CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2013; **00**:1–15
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe

Towards Energy Efficiency in Heterogeneous Processors: Findings on Virtual Screening Methods

Ginés D. Guerrero[1*], Juan M. Cebrián[2], Horacio Pérez-Sánchez[3], José M. García[1]
Manuel Ujaldón[4] and José M. Cecilia[3]

[1]*Dept. of Computer Architecture, University of Murcia, 30080, Murcia, Spain*
[2]*Dept. of Computer and Information Science, 7034, Trondheim, Norway*
[3]*Dept. of Computer Science, Catholic University of Murcia, 30107, Murcia, Spain*
[4]*Dept. of Computer Architecture, University of Malaga, 29071, Malaga, Spain*

SUMMARY

The integration of the latest breakthroughs in computational modeling and High Performance Computing (HPC) has leveraged advances in the fields of healthcarea and drug discovery, among others. By integrating all these developments together, scientists are creating new exciting personal therapeutic strategies for living longer that were unimaginable not that long ago. However, we are witnessing the biggest revolution in HPC of the last decade. Several Graphics Processing Unit architectures have established their niche in the HPC arena, but at the expense of an excessive power and heat. A solution for this important problem is based on heterogeneity.
In this paper, we analyze power consumption on heterogeneous systems, benchmarking a bioinformatics kernel within the framework of Virtual Screening (VS) methods. Cores and frequency are tuned to further improve the performance or energy efficiency on those architectures. Our experimental results show that targeted low-cost systems are the lowest power consumption platforms, although the most energy efficient platform and the best suited for performance improvement is the Kepler GK110 GPU from Nvidia using CUDA. Finally, the OpenCL version of VS shows a remarkable performance penalty compared to its CUDA counterpart.

Received . . .

KEY WORDS: Energy Efficiency; Heterogenous Computing; Molecular Docking

1. INTRODUCTION

We are witnessing a steady transition from single to multi/many core processors with the Moore's law looming on the horizon. Although several improvements have continued to provide smaller semiconductor devices, physical limitations of the silicon-based architectures are pushing this movement towards parallelism [1]. At this rate, the increasing amount of transistors in the same area due to technology scaling will exacerbate power dissipation problems. In the near future, it is uncertain if all the transistors of the same chip will be available in a given cycle due to temperature constraints [2].

The adoption of heterogeneous computational elements at the moment is mandatory for energy efficiency, being the new approach to reduce the Energy per instruction (EP) consumption. These architectures use several cores with different functionality, performance, and energy efficiency, comprising latency-oriented cores for control-dominated tasks, throughput-oriented cores for data-driven tasks, and low-power cores to meet low power constraints in low/mid-performance tasks.

*Correspondence to: gines.guerrero@ditec.um.es

This way, the chip can exploit the characteristics of each task to efficiently run them from both execution time and power consumption perspectives. The run-time system is still immature to efficiently map processors and computations, and thus programmers play a fundamental role in this emergent landscape of computation to extract maximum performance. It is not an easy task, as they have to deal with different hardware components, instruction sets and programming models, while keeping power consumption on a reasonable threshold.

However, the integration of these latest breakthroughs in high performance computing with those obtained in other fields such as image processing and computational modeling enables remarkable advances in the fields of healthcare, genome research, drug discovery, etc. For instance, Virtual Screening (VS) methods can enormously help to discover new drugs [3] and energy materials [4]. The different approaches used in VS methods differ mainly in the way they model the interacting molecules, but all screen databases of chemical compounds containing up to millions of ligands [5]. Larger databases increase the chances of generating hits or leads, but the computational time needed for the calculations increases not only with the size of the database but also with the accuracy of the chosen VS method. Fast docking methods with atomic resolution require a few minutes per ligand [6], while more accurate molecular dynamics-based approaches still require hundreds or thousands of hours per ligand [7]. Therefore, the limitations of VS predictions are directly related to a lack of computational resources, a major bottleneck that prevents the application from detailed, high-accuracy models to VS.

In this paper, we use a molecular docking kernel within drug discovery field to benchmark a wide range of novel architectures manufactured by Intel, ARM, AMD/ATI and Nvidia, with an emphasis on power, performance and energy. Major findings include the following:

1. We have implemented the electrostatic interactions kernel to fully leverage Chip Multiprocessors with vectorization from different manufacturers. This implementation is based on C/OpenMP to fully utilize the cores and SSE and NEON vector instructions.
2. A data-parallel scheme on GPUs is deployed using CUDA [8] and OpenCL [9] programming models. Our design proposes a tiling technique to exploit data-locality via shared memory. The OpenCL version, which was successfully ported to other platforms without requiring any changes, obtains a poor performance compared to CUDA.
3. Different Nvidia GPUs are analyzed, all endowed with a Fermi GF100 but having different bandwidth, memory size and, more importantly, number of multiprocessors. This allows us to estimate the impact of power gating [10], a technique to reduce power consumption by shutting off the flow of current to unused blocks of the circuit. As the targeted kernel is fully scalable, the impact of power gating here is low.
4. We offer an in-depth analysis of the hardware particularities of each processor, by tuning the number of cores and clock speeds to further improve performance and energy efficiency.
5. The hardware generations based on Nvidia Kepler GPUs and ARM dual-core Cortex A15 are compared to reveal solid advantages of Kepler concerning energy efficiency and performance enhancements.
6. Our results nominate low power GPUs and embedded processors as the lowest power consumption platforms. Moreover, considering performance-oriented metrics like the Energy Delay Product (EDP or ED^2P), Nvidia high-end GPUs are better suited for computing the VS kernels.

The rest of the paper is organized as follows. Section 2 briefly introduces basic concepts. Section 3 describes the implementation details of the targeted VS kernel in different platforms. Section 4 reports our evaluation methodology before showing the main experimental results in Section 5. We summarize the related work in Section 6, and finally, Section 7 concludes the paper and shows possible directions for future work.

DOI: 10.1002/cpe

2. PRELIMINARIES

2.1. Virtual Screening

Virtual Screening (VS) has played an important role in several areas such as catalysts and energy materials [4] and drug discovery, in which experimental techniques are increasingly complemented by numerical simulation [11]. Although VS methods have been investigated for many years and several compounds could be identified that evolved into drugs, the impact of VS has not yet fulfilled all expectations. Neither the docking methods nor the scoring functions used presently are sufficiently accurate to reliably identify high-affinity ligands. To deal with a large number of potential candidates (many databases comprise hundreds of thousands of ligands), VS methods must be very fast and yet identify "the needles in the haystack".

In many VS applications, the predicted ligands turn out to have low affinity (false positives), while high affinity ligands rank low in the database (false negatives). In contrast, established simulation (not scoring) methods, such as free-energy perturbation theory, can determine relative changes in the affinity when ligands are changed slightly (group substitutions). These techniques require hundreds of CPU hours for each ligand, reaching thousands of CPU hours for each ligand when simulation strategies are used to compute absolute binding affinities [12]. In comparison to these techniques, VS methods must make significant approximations regarding the affinity calculation and the sampling of possible receptor complex conformations. These approximations would be justifiable, as long as the relative order of affinity is preserved at the high-affinity end of the database.

In most of the VS methods, the biological system is represented in terms of interacting particles. For the calculation of the interaction energies, classical potentials are commonly used, separated into bonded and non-bonded terms. In VS methods, and in many other molecular mechanics based methodologies, the most intensive computations are spent on the calculation of the non-bonded interactions kernel. For example in Molecular Dynamics, the calculation of these kernels takes up to 80% of the total execution time [13]. Therefore, these kernels can be considered bottlenecks, and it has been shown that their parallelization and optimization permits VS methods to deal with more complex systems, simulate longer time scales or screen larger chemical compound databases [14].

The relevant non-bonded potentials used in VS calculations are the Coulomb or electrostatic 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. In this paper, we focus on the electrostatic kernel (ES). This kernel is the baseline for several methodologies used in VS methods, such as Molecular Dynamics and protein-protein docking.

2.2. Programming Models

Major hardware vendors are releasing heterogeneous architectures with features having a direct impact on the application runtime performance. This performance is unleashed using programming models like Compute Unified Device Architecture (CUDA) [8] and Open Computing Language (OpenCL) [9]. Latency-oriented architectures such as Intel and AMD CPUs still relay on traditional parallel programming models such as OpenMP [15] and MPI [16], including vectorization through SSE and NEON Intrinsics, although OpenCL is also compatible for some of them. In this section, we briefly introduce the emergent programming models like CUDA and OpenCL used to calculate the electrostatic potential kernel on our massively parallel architectures.

2.3. CUDA Programming Model

Nvidia GPU platforms can be programmed using the Compute Unified Device Architecture (CUDA) programming model, which makes the GPU to operate as a highly parallel computing device. Each GPU device is a scalable processor array consisting of a set of SIMT (Single Instruction Multiple Threads) multiprocessors (SM), each of them containing several stream processors (SPs). Different memory spaces are available in each GPU on the system. The global memory (also called device or video memory) is the only space accessible by all multiprocessors. This is the largest and the slowest memory space and it is private to each GPU on the system. Moreover, each multiprocessor

DOI: 10.1002/cpe

has its own private memory space, called shared memory, because this memory is stored among all SPs in a SM. The shared memory is smaller and has lower latency than global memory.

The CUDA programming model is based on a hierarchy of abstraction layers: The **thread** is the basic execution unit that is mapped to a single SP. A **block** is a batch of threads, which can cooperate together because they are assigned to the same multiprocessor, and therefore they share all the resources included in this multiprocessor, such as register file and shared memory. A **grid** is composed of several blocks, which are equally distributed and scheduled among all multiprocessors. Finally, threads included within a block are divided into batches of 32 threads called **warps**. The warp is the scheduled unit, so the threads of the same block are scheduled in a given multiprocessor warp by warp. The programmer declares the number of blocks, the number of threads per block and their distribution to arrange parallelism given the program constraints (i.e., data and control dependencies).

2.4. OpenCL

OpenCL is a framework for developing parallel algorithms on any hardware device which supports it, either a CPU, APU or GPU. In short, OpenCL enables CUDA functionality through very similar methods, but enabling the programmer to deal with non-Nvidia hardware. Optimizations for a device from one manufacturer could not likely result in optimal performance on hardware from another manufacturer, but that is a toll to pay in favor of portability.

OpenCL enforces a threading model almost identical to the CUDA model, using a different naming scheme and adding a new identifier. At the highest level, the global workgroup groups every thread that executes the kernel. Underneath, groups of threads referred to as workgroups receive a unique identifier and each thread in every workgroup is given both a local ID to mark its location in the workgroup, and a global ID to mark its position in the global workgroup.

Unfortunately, due to this threading model, there are strict limitations on how data is shared between workgroups. In the context of GPUs, communicating information in local memory between workgroups involves copying the data into global memory, and then having the other workgroups copy that information into its own local memory. This type of forced access causes a large latency, which may result in performance degradation for the application. Other devices have these same memory types, though they may be emulated in the device's global memory, which may reside in system memory if the device has no global memory, such as with APUs.

3. KERNELS IMPLEMENTATION

In this section we describe our implementations of the virtual screening kernel that calculates electrostatic potential. Starting from the sequential code, we are going to introduce: (*i*) an optimized latency-oriented multicore implementation that enables vector operations and uses all cores in a chip through OpenMP programming model, (*ii*) an optimized CUDA implementation previously published in [17, 18], and finally, (*iii*) an OpenCL implementation designed specifically for this paper for comparison purposes to unleash non-Nvidia architectures.

3.1. Sequential Baseline

In our study, we focus on the particular case of protein-ligand docking, and particularly, in the calculation of the electrostatic potential kernel. Algorithm 1 shows the sequential baseline of this kernel. Both receptor and ligand molecules are represented by rec and lig particles, which are specified by their positions (x, y, z) and charges ($qLigand$ and $qReceptor$). The number of atoms is given by $rec.length$ and $lig.lenght$ for both receptor and ligand molecules, respectively. This algorithm is computationally intensive, with a regular memory pattern which can take advantage of data-locality. The number of floating-point operations depends on the molecules size, with some of them potentially vectorized.

DOI: 10.1002/cpe

Algorithm 1 The sequential pseudocode for the calculation of the electrostatic potential.

```
1: Sum = 0
2: for i = 0; i ≤ rec.length; i++ do
3:    for j = 0; j ≤ lig.length; j = j++ do
4:       dif = rec[i] − lig[j]
5:       aux = rsqrt(dif^2)
6:       Sum+ = qReceptor[i] ∗ qLigand[j] ∗ aux
7:    end for
8: end for
9: return Sum
```

3.2. OpenMP Implementation with Vectorization

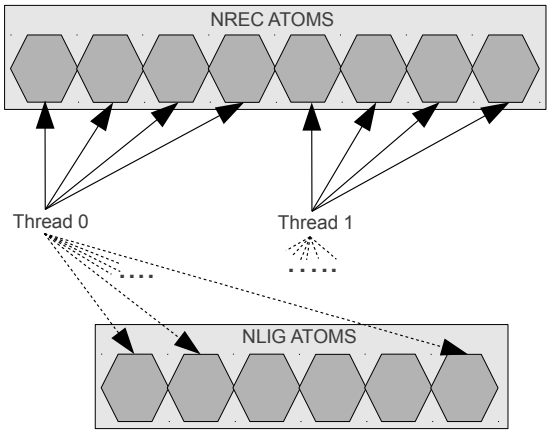

Figure 1. OpenMP design for the calculation of the electrostatic interaction kernel with 2 threads.

The OpenMP API supports multi-platform shared-memory parallel programming in C/C++. OpenMP is portable, scalable model for developing parallel applications on different multicore platforms. OpenMP is based on a fork-join model, starting with an initial thread that soon forks a team of threads. The OpenMP design for the electrostatic kernel, showed in Algorithm 1, divides the computation among different threads as showed in Figure 1. Each thread performs the computation associated with its own part of the receptor data ($nrec$ atoms). Thus, each thread computes the energy interactions between its own private $nrec$ atoms and all atoms from the ligand ($nlig$ atoms). These two set of atoms are stored within the same memory space without requiring data duplication.

An additional performance gain can be obtained by taking advantage of the vector instructions to enhance the energy calculation. Algorithm 2 shows the vectorized kernel, where each $nrec$ atom is placed in a 128-bit vector ($copy1to4$ function), and each element of $nlig$ is copied four times into another 128-bit vector ($copy4To4$ function). The energy calculation can then be vectorized on each processor to compute values in groups of four elements at a time.

3.3. CUDA Implementation

Our departure point is the CUDA implementation of electrostatic potential kernel previously introduced in [18]. Figure 2 illustrates the idea behind this design. Each atom from the receptor molecule is represented by a single thread. Then, every CUDA thread goes through all the atoms of the ligand molecule.

CUDA enables a double layer of parallelism: First, among all multiprocessors of the GPU via **concurrent CUDA blocks**, which are independent batches exploiting data-parallelism; second, among all threads within each block via the **block size**, which can cooperate through shared memory and synchronize through atomic operations. We maximize parallelism on these two layers by having:

DOI: 10.1002/cpe

Algorithm 2 Vectorization for the calculation of the electrostatic potential.

```
1: vec_Sum = copy1To4(0)
2: for i = 0; i ≤ rec.length; i++ do
3:    vec_Receptor = copy1To4(rec[i])
4:    vec_qReceptor = copy1To4(qReceptor[i])
5:    for j = 0; j ≤ lig.length/4; j = j + 4 do
6:       vec_Ligand = copy4To4(lig[j], lig[j + 1], lig[j + 2], lig[j + 3])
7:       vec_qLigand = copy4To4(qLigand[j], qLigand[j + 1], qLigand[j + 2], qLigand[j + 3]
8:       vec_dif = vec_Receptor − vec_Ligand
9:       vec_aux = rsqrt(vec_dif^2)
10:      vec_Sum+ = vec_qReceptor ∗ vec_qLigand ∗ vec_aux
11:   end for
12: end for
13: return vec_Sum[0] + vec_Sum[1] + vec_Sum[2] + vec_Sum[3]
```

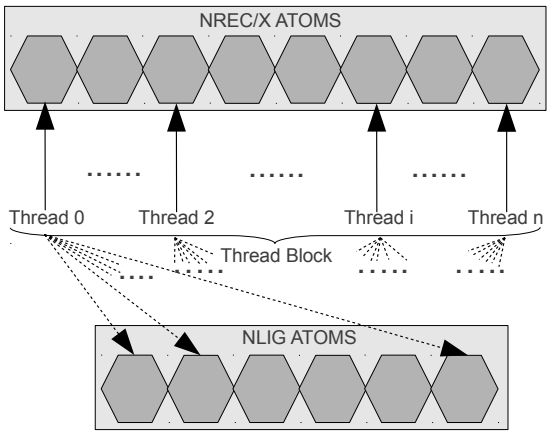

Figure 2. CUDA design for X thread blocks (for $X = 1$) with n threads layout.

1. As many thread blocks as the number of $nrec$ atoms divided by the number of threads within a block. This number is a set up parameter for our application.
2. As many threads as $nrec$ atoms. This way, each thread computes the energy calculations with the entire ligand data.

Algorithm 3 The host-side of the CUDA implementation.

```
1: CopyDataFromCPUtoGPU(rec)
2: CopyDataFromCPUtoGPU(lig)
3: numBlocks := nrec/numThreads
4: Kernel(numBlocks, numThreads)
5: CopyDataFromGPUtoCPU(result)
```

Algorithm 3 shows the host side of the CUDA implementation. Here, the receptor and ligand information is copied to the GPU global memory before electrostatic interactions are computed by the kernel. Finally, results are copied back to the host DRAM memory. The number of threads is a parameter of our implementation matching the number of launched blocks.

Kernels 1 and 2 show two different CUDA implementations of the electrostatic kernel. The former reflects the idea of accessing solely to device memory (see Figure 2). We also enable a tiling technique in our kernel 2 to take advantage of data locality similarly to other memory intensive applications [19]. Tile groups atoms of the ligand molecule, and thus threads can cooperate in order to move that information from *global* to *shared memory*.

DOI: 10.1002/cpe

CUDA kernel implementations

Kernel 1. Basic implementation

```
1: for all Blocks do
2:   for i = 0 to nlig do
3:     calculus(myAtomRec, lig[i])
4:   end for
5: end for
```

Kernel 2. Tile implementation

```
1: for all Blocks do
2:   numIt = nlig/numThreads
3:   for i = 0 to numIt do
4:     copyBlockDataToSharedMemory(lig)
5:     calculusBlock(myAtomRec, ligBlock)
6:   end for
7: end for
```

3.4. OpenCL Implementation

OpenCL foundations are based on the CUDA threading model, but with a different naming scheme and a new identifier. Thus, we use source to source translation to migrate our CUDA based-kernels for the targeted VS methods to OpenCL programs. This requires a depth knowledge of these two APIs, being more complicated than a mere instruction mapping. Moreover, OpenCL is still on an early stage of development (v. 1.2), which does not provide the rich functionality offered by CUDA on its 4.2 version and library-based programming model.

Design of OpenCL kernels is similar to that of CUDA ones, but some differences arise at implementation level, namely for device set up, context creation and data copying. This way, mapping the kernel onto the device processing elements may differ in the programming effort required to code and debug a parallel application.

Among some other limitations to prevent a smooth transition from CUDA to OpenCL, OpenCL lacks support for C++ device code and uses *cl_mem* type on the host for abstracting pointers to device memory. CUDA handles device memory in host and device code through direct pointers (e.g. *float* *). This allows applications to reserve space in device memory for structs that have pointers to device memory nested within them. As *cl_mems* are translated to pointers only when passed in as kernel arguments, there is no such way to nest device pointers, which generates a collection of warnings at compilation level [20].

4. BENCHMARKING ENVIRONMENT

In this section we introduce the hardware-software environment, the input data sets, and the way we deal with those power measurements later shown in section 5.

4.1. Hardware Systems

Our experiments have been conducted in four different configurations aimed to Nvidia, Intel, AMD/ATI and ARM based machines (see Table I for detailed specifications). The first targeted system is a high-performance platform composed by an Intel Xeon processor and a Nvidia Geforce 7300GT GPU (namely GPU 0 in Table I), which is tailored for graphics, and therefore, it is always connected in the system during the evaluation.

Moreover, three different GPUs (namely GPUs 1, 2 and 3 in Table I) are connected separately to this system through the PCI Express bus as accelerators; i.e. only one of them is connected to the motherboard during the tests at a given time. We have used these GPUs because all shared the Fermi GF100 architecture, with 3200 million transistors in $529mm^2$. However, they have different bandwidth, memory size and, more importantly, number of active SMs. The Nvidia GTX465 is endowed with 11 SMs while the Nvidia GTX480 has 15 active SMs and the Nvidia Tesla C2070 has 14 active SMs. This allows us to estimate the impact of power gating [10] on the analyzed benchmarks. Additionally, we analyze the last generation of Nvidia GPUs; i.e. Tesla K20c that

DOI: 10.1002/cpe

Table I. Hardware features on each of our implementation platforms.

Intel System	
Processor:	Intel Xeon E5620 @ 2.4 GHz
GPU 0:	Nvidia 7300GT
Memory:	16GB DDR3 @ 1333 MHz
Maximum Power Draw:	80 W
Experimental Idle Power:	138 W
GPU 1: Nvidia GTX 465	
GPU family:	GF100
Manufacturing process:	40 nm.
Core Clock:	607 MHz
Memory Size:	1024 MB
Memory Clock:	2x 1603 MHz
Memory Bus Width:	256 bits
Memory Bandwidth:	102.6 GB/sec
Stream Processors:	352
Maximum Power Draw:	200 W
Experimental Idle Power:	24 W
GPU 2: Nvidia GTX 480	
GPU family:	GF100
Manufacturing process:	40 nm.
Core Clock:	700 MHz
Memory Size:	1536 MB
Memory Clock:	2x 1848 MHz
Memory Bus Width:	384 bits
Memory Bandwidth:	177.4 GB/sec
Stream Processors:	480
Maximum Power Draw:	250 W
Experimental Idle Power:	37 W
GPU 3: Nvidia Tesla C2070	
GPU family:	GF100
Process:	40 nm.
Core Clock:	573.5 MHz
Memory Size:	6143 MB
Memory Clock:	2x 1494 MHz
Memory Bus Width:	384 bits
Memory Bandwidth:	143.4 GB/sec
Stream Processors:	448
Maximum Power Draw:	247 W
Experimental Idle Power:	107 W
GPU 4: Nvidia Tesla K20c	
GPU family:	GK110
Manufacturing process:	28 nm.
Core Clock:	705 MHz
Memory Size:	5120 MB
Memory Clock:	2x 2600 MHz
Memory Bus Width:	320 bits
Memory Bandwidth:	208 GB/sec
Stream Processors:	2496
Maximum Power Draw:	225 W
Experimental Idle Power:	27 W

AMD System	
Processors:	2x AMD Magny-Cours 6134 8 cores @ 2.3 GHz
Memory:	16 GB DDR3 @ 1333 MHz
Maximum Power Draw:	2x115 W
Experimental Idle Power:	132.1 W
GPU 5: ATI FirePro V8800	
GPU family:	RV870
Manufacturing process:	40 nm.
Memory Size:	2048 MB
Core Clock:	825 MHz
Memory Clock:	2x 1150 MHz
Memory Bus Width:	256 bits
Memory Bandwidth:	147.2 GB/sec
Stream Processors:	1600
Maximum Power Draw:	208 W
Experimental Idle Power:	24.5 W

Sandy Bridge System	
Processor:	Intel Core i5 2430M @ 2.4 GHz
Memory:	4GB DDR3 @ 1333 MHz
Max Power Draw:	35 W
Experimental Idle Power:	10.5 W
GPU 5: Nvidia GTX 540M	
GPU family:	GF108
Manufacturing process:	40 nm.
Core Clock:	672 MHz
Memory Size:	1024 MB
Memory Clock:	2x 900 MHz
Memory Bus Width:	128 bits
Memory Bandwidth:	28.8 GB/sec
Stream Processors:	96
Maximum Power Draw:	35 W
Experimental Idle Power:	14.5 W

Embedded Systems	
Exynos 4 Quad	
Processor:	Samsung Exynos 4 “Quad” (or 4412) Quad-core Cortex-A9 @ 1.6 GHz
Memory:	1GB LPDDR2 @ 400 MHz
Maximum Power Draw:	10 W
Experimental Idle Power:	3.2 W
Exynos 5 Dual	
Processor:	Samsung Exynos 5 “Dual” (or 5250) Dual-core Cortex-A15 @ 1.7 GHz
Memory:	2GB LPDDR3 @ 800 MHz
Max Power Draw:	19 W
Experimental Idle Power:	6.5 W

unveils Titan, the world’s fastest supercomputer in mid 2013 [21], together with two of the top five most energy-efficient supercomputers in the world [22].

The second system is also a high-performance platform based on a dual AMD Magny-Cours processor plus the ATI FirePro V8800 graphics card (namely GPU 5 in Table I). The last three platforms are low-cost and energy-efficient solutions, one of them based on an Intel Sandy Bridge with an Nvidia GT540M, and the other two platforms based on an Exynos 4 Quad and an Exynos 5 Dual. In fact, the Mont-Blanc project [23] includes Exynos 5 Dual as the main processor to design a low-power and high-performance supercomputer.

DOI: 10.1002/cpe

Table II. Software resources used for each hardware platform in our experimental study. "n.a." means not available.

Language	Target hardware	Software tools	Vector instructions used
C/OpenMP	Intel Xeon	gcc compiler 4.7.2	SSE
	AMD Magny-Cours	gcc compiler 4.7.2	SSE
	Intel Sandy Bridge	gcc compiler 4.7.2	SSE
	Exynos 4/5	gcc compiler 4.7.2	NEON
OpenCL	Intel Xeon	Intel SDK 2012	n.a.
	AMD Magny-Cours	AMD APP SDK 2.6	
	Intel Sandy Bridge	Intel SDK 2012	
	Nvidia GPUs	CUDA toolkit 5.0	
	ATI FirePro	AMD APP SDK 2.6	
CUDA	Intel Xeon	n.a.	n.a.
	AMD Magny-Cours	n.a.	
	Intel Sandy Bridge	n.a.	
	Nvidia GPUs	CUDA toolkit 5.0	
	ATI FirePro	n.a.	

4.2. Software Environment

Table II shows the main software tools used in our implementations. They are structured depending on targeted hardware. The CUDA and OpenCL programming models are used to program manycore architectures. More precisely, CUDA toolkit 5.0 leverages Nvidia architectures while different OpenCL SDK versions from different vendors are used to program both Nvidia and ATI-AMD architectures as it is the strength of the standard. We also use OpenCL on the multicore side for those platforms that support the standard (the low-power Exynos 4 architecture does not, and the Linux drivers for Exynos 5 are still under development). Multicores are also targeted through gcc compiler 4.7.2 version and vectorization. The vectorization on Intel and AMD based platforms is enabled by SSE extensions while the ARM-based processor uses NEON technology [24] which are included in ARM cortex-A series processor.

4.3. Input Data Sets

The input data set for benchmarking our virtual screening kernels is a common scenario routinely used in typical virtual screening calculations. An arbitrary system with number of atoms of the receptor ($nrec$) equal to 65536 and number of atoms of the ligand ($nlig$) equal to 65536 atoms is defined. It represents the electrostatic interaction between two average size biomolecules, situation that arises in many protein-protein docking simulations used by VS methods.

4.4. Power Measurement

Power dissipation numbers are obtaining using the *Watts up? .Net* power meter [25]. This device is connected between the power source and the power supply of the system, and provides power dissipation information every second. Power information is logged by a different machine on the same room. Room temperature is controlled and set to 26°C during the measurements to minimize temperature impact on static power. We decided not to isolate power consumed by the GPU from the rest of the system (CPU, Motherboard, Memory, etc), as the hardware used in our experiments is required to execute the GPU kernels, so it is fair to account this hardware when optimizing for energy efficiency (Energy and Energy Delay Square Product -ED^{2}P- results). The real-time measurement of individual GPU components using a software approach is new and only supported by the Nvidia GPU K20. This is done by using NVML (Nvidia Management Library) [26], which reports the GPU power usage at real-time.

We decided to tweak the kernels, making 2048 launches so that the base execution lasted around 2.5 minutes for the fastest platform, giving ample time for the GPU processor to warm up.

DOI: 10.1002/cpe

5. EXPERIMENTAL RESULTS

We now analyze the virtual screening kernel that calculates electrostatic potential in terms of power, time, energy, energy delay square product on different multi-and-many core architectures, and using different programming models as previously explained. The input data set described in Section 4.3 was used in all cases.

Platforms are grouped into two clusters: *m*any-cores and *m*ulticores. The OpenCL and CUDA kernels are used for benchmarking many-core architectures. The multicore architectures are evaluated using the OpenCL kernel and the vectorized OpenMP kernel. All kernels are set up with the best experimentally demonstrated configuration depending on the programming model and underlying architecture.

In the multicore architectures, we vary the linux power-mode selector of AMD Magny-Cours (MC) and Intel Sandy Bridge (SB), affecting the frequency of these microprocessors. Available modes are *powersave* (LP), *ondemand* (OD) and *performance*. LP mode forces all processor cores to work at minimal frequency, OD adapts frequency depending on the processor load, and *performance* mode forces the processor to run at maximum frequency all the time. The evaluation of the performance mode is omitted because the processor load is at maximum and thus OD mode behaves similarly than *performance* mode.

5.1. Performance Evaluation

Table III. Execution time in seconds for the electrostatic kernel when measured on different platforms and programing languages. "MC LP" stands for Magny-Cours on power save mode, "MC OD" for Magny-Cours on demand mode, "SB LP" for Sandy Bridge on power save mode and finally "SB OD" stands for Sandy Bridge on demand mode.

(a) Many Cores

	CUDA (sec.)	OpenCL (sec.)
GTX 465	280	369
GTX 480	188	249
Tesla C2070	238	309
Tesla K20	132	162
GTX 540M	1194	1558
FirePro V8800	-	469

(b) Multi Cores

	C/OpenMP (sec.)	OpenCL (sec.)
MC LP	4893	7325
MC OD	1712	2751
SB LP	25379	51522
SB OD	8758	14974
Exynos 4	25792	-
Exynos 5	32640	-

Table III shows the execution times for the ES kernel run on all architectures and programming models. Missing columns report an unfeasible combination of language and architecture. The left hand side of the Tables belong to Nvidia architectures and CUDA source codes. All GF100-based GPUs (i.e. Tesla C2070, GTX 465 and GTX 480) obtain similar performance, with the last one behaving slightly better due to its higher clock frequency and number of SMs within the GTX 480 card. The low power GPU GTX 540M obtains the highest execution times as expected, being even higher than the ATI FirePro GPU, which is half-way between Nvidia desktop and mobile solutions. Finally, the fastest targeted manycore platform is the newest Nvidia Tesla K20, obtaining up to 1.42x speed-up factor compared to a GeForce GTX 480.

Performance of the analyzed multicore platforms is shown on the right-hand side of Table III. The worst performance is obtained by the ARM low-power platforms Exynos 5 and 4, followed by Intel Sandy Bridge (SB), and finally the AMD Magny-Cours (MC) processor. On one hand, Exynos family are embedded processors designed for ultra-low power devices rather than performance. Exynos 5 is newer than Exynos 4, but the latter is a quad-core which can better exploit the massively parallelism of the electrostatic kernel. On the other hand, Magny-Cours defeats intel Sandy Bridge by a wide margin. However, this is not a fair comparison as the targeted Sandy Bridge is a mobile version while Magny-Cours is a server one. These couple of processors reach better performance with the `ondemand` configuration when cores are running at higher clock rates.

DOI: 10.1002/cpe

Using OpenCL on multicore architectures is generally penalized by adding some extra overhead, usually due to the immature compiler versions developed for these architectures. However, OpenCL only creates and schedules a single but parallel task, which fully exploits all computational resources and obtains better performance compared to the OpenMP program without vectorization.

5.2. *Energy Evaluation*

Table IV. Power consumption in Watts for the electrostatic kernel when measured on different combinations of hardware and software support.

(a) Many Cores.

	CUDA (Watts)	OpenCL (Watts)
GTX 465	340	340
GTX 480	390	389
Tesla C2070	363	361
Tesla K20	252	251
GTX 540M	59	56
FirePro V8800	-	330

(b) Multi Cores.

	C/OpenMP (Watts)	OpenCL (Watts)
MC LP	234	233
MC OD	316	303
SB LP	11	14
SB OD	32	39
Exynos 4	8	-
Exynos 5	16	-

Table V. Total energy consumption in $Joules/1000$ (mJ) for the electrostatic kernel when measured on different selections of hardware platforms and programming paradigms.

(a) Many Cores.

	CUDA (mJ)	OpenCL (mJ)
GTX 465	95	125
GTX 480	73	97
Tesla C2070	86	111
Tesla K20	33	40
GTX 540M	70	87
FirePro V8800	-	154

(b) Multi Cores.

	C/OpenMP (mJ)	OpenCL (mJ)
MC LP	1147	1709
MC OD	542	834
SB LP	299	752
SB OD	282	583
Exynos 4	261	-
Exynos 5	522	-

Tables IV and V show the power consumption and the overall energy consumption during the execution of the electrostatic kernel. Remember that the power measurements are "at the wall" (they include the power consumption of the whole system). The total energy consumed is correlated to the total execution time of the application, and thus, CUDA-based systems are the best suited for low ratios of energy consumption as they are getting the best performance for the targeted virtual screening kernels. The total energy consumed by a multicore-based system is generally higher than using accelerators (GPUs). Greater performance over manycores is wide enough to amortize the inclusion of an extra source of power dissipation.

The left-hand side of Table IV shows the power consumption when manycore architectures are included in the system. The best power-efficient manycore architecture is the GeForce GTX 540M, even though it is the slowest one (see Table III). The GTX 540M card is designed for low power, optimizing energy in motherboard, hard disk, etc, which makes a difference overall. The Kepler architecture proposed by Nvidia as a step forward in performance and power consumption fulfills expectations as the Tesla K20 GPU reaches the best ratios in these two respects.

Among multicore architectures (right-hand side of the Table IV), the embedded architecture Exynos 4 is the most power efficient high-end processor, but at the expense of a higher execution time. The reported numbers of Exynos 5 related to power and energy can be somehow confusing as they are higher than Exynos 4. This is because Exynos 5 is plugged into a motherboard where the audio, video and MicroSC storage hardware introduce an extra overhead.

Table VI shows the Energy Delay Product (EDP) of the analyzed kernels. This metric gives priority to performance over energy. When we use it to compare multicore and manycore architectures the gap between them increases, particularly for highly parallel applications such as

DOI: 10.1002/cpe

Table VI. EDP in $Joules/1000 * seconds$ for the electrostatic kernel when measured on .

(a) Many Cores

	CUDA (EDP)	OpenCL (EDP)
GTX 465	26691	46394
GTX 480	13808	24164
Tesla C2070	20580	34549
Tesla K20	4458	6647
GTX 540M	84385	136836
FirePro V8800	-	72720

(b) Multi Cores

	C/OpenMP (EDP)	OpenCL (EDP)
MC LP	5613472	12522958
MC OD	928955	2296587
SB LP	7600685	38756075
SB OD	2477878	8744719
Exynos 4	5321818	-
Exynos 5	17045913	-

the electrostatic kernel. This metric also shows that the energy efficiency in embedded and mobile hardware is not enough to overcome the double weight of performance. In addition, Table VII shows the Energy Delay Square Product (triple weight on performance), another metric widely used in HPC environments, which points to high energy-efficient hardware as the worst option, overall for highly parallel applications.

Table VII. ED^{2}P in $Joules/10^9 * seconds$ for different pairs of Platform/Programing-Language for the Electrostatic Kernel.

(a) Many Cores

	CUDA (ED^{2}P)	OpenCL (ED^{2}P)
GTX 465	7,47	17,11
GTX 480	2,59	6,01
Tesla C2070	4,89	10,67
Tesla K20	0,59	1,08
GTX 540M	100,75	213,19
FirePro V8800	-	34,12

(b) Multi Cores

	C/OpenMP (ED^{2}P)	OpenCL (ED^{2}P)
MC LP	27470	91741
MC OD	1590	6319
SB LP	192902	1996794
SB OD	21702	130944
Exynos 4	137260	-
Exynos 5	556378	-

5.3. Hardware Resource Optimization

This section compares advantages and drawbacks of increasing frequency versus number of multiprocessors (SMs) when running our VS kernels on Nvidia high-end GPUs.

We use MSI Afterburner tool [27] to modify the working frequency of the SMs, as Nvidia decided to remove the official support for frequency settings at a driver level. MSI Afterburner is only available for Microsoft Windows, so we perform our analysis under Windows 7, using Nvidia's driver v285.62 for this test. We have simulated the increment and decrement of SMs for the GTX400 chipset by comparing the GTX465 and GTX480 as mentioned in Section 4.

Table VIII shows performance against the fastest GPU (GTX480 at default frequency). We can see how, for the electrostatic kernel, increasing frequency benefits performance all the time, but there is an equivalence between adding 4 additional SMs and increasing frequency by 100 MHz (480 @ 400 MHz vs. 465 @ 500 MHz). This benefit from SMs is barely perceived when comparing the 480 @ 500 MHz vs. C2070 @ 513 MHz, leading us to believe that we are close to the maximum number of cores we can use, probably because of bandwidth limitations. This enables us to increase efficiency by trading SMs by frequency.

Finally, Table VIII shows energy against the fastest GPU (GTX480 at default frequency). As we saw in the previous section, the mobile GPU (540M) is the most energy efficient choice for the electrostatic kernel, being able to surpass the GTX480 and C2070 when overclocked. These chips not only reduce running costs but also cooling, cluster size and production costs, being an excellent choice for high performance computing.

DOI: 10.1002/cpe

Table VIII. Electrostatic kernel analyzed in terms of time (in seconds) and power (in Joules/1000) for different pairs of of GPU/Frequency.

	Frequency (MHz)	Time	Power Consumption (Watts)	Energy
465	400	413	157	65
	450	373	166	61
	500	338	179	60
	550	308	189	58
	600	280	202	56
480	400	315	189	59
	450	285	199	56
	500	258	209	53
	550	235	219	51
	600	219	227	49
	650	201	241	48
	700	188	252	47
C2070	513	264	217	57
	573	238	225	53
540M	400	1990	31	61
	500	1592	34	55
	600	1342	37	50
	670	1194	34	40

6. RELATED WORK

Multicore architectures exhibit some peculiarities when running parallel workloads, especially in terms of power and performance. Threads must synchronize periodically (e.g., for communication purposes), and any delay suffered in one of the threads may end up with a slow down of the whole application. It is unclear wether future GPUs will implement per-core DVFS, as current generations like Kepler from Nvidia have limitations on manually setting frequencies for the set of cores due to the complexity of the clock trees [28].

Energy efficiency in GPUs is high as long as the application keep using available resources, but there are just few hardware mechanisms to tailor resources to application needs. Hong *et al.* [29] propose a power and performance model which is used to select the number of optimal cores based on the available memory bandwidth. In [30], Sheaffer *et al.* studied a thermal management for GPUs, whereas Fu *et al.* [31] performed a complete experimental survey on GPU data.

More recently, Wang [32] *et al.* propose an instruction-level energy estimation methodology for GPUs. Gebhart [33] *et al.* introduce a couple of techniques for reducing energy on massively-threaded processors acting over the register file and thread scheduler, which are two of the key functional units concerning storage and occupancy on a typical GPU nowadays.

In terms of benchmarking energy-efficiency in heterogeneous systems, Mistry [34] *et al.* present Valar, a benchmark suite to study the dynamic behavior of heterogeneous systems through OpenCL applications. Finally, Johnson [35] investigate the energy efficiency of accelerated HPC servers using throughput oriented techniques. These works are more related to ours, which we consider a superset given the range of architectures and software paradigms analyzed.

7. CONCLUSIONS

Applications with a real impact on the society, such as those for discovering new drugs, can take advantage of the great advances in the field of high performance computing to overcome emerging challenges. When physical limitations of silicon-based architectures are threatening the evolution of processors, heterogeneous computing involving GPUs, CMPs or low-power processors come to the rescue when no answer looms on the horizon.

We have analyzed this synergy between application and hardware, benchmarking flagship processors from major vendors like Intel, ARM, AMD/ATI and Nvidia. We care about power,

Prepared using cpeauth.cls
DOI: 10.1002/cpe

performance and energy using a molecular docking kernel (electrostatic interactions) coming from a drug discovery process. Our work deploys three different implementations for this kernel:

1. A C/OpenMP version with vectorization to leverage the computational power of CMPs and low-power platforms.
2. A data-parallel scheme on GPUs using CUDA to target Nvidia platforms, where we propose a tiling technique to exploit data locality using shared memory.
3. An OpenCL version oriented to more general platforms and manufacturers.

After the performance evaluation, the best positioned architectures to run this kernel are clearly GPUs. Low power GPUs such as 540M, and embedded processors based on ARM architectures, like Exynos 4, are the most energy efficient platforms in our experiments. Surprisingly, even more than traditional GPUs and CPUs despite their increase in execution time. However, whenever performance is a priority over power, the old *cliché* of "the fastest the best" becomes valid for EDP and ED^2P metrics, making the Nvidia Tesla K20 the best suited counting all metrics. A step forward in energy efficiency and/or performance for these architectures can be attained by handling cores and their underlying frequency, where we have found greater opportunities to reduce power in CPUs than in low power GPUs or embedded devices.

The OpenCL code gave us further opportunities to compare multiple platforms using exactly the same code. OpenCL promises future JIT optimizations, but suffers performance penalties when compared to more mature compilers like nvcc (Nvidia - CUDA). This difference widens when low power devices and less resources are involved, reaching almost an order of magnitude in the case of the 540M low-power GPU.

Virtual screening methods on GPUs are still at a relatively early stage of evolution, and we have just tested a simple bioinformatics kernel. Other kernels within this emerging and fruitful area of research remain to be explored to confirm our analysis here.

On the hardware side, we have seen great benefits and potential within the last generation of GPUs in terms of performance and power consumption. The ratios compared to CPUs are expected to get even better whenever the problem size keeps growing and GPU architectures evolve, particularly considering the novel interest of governments in green computing as far as domestic markets are concerned.

Finally, we envision our approach to be rewarded from increasingly heterogeneous platforms endowed with specialized cores and eventually integrated within the same silicon die. They could monitor performance and power consumption more closely, allowing energy efficiency plays a decisive role, particularly when massive parallelism arises in HPC.

ACKNOWLEDGMENTS

This work has been jointly supported by the Fundación Séneca (Agencia Regional de Ciencia y Tecnología de la Región de Murcia) under grant 15290/PI/2010, by the Spanish MINECO and the European Commission FEDER funds under grants TIN2009-14475-C04 and TIN2012-31345, and by the Catholic University of Murcia (UCAM) under grant PMAFI/26/12. We also thank Nvidia for hardware donation under Professor Partnership 2008-2010, CUDA Teaching Center 2011-2013, CUDA Research Center 2012-2013 and CUDA Fellow 2012-2013 Awards.

References

1. Asanovic K, Bodik R, Catanzaro BC, Gebis JJ, Husbands P, Keutzer K, Patterson DA, Plishker WL, Shalf J, Williams SW, *et al.*. The landscape of parallel computing research: A view from berkeley. *Technical Report UCB/EECS-2006-183*, EECS Department, University of California, Berkeley Dec 2006.
2. Esmaeilzadeh H, Blem E, St Amant R, Sankaralingam K, Burger D. Dark silicon and the end of multicore scaling. *Proceedings of the 38th annual international symposium on Computer architecture*, ISCA '11, ACM: New York, NY, USA, 2011; 365–376, doi:10.1145/2000064.2000108.
3. Jorgensen WL. The Many Roles of Computation in Drug Discovery. *Science* Mar 2004; **303**(5665):1813–1818, doi:10.1126/science.1096361.

4. Franco AA. Multiscale modelling and numerical simulation of rechargeable lithium ion batteries: concepts, methods and challenges. *RSC Advances* 2013; doi:10.1039/C3RA23502E.
5. Irwin JJ, Shoichet BK. ZINC A Free Database of Commercially Available Compounds for Virtual Screening. *J. Chem. Inf. Model.* Dec 2004; **45**(1):177–182, doi:10.1021/ci049714+.
6. Zhou Z, Felts AK, Friesner RA, Levy RM. 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* 2007; **47**(4):1599–1608.
7. Wang J, Deng Y, Roux B. Absolute Binding Free Energy Calculations Using Molecular Dynamics Simulations with Restraining Potentials. *Biophys J* Oct 2006; **91**(8):2798–2814, doi:10.1529/biophysj.106.084301.
8. NVIDIA. *NVIDIA CUDA C Programming Guide 5.0*. 2012.
9. The open standard for parallel programming of heterogeneous systems. `http://www.khronos.org/opencl/` [23 February 2013].
10. Hu Z, Buyuktosunoglu A, Srinivasan V, Zyuban V, Jacobson H, Bose P. Microarchitectural techniques for power gating of execution units. *ISLPED '04: Proceedings of the 2004 international symposium on Low power electronics and design*, ACM: New York, NY, USA, 2004; 32–37, doi:10.1145/1013235.1013249.
11. Schneider G. Virtual screening and fast automated docking methods. *Drug Discovery Today* Jan 2002; **7**:64–70, doi:10.1016/s1359-6446(02)00004-1.
12. Wang J, Deng Y, Roux B. Absolute Binding Free Energy Calculations Using Molecular Dynamics Simulations with Restraining Potentials. *Biophys J* Oct 2006; **91**(8):2798–2814, doi:10.1529/biophysj.106.084301.
13. Kuntz SK, Murphy RC, Niemier MT, Izaguirre JA, Kogge PM. Petaflop Computing for Protein Folding. *In Proceedings of the Tenth SIAM Conference on Parallel Processing for Scientific Computing*, 2001; 12–14.
14. Pérez-Sánchez H, Wenzel W. Optimization Methods for Virtual Screening on Novel Computational Architectures. *Current computer-aided drug design* Sep 2010; :44–52.
15. OpenMP Architecture Review Board: The OpenMP Specification. `http://www.openmp.org/` [23 February 2013].
16. The Message Passing Interface (MPI) standard. `http://www.mcs.anl.gov/research/projects/mpi/` [23 February 2013].
17. Guerrero GD, Sánchez HEP, Wenzel W, Cecilia JM, García JM. Effective parallelization of non-bonded interactions kernel for virtual screening on gpus. *PACBB*, 2011; 63–69.
18. Guerrero GD, Sánchez HEP, Cecilia JM, García JM. Parallelization of virtual screening in drug discovery on massively parallel architectures. *PDP*, 2012; 588–595.
19. Garland M, Le Grand S, Nickolls J, Anderson J, Hardwick J, Morton S, Phillips E, Zhang Y, Volkov V. Parallel Computing Experiences with CUDA. *Micro, IEEE* Jul 2008; **28**(4):13–27, doi:10.1109/mm.2008.57.
20. Martinez G, Gardner MK, chun Feng W. Cu2cl: A cuda-to-opencl translator for multi- and many-core architectures. *International Conference on Parallel and Distributed Systems*, 2011; 300–307.
21. Top 500 supercomputer sites. `http://www.top500.org/` [23 February 2013].
22. Top Green500 List. `http://www.green500.org/` [23 February 2013].
23. European Aproach Towards Energy Efficienct High Performance. `http://http://www.montblanc-project.eu/` [23 February 2013].
24. The ARM NEON technology. `http://www.arm.com/products/processors/technologies/neon.php` [23 February 2013].
25. Wattup.net power meter. `https://www.wattsupmeters.com` [23 February 2013].
26. Nvidia Corporation. NVML API Reference. `http://developer.download.Nvidia.com/assets/cuda/files/CUDADownloads/NVML/nvml.pdf` [23 February 2013].
27. MSI Afterburner overclocking tool. `http://event.msi.com/vga/afterburner/index.htm` [23 February 2013].
28. Clock manipulation on Fermi and newer GPUs. `http://www.phoronix.com/scan.php?page=news_item&px=OTgxNQ` [23 February 2013].
29. Hong S, Kim H. An integrated gpu power and performance model. *SIGARCH Comput. Archit. News* Jun 2010; **38**(3):280–289, doi:10.1145/1816038.1815998.
30. Sheaffer JW, Luebke D, Skadron K. A flexible simulation framework for graphics architectures. *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, HWWS '04, ACM: New York, NY, USA, 2004; 85–94, doi:10.1145/1058129.1058142.
31. Fu R, Zhai A, chung Yew P, chung Hsu W. Reducing queuing stalls caused by data prefetching. *In INTERACT-11*, 2007.
32. Wang Y, Ranganathan N. An instruction-level energy estimation and optimization methodology for gpu. *Proceedings of the 2011 IEEE 11th International Conference on Computer and Information Technology*, CIT '11, IEEE Computer Society: Washington, DC, USA, 2011; 621–628, doi:10.1109/CIT.2011.69.
33. Gebhart M, Johnson DR, Tarjan D, Keckler SW, Dally WJ, Lindholm E, Skadron K. Energy-efficient mechanisms for managing thread context in throughput processors. *SIGARCH Comput. Archit. News* Jun 2011; **39**(3):235–246, doi:10.1145/2024723.2000093.
34. Mistry P, Ukidave Y, Schaa D, Kaeli D. Valar: a benchmark suite to study the dynamic behavior of heterogeneous systems. *Proceedings of the 6th Workshop on General Purpose Processor Using Graphics Processing Units*, GPGPU-6, ACM: New York, NY, USA, 2013; 54–65, doi:10.1145/2458523.2458529.
35. Johnsson L. Efficiency, energy efficiency and programming of accelerated hpc servers: Highlights of prace studies. *GPU Solutions to Multi-scale Problems in Science and Engineering*, Yuen DA, Wang L, Chi X, Johnsson L, Ge W, Shi Y (eds.). Lecture Notes in Earth System Sciences, Springer Berlin Heidelberg, 2013; 33–78, doi:10.1007/978-3-642-16405-7_3.