REVIEW ARTICLE

Structure-based Drug Discovery Accelerated by Many-core Devices

Wei Feinstein[1] and Michal Brylinski[2,3,*]

[1]High Performance Computing, Louisiana State University, Baton Rouge, LA 70803, USA; [2]Department of Biological Sciences, Louisiana State University, Baton Rouge, LA 70803, USA; [3]Center for Computation & Technology, Louisiana State University, Baton Rouge, LA 70803, USA

ARTICLE HISTORY

Received: May 26, 2015
Revised: October 07, 2015
Accepted: November 20, 2015

DOI: 10.2174/1389450117666160112112854

Michal Brylinski

Abstract: Computer-aided design is one of the critical components of modern drug discovery. Drug development is routinely streamlined using computational approaches to improve hit identification and lead selection, enhance bioavailability, and reduce toxicity. A mounting body of genomic knowledge accumulated during the last decade or so presents great opportunities for pharmaceutical research. However, new challenges also arose because processing this large volume of data demands unprecedented computing resources. On the other hand, the state-of-the-art heterogeneous systems deliver petaflops of peak performance to accelerate scientific discovery. In this communication, we review modern parallel accelerator architectures, mainly focusing on Intel Xeon Phi many-core devices. Xeon Phi is a relatively new platform that features tens of computing cores with hundreds of threads offering massively parallel capabilities for a broad range of application. We also discuss common parallel programming frameworks targeted to this accelerator, including OpenMP, OpenCL, MPI and HPX. Recent advances in code development for many-core devices are described to demonstrate the advantages of heterogeneous implementations over the traditional, serial computing. Finally, we highlight selected algorithms, *e*FindSite, a ligand binding site predictor, a force field for bio-molecular simulations, and BUDE, a structure-based virtual screening engine, to demonstrate how modern drug discovery is accelerated by heterogeneous systems equipped with parallel computing devices.

Keywords: Xeon phi, parallel accelerators, drug discovery, *e*FindSite, BUDE, programming frameworks.

INTRODUCTION

Computer-aided Drug Discovery

Due to extremely high costs associated with drug development, integrating computational approaches with bench-top experiments holds a significant promise to speed up the discovery of novel biopharmaceuticals. The importance of computer modeling techniques has dramatically increased over the past years on account of advances in algorithm development as well as the constantly increasing processing power of modern computer architectures. Contemporary high-performance computing (HPC) platforms facilitate large-scale applications of existing modeling tools to address various problems in modern drug discovery. However, new hardware also creates an urgent need for the development of codes to more efficiently utilize HPC resources available to the scientific community. In this communication, we review selected modeling techniques that are commonly used at the outset of drug development, including protein structure modeling, ligand binding site prediction, and receptor-based virtual screening. We also discuss how Intel Xeon Phi, cutting-edge computer architecture, is likely to accelerate the process of pharmaceutical discovery as exemplified by several studies recently reported in the literature.

Protein Structure Modeling

Drug targets are typically selected by disease linkage studies that give evidence of an association between biological targets and certain disease states [1, 2]. The atomic-level structure of a drug target is then used to either select from existing chemical libraries or design from scratch small molecules that are capable of modulating its function. Ideally, a three-dimensional structure of the target macromolecule solved by X-ray crystallography or NMR spectroscopy is available for rational drug discovery. However, in the absence of the experimental structure, protein homology modeling can be used to construct a theoretical model of the drug target using one of many currently available software packages, *e.g.* Modeller [3], SWISS-MODEL [4], Rosetta [5], and I-TASSER [6].

There are many examples of a successful application of homology modeling in structure-based drug discovery. For instance, potential candidates for new antibiotics have been

*Address correspondence to this author at the Department of Biological Sciences, Louisiana State University, Baton Rouge, LA 70803, USA; Tel/Fax: +1 225 578 4003; E-mail: michal@brylinski.org

recently identified by virtual screening of a large chemical library against the homology model of VEB-1 β-lactamase, an important microbial target, for which no clinically effective inhibitors are currently available [7]. Another study reports a successful homology modeling of α1A adrenoreceptor followed by a hierarchical virtual screening procedure guided by various 2D filters and 3D pharmacophore models [8]. Out of 80 diverse compounds identified by the structure-based modeling, 37 compounds exhibit binding affinity, K_i, values better than 10 μM with the most active compound binding to the target receptor at K_i of 1.4 nM. Furthermore, a new protocol was developed specifically for the template-based modeling of G-protein coupled receptors (GPCRs) and validated using CXCR7 chemokine receptor, an appealing drug target for cancer chemotherapy [9]. Encouragingly, structure-based virtual screening against the homology model of CXCR7 resulted in 21 ligands with novel scaffolds, whose experimentally confirmed half maximal inhibitory concentration, IC_{50}, values are between 1.29 and 11.4 μM. These studies demonstrate that template-based protein structure modeling is an important component of modern computer-aided drug discovery. It can be used to generate reliable three-dimensional models for a large number of potential drug targets whose structures will not be solved experimentally in the near future.

Identification of Ligand Binding Sites

Drug binding site prediction is another computational method commonly used in structure-based drug discovery. Here, the goal is to identify those surface regions of pharmacologically relevant proteins that can be targeted by small molecules in order to modulate their molecular functions. Although, in many cases, the target sites are known from experiments, drugs can interact with alternative surface locations that are distinct from the primary binding sites. For example, allosteric modulators of cell-surface receptors offer greater selectivity over those compounds interacting with the binding site for endogenous agonists [10]. Consequently, the past couple of decades have seen a rapid development of techniques for the identification of new target sites for pharmacotherapy (see [11] for a recent comprehensive review). Structure-based methods, such as LIGSITE [12], Fpocket [13] and SURFNET [14], predict binding sites by searching for pockets and cavities in a protein structure, whereas energy-based approaches, such as Q-SiteFinder [15], FTSite [16] and PocketFinder [17], locate binding residues by analyzing interaction energies with small molecular probes. Despite a high accuracy of pocket detection for experimentally solved macromolecular structures, using protein models still renders significant challenges due to inevitable structural imperfections in their atomic coordinates.

To address this problem, a number of evolution/structure-based approaches for ligand binding site prediction have been developed. For instance, COFACTOR [18] employs global and local structure alignments of structural analogs in order to predict various aspects of protein function. First, the structures of template proteins selected from the Protein Data Bank (PDB) [19] are globally aligned onto the target using TM-align [20]. Next, the best local geometric and sequence match between the target and template structures is calculated taking into account the evolutionary conservation of functional sites. Binding site prediction is followed by the annotation of target proteins with Enzyme Commission numbers (for enzymes) and molecular function according to the Gene Ontology classification [21]. In fact, COFACTOR has been used to support the structural and functional characterization of a novel molluskan ortholog of TNF receptor-associated factor from *Haliotis discus* [22], glucose dehydrogenase from *Leclercia* sp. [23], trehalose-6-phosphate phosphatase from *Mycobacterium tuberculosis* [24], and a cysteine-rich protein enriched in salivary glands [25]. Other evolution/structure-based methods for ligand-binding site prediction include 3DLigandSite [26], FINDSITE [27, 28], and recently developed *e*FindSite [29, 30], which is discussed later in the text.

Molecular Docking and Virtual Screening

Virtual screening is perhaps the single most recognizable computational technique in drug discovery. Its primary application is to select a handful of potential lead candidates from a vast organic chemical space that is estimated to contain at least 10^{60} molecules of interest to drug developers [31]. Early similarity-based approaches to virtual screening using molecular fingerprints [32], geometric hashing [33], and topo-geometrical features [34] are increasingly being replaced by molecular docking techniques, which generally offer higher prospects to discover compounds with novel chemical scaffolds [35]. Molecular docking predicts the atomic details of the preferred binding pose of a drug candidate with respect to its protein target [36]. Subsequently, the strength of association between these molecules, also referred to as binding affinity, is estimated from molecular interactions. A number of algorithms for ligand docking have been developed, *e.g.* AutoDock [37], DOCK [38], Q-Dock [39], GOLD [40], *e*SimDock [41], and ICM [42]. The main differences among individual methods are the implementation of scoring functions and techniques for conformational space sampling. There are many reports documenting successful applications of molecular docking in biopharmaceutical discovery; see [43-45] for examples.

In contrast to simple ligand-based methods, docking techniques are computationally challenging. The first difficulty stems from a large conformational space of interactions between small organic compounds and their macromolecular targets, *viz.* there is a large number of possible arrangements to form a complex structure [46]. Second, the conformational space grows exponentially with the size of the interacting molecules because of the increasing number of degrees of freedom [47]. Therefore, addressing the docking problem requires efficient sampling algorithms that cover the relevant conformational space, as well as sensitive scoring functions to effectively discriminate between native and non-native binding poses. Finally, virtual screening applications routinely involve performing molecular docking calculations for a large number of drug candidates; for instance, the ZINC database of commercially available molecules in ready-to-dock formats contains millions of purchasable compounds [48]. On that account, structure-based virtual screening heavily depends on the availability of computing resources.

Strikingly, virtual screening by molecular docking constitutes highly parallel computational procedures. At the coarse-grain level, a substantial number of drug molecules are subjected to independent docking simulations that can be processed in parallel. At the fine-grain level, most scoring functions for the modeling of molecular complexes build upon non-bonded pairwise interactions between protein and ligand atoms in the form of statistical potentials [49, 50] and classical physics-based energy terms [38, 42], which can be efficiently parallelized. Moreover, many sampling techniques offer a possibility for a medium-grained parallelization as well. For instance, parallel implementations of the Ant Colony Optimization [51], Replica Exchange Molecular Dynamics [52], and Genetic Algorithms [53] have been developed. Consequently, the emerging massively parallel technologies have a great potential to accelerate large-scale virtual screening applications using molecular docking.

High-performance Computing Using Accelerators

High-performance parallel computing is an effective solution for solving large and complex problems. It has a capability to deliver reliable results within a reasonable time frame for memory, computing and data-intensive applications. Unlike traditional serial execution using single core CPUs, multi-core computer architectures allow programs to run in parallel using multiple processor cores on a single chip to reduce the computation time. Moreover, these units can be assembled into shared and distributed memory computing systems. Collectively, thousands of cores from different nodes being deployed together further improve the computational throughput. For example, a dual 18-core CPUs featuring Hyper Threading can execute 72 parallel tasks concurrently, thus using 10 of such computing units delivers as many as 720 threads for parallel jobs. Potentially, the time-to-completion can be significantly shortened as the scale of parallelization increases. Compared to multi-core architectures, many-core hardware accelerators deliver even greater parallel processing capabilities. These devices can considerably enhance the overall performance of applications featuring a high ratio of computation to data access. Currently, the most popular parallel accelerators include NVIDIA GPUs (Graphic Process Unit), Intel Xeon Phi coprocessor and AMD APU (Accelerated Processing Units).

GPUs with hundreds of small cores were specifically designed by NVIDIA to concurrently handle multiple calculations in video games. Later on, the GPU technology has evolved to a general purpose GPU-accelerated platform, where GPUs are utilized along with CPUs to speed up various applications [54]. These advances in the parallel computing architecture demand adequate programming models for a full utilization of the hardware potential. For instance, CUDA (Compute Unified Device Architecture) is a programing model that gives a full low-level control of the parallel processing on GPUs [55]. OpenACC was designed in 2010 as a compiler directive-based standard aiming to ease coding efforts [56]. Conceptually, the computation-intensive sections of a source code can be offloaded to the GPU, where parallel tasks are grouped into grids, blocks and warps. The workload is eventually divided among individual threads that make up the basic computing units on the GPU for parallel calculations to enhance the overall performance. As such, GPU-accelerated computing has made steady headway to achieve a shorter time-to-completion through the efficient code parallelization. Much scientific research has already benefited from the GPU technology, which was successfully applied to protein sequence [57] and structure alignments [58], Brownian dynamics [59], spin model simulations [60] as well as the modeling of *in vivo* diffusion [61].

AMD is another hardware vendor offering massively parallel devices that has been pursuing the heterogeneous system architecture (HSA), where CPU and GPU cores use a common programing language and share the same workload and memory space. For example, GPU-based Radeon features up to 12 compute cores, which are equivalent to 8 GPUs attached to 4 CPUs [62]. Although AMD GPUs account for a relatively small market share compared to NVIDIA GPUs, these devices support a wide range of projects. For example, the Berkeley Open Infrastructure for Network Computing platform running on AMD GPUs has been used to accelerate numerous codes, such as the Binary Radio Pulsa Search application, a part of the astronomy project Einstein@Home [63], as well as MilkyWay@Home that creates a highly accurate 3D model of the Milky Way galaxy [64]. Carrizo APU is the newest parallel hardware platform designed by AMD to avoid data flow bottlenecks by placing CPU and GPU cores on a single chip [65]. This high-density design is expected to bring significant reductions in the die area as well as the power consumption.

Intel Xeon Phi is a relatively new member of the accelerator family featuring Many Integrated Cores (MIC). The coprocessor offers tens of general-purpose x86 cores and hundreds of computing threads for parallel tasks. In addition, the Xeon Phi coprocessor is uniquely equipped with wide, 512-bit vector processors allowing powerful parallel SIMD (Single Instruction, Multiple Data) execution [66]. Unlike the GPU technology, Intel Xeon Phi coprocessor is designed to work as a commodity CPU, so that programming tools traditionally used for multi-core CPUs can also be seamlessly used on a coprocessor. To make code porting relatively easy, compiler pragma-based approach was introduced as one the important components of the parallel programming for the MIC architecture. In contrast to laborious code re-writing for the GPU platform, parallel MIC programming requires considerably fewer modifications, although a thorough code optimization is mandatory to achieve the high performance. In this review, we focus primarily on the Intel Xeon Phi accelerator as a parallel platform for computer-aided drug discovery, mainly because of its high portability and relatively short code porting cycles.

INTEL XEON PHI ARCHITECTURE

Coprocessor Design

Intel Xeon Phi is a massively parallel architecture featuring many integrated cores to effectively accelerate compute-intensive applications. Systems equipped with Xeon Phi cards have two key components, processor(s) and coprocessor(s) connected *via* the PCIe bus [66]. Each coprocessor comprises up to 61 in-order processor cores, where each core has 4 hardware threads switchable in a round-robin fashion to hide latency. Moreover, each core has a wide 512-bit SIMD vector that allows for simultaneous operations on 8 double precision or 16 integer instructions. When only a single precision is needed, the number of concurrently executed vector instructions is doubled, therefore, a 61-core coprocessor is capable of executing 3,840 (60×4×16) single-precision instructions at a time. In fact, the Xeon Phi coprocessor provides more parallelism than the Xeon processor, which features 16 cores with 2 threads per core and narrower 256-bit SIMD vectors. In terms of the memory organization, the coprocessor provides 8 GB global memory as well as 32 kB Level 1 (L1) and 256 kB Level 2 (L2) data cache for each coprocessor core; individual cores are interconnected and remotely accessible *via* a fast bi-directional ring [67]. The memory architecture of Xeon Phi allows the L1 and L2 data cache of each coprocessor core to be shared, however, memory sharing is not permitted between the processor and the coprocessor.

Thread Affinity

Intel Xeon Phi supports multi-threading at 4 hardware threads per core, providing up to 244 threads. The physical distribution of these threads, or thread affinity, can have a considerable impact on the parallel performance [68]; three types of thread affinity are available, compact, balanced and scatter. Specifically, the compact thread affinity minimizes the number of cores to be utilized, whereas for the scatter and balanced settings, the maximum number of cores are allocated in a round-robin or a uniform fashion, respectively. The thread affinity is controlled by environment variables, for example, using export MIC_KMP_AFFINITY=compact instructs the coprocessor to pack threads densely next to each other on a single core before moving on to the next one.

Programming Models

Intel Xeon Phi runs a light version of Linux operating system called BusyBox. Parallel tasks on the coprocessor can be processed using either the native mode, where the entire application is executed on the coprocessor, or the offload mode, which offloads only parts of the code to the accelerator. For example, adding the -mmic compiler flag generates a native binary that can be executed directly on the coprocessor. However, the offload mode is often more suitable for those applications having serial portions of the code. Here, functions and variables that need to be accessed by the coprocessor are marked by directive pragmas, *e.g.* #pragma offload target (mic) defines sections of the code to be offloaded to the coprocessor for parallel execution. Note that the full utilization of the accelerator often requires tuning the data transfer protocols in order to establish an efficient communication between the processor and the coprocessor.

Parallel Capabilities

The accelerator architecture delivers two layers of parallelism at the coarse- and fine-grained level to achieve a high parallel performance (Fig. **1**). Both the multi-core processor and the many-core coprocessor provide a convenient coarse-grained parallel capability through individual computing cores, as it is shown in Fig. (**1A**). For instance, the Intel Xeon Sandy Bridge microarchitecture features 2-8 computing cores with 2 threads per core, whereas high-end Intel Xeon Phi accelerators equipped with 61 cores and 4 threads per core provide up to 244 threads. Practically, 240 threads are available on the accelerator since the 61st core is usually reserved for operating system and I/O operations. This level of parallelism can be exploited using pragma-based programming frameworks that effectively allocate individual threads to handle concurrent tasks. In contrast, the fine-grained level of parallelism is accomplished by executing data-level SIMD instructions (Fig. **1B**). The Intel Xeon Phi SE10P coprocessor has a 512-bit wide SIMD vector unit per core capable of processing multiple data at a time, *e.g.* 8 double-precision floating-point operations can be executed simultaneously. Moreover, 4 threads per core permit interleaving of up to 4 such SIMD operations to hide latency. In comparison, the Intel Xeon E5-2680 processor also offers data level parallelism yet with narrower 256-bit wide SIMD vectors and only 2 threads per core. Overall, the fine-grained parallelism can significantly boost the performance, therefore, an effective utilization of SIMD resources through a proper code vectorization is critical to fully benefit from the Xeon Phi platform [69].

Large-scale Production Systems

Similar to other accelerator architectures, Intel Xeon Phi is designed to provide massively parallel capabilities to meet the demands of large-scale scientific computing. In the past several years, a number of supercomputers equipped with Intel Xeon Phi cards have been put into production around the world. For example, Tianhe-2, deployed in 2013 at the National Supercomputer Center in Guangzhou, is the fastest Xeon Phi-based supercomputer in the world with an impres-

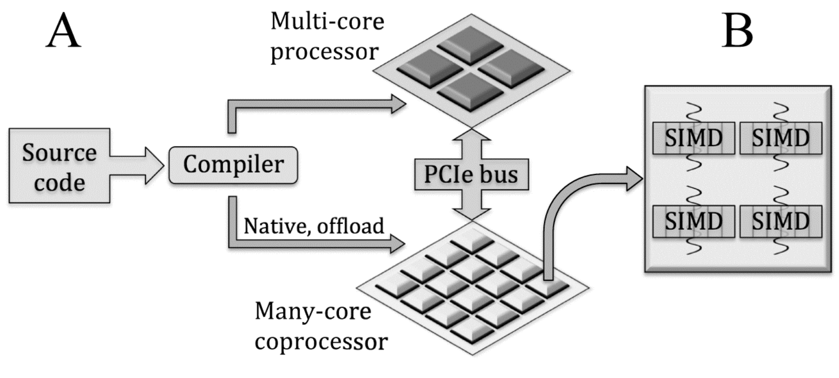

Fig. (1). Levels of parallelism on Intel Xeon and Xeon Phi. (**A**) The coarse-grained parallelism provided by multiple processor cores and many coprocessor cores. A compiler is instructed to generate binaries for a given architecture. (**B**) The fine-grained parallelism provided by wide SIMD execution slots within each core.

sive number of 16,000 computer nodes. Each node features two Intel Ivy Bridge Xeon processors and three Intel Xeon Phi coprocessors capable of providing the aggregate peak performance of 33.86 PFLOPS [70]. Beacon system operated by the National Institute for Computational Sciences at the University of Tennessee, Knoxville is an energy efficient cluster composed of 48 compute nodes with two 8-core Intel Xeon E5-2670 processors and four Intel Xeon Phi 5110P coprocessors delivering up to 210 TFLOPS of computational performance [71]. The Stampede at the Texas Advanced Computer Center is a supercomputer available to the scientific community through the Extreme Science and Engineering Discovery Environment (XSEDE). Stampede is built with 6,400 Dell DCS Zeus nodes, the majority of which are equipped with two Intel Xeon E5-2680 Sandy Bridge processors and either one or two Intel Xeon Phi SE10P coprocessors. The aggregate peak performance of the processors is over 2 PFLOPS, while the coprocessors deliver an additional aggregate peak performance of over 7 PFLOPS [72]. Finally, SuperMIC at the Louisiana State University is a recently deployed system containing a total of 382 nodes, each with two 10-core Intel Ivy Bridge-EP processors [73]. Among those, 360 nodes have two Intel Xeon Phi 7120P coprocessors. SuperMIC delivers the theoretical peak performance of over 925 TFLOPS and, similar to Stampede, it is also available to the scientific community through XSEDE.

APPLICATION PROGRAMMING INTERFACES

Although Intel Xeon and Xeon Phi units have different designs, to a certain degree, they share a similar x86-based architecture. This is an important feature of the coprocessor, because programming techniques developed for the processor can also be used on the accelerator. In this section, we review selected parallel programming frameworks relevant to the code development for the coprocessor, including MPI, OpenMP, OpenCL, and HPX.

MPI

The Message Passing Interface (MPI) is a widely used parallel programming model for distributed memory architectures. Each processor has an access to its own memory and different processors are connected through high-speed communication links, such as InfiniBand. MPI provides a set of language-independent communication protocols for parallel computing, featuring point-to-point message passing as well as collective operations *via* user-specified processors [74]. Specifically, processes are typically created by discrete processors/computing nodes executing different sections of the code. Each process has its own local variables and the memory space; the parallelism is achieved by establishing communications between processes by sending and receiving messages. Expanded from the original support for distributed memory architectures, MPI is now adapted to include shared memory symmetric multiprocessing (SMPs) as a hybrid programming model. Thus far, the MPI standard has been implemented by several groups, such as the open source MPICH [75] developed at the Argonne National Laboratory and Open MPI supported by a consortium of academic, research, and industry partners [76]. MPI is widely used to parallelize serial applications in drug design; for instance, MPI-based parallelization of AudoDock4 was used in virtual screening of large databases of compounds [77].

MPI application programming interface (API) provides well-defined object-like constructors, destructors and user-defined data types allowing for heterogeneous communications as well as an efficient data description and exchange. Because of its vendor-independent portability and flexibility, MPI is regarded as the “industry standard” for the message passing programming on high-performance computing platforms. MPI can be easily compiled for the Intel Xeon Phi architecture with special flags, *e.g.* compiling a simple hello_world.c source code using mpicc –mmic –o hello_world.MIC generates a native executable for the coprocessor. Once the executable is copied to the coprocessor and the environment variable I_MPI_MIC is set to 1, an MPI launcher can be used to start the MPI-based application on the accelerator, *e.g.* mpiexec –f mpi_hosts hello_world.MIC. Nonetheless, the communication bottleneck caused by the heavy data traffic between the processors and coprocessors can limit the performance of MPI-based applications. In addition, debugging may be challenging and, in contrast to other programming models, such as OpenMP, substantial coding efforts are often required in order to port a serial code to the accelerator.

OpenMP

OpenMP is a widely adopted standard developed in the 1990’s for parallel programming on multi-threaded shared-memory systems, where a single address space is accessed with unique memory locations by different processes/cores. It features compiler-specific pragmas to identify sections of the code that need to be parallelized and instruct how data is transferred and distributed across different computing units. OpenMP has been implemented as C/C++/Fortran compiler extensions allowing parallelism to be added using pragmas/directives (#pragma omp in C/C++ and $OMP in Fortran) to a serial codes without the need for significant code conversions [78]. OpenMP pragmas are designed to facilitate and coordinate various parallelization components, *e.g.* spawning threads, dividing computation among threads, and synchronizing work among threads. In addition to compiler directives, a runtime library including callable functions as well as environment variables can be easily used to control the code execution at either the runtime or the compilation time. Specifically, a typical format of a directive pragma is composed of a directive name followed by clauses. For instance, #pragma omp parallel for instructs the compiler to create a set of threads to handle the parallel execution of a block of code within a loop. OpenMP is particularly effective in parallelizing for-loops, where each thread is assigned to execute a single iteration of the loop. Using environment variables, such as OMP_NUM_THREADS, the maximum number of threads to conduct the parallel computation can be set at the runtime. In addition, a similar effect can be achieved during the compilation using the library function omp_set_num_threads().

Pragma-based OpenMP makes the parallelization of serial codes relatively easy and allows for a gradual parallelization by progressively adding pragmas to the code and testing its parallel performance. Another critical feature is that pragmas are ignored if the code is compiled without OpenMP-specific flags; the parallelized code will be executed as the original serial version. Due to minimal code conversions and the compiler capacity, the maintenance of both serial and parallel versions is greatly simplified. Consequently, the open source community and major vendors, including IBM, Intel, Oracle and the Portland Group, endorse and actively support OpenMP making OpenMP-based applications platform-independent [79]. Since OpenMP offers a portable programming model for shared memory architectures, it is well suited for Intel Xeon Phi as well. Not surprisingly, a growing body of research use the coprocessors across a variety of scientific fields [68, 80-86]. Unfortunately, the shared memory requirement of OpenMP limits its application to single nodes. Nonetheless, combining OpenMP with MPI expands the parallelization scale beyond a single node with MPI typically managing inter-node parallelization and OpenMP used as the intra-node programming method [87-89].

OpenCL

On account of the expanding family of parallel accelerators, cross-platform coding has become the central issue in the development of portable software. To address this problem, Open Computing Language (OpenCL) was developed as a cross-platform framework for parallel programming on heterogeneous accelerators. Codes implemented in OpenCL can be executed across different hardware platforms, therefore, it offers a unique portability. Many vendors, such as Intel, AMD, NVIDIA, Altera Corp., and IBM, support OpenCL for their hardware [90], which stimulates the development of OpenCL-based applications [91]. OpenCL defines a set of APIs in C-like language to control a host processor and a variety of parallel devices and accelerators. A typical parallel application comprises a C/C++ code for the host and a collection of kernels and special functions written in OpenCL for the accelerators. The parallelism is achieved at different levels, including SIMT (Single Instruction Multiple Threads), work-items, which are the smallest execution units, and work-groups in the order of increasing degree of coarse-grained parallelization level. In addition, OpenCL can be used in conjunction with other parallel frameworks; for example, the FEASTFLOW code was implemented using both OpenCL and OpenMP [69]. Combined with OpenMP and MPI, OpenCL has also been applied to conduct direct numerical simulations of turbulent flows on AMD, NVIDIA and Intel accelerators [92]. In contrast to other parallel frameworks, researchers who do not already have an extensive programming experience may find OpenCL quite difficult to learn due to its complex syntax and unique data structures and functions.

HPX

The parallel implementations of scientific applications for multi- and many-core devices greatly improve their overall performance. Nevertheless, workload starvation, latencies and overheads create unprecedented bottlenecks in parallel computing, hindering the scalability at peta and exaflop levels. In order to address these problems, the Ste||ar group at Louisiana State University developed High Performance ParalleX (HPX), which is an open source runtime system based on ParalleX [93]. HPX effectively leverages asynchrony to support large-scale multi-core computations by improving the communication between inter- and intra-node processes [94]. In addition to the standard C++ local asynchronous functions, HPX also provides remote asynchronous mechanisms through Actions, Futures and Dataflow constructs. When scaling goes beyond tens of thousands of cores, many applications create significant latencies complicating the communication between processes, therefore, new methods for indexing the address space become essential. To deal with this challenge, HPX offers an effective mechanism called Active Global Address Space to manage remote resources allowing for the dynamic suspension of remote threads and re-assigning different resources to active tasks when necessary [95]. HPX was recently demonstrated to outperform MPI in 3D N-body simulations with local interactions [96]. Furthermore, unlike many other parallel frameworks, HPX provides C++ APIs, which are convenient to those programmers, who already have some experience developing parallel C++ codes for multi-core and multi-threaded heterogeneous architectures. Nevertheless, programming in HPX usually requires a relatively steep learning curve; the Ste||ar group has been working diligently to address this issue in order to ease porting efforts.

PERFORMANCE BENCHMARKS FOR INTEL XEON PHI

The availability of Intel Xeon Phi accelerators in many contemporary high-performance computing systems draws attention of a broad research community to this new architecture. Consequently, numerous studies have been recently published describing early experience of researchers porting scientific codes to Xeon Phi and reporting preliminary performance benchmarks [69, 80-86, 88, 89, 97, 98]. In Table **1**, we list several evaluations selected from the current literature sorted by the performance of accelerator threads with respect to that of the host processor. Note that results described in individual papers often are irreconcilable because of different clock speeds and hardware architecture used in benchmarks, hyper-threading settings, programming models, and code optimization and tuning techniques. Nonetheless, these reports are published within the past couple of years and cover numerous areas of modern research in physics, math, engineering, finance and medicine, exemplifying a continuously growing interest in heterogeneous computing using Xeon Phi.

An impressive speedup was achieved in optimizing a Lattice Quantum Chromodynamics (LQCD) code for the coprocessor with the OpenMP parallelization [88]. By exploiting cache-blocking techniques, an inter-core communication and the hardware support for irregular memory accesses, the Dslash kernel for the matrix vector-vector multiplication in LQCD was demonstrated to sustain 280 GFLOPS on the coprocessor. This corresponds to nearly 80% of the achievable performance and leads to impressive speedups over Xeon CPU of 1.9× and 2.2-2.4× using 5110P and 7110P cards, respectively. Moreover, accelerating an iterative algorithm for solving large sparse linear equations, PQMRCGSTAB, yields one of the highest improvements reported for Xeon Phi [83]. Thorough optimizations including data prefetching to hide data latency, reusing vector

Table 1. Accelerating scientific codes using Intel Xeon Phi.

Code (Domain)	Reference System (Intel Xeon)			Testing System (Intel Xeon Phi)			Speedup[c]	Ref.
	Model[a]	Cores[b]	Threads[b]	Model[a]	Cores[b]	Threads[b]		
Lattice Quantum Chromodynamics (Physics) [d]	E5-2670 @2.60	8 (16)	16 (32)	7110P @1.1	61 (60)	244 (60)	2.4× (128%)	[88]
Lattice Quantum Chromodynamics (Physics) [e]	E5-2670 @2.60	8 (16)	16 (32)	7110P @1.1	61 (60)	244 (60)	2.2× (117%)	[88]
Lattice Quantum Chromodynamics (Physics) [d]	E5-2670 @2.60	8 (16)	16 (32)	5110P @1.053	60 (60)	240 (60)	1.9× (101%)	[88]
Lattice Quantum Chromodynamics (Physics) [e]	E5-2670 @2.60	8 (16)	16 (32)	5110P @1.053	60 (60)	240 (60)	1.9× (101%)	[88]
PQMRCGSTAB (Math) [f]	E5-2670 @2.60	8 (8)	16 (8)	5110P @1.053	60 (60)	240 (60)	5.6× (75%)	[83]
FEASTFLOW (Engineering) [g]	E5-2658 @2.10	8 (16)	16 (16)	5120D @1.053	60 (60)	240 (128)	4.7× (59%)	[69]
3D MPDATA (Geophysics) [h]	E5-2697 @2.70	12 (24)	24 (48)	7120P @1.238	61 (61)	244 (244)	2.0× (39%)	[84]
3D MPDATA (Geophysics) [h]	E5-2697 @2.70	12 (24)	24 (48)	3120A @1.1	57 (57)	228 (228)	1.7× (36%)	[84]
Sparse-matrix matrix multiplication (Math) [i]	E5-2670 @2.60	8 (16)	16 (32)	SE10P @1.1	61 (60)	244 (240)	2.2× (29%)	[81]
Sparse-matrix vector multiplication (Math) [i]	E5-2670 @2.60	8 (16)	16 (32)	SE10P @1.1	61 (60)	244 (240)	2.1× (28%)	[81]
Microscopy Image Analysis (Medicine) [j]	E5-2680 @2.70	8 (16)	16 (16)	SE10P @1.1	61 (60)	244 (120)	2.1× (28%)	[82]
Sparse-matrix matrix multiplication (Math) [i]	X5680 @3.33	6 (12)	12 (12)	SE10P @1.1	61 (60)	244 (240)	4.5× (23%)	[81]
Sparse-matrix vector multiplication (Math) [i]	X5680 @3.33	6 (12)	12 (12)	SE10P @1.1	61 (60)	244 (240)	4.2× (21%)	[81]
NINA (Medicine)	E5-2620 @2.00	8 (16)	16 (16)	5110P @1.053	60 (60)	240 (177)	2.2× (20%)	[85]
Leukocyte Tracking (Medicine) [k]	E5-2620 @2.00	6 (12)	12 (12)	5110P @1.053	60 (40)	240 (40)	0.5× (20%)	[86]
miniMD (Atomistic Simulations) [l]	E5-2660 @2.20	8 (16)	16 (32)	5110P @1.053	60 (60)	240 (240)	1.4× (19%)	[98]
3D MPDATA (Geophysics) [h]	E5-2643 @3.30	4 (8)	8 (16)	7120P @1.238	61 (61)	244 (244)	2.9× (19%)	[84]
3D MPDATA (Geophysics) [h]	E5-2643 @3.30	4 (8)	8 (16)	3120A @1.1	57 (57)	228 (228)	2.5× (18%)	[84]
Monte Carlo LIBOR Swaption Portfolio Pricer (Finance) [m]	E5-2670 @2.60	8 (8)	16 (16)	5110P @1.053	60 (60)	240 (240)	2.3× (15%)	[97]
FEASTFLOW (Engineering) [n]	E5-2658 @2.10	8 (16)	16 (16)	5120D @1.053	60 (60)	240 (128)	1.2× (15%)	[69]
Microscopy Image Analysis (Medicine) [o]	E5-2680 @2.70	8 (16)	16 (16)	SE10P @1.1	61 (60)	244 (120)	1.0× (13%)	[82]
Microscopy Image Analysis (Medicine) [p]	E5-2680 @2.70	8 (16)	16 (16)	SE10P @1.1	61 (60)	244 (180)	1.3× (12%)	[82]

(Table 1) contd….

Code (Domain)	Reference System (Intel Xeon)			Testing System (Intel Xeon Phi)			Speedup[c]	Ref.
	Model[a]	Cores[b]	Threads[b]	Model[a]	Cores[b]	Threads[b]		
NAS Parallel Benchmarks (Computer Science) [q]	E5-2620 @2.00	8 (16)	16 (32)	5110P @1.053	60 (60)	240 (240)	0.6× (8%)	[85]
Monte Carlo Pricing of American Options (Finance) [m]	E5-2670 @2.60	8 (8)	16 (16)	5110P @1.053	60 (60)	240 (240)	0.9× (6%)	[97]
iMOOSE (Engineering)	E5-2620 @2.00	8 (16)	16 (16)	5110P @1.053	60 (60)	240 (240)	0.8× (5%)	[85]
FIRE (Image Recognition)	E5-2620 @2.00	8 (16)	16 (32)	5110P @1.053	60 (60)	240 (234)	0.4× (5%)	[85]
Conjugate Gradient solver (Math) [r]	X7550 @2.00	8 (128)[s]	16 (128)[s]	SE10P @1.09	61 (61)	244 (244)	0.6× (3%)	[80]
NestedCP Tasking (Engineering)	E5-2620 @2.00	8 (16)	16 (32)	5110P @1.053	60 (60)	240 (240)	0.2× (3%)	[85]
CP2K (Atomistic Simulations) [t]	E5-2678W @3.10	8 (16)	16 (16)	5110P @1.053	60 (60)	240 (240)	0.2× (1%)	[89]

The performance of individual codes on the coprocessor (testing system) is compared to that on the host processor (reference system): [a] the (co)processor model and @the base frequency in GHz, [b] the number of physical cores and computing threads per processor (numbers in parentheses were used in benchmarking simulations to calculate speedups), and [c] speedups as reported in the original publication (numbers in parentheses show the percentage of the performance of a single processor thread achieved by a single co-processor thread). Simulation details for those studies that report a series of benchmarking calculations with different parameters: [d] for the Wilson Dslash kernel, compressed gauge fields, and the V=32×40×24×96 case; [e] for the Conjugate Gradient solver, compressed gauge fields, and the V=32×40×24×96 case; [f] for a 3200×3200 matrix; [g] for vector operation (axpy, vectorized); [h] for 500 time steps and the grid of size 1022×512×63; [i] against the ldoor matrix; [j] for Gradient Stats with regular data access; [k] without code modifications; [l] for 2048K atoms at a cut-off of 2.5Å and with Advanced Vector Extensions enabled on the processor; [m] for 512K paths; [n] for sparse matrix-vector multiplication against the Hamrle3 matrix (spmv, manual vectorization); [o] for Connected Component Labeling with atomic functions; [p] for FillHoles with irregular data access; [q] for an embarrassingly parallel (EP) kernel; [r] 1000 iterations; [s] symmetric multiprocessing using the Bull Coherence Switch technology; [t] for the average time using the POPT version on the processor and the PSMP version on the co-processor.

registers, and the SIMD-friendly reduction deliver a speedup close to a factor of 6 compared to an 8-core processor solving the same linear equation problems.

Code parallelization and vectorization significantly impact the coprocessor performance as demonstrated for the software package FEASTFLOW for the modeling of technical flows, fluid-structure interactions, chemical reactions, and the multiphase flow behavior [69]. A parallel, vectorized code running on Xeon Phi gives a speedup of 41.9× (11.4×) compared to a serial un-vectorized coprocessor (processor) version, whereas the speedup over the best parallel code executed on CPU is 4.7×. A similar approach exploiting the task parallelism to utilize hundreds of logical cores on the coprocessor, and the data parallelism to efficiently use 512-bit vector processing units, significantly increased the performance of the multi-dimensional positive definite advection transport algorithm (MPDATA) [84]. This algorithm used in the code for numerical weather prediction executed on the top-notch 7120P model performs double precision stencil computations 2 and 2.9 times faster than dual-socket machines equipped with 12-core (at 2.70 GHz) and 4-core (at 3.30 GHz) processors, respectively.

Xeon Phi was also evaluated for its application in computational finance [97]. The accelerator-ported implementation of the LIBOR Swaption Portfolio Pricer, an embarrassingly parallel Monte-Carlo algorithm, deployed on a 5110P card outperforms a multi-core dual-processor machine 2.3 times. However, another financial application for the pricing of American put options using Monte-Carlo simulations with cross-path dependencies runs faster on the commodity multi-core CPU than on the coprocessor, indicating that speedups are heavily application-dependent [97]. Similar conclusions arose from comparative benchmarks of several algorithms including the Flexible Image Retrieval Engine that identifies sets of similar images to a query image in a database [85]. Here, despite a good scalability on Xeon Phi systems between 50 and 113, a processor core significantly outperforms a coprocessor core by a factor of 8-12.

Application-dependent improvements were also reported for Molecular Dynamics (MD) simulations that are one more example of highly CPU-intensive codes. CP2K is a state-of-the-art program for atomistic simulations of solid state, liquid, molecular, and biological systems. Notwithstanding a 50% increase in its performance on Xeon Phi, running the same MD calculations on a 16-core CPU node still yields better timings [89]. On the other hand, a single 5110P card is 1.4× faster than a dual-socket, eight-core Intel Xeon server in performing MD simulations using miniMD, a simplified version of the popular LAMMPS package [98]. Compared to the original serial code, the optimized parallel version executed simultaneously on the host processor and the accelerator brings about a 10-fold increase in computational speed. These real world applications once again demonstrate that, adding Xeon Phi yields considerable performance improvements for highly parallel workloads.

Several interesting examples of the application of Intel Xeon Phi in biology and medicine have also been reported. In microscopy image analysis, low-dimensional spatial datasets captured by high-resolution microscopy scanners for whole slide tissue specimens are subjected to an advanced image processing. Among many common operations in this domain, object segmentation and feature computation are the most CPU expensive algorithms, thus there is a dire need to improve the performance of these codes. Using a 61-core Xeon Phi and the parallel OpenMP implementation yields satisfactory speedups for algorithms with regular data access; for instance, the Gradient Stats code runs 2.1 times faster on the coprocessor than on a 16-core CPU-based system [82]. However, only moderate improvements were reported for those algorithms with irregular data access patterns and relying on atomic instructions.

Leukocyte Tracking is another medical imaging application, where white blood cells are tracked in *in vivo* video microscopy of blood vessels to help investigate the inflammation process. It was demonstrated that the execution of Leukocyte Tracking on Xeon Phi using 40 threads is twice as slow as on a traditional CPU-based system with 12 threads; a single coprocessor thread is about 5 times slower than a processor thread. However, the small number of 36 leukocytes used in this study does not provide sufficient parallelism to fully utilize the hardware resources of the coprocessor. Further optimization of the code using vectorization and the first- and second-level multi-threading significantly improved the performance, indicating that the manual vectorization and a massive parallelism are required in order to take full advantage of the computing capabilities of Xeon Phi.

APPLICATION OF INTEL XEON PHI TO DRUG DISCOVERY

Although a wide variety of applications using Intel Xeon Phi have been developed in physics, mathematics and statics, the repository of codes for drug design in pharmaceutical research is still relatively limited. Here we review three applications that have been recently ported to the accelerator, *e*FindSite, a structural bioinformatics tool for the prediction of ligand-protein binding sites in proteins, a classical force field used in bio-molecular simulations, and a virtual screening algorithm, Bristol University Docking Engine.

Ligand Binding Site Prediction

A drug binding to the specific site on a target protein triggers a series of cellular reactions leading to the desired therapeutic effect. In drug design, the knowledge of ligand binding sites is essential for conducting virtual screening experiments to identify potential lead compounds. In this regard, we recently developed *e*FindSite, an algorithm for evolution/structure-based identification of ligand binding sites in proteins. *e*FindSite was designed to maximize the accuracy of binding pocket detection at the improved tolerance to the structural deformation in protein models. Such a high tolerance to structure imperfections, especially at ligand binding sites, is particularly important for proteome-scale applications using computer-generated protein models. Since *e*FindSite employs information extracted from ligand-bound templates to detect binding sites, template-to-target structure alignments are one of its key components. Briefly, using a sensitive meta-threading technique [99, 100], weakly homologous templates bound to ligands are identified in the PDB [19] and structurally aligned to the target. Next the template-bound ligands are clustered into putative binding sites, which are then ranked using machine learning. Typically, *e*FindSite takes about 30 minutes to detect binding sites for a protein [30] with 88% of the computing time consumed by structure alignments calculations [68]. Therefore, we decided to parallelize structure alignments in order to speed up the entire prediction process.

The workflow for the parallel version of *e*FindSite is shown in Fig. (**2**). Here, the target protein is a bacterial peptide deformylase responsible for the removal of the N-terminal formyl group from newly synthesized proteins (PDB-ID: 1lru, chain A) [101]. Compounds that bind to this enzyme and block the protein synthesis could be used as antibiotics to inhibit bacterial growth. Binding site prediction using *e*FindSite comprises three distinct stages, the pre-processing of ligand-bound templates identified by *e*Thread [99, 100] (Fig. **2A**), template-to-target structure alignments (Fig. **2B**), and the post-processing calculations including pocket clustering and ranking (Fig. **2C**). Fig. (**2C**) also shows that the top-ranked binding site (a blue ball) was accurately predicted, because it overlaps with a naturally occurring antibiotic actinonin (red sticks) bound to the target structure (green ribbons) in the experimental complex structure. As indicated in Fig. (**2B**), structure alignment calculations are parallelized using OpenMP for the processor and the coprocessor.

Specifically, each template-to-target structure alignment is either mapped to a hardware thread on the processor or offloaded to the coprocessor. OpenMP pragmas with the dynamic scheduling are used to parallelize the for-loop iteration over the template library. Parts of the *e*FindSite code that handle pre and post-alignment calculations are written in C++, however, structure alignments are implemented in Fortran 77 with many thread-unsafe common blocks. Therefore, common blocks are marked using OpenMP pragmas as private in order to avoid memory conflicts during the parallel execution of structure alignments. For example, !$omp threadprivate (/block_name/) are used to mark all common blocks in Fortran 77 subroutines. In addition, the execution of portions of the code on the coprocessor requires data transfer. Since the data is copied only once and shared by all structure alignment threads, there is virtually no overhead caused by moving data back and forth. Finally, subroutines and variables offloaded to the coprocessor are marked with, respectively, !dir$ attributes offload:mic::subroutine_name and !dir$ attributes offload:mic:variable_name.

In order to fully utilize the parallel power of nodes equipped with Intel Xeon Phi cards, a dynamic workload balancing mechanism is implemented to launch parallel tasks to both processor and coprocessor concurrently. Up to 4 parallel tasks, each with 4 threads, on the processor and up to 10 parallel tasks, each with 24 threads, on the coprocessor are executed in parallel. This way, two 8-core Xeon processors and one 61-core Xeon Phi accelerator remain fully utilized. Encouragingly, the OpenMP-based version of *e*FindSite yields a 17.6× speedup over the serial version. Specifically,

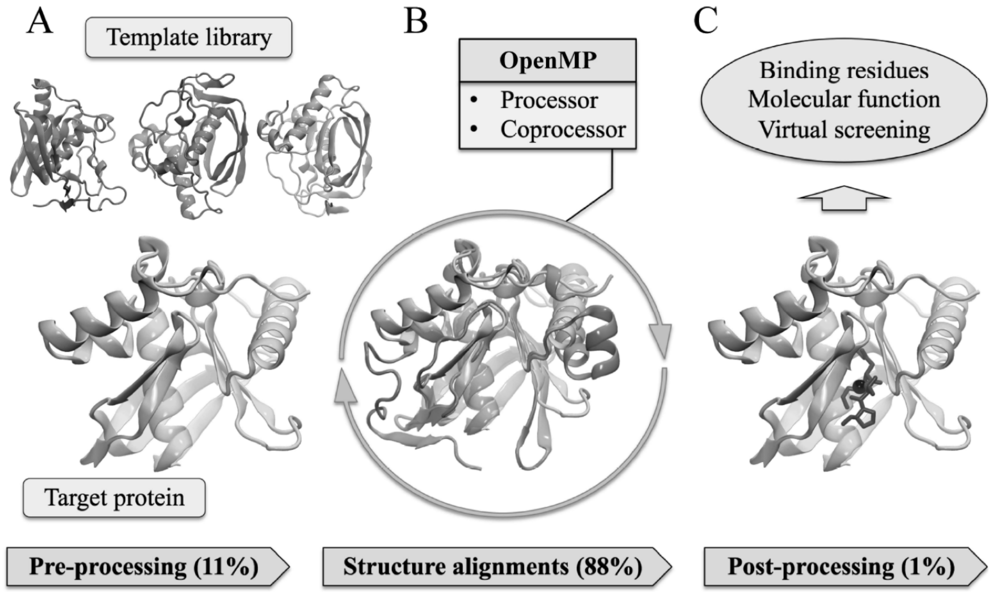

Fig. (2). Parallel implementation of *e*FindSite. (**A**) Pre-processing: for a given target protein (green cartoon) template proteins (shown in orange, pink, and cyan) are selected from the *e*FindSite library. (**B**) The computation of template-target structure alignments is parallelized using OpenMP for processor and coprocessor units. (**C**) Post-processing: ligand-binding sites are predicted. A blue ball represents the top-ranked pocket, whereas a bound ligand molecule in the crystal structure is shown as red sticks. Results from *e*FindSite can be subsequently used for binding residue prediction, functional annotation, and virtual screening. Numbers in parentheses inside pentagons at the bottom correspond to the percentage of the total wall time for a serial execution.

processing a dataset of 501 proteins that would take 36.8 hours on a single processor core can now be completed in 2.1 hours on a single Stampede node. This dramatically improved performance of *e*FindSite is particularly important for its large-scale applications, such as drug repositioning, where tens of thousands of proteins need to be targeted. From the programming standpoint, *e*FindSite represents typical scientific software written by domain scientists using different languages and styles. The parallelization of *e*FindSite demonstrates that with minor code modifications, a complex C++/Fortran77 code could be ported to Intel Xeon Phi yielding satisfactory speedups. Thus, adapting scientific software for the Intel Xeon Phi architecture is relatively straightforward yet very rewarding.

Parallelization of Molecular Force Fields

Most molecular docking algorithms used in structure-based virtual screening model protein-drug systems as sets of interacting particles that correspond to either individual atoms [102-105] or pseudo-atoms representing multi-atom moieties, such as aromatic rings and functional groups [39, 106-108]. A variety of force fields include non-bonded interactions between particles, such as electrostatic and van der Waals, as important components of scoring functions. It has been estimated that computing these interactions, often over long distances, makes up 80-95% of the total execution time [109]. Since the calculation of non-bonded interactions is easily parallelizable, significant speedups of molecular docking can be expected by moving these operations to massively parallel accelerators. Indeed, a recent study demonstrated that Intel Xeon Phi is well-suited for the acceleration of non-bonded electrostatic interaction kernels [110]. Particularly for large systems composed of 2^{26} protein and 2^{16} ligand atoms, the performance of Xeon Phi is comparable to that of NVIDIA Tesla K20x GPU. Note that the latter features two times higher peak performance for single-precision operations than Xeon Phi; using double-precision computations, the performance of Xeon Phi is expected to be even closer to that of K20x.

Calculating all-against-all pairwise non-bonded interactions involves iterating over all ligand and protein atoms inside a double loop. These calculations can be significantly accelerated using SIMD instructions that are capable of performing the same operation simultaneously on multiple data points. However, an appropriate construction of data structures is critical for the parallel performance and the full utilization of the SIMD units. In molecular docking, ligand and protein atoms are described in terms of their positions and parameters. In Fig. (**3**), we illustrate two possible models for the memory allocation for ligand caffeine. Fig. (**3A**) shows the chemical structure of caffeine that is composed of 14 heavy atoms. Fig. (**3B**) exemplifies the Array of Structures (AoS) memory layout for caffeine, where each atom is represented by its Cartesian *x*, *y* and *z* coordinates, and an arbitrary parameter *p*, which can carry a partial charge (for electrostatic interactions), or the atomic radius (for van der Waals interactions). Since, the parameters for each atom are stored contiguously, computing leads to horizontal operations that consume multiple SIMD execution slots but produce only a single result. In other words, the AoS format may cause data elements required for a single computation, *e.g.* $x1$, $y1$, $z1$ and $p1$, to be located far from each other in memory, thus falling into separate cache-lines. This inefficient pattern typically results in a poor cache utilization severely limiting the parallel performance.

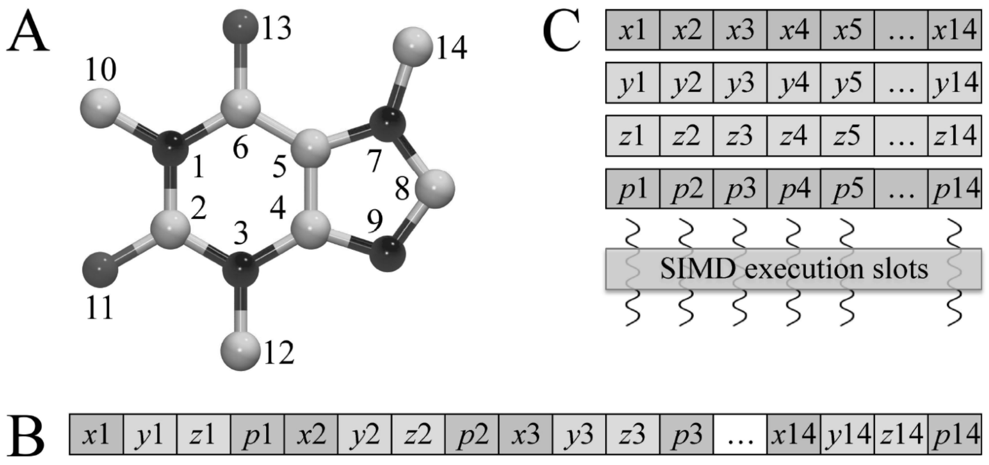

Fig. (3). Comparison of Structure of Arrays and Array of Structures data layouts. (**A**) Chemical structure of caffeine with individual heavy atoms numbered and colored by the atom type (carbon – green, nitrogen – blue, oxygen – red). (**B**) Array of Structures memory layout. (**C**) Structure of Arrays memory layout; SIMD execution slots are represented by black waves. In **B** and **C**, each heavy atom is associated with four data points: Cartesian coordinates *x*, *y* and *z*, and an arbitrary parameter *p*.

In contrast, the Structure of Arrays (SoA) memory layout allows for a more efficient use of the SIMD instructions. This is shown for caffeine in Fig. (**3C**), where the data is ready for computation in an optimal vertical arrangement. The SoA format takes advantage of all SIMD elements available, *i.e.* computations involving multiple atoms are performed simultaneously using multiple SIMD execution slots producing a unique result for each slot. This leads to a better utilization of the coprocessor bandwidth and cache. Consequently, calculating electrostatic non-bonded interactions on Xeon Phi using SoA is almost twice as fast as for the AoS. As demonstrated for a model system comprising 10,240,000 protein and 8,192 ligand atoms, using all 240 threads available on the coprocessor completes the calculations in 4,159 and 6,848 ms for the SoA and AoS implementations, respectively [110].

Optimally arranged data structures also significantly improved the computational throughput of similar calculations involving simple rotations and translations of 1,500×1,500 grid points [111]. The SoA model on the 1.09 GHz Xeon Phi SE10P coprocessor with the new 512-bit Intel Initial Many Core Instruction vectorization achieves a single- (double-) precision performance of 483.6 (199.9) GFLOPS. For comparison, the fastest implementation of the same algorithm on 2.7GHz Xeon E5-2680 processor using 256-bit Advanced Vector Extensions (AVX) yields 257.7 (111.6) GFLOPS. Therefore, SIMD capable hardware offers a great platform for the development of molecular force field codes dominated by non-bonded interaction calculations, however, the selection of suitable data structures is mandatory in order to maximize the accelerator performance.

Virtual Screening

The Bristol University Docking Engine (BUDE), an algorithm that simulates the binding of drug candidates to their macromolecular targets, was one of the first projects to speed up molecular docking in structure-based virtual screening by using massively parallel accelerators [112, 113]. The force field implemented in BUDE comprises physicochemical potentials developed for *ab initio* protein folding simulations [114]. The conformational space is efficiently explored using the Evolutionary Monte Carlo protocol with six translational and rotational degrees of freedom. The docking procedure first constructs a set of configurations uniformly covering the search space, which are subsequently used to seed the subpopulations of protein-drug configurations. The system progressively evolves exploring the conformational space in order to locate the global energy minimum that is finally taken as the predicted binding pose. The docking procedure involves generating a large number of configurations, whose binding affinity towards the receptor protein is evaluated by a scoring function. Therefore, virtual screening applications employing hundreds of thousands to even millions of drug candidates require a significant acceleration of docking simulations for individual compounds.

In this spirit, a port of BUDE for heterogeneous computing was developed using OpenCL. This particular programing model was chosen because OpenCL offers high performance portability, so that the same code can be deployed across a variety of hardware architectures. Moreover, the application of docking algorithms, such as BUDE, in structure-based virtual screening exploits different degrees of parallelism at both data and task levels, which is supported by OpenCL. Since the vast majority of the computations in BUDE are related to the evaluation of binding energy for a given drug-protein configuration, this part of the code was implemented as a single, highly optimized OpenCL kernel. Special attention was paid to the elimination of branches, which are known to severely impact the performance of parallel codes. Here, conditional branches in the energy evaluation kernel were converted into the combination of predicated selection and multiplication, which eliminate the control flow from the code. Furthermore, the memory footprint of the force field was significantly reduced to more efficiently align with memory interfaces of current accelerator devices. Similar to the implementation of molecular force field described in the preceding section, the AoS data layout used for transformation descriptors was replaced with a SIMD-friendly SoA format. Other optimizations include storing force field parameters in the fast on-chip memory, removing parameter redundancy and building some constant values directly into the energy kernel, as well as increasing data reuse within the OpenCL kernel to improve arithmetic intensity.

Tested on Xeon Phi SE10P, BUDE achieved an efficiency of 32% of peak performance with sustained 680 GFLOPS. Since the code was primarily optimized for NVIDIA GPUs, further modifications to the kernel would be necessary in order to improve the performance of BUDE on Xeon Phi devices. The same code executed on the host dual-processor with a total of 32 cores (Xeon E5-2687W clocked at 3.1 GHz) achieved the sustained efficiency of 44% and 350 GFLOPS. Compared to the baseline Fortran implementation utilizing 32 hardware threads, executing the OpenCL version on the processor and coprocessor delivers 1.3× and 2× speedups, respectively. Although we reviewed only the performance of BUDE on Xeon Phi, this study is an example of a remarkably successful implementation of a molecular docking code that features high performance portability targeting various many-core platforms. In contrast to other approaches that often move only computationally intensive portions of the code to the device, the entire molecular docking algorithm was ported to the accelerator. Finally, as a unique feature of BUDE, the same OpenCL code was demonstrated to sustain a high fraction of peak performance of about 40% across a variety of hardware architectures.

CONCLUSION

Modern drug discovery is no longer performed exclusively in wet labs; computer-aided drug design has become an integral component of almost every aspect of drug development. The exponential growth of genomic knowledge resulting from continuous advances in gene sequencing technologies holds a significant promise for the pharmaceutical industry to develop better and safer therapeutics. It has also created new challenges because processing these vast datasets for drug design requires an unprecedented computing power. Consequently, massively parallel accelerators have a great potential to accelerate scientific discovery. Here, we reviewed a new heterogeneous computing platform equipped with many-core Intel Xeon Phi coprocessors, focusing on the application of these devices to modern computer-aided drug discovery. Specifically, we highlighted various algorithms used in drug binding site prediction, biomolecular simulations, and structure-based virtual screening to demonstrate that significant speedups can be achieved by porting serial codes to the accelerator. In contrast to other parallel architectures, coding for Intel Xeon Phi is relatively straightforward with fairly short programming cycles, however, a thorough code optimization is mandatory to fully utilize the parallel capability of this many-core platform. Special attention should be drawn to a proper code vectorization, since making use of SIMD executions is essential to bring out the full potential of the coprocessor. There is no doubt that the heterogeneous parallel computing using many-core devices is vital to accelerate the computational components of modern drug discovery pipelines, thus we expect a vigorous development of new algorithms and codes for the coming years.

CONFLICT OF INTEREST

The authors confirm that this article content has no conflict of interest.

ACKNOWLEDGEMENTS

This study was supported by the National Science Foundation under the NSF EPSCoR Cooperative Agreement No. EPS-1003897 and the Louisiana Board of Regents through the Board of Regents Support Fund [contract LEQSF(2012-15)-RD-A-05]. The authors are grateful for discussions and comments from the members of the Technologies for Extreme Scale Computing (TESC) team formed within the Louisiana Alliance for Simulation-Guided Materials Applications (LA-SiGMA). This project used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by the National Science Foundation grant number OCI-1053575.

REFERENCES

[1] Reiss T. Drug discovery of the future: the implications of the human genome project. Trends Biotechnol 2001; 19: 496-9.
[2] Williams M. Genome-based drug discovery: prioritizing disease-susceptibility/disease-associated genes as novel drug targets for schizophrenia. Curr Opin Investig Drugs 2003; 4: 31-6.
[3] Sali A, Blundell TL. Comparative protein modelling by satisfaction of spatial restraints. J Mol Biol 1993; 234: 779-815.
[4] Biasini M, Bienert S, Waterhouse A*, et al.* SWISS-MODEL: modelling protein tertiary and quaternary structure using evolutionary information. Nucleic Acids Res 2014; 42: W252-8.
[5] Rohl CA, Strauss CE, Misura KM, Baker D. Protein structure prediction using Rosetta. Methods Enzymol 2004; 383: 66-93.
[6] Yang J, Yan R, Roy A, Xu D, Poisson J, Zhang Y. The I-TASSER Suite: protein structure and function prediction. Nat Methods 2014; 12: 7-8.
[7] Messaoudi A, Belguith H, Ben Hamida J. Homology modeling and virtual screening approaches to identify potent inhibitors of VEB-1 beta-lactamase. Theor Biol Med Model 2013; 10: 22.
[8] Evers A, Klabunde T. Structure-based drug discovery using GPCR homology modeling: successful virtual screening for antagonists of the alpha1A adrenergic receptor. J Med Chem 2005; 48: 1088-97.
[9] Yoshikawa Y, Oishi S, Kubo T, Tanahara N, Fujii N, Furuya T. Optimized method of G-protein-coupled receptor homology modeling: its application to the discovery of novel CXCR7 ligands. J Med Chem 2013; 56: 4236-51.
[10] Christopoulos A. Allosteric binding sites on cell-surface receptors: novel targets for drug discovery. Nat Rev Drug Discov 2002; 1: 198-210.
[11] Dukka BK. Structure-based methods for computational protein functional site prediction. Comput Struct Biotechnol J 2013; 8: e201308005.
[12] Hendlich M, Rippmann F, Barnickel G. LIGSITE: automatic and efficient detection of potential small molecule-binding sites in proteins. J Mol Graph Model 1997; 15: 359-63, 389.
[13] Le Guilloux V, Schmidtke P, Tuffery P. Fpocket: an open source platform for ligand pocket detection. BMC Bioinformatics 2009; 10: 168.
[14] Laskowski RA. SURFNET: a program for visualizing molecular surfaces, cavities, and intermolecular interactions. J Mol Graph 1995; 13: 323-30, 307-8.
[15] Laurie AT, Jackson RM. Q-SiteFinder: an energy-based method for the prediction of protein-ligand binding sites. Bioinformatics 2005; 21: 1908-16.
[16] Ngan CH, Hall DR, Zerbe B, Grove LE, Kozakov D, Vajda S. FTSite: high accuracy detection of ligand binding sites on unbound protein structures. Bioinformatics 2012; 28: 286-7.
[17] An J, Totrov M, Abagyan R. Pocketome *via* comprehensive identification and classification of ligand binding envelopes. Mol Cell Proteomics 2005; 4: 752-61.

[18] Roy A, Yang J, Zhang Y. COFACTOR: an accurate comparative algorithm for structure-based protein function annotation. Nucleic Acids Res 2012; 40: W471-7.

[19] Berman HM, Westbrook J, Feng Z, *et al.* The protein data bank. Nucleic Acids Res 2000; 28: 235-42.

[20] Zhang Y, Skolnick J. TM-align: a protein structure alignment algorithm based on the TM-score. Nucleic Acids Res 2005; 33: 2302-9.

[21] Ashburner M, Ball CA, Blake JA*, et al.* Gene ontology: tool for the unification of biology. The gene ontology consortium. Nat Genet 2000; 25: 25-9.

[22] Lee Y, Elvitigala DA, Whang I*, et al.* Structural and functional characterization of a novel molluskan ortholog of TRAF and TNF receptor-associated protein from disk abalone (Haliotis discus discus). Fish Shellfish Immunol 2014; 40: 32-9.

[23] Naveed M, Ahmed I, Khalid N, Mumtaz AS. Bioinformatics based structural characterization of glucose dehydrogenase (gdh) gene and growth promoting activity of *Leclercia sp*. QAU-66. Braz J Microbiol 2014; 45: 603-11.

[24] Shi L, Zhang H, Qiu Y*, et al.* Biochemical characterization and ligand-binding properties of trehalose-6-phosphate phosphatase from Mycobacterium tuberculosis. Acta Biochim Biophys Sin (Shanghai) 2013; 45: 837-44.

[25] Guo K, Wang W, Luo L, Chen J, Guo Y, Cui F. Characterization of an aphid-specific, cysteine-rich protein enriched in salivary glands. Biophys Chem 2014; 189: 25-32.

[26] Wass MN, Kelley LA, Sternberg MJ. 3DLigandSite: predicting ligand-binding sites using similar structures. Nucleic Acids Res 2010; 38: W469-73.

[27] Brylinski M, Skolnick J. A threading-based method (FINDSITE) for ligand-binding site prediction and functional annotation. Proc Natl Acad Sci USA 2008; 105: 129-34.

[28] Skolnick J, Brylinski M. FINDSITE: a combined evolution/structure-based approach to protein function prediction. Brief Bioinform 2009; 10: 378-91.

[29] Feinstein WP, Brylinski M. eFindSite: Enhanced fingerprint-based virtual screening against predicted ligand binding sites in protein models. Mol Inf 2014; 33: 135-50.

[30] Brylinski M, Feinstein WP. eFindSite: Improved prediction of ligand binding sites in protein models using meta-threading, machine learning and auxiliary ligands. J Comput Aided Mol Des 2013; 27: 551-67.

[31] Reymond J-L, van Deursen R, Blum LC, Ruddigkeit L. Chemical space as a source for new drugs. Med Chem Commun 2010; 1: 30-38.

[32] Willett P, Winterman V, Bawden D. Implementation of nearest-neighbor searching in an online chemical structure search system. J Chem Inf Comput Sci 1986; 26: 36-41.

[33] Wang X, Wang JT. Fast similarity search in three-dimensional structure databases. J Chem Inf Comput Sci 2000; 40: 442-51.

[34] Gironés X, Robert D, Carbó-Dorca R. TGSA: A molecular superposition program based on topo-geometrical considerations. J Comput Chem 2001; 22: 255-63.

[35] Cavasotto CN, Orry AJ. Ligand docking and structure-based virtual screening in drug discovery. Curr Top Med Chem 2007; 7: 1006-14.

[36] Lengauer T, Rarey M. Computational methods for biomolecular docking. Curr Opin Struct Biol 1996; 6: 402-6.

[37] Goodsell DS, Morris GM, Olson AJ. Automated docking of flexible ligands: applications of AutoDock. J Mol Recognit 1996; 9: 1-5.

[38] Kuntz ID, Blaney JM, Oatley SJ, Langridge R, Ferrin TE. A geometric approach to macromolecule-ligand interactions. J Mol Biol 1982; 161: 269-88.

[39] Brylinski M, Skolnick J. Q-Dock: Low-resolution flexible ligand docking with pocket-specific threading restraints. J Comput Chem 2008; 29: 1574-88.

[40] Verdonk ML, Cole JC, Hartshorn MJ, Murray CW, Taylor RD. Improved protein-ligand docking using GOLD. Proteins 2003; 52: 609-23.

[41] Brylinski M. Nonlinear scoring functions for similarity-based ligand docking and binding affinity prediction. J Chem Inf Model 2013; 53: 3097-112.

[42] Neves MA, Totrov M, Abagyan R. Docking and scoring with ICM: the benchmarking results and strategies for improvement. J Comput Aided Mol Des 2012; 26: 675-86.

[43] Ghosh S, Nie A, An J, Huang Z. Structure-based virtual screening of chemical libraries for drug discovery. Curr Opin Chem Biol 2006; 10: 194-202.

[44] Seifert MH, Lang M. Essential factors for successful virtual screening. Mini Rev Med Chem 2008; 8: 63-72.

[45] Villoutreix BO, Eudes R, Miteva MA. Structure-based virtual ligand screening: recent success stories. Comb Chem High Throughput Screen 2009; 12: 1000-16.

[46] Halperin I, Ma B, Wolfson H, Nussinov R. Principles of docking: An overview of search algorithms and a guide to scoring functions. Proteins 2002; 47: 409-43.

[47] Shoichet BK, Kuntz ID, Bodian DL. Molecular docking using shape descriptors. J Comput Chem 1992; 13: 380-97.

[48] Irwin JJ, Sterling T, Mysinger MM, Bolstad ES, Coleman RG. ZINC: a free tool to discover chemistry for biology. J Chem Inf Model 2012; 52: 1757-68.

[49] Neudert G, Klebe G. DSX: a knowledge-based scoring function for the assessment of protein-ligand complexes. J Chem Inf Model 2011; 51: 2731-45.

[50] Zhang C, Liu S, Zhu Q, Zhou Y. A knowledge-based energy function for protein-ligand, protein-protein, and protein-DNA complexes. J Med Chem 2005; 48: 2325-35.

[51] Pedemonte M, Nesmachnow S, Cancela H. A survey on parallel ant colony optimization. Appl Soft Comput 2011; 11: 5181-97.

[52] Eleftheriou M, Rayshubski A, Pitera JW, Fitch BG, Zhou R, Germain RS. Parallel implementation of the replica exchange molecular dynamics algorithm on Blue Gene/L. 20th International Symposium on Parallel and Distributed Processing, Rhodes Island, Greece, 2006.

[53] Knysh DS, Kureichik VM. Parallel genetic algorithms: a survey and problem state of the art. J Comput Syst Sci Int 2010; 49: 579-89.

[54] What is GPU accelerated computing? Available at: http://www.nvidia.com/object/what-is-gpu-computing.html [accessed January 31, 2015].

[55] Parallel programming and computing platform. Available at: http://www.nvidia.com/object/cuda_home_new.html [accessed January 31, 2015].

[56] OpenACC directives for accelerators. Available at: http://www.openacc-standard.org/ [accessed January 31, 2015].

[57] Vouzis PD, Sahinidis NV. GPU-BLAST: using graphics processors to accelerate protein sequence alignment. Bioinformatics 2011; 27: 182-8.

[58] Pang B, Zhao N, Becchi M, Korkin D, Shyu CR. Accelerating large-scale protein structure alignments with graphics processing units. BMC Res Notes 2012; 5: 116.

[59] Dlugosz M, Zielinski P, Trylska J. Brownian dynamics simulations on CPU and GPU with BD_BOX. J Comput Chem 2011; 32: 2734-44.

[60] Weigel M. Simulating spin models on GPU. Comp Phys Commun 2011; 182: 1833-6.

[61] Roberts E, Stone JE, Sepulveda L, Hwu WMW, Luthey-Schulten Z. Long time-scale simulations of *in vivo* diffusion using GPU hardware. Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing, Rome, Italy, 2009: 1-8.

[62] AMD compute cores. Available at: http://www.amd.com/en-us/innovations/software-technologies/processors-for-business/compute-cores [accessed January 31, 2015].

[63] Einstein@Home. Available at: http://www.einsteinathome.org/ [accessed January 31, 2015].

[64] Projects with ATI/AMD applications. Available at: http://boinc.berkeley.edu/wiki/ATI_Radeon [accessed January 31, 2015].

[65] AMD energy-efficient “Carrizo” system-on-chip. Available at: http://www.amd.com/en-us/press-releases/Pages/amd-discloses-architecture-2015feb23.aspx [accessed January 31, 2015].

[66] Intel Xeon Phi coprocessor. Available at: https://software.intel.com/en-us/mic-developer [accessed January 31, 2015].

[67] Intel$^{®}$ Xeon Phi™ core micro-architecture. Available at: https://software.intel.com/en-us/articles/intel-xeon-phi-core-micro-architecture [accessed January 31, 2015].

[68] Feinstein W, Moreno J, Jarrell M, Brylinski M. Accelerating the pace of protein functional annotation with Intel Xeon Phi coprocessors. IEEE Trans Nanobioscience 2015; 14: 429-39.

[69] Venetis IE, Goumas G, Geveler M, Ribbrock D. Porting FEAST-FLOW to the Intel Xeon Phi: Lessons learned. PRACE 2014: White Paper 139.

[70] Hemsoth N. Full Details Uncovered on Chinese Top Supercomputer. HPCwire, Available at: http://www.hpcwire.com/ 2013/06/02/full_details_uncovered_on_chinese_top_super computer/ 2013.

[71] Beacon system configuration. Available at: https:// www.nics.tennessee.edu/computing-resources/beacon/configuration [accessed January 31, 2015].

[72] Stampede at TACC. Available at: https://www.tacc.utexas.edu/ user-services/user-guides/stampede-user-guide [accessed January 31, 2015].

[73] SuperMIC. Available at: http://www.hpc.lsu.edu/resources/hpc/ system.php?system=SuperMIC [accessed January 31, 2015].

[74] Message Passing Interface forum. Available at: http://www.mpi-forum.org/ [accessed January 31, 2015].

[75] MPICH. Available at: http://www.mpich.org/ [accessed January 31, 2015].

[76] Open MPI: open source high performance computing. Available at: http://www.open-mpi.org/ [accessed January 31, 2015].

[77] Collignon B, Schulz R, Smith JC, Baudry J. Task-parallel message passing interface implementation of Autodock4 for docking of very large databases of compounds using high-performance super-computers. J Comput Chem 2011; 32: 1202-9.

[78] OpenMP specifications. Available at: http://openmp.org/wp/ openmp-specifications/ [accessed January 31, 2015].

[79] OpenMP compilers. Available at: http://openmp.org/wp/openmp-compilers/ [accessed January 31, 2015].

[80] Cramer T, Schmidl D, Klemm M, an Mey D. OpenMP programming on Intel Xeon Phi coprocessors: An early performance comparison. Proceedings of Many-core Applications Research Community (MARC) Symposium at RWTH Aachen University, Aachen, Germany, 2012: 38-44.

[81] Saule E, Kaya K, Çatalyürek ÜV. Performance evaluation of sparse matrix multiplication kernels on Intel Xeon Phi. In: Wyrzykowski R, Dongarra J, Karczewski K, Waśniewski J, Parallel Processing and Applied Mathematics. Springer Berlin Heidelberg, 2014; pp. 559-70.

[82] Teodoro G, Kurc T, Kong J, Cooper L, Saltz J. Comparative performance analysis of Intel Xeon Phi, GPU, and CPU: A case study from microscopy image analysis. 28th IEEE International Symposium on Parallel and Distributed Processing, Phoenix, AZ, 2014: pp. 1063-72.

[83] Chen C, Yang CQ, Yao WK, Qi J, Wu Q. Accelerating PQMRCGSTAB algorithm on Xeon Phi. Adv Materials Res 2013; 709: 555-62.

[84] Szustak L, Rojek K, Gepner P. Using Intel Xeon Phi coprocessor to accelerate computations in MPDATA algorithm. In: Wyrzykowski R, Dongarra J, Karczewski K, Waśniewski J, Parallel Processing and Applied Mathematics. Springer Berlin Heidelberg, 2014; pp. 582-92.

[85] Schmidl D, Cramer T, Wienke S, Terboven C, Müller MS. Assessing the performance of OpenMP programs on the Intel Xeon Phi. In: Wolf F, Mohr B, an Mey D, Euro-Par 2013 Parallel Processing. Springer Berlin Heidelberg, 2013; pp. 547-58.

[86] Fang J, Sips H, Zhang L, Xu C, Che Y, Varbanescu AL. Test-driving Intel Xeon Phi. Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering, Dublin, Ireland, 2014: pp. 137-48.

[87] Gorobets A, Trias FX, Borrell R, Lehmkuhl O, Oliva A. Hybrid MPI +OpenMP parallelization of an FFT-based 3D Poisson solver with one periodic direction. Comput Fluids 2011; 49: 101-9.

[88] Joó B, Kalamkar DD, Vaidyanathan K*, et al.* Lattice QCD on Intel Xeon Phi coprocessors. In: Kunkel JM, Ludwig T, Meuer HW, Supercomputing. Springer Berlin Heidelberg, 2013; pp. 40-54.

[89] Reid F, Bethune I. Optimising CP2K for the Intel Xeon Phi. PRACE 2014: White Paper 140.

[90] The open standard for parallel programming of heterogeneous systems. Available at: https://www.khronos.org/opencl/ [accessed January 31, 2015].

[91] OpenCL applications. Available at: https://www.khronos.org/ opencl/resources/opencl-applications-using-opencl [accessed January 31, 2015].

[92] Gorobets A, Trias FX, Borrell R, Oyarzun G, Oliva A. Direct numerical simulation of turbulent flows with parallel algorithms for various computing architectures. 6th European Conference on Computational Fluid Dynamics, Barcelona, Spain, 2014.

[93] Kaiser H, Brodowicz M, Sterling T. ParalleX: An advanced parallel execution model for scaling-impaired applications. Proceedings of the 2009 International Conference on Parallel Processing Workshops, Vienna, Austria, 2009: 394-401.

[94] HPX documents. Available at: http://stellar.cct.lsu.edu/docs/ [accessed January 31, 2015].

[95] Kaiser H, Heller T, Adelstein-Lelbach B, Serio A, Fey D. HPX - A task based programming model in a global address space. PGAS '14 Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models, Eugene, Oregon, USA, 2014: 1-11.

[96] Heller T, Kaiser H, Schäfer A, Fey D. Using HPX and LibGeoDecomp for scaling HPC applications on heterogeneous supercomputers. ScalA '13, Proceedings of the Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, Denver CO, USA, 2013.

[97] Computing benchmarks. Available at: http://blog.xcelerit.com/ intel-xeon-phi-*vs*-nvidia-tesla-gpu/ [accessed January 31, 2015].

[98] Pennycook SJ, Hughes CJ, Smelyanskiy M, Jarvis SA. Exploring SIMD for Molecular Dynamics, using Intel Xeon processors and Intel Xeon Phi coprocessors. 27th IEEE International Symposium on Parallel and Distributed Processing, Boston, MA, 2013: pp. 1085-97.

[99] Brylinski M, Feinstein WP. Setting up a meta-threading pipeline for high-throughput structural bioinformatics: eThread software distribution, walkthrough and resource profiling. J Comput Sci Syst Biol 2012; 6: 1-10.

[100] Brylinski M, Lingam D. eThread: a highly optimized machine learning-based approach to meta-threading and the modeling of protein tertiary structures. PLoS One 2012; 7: e50200.

[101] Guilloteau J-P, Mathieu M, Giglione C*, et al.* The crystal structures of four peptide deformylases bound to the antibiotic actinonin reveal two distinct types: A platform for the structure-based design of antibacterial agents. J Mol Biol 2002; 320: 951-962.

[102] Ewing TJ, Makino S, Skillman AG, Kuntz ID. DOCK 4.0: search strategies for automated molecular docking of flexible molecule databases. J Comput Aided Mol Des 2001; 15: 411-28.

[103] Meiler J, Baker D. ROSETTALIGAND: protein-small molecule docking with full side-chain flexibility. Proteins 2006; 65: 538-48.

[104] Rarey M, Kramer B, Lengauer T, Klebe G. A fast flexible docking method using an incremental construction algorithm. J Mol Biol 1996; 261: 470-89.

[105] Trott O, Olson AJ. AutoDock Vina: improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading. J Comput Chem 2010; 31: 455-61.

[106] Brylinski M, Skolnick J. Q-Dock(LHM): Low-resolution refinement for ligand comparative modeling. J Comput Chem 2010; 31: 1093-105.

[107] Vakser IA. Low-resolution docking: prediction of complexes for underdetermined structures. Biopolymers 1996; 39: 455-64.

[108] Wojciechowski M, Skolnick J. Docking of small ligands to low-resolution and theoretically predicted receptor structures. J Comput Chem 2002; 23: 189-97.

[109] Kuntz SK, Murphy RC, Niemier MT, Izaguirre JA, Kogge PM. Petaflop computing for protein folding. Proceedings of the Tenth SIAM Conference on Parallel Processing for Scientific Computing, Portsmouth, Virginia, 2001: pp. 12-4.

[110] Fang J, Varbanescu AL, Imbernon B, Cecilia JM, Perez-Sanchez H. Parallel computation of non-bonded interactions in drug discovery: Nvidia GPUs *vs*. Intel Xeon Phi. 2nd International Work-Conference on Bioinformatics and Biomedical Engineering, Granada, Spain, 2014.

[111] Besl P. A case study comparing AoS (Arrays of Structures) and SoA (Structures of Arrays) data layouts for a compute-intensive loop run on Intel Xeon processors and Intel Xeon Phi product family coprocessors. Intel Article: 392271.

[112] Sessions RB, McIntosh-Smith S. An accelerated, computer assisted molecular modeling method for drug design. International Supercomputing Conference, Dresden, Germany, 2008.

[113] McIntosh-Smith S, Price J, Sessions RB, Ibarra AA. High performance *in silico* virtual drug screening on many-core processors. Int J High Perform Comput Appl 2014; [in press].

[114] Gibbs N, Clarke AR, Sessions RB. Ab initio protein structure prediction using physicochemical potentials and a simplified off-lattice model. Proteins 2001; 43: 186-202.