Accelerating the Pace of Protein Functional Annotation With Intel Xeon Phi Coprocessors

Wei P. Feinstein, Juana Moreno, Mark Jarrell, and Michal Brylinski*

Abstract—**Intel Xeon Phi is a new addition to the family of powerful parallel accelerators. The range of its potential applications in computationally driven research is broad; however, at present, the repository of scientific codes is still relatively limited. In this study, we describe the development and benchmarking of a parallel version of *e*FindSite, a structural bioinformatics algorithm for the prediction of ligand-binding sites in proteins. Implemented for the Intel Xeon Phi platform, the parallelization of the structure alignment portion of *e*FindSite using pragma-based OpenMP brings about the desired performance improvements, which scale well with the number of computing cores. Compared to a serial version, the parallel code runs 11.8 and 10.1 times faster on the CPU and the coprocessor, respectively; when both resources are utilized simultaneously, the speedup is 17.6. For example, ligand-binding predictions for 501 benchmarking proteins are completed in 2.1 hours on a single Stampede node equipped with the Intel Xeon Phi card compared to 3.1 hours without the accelerator and 36.8 hours required by a serial version. In addition to the satisfactory parallel performance, porting existing scientific codes to the Intel Xeon Phi architecture is relatively straightforward with a short development time due to the support of common parallel programming models by the coprocessor. The parallel version of *e*FindSite is freely available to the academic community at www.brylinski.org/efindsite.**

Index Terms—***e*FindSite, heterogeneous computer architectures, high performance computing, Intel Xeon Phi, ligand-binding site prediction, Many Integrated Cores, offload mode, parallel processing, performance benchmarks, protein functional annotation.**

I. Introduction

CONTINUING advances in genome sequencing technologies lead to the accumulation of raw genomic data, which awaits functional annotation [1], [2]. Presently, computational approaches to protein structure modeling and functional inference represent the most practical strategy to keep up with annotating the massive volume of DNA sequences [3], [4]. The resulting knowledge facilitates a broad range of research in life sciences including systems biology and drug development and discovery. For instance, systems biology focuses on studying cellular mechanisms as a whole by constructing and analyzing the networks of complex molecular interactions and signaling pathways at the level of complete proteomes [5]. Such systems-level approaches hold a significant promise to develop new treatments for complex diseases, which often require a simultaneous modulation of multiple protein targets. This is the domain of polypharmacology, an emerging field that integrates systems biology and drug discovery [6]. Incorporating large biological datasets is critical for the success of many systems-level applications; however, considering the vast amounts of data to be processed, unprecedented computing power is required to achieve a reasonably short time-to-completion. Significant challenges remain given that the currently available biological data may easily outbalance accessible computing resources.

In that regard, traditional single threaded processor systems are no longer viable to meet the challenges of modern computationally driven research. Therefore, parallel high-performance computing has become a key component in solving large-scale computational problems. For instance, graphics processing units (GPUs) developed primarily for gaming purposes, are now routinely used to speed up scientific applications in numerous research areas [7]. Examples include GPU-accelerated molecular [8] and Brownian dynamics [9], spin model simulations [10], the modeling of *in vivo* diffusion [11], phylogenetic analyses [12], as well as protein sequence [13] and structure alignments [14]. Not surprisingly, the number of GPU-powered high-performance systems among the world's top 500 supercomputers continues to grow [15]. Likewise, the newly launched Intel Xeon Phi coprocessor expands the repertoire of high-performance computing architectures offering massively parallel capabilities for a broad range of applications. The Intel Xeon Phi coprocessor, equipped with tens of x86-like computing cores, plugs into the standard PCIe port communicating directly with the main processor to accelerate computations. The underlying x86 architecture supports common parallel programming models and offers familiarity and flexibility in porting existing codes to benefit from the heterogeneous computing environment. In contrast to the code development for GPU, which often requires a substantial programming expertise and laborious code conversions, the transition to Intel Xeon Phi is relatively straightforward and time efficient.

Since the Intel Xeon Phi technology featuring many integrated cores (MIC) is very new, the repository of scientific

Manuscript received September 12, 2014; revised November 30, 2014; accepted February 08, 2015. Date of publication March 05, 2015; date of current version May 29, 2015. This work 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]. This project used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by the National Science Foundation grant number OCI-1053575. *Asterisk indicates corresponding author.*

W. P. Feinstein is with Department of Biological Sciences, Louisiana State University, Baton Rouge, LA 70803 USA, and also with the Center for Computation & Technology, Louisiana State University, Baton Rouge, LA 70803 USA (e-mail: wfeinstein@lsu.edu).

J. Moreno and M. Jarrell are with the Department of Physics and Astronomy, Louisiana State University, Baton Rouge, LA 70803 USA, and also with the Center for Computation & Technology, Louisiana State University, Baton Rouge, LA 70803 USA (e-mail: moreno@phys.lsu.edu; jarrellphysics@gmail.com).

*M. Brylinski is with the Department of Biological Sciences, Louisiana State University, Baton Rouge, LA 70803 USA, and also with the Center for Computation & Technology, Louisiana State University, Baton Rouge, LA 70803 USA (e-mail: michal@brylinski.org).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNB.2015.2403776

codes is still relatively limited. However, the availability of MIC accelerators installed in many contemporary supercomputing systems, e.g., Stampede at the Texas Advanced Computing Center (TACC), stimulates a broad interest in this new architecture. Consequently, significant efforts are directed towards developing new and porting existing state-of-the-art codes to Xeon Phi as well as exploring the performance and functionality of the accelerator [16]–[18]. In this spirit, we describe a new version of the *e*FindSite algorithm, a modeling tool used in structural bioinformatics and drug discovery, that can be deployed on computing nodes equipped with Xeon Phi cards. *e*FindSite was developed to accurately identify ligand-binding sites and binding residues across large datasets of protein targets using weakly homologous templates [19], [20]. Devised to efficiently operate within the "twilight zone" of sequence similarity [21], it is especially applicable to genome-wide protein function annotation, drug design, and systems biology in general [22]. Briefly, *e*FindSite extracts ligand-binding information from evolutionarily related proteins identified in the Protein Data Bank (PDB) [23] using sensitive protein meta-threading techniques [24], [25]. Template proteins are structurally aligned onto the target protein, which is followed by the clustering of template-bound ligands to detect ligand-binding sites and residues. These predictions can be subsequently used e.g., in ligand docking and virtual screening [26], [27], molecular function inference [28] as well as in the reconstruction and analysis of biological networks and pathways [29]. In the original *e*FindSite algorithm, template-to-target structure alignments are executed sequentially, thus the identification of ligand-binding sites even for one target protein may require many CPU hours [19]. This in turn complicates genome-wide applications, where the number of protein targets and the size of the template library can be very large. Therefore, accelerating the *e*FindSite code shortens the simulation time resulting in faster genome-wide protein function annotation. In this article, we describe porting *e*FindSite to the Intel Xeon Phi platform and demonstrate that its parallel execution on nodes equipped with accelerator cards significantly reduces computational time required for the identification of ligand-binding sites across large protein datasets.

II. Procedures and Methods

A. Intel Xeon Phi Architecture

Intel Xeon Phi featuring many integrated cores (MIC) is a promising new member to the family of powerful parallel hardware accelerators. Computing systems powered by Xeon Phi are composed of a traditional CPU processor, referred to as the host, and a MIC coprocessor connected via the PCIe bus, referred to as the target [30]. The shared-memory architecture of the coprocessor allows the L1 and L2 data cache of each coprocessor core to be interconnected and remotely accessible via a fast bidirectional ring; however, memory sharing is not permitted between the host and the target. Our primary development and benchmarking system is Stampede at TACC. Each Stampede's Xeon Phi node is equipped with dual eight-core E5-2680 Sandy Bridge processors and one or two Xeon Phi SE10P coprocessors [31]; in this study, we use nodes equipped with one coprocessor. Each accelerator card deploys a stripped down Linux OS called BusyBox and removes many power-hungry operations, yet it offers a wider vector unit and a larger hardware thread count compared to host processors. Each 61-core coprocessor supports 4 hardware threads per core providing up to 244 threads, whereas the two host processors feature 8-cores each and 2 threads per core totaling up to 32 threads. However, to maximize the performance, only 16 host threads are available on Stampede nodes (1 thread per physical core). Intel Xeon Phi also has some other unique features; each computing core clocks at 1090 MHz and the device is equipped with 8 GB of GDDR5 memory and four-way simultaneous multi-threading (SMT). Its 512-bit wide single instruction, multiple data (SIMD) vectors translate into 8 double-precision or 16 single-precision floating-point numbers. In comparison, the host processor runs at a faster speed of 2700 MHz and has a larger memory of 32 GB DDR3, yet narrower 256-bit SIMD vectors.

Although the Intel processor and coprocessor have different designs, to a certain degree they share a similar architecture, which facilitates deploying codes on MIC. Currently available programming techniques include the native and the offload modes. In the former, the entire code is first cross-compiled on the host with OpenMP pragmas [32] and then executed on the coprocessor, whereas the latter offloads only parts of the code to the coprocessor [16]. Moreover, Xeon Phi also support the MPI protocol [33], thus the coprocessor can work as a self-sufficient computing node as well [18].

B. Overview of eFindSite

To induce cellular and therapeutic effects, small ligand molecules such as metabolites and drugs bind to specific sites on the target protein surface, often referred to as binding pockets. However, for many pharmacologically relevant proteins only their ligand-free experimental structures or computer-generated models are available. Therefore, at the outset of drug discovery, the identification of possible binding sites is typically the first step in inferring and modulating protein molecular function. Currently, evolution/structure-based approaches to ligand-binding prediction are the most accurate and, consequently, the most widely used [34]. These algorithms exploit structural information extracted from evolutionarily weakly related proteins, called templates, to predict ligand-binding sites and binding residues in target proteins. *e*FindSite is a recently developed evolution/structure-based approach [19], [20], which features a series of improvements over its predecessor, FINDSITE [35]. It employs a collection of new algorithms, such as highly sensitive meta-threading methods, advanced clustering techniques, and machine learning models to further increase the accuracy of pocket detection as well as to improve the tolerance to structural imperfections in protein models. Consequently, it is especially applicable in genome-wide protein function annotation projects. For a given target protein, *e*FindSite uses meta-threading [24], [25] to identify closely and remotely homologous ligand-bound templates. Subsequently, these templates are structurally aligned onto the target using Fr-TM-align [36], which is followed by the clustering of template-bound ligands using Affinity Propagation [37] and a machine learning-based ranking of the detected pockets. The

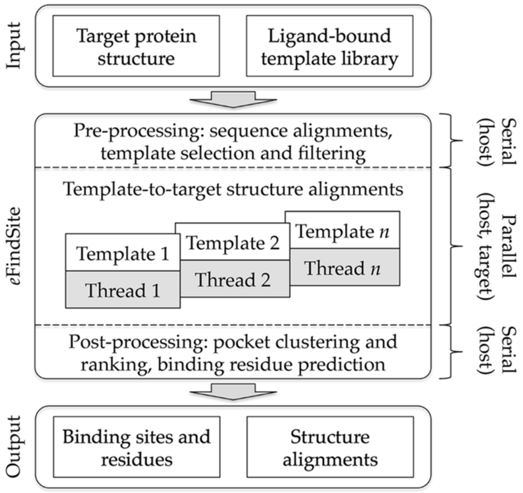

Input
Target protein structure
Ligand-bound template library
eFindSite
Pre-processing: sequence alignments, template selection and filtering
Serial (host)
Template-to-target structure alignments
Template 1
Thread 1
Template 2
Thread 2
Template *n*
Thread *n*
Parallel (host, target)
Post-processing: pocket clustering and ranking, binding residue prediction
Serial (host)
Output
Binding sites and residues
Structure alignments

Fig. 1. Workflow of the *e*FindSite algorithm. Pre- and post-processing calculations are performed using a single host thread. Structure alignments are executed in parallel using multiple host or target threads.

framework of *e*FindSite is written in C + + with the protein structure alignment portion implemented in Fortran77 and the Affinity Propagation clustering algorithm incorporated into the code as a library pre-compiled for Linux systems. Previous profiling of the *e*FindSite code revealed that, depending on the target protein length as well as the number of templates, the memory requirements could be fairly high [19]. Therefore, we decided to use the offload mode in porting *e*FindSite to the accelerator.

C. Porting eFindSite to Xeon Phi

The flowchart of the *e*FindSite algorithm is shown in Fig. 1. The input includes a target protein structure and a ligand-bound template library. The output contains predicted ligand-binding sites and residues, as well as structure alignments between the target protein and the associated templates. The procedure of binding site prediction in *e*FindSite breaks down into three consecutive stages. During pre-processing, template information is extracted from the library and sequence alignments to the target are constructed. Next, each template is structurally aligned onto the target; based on the code profiling results shown in Section III-A, we identify this portion of the code as a promising target for parallelization and porting to the accelerator. Specifically, structure alignments of a target protein against the identified templates are implemented in parallel with each template-to-target alignment computed by a different hardware thread. Pragma-based OpenMP is used to parallelize structure alignments and to perform the calculations within the host, offload them to the target, or use both resources simultaneously. As shown in Fig. 1, each individual alignment is assigned to a different thread to carry out calculations in parallel. Finally, template-to-target structure alignments are collected and used in the post-processing step to cluster the identified pockets, rank them using machine learning, and predict the corresponding binding residues.

Parallelization of the loop iterating over structure alignments is implemented using standard OpenMP pragmas with the dynamic scheduling [`schedule (dynamic)`]. The actual code for alignment calculations is written in Fortran77 and extensively uses common blocks to access the memory, which is well known to be thread unsafe. To fix this issue, we marked all common blocks in Fortran subroutines as private to threads [`!$omp threadprivate (/block_name/)`]. Specifying the number of computing threads, n, is straightforward by setting the relevant OpenMP environment variable [`export OMP_NUM_THREADS=n`]. Moreover, individual structure alignments in *e*FindSite have a larger memory footprint than the default OpenMP stack size, therefore, we also increase the memory available to each thread to 64 M [`export OMP_STACKSIZE=64 M`].

Offloading and executing portions of the code on the coprocessor requires a series of additional modifications. Because Intel Xeon Phi architecture does not allow memory sharing between the host and the target, the actual data used within the offload block need to be transferred from the host to the target prior to the calculations. To facilitate the data exchange, extra work was put in to marshal the data into flat, bitwise copy-able data structures. We note that the data is copied in and out of the target only once, thus there is virtually no overhead from moving data back and forth between the host and the target. In addition, directives for offloading the code to the target are used to instruct the compiler that the offload mode is activated and that portions of the code should be executed on the coprocessor. All subroutines and global variables used within the structure alignment code are tagged with the offload attributes [`!dir$ attributes offload:mic::subroutine_name`] and [`!dir$ attributes offload:mic::variable_name`], respectively. OpenMP executed within the offloaded code requires setting environment variables containing the "`MIC_`" prefix, i.e., [`export MIC_OMP_NUM_THREADS=n`] and [`export MIC_OMP_STACKSIZE=64 M`].

The 61-core Intel Xeon Phi coprocessor supports a large number of hardware threads for programming usage, up to 244 theoretically and 240 practically (the 61st core is reserved for the operating system, I/O operations, etc.). The physical distribution of these threads is called the thread affinity and can be controlled using Intel KMP affinity settings [38]. As shown in Fig. 2, three types of thread affinity are available on the coprocessor, balanced, scatter, and compact. Specifically, the affinity setup of [`export MIC_KMP_AFFINITY=balanced`] distributes threads uniformly to ensure that the maximum number of cores are deployed (Fig. 2(a)). Similarly, [`export MIC_KMP_AFFINITY=scatter`] assigns threads evenly across the entire core range in a round-robin fashion to maximize hardware utilization (Fig. 2(b)). In contrast, [`export MIC_KMP_AFFINITY=compact`] packs threads densely next to each other, therefore, the least amount of cores are utilized (Fig. 2(c)). Since the thread affinity can have an impact on the parallel performance, we benchmark *e*FindSite using different affinity types.

By default, an offloaded task assumes that the coprocessor is used exclusively without interfering with other processes. Therefore, when offloading concurrent multi-threaded tasks to the same coprocessor, we need to specify the core range for each task. We coordinate the task offloading using a

TABLE I
NUMBER OF TARGET PROTEINS IN THE BENCHMARKING DATASET

Protein length	Number of templates			
	50-100	100-150	150-200	200-250
200-300 aa	50	50	50	50
300-400 aa	50	50	50	50
400-500 aa	34	32	23	12

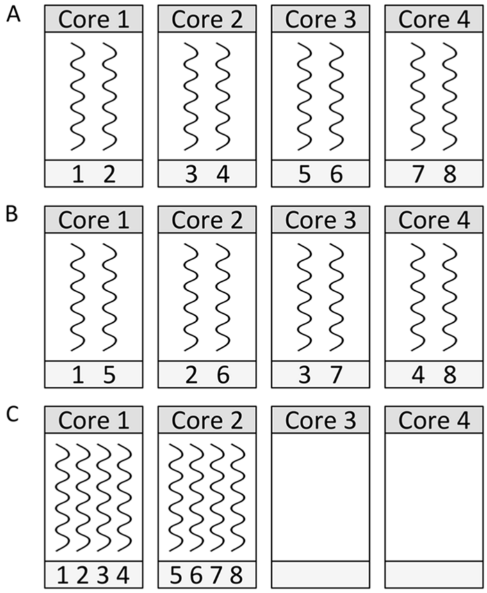


Fig. 2. Thread affinity modes available for Intel Xeon Phi. The schematic shows the distribution of 8 computing threads in a simplified system composed of 4 coprocessor cores. Threads (numbered sequentially) are assigned to different cores according to (A) balanced, (B) scatter, and (C) compact affinity settings.

thread pinning scheme to prevent spawning of threads on the same execution units. Specifically, the non-overlapping partitioning of the target device is accomplished using the `MIC_KMP_PLACE_THREADS` environment variable that allows requesting the number of cores, the number of threads, and the core range. For instance, [`export MIC_KMP_PLACE_THREADS=6c,4 t,12o`] spawns 24 threads with the compact affinity on cores 12–17. In this example, we request the total of 6 cores (argument `c`) with 4 threads per core (argument `t`), and the core range starting from core 12 (argument `o`, offset). Using this pinning scheme, we can launch multiple parallel tasks without the oversubscription of coprocessor cores as long as the total number of concurrent threads is no greater than 240.

D. Benchmarking Dataset

The set of protein-ligand complexes used in this study is a subset of the original *e*FindSite benchmarking dataset [19]. It consists of proteins of different sizes and a varying number of associated ligand-bound templates, selected to mimic real proteomic data. As shown inTable I, we first defined 12 bins depending on the number of amino acids in the target protein (200–500) and the number of templates (50–250). Next, we populated each bin with up to 50 structures by randomly selecting proteins from the original *e*FindSite dataset. For each protein, its crystal structure is used as the target; weakly homologous templates sharing less than 40% sequence identity are identified by meta-threading using *e*Thread [24], [25]. The benchmarking dataset used in this study comprises 501 protein targets.

III. RESULT AND DISCUSSION

A. Code Profiling

Most computer programs follow the 80/20 rule and spend 80% of the wall time executing 20% of the code, thus the Pareto principle can be applied to guide optimization efforts [39]. Before converting the serial version of *e*FindSite to a parallel version, we conducted a thorough profiling in order to identify portions of the code consuming the most CPU cycles. Fig. 3 shows the results of code profiling using 12 proteins randomly selected from the benchmarking dataset (1 protein from each bin in Table I). In Fig. 3(a), we measure the execution time for the three individual stages of *e*FindSite calculations, according to the flowchart presented in Fig. 1. Pre-processing, structure alignments, and post-processing steps take 11%, 88% and 1% of the total simulation time, respectively. Next, we use gprof, a performance analysis tool, to generate a function list ordered by computing time. Fig. 3(b) shows that four functions, *tmsearch_*, *cal_tmscore_*, *dp_* and *get_score_*, are the most time consuming taking up 29%, 27%, 21%, and 15% of the execution time, respectively. All these functions are involved in structure alignment calculations utilizing 92% of the entire computing time. Based on these profiling results, we identify template-to-target structure alignments as the most computationally expensive, thus parallelizing this portion of the code and moving calculations to the accelerator should bring about the desired performance improvement in predicting ligand-binding sites using *e*FindSite.

We also looked into the structure of the code for alignment calculations in *e*FindSite to estimate the difficulty of parallelization and porting it to the external accelerator card. Each individual template-to-target structure alignment starts by calling the subroutine *frtmalign* in the main function. Fig. 4 shows a detailed call graph generated by Doxygen [40] for functions associated with *frtmalign*. *Frtmalign* calls a number of subroutines, which in turn call other functions and so forth. The subroutine $u3b$ at the bottom, the most frequently called by subroutines at higher levels, performs root-mean-square deviation (RMSD) calculations for structure comparisons. We note that all protein structure alignment functions are written in Fortran77, consist of about 2100 lines of source code, and routinely access data stored in the memory using common blocks. Porting this code

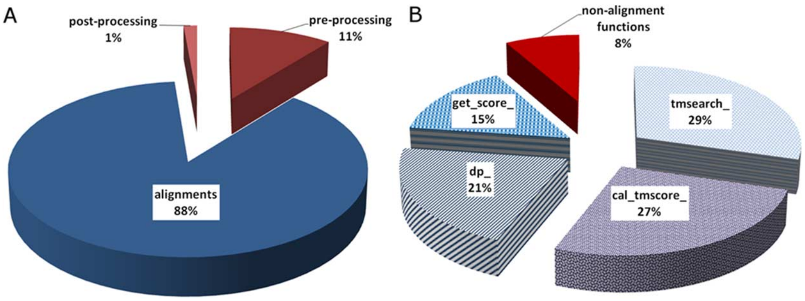


Fig. 3. Profiling of *e*FindSite. (A) Execution time is analyzed for the individual stages of *e*FindSite calculations. (B) Low level function usage reported by a performance analysis tool. Structure alignments and other functions are colored in patterned blue and red, respectively.

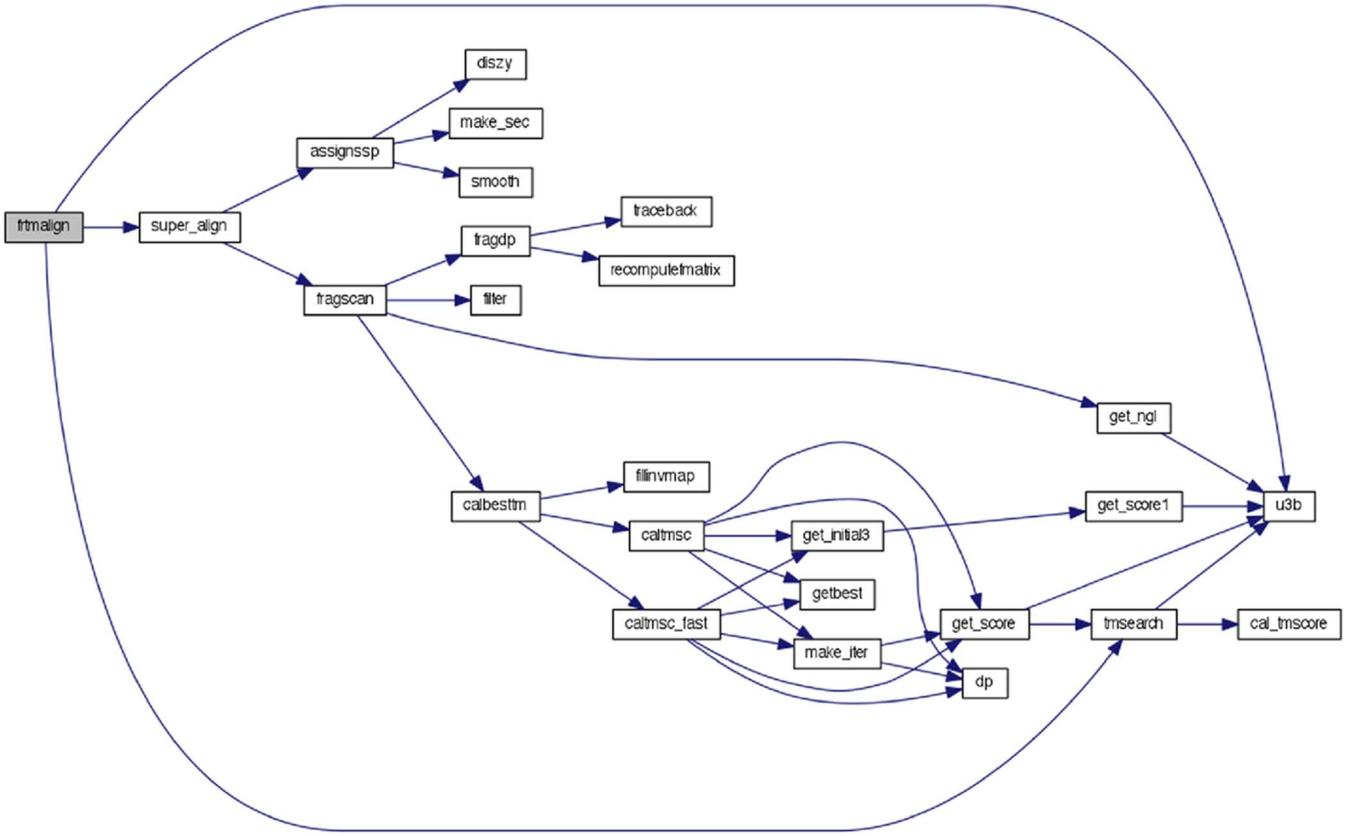


Fig. 4. Call graph of the protein structure alignment subroutine. The top subroutine, *frtmalign*, is called within the main function of *e*FindSite.

to the GPU platform using CUDA or OpenCL would require a great deal of effort; in contrast, using Intel Xeon Phi combined with pragma-based OpenMP requires significantly shorter development time to yield satisfactory speedups.

B. Performance Metrics

Before start analyzing the performance of *e*FindSite, we need to develop an evaluation metric for speed measurements. Intuitive metrics are the target protein size and the number of templates to be structurally aligned. Indeed, as demonstrated for 501 benchmarking proteins in Fig. 5(a), increasing both quantities results in a longer time required by *e*FindSite to predict ligand-binding sites. The execution time is measured for the total wall time (dark gray circles) and the time spent calculating structure alignments (light gray triangles). The corresponding Pearson correlation coefficients are 0.809 and 0.781, respectively. To search for a better performance evaluation metric, in Fig. 5(b), we plot the execution time against the number of single RMSD calculations realized by the subroutine $u3b$, which is the most frequently called function in the process of computing structure alignments. The correlation with the total wall time and the alignment time improves to 0.963 and 0.960, respectively. This analysis suggests that the number of RMSD calculations per second provides the best performance measure for evaluating the speed of *e*FindSite. We note that the target protein size and the number of templates are known *a priori*, before the simulations start, whereas the number of RMSD calculations can be obtained only after the simulations are completed because it depends on the convergence of structure alignments (templates that are structurally similar to the target converge faster than those less similar).

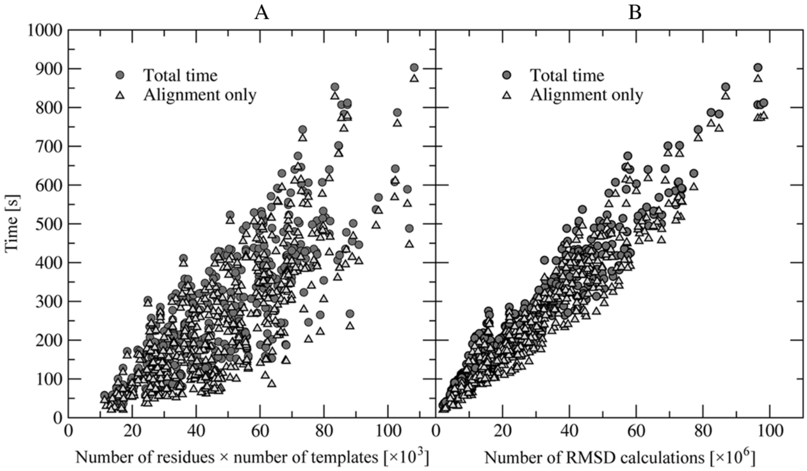


Fig. 5. Performance metrics for *e*FindSite. The execution time is plotted against (A) the product of the number of protein residues and the number of templates, and (B) the number of single RMSD calculations. Dark gray circles and light gray triangles show the total time and the time spent calculating structure alignments, respectively.

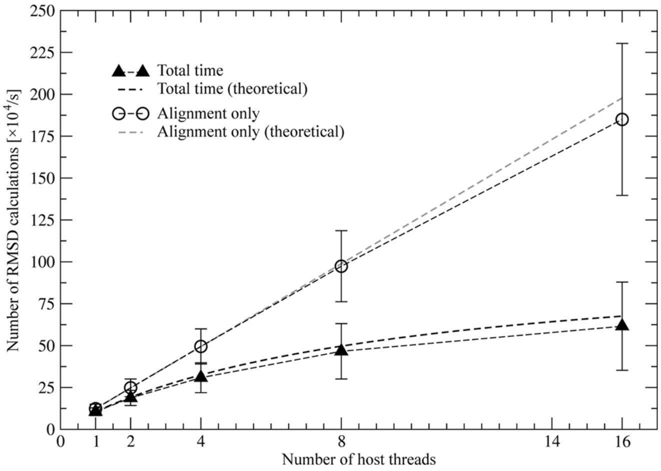


Fig. 6. Performance of *e*FindSite using parallel host threads. The speed is measured in RMSD calculations per second (mean ±standard deviation) for 2, 4, 8, and 16 threads across the benchmarking dataset of 501 proteins. Solid triangles and open circles correspond to the total time and the time spent calculating structure alignments, respectively. Thick dashed lines show the maximum theoretical performance according to Amdahl's law.

C. Performance of eFindSite on the Host

All benchmarking results for different versions of *e*FindSite were obtained using Stampede nodes at TACC. In Fig. 6, we compare serial and multi-threaded versions executed exclusively on the host. The latter uses pragma-based OpenMP parallelization of the *e*FindSite code; we note that both pre- and post-processing steps are executed sequentially, while only structure alignments are processed in parallel using multiple host threads. Increasing the number of threads for structure alignment calculations clearly improves the performance of *e*FindSite. Using the total simulation time, 1 (serial) and 16 (parallel) threads give an average performance across the benchmarking dataset of 1.06×10^5 and 6.16×10^5 RMSD calculations per second, respectively. Considering the time spent calculating structure alignments, the average performance is 1.24×10^5 and 1.85×10^6 RMSD calculations per second, respectively. For structure alignment calculations with 100% of the code parallelized, almost a perfect linear increase in the performance is achieved; the speedup of 16 host threads over the serial execution is 15.0. Contrastingly, the performance reaches a plateau with 5.8 speedup using 16 threads when the total wall time is considered. These results demonstrate that *e*FindSite follows Amdahl's law, which describes the relationship between the expected speedup of parallelized implementations of an algorithm relative to the serial algorithm [41]. According to the profiling results showing that about 90% of the calculations are parallelized, the maximum expected improvement using 16 threads is 6.4. With the increasing number of computing threads, serial portions of the code dominate, which is represented by the plateau in Fig. 6.

D. Performance of eFindSite on the Target

Next, we benchmark the performance of *e*FindSite using the Intel Xeon Phi coprocessor. We note that the pre- and post-processing steps are executed sequentially on the host, whereas only structure alignments are offloaded to the accelerator and processed in parallel using multiple target threads. Moreover, we measure the performance up to 24 threads because there is not enough parallelism to fully utilize all target threads for one protein target, considering that one thread processes one template and that some proteins in the benchmarking dataset have only 50 templates. Later on, to maximize the utilization of resources in production runs, we will launch multiple parallel tasks simultaneously processing multiple proteins. Specifically, 10 tasks, each processing one target protein using 24 threads, will be launched in parallel to fully utilize 240 hardware threads on the accelerator.

The Intel Xeon Phi coprocessor provides a large number of computing cores, however, individual target cores in our benchmarks are notably slower than those of the host. A single host thread performs 1.24×10^5 RMSD calculations per second, whereas a single target thread performs 1.19×10^4 (balanced and scatter modes) and 6.10×10^3 (compact mode) RMSD calculations per second. Thus, depending on the affinity settings, target threads are about 10 (balanced and scatter) and 20 (compact) times slower than host threads. We note that if adjacent threads share common data, placing them on the same execution unit by using the balanced thread affinity may bring some improvement over the scatter mode (Fig. 2). Nevertheless, individual threads in eFindSite are fully independent, therefore, the balanced and scatter thread affinities yield identical results; henceforth we report this performance as balanced/scatter.

Encouragingly, Fig. 7 shows that increasing the number of target threads leads to a linear scaling of the performance. For instance, increasing the number of threads from 4 to 24 with the affinity mode set to balanced/scatter improves the number of RMSD calculations per second from 3.98×10^4 to 1.76×10^5 for the total time, and from 4.27×10^4 to 2.30×10^5 for the alignment time. Switching to the compact thread affinity mode improves the number of RMSD calculations per second from 2.19×10^4 to 1.06×10^5 for the total time, and from 2.27×10^4 to 1.25×10^5 for the alignment time. It is clear that the scatter/balanced thread affinity yields approximately 1.8 times higher performance per thread than the compact mode. This is because both scatter and balanced modes evenly spread threads across the target computing cores to maximize the hardware utilization, whereas the compact affinity mode places up to four threads on a core before moving to the next one (Fig. 2). Thus, with the scatter/balanced setting, the best performance is achieved when 120 threads are spawned on the coprocessor, whereas 240 threads are required for the best performance in the compact mode. The advantage of using the scatter/balanced affinity starts diminishing when the number of threads is greater than 120. When the thread count reaches 240, different affinity settings yield the same performance corresponding to the compact mode. A core-to-core comparison shows 2.38×10^4 RMSD calculations per second for the scatter/balanced and 2.44×10^4 for the compact affinity. Therefore, assuming that all available hardware threads are fully utilized by eFindSite, offloading the structure alignment portion of the code using the compact thread affinity gives the best performance.

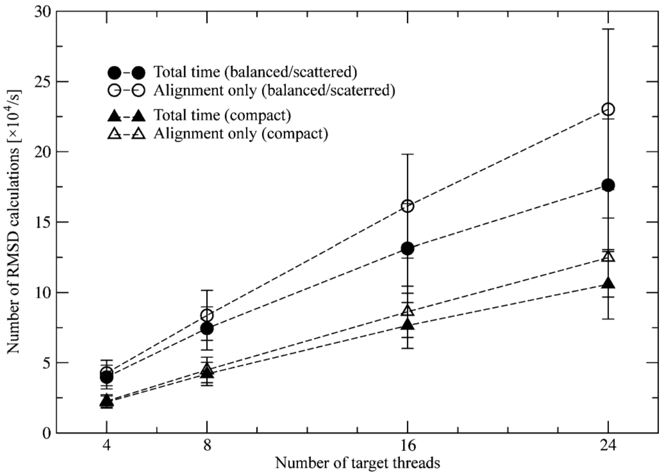


Fig. 7. Performance of eFindSite using parallel target threads. The speed is measured in RMSD calculations per second (mean ± standard deviation) for 4, 8, 16, and 24 threads across the benchmarking dataset of 501 proteins. Solid and open symbols correspond to the total time and the time spent calculating structure alignments, respectively. Circles and triangles show the performance using scatter/balanced and compact thread affinity, respectively.

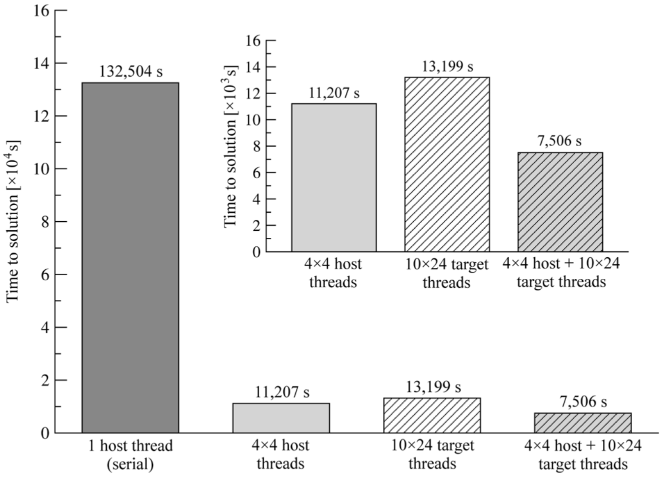


Fig. 8. Time-to-solution for pocket prediction across the entire benchmarking dataset. Computations are performed using a serial version (1 host thread) as well as three parallel versions: multiple host threads (4 tasks, each running 4 threads), multiple target threads (10 tasks, each running 24 threads), and multiple host and target threads running simultaneously. Inset shows the comparison of parallel versions only.

E. Performance of eFindSite on Both Host and Target

The ultimate goal of this study is to take full advantage of the entire node equipped with the Intel Xeon Phi card, $viz.$ process the calculations using both the host and the target simultaneously. Performance benchmarks carried out independently for the host and the target reveal several important results and constraints related to software, hardware and dataset characteristics. These should be taken into consideration when designing an optimized production code. First, because of Amdahl's law, it is beneficial to execute multiple parallel versions of eFindSite on the host, each using relatively few threads. Moreover, we demonstrated that using all available target threads with the compact affinity maximizes the performance of eFindSite on the coprocessor. Therefore, our production scheme for predicting ligand-binding sites using eFindSite across large datasets of proteins comprises multiple tasks running simultaneously that utilize both the host and the target. Specifically, we further modified the code to launch up to 4 parallel tasks on the host, each using 4 threads, and up to 10 parallel tasks on the target, each using 24 threads.

Fig. 8 compares the time-to-solution, defined as the total time required to predict ligand-binding sites for the entire dataset of 501 proteins, for the serial version as well as three parallel processing schemes. The time-to-solution for serial eFindSite, parallel eFindSite on the host only, the target only, and both the host and the target is 36.8, 3.1, 3.6, and 2.1 hours, respectively. Fig. 9 shows that this corresponds to the speedup over the serial version of 11.8, 10.1, and 17.6 for the parallel processing using the host, the target, and both resources, respectively. Therefore, using the coprocessor in addition to the main processor provides the desired acceleration in predicting binding sites for large protein datasets. It is noteworthy that at this point, the time-to-solution increases linearly with the dataset size, i.e., ligand-binding site prediction using eFindSite for 5000 proteins, which is a typical size of a bacterial proteome, would require 21 hours on a single Stampede node equipped with the Intel Xeon Phi card, compared to 31 hours without the accelerator and 368 hours using the serial version.

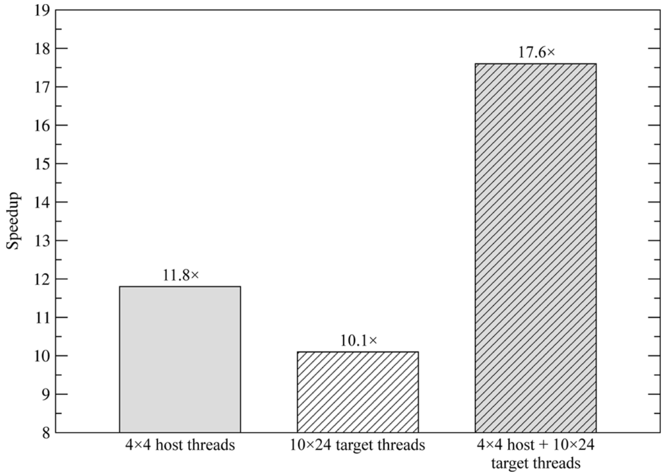


Fig. 9. Speedups of the parallel versions of *e*FindSite over the serial code. Computations are performed using three parallel versions: multiple host threads (4 tasks, each running 4 threads), multiple target threads (10 tasks, each running 24 threads), and multiple host and target threads running simultaneously.

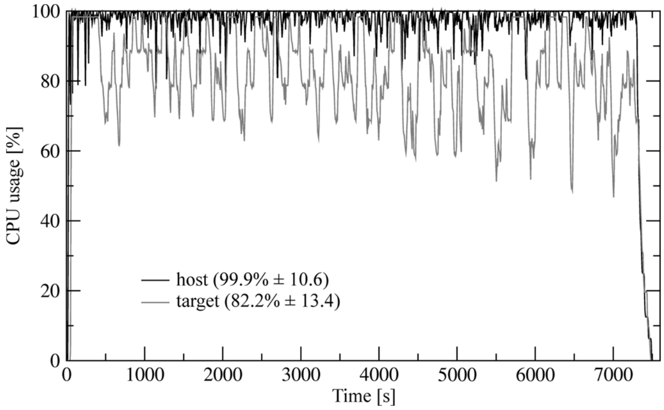


Fig. 10. Utilization of resources during the execution of the heterogeneous parallel version of *e*FindSite. Time courses of the percentage of CPU usage on the host (black line) are compared to that on the target (gray line).

In addition to the time-to-solution, we also monitor the utilization of computing resources. Fig. 10 shows that the average usage of host and target cores during the production multi-task/multi-threaded simulation is 99.9% and 82.2%, respectively. Target threads periodically become idle while waiting for the host to finish pre- and post-processing steps, which results in a somewhat lower utilization of the coprocessor. On the other hand, the host remains fully utilized not only facilitating multiple tasks offloaded to the target, but also performing *e*FindSite calculations on its own. Looking at the performance results presented in Figs. 8 and 9, the computational capabilities of 240 coprocessor threads are fairly comparable to those of 16 host threads, thus we can expect an even partitioning of the computations between the host and the target when processing a large dataset. Table II shows that this is indeed the case; each parallel 4-thread task executed on the host performed about 12% of the total computations, whereas each parallel 24-thread task executed on the target performed about 5% of the total computations. Adding up all computations performed by the host and the target gives a roughly even split of 49.2% and 50.8%, respectively. Therefore, dividing the workload evenly between the target and the host will ensure the optimal utilization of computing resources in large production runs.

F. Accuracy of Pocket Prediction

Thus far, we demonstrated that the parallel versions of *e*FindSite, including that accelerated by the Intel Xeon Phi coprocessor, offer an improved performance, however, the results must also be technically correct. Fig. 11 assesses the accuracy of pocket prediction by the code described in this paper, labeled as 1.2, compared to the original version of *e*FindSite [19], labeled as 1.0. All three 1.2 versions, serial, parallel on the host, and parallel on the target, produce identical results demonstrating that the implementation of *e*FindSite using OpenMP and the accelerator maintains the functionality of the code providing shorter simulation times. Version 1.2 is slightly more accurate than 1.0 because a couple of coding bugs in 1.0 have been identified and fixed during the porting process. For instance, for 65.7% and 84.3% of benchmarking proteins the best of top three ligand-binding sites is predicted within 4 Å and 8 Å, respectively. This high prediction accuracy is accompanied by high ranking capabilities; the best binding site is at rank 1 in 69.0% of the cases. We note that closely related templates with >40% sequence identity to the target are excluded in our benchmarking calculation, thus quantitatively similar results can be expected for real applications of *e*FindSite in function annotation at the level of entire proteomes.

G. Case Study

To conclude, we present a case study illustrating binding pocket prediction using *e*FindSite. Our target selected from the benchmarking dataset is penicillin-binding protein from *Pseudomonas aeruginosa*, PBP3 (PDB-ID: 3pbr, chain A), a critical enzyme responsible for peptidoglycan synthesis [42]. *e*FindSite predicted the total number of 22 pockets for this protein and assigned a confidence of 91.1% to the top-ranked binding site. Fig. 12 shows the crystal structure of the target protein with the top three binding pockets represented by balls; the yellow ball corresponds to the top-ranked pocket, whereas the two red balls show the location of pockets at ranks 2 and 3. In addition to the pocket center, orange sticks and the transparent yellow surface depict residues predicted to bind a ligand. The structure displayed in Fig. 12 also contains a β-lactam antibiotic bound to this enzyme; we note that this compound is used only to assess the prediction accuracy and it was not included in *e*FindSite simulations. The distance between the predicted top-ranked binding site and the geometric center of the antibiotic is only 2.4 Å with as many as 63% binding residues correctly identified. This case study illustrates that *e*FindSite is a reliable tool for ligand-binding prediction, which has a broad range of applications in protein function annotation, virtual screening and drug discovery.

IV. Conclusions

In this study, we developed a new version of *e*FindSite, ligand-binding site prediction software used in structural biology and drug design, for processing large datasets using modern heterogeneous high-performance systems, i.e., multicore processor platforms equipped with Intel Xeon Phi

TABLE II
PARTITIONING DETAILS FOR PROCESSING 501 BENCHMARKING PROTEINS USING *e*FINDSITE SIMULTANEOUSLY ON THE HOST AND THE TARGET

Hardware	Core range	Number of threads	Percentage of computations[a]
Host	1-4	4	12.2%
	5-6	4	12.5%
	7-11	4	12.7%
	12-16	4	11.8%
Target	1-6	24	5.1%
	7-12	24	5.1%
	13-18	24	5.1%
	19-24	24	5.0%
	25-30	24	5.3%
	31-36	24	5.0%
	37-42	24	4.7%
	43-48	24	5.0%
	49-54	24	5.4%
	55-60	24	5.1%
Host	1-16	16	49.2%
Target	1-60	240	50.8%

[a] The amount of computations is approximated by the product of the target protein length and the number of template structures.

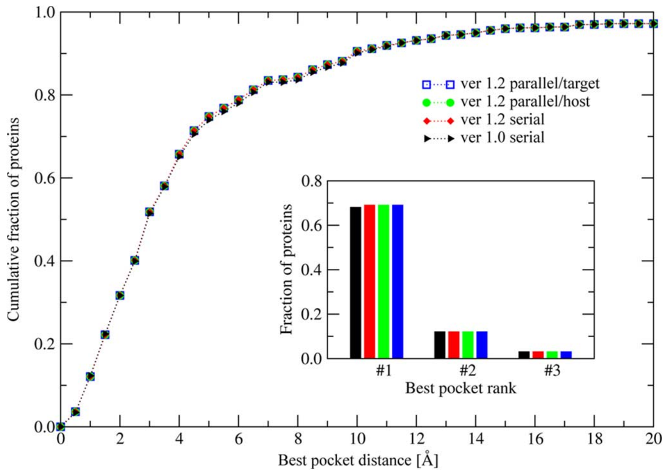


Fig. 11. Evaluation of the accuracy of binding site prediction using different versions of *e*FindSite. The cumulative fraction of proteins is plotted against the distance between the center of the best top three predicted pockets and the geometric center of bound ligand in the native complex structure. Inset shows the accuracy of pocket ranking when multiple pockets are detected.

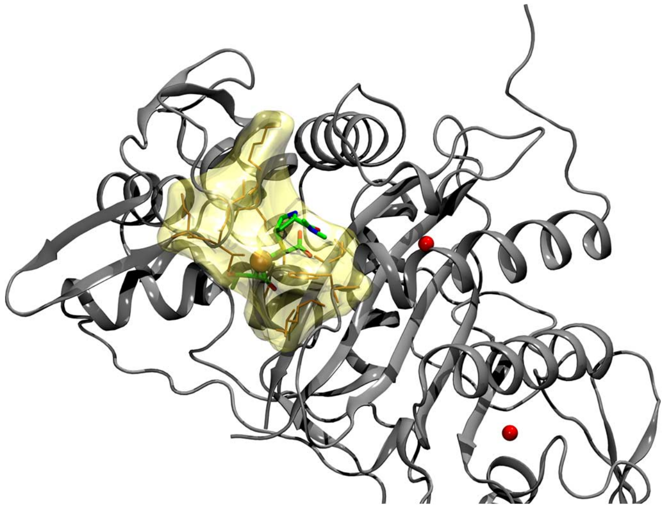


Fig. 12. Pocket prediction for a penicillin-binding protein from *P. aeruginosa* using *e*FindSite. The crystal structure of the target protein and the binding ligand are displayed as a gray cartoon and sticks colored by atom type, respectively. The top-ranked predicted pocket is shown in yellow; the ball corresponds to the pocket center and the predicted binding residues are shown as sticks and a transparent surface. Two red balls show the centers of binding pockets predicted at ranks 2 and 3.

accelerators. There are two major conclusions arising from this project. First, offloading parts of the computations to the coprocessor device indeed provides the desired speedups, which are considered significant when normalized by the cost of the hardware. Compared to a serial version, *e*FindSite code runs 11.8, 10.1, and 17.6 times faster using the parallel processing on the host, the target, and both resources, respectively. Of course, the acceleration strongly depends on the nature of the code. For example, the benchmarking of compute-intensive applications in finances reported speedups of 20-–40 for a Xeon Phi code over a sequential reference implementation running on a single processor core [43]. Another study systematically compared a number of applications for microscopy image analysis and showed speedups ranging from 5 to 32 on Intel Xeon Phi accelerators [44]. Thus, our results are fairly comparable to those reported by other developers.

Second, programming models available for the Intel Xeon Phi architecture are open-standard and portable between traditional processors and coprocessors; moreover, OpenMP makes it relatively easy to execute programs in parallel. *e*FindSite represents a piece of typical scientific software written mostly by domain scientists, who contributed different components to the code using different programming languages and styles. With minimal modifications, a complex, hybrid C++/Fortran77 code was successfully ported to the coprocessor yielding satisfactory speedups. This illustrates that the process of modifying scientific software to benefit from the Intel Xeon Phi architecture is relatively straightforward with quite short development times.

Nonetheless, since the coprocessor features wide SIMD vector instructions, a proper loop vectorization is particularly important for its full utilization. Propitiously, vectorization reports collected for *e*FindSite show that the majority of loops taking the most execution time are indeed vectorized. Still, there are other issues related to data dependency and alignment, which need to be addressed by rearranging loops, data structure padding, improving register utilization, and data caching. Therefore, in addition to the parallelization of the remaining portions of the serial code, future directions of this project include thorough code optimizations to take a better advantage of the Intel Xeon and Xeon Phi architectures. The up-to-date versions of *e*FindSite are freely available to the academic community at www.brylinski.org/efindsite; this website also provides compilation and installation instructions, as well as a detailed tutorial on processing large datasets using heterogeneous computing platforms.

Acknowledgment

The authors are grateful for discussions and comments from the members of the Heterogeneous Computing group formed within the Louisiana Alliance for Simulation-Guided Materials Applications (LA-SiGMA).

References

[1] M. L. Metzker, “Sequencing technologies—the next generation,” *Nat. Rev. Genet.*, vol. 11, no. 1, pp. 31–46, 2010.

[2] J. Zhao and S. F. Grant, “Advances in whole genome sequencing technology,” *Curr. Pharm. Biotechnol.*, vol. 12, no. 2, pp. 293–305, 2011.

[3] A. S. Juncker, L. J. Jensen, A. Pierleoni, A. Bernsel, M. L. Tress, P. Bork, G. von Heijne, A. Valencia, C. A. Ouzounis, R. Casadio, and S. Brunak, “Sequence-based feature prediction and annotation of proteins,” *Genome Biol.*, vol. 10, no. 2, p. 206, 2009.

[4] Y. Loewenstein, D. Raimondo, O. C. Redfern, J. Watson, D. Frishman, M. Linial, C. Orengo, J. Thornton, and A. Tramontano, “Protein function annotation by homology-based inference,” *Genome Biol.*, vol. 10, no. 2, p. 207, 2009.

[5] H. Kitano, “Systems Biology: A brief overview,” *Science*, vol. 295, no. 5560, pp. 1662–1664, 2002.

[6] A. D. Boran and R. Iyengar, “Systems approaches to polypharmacology and drug discovery,” *Curr. Opin. Drug Discov. Devel.*, vol. 13, no. 3, pp. 297–309, 2010.

[7] NVIDIA, GPU Accelerated Applications [Online]. Available: http://www.nvidia.com/object/gpu-accelerated-applications.html

[8] J. E. Stone, J. C. Phillips, P. L. Freddolino, D. J. Hardy, L. G. Trabuco, and K. Schulten, “Accelerating molecular modeling applications with graphics processors,” *J. Comput. Chem.*, vol. 28, no. 16, pp. 2618–2640, 2007.

[9] M. Januszewski and M. Kostur, “Accelerating numerical solution of stochastic differential equations with CUDA,” *Comp. Phys. Commun.*, vol. 181, no. 1, pp. 183–188, 2010.

[10] M. Weigel, “Simulating spin models on GPU,” *Comp. Phys. Commun.*, vol. 182, no. 9, pp. 1833–1836, 2011.

[11] E. Roberts, J. E. Stone, L. Sepulveda, W. M. W. Hwu, and Z. Luthey-Schulten, “Long time-scale simulations of *in vivo* diffusion using GPU hardware,” pp. 1–8, 2009.

[12] J. Bao, H. Xia, J. Zhou, X. Liu, and G. Wang, “Efficient implementation of MrBayes on multi-GPU,” *Mol. Biol. Evol.*, vol. 30, no. 6, pp. 1471–1479, 2013.

[13] P. D. Vouzis and N. V. Sahinidis, “GPU-BLAST: Using graphics processors to accelerate protein sequence alignment,” *Bioinformatics*, vol. 27, no. 2, pp. 182–188, 2011.

[14] B. Pang, N. Zhao, M. Becchi, D. Korkin, and C. R. Shyu, “Accelerating large-scale protein structure alignments with graphics processing units,” *BMC Res. Notes*, vol. 5, p. 116, 2012.

[15] [Online]. Available: http://top500.org

[16] T. Cramer, D. Schmidl, M. Klemm, and D. Mey, “OpenMP programming on Intel Xeon Phi Coprocessors: An early performance comparison” [Online]. Available: https://sharepoint.campus.rwth-aachen.de/units/rz/HPC/public/Lists/Publications/Attachments/86/2012_MARC_Xeon_Phi.pdf, 2012

[17] C. Rosales, “Porting to the Intel Xeon Phi: Opportunities and challenges” [Online]. Available: http://www.xsede.org/documents/271087/586927/CRosales_TACC_porting_mic.pdf 2012

[18] A. Vladimirov and V. Karpusenko, “Heterogeneous clustering with homogeneous code: accelerate MPI applications without code surgery using Intel Xeon Phi coprocessors,” Colfax Int. 2013 [Online]. Available: http://research.colfaxinternational.com/post/2013/10/17/Heterogeneous-Clustering.aspx

[19] M. Brylinski and W. P. Feinstein, “eFindSite: Improved prediction of ligand binding sites in protein models using meta-threading, machine learning and auxiliary ligands,” *J. Comput. Aided Mol. Des.*, vol. 27, no. 6, pp. 551–567, 2013.

[20] W. P. Feinstein and M. Brylinski, “eFindSite: Enhanced fingerprint-based virtual screening against predicted ligand binding sites in protein models,” *Mol. Inf.*, vol. 33, no. 2, pp. 135–150, 2014.

[21] B. Rost, “Twilight zone of protein sequence alignments,” *Protein Eng.*, vol. 12, no. 2, pp. 85–94, Feb 1999.

[22] J. Skolnick and M. Brylinski, “FINDSITE: A combined evolution/structure-based approach to protein function prediction,” *Brief Bioinf.*, vol. 10, no. 4, pp. 378–391, 2009.

[23] H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne, “The protein data bank,” *Nucleic Acids Res.*, vol. 28, no. 1, pp. 235–242, 2000.

[24] M. Brylinski and D. Lingam, “eThread: A highly optimized machine learning-based approach to meta-threading and the modeling of protein tertiary structures,” *PLoS One*, vol. 7, no. 11, p. e50200, 2012.

[25] M. Brylinski, “Unleashing the power of meta-threading for evolution/structure-based function inference of proteins,” *Front Genet.*, vol. 4, p. 118, 2013.

[26] M. Brylinski, “Nonlinear scoring functions for similarity-based ligand docking and binding affinity prediction,” *J. Chem. Inf. Model*, vol. 53, no. 11, pp. 3097–3112, 2013.

[27] M. Brylinski and J. Skolnick, “Q-Dock(LHM): Low-resolution refinement for ligand comparative modeling,” *J Comput. Chem.*, vol. 31, no. 5, pp. 1093–1105, 2010.

[28] S. B. Pandit, M. Brylinski, H. Zhou, M. Gao, A. K. Arakaki, and J. Skolnick, “PSiFR: An integrated resource for prediction of protein structure and function,” *Bioinformatics*, vol. 26, no. 5, pp. 687–688, 2010.

[29] P. Aloy and R. B. Russell, “Structural systems biology: Modelling protein interactions,” *Nat. Rev. Mol. Cell Biol.*, vol. 7, no. 3, pp. 188–197, 2006.

[30] *Intel® Xeon Phi™ Coprocessor Instruction Set Architecture Reference Manual*, Intel Corp., 2012 [Online]. Available: https://software.intel.com/sites/default/files/forum/278102/327364001en.pdf

[31] “Stampede user guide” [Online]. Available: https://portal.tacc.utexas.edu/user-guides/stampede

[32] “OpenMP specifications” [Online]. Available: http://openmp.org/wp/openmp-specifications/

[33] “MPI: A message-passing interface standard version 3.0”, MPI Forum, 1997.

[34] S. Leis, S. Schneider, and M. Zacharias, “In silico prediction of binding sites on proteins,” *Curr. Med. Chem.*, vol. 17, no. 15, pp. 1550–1562, 2010.

[35] M. Brylinski and J. Skolnick, “A threading-based method (FINDSITE) for ligand-binding site prediction and functional annotation,” *Proc. Natl. Acad. Sci. USA*, vol. 105, no. 1, pp. 129–134, 2008.

[36] S. B. Pandit and J. Skolnick, “Fr-TM-align: A new protein structural alignment method based on fragment alignments and the TM-score,” *BMC Bioinformat.*, vol. 9, p. 531, 2008.

[37] B. J. Frey and D. Dueck, “Clustering by passing messages between data points,” *Science*, vol. 315, no. 5814, pp. 972–976, 2007.

[38] J. Reinders and J. Jeffers, *High Performance Parallelism Pearls: Multicore and Many-Core Programming Approaches*. San Francisco, CA, USA: Morgan Kaufmann, 2014.

[39] M. Gen and R. Cheng, *Genetic Algorithms and Engineering Optimization*. New York: Wiley, 2002.

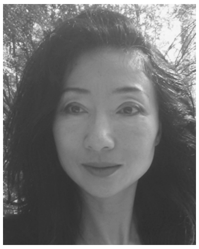
[40] D. van Heesch, “Announcing: The first release of Doxygen, a C + + documentation system,” 1997.

[41] G. M. Amdahl, “Validity of the single processor approach to achieving large-scale computing capabilities,” in *AFIPS Conf. Proc.*, 1967, vol. 30, pp. 483–485.

[42] S. Han, R. P. Zaniewski, E. S. Marr, B. M. Lacey, A. P. Tomaras, A. Evdokimov, J. R. Miller, and V. Shanmugasundaram, “Structural basis for effectiveness of siderophore-conjugated monocarbams against clinically relevant strains of Pseudomonas aeruginosa,” *Proc. Natl. Acad. Sci. USA*, vol. 107, no. 51, pp. 22002–22007, 2010.

[43] “Benchmarks: Intel Xeon Phi vs. NVIDIA Tesla GPU” [Online]. Available: http://blog.xcelerit.com/intel-xeon-phi-vs-nvidia-tesla-gpu/

[44] G. Teodoro, T. Kurc, J. Kong, L. Cooper, and J. Saltz, “Comparative performance analysis of Intel Xeon Phi, GPU, CPU: A case study from microscopy image analysis,” *IEEE Trans Parallel Distrib. Syst.*, vol. 2014, pp. 1063–1072, May 2014.



Wei P. Feinstein received her M.Sc. degree in computer science from University of Alabama, Tuscaloosa, AL, USA, and the Ph.D. degree in medical sciences from the College of Medicine at the University of South Alabama, Mobile, AL, USA. She is a Postdoctoral Researcher in the Center for Computation & Technology, or CCT, at Louisiana State University, Baton Rouge, LA, USA. Her research interests are in developing algorithms for computational biophysics with applications in drug discovery and biomaterial research. In addition, she develops high-performance parallel and heterogeneous codes to boost the performance of applications in biology, physics, and chemistry.

Juana Moreno received her B.S. degree in physics from the Universidad Autonoma, Madrid, Spain, and the Ph.D. degree in condensed matter physics from Rutgers University, New Brunswick, NJ, USA. She is an associate professor in the Department of Physics and Astronomy and CCT at Louisiana State University, Baton Rouge, LA, USA. Her research focuses on modeling the transport and magnetic properties of correlated electron systems, including diluted magnetic semiconductors, heavy fermion compounds, and low-dimensional systems, using a variety of computational tools, such as the dynamical mean-field theory and the dynamical cluster approximation.

Mark Jarrell received his Ph.D. degree in physics from the University of California at Santa Barbara, CA, USA. He is a Professor in the Department of Physics and Astronomy and the Material World focus area lead in CCT at Louisiana State University, Baton Rouge, LA, USA. His interests lie in the physics of strongly correlated electronic materials, which include many nanostructures, high-Tc superconductors, heavy fermion, and magnetic materials. He is also actively involved in the design and development of multidisciplinary codes to address the problem of efficient scaling on the next generation of hyper-parallel and heterogeneous machines.

Michal Brylinski received his M.Sc. degree in pharmacy from Wroclaw Medical University, Poland, and the Ph.D. degree in chemistry from Jagiellonian University, Poland. He is an Assistant Professor in the Department of Biological Sciences and CCT, Louisiana State University, Baton Rouge, LA, USA. His work involves the design and development of novel algorithms and codes for drug discovery and repositioning using systems-level approaches. He is interested in the application of massively parallel hardware accelerators in structural bioinformatics, functional genomics, cheminformatics, and pharmacogenomics.