Hardware Accelerators in Computational Biology: Application, Potential, and Challenges

Turbo Majumder
Indian Institute of Technology Delhi

Partha Pratim Pande and Ananth Kalyanaraman
Washington State University

Editor's notes:
Computational biology is increasingly relying on hardware accelerators to allow data processing to keep up with the increasing amount of data generated from biology applications. This paper gives an introduction to the area of hardware accelerators for computational biology and a comparative study of a set of biological applications.

—Jan Madsen, Technical University of Denmark

■ **COMPUTING RESEARCH HAS** become a vital cog in the machinery required to drive biological discovery. Unearthing and analyzing large amounts of biological data involve exploration of a variety of computational paradigms, collectively referred to as computational biology. Most of the data processing for computational biology applications is currently done in software, which results in a high turnaround time. For example, aligning even hundreds of genome sequences using progressive alignment tools such as ClustalW requires several hours on state-of-the-art workstations. With the increased throughput demands and popularity of such computational biology tools, reducing time-to-solution during computational analysis has become a significant challenge in the path to scientific discovery. These aspects collectively position computational biology as a domain that has the potential to immensely benefit from the latest advances in the computing and architecture community. The products of research in this field, functioning as hardware accelerators, act as technological enablers in order to fast-track the process of scientific discovery.

A wide range of such hardware accelerator platforms has been proposed in recent work. Among these, FPGA-based reconfigurable hardware platforms, graphics processing unit (GPU), cell broadband engine (CBE), and multicore processors are notable. Each of these platforms has advantages and limitations. The principal advantages of using FPGA-, GPU-, or CBE-based systems are fast prototyping and ease of implementation. These systems use an existing hardware platform to map algorithms and their software implementations. On the other hand, the massive scale of fine-grain parallelism inherent in several computational biology applications can be exploited efficiently in a multicore platform by integrating a large number of processing elements on a single chip. However, rapid prototyping with such multicore processors is still not mature. Figure 1 summarizes the current state of the art in both the architecture space and the computational biology application space. The rest of the paper explores existing solutions and open challenges over a wide range of computational biology applications depicted in Figure 1, viz., sequence analysis, phylogenetics,

Digital Object Identifier 10.1109/MDAT.2013.2290118
Date of publication: 07 November 2013; date of current version: 20 February 2014.

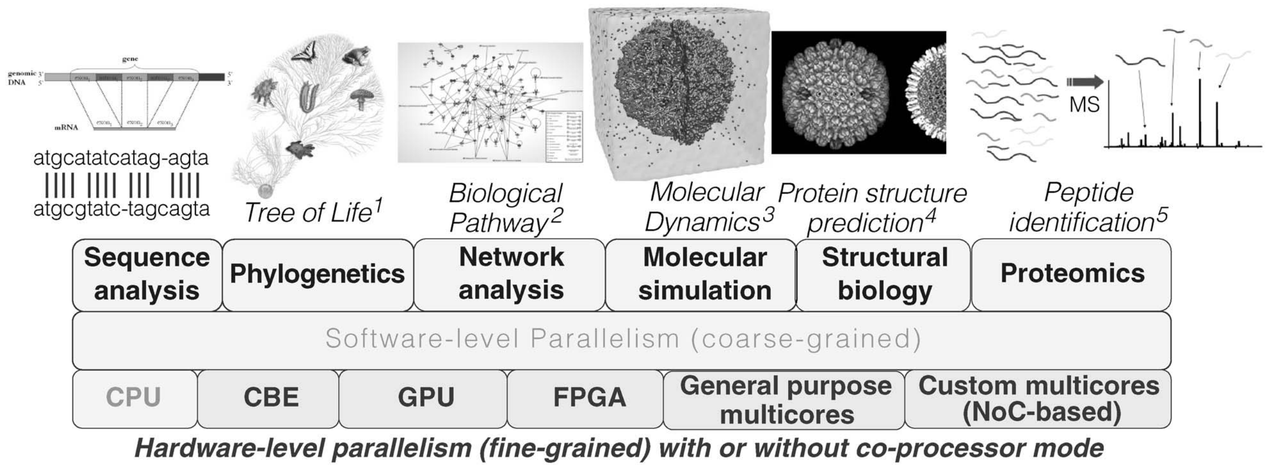
atgcatatcatag-agta
atgcgtatc-tagcagta
Tree of Life[1]
Biological Pathway[2]
Molecular Dynamics[3]
Protein structure prediction[4]
Peptide identification[5]
MS
Sequence analysis
Phylogenetics
Network analysis
Molecular simulation
Structural biology
Proteomics
Software-level Parallelism (coarse-grained)
CPU
CBE
GPU
FPGA
General purpose multicores
Custom multicores (NoC-based)
Hardware-level parallelism (fine-grained) with or without co-processor mode

Image credits:

[1]Tree of life Web Project: http://www.tolweb.org/tree/

[2]Moser JJ, Fritzler MJ (2010) The MicroRNA and MessengerRNA Profile of the RNA-Induced Silencing Complex in Human Primary Astrocyte and Astrocytoma Cells. PLoS ONE 5(10): e13445.

[3]Theoretical and computational Biophysics Group, UIUC, http://www.ks.uiuc.edu/Research/STMV/

[4]Structural biology http://www.niams.nih.gov/research/ongoing_research/branch_lab/structural_biology/

[5]Xue Wu, Nathan Edwards, Chau-Wen Tseng, Peptide Identification via Tandem Mass Spectrometry, In: Chau-Wen Tseng, Editor(s), Advances in Computers, Elsevier, 2006, Volume 68, Pages 253-278.

Figure 1. Illustration of a representative subset of biocomputing applications alongside state-of-the-art hardware accelerator platforms considered in this tutorial.

molecular-simulation-based methods, biological network analysis, structural biology, and others. We conclude by setting out broad research directions.

Sequence analysis

Sequence homology detection (or sequence alignment) is a pervasive compute operation carried out in almost all bioinformatics sequence analysis (SA) applications. With exponentially growing sequence databases, large-scale computing of this operation is becoming prohibitive. The operation can be carried out in three modes: as one-to-one, one-to-many, or many-to-many comparisons. The one-to-one alignment operation, called the pairwise sequence alignment (PSA), is used to compute an optimal edit distance between two sequences, after taking into account evolutionary manifestations of mutation such as substitution, insertion, and deletion. In the one-to-many comparison model, a query sequence is searched against a database of sequences. In the many-to-many comparison model, multiple sequences are analyzed collectively for the purpose of identifying subgroups of sequences that share a common characteristic such as homology. This operation is often implemented using an all-against-all sequence comparison strategy. Multiple sequence alignment (MSA) is an example of this class. In all of the above, a sequence can be either a DNA or a protein, and the bulk of the computations involve integer arithmetic.

Algorithmically, computing an optimal PSA between two sequences of lengths m and n, respectively, can be achieved using dynamic programming (DP) in $O(mn)$ time and $O(m+n)$ space [1]. The algorithm computes an $(m+1)*(n+1)$ table in two passes. In the forward pass, the table is computed from cell $(0,0)$ to cell (m,n), wherein a recurrence is applied at every cell (i,j) based on the values at cells $(i-1,j)$, $(i-1,j-1)$, and $(i,j-1)$. The main challenge in the backward pass is to be able to retrace without storing the DP table that was computed during the forward pass.

Hardware accelerators using FPGA have been developed for implementing ClustalW [2], which is a popular MSA program. Since the underlying problem is NP-hard, ClustalW approximates a solution in polynomial time. A K-sequence MSA problem involves computing ${}^{K}C_2$ PSA comparisons. This all-against-all sequence comparison is the dominant phase within ClustalW, taking more than 90% of the total time. The FPGA implementation in [2] uses Xilinx Virtex II XC2V6000, platform accommodating 92 processing elements (PEs) with a maximum clock speed of 34 MHz. This gives a speedup of around 10× for the overall MSA and about 50× for PSA. It achieves a sustained performance (including all data transfer) of ∼1 GCUPS (billion cell updates per second in the DP matrix).

The sequence search tool BLAST proceeds by first identifying a subset of database sequences that have short matching segments with the query sequence and then performing a more thorough evaluation of the query against each short-listed candidate. The filtering step is implemented using a lookup table data structure, and a subsequent evaluation as a unit PSA. An FPGA implementation of BLAST [3] on Annapolis Microsystems WildstarII-Pro board with two Xilinx Virtex-II FPGAs evaluates two algorithms: TREE BLAST and SERVER BLAST. TREE BLAST is based on iterative merging using a tree structure. The FPGA is initialized with the query sequence and the DP (scoring) matrix. The indexing of the scoring arrays is done using the block RAMs (BRAMs). The database is streamed from the memory through the FPGA. The main component of SERVER BLAST is a systolic array that holds a query string while the database flows through it. This is implemented using a FIFO buffer in FPGA. The performance reported was comparable to that of a dedicated server at the National Center for Biotechnology Information, U.S. National Library of Medicine, Bethesda, MD, USA. Sachdeva et al. [4] demonstrated acceleration of BLAST on CBE. The system consisted of a 64-bit power processor element (PPE) and eight synergistic processing elements (SPEs). They achieved a speedup of 2× compared to that implemented on a single power PC processor. Liu et al. [5] demonstrate about 16× speedup over an MSA tool, OSEARCH, using nVidia GeForce 7800 GTX GPU. They map the algorithm onto GPU by exploiting the fact that all elements in the same antidiagonal of the DP matrix can be computed independent of each other in parallel. They have reformulated the Smith–Waterman algorithm [1] in terms of computer graphics primitives, in an attempt to exploit the GPU platform for optimum performance. An enhanced version of the Smith–Waterman protein database search algorithm, CUDASW++ 3.0, is presented in [6], which provides up to 186 GCUPS performance on a dual-GPU GeForce GTX 690 graphics card.

Benkrid et al. [7] present a detailed design and implementation of a generic and highly parameterized FPGA skeleton for PSA using a high-level language Handel-C, which is independent of the underlying FPGA platform hardware architecture. They evaluated their platform for different algorithms (e.g., [1]) and reported peak performance of over 10 GCUPS, which represented two orders of magnitude improvement over software. Taking into account host-to-FPGA communication overheads on an Alpha Data XP FPGA Mezzanine PCI-board, which has an XC2VP100-6 Virtex-II Pro FPGA, they implemented a 135-PE system that shrank time-to-solution by 62× relative to software. An effective way to deliver even higher values of speedup is to integrate a large number of PEs using a network-on-chip (NoC). Using this approach for a hardware accelerator for PSA, Sarkar et al. [8] report significant improvement over other hardware accelerators owing to the custom-made architecture and interconnection topology. Even with 64 PEs in such a system, they achieved two to three orders of magnitude better performance compared to other existing hardware accelerators. NoCs also provide the freedom to design and experiment with different network topologies and their suitability to different algorithmic settings.

Another related application is read mapping, which involves mapping shorter DNA fragments through alignment against a known reference genome. This is necessitated by the fact that sequencing tools take small fragments of whole genomes as inputs. The problem is compounded by presence of genomic repeats, nucleotide variations, and sequencing errors. So far, read mapping has been the focus of hardware acceleration initiatives that are mostly based on FPGA. For example, in [9], Olson et al. propose an FPGA-based solution that demonstrates 31× speedup over the Bowtie software tool and aligns a higher percentage (91%) of short reads compared to software (80%) while consuming less than 5% of the energy consumed by the general-purpose microprocessor running the software.

Table 1 Overview of speedups achieved through hardware acceleration for different sequence analysis applications [2]–[10]. The "–" entries correspond to data not available.

Type of SA	Speedup over Serial Implementation			
	FPGA	*GPU*	*CBE*	***Multicore (NoC)***
PSA	~100	70	6	**22,000**
MSA	13	7	2	**-**
BLAST	-	16	2	**-**
Read mapping	70	-	-	**-**
Repeat identification	28,255	-	-	**-**
Sequence motif discovery	103	-	-	**-**

In a related problem, where the purpose is to characterize the variations within tandem repeats in large genomes, acceleration by over three orders of magnitude has been achieved by building a systolic array on an FPGA platform in [10]. Another application of the FPGA platform has been on accelerating sequence-to-profile hidden Markov model (HMM) matching [11]. Table 1 summarizes the performance of hardware accelerators for different sequence analysis applications.

As the discussion above reveals, we have several flavors of problems within the sequence analysis domain. At the basic level is the comparison of two sequences, and this task maps well to a wide range of accelerating platforms. More complex functions such as genome assembly and read mapping could also benefit from hardware acceleration as they generally entail computation of a large volume (oftentimes, billions) of individual pairwise comparisons. As Table 1 shows, while accelerators based on FPGAs and GPUs have been delivering decent speedup, NoC-based multicores perform orders of magnitude better by being able to integrate a large number of PEs. This has held promise since the emergence of such massively parallel coprocessors as the 60-core Intel Xeon Phi.

Phylogenetics

While SA represents a data-intensive application class in computational biology, phylogenetics represents a data-driven compute-intensive class of applications. In phylogenetics research, the primary goal is to reconstruct evolutionary trees that best describe the evolutionary relationship among different species, by observing and characterizing variations at the DNA and protein level. The "tree of life" is an example of an ambitious project for inferring the phylogeny linking all known life forms. Typical probability models of evolution used for this purpose are Jukes–Cantor (JC) and general time reversible (GTR). Unlike SA, the computational intractability of the problem is the primary stumbling block in advancing the state of research in phylogenetic inference, as the underlying problems have been proven to be NP-hard under various formulations.

Phylogeny reconstruction approaches are based on genomic distance (e.g., neighbor joining), combinatorial optimization (e.g., breakpoint phylogeny), or statistical methods [e.g., maximum likelihood (ML)]. The choice of the approach represents a tradeoff between accuracy and complexity—with the statistical methods being the most computationally complex, having super-exponential complexity, but also being the most accurate. Additionally, statistical methods involve real number computations as opposed to integer arithmetic for the other approaches.

Blanchette et al. pioneered the work on breakpoint phylogeny [12]. They reduced the problem of reconstructing an optimal phylogenetic tree to one of solving numerous instances of a version of the traveling salesman problem (TSP) where edge weights of the input graph are bounded to a fixed set of integer values. Bakos and Elenis [13] proposed a coprocessor design for whole-genome phylogeny reconstruction using a parallelized version of breakpoint median computation. The coprocessor uses an FPGA-based multicore implementation of the combinatorial search space of the TSP, while the input graph construction is performed in software. The search tree partitioning is carried out in such a manner that each core explores the tree in a different order. This is done to avoid complex load-balancing and intercore communication issues that occur if disjoint subtrees are assigned to different cores, because any of them might be subject to pruning. Their best average speedup of 1005× over software is observed using three cores. The best overall reduction in execution time is by a factor of 417×. However, these observations are for synthetic data, and hence difficult to correlate with real-life examples. Majumder et al. [14] demonstrated speedup up to 8430× on their NoC-based custom multicore platform, and evaluated both synthetic and biological input data. Their solution integrated up to 64 lightweight custom PEs, each designed with fine-grained parallel and pipelined architecture to efficiently carry out matrix reduction and tree traversal during the solution of TSP for breakpoint phylogeny. Their search space subtree partitioning was simple—each PE choosing from a pool of subtrees assigned in no particular order—and yet managed to achieve good load balancing and branch pruning by dint of an efficient interconnection topology, which also made the system energy efficient.

Statistical methods like ML and Bayesian inference (BI) involve millions of computations of the phylogenetic likelihood function (PLF), which calls for high-throughput real-number arithmetic. Owing to the computational intractability of the problems, several approximate algorithms and heuristics have been proposed, which have then been the subject of hardware acceleration.

A hybrid hardware/software approach proposed by Mak and Lam [15] takes the genetic algorithm for maximum likelihood (GAML) approach. The genetic algorithm is implemented in software and the computationally intensive ML equation is implemented in hardware. This work uses a Xilinx Virtex XCV800 FPGA as the hardware accelerator and a Pentium 4 PC with 1-GB RAM for running the software. Their results while reconstructing a 4-taxa phylogenetic tree under the JC model demonstrate an overall speedup of 30× over software and an ML speedup of over 300×, despite the communication overhead of the hybrid system. This work, however, does not explicitly state how the acceleration scales for larger taxa or more realistic complex models like GTR. Randomized axelerated maximum likelihood version VI for high-performance computing (RAxML-VI–HPC) is an efficient parallel algorithm for ML phylogeny reconstruction that can incorporate different models of evolution, including rate variation across sites. Blagojevic et al. have explored the porting, optimization, and evaluation of RAxML-VI–HPC on CBE [16]. They have taken advantage of different layers of parallelism afforded by the software and CBE—task-level parallelism across SPEs, task vectorization within SPEs, and/or loop-level parallelization across SPEs. They show that CBE speeds up the software over standard processing architectures—Intel Xeon and IBM Power5—and is more cost effective and power efficient than either architecture. However, the sheer complexity of porting the algorithm and the various optimizations required for CBE collectively pose a significant roadblock.

As mentioned earlier, PLF computation is the key to these statistical methods. An FPGA platform for accelerating PLF computation was proposed by Alachiotis et al. [17], where a Xilinx Virtex 5 SX240T with 1056 DSP48E slices was used to implement double-precision floating point multipliers and adders. Due to the limited amount of DSP48E slices on the FPGA, several multiplexer units are deployed to optimally exploit the available computational resources. A Sun x4600 system equipped with eight dual-core AMD Opteron processors running at 2.6 GHz with 64 GB of main memory was used as the baseline. An average speedup of 8.3× over a single core has been demonstrated for trees comprising 4–512 sequences on FPGA. The FPGA implementation also outperforms OpenMP-based parallel implementation on 16 cores in most cases, achieving speedups from 0.96× to 7.46×. The projected computational time for a full tree traversal using Felsenstein's pruning algorithm for 512 taxa is less than 1 ms, based on reported clock speed of 284.152 MHz. Through its partial limitations, this study highlights two desirable features of a hardware accelerator: efficient floating point computation hardware and seamless resource allocation.

A holistic comparison of several hardware accelerators is presented in [18], where Pratas et al. apply the popular BI tool MrBayes to evaluate performance, scalability, and programmability. They consider multiple-instruction–multiple-data (MIMD) architectures such as general purpose multicore (dual-core and quad-core Intel and AMD) processors and CBE, and single-instruction–multiple-data (SIMD) architectures such as GPU. The PLF in MrBayes is parallelized using OpenMP directives for the general-purpose multiprocessors, POSIX threads for the CBE systems, and compute unified device architecture (CUDA) for the GPU systems. For hardware-managed caches, the sharing of a cache level within the chip by all cores is a determining factor for efficient synchronization, and hence scalability. Systems with software-managed caches like CBE compensate the user effort by efficient synchronization mechanisms. On the other hand, there are fewer data transfers between the device memory and CPU because GPU has sufficient memory to handle input data. CUDA automatically handles data transfer synchronization, thus relieving the user of the responsibility of providing any explicit synchronization mechanism. PLF computation speedup is penalized by computation intensity and communication overhead inside the multicores. Quad-core AMD Opteron, where four cores are on a single die and share the same L2 cache, scales better compared to quad-core Intel Xeon, which has two L2 caches, each shared by a pair of cores. For CBE, speedup values are close to ideal for small data sets and performance is stable across different computation intensities. Even though SPEs do not share a common cache, CBE is more tolerant to synchronization, primarily because it relies on user-generated software for this. However, speedup values for large data sets and computation intensities are almost equal for general-purpose multicores and CBEs. GPUs display an increase in speedup as the computation intensity increases because they are designed to perform efficient execution of small parallel threads in a scenario where the computation-to-data ratio is high. The end-to-end time-to-solution encompassing the serial and parallelized (PLF) portions of the code is the minimum for general-purpose multicore. The bottleneck in CBE with regard to total execution time comes from the PPE that handles the serial portion of the code: it is a rather simple core with a small cache, in order execution capability and is burdened with the additional responsibility of synchronizing among SPEs.

A many-core system with custom lightweight PEs for efficient floating-point computation, integrated with a folded torus NoC, and colocality preserving resource allocation methods was proposed and evaluated with RAxML-VI–HPC in [19]. Majumder et al. implemented fast multiplication, addition, log, and antilog computations based on fixed-point hybrid number system. They explored the benefits offered by 3-D NoC architectures both in terms of speedup (up to 62% higher than 2-D NoC) and energy consumption (up to 38% lower than 2-D NoC). They devised Hilbert-curve-based locality-preserving task allocation strategies. The application model offloaded the computation-heavy kernels to the NoC-based platform, where up to 6594× application speedup was achieved. Taking into account the serial software time, the end-to-end runtime was reduced by 5×.

Table 2 summarizes the speedups achieved by different hardware accelerators for phylogeny reconstruction. These applications are characterized by high computation throughput requirements and the consequent demand for massively parallel architectures. FPGAs can suffice up to a certain extent, but general purpose GPUs (GPGPUs) and NoC-based multicores seem to be the way going forward. At the application mapping level, the current challenges come from on-chip resource allocation and consequent intrachip communication throughput requirement, which follow a time-varying, irregular pattern, though not random; and the serial software bottleneck.

Molecular simulation

Molecular dynamics (MD) is a widely used technique for studying the structural, functional, and

Table 2 Performance comparison for phylogenetic inference [12]–[18]. The "–" entries correspond to data not available.

Phylogeny reconstruction strategies	FPGA		GPU		Cell Broadband Engine		General Purpose Multicore		Custom Multicore (NoC)	
	Application speedup	*Total speedup*	*Application speedup*	*Total speedup*	*Application speedup*	*Total speedup*	*Application speedup*	*Total speedup*	*Application speedup*	*Total speedup*
Breakpoint phylogeny	1005	417	-	-	-	-	-	-	8430	-
Maximum likelihood (ML)	381	32, 13.68	8.5	1.9	12	1.5	12	10	6594	5

thermodynamic characteristics of biomolecules. It is a particular case of an N-body problem where hundreds of thousands of atoms of a molecular complex, such as a protein, are allowed to interact over a fixed spatial dimension, for over millions of discrete time steps. As such, the problem is heavily compute cycle bound. In [20], Stone et al. target various MD algorithms, such as direct Coulomb summation, multilevel Coulomb summation, and molecular dynamic force evaluation, and demonstrate that a CUDA-based GPU implementation provides up to 40× speedup over a highly optimized CPU-based implementation. Chiu et al. [21] demonstrate MD acceleration on a high-performance reconfigurable computing (HPRC) platform, where they use Altera Stratix-III EP3SE260 and fit eight force pipelines running at up to 198 MHz, and report 80× speedup over the GPU-based design in [20]. Another notable approach for MD acceleration has been the design of a massively parallel machine ANTON [22] by Shaw et al. This design consists of 512 identical MD-specific ASICs that interact in a high-speed 3-D torus NoC. The projected speedup was 100× that achieved using the dihydrofolate reductase (DHFR) benchmark on IBM Blue Gene/L with 8192 processing cores. By leveraging its ultralow-latency communication infrastructure to establish optimized communication patterns, ANTON improves the latency of the critical path communication step up to 30× the next fastest MD platform Desmond [23].

Molecular docking is another simulation class application that is heavily used to study the binding orientation of small molecule drug candidates to their protein drug targets. Due to its high computational complexity, dedicated hardware platforms have been considered for accelerating docking. Sukhwani and Herbordt mapped the FFT, modulation, accumulation, and some filtering tasks to GPU in [24] to achieve 17.7× speedup over single-core CPU and 6.1× speedup over a quad-core CPU. They also implemented a docking code on an FPGA platform [25] to achieve a 100× application speedup and 36× total speedup over single-core CPU and 10× speedup over quad-core CPU. The FPGA-based accelerator was found to be better for small-molecule docking while GPU excelled in protein–protein docking, which uses FFT techniques for 3-D grid representation of proteins. A method that incorporates rotational correlations, called spherical polar Fourier (SPF) technique, has been accelerated by 45× using GTX 285 GPU over a single-core CPU, but the same hardware does not perform well enough for conventional 3-D FFT docking [26]. A docking code, Autodock, was evaluated on FPGA (SGI RASC RC100 blade containing two Xilinx Virtex-4 LX200 FPGAs) and GPU (GeForce GT220 and GeForce GTX260) in [27]. The interesting observation is that while FPGA provided a constant speedup of 35× irrespective of the number of docking runs, the speedup provided by the GPUs varies from 5× for one run to 75× for 100 runs, reaffirming the fact GPUs perform better for larger molecules. Korb et al. [28] used GeForce 8800 GTX GPU and implemented the docking code using OpenGL and Cg (significantly, they did not use CUDA to ensure backward compatibility) and achieved speedup over 60× for rigid protein–protein docking, and up to 16× for the more complex case where the ligand and donor groups in the protein binding site are treated as flexible. The current research direction in molecular docking is increasingly focusing on GPUs, which are having an ever-increasing number of stream processors. However, with the multitude of algorithms that are used for docking, successfully mapping these onto GPUs using CUDA or other graphics-oriented languages appears to be a major challenge.

Biological network analysis

In addition to computational molecular biology, there is a great deal of interest in using the computational power of accelerator platforms, such as GPUs, for biological pathway analysis or brain neural network analysis. The complexity of the interaction graphs involved in these problems leads to HPC requirements, and quite a bit of recent research has been devoted to utilizing hardware acceleration platforms. For example, Han and Taha [29] present the use of GPUs (GeForce 9800 GX2) and GPGPUs (Tesla C1060 and Tesla S1070) to demonstrate speedups of 5.6× and 84.4× on Izhikevich and the Hodgkin–Huxley models, respectively, in the context of modeling image recognition systems in the cortex of the brain. Another work uses a CPU–GPU combination for brain network analysis, where the GPU (AMD Radeon 5870) is used to accelerate correlation calculation, modular detection, and all-pairs shortest path [30], delivering speedups of about 28×, 44×, and 166×, respectively. Hybrid functional Petri net simulation of biological pathways has been accelerated using GeForce GTX 285 GPU in [31], where biological pathway model simulations were accelerated by 18× for models with 20 000 processes and boundary formation by Delta–Notch signaling was sped up by 7× for tissue samples containing up to 1600 cells. Interestingly, GPU underperforms for models with less than 5000 processes because frequent kernel switching and weak thread utilization lead to a global memory access overhead for the CUDA-based application. As also concluded from the findings reported in Section IV, it is apparent that GPU-based acceleration is profitable above a certain problem-size threshold, below which overheads dominate, and above which these get amortized and do not affect the overall speedup.

Structural biology

Some of the most important problems in structural biology involve predicting the structure of macromolecules, such as proteins or RNAs. These contain chains of smaller molecules (e.g., amino acids or nucleic acids), which in turn form secondary structures (e.g., alpha-helices or loops), and eventually "fold" to form tertiary structures. Existing databases, such as the National Center for Biotechnology Information Protein Database (http://www.ncbi.nlm.nih.gov/protein), sourced from databanks such as GenBank, contain a repository of tens of thousands of molecules, and new molecules are discovered every day. Algorithms for predicting the 3-D structure and function of these molecules are based on comparison with existing molecules in the database, and are usually NP-hard. As such, there is strong focus on employing heuristics and hardware acceleration. In [32], Hung and Samudrala used a GPU-based system to accelerate protein structure prediction based on template modeling score, which is computationally more complex and more accurate than the conventional root mean square deviation score. They used an ATI 5870 GPU to obtain a 68× speedup over a general purpose processor. Jacob et al. [33] present an FPGA-based implementation of the Zuker DP algorithm to predict RNA secondary structure—a problem that is polynomial time solvable for simple cases, but becomes significantly harder [34] for consideration with pseudoknots. They use a Xilinx Virtex 4 LX100-12 FPGA to deliver 103× speedup over a CPU, and higher speedups than prior implementations on other FPGA and GPU platforms.

Other fields of application

Computation intensity, driven by either the volume of data or the underlying problem's intractability, is a key challenge in a wide range of other biological applications. The previous sections present the degree of progress and unmet challenges in some major fields, but there are many others that are being benefited by hardware accelerators. The field of mass spectrometry-based proteomics has seen the application of hardware acceleration, e.g., using GPU [35]. The problem here is to compare an experimentally acquired spectrum against a spectral library and/or sequence database. The GPU-based approach delivers up to 26× speedup for a parallelized spectral search algorithm [35]. In addition, there are applications that have been shown to benefit from hardware acceleration carried out using massively parallel hybrid systems such as those available at Texas Advanced Computing Center (Austin, TX, USA, http://www.tacc.utexas.edu/resources/hpc/stampede) and Convey Computer (Richardson, TX, USA, http://www.conveycomputer.com/solutions/life-sciences/). These studies clearly demonstrate the potential for hardware acceleration in this application domain.

It is evident that a lot of progress in computational biology is dependent on the synergy with computer architecture (for various types of hardware platforms), which would act as an enabler in the process. Most of these applications are data intensive and/or compute intensive, and can readily benefit from parallel architectures. Apart from traditional high-performance computing architectures such as supercomputers or server clusters, even single-chip multicore solutions—FPGA, GPU, CBE, and custom multicore—have provided extensive performance benefits in terms of speed, energy consumption, cost, and portability. Some of these architectures such as FPGAs and custom multicores have been designed to accelerate a specific target application, while for others such as GPU and CBE, the application has to be ported using platform-specific software tools. GPUs and their descendants GPGPUs currently integrate thousands of stream processors, and provide a clear edge in case of applications that can be cast into a SIMD model. Communication between the CPU and the GPU has been a bottleneck, but with better on-chip memory and software tools such as CUDA, GPGPUs have

Table 3 Design tradeoffs for acceleration platforms for computational biology.

Design Parameters	FPGA	GPU	Cell Broadband Engine	General Purpose Multicore	Custom Multicore (NoC)
Cost	low - medium	medium	low	medium	high
Development effort	medium	low - medium	medium	low	high
Speedup performance	medium - high	medium - high	low	medium - high	high
Energy efficiency	medium	medium	low	low	high

been able to minimize such data transfers and work around the I/O bandwidth issue. Due to their reconfigurable nature, FPGA platforms can provide problem-specific tailor-made parallel architectures. However, as we have seen, they often run into capacity limitations even with the latest FPGA boards that allow hundreds of cores to be implemented. In fact, certain applications seem to be better addressed using GPUs rather than FPGAs. On the other hand, custom multicores embedded on a network-on-chip have the advantage of a VLSI application-specific IC (ASIC), where the NoC can provide a much higher level of integration than in an FPGA. This has been shown to make a significant difference in certain classes of applications, and justifies the investment in research. Table 3 shows a comparison among different acceleration platforms, not only with respect to their speedup performance and energy-efficiency, but also with respect to design cost and development effort. These latter parameters will enable the designer to factor in the R&D and production expenses in developing the solution. This table should serve as a general guideline for choosing a particular platform based on the application requirement and evaluation of tradeoffs. While the architecture community is ceaselessly innovating, applications are coming up with newer demands, and it will ultimately be the researcher who decides what combination is best for the problem at hand. ■

■ References

[1] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *J. Molecular Biol.*, vol. 147, no. 1, pp. 195–197, Mar. 25, 1981.

[2] T. Oliver, B. Schmidt, D. Nathan, R. Clemens, and D. Maskell, "Using reconfigurable hardware to accelerate multiple sequence alignment with ClustalW," *Bioinformatics*, vol. 21, no. 6, pp. 3431–3432, 2005.

[3] M. C. Herbordt, J. Model, G. Yongfeng, B. Sukhwani, and T. VanCourt, "Single pass, BLAST-like, approximate string matching on FPGAs," in *Proc. 14th Annu. IEEE Symp. Field-Programmable Custom Comput. Mach.*, 2006, pp. 217–226.

[4] V. Sachdeva, M. Kistler, E. Speight, and T.-H. K. Tzeng, "Exploring the viability of the cell broadband engine for bioinformatics applications," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2007, DOI: 10.1109/IPDPS.2007.370449.

[5] W. Liu, B. Schmidt, G. Voss, and W. Muller-Wittig, "Streaming algorithms for biological sequence alignment on GPUs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 9, pp. 1270–1281, Sep. 2007.

[6] Y. Liu, A. Wirawan, and B. Schmidt, "CUDASW++ 3.0: Accelerating Smith-Waterman protein database search by coupling CPU and GPU SIMD instructions," *BMC Bioinf.*, vol. 14, p. 117, 2013.

[7] K. Benkrid, Y. Liu, and A. Benkrid, "A highly parameterized and efficient FPGA-based skeleton for pairwise biological sequence alignment," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 4, pp. 561–570, Apr. 2009.

[8] S. Sarkar, G. R. Kulkarni, P. P. Pande, and A. Kalyanaraman, "Network-on-chip hardware accelerators for biological sequence alignment," *IEEE Trans. Comput.*, vol. 59, no. 1, pp. 29–41, Jan. 2010.

[9] C. B. Olson, M. Kim, C. Clauson, B. Kogon, C. Ebeling, S. Hauck, and W. L. Ruzzo, "Hardware acceleration of short read mapping," in *Proc. IEEE 20th Annu. Int. Symp. Field-Programmable Custom Comput. Mach.*, 2012, pp. 161–168.

[10] T. Martínek and M. Lexa, "Hardware acceleration of approximate tandem repeat detection," in *Proc. IEEE*

18th Annu. Int. Symp. Field-Programmable Custom Comput. Mach., 2010, pp. 79–86.

[11] M. N. M. Isa, K. Benkrid, and T. Clayton, ''A novel efficient FPGA architecture for HMMER acceleration,'' in *Proc. Int. Conf. Reconfigurable Comput. FPGAs*, 2012DOI: 10.1109/ReConFig.2012.6416723.

[12] M. Blanchette, G. Bourque, and D. Sankoff, ''Breakpoint phylogenies,'' in *Genome Informatics Workshop.* Tokyo, Japan: Univ. Academy Press, 1997, pp. 25–34.

[13] J. D. Bakos and P. E. Elenis, ''A special-purpose architecture for solving the breakpoint median problem,'' *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 16, no. 12, pp. 1666–1676, Dec. 2008.

[14] T. Majumder, S. Sarkar, P. P. Pande, and A. Kalyanaraman, ''NoC-based hardware accelerator for breakpoint phylogeny,'' *IEEE Trans. Comput.*, vol. 61, no. 6, pp. 857–869, Jun. 2012.

[15] T. S. T. Mak and K. P. Lam, ''High speed GAML-based phylogenetic tree reconstruction using HW/SW codesign,'' in *Proc. IEEE Bioinf. Conf.*, 2003, pp. 470–473.

[16] F. Blagojevic, A. Stamatakis, C. D. Antonopoulos, and D. S. Nikolopoulos, ''RAxML-Cell: Parallel phylogenetic tree inference on the cell broadband engine,'' in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, 2007, DOI: 10.1109/IPDPS.2007.370267.

[17] N. Alachiotis, E. Sotiriades, A. Dollas, and A. Stamatakis, ''Exploring FPGAs for accelerating the phylogenetic likelihood function,'' in *Proc. IEEE Int. Symp. Parallel Distrib. Process.*, 2009, DOI: 10.1109/IPDPS.2009.5160929.

[18] F. Pratas, P. Trancoso, A. Stamatakis, and L. Sousa, ''Fine-grain parallelism using multi-core, cell/BE, and GPU systems: Accelerating the phylogenetic likelihood function,'' in *Proc. Int. Conf. Parallel Process.*, 2009, pp. 9–17.

[19] T. Majumder, M. E. Borgens, P. P. Pande, and A. Kalyanaraman, ''On-chip network-enabled multicore platforms targeting maximum likelihood phylogeny reconstruction,'' *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 31, no. 7, pp. 1061–1073, Jul. 2012.

[20] 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, Dec. 2007.

[21] M. Chiu and M. C. Herbordt, ''Molecular dynamics simulations on high-performance reconfigurable computing systems,'' *ACM Trans. Reconfigurable Technol. Syst.*, vol. 3, no. 4, Nov. 2010, article 23.

[22] D. E. Shaw, M. M. Deneroff, R. O. Dror, J. S. Kuskin, R. H. Larson, J. K. Salmon, C. Young, B. Batson, K. J. Bowers, J. C. Chao, M. P. Eastwood, J. Gagliardo, J. P. Grossman, C. R. Ho, D. J. Ierardi, I. Kolossváry, J. L. Klepeis, T. Layman, C. McLeavey, M. A. Moraes, R. Mueller, E. C. Priest, Y. Shan, J. Spengler, M. Theobald, B. Towles, and S. C. Wang, ''Anton, a special-purpose machine for molecular dynamics simulation,'' *SIGARCH Comput. Archit. News*, vol. 35, no. 2, pp. 1–12, Jun. 2007.

[23] R. O. Dror, J. P. Grossman, K. M. Mackenzie, B. Towles, E. Chow, J. K. Salmon, C. Young, J. A. Bank, B. Batson, M. M. Deneroff, J. S. Kuskin, R. H. Larson, M. A. Moraes, and D. E. Shaw, ''Exploiting 162-nanosecond end-to-end communication latency on Anton,'' in *Proc. Int. Conf. High Performance Comput. Netw. Storage Anal.*, 2010, DOI: 10.1109/SC.2010.23.

[24] B. Sukhwani and M. C. Herbordt, ''GPU acceleration of a production molecular docking code,'' in *Proc. 2nd Workshop General Purpose Process. Graphics Process. Units*, 2009, pp. 19–27.

[25] B. Sukhwani, M. C. Herbordt, and M. C., ''FPGA acceleration of rigid-molecule docking codes,'' *IET Comput. Digit. Tech.*, vol. 4, no. 3, pp. 184–195, May 2010.

[26] D. W. Ritchie and V. Venkatraman, ''Ultra-fast FFT protein docking on graphics processors,'' *Bioinformatics*, vol. 26, no. 19, pp. 2398–2405, 2010.

[27] I. Pechan and B. Feher, ''Molecular docking on FPGA and GPU platforms,'' in *Proc. Int. Conf. Field Programmable Logic Appl.*, 2011, pp. 474–477.

[28] O. Korb, T. Stützle, and T. E. Exner, ''Accelerating molecular docking calculations using graphics processing units,'' *J. Chem. Inf. Model.*, vol. 51, no. 4, pp. 865–876, 2011.

[29] B. Han and T. M. Taha, ''Acceleration of spiking neural network based pattern recognition on NVIDIA graphics processors,'' *Appl. Opt.*, vol. 49, pp. B83–B91, 2010.

[30] Y. Wang, M. Xu, L. Ren, X. Zhang, D. Wu, Y. He, N. Xu, and H. Yang, ''A heterogeneous accelerator platform for multi-subject voxel-based brain network analysis,'' in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, 2011, pp. 339–344.

[31] G. Chalkidis, M. Nagasaki, S. Miyano, and S., ''High performance hybrid functional petri net simulations of biological pathway models on CUDA,'' *IEEE/ACM*

Trans. Comput. Biol. Bioinf., vol. 8, no. 6, pp. 1545–1556, Nov.–Dec. 2011.

[32] L.-H. Hung and R. Samudrala, ''Accelerated protein structure comparison using TM-score-GPU,'' *Bioinformatics*, vol. 28, no. 16, pp. 2191–2192, 2012.

[33] A. C. Jacob, J. D. Buhler, and R. D. Chamberlain, ''Rapid RNA folding: Analysis and acceleration of the Zuker recurrence,'' in *Proc. 18th IEEE Annu. Int. Symp. Field-Programmable Custom Comput. Mach.*, 2010, pp. 87–94.

[34] E. Rivas and S. R. Eddy, ''A dynamic programming algorithm for RNA structure prediction including pseudoknots,'' *J. Molecular Biol.*, vol. 285, no. 5, pp. 2053–2068, 1999.

[35] L. A. Baumgardner, A. K. Shanmugam, H. Lam, J. K. Eng, and D. B. Martin, ''Fast parallel tandem mass spectral library searching using GPU hardware acceleration,'' *J. Proteome Res.*, vol. 10, no. 6, pp. 2882–2888, 2011.

Turbo Majumder is an Assistant Professor in the Department of Electrical Engineering, Indian Institute of Technology Delhi, New Delhi, India. He works on many-core network-on-chip platforms, system-on-chip platforms, hardware acceleration, and high-performance computing. He has a BS in electronics and electrical communication engineering and an MS in automation and computer vision from Indian Institute of Technology, Kharagpur, India, and a PhD in electrical engineering from Washington State University, Pullman, WA, USA. He is a Member of the IEEE.

Partha Pratim Pande is an Associate Professor and the holder of the Boeing Centennial chair in Computer Engineering at the School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA, USA. His current research interests are novel interconnect architectures for multicore chips, on-chip wireless communication networks, and hardware accelerators for biocomputing. He has an MS in computer science from the National University of Singapore, Singapore and a PhD in electrical and computer engineering from the University of British Columbia, Vancouver, BC, Canada. He is a Senior Member of the IEEE.

Ananth Kalyanaraman is an Associate Professor at the School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA, USA. His research interest is in high-performance computational biology. He has a PhD in computer engineering from Iowa State University, Ames, IA, USA (2006). He is a member of the American Association for the Advancement of Science (AAAS), the Association for Computing Machinery (ACM), the IEEE Computer Society, and the International Society for Computational Biology (ISCB).

■ Direct questions and comments about this article to Partha Pratim Pande, School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA 99164-2752 USA; pande@eecs.wsu.edu.