in the quality of the results (reductions in execution time do not necessarily mean a better affinity quality). In this paper, the search procedure of *METADOCK* is configured to use three different metaheuristics (genetic algorithm, scatter search and local search) in order to analyze the evolution of the fitness along with the performance improvements.

The rest of the paper is structured as follows. Section 2 includes the background about Virtual Screening methods and the scoring problem we are working on and describes some relevant knowledge about HPC architectures. Next, our metaheuristic-based virtual screening technique is introduced in Section 3. The parallel strategies used for the efficient application of these techniques in heterogeneous clusters are explained in Section 4, whereas the experimental results are presented and analyzed in Section 5. Related work are reviewed in Section 6. The final section summarizes the conclusions and gives some directions for future work.

2. Background

This section introduces the main characteristics of Virtual Screening methods and heterogeneous clusters to better understand the rest of the paper.

2.1. Virtual screening methods

We draw on our description of Virtual Screening (VS), which was first given in [17,23]. VS methods are computational techniques used in several scientific areas, such as catalysts and energy materials [24], and mainly drug discovery [5], where experimental techniques can benefit from computational simulation.

VS methods search for libraries of small molecules that can potentially bind to a drug target, typically a protein receptor or enzyme, with high affinity. They actually "dock" small molecules into the structures of macromolecular targets. Moreover, they look for (i.e., score) the optimal binding sites by providing a ranking of chemical compounds according to the estimated affinity or *scoring* [25]. In general, VS methods optimize *scoring functions*, which are mathematical models used to predict the strength of the non-covalent interaction between two molecules after docking [26]. Indeed, these candidate molecules will continue the drug discovery

[1] Notice that the GRID GPU by NVIDIA is designed to be shared among VMs. However, the shared usage of this GPU is limited to desktop virtualization. When GRID GPUs are used as CUDA accelerators they cannot be shared among VMs.

process road-map that goes from in-vitro studies to animal investigations and, eventually, to human trials [27].

Although VS methods have been used for many years and have identified several compounds to be used in drugs, VS has not yet fulfilled all its expectations. Neither the VS methods nor the scoring functions used are sufficiently accurate to identify high-affinity ligands reliably. To deal with large numbers of potential candidates (many databases comprise hundreds of thousands of ligands), VS methods must be very fast and still they would require hundreds of CPU hours for each ligand, and, according to [28], even thousands of CPU hours for each ligand when simulation strategies are used to compute absolute binding affinities.

2.2. Metaheuristics

A wide range of optimization problems, like VS methods, are NP-hard and cannot afford to compute all possible solutions. In such scenarios, metaheuristics provide an abstraction layer for good enough solutions which are found in a reduced search space focused just on promising areas [29]. Metaheuristics can be specially useful in VS methods.

Metaheuristics of interest to us fall into two prominent classes:

- *Distributed metaheuristics.* These metaheuristics search within the entire solution space, and work with populations that are combined to improve solutions progressively. Examples of this group include Ant Colony, Particle Swarm Optimization, Genetic Algorithms and Scatter Search.
- *Neighborhood metaheuristics.* These metaheuristics work with an element in the solution space and search for better elements in its neighborhood. Examples include Guided Local Search, Hill Climbing, Simulated Annealing, Tabu Search, Variable Neighborhood and GRASP (Greedy Randomized Adaptive Search Procedures). GRASP is a metaheuristic close to one of the parameter configurations of the metaheuristic later used in the experiments in this paper.

All the previous metaheuristics can be combined among them in order to improve the quality of the methodology, although in this paper they will not be combined. We will focus on separately using metaheuristics similar to Genetic Algorithms, Scatter Search and GRASP.

Diversity in metaheuristics [30] is worth investigating. We can first define a subset of alternatives, and then apply a tuning process which is fuzzy, or even blind, for the effects of certain values in the experimental praxis. This paper sheds some light on scenarios guided by computational criteria: minimize execution time and fitness using parameters which give similar results. Even so, the procedure may be different for each application area and, hence, our experimental analysis (Section 5) focuses more on quantifying potential gains. An artificial intelligent system or human expert can then use our findings to complete the selection process with clear benefits.

Algorithm 1 A parameterized metaheuristic schema to generate several types of Metaheuristics

```
Initialize(S,ParamIni)
while no End condition(S) do
    Select(S,Ssel,ParamSel)
    Combine(Ssel,Scom,ParamCom)
    Improve(Scom,ParamImp)
    Include(Scom,S,ParamInc)
end while
```

Many metaheuristics follow similar patterns, particularly those based on populations share six basic functions (see Algorithm 1): *Initialize*, *End condition*, *Select*, *Combine*, *Improve* and *Include* [31,32]. These functions work with several populations: *S*, *Ssel* and *Scom*. *S* represents the set of candidate solutions, where some selected solutions (*Ssel*) are combined to generate a new set of elements, *Scom*.

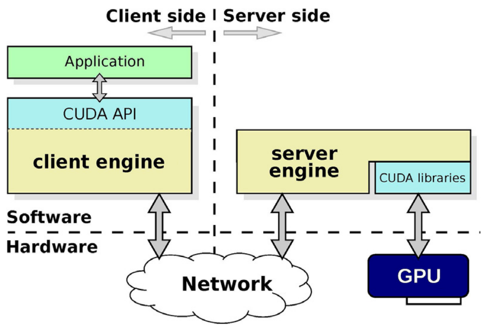


Fig. 1. Architecture of the rCUDA framework.

Within the above template, programmers can provide different instances and/or implementations, and the final set of candidate solutions (S) uses population insights to guide the search. Local search metaheuristics are also within the schema ($|S| = 1$).

Unified metaheuristic schemes can be enriched by introducing parameters for each function [33–35] (see Param-prefixes in Algorithm 1), and hybrid metaheuristic schemes can be considered too, with computational complexity increasing continuously. In the benchmarks throughout this work, we use the parameterized schema with two configurations of the parameters which give metaheuristics close to Genetic Algorithms and Scatter Search.

2.3. Programming heterogeneous clusters

Since the early days, computer architects have relied on technology scaling to provide increased performance. Heterogeneous architecture design is now seen as the only solution to continue Moore's law scaling through innovation alone [14,36], with systems where nodes combine traditional multicore architectures (CPUs) and accelerators (mostly GPUs) [37].

Traditional parallel implementations are not always efficient when ported to heterogeneous 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 asymmetric 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. Therefore, programmers play a fundamental role in this heterogeneous context as they have to deal with different programming models such as OpenMP [38], MPI [39] and OpenCL [40] or CUDA [37] to fully leverage all computing resources in current clusters. In this paper we address different programming models and techniques to accommodate our VS application to an increasingly heterogeneous underlying hardware.

2.3.1. The rCUDA middleware

Fig. 1 depicts the architecture of the rCUDA framework, which follows a client–server distributed approach. The client part of rCUDA is installed in the cluster node executing the application requesting GPU services, whereas the server side runs in the computer owning the actual GPU. The client side of the middleware offers the same application programming interface (API) as does the NVIDIA CUDA API. In this manner, the client receives a CUDA request from the accelerated application and appropriately processes and forwards it to the remote server. In the server node, the middleware receives the request and interprets and forwards it to the GPU, which completes the execution of the request and provides the execution results to the server middleware. In turn, the server sends back the results to the client middleware, which forwards them to the initial application, which is not aware that its request has been served by a remote GPU instead of a local one.

rCUDA is binary compatible with CUDA 8.0 and implements the entire CUDA Runtime and Driver APIs (except for graphics functions). It also provides support for the libraries included within CUDA (cuBLAS, cuFFT, etc.). Additionally, it supports several underlying interconnection technologies by making use of a set of runtime-loadable, network-specific communication modules (currently TCP/IP, RoCE and InfiniBand). Independently of the exact network used, data exchange between rCUDA clients and servers is pipelined in order to attain high performance. Internal pipeline buffers within rCUDA use preallocated pinned memory, given the higher throughput of this type of memory [41].

The InfiniBand communication module is based on the IB Verbs (IBV) API. This API offers two communication mechanisms: the channel semantics and the memory semantics. The former refers to the standard send/receive operations typically available in any networking library, while the latter offers RDMA operations where the initiator of the operation specifies both the source and destination of a data transfer, resulting in zero-copy transfers with minimum involvement of the CPUs. rCUDA employs both IBV mechanisms, selecting one or the other depending on the exact communication to be carried out.

3. Population-based metaheuristics for virtual screening: METADOCK

METADOCK [23] is a virtual screening (VS) application that simulates the interaction between two molecules. Indeed, it attempts to predict non-covalent binding of molecules or, more frequently, of a macromolecule (receptor) and a small molecule (ligand). This prediction is computationally performed through an iterative procedure that tries to minimize a scoring function that models such molecular interaction. Therefore *METADOCK*, as a VS application, has two main ingredients. First, the *optimization algorithm* is based on a metaheuristic schema (see Algorithm 1) to be able to select a metaheuristic that will guide the search procedure. Second, the scoring function calculation is offloaded to GPUs in order to speedup execution time.

With this in mind, we now provide some details about *METADOCK* and we refer the reader to [23] for further insights. *METADOCK* divides the whole receptor surface into arbitrary and independent regions (or spots) where the optimization process is performed independently. This enables the so-called *blind docking* that offers the opportunity to find novel binding sites, but drastically increasing the computational cost. The optimization at each spot consists of looking for the best ligand conformation that interacts with the lowest fitness value (it is a minimization problem). The fitness is given by the scoring function that mathematically represents the chemical interaction between the protein receptor and ligand conformation. In our case, the scoring function is based on the relevant non-bonded potentials used in VS calculations, which are the Coulomb, or electrostatic term, and the Lennard-Jones potentials, since they describe very accurately the most important short and long-range interactions between atoms of the protein–ligand system [42]. The calculation of the scoring function requires the highest percentage of the overall execution time, and it is offloaded to the GPU to be accelerated.

The simulation starts by minimizing the value of the scoring function by continuously making random or predefined perturbations of the initial population (S) at each spot. Particularly, each candidate solution is a conformation ligand that differs in the application of some movement (translating and/or rotating) with respect to a given region. Then, the new value of the scoring function for each candidate solution is obtained, being eventually accepted according to metaheuristic criteria.

As previously explained, *METADOCK* is able to configure the search procedure at compile time. This is performed by setting values to different parameters, which are listed in Table 1. These parameters are introduced in several functions of the general computational pattern (schema) that many population-based metaheuristics have in common (see Algorithm 1). The functions are briefly explained:

- **Initialize** returns an initial set of solutions. `INEIni` conformations are generated randomly. In this first generation, conformations are created with a different position and orientation around each spot. Then a percentage (`PEIIni`) of the initial conformations of each spot is improved. The intensification of the improvement is indicated by `IIEIni`. This improvement is a local search in which each conformation is modified within its neighborhood in the solutions space; i.e., it is translated or rotated with respect to its corresponding protein spot. Then, the scoring function is calculated to evaluate those new conformations. Finally, $(\texttt{PBEIni} + \texttt{PWEIni})^{*}$ `INEIni` conformations from each spot are selected for the execution of the subsequent functions. `PBEIni` and `PWEIni` represent the percentage of best and worst conformations according to the scoring function. The best conformations are those with the best fitness, and the “worst” conformations are randomly selected from the remaining ones. *METADOCK* does not select only the best conformations so as to diversify the search and prevent falling in local optima.
- **End condition** determines the stop criteria for METADOCK, which is either `MNIEnd` (maximum number of iterations), or `NIREnd` (maximum number of steps without improvement of the best solution among all the spots).
- **Select** chooses working conformations for subsequent phases. A percentage of the best (`PBESel`) and worst (`PWESel`) conformations relative to each spot are selected.
- **Combine** mixes conformations in pairs depending on their scoring. Three parameters represent the percentage of best–best, worst–worst and best–worst conformations to be combined: `PBBCom`, `PWWCom` and `PBWCom`, respectively. Combinations are performed among conformations at the same spot. In each combination, two conformations are generated with a different orientation and located on the line connecting the two elements to be combined.
- **Mutation** maintains the diversity of conformations after the *Combine* stage. For those conformations affected by the mutation, their position in the space or its orientation is modified randomly around the spot they are associated to. Two parameters are involved in this function: `PMUCom`, to define the percentage of conformations that the mutation procedure receives as an input, and `IMUCom`, the intensification of improvement of the elements obtained by mutation.
- **Improve** performs a local search within the neighborhood of some of the conformations previously generated by *Combine*. As in the improvement after initialization, two metaheuristic parameters are considered for each spot: `PEIImp`, for the percentage of conformations the local search is applied to, and `IIEImp`, for the number of trials for the local search. Hence, METADOCK can generate hybrid metaheuristics with different degrees of intensification.
- **Include** updates the reference set for the next iteration of the schema. Here, `PBEInc` establishes the percentage of best conformations associated to each spot to be included in its

Table 1
The seventeen metaheuristic parameters used in the unified parameterized metaheuristic schema for *METADOCK*.

Metaheuristic parameters	Description
INEIni	Number of initial ligand conformations.
PEIIni	Percentage of the best conformations that are improved in the function initialize.
IIEIni	The intensification of the improvement in the function initialize.
PBEIni	Percentage of best conformations to be included in the initial set for the next iterations.
PWEIni	Percentage of worst conformations to be included in the initial set for the next iterations.
PBESel	Percentage of the best conformations to be selected for combination.
PWESel	Percentage of the worst conformations to be selected for combination.
PBBCom	Percentage of best–best conformations to be combined.
PWWCom	Percentage of worst–worst conformations to be combined.
PBWCom	Percentage of best–worst conformations to be combined.
PMUCom	Percentage of best conformations of the combination to be muted.
IMUCom	The intensification of the mutation of elements generated by combination.
PEIImp	Percentage of best conformations of the combination to be improved.
IIEImp	The intensification of the improvement of elements generated by combination.
PBEInc	Percentage of best conformations to be included in the reference set.
NIREnd	Maximum number of steps without improvement.
MNIEnd	Maximum number of iterations with or without improvement.

reference set. The remaining conformations in the reference set are randomly selected and contribute to diversify the search, so avoiding stalling in local minima.

4. Targeting heterogeneous clusters

This section introduces the parallelization strategy of our docking methodology presented in Section 3 for a heterogeneous cluster based on CPUs–GPUs. First, our algorithm defines an MPI process for each existing node in the cluster where we run our simulation. The number of spots is sent to each node, where supporting data structures are also created to avoid communication overhead. Then, each MPI process creates as many OpenMP threads as GPUs are available at the node, which is easily obtained by querying the GPU properties at runtime using `cudaGetDeviceCount`.

Algorithm 2 Scoring function computation for multiGPU on each node

```
1:  omp_set_num_threads(number_GPUs)
2:  #pragma omp parallel for
3:  for i=1 to number_GPUs do
4:      Select_device(Devices[i].id)
5:      Host_To_GPU(Rhost,Sdevice)
6:      Conformations=Devices[i].conformations
7:      threads=Devices[i].Threadsblock
8:      stride=Devices[i].stride
9:      Calculate_scoring<Conformations/threads,threads>
        (Rdevice+Devices[i].stride)
10:     GPU_To_Host(Rhost,Rdevice)
11: end for
```

Algorithm 2 shows the parallelization schema we use to leverage heterogeneous nodes with shared-memory multiprocessors and multiple GPUs. OpenMP is used to manage several CPU threads, where each thread is responsible for controlling a GPU (lines 2 and 3). The targeted GPU for the actual CPU process is selected (line 4) and, from that point, all operations will be related to a different GPU. Some functions in Algorithm 2 work with various sets or populations (*Rhost* and *Rdevice*). These sets represent the receptor molecule on the CPU and GPU memories respectively. Line 5 copies *Rhost* into each GPU's device memory. Note that the whole receptor molecule is copied in all GPU memories. Although the computation of the scoring function at each spot is distributed among the different GPUs, all atoms of the receptor are needed to calculate the Lennard-Jones potentials (see [23]). Moreover, an additional structure, called *Devices*, is created to manage several configuration parameters. This structure stores, among other things, the number of conformations (line 6) assigned to each GPU; i.e., the number of different ligand configurations that will be executed at each spot. *Devices* also includes information about the ligand compound and, with that information, the different ligand conformations will be generated (translating or rotating this information) on the GPU. *Devices* also have some GPU runtime parameters such as memory, grid size, maximum threads per block (line 7), stride on set of population (line 8) and so on. Then, each GPU calculates the scoring function (line 9) for a set of conformations (i.e., candidate solutions). In our homogeneous implementation, those conformations are equally distributed among GPUs in form of CUDA thread blocks. Actually, we associate each conformation to a CUDA warp, and warps are grouped into blocks depending on the CUDA thread block granularity.

Parallel runs do not incur any communication overhead, and the final solution is chosen from all the independent executions, given the stochastic nature of metaheuristics. The execution time of each independent execution can differ, as it depends on (1) the underlying GPU each metaheuristic instance runs on, which is actually unknown at compile-time, and (2) the number of conformations (the same in principle for each computing unit, but the execution time is affected by GPU heterogeneity). Given that the

Table 2
GPU architectures involved in the experimental study.

Feature	GTX 590	K20m	K40m
GPU generation	Fermi	Kepler	Kepler
Year released	2011	2013	2014
Raw computational power			
Number of cores	512	2496	2880
Core frequency (MHz)	1215	706	745
Peak processing (GFLOPS)	2x 1244	3520	4290
CUDA compute capability	2.0	3.5	3.5
Memory			
Size (GB)	2x 1.5	5.2	12
Frequency (MHz)	2x 1215	2x 2600	2x 3004
Width (bits)	384	320	384
Bandwith (GB/s)	163.8	208	288
Cache			
Shared memory/multipr.	48 KB	64 KB	64 KB
L2 cache	768 KB	1.5 MB	1.5 MB

Table 3
Hardware location and models of the experimental environment.

Device	Model	Node	Device	Model	Node
0	K40m	node K1	6	K20m	node K3
1	K20m	node K1	7	K20m	node K3
2	K40m	node K2	8	GTX 590	node Gtx1
3	K20m	node K2	9	GTX 590	node Gtx1
4	K20m	node K3	10	GTX 590	node Gtx2
5	K20m	node K3	11	GTX 590	node Gtx2

Table 4
Relation between experiment tag and hardware location shown in Table 3.

(Number of GPUs)	Devices used (IDs from Table 3)	Tag (Number of GPUs)	Devices used (IDs from Table 3)
2 GPUs	0,1	8 GPUs	0,1,2,3,4,5 6,7
4 GPUs	0,1,2,3	10 GPUs	0,1,2,3,4,5 6,7,8,9,10
6 GPUs	0,1,2,3,4,5	12 GPUs	0,1,2,3,4,5 6,7,8,9,10,11,12

slowest GPU will determine the overall execution time, our mission in the next steps is to make use of the idle time offered by the most powerful GPUs. This requires the implementation of a load balancing strategy that can somehow distribute a higher number of conformations to the most powerful GPUs. Indeed, there is a trade off between performance and overhead introduced in the design of this load balancing strategy. Here, we propose two different alternatives. A load balancing strategy based on the features offered by the manufacturer for each device (*Theoretical Distribution*) and a load-balancing strategy that explores the application performance on each GPU before the computation is actually carried out. Indeed, the former is straightforward as peak performances provided by manufacturer are under "ideal" conditions but it does not require an additional computational time. The latter, however, would introduce an overhead but it will theoretically obtain better application performances. This overhead is mainly due to a *warm-up* phase where you can get the real performance of each GPU for our problem. This phase is common for all metaheuristics, and it is done to establish performance differences among all targeted GPU by running the scoring function for a few simulated solutions. This phase measures the execution time of a small number of iterations in order to detect these differences. Importantly, at this stage, the algorithm is not trying to *solve* the docking problem in any meaningful sense (five to ten iterations are not enough to do this), but these runs do allow us to calculate the performance differences between GPUs. The execution times in this *warm-up* phase in all GPUs are reduced to obtain the maximum value using `mpi_Reduce`. Each node then calculates the number of conformations to deal with based on this information.

5. Evaluation

This section shows an experimental evaluation of our three different virtual screening strategies running on a heterogeneous cluster based on Intel CPUs and NVIDIA GPUs. First of all, we briefly introduce the hardware and software environment where the experiments are carried out. Afterwards, three different studies are carried out: (a) two different runtime environments are evaluated, namely the traditional MPI based programming approach and the rCUDA approach[2]; (b) different load balancing strategies are also analyzed in order to get peak performance of the system; (c) we carry out an analysis of the quality of our applications in terms of fitness.

5.1. Hardware environment and benchmarking

Hardware and software environment: Experiments have been conducted in a cluster based on five 1027GR-TRF Supermicro nodes. Each node contains two Intel Xeon E5-2620 v2 processors, and has a Mellanox ConnectX-3 VPI single-port InfiniBand adapter (FDR InfiniBand). The nodes are connected by a Mellanox switch SX6025 with FDR compatibility (a maximum rate of 56 Gb/s). Two different GPUs, NVIDIA Tesla K20m and NVIDIA Tesla K40m, are available for acceleration purposes at nodes K1 and K2. In nodes Gtx1 and Gtx2 one GeForce GTX 590 with 2 GPUs is available in each one. Additionally, one SYS7047GR-TRF Supermicro server, referred to as node K3 with identical processors is available with 4 NVIDIA Tesla K20m GPUs and 128 GB of DDR3 SDRAM memory at 1600MHz. The CentOS 6.4 operating system and the Mellanox OFED 2.4-1.0.4 were used at the servers along with the NVIDIA driver 346.96 and CUDA 7.0. The rCUDA version is 15.10 and the MPI configuration is based on MVAPICH2 2.0.

Table 2 gives insights about each GPU architecture found on these nodes. The experimental environment is summarized in Tables 3 and 4. The former shows each GPU location and Device Identifier. In other words, Table 3 shows the node where each GPU is located and the identifier assigned to each computational component. Table 4, however, shows the tag used in the experiments. For instance, the simulations performed with 2 GPUs use devices 0 and 1, which means K40m and K20m in the node K1, while simulations with 4 GPUs involve 0, 1, 2 and 3 GPUs, which means two nodes (K1 and K2).

Metaheuristics: Three different configurations of our metaheuristics are under study (referred to as M1, M2 and M3 in Table 5). The first hybrid metaheuristic (M1) is close to a *Genetic Algorithm* with a population of 2048 individuals for each spot. Elements are selected from the best ones for combination, and half of the resulting elements are mutated. This metaheuristic does not include local search to improve the conformations. The second metaheuristic (M2) is also an evolutionary method but, in this case, it is closer to a *Scatter Search* algorithm with a population of 512 individuals. In this case, all the elements are improved after the initial generation and the combination, and the improvement is a local search in the neighborhood of each element to obtain better solutions. The last metaheuristic (M3) is a method of search in the neighborhood. A local search is carried out at each spot by changing position and orientation of the conformations. Notice that we have used different population sizes for our metaheuristics because we are interested in the scalability of our system and, therefore metaheuristics are designed to have different populations, which means having different computing intensity. All individuals are considered for selection and combination to perform the algorithm previously explained in Section 2.2.

Databases: A set of benchmark instances from the well-known Directory of Useful Decoys (DUD) [43] was used for testing. Surface screening was performed on proteins GPB, SRC and COMT and their corresponding crystallography ligands. Table 6 shows the size of each complex protein–ligand, with the number of atoms of each component and the number of spots of the receptor. They have different sizes to test the scalability of the methods implemented.

5.2. Runtime evaluation

This section shows the performance evaluation of our VS methodology on a heterogeneous cluster based on CPU + GPU. We

[2] Remember that the rCUDA approach consists of providing the application GPUs located anywhere in the cluster. In this way, the application is programmed as a shared-memory application without having to use MPI in order to make use of the GPUs located in other cluster nodes. Furthermore, the application code does not need to be specifically designed for rCUDA but it is just designed to use multiple GPUs.

Table 5
Metaheuristics used for experimentation.

	COMBINATIONS		
	M1	M2	M3
INEIni	2048	512	2048
PEIIni	0	100	100
IIEIni	0	20	100
PBEIni	100	4	0
PWEIni	0	4	0
PBESel	2	50	0
PWESel	0	50	0
PBBCom	100	100	0
PWWCom	0	100	0
PBWCom	0	100	0
PMUCom	50	0	0
IMUCom	20	0	0
PEIImp	0	100	0
IIEImp	0	20	0
PBEInc	100	70	0

Table 6
Size of receptor and crystallography ligand (number of atoms) used for performance comparison, and the number of spots for each receptor.

Targets	Number of spots	Receptor size (number of atoms)	Crystallography ligand size (number of atoms)
GPB	813	13 261	52
SRC	452	7158	67
COMT	214	3419	29

compare our MPI + OpenMP + CUDA version to leverage the heterogeneous cluster, which requires a larger programming effort, with an OpenMP + CUDA version that uses rCUDA as underlying runtime system. We ensured that simulations in both systems featured the same starting point by setting the seed in the random number generator. The idea after this comparison is that four GPUs are actually the maximum number of GPUs that are physically installed within the same node in our cluster. Therefore, the only way to use all GPUs in the cluster would be to use libraries like MPI. This introduces an extra programming effort to access all the available computational resources. Another way to do that, bypassing the MPI requirement, is to use a runtime system such as rCUDA that transparently "brings" all GPUs available in the cluster to a given node where the computation is carried out. The question that comes up here is if there is any overhead related to the use of runtime systems like rCUDA instead of a hand-made MPI code. Fig. 2 shows the execution time of *METADOCK* using both alternatives. In particular, we use a GRASP metaheuristic ($M3$) The number of the overall number of GPUs is incremented on the X-axis and different protein–ligand conformations are simulated (see Table 6) to analyze the system scalability. Indeed, the *METADOCK* performance increases with the number of GPUs. As previously explained, the metaheuristic runs simultaneously at each spot and the spots are equally distributed among GPUs. These results justify the use of multiple GPUs to speedup our simulations. Execution times of the MPI and rCUDA versions are very close. Actually, a small performance gain is reported in the rCUDA version.

The reason for the performance gain with rCUDA is the following. In a classical approach, programming interfaces like MPI are used to involve several nodes to solve a given problem. In contrast, rCUDA avoids the use of MPI, which eases the development and avoids the MPI runtime overhead by using a very efficient communication pipeline between client and server nodes [21]. In addition to make the program more complex from the programming point of view, MPI also presents a non-negligible overhead when dealing with exchanged messages. This means that a shared-memory application may present a better performance than an MPI one for the same amount of leveraged CPU cores. As a result of the combination of both factors, our application provides better performance when it accesses GPUs in other nodes of the cluster by using rCUDA instead of MPI.

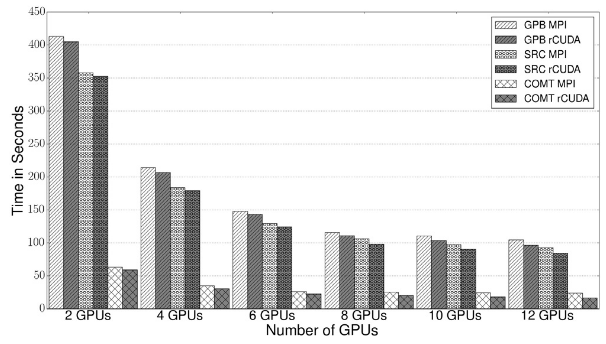


Fig. 2. Execution time (in seconds) with COMT, GPB and SRC complexes from two to twelve GPUs with M3 for our MPI + OpenMP + CUDA implementation vs. OpenMP + CUDA with rCUDA middleware.

Table 7
Workload (in percentages) for each GPU with two distributions (homogeneous and theoretical) for COMT, GPB and SRC complexes.

Numbers of GPUs	Homogeneous distribution device/workload		Theoretical distribution device/workload	
2 GPUs	0–50%	1–50%	0–55%	1–45%
4 GPUs	0–25%	2–25%	0–27.5%	2–27.5%
	1–25%	3–25%	1–22.5%	3–22.5%
6 GPUs	0–16.6%	3–16.6%	0–19.3%	3–15.3%
	1–16.6%	4–16.6%	1–15.3%	4–15.3%
	2–16.6%	5–16.6%	2–19.3%	5–15.3%
8 GPUs	0–12.5%	4–12.5%	0–14%	4–12%
	1–12.5%	5–12.5%	1–12%	5–12%
	2–12.5%	6–12.5%	2–14%	6–12%
	3–12.5%	7–12.5%	3–12%	7–12%
10 GPUs	0–10%	5–10%	0–13%	5–11%
	1–10%	6–10%	1–11%	6–11%
	2–10%	7–10%	2–13%	7–11%
	3–10%	8–10%	3–11%	8–4%
	4–10%	9–10%	4–11%	9–4%
12 GPUs	0–8.3%	6–8.3%	0–12%	6–10%
	1–8.3%	7–8.3%	1–10%	7–10%
	2–8.3%	8–8.3%	2–12%	8–4%
	3–8.3%	9–8.3%	3–10%	9–4%
	4–8.3%	10–8.3%	4–10%	10–4%
	5–8.3%	11–8.3%	5–10%	11–4%

5.3. *Load-balancing strategy evaluation*

The existence of different families of GPUs (see Table 2) in the same cluster requires to adapt the amount of work to be carried out by the different kernels to the different computing power of the devices in the cluster. We have adapted *METADOCK* to work with heterogeneous architectures and massively parallel techniques applied to areas of intensive computing, distributing the work onto all the available devices. The independence of a conformation with respect to the others when calculating their potential makes it easier to take advantage of the heterogeneity by dividing computation.

This section evaluates three different load-balancing techniques:

- In the *Homogeneous Distribution* all the devices receive the same workload.

Table 8
Workload (in percentages) for each GPU with heterogeneous distributions for COMT, GPB and SRC complexes.

Numbers of GPUs	COMT heterogeneous distribution device/workload		GPB heterogeneous distribution device/workload		SRC heterogeneous distribution device/workload	
2 GPUs	0–55.7%	1–44.3%	0–55.1%	1–44.9%	0–5.55%	1–44.5%
4 GPUs	0–28.1%	2–27.4%	0–27.1%	2–27.4%	0–27.7%	2–27.9%
	1–21.9%	3–22.6%	1–22.6%	3–22.9%	1–22.3%	3–22.1%
6 GPUs	0–19.1%	3–15.3%	0–18.9%	3–15%	0–18.7%	3–15.3%
	1–15.1%	4–15.7%	1–15.9%	4–15.9%	1–15.8%	4–15.7%
	2–18.3%	5–16.5%	2–17.9%	5–16.4%	2–18.2%	5–16.3%
8 GPUs	0–14.4%	4–11.8%	0–15%	4–11.1%	0–14.7%	4–11.5%
	1–12.1%	5–11.9%	1–11.8%	5–11.9%	1–11.7%	5–11.7%
	2–15%	6–11.7%	2–14.6%	6–11.8%	2–14.8%	6–11.9%
	3–11.6%	7–11.5%	3–12.1%	7–11.7%	3–11.9%	7–11.8%
10 GPUs	0–14.1%	5–10.5%	0–14%	5–11.2%	0–13.9%	5–11.0%
	1–11.6%	6–10.7%	1–11.2%	6–10.6%	1–11.1%	6–10.7%
	2–14.1%	7–11%	2–14.3%	7–10.4%	2–14.1%	7–10.8%
	3–11.1%	8–3.2%	3–10.8%	8–3.1%	3–10.9%	8–3.4%
	4–10.6%	9–3.1%	4–11%	9–3.4%	4–10.8%	9–3.3%
12 GPUs	0–13%	6–10.1%	0–13%	6–10.4%	0–12.8%	6–10.3%
	1–10.5%	7–10.7%	1–10.6%	7–10.8%	1–10.6%	7–10.5%
	2–13.1%	8–2.5%	2–13.1%	8–2.6%	2–12.9%	8–2.7%
	3–10.4%	9–2.9%	3–10.3%	9–2.9%	3–10.4%	9–3.0%
	4–10.2%	10–3.1%	4–10.2%	10–2.8%	4–10.3%	10–3.0%
	5–10.6%	11–2.9%	5–10.6%	11–2.7%	5–10.7%	11–2.8%

- In *Theoretical Distribution* the workload received by each device is calculated from the theoretical peak performance provided by the manufacturer.
- In the *Heterogeneous Distribution* the workload is established through a warm-up phase, previously explained in Section 4.

Tables 7 and 8 show the homogeneous, theoretical and heterogeneous distribution used for three protein–ligand complexes, varying the number of GPUs. In Table 7, the first and second columns show the percentage assigned to each device with the homogeneous distribution, and the percentages with the theoretical distribution are shown in columns three and four. With the homogeneous distribution the workload is equally distributed among all the GPUs, independently of the relative performance of the devices. The theoretical distribution uses the device performance provided by the manufacturer to obtain the percentages. In this case, Kepler GPUs are assigned more workload than Fermi GPUs. Table 8 shows the distribution obtained for the three complexes with the heterogeneous distribution. In this case, a higher percentage of workload is assigned to the GPUs that offer better performance for our problem. Some conclusions are drawn: (1) The theoretical distribution for the Kepler family is close to the heterogeneous distribution which actually explores the real computational differences among GPUs. (2) The workload for the three complexes in the heterogeneous distribution is very similar and does not depend on their sizes.

Table 9 shows the execution time for the warm-up phase using MPI. The warm-up phase in the MPI case explores the computational capabilities and distributes the work within each node; i.e. an MPI process is launched to each node where the warm-up phase is performed. Therefore, this is done in parallel at each node, and thus the execution time reported in Table 9 corresponds to the slowest node in computing this warm-up stage. The slowest GPUs targeted (GeForce GTX 590) are involved in the execution for 10 and 12 GPUs (see Tables 3 and 4), and they determine the overall execution time. Table 10 shows the execution time for the warm-up phase using rCUDA. Here, all the GPUs are virtually on a node and so the warm-up is not parallelized like in the MPI case. The progressive increase in execution time with the number of GPUs reflects this idea.

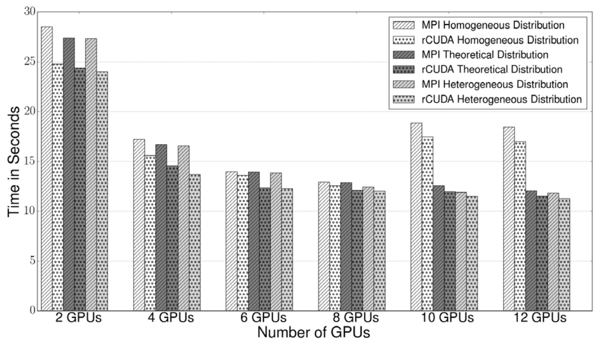


Fig. 3. Execution time (in seconds) with COMT complex and M1 for different distributions with MPI + OpenMP + CUDA and OpenMP + rCUDA implementations.

The warm-up phase introduces an overhead in both cases, that is higher for rCUDA whenever many GPUs are targeted. After evaluating the workload percentages assigned to each GPU, we observe in Table 8 that small and large complexes have similar workloads. This leads us to think about a clever strategy to reduce the overhead. The warm-up phase could be executed once at the installation stage, when compounds of several sizes could be evaluated, so storing meta information for future executions. It is important to bear in mind that, when solving a real problem, many iterations are carried out so representative values are obtained, and the overhead incurred in the *warm-up* phase is less significant.

Figs. 3–5 show the execution time in seconds for the load-balancing strategies considered (homogeneous, theoretical and heterogeneous), using MPI and rCUDA versions for the COMT, GPB and SRC protein–ligand complexes. We use only one metaheuristic (M1) for clarity, but the results are representative of the other metaheuristics experimented with. For low experimentation times, the results correspond to only one step of the metaheuristic, with a single execution of each function. The scalability is compared for the different complexes and implementations. In all the cases the homogeneous distribution obtains worse results, which

Table 9
Execution time (in seconds) of the warm-up phase for the MPI + OpenMP + CUDA implementation with three complexes (COMT, GPB and SRC).

Number of GPUs	2 GPUs	4 GPUs	6 GPUs	8 GPUs	10 GPUs	12 GPUs
COMT	2.53	2.49	2.51	3.36	5.28	5.55
GPB	5.13	5.11	5.06	5.98	6.91	6.87
SRC	8.28	8.45	8.35	8.86	9.98	9.79

Table 10
Execution time (in seconds) of the warm-up phase for the OpenMP + CUDA implementation with rCUDA middleware for three complexes (COMT, GPB and SRC).

Number of GPUs	2 GPUs	4 GPUs	6 GPUs	8 GPUs	10 GPUs	12 GPUs
COMT	1.08	2.21	3.42	5.57	8.01	10.823
GPB	2.07	3.62	4.21	6.34	9.71	12.51
SRC	2.26	4.24	4.71	7.19	11.25	14.43

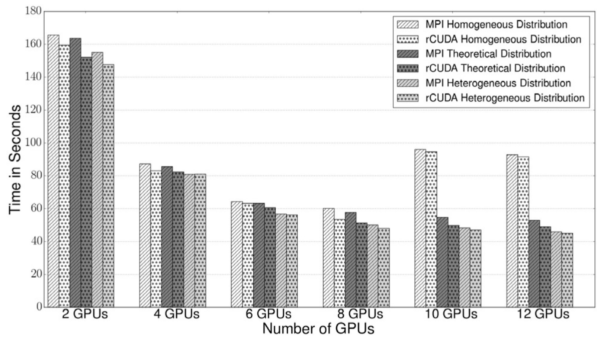


Fig. 4. Execution time (in seconds) with GPB complex and M1 for different distributions with MPI + OpenMP + CUDA and OpenMP + rCUDA implementations.

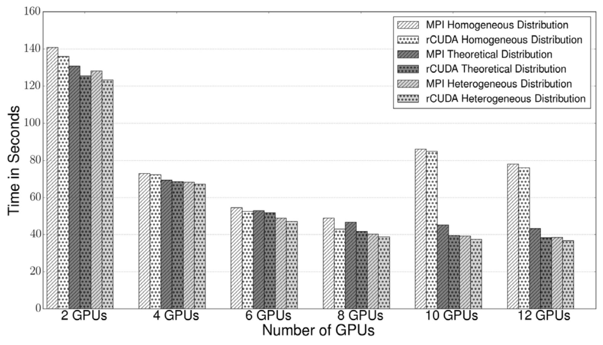


Fig. 5. Execution time (in seconds) with SRC complex and M1 for different distributions with MPI + OpenMP + CUDA and OpenMP + rCUDA implementations.

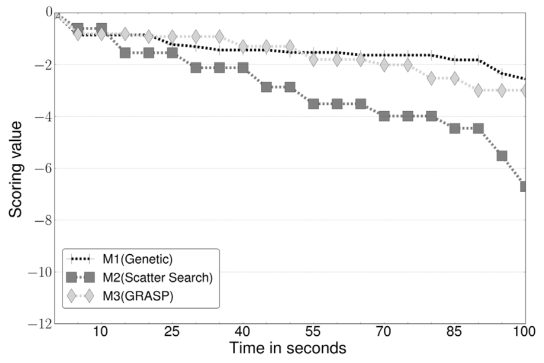


Fig. 6. Comparison of fitness with the COMT complex and 12 GPUs on rCUDA for three metaheuristics at different time-steps.

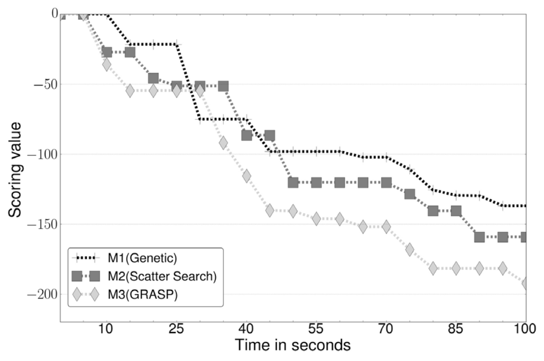


Fig. 7. Comparison of fitness with the GPB complex and 12 GPUs on rCUDA for three metaheuristics at different time-steps.

are particularly noticeable when Fermi GPUs are introduced (10–12 GPUs case). The reason is clear, we introduce a GPU that offers a lower theoretical peak performance and the same workload is assigned to each GPU. On the contrary, the theoretical distribution has a similar behavior to the heterogeneous distribution, increasing the difference between MPI and rCUDA as the complex is bigger (see Figs. 4 and 5). The heterogeneous distribution obtains the best performance. However, these execution times do not include the *warm-up* stage, which may be representative, as previously commented. For a real application where many steps are needed, the overhead would be less important. Furthermore, the application of the *warm-up* at installation stage would hide overhead.

5.4. Metaheuristic evaluation

In Sections 5.2 and 5.3 we have verified that using a greater number of GPUs reduces the execution time. In this section we show that: (1) different metaheuristics have different behavior for the same compound; (2) parallelism contributes to better fitnesses, with larger number of GPUs enabling more evaluations, and consequently better results, in the same execution time; (3) for the same reason, the exploitation of heterogeneity also contributes to better fitnesses. rCUDA is used in the experiments.

Figs. 6–8 show the evolution of the fitness with the time of the different metaheuristics and complexes considered with rCUDA. As commented, the behavior of the metaheuristics is different depending on the complex and no metaheuristic offers the best results with all the complexes. The worst results are obtained with metaheuristic M1 (close to a Genetic Algorithm). With COMT and SRC complexes, M2 (close to Scatter Search) obtains the best fitness (minor value), while for GPB, with a greater number of spots to look

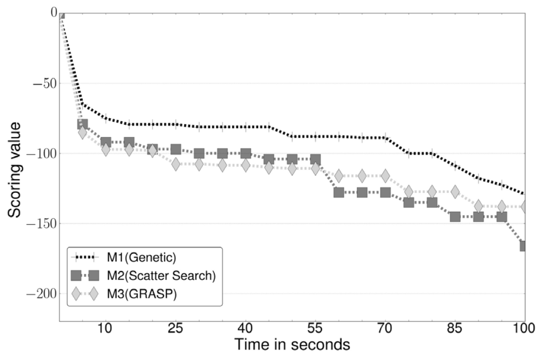


Fig. 8. Comparison of fitness with the SRC complex and 12 GPUs on rCUDA for three metaheuristics at different time-steps.

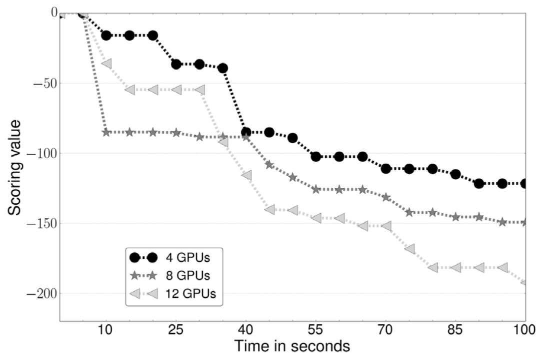


Fig. 11. Comparative fitness for heterogeneous distribution with GPB complex on 4, 8 and 12 GPUs on rCUDA with metaheuristic M3 and time-limit.

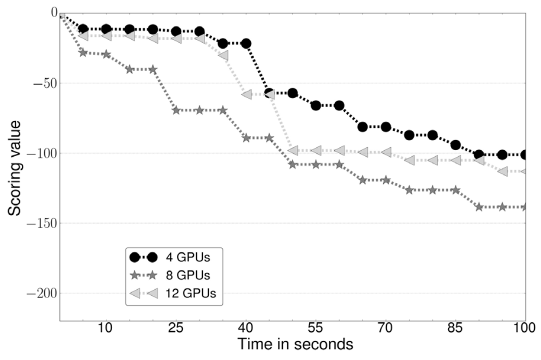


Fig. 9. Comparative fitness for homogeneous distribution with GPB complex on 4, 8 and 12 GPUs on rCUDA with metaheuristic M3 and time-limit.

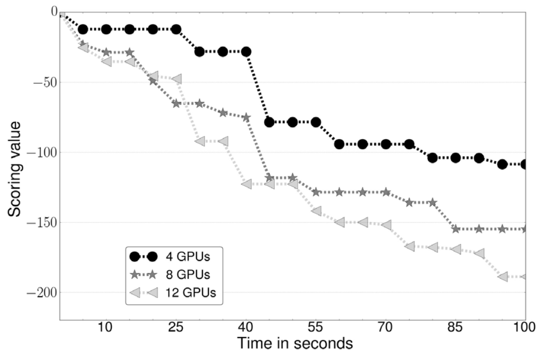


Fig. 10. Comparative fitness for theoretical distribution with GPB complex on 4, 8 and 12 GPUs on rCUDA with metaheuristic M3 and time-limit.

for and more individuals per spot, the behavior of the local search metaheuristic is better. Metaheuristic M3 is used in the following experiments to analyze the influence of the workload distributions in Section 5.3 on the fitness.

Figs. 9–11 analyze the quality of the solution with the number of GPUs with the three distributions (homogeneous, theoretical and heterogeneous). In Fig. 9 we observe that with homogeneous distribution the fitness with 12 GPUs is worse than with 8, which is due to the same workload for Fermi and Kepler GPUs. This makes the execution slower and the number of steps is smaller than if only Kepler GPUs are used. Figs. 10 and 11 show the behavior with the theoretical and heterogeneous distribution, in this case the quality of the fitness increases with the number of GPUs.

6. Related work

This section analyzes different docking techniques from a computational point of view. Widely-used docking programs like Autodock [6] and Autodock VINA [7] are CPU-based. They accelerate their computations by means of the OpenMP runtime to fully leverage multicore architectures. Other CPU-based docking programs are LeadFinder [9], Glide [8], SurFlex [10], ICM+ [11], DOCK [13] or FMD [12]. These applications use OpenMP, and some of them also use the MPI library in order to distribute their computations across the CPUs of several nodes of the cluster.

Moreover, there are other docking applications that use the GPU to offload the computational intensive parts of the computation. BUDE [15] is a software for molecular docking that leverages the heterogeneity of CPU–GPU systems. This software is programmed using OpenMP and OpenCL for portability to different architectures like NVIDIA and AMD GPUs, but it does not consider the use of the MPI technology. AMBER [16] also uses GPU acceleration. However, the possibility of using multiple GPUs requires using OpenMP or MPI, and therefore it cannot be directly compared with our environment. *BINDSURF* [17] is fully developed in GPU, although it does not support MultiGPU or MPI to extend the computation to several nodes in a cluster.

7. Conclusions and future work

Virtual screening methods are computational techniques that aid the drug discovery process. They are very computational demanding applications, and the use of clusters combining multicore CPUs and several GPUs contribute to accelerate their solution. However, these are heterogeneous systems, and heterogeneity may limit application performance unless programmers: (1) develop smart applications to control those features wisely on the road towards an optimal performance or (2) use middleware tools that manage these computing issues efficiently and transparently. This paper analyzes both options by developing initially an MPI, OPENMP and CUDA based Virtual Screening application and also by using rCUDA with only OpenMP and CUDA counterpart version of the same application. rCUDA offers an easy-to-use framework to develop data-intensive applications that use several remote GPUs transparently. We have not noticed any substantial pay off in terms of performance; actually some slightly improvements are reported in some cases, thanks to an efficient implementation of memory transfers through pinned buffers on rCUDA.

Virtual Screening requires the analysis of large data-bases of chemical compounds. Those compounds are independent of each other and, therefore, a load-balancing technique is necessary to distribute the workload efficiently among all GPUs, which can be from different generations. Here, three different load-balancing techniques are studied. Our baseline technique is a homogeneous distribution among GPUs which is not efficient as long as there are notable computational differences between GPUs. The theoretical distribution, based on the peak performance reported by the manufacturer, is a good option as it does not require extra computation and it determines relatively well performance differences between GPUs. The best performance is achieved by our load balancing technique based on a warm-up strategy. The execution time of the warm-up phase largely increases as long as the number of GPUs does so. In particular, the virtualization offered by rCUDA avoids parallelization at the warm-up and therefore the execution time can increase notably. Anyway, the warm-up times reported are affordable in real applications where the number of steps of the metaheuristics for solutions of high quality is large, or when the warm-up is carried out for representative metaheuristics and sizes during the installation of the docking method for a computational system. In summary, the main conclusions are that: (1) population-based metaheuristics hybridized with local search methods give satisfactory results for our docking problem; (2) parallelism can help to reduce both the execution time of this computationally demanding problem and the quality of the solutions; (3) a virtual system as rCUDA eases the exploitation of heterogeneous systems for the problem in hand; (4) to fully exploit this type of systems the heterogeneity should be considered for workload distribution, with a result in the improvement of the solutions as a consequence of the reduction of execution times.

For future work, and in order to deal with larger problems or for better solutions with limited execution times, it could be convenient to adapt our virtual screening method to even more complex systems, with other types of accelerators and with accelerators of various types and at different speeds in the same node. Energy efficiency should also be considered. In this paper, GPU virtualization has been proved as a very promising technique, and, therefore, we will follow this path, including multi-tenancy at a GPU level, by running several instances of our program in the same physical GPU to increase the overall throughput. Indeed, an energy efficiency evaluation in this context will be a very interesting subject of study, which will also comprise the usage of other system architectures like 64-bit ARM-based systems. Moreover, virtual screening is still at a relatively early stage, and we acknowledge that we have tested a relatively simple variant of the algorithm. But, with many other types of scoring functions still to be explored, this field seems to offer a promising and potentially fruitful area of research.

Acknowledgments

This work is jointly supported by the Fundación Séneca (Agencia Regional de Ciencia y Tecnología, Región de Murcia) under grant 18946/JLI/13, and by the Spanish MEC and European Commission FEDER under grants TIN2015-66972-C5-3-R and TIN2016-78799-P (AEI/FEDER, UE). We also thank NVIDIA for hardware donation under GPU Educational Center 2014–2016 and Research Center 2015–2016. Furthermore, researchers from Universitat Politècnica de València are supported by the Generalitat Valenciana under Grant PROMETEO/2017/077. Authors are also grateful for the generous support provided by Mellanox Technologies Inc.

References

[1] P.J. Hajduk, J. Greer, A decade of fragment-based drug design: Strategic advances and lessons learned, Nat. Rev. Drug Discov. 6 (3) (2007) 211–219.
[2] W.L. Jorgensen, The many roles of computation in drug discovery, Science 303 (2004) 1813–1818.
[3] J.M. Rollinger, H. Stuppner, T. Langer, Virtual screening for the discovery of bioactive natural products, in: Natural Compounds as Drugs, vol. I, Springer, 2008, pp. 211–249.
[4] J.J. Irwin, B.K. Shoichet, ZINC–a free database of commercially available compounds for virtual screening, J. Chem. Inf. Modeling 45 (1) (2005) 177–182.
[5] D.B. Kitchen, H. Decornez, J.R. Furr, J. Bajorath, Docking and scoring in virtual screening for drug discovery: Methods and applications, Nat. Rev. Drug Discov. 3 (11) (2004) 935–949.
[6] G.M. Morris, D.S. Goodsell, R.S. Halliday, R. Huey, W.E. Hart, R.K. Belew, A.J. Olson, Automated docking using a Lamarckian genetic algorithm and an empirical binding free energy function, J. Comput. Chem. 19 (14) (1998) 1639–1662.
[7] O. Trott, A.J. Olson, AutoDock VINA: Improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading, J. Comput. Chem. 31 (2) (2010) 455–461.
[8] R.A. Friesner, et al., Glide: A new approach for rapid, accurate docking and scoring: Method and assessment of docking accuracy, J. Med. Chem. 47 (7) (2004) 1739–1749.
[9] O.V. Stroganov, F.N. Novikov, V.S. Stroylov, Val Kulkov, G.G. Chilov, Lead finder: An approach to improve accuracy of protein-ligand docking, binding energy estimation, and virtual screening, J. Chem. Inf. Modeling 48 (12) (2008) 2371–2385.
[10] A.N. Jain, Surflex: Fully automatic flexible molecular docking using a molecular similarity-based search engine, J. Med. Chem. 46 (4) (2003) 499–511.
[11] P. Smielewski, A. Lavinio, I. Timofeev, D. Radolovich, I. Perkes, J.D. Pickard, M. Czosnyka, ICM+, a flexible platform for investigations of cerebrospinal dynamics in clinical practice, in: Acta Neurochirurgica Supplements, Springer, 2008, pp. 145–151.
[12] R. Dolezal, T.C. Ramalho, T.C. França, K. Kuca, Parallel flexible molecular docking in computational chemistry on high performance computing clusters, in: Computational Collective Intelligence, Springer, 2015, pp. 418–427.
[13] T.J.A. Ewing, S. Makino, A.G. Skillman, I.D. Kuntz, DOCK 4.0: Search strategies for automated molecular docking of flexible molecule databases, J. Comput. Aided Mol. Des. 15 (5) (2001) 411–428.
[14] Top500, Top500 supercomputer site, http://www.top500.org/, 2017 (accessed: 03.04.2017).
[15] S. McIntosh-Smith, J. Price, R.B. Sessions, A.A. Ibarra, High performance in silico virtual drug screening on many-core processors, Int. J. High Perform. Comput. 29 (2) (2015) 119–134.
[16] R. Salomon-Ferrer, A.W. Gotz, D. Poole, S. Le Grand, R.C. Walker, Routine microsecond molecular dynamics simulations with AMBER on GPUs. 2. Explicit solvent particle mesh Ewald, J. Chem. Theory Comput. 9 (9) (2013) 3878–3888.
[17] I. Sánchez-Linares, H. Pérez-Sánchez, J.M. Cecilia, J.M. García, High-throughput parallel blind virtual screening using BINDSURF, BMC Bioinformatics 13 (Suppl 14) (2012) S13.
[18] J. Carretero, et al., Optimizations to enhance sustainability of MPI applications, in: Proceedings of the 21st European MPI Users' Group Meeting, ACM, 2014, p. 145.
[19] M. Rosenblum, T. Garfinkel, Virtual machine monitors: Current technology and future trends, Computer 38 (5) (2005) 39–47.
[20] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, Xen and the art of virtualization, ACM SIGOPS Operating Systems Review 37 (5) (2003) 164–177.
[21] C. Reaño, F. Silla, G. Shainer, S. Schultz, Local and remote GPUs perform similar with EDR 100G InfiniBand, in: Proceedings of the 16th International Middleware Conference, in: Middleware '15, ACM, New York, NY, USA, 2015, pp. 4:1–4:7.
[22] S. Iserte, J. Prades, C. Reaño, F. Silla, Increasing the performance of data centers by combining remote GPU virtualization with slurm, in: 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), ACM, 2016, pp. 98–101.
[23] B. Imbernón, J.M. Cecilia, H. Pérez-Sánchez, D. Giménez, METADOCK: A parallel metaheuristic schema for virtual screening methods, Int. J. High Perform. Comput. (2017). http://dx.doi.org/10.1177/1094342017697471.
[24] A.A. Franco, Multiscale modelling and numerical simulation of rechargeable lithium ion batteries: Concepts, methods and challenges, RSC Adv. 3 (32) (2013) 13027–13058.
[25] N. Lagarde, J.-F. Zagury, M. Montes, Benchmarking data sets for the evaluation of virtual ligand screening methods: Review and perspectives, J. Chem. Inf. Modeling 55 (7) (2015) 1297–1307.
[26] A.N. Jain, Scoring functions for protein-ligand docking, Current Protein and Peptide Science 7 (5) (2006) 407–420.
[27] P. Csermely, T. Korcsmáros, H.J. Kiss, G. London, R. Nussinov, Structure and dynamics of molecular networks: A novel paradigm of drug discovery: A comprehensive review, Pharmacology & Therapeutics 138 (3) (2013) 333–408.
[28] J. Wang, Y. Deng, B. Roux, Absolute binding free energy calculations using molecular dynamics simulations with restraining potentials, Biophys J 91 (8) (2006) 2798–2814.
[29] L. Bianchi, M. Dorigo, L.M. Gambardella, W.J. Gutjahr, A survey on metaheuristics for stochastic combinatorial optimization, Natural Computing: An International Journal 8 (2) (2009) 239–287.

[30] C. Blum, A. Roli, Metaheuristics in combinatorial optimization: Overview and conceptual comparison, ACM Computing Surveys (CSUR) 35 (3) (2003) 268–308.
[31] G.R. Raidl, A unified view on hybrid metaheuristics, in: Hybrid Metaheuristics, Springer, 2006, pp. 1–12.
[32] P. Hansen, N. Mladenović, Variable neighborhood search, in: Search Methodologies, Springer, 2014, pp. 313–337.
[33] F. Almeida, D. Giménez, J.-J. López-Espín, M. Pérez-Pérez, Parameterised schemes of metaheuristics: Basic ideas and applications with genetic algorithms, scatter search and GRASP, IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans 43 (3) (2013) 570–586.
[34] L.-G. Cutillas-Lozano, J.-M. Cutillas-Lozano, D. Giménez, Modeling shared-memory metaheuristic schemes for electricity consumption, in: Distributed Computing and Artificial Intelligence – 2012 9th International Conference, 2012, pp. 33–40.
[35] J.-M. Cutillas-Lozano, D. Giménez, Determination of the kinetic constants of a chemical reaction in heterogeneous phase using parameterized metaheuristics, Procedia Computer Science 18 (2013) 787–796.
[36] T. Austin, Bridging the Moore's Law performance gap with innovation scaling, in: Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering, ACM, 2015, p. 1.
[37] D.B. Kirk, W.H. Wen-mei, Programming massively parallel processors: A hands-on approach, Elsevier, 2013.
[38] OpenMP Architecture Review Board, The OpenMP Specification, http://www.openmp.org, 2017 (accessed: 02.04.2017).
[39] H. Wang, S. Potluri, M. Luo, A.K. Singh, S. Sur, D.K. Panda, MVAPICH2-GPU: Optimized GPU to GPU communication for infiniband clusters, Comput. Sci. Res. Dev. 26 (3–4) (2011) 257.
[40] D.R. Kaeli, P. Mistry, D. Schaa, D.P. Zhang, Heterogeneous computing with OpenCL 2.0, Morgan Kaufmann, 2015.
[41] A.J. Peña, C. Reaño, F. Silla, R. Mayo, E.S. Quintana-Ortí, J. Duato, A complete and efficient CUDA-sharing solution for HPC clusters, Parallel Comput. 40 (10) (2014) 574–588.
[42] S.K. Kuntz, R.C. Murphy, M.T. Niemier, J.A. Izaguirre, P.M. Kogge, Petaflop computing for protein folding, in: Proceedings of the 2001 Tenth SIAM Conference on Parallel Processing for Scientific Computing, 2001, pp. 12–14.
[43] DUD, Directory of Useful Decoys, http://dud.docking.org/, 2006 (accessed: 03.04.2016).

Baldomero Imbernón received his B.S. degree in Computer Science from Catholic University of Murcia (Spain, 2013) and M.S. degree in New Technologies in Computer Science in University of Murcia (Spain, 2015) specialism High Performance Architectures and Supercomputing. In the last two years, he has authored several journal papers in the areas of bioinformatics and high performance computing. Actually, He is a predoctoral researcher at the Catholic University of Murcia (Spain)

Javier Prades obtained a MS degree in Computer Engineering from the Technical University of Valencia, Spain, in 2015. He is currently pursuing his Ph.D. studies in Computer Science and performs research tasks in the Parallel Architectures Group of this University. His areas of interest include distributed systems, parallel programming, virtualization solutions, interconnection networks, and cluster architectures.

Domingo Giménez is a Professor in the Computer Science Department at the University of Murcia, Spain. He has been a faculty member of the university since 1988, where he teaches algorithms and parallel computing. He received his degree in Mathematics from the University of Murcia in 1982, and his Ph.D. in Computer Science from the Polytechnic University of Valencia in 1995. His research interests include scientific applications of parallel computing, matrix computation, scheduling and software auto-tuning techniques.

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.

Federico Silla received the MS and Ph.D. degrees in Computer Engineering from Technical University of Valencia, Spain, in 1995 and 1999, respectively. He is currently an associate professor at that university. He worked for two years at Intel Corporation, developing on-chip networks. His research addresses high performance on-chip and off-chip interconnection networks as well as distributed memory systems and remote GPU virtualization mechanisms. In this regard, he is the coordinator of the rCUDA remote GPU virtualization project since it began in 2008. He has published more than 100 papers in peer-reviewed conferences and journals, as well as 10 book chapters, what currently translates into an H-index impact factor equal to 25 according to Google Scholar. He has been member of the Program Committee in several of the most prestigious conferences in his area, including IPDPS, ICPP, SC, PACT, ISCA, ICS, MICRO, etc. He is also an Associate Editor of Journal of Parallel and Distributed Computing.

Capítulo 3

Conclusiones y vías futuras

La tesis doctoral que se presenta en este compendio de publicaciones, tiene un hilo conductor y una línea investigadora ligada a la simbiosis entre paralelismo y bioinformática estructural. En este capítulo se sintetizan las conclusiones que podemos extraer de todo el desarrollo de la tesis y que perfectamente pueden convertirse en premisas para continuar la línea investigadora.

3.1 Conclusiones

Durante la realización de esta tesis doctoral, se han ido obteniendo una serie de resultados de forma gradual, que han ido cumpliendo los objetivos propuestos, y que se comentan a continuación.

Las resultados obtenidos de la aplicación de metaheurísticas paralelas para procesos de cribado virtual, han derivado en el desarrollo y publicación de una nueva metodología que comprende la primera de las publicaciones que componen el compendio. Esta primera publicación plasma la unión entre metaheurísticas y los procesos de cribado virtual creando de un método basado en un esquema parametrizado paralelo denominado METADOCK [38]. Este esquema es la base y el punto de partida para el resto de trabajos que se presentan en esta tesis doctoral. Con esta publicación, se convierten los primeros resultados obtenidos por la aplicación de metaheurísticas a procesos de cribado virtual [37] en un método general basado en metaheurísticas parametrizadas paralelas, con una calidad predictiva contrastada en bases de datos de relevancia.

METADOCK incorpora un conjunto paramétrico que incluye por un lado los referentes a la metaheurística elegida, y por otro, los valores relativos al paralelismo. Esta posibilidad permite trabajar con diferentes configuraciones en arquitecturas de memoria compartida y en GPUs de NVIDIA. Podemos destacar varias aportaciones que derivan de este trabajo:

- METADOCK utiliza la GPU para los cálculos más costosos, reduciendo el tiempo de ejecución con respecto al uso exclusivo de *multicore* CPU. Con los datos del estudio realizado, el incremento de aceleradores produce en METADOCK un aumento de rendimiento, dividendo la carga computacional entre todas las GPUs.

- Los nodos de cómputo disponen habitualmente de GPUs de diferentes generaciones. METADOCK es consciente de esta heterogeneidad, y al trabajar con varios dispositivos al mismo tiempo de diferentes familias tecnológicas, plantea la necesidad de diseñar una estrategia de balanceo de carga para conseguir equidad en el uso de las GPUs. Esta estrategia se basa en el diseño de una fase de preliminar, donde se reparte el trabajo en función de la capacidad computacional del dispositivo.

- Los primeros estudios de calidad predictiva del método, obtienen resultados satisfactorios en los primeros *benchmarks* sometidos a estudio.

Desarrollado METADOCK, podemos concluir que (1) el proceso de cribado virtual tiene un alto coste computacional, (2) METADOCK tiene un alto grado de parametrización que influye en la calidad predictiva del método y, (3) tiene un paralelismo natural inherente a la propia naturaleza del problema.

En el segundo trabajo de la tesis doctoral nos centramos en el grado de parametrización de METADOCK, que permite explorar un amplio conjunto de metaheurísticas. Este elevado número de metaheurísticas complica encontrar una combinación óptima. La estrategia propuesta con un nivel de abstracción superior denominado hiperheurística, permite explorar un conjunto elevado de metaheurísticas de forma aleatoria y obtener la mejor combinación paramétrica posible para abordar el problema.

El siguiente trabajo realizado, que figura como segundo artículo del compendio, eleva un nivel más de abstracción a METADOCK, diseñando un nuevo esquema superior aplicado a los procesos de cribado virtual que varía aleatoriamente el valor de los parámetros metaheurísticos. En los párrafos siguientes describimos las principales aportaciones de este trabajo:

- El nuevo esquema, denominado *hiperheurística*, toma decisiones en función del valor de la función de *scoring* obtenida con cada una de las combinaciones metaheurísticas. Este trabajo aborda además la paralelización en arquitecturas MIC *(Many Integrated Core)* como Intel Xeon Phi.

- La arquitectura *multicore* en CPU permite asignar hilos por niveles. Con el esquema metaheurístico aplicado al problema de cribado virtual se han implementando dos niveles de paralelismo, uno dedicado a trabajar los átomos del ligando y otro inferior para trabajar los átomos de la proteína. Con esta nueva abstracción tenemos dos niveles más y la ejecutamos sobre una arquitectura MIC. La posibilidad de generar entre 20 y 250 hilos permite explorar la asignación de diferente número de hilos a los distintos niveles de paralelismo, obteniendo mejores tiempos de ejecución conforme aumentamos el número de hilos asignados al esquema metaheurístico, en detrimento del nivel hiperheurístico superior.

- La arquitectura MIC y el uso masivo de los núcleos que lo componen permite obtener mejores tiempos de ejecución que el uso de arquitecturas *multicore* tradicionales, dado que permite una mayor exploración del espacio conformacional en el límite de tiempo establecido.

En el tercer artículo de la tesis doctoral, el alto coste computacional y la escalabilidad de METADOCK han sido sometidos a un estudio más profundo. En este estudio utilizamos un clúster de cómputo con GPUs de diferentes generaciones para explorar tanto la escalabilidad de la nueva metodología como su adaptabilidad a entornos virtualizados. En este caso, utilizamos rCUDA, que es un *framework* capaz de virtualizar un conjunto de GPUs que soporten CUDA repartidas por un clúster, haciendo que aparezcan como locales a un nodo. La publicación contiene un amplio estudio que abarca diversos ámbitos de la nueva metodología que pasamos a continuación a describir con mayor detalle:

- El tratamiento de la heterogeneidad del sistema a través de la función de balanceo de carga es vital para conseguir que la computación se distribuya de forma equitativa en función de las características de cada acelerador. Este primer estudio y posterior discusión compara el efecto de usar distintas funciones de balanceo, marcando los porcentajes y rendimientos del sistema en cada caso. En todos los casos estudiados, teniendo en cuenta la heterogeneidad de los dispositivos disponibles y realizando un reparto de trabajo acorde con las prestaciones, obtenemos importantes beneficios de rendimiento.

- Un segundo punto clave en este estudio, es analizar la facilidad de programación que podría aportar rCUDA evitando el uso de MPI a la hora de ejecutar la aplicación en un clúster heterogéneo. Tras los resultados comprobamos que el entorno virtualizado aporta, no solo una facilidad de programación, sino mejoras en el tiempo de ejecución.

- Cada una de las generaciones de GPUs aporta beneficios sobre la anterior y tiene diferentes evoluciones tecnológicas. Para afrontar este hecho, METADOCK identifica la computabilidad de la GPU en la ejecución de los respectivos *kernels*, pudiendo ejecutar conjuntos de instrucciones en unos dispositivos que otros por su familia tecnológica no permiten.

- El objetivo de la ejecución es minimizar el valor de la función de *scoring*. Se ha plasmado en esta publicación, un estudio sobre la evolución del valor de esta función entre las diferentes configuraciones paramétricas. Con los números obtenidos en el estudio, se observa la tendencia a obtener mejores valores en el mismo instante de tiempo conforme aumentamos los recursos computacionales, dado que podemos realizar un mayor número de operaciones y búsquedas en el mismo tiempo, conforme se incrementa la cantidad de recursos disponibles.

3.2 Vías futuras

En esta tesis doctoral se ha analizado, diseñado e implementado un aplicación para el descubrimiento de fármacos que sienta las bases de futuras líneas de trabajo.

Todas ellas van en la dirección de proporcionar soluciones computacionalmente más eficientes y de una mejor calidad predictiva en los procesos de cribado virtual.

En términos computacionales podemos destacar dos líneas de trabajo fundamentales: la ubicuidad del cómputo y la exploración de nuevos paradigmas de computación. En la primera de ellas, destacamos el análisis del consumo energético de nuestra aplicación para poder desarrollar una aplicación *power-aware* que sea capaz de ser ejecutada en múltiples dispositivos más allá de las soluciones HPC. Por otro lado, la segunda línea es la que representa una componente más disruptiva en términos computacionales. Nuestra aplicación simula el comportamiento molecular, este comportamiento llevado a nivel subatómico ha definido una nueva tendencia computacional denominada *quantum computing* [22]. Arquitecturas como la incorporada en *IBM Q systems* [36] o D-Wave Systems [2] y trabajos con hardware especializado [42], abren una vía de investigación para explorar nuevos caminos algorítmicos más eficientes y adaptados a procesos de cribado virtual.

La calidad predictiva del método viene determinada fundamentalmente por el modelo químico-físico que define la interacción molecular. Matemáticamente se define una función de *scoring* con diferentes términos donde (1) se puede incluir más términos dependiendo del problema químico a analizar o (2) se puede paramétrizar cada término para asignar un peso que ofrezca un nuevo grado de libertad en la aplicación. Sin lugar a dudas, esto ampliar la posibilidad de explorar nuevas soluciones de búsqueda para optimizar este nuevo grado de libertad.

Otra posible línea de continuidad va ligada a maximizar el *throughput* final del sistema en entornos virtualizados, planteando la ejecución de diversas instancias y ocupando así el tiempo ocioso de las GPUs. Esto se puede potenciar en este tipo de entornos donde el software aproveche los periodos ociosos de los recursos de alto rendimiento para superponer varias ejecuciones. Esta línea de trabajo, puede estar ligada con los métodos de dinámica molecular como GROMACS, que se comportan de forma distinta con distintas combinaciones de *multicore* CPU y GPU. Los patrones de cómputo en GPU de GROMACS son muy característicos, lanzando gran cantidad de *kernels* de corta duración, por lo que la optimización del *throughput* con este tipo de patrones de cómputo es una línea de investigación que podemos aplicar a otras disciplinas con comportamientos similares.

Finalmente, también estamos explorando la vía de aplicar métodos de aprendizaje máquina tipo *machine learning* tipo *deep learning*, que complementen o sustituyan a los métodos matemáticos tradicionales basados en función de *scoring*. Entrenando una red neuronal con bases de datos de éxitos, podemos determinar en una manera de búsqueda basada en el conocimiento previo.

Capítulo 4

Publicaciones, colaboraciones y calidad de las revistas

La tesis doctoral que se presenta, está formada por un compendio de publicaciones científicas publicadas en revistas del alto impacto e indexadas por JCR. A continuación se muestran sus referencias bibliográficas completas.

Título	*METADOCK: A parallel metaheuristic schema for virtual screening methods*
Revista	The International Journal of High Performance Computing Applications
Año	2017
DOI	10.1177/1094342017697471
Estado	Publicado

Autores – Afiliación	
Nombre	**D. Baldomero Imbernón Tudela**
Universidad	Universidad Católica de Murcia (UCAM)
Nombre	**Dr. José M. Cecilia Canales**
Universidad	Universidad Católica de Murcia (UCAM)
Nombre	**Dr. Horacio Pérez-Sánchez**
Universidad	Universidad Católica de Murcia (UCAM)
Nombre	**Dr. Domingo Giménez Cánovas**
Universidad	Universidad de Murcia

Detalles de la revista The International Journal of High Performance Computing Applications
Redactor jefe: Jack Dongarra
ISSN: 1094-3420 (versión impresa)
ISSN: 1741-2846 (versión electrónica)
Editorial: SAGE
Factor de impacto (2016): 2.097 (Journal Citation Reports)
Categoría: Computer Science, Hardware & Architecture
Ranking: Q2, posición: 15 de 52
Página Web: `http://journals.sagepub.com/home/hpc`

Título	*Exploiting multilevel parallelism on a many-core system for the application of hyperheuristics to a molecular docking problem*
Revista	The Journal of Supercomputing
Año	2017
DOI	10.1007/s11227-017-1989-7
Estado	Publicado

Autores – Afiliación (Orden Alfabético)	
Nombre	**Dr. José M. Cecilia Canales**
Universidad	Universidad Católica de Murcia (UCAM)
Nombre	**Dr. José Matías Cutillas Lozano**
Universidad	Universidad de Murcia
Nombre	**Dr. Domingo Giménez Cánovas**
Universidad	Universidad de Murcia
Nombre	**D. Baldomero Imbernón Tudela**
Universidad	Universidad Católica de Murcia (UCAM)

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 (2016): 1.326 (Journal Citation Reports)
Categoría: Computer Science, Hardware & Architecture
Ranking: Q2, posición: 23 de 51
Página Web: `https://link.springer.com/journal/11227`

Título	*Enhancing large-scale docking simulation on heterogeneous systems: an MPI vs rCUDA study*
Revista	Future Generation Computer Systems (FGCS)
Año	2017
DOI	10.1016/j.future.2017.08.050
Estado	Publicado

Autores – Afiliación	
Nombre	**D. Baldomero Imbernón Tudela**
Centro	Universidad Católica de Murcia (UCAM)
Nombre	**D. Javier Prades Gasulla**
Universidad	Universidad Politécnica de Valencia (UPV)
Nombre	**Dr. Domingo Giménez Cánovas**
Universidad	Universidad de Murcia
Nombre	**Dr. José M. Cecilia**
Universidad	Universidad Católica de Murcia (UCAM)
Nombre	**Dr. Federico Silla**
Universidad	Universidad Politécnica de Valencia (UPV)

Detalles de la revista *Future Generation Computer Systems*
Editor-in-Chief: Peter Sloot
ISSN: 0167-739X
Editorial: Elsevier
Factor de impacto (2016): 3.997 (Journal Citation Reports)
Categoría: Computer Science, Theory & Methods
Ranking: Q1, posición: 10 de 104
Página Web: `https://www.evise.com/profile/#/FGCS/login`

4.1 Colaboraciones

Durante el desarrollo de esta tesis doctoral, se han ido estrechando lazos de colaboración con otros grupos de investigación de diferentes universidades, relacionados con los diferentes objetivos perseguidos con esta tesis. Esto ha permitido construir una base de trabajo para futuras líneas de investigación. Vamos a detallar estas líneas de trabajo:

- La escalabilidad y trabajar con una arquitectura virtualizada de GPUs, más concretamente con rCUDA, está siendo un gran reto y ha sido una línea de trabajo muy importante en el desarrollo de esta tesis. La colaboración con el Grupo de Arquitecturas Paralelas (GAP) de la Universidad Politécnica de Valencia, y, en particular, con el Dr. Federico Silla Jiménez durante los años 2016 y 2017, ha dado sus frutos con una importante contribución que forma parte del compendio de esta tesis, *Enhancing large-scale docking simulation on heterogeneous systems: an MPI vs rCUDA study*. Además, seguimos una línea continuista en cuanto a la escalabilidad de sistemas, explorando comportamientos de otros software de referencia en el campo bioinformático en un sistema virtualizado.

- La eficiencia energética es un campo de estudio cada vez más importante en la computación de alto rendimiento, intentando equilibrar el binomio coste-resultados. Está colaboración se realiza con el Departamento de Arquitectura de Computadores de la Universidad de Málaga, y más concretamente con el Dr. Manuel Ujaldón Martínez. En el trabajo *Energy-based Tuning of Metaheuristics for Molecular Docking on Multi-GPUs*, primer *journal* de esta colaboración y sometido a una *major revision*, exploramos aspectos relacionados entre patrones de cómputo de diferentes estrategias heurísticas con el consumo energético.

- Durante el desarrollo de esta tesis doctoral, se ha comentado que el número de parámetros que tiene METADOCK es elevado. Para trabajar en la optimización paramétrica, se ha trazado una línea de colaboración con el Dr. José Juan López Espín de la Universidad Miguel Hernández de Elche, en un estudio estadístico para estudiar la influencia de cada uno de los parámetros del esquema en el resultado predictivo. Actualmente se están realizando los trabajos que van a formar parte del estudio y de la futura publicación.

- Otra colaboración que ha dado fruto a varias contribuciones, se realiza con el Grupo de Computación Científica y Programación Paralela de la Universidad de Murcia, concretamente con el Dr. Domingo Giménez Cánovas, que forma parte de la dirección de esta tesis, y con el que desde el año 2015 se han desarrollado dos contribuciones en congresos, directamente relacionadas con la consecución de esta tesis (1) y (2), y que sus datos de publicación figuran a continuación:

    1. *Imbernón, B., Cecilia, J. M., & Giménez, D. (2016). Enhancing metaheuristic-based virtual screening methods on massively parallel and heterogeneous systems. In Proceedings of the 7th international workshop on*

programming models and applications for multicores and manycores (pp. 50-58), PMAM 12-13 March 2016. ACM.

2. *Cecilia, J. M., Cutillas-Lozano, J. M., Giménez, D., & Imbernón, B. (2016). Exploiting multilevel parallelism on a many-core system for the application of hyperheuristics to a molecular docking problem. In Proceedings of the 16th International Conference on Computational and Mathematical Methods in Science and Engineering (pp. 346-353), CMMSE 2016 4–8 July, 2016.*

Ambas contribuciones a congresos han terminado por derivar a dos *journals* que forman parte del compendio de esta tesis, *METADOCK: A parallel metaheuristic schema for virtual screening methods* y *Exploiting multilevel parallelism on a many-core system for the application of hyperheuristics to a molecular docking problem.*

4.2 Otras publicaciones

Durante estos años, en el periodo de aprendizaje de la aplicación de computación de alto rendimiento a los procesos de cribado virtual, se han realizado contribuciones en otros campos de investigación, que derivaron en publicaciones en congresos y revistas de impacto. Los esfuerzos iniciales se centraron en la paralelización de la ecuación de difusión acústica y el cálculo del tiempo de reverberación en plataformas de cómputo de alto rendimiento como Intel Xeon Phi y GPUs de NVIDIA dando lugar a la publicación (1). Seguidamente se comenzó a explorar los procesos de cribado virtual, adaptando una aplicación orientada al descubrimiento de nuevos fármacos para la computación en la nube a través de la plataforma BOINC, publicando los resultados en (2). También se ha explorado el campo del ahorro energético, con un estudio realizado de la aplicación METADOCK en familias de NVIDIA de diferentes generaciones, observando los patrones de consumo y evaluando los resultados obtenidos por la aplicación con diferentes estrategias heurísticas con dichos patrones. En la publicación (3) se muestra el estudio completo, donde se identifican y comentan todos estos comportamientos. Además, se está profundizando en patrones paralelos para ejecutar modelos matemáticos que reproducen patrones de comportamiento atómicos en la naturaleza y que contribuyen a mejorar la calidad predictiva de métodos de cribado virtual basados en una función de *scoring* como METADOCK. En la publicación (4) se estudia el rendimiento de ejecutar un modelo matemático presente en funciones de *scoring* de varios programas de referencia, como Autodock o Autodock Vina, en la GPU. Los datos de publicación de dichas contribuciones expuestas son los siguientes:

1. *Hernández, M., Imbernón, B., Navarro, J. M., García, J. M., Cebrián, J. M., & Cecilia, J. M. (2015). Evaluation of the 3-D finite difference implementation of the acoustic diffusion equation model on massively parallel architectures. Computers & Electrical Engineering, 46, 190-201.*

2. *Guerrero, G. D., Imbernón, B., Pérez-Sánchez, H., Sanz, F., García, J. M., & Cecilia, J. M. (2014). A performance/cost evaluation for a GPU-based drug discovery application on volunteer computing. BioMed research international, 2014.*

3. *Pérez-Serrano, J., Imbernón, B., Cecilia, J. M. & Ujaldon, M.(2017). Energy-based Tuning of Metaheuristics for Molecular Docking on Multi-GPUs. Concurrency and Computation: Practice and Experience. (Major Revisión).*

4. *Saadi, H., Nouali-Taboudjemat, N., Rahmoun, A., Imbernón, B., Peréz-Sánchez, H., & Cecilia, J. M. (2017). Parallel desolvation energy term calculation for blind docking on GPU architectures. The Journal of Supercomputing (Under Review).*

Trabajar con patrones de paralelismo bioinspirados, es un campo que también se ha comenzado a explorar, extrayendo resultados que permitan aplicar estos patrones a aplicaciones como METADOCK para aumentar su capacidad predictiva, además de mejorar su rendimiento. Estos estudios preliminares han dado lugar a un conjunto de publicaciones en congresos y conferencias que paso a enumerar:

1. *Fang, J., Varbanescu, A. L., Imbernón, B., Cecilia, J. M., & Sánchez, H. E. P. (2014). Parallel Computation of Non-Bonded Interactions in Drug Discovery: Nvidia GPUs vs. Intel Xeon Phi. In International Conference on Bioinformatics and Biomedical Engineering (IWBBIO) 7 - 9 April, (pp. 579-588)*

2. *Cerón-Carrasco, J. P., Cerezo, J., Zuñiga, J., Requena, A., Contreras-Garcia, J., Chavan, S., Imbernon, B. Cecilia, J. M. & Sánchez, H. E. P. (2014). Application of parallel blind docking with BINDSURF for the study of platinum derived compounds as anticancer drugs. In International Conference on Bioinformatics and Biomedical Engineering (IWBBIO) 7 - 9 April, (pp. 972-976).*

3. *Imbernón, B., Llanes, A., Peña-García, J., Abellán, J. L., Pérez-Sánchez, H., & Cecilia, J. M. (2015). Enhancing the parallelization of non-bonded interactions kernel for virtual screening on GPUs. In International Conference on Bioinformatics and Biomedical Engineering (IWBBIO), 15 - 17 April, (pp. 620-626). Springer, Cham.*

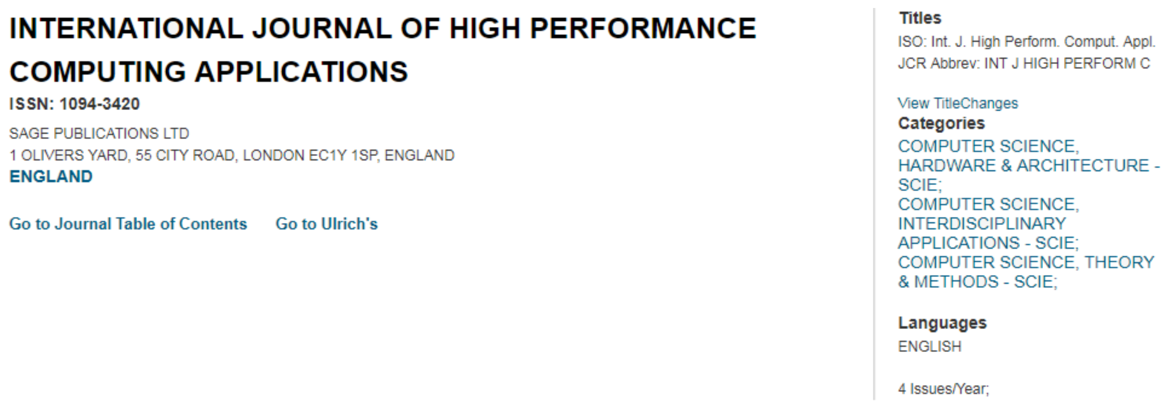
4. *Saadi, H., Nouali-Taboudjemat, N., Rahmoun, A., Imbernón, B., Peréz-Sánchez, H., & Cecilia, J. M. (2017). Parallel desolvation energy term calculation for blind docking on GPU architectures. In Parallel Processing Workshops (ICPPW), 2017 46th International Conference on (pp. 16-22). IEEE.*

4.3 Datos relativos a la calidad de las publicaciones

La calidad de una publicación se mide en función de una serie de indicadores estandarizados aplicados a las revistas. En esta tesis doctoral, los artículos que forman parte del compendio están publicados en revistas situadas en los dos primeros quartiles según el índice JCR. En las siguientes subsecciones se muestran una serie de indicadores asociados a cada revista donde se han publicado los trabajos.

4.3.1 METADOCK: A parallel metaheuristic schema for virtual screening methods - The International Journal of High Performance Computing Application

En este apartado presentamos los datos asociados a la revista donde ha sido publicado el primer artículo que forma parte de esta tesis *METADOCK: A parallel metaheuristic schema for virtual screening methods*. En las figuras 4.1, 4.2 y 4.3 presentamos los indicadores de calidad y el factor de impacto de la revista *The International Journal of High Performance Computing Applications*.



INTERNATIONAL JOURNAL OF HIGH PERFORMANCE COMPUTING APPLICATIONS

ISSN: 1094-3420

SAGE PUBLICATIONS LTD
1 OLIVERS YARD, 55 CITY ROAD, LONDON EC1Y 1SP, ENGLAND
ENGLAND

Go to Journal Table of Contents   Go to Ulrich's

Titles
ISO: Int. J. High Perform. Comput. Appl.
JCR Abbrev: INT J HIGH PERFORM C

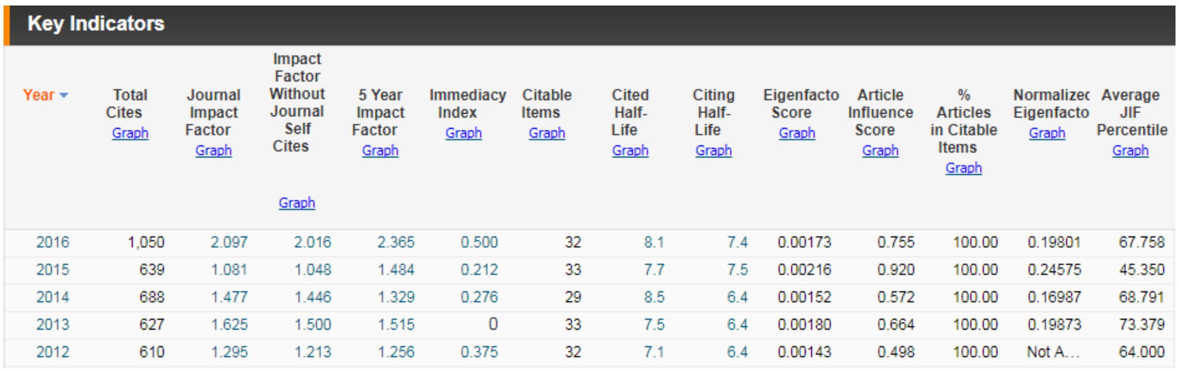
View TitleChanges

Categories
COMPUTER SCIENCE, HARDWARE & ARCHITECTURE - SCIE;
COMPUTER SCIENCE, INTERDISCIPLINARY APPLICATIONS - SCIE;
COMPUTER SCIENCE, THEORY & METHODS - SCIE;

Languages
ENGLISH

4 Issues/Year;

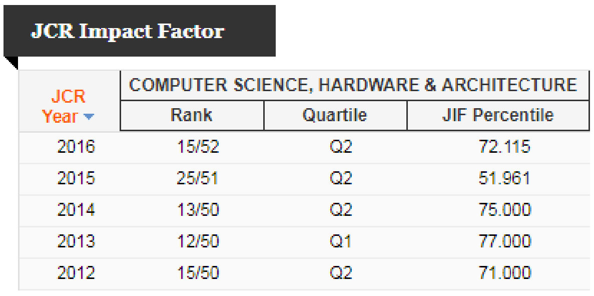
Figura 4.1: Título y datos de la revista donde se ha publicado el artículo *METADOCK: A parallel metaheuristic schema for virtual screening methods*.



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
2016	1,050	2.097	2.016	2.365	0.500	32	8.1	7.4	0.00173	0.755	100.00	0.19801	67.758
2015	639	1.081	1.048	1.484	0.212	33	7.7	7.5	0.00216	0.920	100.00	0.24575	45.350
2014	688	1.477	1.446	1.329	0.276	29	8.5	6.4	0.00152	0.572	100.00	0.16987	68.791
2013	627	1.625	1.500	1.515	0	33	7.5	6.4	0.00180	0.664	100.00	0.19873	73.379
2012	610	1.295	1.213	1.256	0.375	32	7.1	6.4	0.00143	0.498	100.00	Not A...	64.000

Figura 4.2: Indicadores clave de los últimos cinco años de la revista *The International Journal of High Performance Computing Applications*.



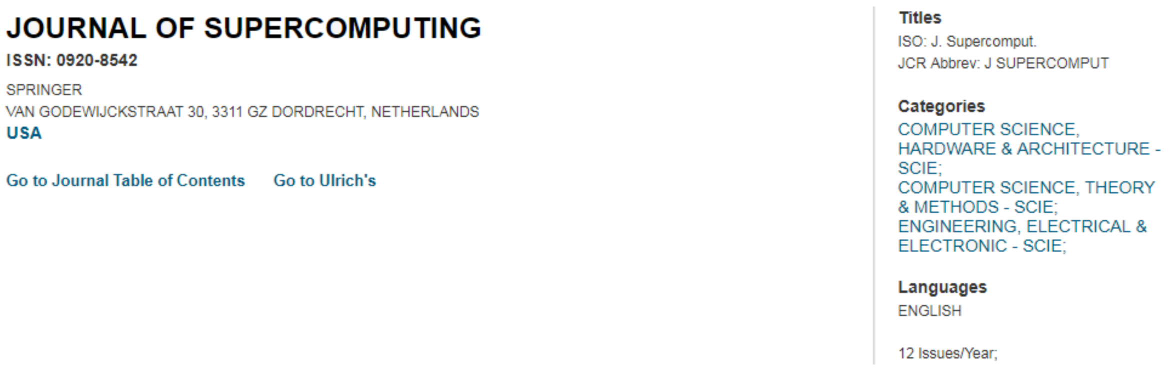
JCR Impact Factor

JCR Year	COMPUTER SCIENCE, HARDWARE & ARCHITECTURE		
	Rank	Quartile	JIF Percentile
2016	15/52	Q2	72.115
2015	25/51	Q2	51.961
2014	13/50	Q2	75.000
2013	12/50	Q1	77.000
2012	15/50	Q2	71.000

Figura 4.3: Factor de impacto de los últimos cinco años de la revista *The International Journal of High Performance Computing Applications*.

4.3.2 Exploiting multilevel parallelism on a many-core system for the application of hyperheuristics to a molecular docking problem - The Journal of Supercomputing

En este apartado presentamos los datos asociados a la revista donde ha sido publicado el segundo artículo que forma parte de esta tesis *Exploiting multilevel parallelism on a many-core system for the application of hyperheuristics to a molecular docking problem*. En las figuras 4.4, 4.5 y 4.6 presentamos los indicadores de calidad y el factor de impacto de la revista *The Journal of Supercomputing*.



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;

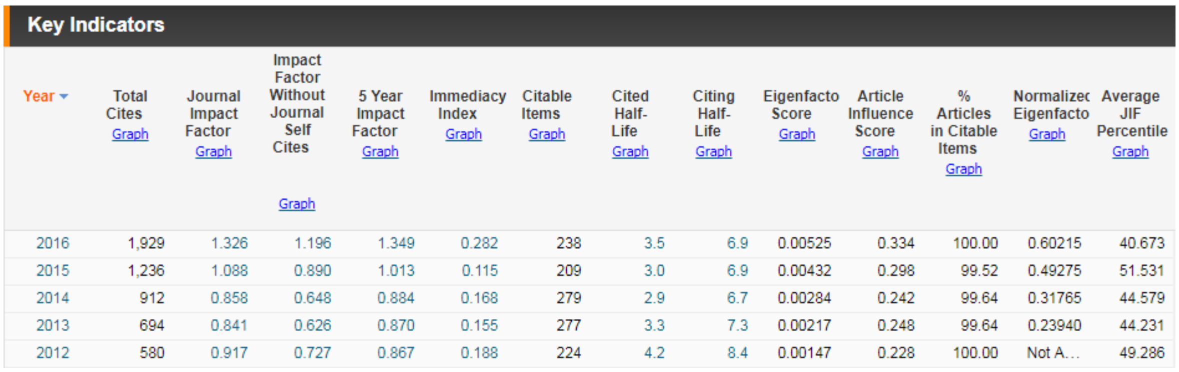
ENGINEERING, ELECTRICAL & ELECTRONIC - SCIE;

Languages

ENGLISH

12 Issues/Year;

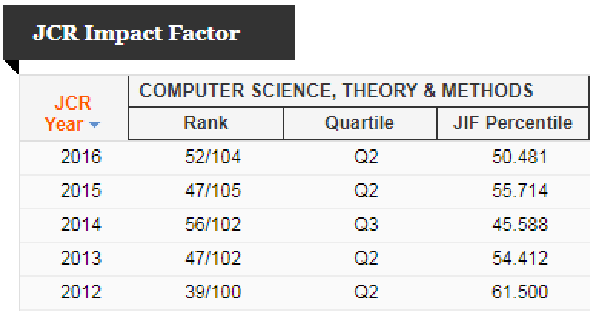
Figura 4.4: Título y datos de la revista donde se ha publicado el artículo *Exploiting multilevel parallelism on a many-core system for the application of hyperheuristics to a molecular docking problem*.



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
2016	1,929	1.326	1.196	1.349	0.282	238	3.5	6.9	0.00525	0.334	100.00	0.60215	40.673
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

Figura 4.5: Indicadores clave de los últimos cinco años de la revista *The Journal of Supercomputing.*



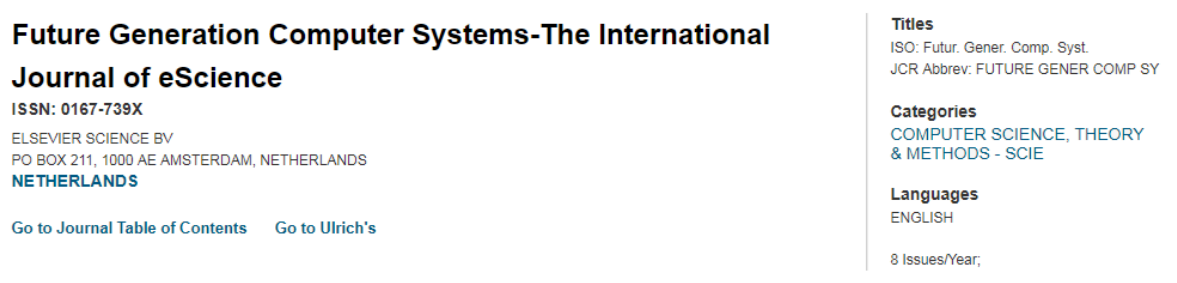
JCR Impact Factor

JCR Year	COMPUTER SCIENCE, THEORY & METHODS		
	Rank	Quartile	JIF Percentile
2016	52/104	Q2	50.481
2015	47/105	Q2	55.714
2014	56/102	Q3	45.588
2013	47/102	Q2	54.412
2012	39/100	Q2	61.500

Figura 4.6: Factor de impacto de los últimos cinco años de la revista *The Journal of Supercomputing.*

4.3.3 Enhancing large-scale docking simulation on heterogeneous systems: an MPI vs rCUDA study - Future Generation Computer System

En este apartado presentamos los datos asociados a la revista donde ha sido publicado el tercer artículo que forma parte de esta tesis *Enhancing large-scale docking simulation on heterogeneous systems: an MPI vs rCUDA study.* En las figuras 4.7, 4.8 y 4.9 presentamos los indicadores de calidad y el factor de impacto de la revista *Future Generation Computer System.*



Future Generation Computer Systems-The International Journal of eScience

ISSN: 0167-739X

ELSEVIER SCIENCE BV
PO BOX 211, 1000 AE AMSTERDAM, NETHERLANDS
NETHERLANDS

Go to Journal Table of Contents   Go to Ulrich's

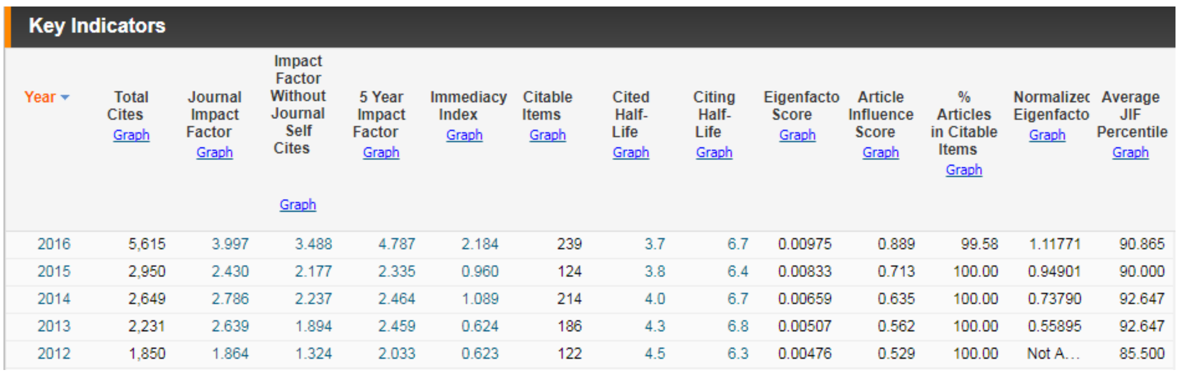
Titles
ISO: Futur. Gener. Comp. Syst.
JCR Abbrev: FUTURE GENER COMP SY

Categories
COMPUTER SCIENCE, THEORY & METHODS - SCIE

Languages
ENGLISH

8 Issues/Year;

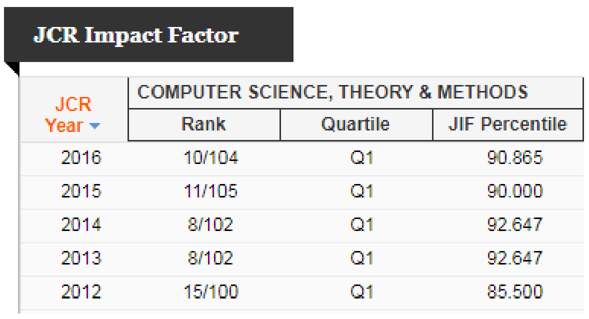
Figura 4.7: Título y datos de la revista donde se ha publicado el artículo *Enhancing large-scale docking simulation on heterogeneous systems: an MPI vs rCUDA study.*



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
2016	5,615	3.997	3.488	4.787	2.184	239	3.7	6.7	0.00975	0.889	99.58	1.11771	90.865
2015	2,950	2.430	2.177	2.335	0.960	124	3.8	6.4	0.00833	0.713	100.00	0.94901	90.000
2014	2,649	2.786	2.237	2.464	1.089	214	4.0	6.7	0.00659	0.635	100.00	0.73790	92.647
2013	2,231	2.639	1.894	2.459	0.624	186	4.3	6.8	0.00507	0.562	100.00	0.55895	92.647
2012	1,850	1.864	1.324	2.033	0.623	122	4.5	6.3	0.00476	0.529	100.00	Not A...	85.500

Figura 4.8: Indicadores clave de los últimos cinco años de la revista *Future Generation Computer System.*



JCR Impact Factor

JCR Year	COMPUTER SCIENCE, THEORY & METHODS		
	Rank	Quartile	JIF Percentile
2016	10/104	Q1	90.865
2015	11/105	Q1	90.000
2014	8/102	Q1	92.647
2013	8/102	Q1	92.647
2012	15/100	Q1	85.500

Figura 4.9: Factor de impacto de los últimos cinco años de la revista *Future Generation Computer System.*

Bibliografía

[1] Automatic Heuristic Generation with Genetic Programming: Evolving a Jack-of-all-Trades or a Master of One.

[2] D-Wave Systems, (accessed, Dicember, 10th, 2017). `https://www.dwavesys.com`.

[3] Top 500 supercomputer site, (accessed, October, 20th, 2017). `http://www.top500.org/`.

[4] Francisco Almeida, Domingo Giménez, Jose Juan López-Espín, and Melquíades Pérez-Pérez. Parameterized schemes of metaheuristics: Basic ideas and applications with genetic algorithms, scatter search, and grasp. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 43(3):570–586, 2013.

[5] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the Art of Virtualization. *ACM SIGOPS Operating Systems Review*, 37(5):164–177, 2003.

[6] Christian Blum and Andrea Roli. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys (CSUR)*, 35(3):268–308, 2003.

[7] Tracy D Braun, Howard Jay Siegel, Noah Beck, Ladislau L Bölöni, Muthucumaru Maheswaran, Albert I Reuther, James P Robertson, Mitchell D Theys, Bin Yao, and Debra Hensgen. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed computing*, 61(6):810–837, 2001.

[8] Edmund Burke, Graham Kendall, Jim Newall, Emma Hart, Peter Ross, and Sonia Schulenburg. Hyper-heuristics: An emerging direction in modern search technology. *Handbook of metaheuristics*, pages 457–474, 2003.

[9] Edmund K Burke, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R Woodward. A Classification of Hyper-heuristic Approaches. In *Handbook of metaheuristics*, pages 449–468. Springer, 2010.

[10] Edmund K Burke, Graham Kendall, and Eric Soubeiga. A tabu-Search Hyperheuristic for Timetabling and Rostering. *Journal of heuristics*, 9(6):451–470, 2003.

[11] Jesus Carretero and et al. Optimizations to enhance sustainability of MPI applications. In *Proceedings of the 21st European MPI Users' Group Meeting*, page 145. ACM, 2014.

[12] Adrián Castelló, Rafael Mayo, Judit Planas, and Enrique S Quintana-Ortí. Exploiting task-parallelism on GPU clusters via OmpSs and rCUDA virtualization. In *Trustcom/BigDataSE/ISPA, 2015 IEEE*, volume 3, pages 160–165. IEEE, 2015.

[13] José M Cecilia, José-Matías Cutillas-Lozano, Domingo Giménez, and Baldomero Imbernón. Exploiting multilevel parallelism on a many-core system for the application of hyperheuristics to a molecular docking problem. *The Journal of Supercomputing*, pages 1–12, 2017.

[14] George Chrysos. Intel® Xeon Phi™ coprocessor-the architecture. *Intel Whitepaper*, 176, 2014.

[15] Stephen A Cook. The Complexity of Theorem-Proving Procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

[16] José-Matías Cutillas-Lozano and Domingo Giménez. Optimizing Shared-memory Hyperheuristics on Top of Parameterized Metaheuristics. *Procedia Computer Science*, 29:20–29, 2014.

[17] Joseph A DiMasi, Henry G Grabowski, and Ronald W Hansen. Innovation in the pharmaceutical industry: New estimates of R&D costs. *Journal of health economics*, 47:20–33, 2016.

[18] James Dinan, Pavan Balaji, Darius Buntinas, David Goodell, William Gropp, and Rajeev Thakur. An implementation and evaluation of the MPI 3.0 one-sided communication interface. *Concurrency and Computation: Practice and Experience*, 28(17):4385–4404, 2016.

[19] Rafael Dolezal, Teodorico C Ramalho, Tanos CC França, and Kamil Kuca. Parallel Flexible Molecular Docking in Computational Chemistry on High Performance Computing Clusters. In *Computational Collective Intelligence*, pages 418–427. Springer, 2015.

[20] Todd J. A. Ewing, Shingo Makino, A. Geoffrey Skillman, and Irwin D. Kuntz. DOCK 4.0: Search strategies for automated molecular docking of flexible molecule databases. *Journal of Computer-Aided Molecular Design*, 15(5):411–428, 2001.

[21] Felcy Fabiola, Richard Bertram, Andrei Korostelev, and Michael S Chapman. An improved hydrogen bond potential: impact on medium resolution protein structures. *Protein Science*, 11(6):1415–1423, 2002.

[22] Richard P Feynman. Quantum mechanical computers. *Foundations of physics*, 16(6):507–531, 1986.

[23] Richard A Friesner and et al. Glide: A New Approach For Rapid, Accurate Docking and Scoring 1. Method and Assessment of Docking Accuracy. *Journal of Medicinal Chemistry*, 47(7):1739–1749, 2004.

[24] Alex Fukunaga. Automated Discovery of Composite SAT Variable-Selection Heuristics. In *American Association for Artificial Intelligence*, pages 641–648, 2002.

[25] Michael R Garey, David S. Johnson, and Larry Stockmeyer. Some simplified NP-complete graph problems. *Theoretical computer science*, 1(3):237–267, 1976.

[26] Michel Gendreau. An Introduction to Tabu Search. *Handbook of Metaheuristics*, pages 37–54, 2003.

[27] Michel Gendreau and Jean-Yves Potvin. *Handbook of Metaheuristics*, volume 2. Springer, 2010.

[28] Michael K Gilson, Kim A Sharp, and Barry H Honig. Calculating the electrostatic potential of molecules in solution: method and error assessment. *Journal of computational chemistry*, 9(4):327–335, 1988.

[29] Ginés D Guerrero, Juan M Cebrián, Horacio Pérez-Sánchez, José M García, Manuel Ujaldón, and José M Cecilia. Toward energy efficiency in heterogeneous processors: findings on virtual screening methods. *Concurrency and Computation: Practice and Experience*, 26(10):1832–1846, 2014.

[30] Thomas A Halgren. Merck molecular force field. ii. mmff94 van der waals and electrostatic parameters for intermolecular interactions. *Journal of Computational Chemistry*, 17(5-6):520–552, 1996.

[31] Pierre Hansen and Nenad Mladenović. Variable neighborhood search. In *Search Methodologies*, pages 313–337. Springer, 2014.

[32] Job Harenberg and Martin Wehling. Current and Future Prospects for Anticoagulant Therapy: Inhibitors of factor Xa and factor IIa. In *Seminars in thrombosis and hemostasis*, volume 34, pages 039–057. © Thieme Medical Publishers, 2008.

[33] Alex Herrera. NVIDIA GRID: Graphics accelerated VDI with the visual performance of a workstation. *Nvidia Corp*, 2014.

[34] Berk Hess, Carsten Kutzner, David Van Der Spoel, and Erik Lindahl. GROMACS 4: Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation. *Journal of Chemical Theory and Computation*, 4(3):435–447, 2008.

[35] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.* MIT press, 1992.

[36] IBM. IBM Q System. `https://developer.ibm.com/dwblog/2017/quantum-computing-16-qubit-processor/`, 2017. (accessed, October, 20th, 2017).

[37] Baldomero Imbernón, José M Cecilia, and Domingo Giménez. Enhancing metaheuristic-based virtual screening methods on massively parallel and heterogeneous systems. In *Proceedings of the 7th International Workshop on Programming Models and Applications for Multicores and Manycores*, pages 50–58. ACM, 2016.

[38] Baldomero Imbernón, José M. Cecilia, Horacio Pérez-Sánchez, and Domingo Giménez. METADOCK: A Parallel Metaheuristic schema for Virtual Screening methods. *The International Journal of High Performance Computing Applications*, March 2017.

[39] Baldomero Imbernón, Javier Prades, Domingo Giménez, José M Cecilia, and Federico Silla. Enhancing large-scale docking simulation on heterogeneous systems: An MPI vs rCUDA study. *Future Generation Computer Systems*, 79:26–37, 2018.

[40] Ajay N Jain. Surflex: Fully Automatic Flexible Molecular Docking Using a Molecular Similarity-Based Search Engine. *Journal of Medicinal Chemistry*, 46(4):499–511, 2003.

[41] Ajay N Jain. Scoring Functions for Protein-Ligand Docking. *Current Protein and Peptide Science*, 7(5):407–420, 2006.

[42] Abhinav Kandala, Antonio Mezzacapo, Kristan Temme, Maika Takita, Markus Brink, Jerry M Chow, and Jay M Gambetta. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature*, 549(7671):242–246, 2017.

[43] Arpan Kumar Kar. Bio inspired computing - A review of algorithms and scope of applications. *Expert Systems with Applications*, 59:20–32, 2016.

[44] Robert E Keller and Riccardo Poli. Cost-Benefit Investigation of a Genetic-Programming Hyperheuristic. In *International Conference on Artificial Evolution (Evolution Artificielle)*, pages 13–24. Springer, 2007.

[45] David B Kirk and W Hwu Wen-Mei. *Programming Massively Parallel Processors: A Hands-on Approach.* Morgan kaufmann, 2016.

[46] Douglas B Kitchen, Hélène Decornez, John R Furr, and Jürgen Bajorath. Docking and scoring in virtual screening for drug discovery: methods and applications. *Nature reviews. Drug discovery*, 3(11):935, 2004.

[47] Victor W Lee and et al. Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU. *ACM SIGARCH computer architecture news*, 38(3):451–460, 2010.

[48] Isabella WY Mak, Nathan Evaniew, and Michelle Ghert. Lost in translation: animal models and clinical trials in cancer treatment. *American journal of translational research*, 6(2):114, 2014.

[49] Simon McIntosh-Smith, James Price, Richard B Sessions, and Amaurys A Ibarra. High performance in silico virtual drug screening on many-core processors. *The International Journal of High Performance Computing Applications*, 29(2):119–134, 2015.

[50] Belén Melián, José A Moreno Pérez, and J Marcos Moreno Vega. Metaheurísticas: Una visión global. *Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial*, 7(19):0, 2003.

[51] Elaine C Meng, Brian K Shoichet, and Irwin D Kuntz. Automated Docking with Grid-Based Energy Evaluation. *Journal of Computational Chemistry*, 13(4):505–524, 1992.

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

[53] Gordon E Moore. Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp. 114 ff. *IEEE Solid-State Circuits Society Newsletter*, 20(3):33–35, 2006.

[54] G. M. Morris, D. S. Goodsell, R. S. Halliday, R. Huey, W. E. Hart, R. K. Belew, and A. J. Olson. Automated Docking using a Lamarckian Genetic Algorithm and an Empirical Binding Free Energy Function. *Journal of Computational Chemistry*, 19(14):1639–1662, 1998.

[55] NVIDIA. NVIDIA Tesla P100. `http://www.nvidia.com/object/tesla-p100.html`, 2017. (accessed, October, 20th, 2017).

[56] NVIDIA Corporation. *NVIDIA CUDA C Programming Guide 8.0.* 2017.

[57] Stephen L Olivier and Jan F Prins. Comparison of openmp 3.0 and other task parallel frameworks on unbalanced task graphs. *International Journal of Parallel Programming*, 38(5):341–360, 2010.

[58] Horacio Pérez-Sánchez, Vahid Rezaei, Vitaliy Mezhuyev, Duhu Man, Jorge Peña-García, Helena den Haan, and Sandra Gesing. Developing science gateways for drug discovery in a grid environment. *SpringerPlus*, 5(1):1300, 2016.

[59] David Pisinger and Stefan Ropke. A general heuristic for vehicle routing problems. *Computers & operations research*, 34(8):2403–2435, 2007.

[60] Günther R Raidl. A Unified View on Hybrid Metaheuristics. *Hybrid metaheuristics*, 4030:1–12, 2006.

[61] Carlos Reaño, Federico Silla, Gilad Shainer, and Scot Schultz. Local and Remote GPUs Perform Similar with EDR 100G InfiniBand. In *Proceedings of the 16th International Middleware Conference*, Middleware '15, pages 4:1–4:7, New York, NY, USA, 2015. ACM.

[62] Daniel A Reed and Jack Dongarra. Exascale computing and big data. *Communications of the ACM*, 58(7):56–68, 2015.

[63] Christian P Robert. *Monte carlo methods.* Wiley Online Library, 2004.

[64] Mendel Rosenblum and Tal Garfinkel. Virtual Machine Monitors: Current Technology and Future Trends. *Computer*, 38(5):39–47, 2005.

[65] Peter Ross. Hyper-heuristics. *Search Methodologies*, pages 529–556, 2005.

[66] Boris Ryabko and Anton Rakitskiy. Application of the Computer Capacity to the Analysis of Processors Evolution. *arXiv preprint arXiv:1705.07730*, 2017.

[67] Romelia Salomon-Ferrer, Götz Andreas W, Duncan Poole, Scott Le Grand, and Ross C Walker. Routine Microsecond Molecular Dynamics Simulations with AMBER on GPUs. 2. Explicit Solvent Particle Mesh Ewald. *Journal of chemical theory and computation*, 9(9):3878–3888, 2013.

[68] Irene Sánchez-Linares, Horacio Pérez-Sánchez, José M Cecilia, and José M García. High-throughput parallel blind virtual screening using BINDSURF. *BMC Bioinformatics*, 13(Suppl 14):S13, 2012.

[69] Brian K Shoichet, Irwin D Kuntz, and Dale L Bodian. Molecular Docking using Shape Descriptors. *Journal of Computational Chemistry*, 13(3):380–397, 1992.

[70] P Smielewski, A Lavinio, I Timofeev, D Radolovich, I Perkes, JD Pickard, and M Czosnyka. ICM+, a flexible platform for investigations of cerebrospinal dynamics in clinical practice. In *Acta Neurochirurgica Supplements*, pages 145–151. Springer, 2008.

[71] Oleg V Stroganov, Fedor N Novikov, Viktor S Stroylov, Val Kulkov, and Ghermes G Chilov. Lead Finder: An Approach To Improve Accuracy of Protein-Ligand Docking, Binding Energy Estimation, and Virtual Screening. *Journal of Chemical Information and Modeling*, 48(12):2371–2385, 2008.

[72] T Stützle. Local Search Algorithms for Combinatorial Problems. *Darmstadt University of Technology PhD Thesis*, 20, 1998.

[73] Elie Track, Nancy Forbes, and George Strawn. The End of Moore's Law. *Computing in Science & Engineering*, 19(2):4–6, 2017.

[74] Oleg Trott and Arthur J Olson. Autodock Vina: Improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading. *Journal of Computational Chemistry*, 31(2):455–461, 2010.

[75] Rob JM Vaessens, Emile HL Aarts, and Jan Karel Lenstra. A Local Search Template. *Computers & Operations Research*, 25(11):969–979, 1998.

[76] Hao Wang, Sreeram Potluri, Miao Luo, Ashish Kumar Singh, Sayantan Sur, and Dhabaleswar K Panda. MVAPICH2-GPU: optimized GPU to GPU communication for infiniband clusters. *Computer Science-Research and Development*, 26(3-4):257, 2011.

[77] Gregory L Warren and et al. A Critical Assessment of Docking Programs and Scoring Functions. *Journal of Medicinal Chemistry*, 49(20):5912–5931, 2006.