A High-Performance Heterogeneous Computing Platform for Biological Sequence Analysis

Xiandong Meng, *Senior Member*, *IEEE*, and Vipin Chaudhary, *Member*, *IEEE*

Abstract—Advances in bioinformatics research continue to add complexity to the analyses and interpretation of biological data. Certain sequence database searches may take weeks to complete due to complicated data dependencies by dynamic programming. A reconfigurable coprocessor can remove this computational bottleneck and accelerate the operation. This paper presents a heterogeneous computing platform through Message Passing Interface (MPI) enabled enterprise computing infrastructure for high-throughput biological sequence analysis. The computing platform integrates heterogeneous computer architectures including conventional processors with Streaming Single Instruction Multiple Data Extensions 2 (SSE2) instructions, reconfigurable coprocessors, and legacy processors together into one system, and allows each to perform the task to which it is best suited. With appropriate computation and communication scheduling, the integrated heterogeneous computing infrastructure is designed to accommodate various types of accelerators to provide a High-Performance Computing (HPC) framework to support the most widely used life science applications.

Index Term—Heterogeneous computing platform, Smith-Waterman algorithm, dynamic programming, sequence alignment, SIMD, SSE2, FPGA, MPI, HPC.

1 Introduction

Although CPU architectures are continuing to increase performance, the volume of biological data has increased at a much faster rate in the past several years. For example, GenBank doubles every 9-12 months. The number of bases in these databases continues to grow at an exponential rate. As of April 2006, there are more than 130 billion bases in GenBank and RefSeq alone. Although Moore's law, which predicts a doubling of the density of the transistors on a chip every 18 months, is still going strong, the use of the Smith-Waterman algorithm [26], the "gold standard" for sequence comparison, on a single processor has become too costly, inefficient, and time-consuming.

Biological sequence analysis, which is the most widely used bioinformatics application, has not only become essential for basic genomic and molecular biology research, but is having a major impact on many areas of biomedicine. The areas of biological sequence analysis include sequence alignment, sequence database searching, motif and pattern discovery, reconstruction of evolutionary relationships, and genome assembly and comparison. It has applications, for instance, in knowledge-based drug design, forensic DNA analysis, and agricultural biotechnology. The Smith-Waterman algorithm is an exhaustive sequence database search algorithm based on dynamic programming and is generally considered a lot more sensitive than any other heuristic-based algorithm, such as BLAST [3] and FASTA [21]. But, long computation time limits the use of this algorithm. In our previous study, we have exploited the optimized fine-grained parallelism and coarse-grained parallelism for sequence similarity search using both general-purpose processors [17] and special-purpose hardware [16], [29]. Sequence database searches based on reconfigurable hardware platforms have reported tremendous speedup as compared to a standard desktop PC by Oliver et al. [20]. The Smith-Waterman dynamic programming has been mapped to a linear systolic array of Processing Elements (PEs) on a commercial-off-the-shelf (COTS) FPGA board, where supercomputer performance has been achieved.

The enterprise network architecture is designed to allow organizations to quickly refine the notion of using COTS parts to create enterprise-class data processing systems capable of meeting current and future computing requirements. More enterprises are turning to a heterogeneous computing infrastructure for distributed computing and data resources such as processing, network bandwidth, and storage. Heterogeneity allows enterprises to pool available resources for scalability and high availability. Using several inexpensive, standards-based components, such as multi-core Intel/AMD architecture-based servers, can be an appealing option for the typical enterprise data center. As the need for more processing power arises, administrators increase capacity by adding more of these small components to the aggregate system. A cluster architecture, such as a SunFire cluster, provides applications access to more horsepower when needed, while allowing computing resources to be used for other applications when database resources are not as heavily required.

- *X. Meng is with the Supercomputing Facility, Texas A&M University, 3363 TAMU, College Station, TX 77843. E-mail: x-meng@tamu.edu.*
- *V. Chaudhary is with the Department of Computer Science and Engineering, NYS Center of Excellence in Bioinformatics and Life Sciences, University at Buffalo, The State University of New York, Buffalo, NY 14260. E-mail: vipin@buffalo.edu.*

Manuscript received 26 Nov. 2008; revised 27 Aug. 2009; accepted 21 Oct. 2009; published online 23 Dec. 2009.
Recommended for acceptance by H. Jiang.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2008-11-0467.
Digital Object Identifier no. 10.1109/TPDS.2009.165.

We have developed a heterogeneous computing platform for biological sequence analysis that, by utilizing the existing and available enterprise resources, improves the overall performance of the sequence database searches. The computing platform seamlessly integrates vector computing, reconfigurable computing, and cluster computing into one high-performance computing architecture, which can be tailored specifically to perform sequence comparisons at high speeds. FPGA reconfigurable coprocessors can execute hundreds or even thousands of conventional instructions as a custom instruction set, and be easily configured to meet various computational requirements. By coupling the acceleration services to enterprise infrastructure architectures allows us to manage data and deploy new technologies without impact to regular users. It also provides greater efficiency and business agility. Although our ideas are applicable to all types of sequence comparison algorithms, we focused on Smith-Waterman implementation as a concrete example to illustrate the effectiveness of our technique.

In the reminder of this paper, we first review the related work on parallel implementation of sequence comparison algorithms and details of the Smith-Waterman algorithm in Sections 2 and 3, respectively. Section 4 describes the system architecture and highlights each computing component. Section 5 details the optimization of the computing system utilizing scheduling and multilevel parallelism. Section 6 describes the hardware and software platforms. Section 7 presents results accuracy analysis and system performance. Finally, we end with conclusions in Section 8.

2 Related Work

The well-known sequence comparison algorithms have been implemented in parallel using a general-purpose cluster of machines, such as Beowulf clusters, or special-purpose hardware, such as FPGAs. mpiBLAST [7] is a parallelization of BLAST, which achieves super linear speedup by segmenting a BLAST database and then having each node in a computational cluster search a unique portion of the database. mpiBLAST is benchmarked on the Green Destiny cluster [30], which is a 240-node-processor supercomputer. Each node of this cluster consists of a 667 MHz Transmeta Crusoe TM5600 processor. It is able to achieve a speedup of over 160 for 128 processors. The parallel efficiency of FASTA program on Sun servers can be quite high, especially for the large database searches [25]. The experiment results show a 54-fold speedup of FASTA search when it runs on 62 CPUs of the SUN Enterprise 10,000 (64 CPUs of 400 MHz). The Cray XD1 [5] is a high-performance supercomputing platform that combines the processing power of 12 AMD dual-core 64-bit Opteron processors with the flexibility of six Xilinx Virtex-II Pro FPGAs through a high-speed interconnection. The Cray XD1 System's Smith-Waterman solution has performed 28 times faster when tested against other Opteron processor-based solutions running the same application. Some other examples are Kestrel [10], SAMBA [12], Splash-2 [13], the DSP implementation on a Cradle 3SoC chip [16], the SSE2 implementation [8], [17], Fuzion [24], DeCypher [27], and the FPGA implementations on a Virtex2-8000 [6], and a Virtex XCV2000E [32], respectively. Recently, several efforts have extended to Graphics Processing Units (GPUs) [15] and IBM Cell Broadband Engine (Cell BE) [1], [23], [31] for heterogeneous bioinformatics computing.

However, solutions based on supercomputers, large scale computing clusters [30], or specially designed machines [5], are quite expensive and users have to purchase the high-cost computing systems and upgrade the entire hardware. Solutions based on a single accelerator board [6], [32] only perform a simplified linear gap penalty database search or have constraints on the length of the query/database sequence. Kestrel SIMD parallel processor [10] accepts the maximum query length of 2,555, a packing of five query characters per PE. In a recent FPGA Smith-Waterman implementation [6], the maximum length of the query is 400 and the maximum length of database is 4,096 with a Virtex2-8000 device. Compared to another design presented in [32], the maximum length of the query is 1,024 with Virtex XCV300, and the maximum length of the query is 2,048 with Virtex XCV2000E. Our proposed solution eliminates these drawbacks and performance bottlenecks by integrating COTS FPGA accelerators into the existing enterprise computing infrastructure at low cost.

3 Smith-Waterman Database Searches

Similarities between subsequences of two sequences are usually the evidences in the methods to find homologies by database searches. The Smith-Waterman algorithm [26] is perhaps the most widely used local similarity algorithm for biological sequence comparison. In the Smith-Waterman database searches, the dynamic programming method is used to compare the query sequence to every database sequence and assign a score to each comparison, and entries are calculated using the following recurrences:

$$H[i,j] = \max \begin{cases} E[i,j] \\ H[i-1,j-1] + SS(ai,bj) \\ F[i,j] \\ 0 \end{cases}$$
$$for\ 1 \le i \le m, 1 \le j \le n,$$

where

$$E[i,j] = \max\{H[i-k,j] - g(k)\}, \quad for\ 0 < k < i,$$
$$F[i,j] = \max\{H[i,j-l] - g(l)\}, \quad for\ 0 < l < j.$$

Here, $H[i,j]$ is the score of the optimal alignment ending at position $[i,j]$ in the matrix, while $E[i,j]$ and $F[i,j]$ are the scores of optimal alignments that ends at the same position but with a gap in sequence A or B, respectively. $SS(ai,bj)$ is the match/mismatch value of ai and bj, or amino acid substitution score matrix, while $g(k)$ and $g(l)$ are the gap penalty functions. The computations should be started with $E[i,j] = F[i,j] = H[i,j] = 0$ for all $i = 0$ or $j = 0$, and proceed with i going from 1 to m and j going from 1 to n.

For sequences of lengths m and n, the Smith-Waterman algorithm uses memory space proportional to the product of the lengths of the two sequences, i.e., $O(mn)$, and computing time to $O(mn(m+n))$. Gotoh [11] reduced the time needed by the algorithm to $O(mn)$ when affine gap penalties of the form $g(k) = q + rk; (q \ge 0, r \ge 0)$ are used, where q is the gap open penalty and r is the gap extension

penalty. For affine gap penalties, the above recurrence relations can be changed to:

$$E[i,j] = \max\{H[i-1,j] - q, E[i-1,j] - r\}, \quad for\ 1 \le i \le m, 0 \le j \le n,$$

$$F[i,j] = \max\{H[i,j-1] - q, E[i,j-1] - r\}, \quad for\ 0 \le i \le m, 1 \le j \le n.$$

Entries $H[i,j]$ in all other cells of the matrix are defined as the score of the best alignment ending in the position matching ai and bj. The order of computation is strict, because the value of H in any cell in the alignment matrix cannot be computed once all cells to the left or above it has been computed. The overall optimal alignment score is equal to the maximum value of $H[i,j]$.

4 System Description

In general, the computing platform takes a query DNA or amino acid sequence and searches for similar sequences in a database of known sequences. After the completion of the search, the program reports the top alignment results based on the statistical significance of the similarities between the query and the sequences in the database.

There are three different types of processing components in the system: a conventional processor with SSE2 instruction set, an FPGA coprocessor, and a legacy CPU processor. Each of these processing components has been optimized to execute the Smith-Waterman sequence database search algorithm with the affine gap penalties at high speed. A conventional processor with SSE2 component uses the Single Instruction Multiple Data (SIMD) technology to accelerate the Smith-Waterman search. An FPGA coprocessor component takes advantage of reconfigurable FPGA circuits to speedup the Smith-Waterman algorithm core. A legacy CPU processor component uses only the general-purpose processor of the worker machine to search the database.

4.1 Architecture Overview

The heterogeneous computing platform is intended to be implemented using one master node, and multiple worker nodes with multiple FPGA accelerator boards; however, it is also capable of running on a single machine. Our current implementation works on Linux-based nodes but can as easily be adapted to other operating systems. These nodes run the Linux operating system and are connected by a network with a network file system. The master node controls the overall operation of the computing system. It accepts the queries and parameters from users, assigns the jobs to worker nodes, and merges and outputs the final searching results. The worker nodes receive queries from the master, load the assigned database fragments, and perform the sequence comparison using one of the three processing components. Fig. 1 shows the architecture of the heterogeneous computing platform. Benefits of heterogeneous computing can be immediately realized through this type of usage model not only by IT staff, but also by biological researchers. With dynamic allocation of resources, a common user can run more of the same application, run more applications or run their applications faster.

4.2 SSE2 Vector Computing

Intel introduced the SSE2 instruction set [14] for SIMD calculations with the Pentium IV or higher processors. SSE2 allows for SIMD operations on 128-bit IEEE double-precision floating-point data types, so it's useful in tasks like 3D graphics, gaming, and media encoding. With the techniques like vector-based row wise parallelism, eight-way parallel processing with 16-bit values, data padding and data alignment, the SSE2 implementation [17] was also proven to effectively speedup the Smith-Waterman database searches with high accuracy.

Fig. 2 illustrates the vector computing dataflow in a Smith-Waterman dynamic computation matrix. The sequentially calculated values, such as H, E, and F, have been grouped into vectors. While developing the fine-grained parallelism at the instruction level, it became clear that a small number of critical SIMD transformations could have a significant impact on code efficiency. Our SSE2 implementation of the Smith-Waterman algorithm shows excellent speedup on both Intel Pentium IV and AMD Opteron processors. A striped SSE2 Smith-Waterman implementation with 16 8-bit elements parallel processing can be found at [8].

4.3 FPGA Coprocessor Integration

The FPGA device acts as a highly flexible coprocessor, which is integrated into a host system. The FPGA boards are used to execute the most time-consuming part of an application, which is the Smith-Waterman dynamic programming algorithm core for exhaustive sequence database searches. All other components of the application, e.g., parameter initialization or file handling, which are not time consuming, are still executed on the conventional processor of the host system.

The control of the FPGA board and the exchange of data words between the host system memory and the FPGA board are performed using a board specific Application Programming Interface (API). The content and the structure of exchanged data are defined by the application. For example, database sequences are transferred from the host memory to the FPGA board and the processed similarity scores are written back to the host memory for further processing. The API also provides necessary functions, which are used to read the status and result registers or to write the control registers on the board or within the FPGA architecture. The FPGA device is utilized for the fine-grained parallel execution of the Smith-Waterman algorithm as a coprocessor.

The Smith-Waterman dynamic programming algorithm has been mapped to a linear systolic array of PEs [20], [33] as shown in Fig. 3. The systolic array is composed of identical PEs. The characters of a query sequence are preloaded into each PE, while the database characters are fed serially into the array. Thus, a comparison between pairs of characters, which applies the algorithm's equations, is done on each machine cycle. An array of length n can be compared to an input sequence of length m in $O(n + m - 1)$ steps.

4.4 Legacy System

Our heterogeneous computing platform is designed to accommodate all types of computing resources including legacy systems, which in reality are still being widely used today. If a node doesn't support the SSE2 instruction set or

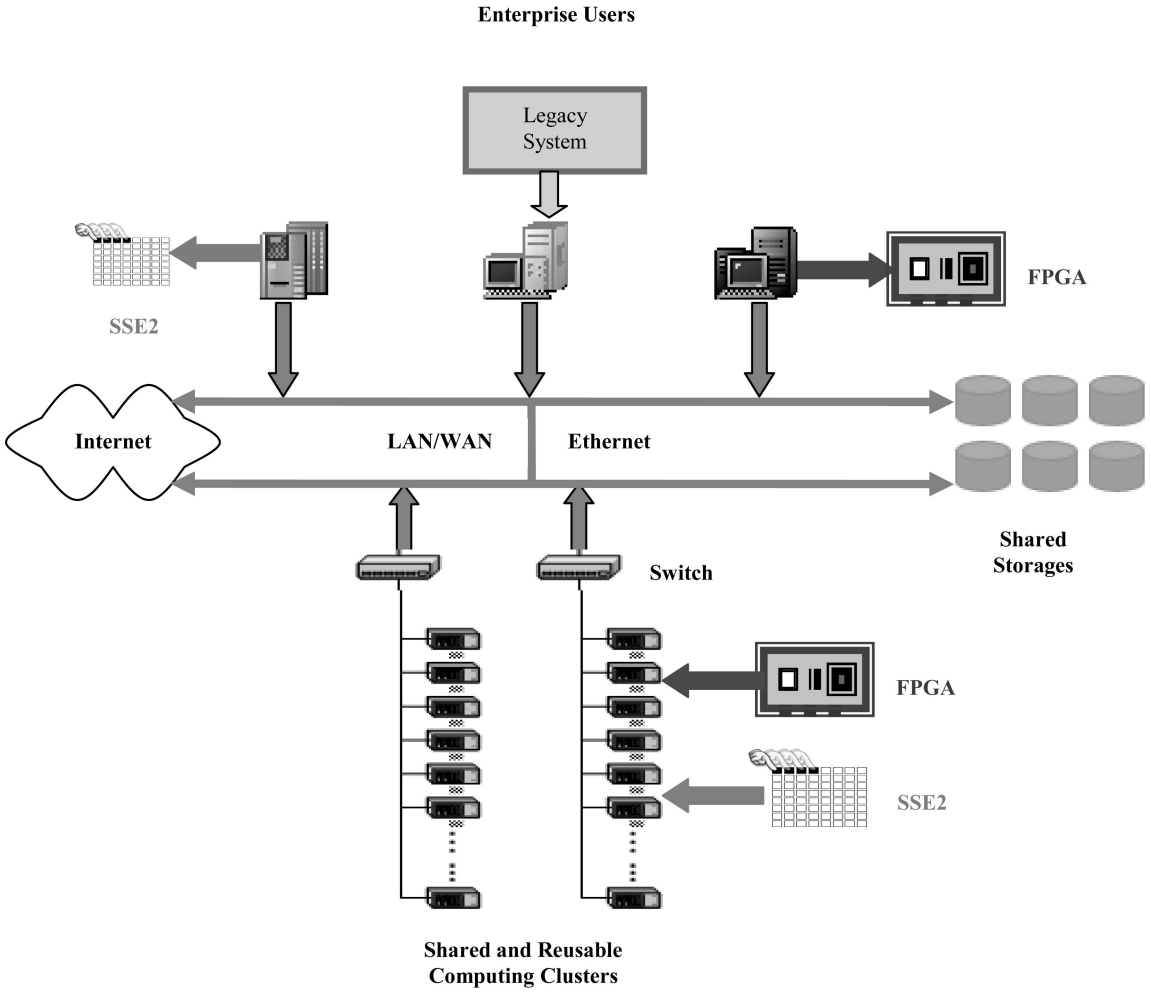


Fig. 1. Heterogeneous computing system architecture.

1) Query Sequence

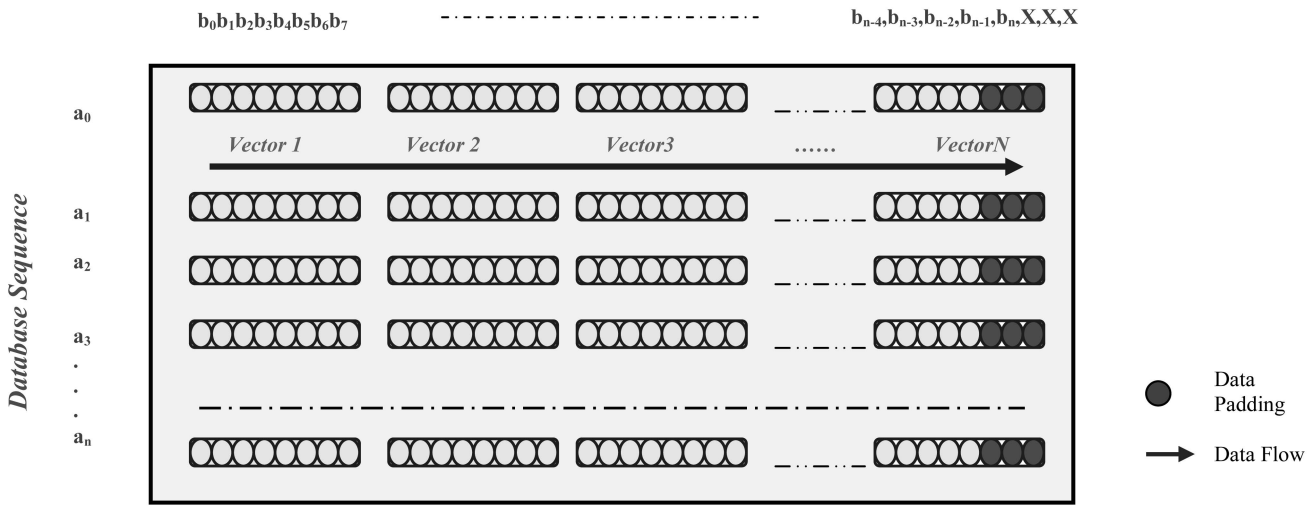


Fig. 2. SSE2 vector-based Smith-Waterman computing matrix.

has no FPGA accelerator boards available, it can still be used as a worker node to get involved in parallel computations. In order to exploit maximal resource utilization, the system will schedule tasks based on their computing power. This scheduling scheme is presented in the next section.

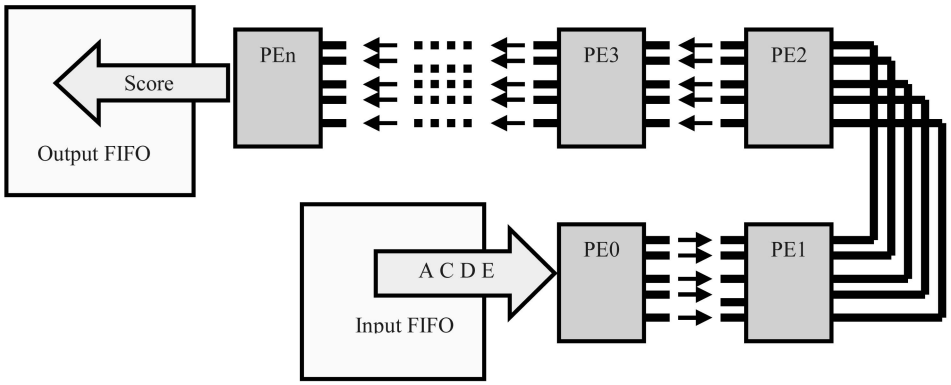


Fig. 3. Diagram for linear systolic processor array.

5 Scheduling and Optimization

Different processors exhibit varying performance for applications. Heterogeneous computing allows us to use the right processor for the right operation within a single workflow. However, it presents a challenge in distributing workloads across multiple processors which are of different types. This section describes how we can optimize performance on a heterogeneous computing platform through multilevel parallelism, flexible scheduling, and a series of optimizations.

5.1 Multilevel Parallelisms

The proposed computing platform takes advantage of combinations of fine-grained and coarse-grained computing paradigm within a single parallel architecture using various types of processing components.

At a higher level, the searching of large databases leads itself to an implementation with multiple subsearches distributed across separate processors. At this level, a large database is split into multiple nearly equal sized fragments. Each computing node is assigned a number of database fragments depending on its processing capacity. Parallel searches using database segmentation can exhibit linear speedup versus searches on a single node. At the lowest level, micro-parallelism techniques have been used to take advantage of specific features of processor architecture. At this level, SSE2 instructions can extract more instruction-level parallelism from application codes on a conventional processor. Moreover, FPGA coprocessors offer huge amounts of inherent parallelism and can be programmed to perform thousands of instructions every clock cycle. Fig. 4

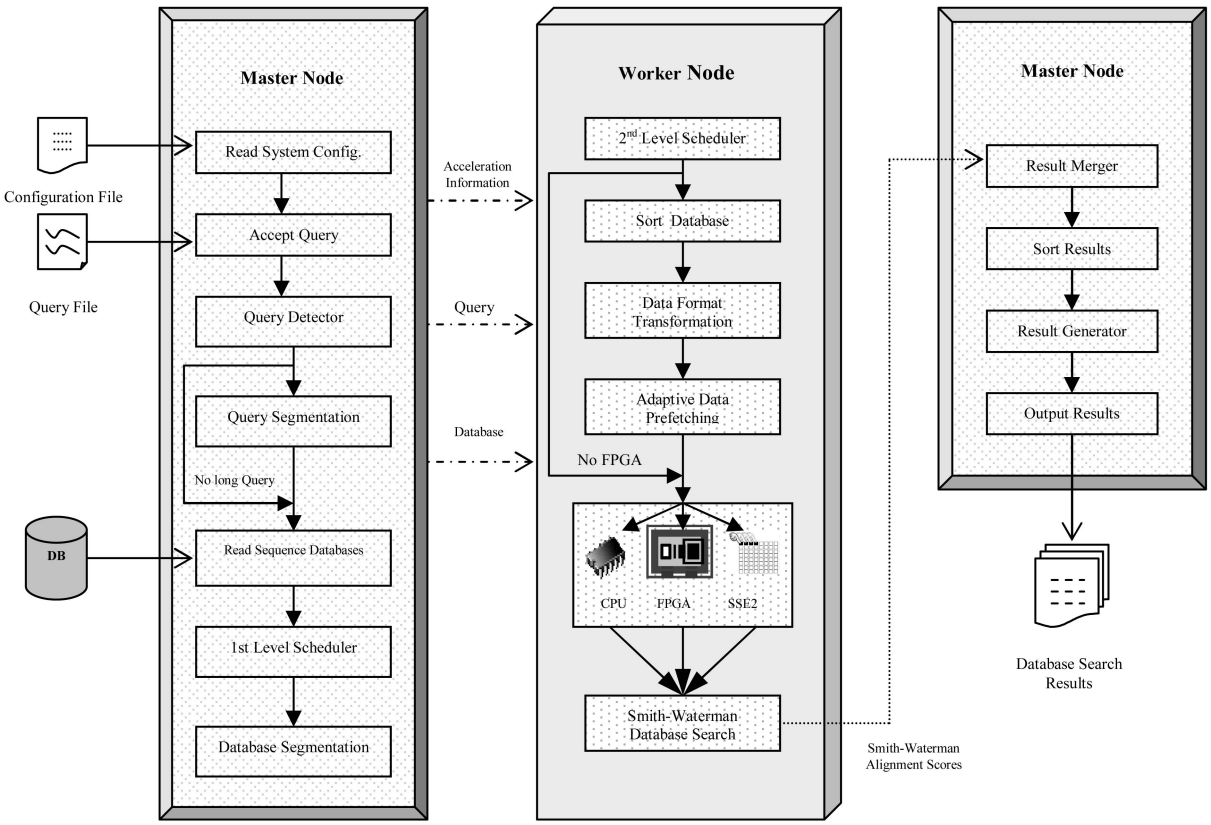


Fig. 4. Flowchart of porting application with multilevel parallelism on heterogeneous computing architecture.

	Legacy	SSE2	FPGA	#Processor Types include legacy CPU processors, CPU processor SSE2 and FPGA coprocessors
Master	1	0	0	#The Master Node
Num_Workers	3			# The number of Workers in the system
Worker1	1	0	0	#Worker Node1 has a legacy processor
Worker2	0	1	0	#Worker Node2 has a conventional CPU processor with SSE2
Worker3	0	1	1	#Worker Node3 has a CPU processor with SSE2 and an FPGA coprocessor

Fig. 5. Sample system configuration file.

shows the process of porting an application with multilevel parallelism on a heterogeneous computing architecture.

5.2 Scheduling Strategy and Load Balancing

Large degrees of heterogeneity in a system add significant additional complexity to the scheduling problem. In an ideal homogeneous system, for example, the assignment of a parallel task to any of the processors will not affect the total execution time of that task by adding extraneous computation or communication time. In a heterogeneous system, however, tasks typically can execute only on a subset of the available processors. Furthermore, the execution time of a task on the different processors can vary dramatically depending on how well matched the task is to the architecture of the processor. Therefore, it is important for us to consider both the type of processor to which a task can be assigned, and the cost of the communication that results from executing a task on a particular processor. The former can be solved by an appropriate scheduling strategy while the latter can be minimized by a data prefetching scheme, which is explained in the next section.

The objective function of our scheduling is to find a policy for assigning processors to tasks so as to maximize the overall throughput and minimize the response time of a single data set for sequence database searches. Because of the increasing size of sequence database and heterogeneous computational resources, a good scheduling strategy is necessary to exploit the effectiveness of multilevel parallelism for sequence database searches.

Balancing workload for better parallelism and workload distribution must be weighed against the available resources in a heterogeneous computing environment. Ideally, a single FPGA coprocessor will deliver up to one hundred times speedup over an Intel Pentium 1.9 GHz processor for the Smith-Waterman database search; therefore, there exists a huge imbalance of processing capabilities between conventional processors and FPGA coprocessors. The computing requirements for such database search problems can only be tackled by using the computing resources efficiently. Since the execution time is associated with the types of processing components, a dynamic load balancing mechanism was implemented based on hardware configuration. Within the computing platform, the master node controls a configuration file, which describes all necessary hardware information with respect to all worker nodes. For each node, the type of processing component in the machine, such as a legacy processor, a conventional processor, or an FPGA coprocessor, is specified. Fig. 5 shows a simplified sample configuration file.

Querying the database is accomplished by directly executing the Smith-Waterman algorithm on the various types of processors. Algorithm 1 gives an overview of the MPI master node strategy used to enable multiple heterogeneous worker nodes and distribute workload based on the hardware configuration. In order to overcome the hardware processing limitation, the query sequence has to be preprocessed using a query segmentation strategy. We discuss details of the query segmentation in Section 5.4.2. Algorithms 2 and 3 show the worker node algorithms for using the general purpose processor and FPGA coprocessor, respectively.

Algorithm 1. Master Node MPI Algorithm

```
Initialization
Read the Configuration File
MPI_Send Hardware Information to Workers
Scheduler: MPI_Send Database Fragments to
Heterogeneous Workers
while Unsearched Query ≠ 0 do
    if Query > Length_Limit then
        Query Segmentation
        loop
            MPI_Send Sub-Query to FPGA
        until No More Sub-Query
    else
        MPI_Send Query to workers
    end if
```

```
end while
MPI_Recv Results from workers
Merge Results from workers
Statistical Analysis of Results
Print Final Alignment Outputs
```

Algorithm 2. Worker Node MPI Algorithm with conventional processors

```
MPI_Recv Hardware Information (SSE2 or Legacy) from Master
MPI_Recv Database Fragments from Master
repeat
    MPI_Recv Query from Master
    while Query ≠ 0 do
        while Database ≠ 0 do
            Retrieve Results from Software
            Processing (Query, Current Sequence)
        end while
            MPI_Send results to Master
    end while
until No More Query
```

Algorithm 3. Worker Node MPI Algorithm with FPGA Accelerators

```
MPI_Recv Hardware Information (FPGA) from Master
MPI_Recv Database Fragments from Master
repeat
    MPI_Recv Query from Master
    while Query ≠ 0 do
        Open FPGA
        Load Query to FPGA
        while Database Fragment ≠ 0 do
            Send Database Fragments to FPGA
            Retrieve Results from FPGA (Query,
            Current Fragment)
        end while
            Close FPGA
            MPI_Send Results to Master
    end while
until No More Query
```

A job scheduler, on the master node, can automatically weigh the workload based on the dynamically selected worker nodes by the user, and then distribute the database fragments proportionally across multiple heterogeneous processors as shown in Algorithm 1. That is, the master node is able to distribute equal sized fragments unequally to all available worker nodes. For example, the node with an FPGA coprocessor should be assigned many more database fragments than the worker node with a legacy processor. In this way, the database resources are fully distributed to each worker node proportionally. By eliminating processor's idle time and keeping all computing components busy during the entire searching phase, a balanced workload can be achieved.

5.3 Data Prefetching

The data communications between the PCI-based FPGA board and the host system with the large volume of biological data strongly impacts the performance. Applying a data prefetching strategy to reduce data starvation and shadow scheduling latency is an effective way to eliminate the communication overhead. We have developed an adaptive data prefetching scheme [18], by using a parallel implementation, for the execution of the Smith-Waterman sequence database search to further improve the performance.

We designed a double buffering parallel implementation using Direct Memory Access (DMA) on the FPGA board, and Pthreads on the host machine. The idea of the double buffering strategy is to transfer database sequences into one intermediate application buffer in the background, where the processing can be completed in the other application buffer utilizing the full FPGA capability. Once processing is done in one buffer, the FPGA coprocessor sends results back to the host and receives new sequences from the other buffer, which has been filled by the host machine during the FPGA processing, and starts computation immediately with a minimum of idle time. The flowchart of the double buffering design is shown in Fig. 6.

Thus, we are able to rearrange the data flow to overlap communication with enough independent computation in each loop. The transformation overlaps the communication time with the computation time in parallel loops to effectively hide the latency of the database sequence transfer time. Since the proportions of database loading and FPGA processing time keep changing with the varied query inputs, a query-based adaptive data prefetching algorithm based on the length of the query sequence has been implemented. In Fig. 7, the experimental results [18] show that an appropriate intermediate buffer size for data transfer could improve the performance by up to 42 percent.

Prefetches should be issued early enough to hide communication latency, but not too early. Data prefetching attempts to leverage this overlap by determining when to initiate data transfer in order to maximize overlap with useful computation on the FPGA coprocessor. The software scheme prefetches a data block at least one iteration before it is used. The prefetch is usually placed immediately after a new block of data has been processed by the FPGA. The gap between arrival time of the prefetched data and its actual use should be minimized. Our proposed scheduling solution is able to resolve the unbalance between loading and processing time by providing a more flexible prefetching mechanism; e.g., the adaptive prefetching buffer size.

5.4 Sequence Comparison Optimization

5.4.1 Data Format

Sequence formats are simply the way in which the DNA or amino acid sequences are recorded in a computer file. Some of the most widespread sequence formats are EMBL, FASTA, GCG, GenBank, and so on. The programming complexity of FPGA requires that the FPGA implementation of database searches have its own set of data formats, such as database format and scoring substitution matrix format, which are incompatible with any of the standard sequence formats.

Sequence databases are updated weekly; therefore, it is impractical to maintain separate databases for the FPGA processing only, and it also brings additional processing overhead to convert the standard database format to the FPGA format. We made seamless transformation on data formats between conventional processing and FPGA processing, and incorporated FPGA processing into the FASTA

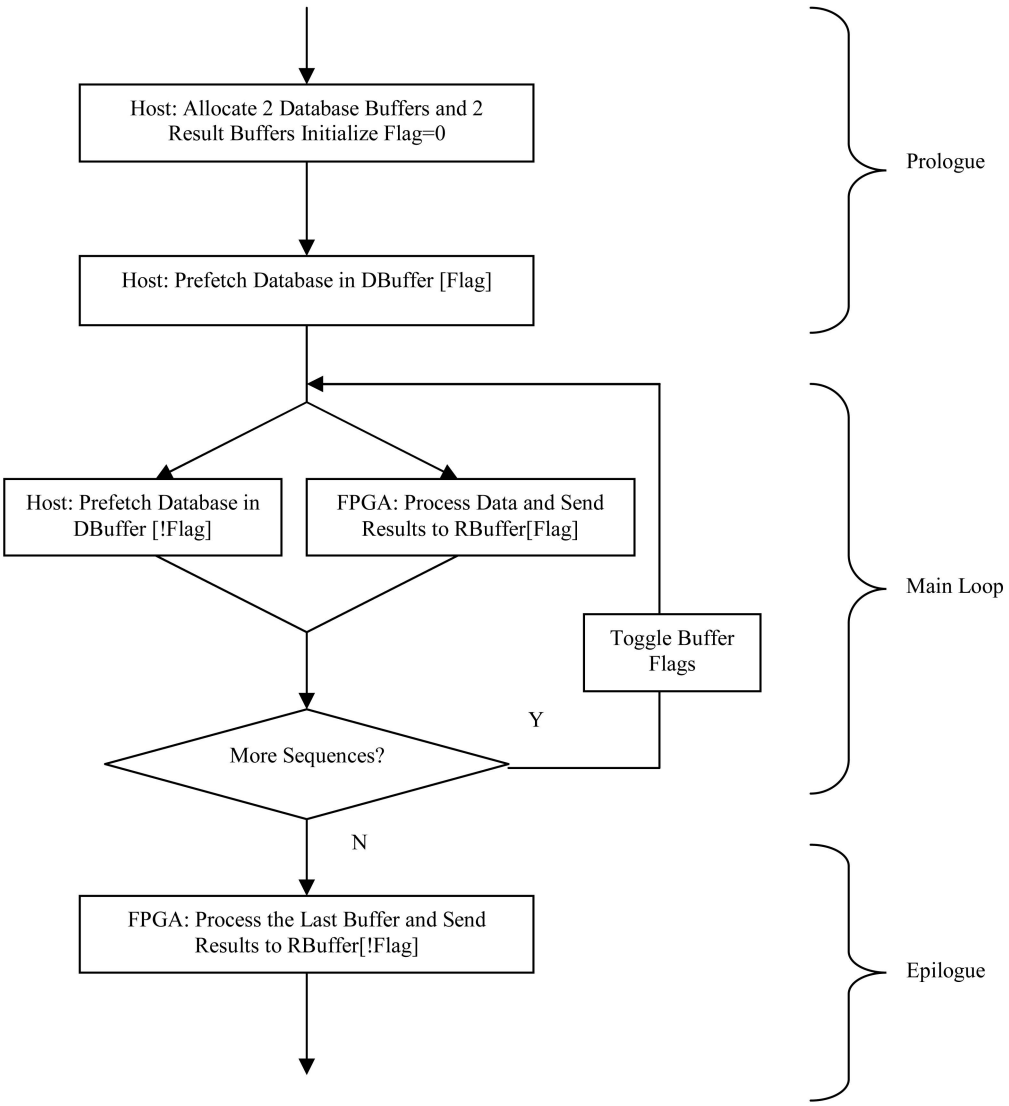


Fig. 6. Flowchart of data prefetching implementation.

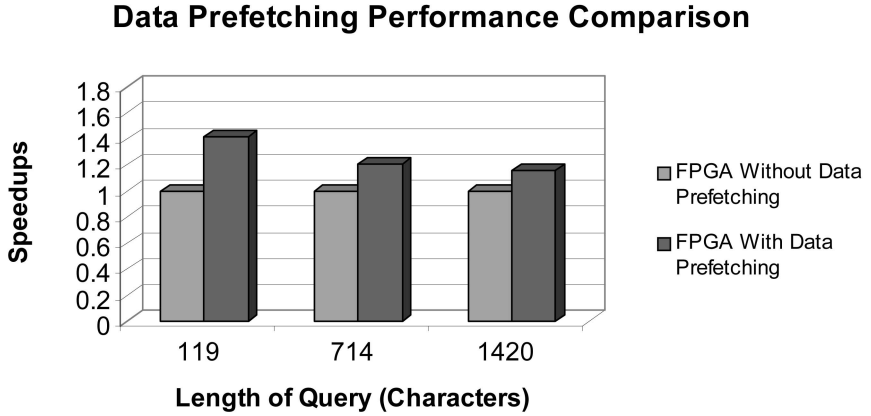


Fig. 7. Performance of adaptive data prefetching.

program [9]. Now the heterogeneous computing platform can support query and database sequences in any of those formats supported by the FASTA program, as well as the two most commonly used types of scoring matrices, the percent accepted mutations (PAM), and blocks substitution matrix (BLOSUM) series of matrices.

5.4.2 FPGA Processing with Query and Database Segmentations

A query sequence or sequence database contains a varied number of characters. The sequences range in length from several characters to greater than a million characters. A

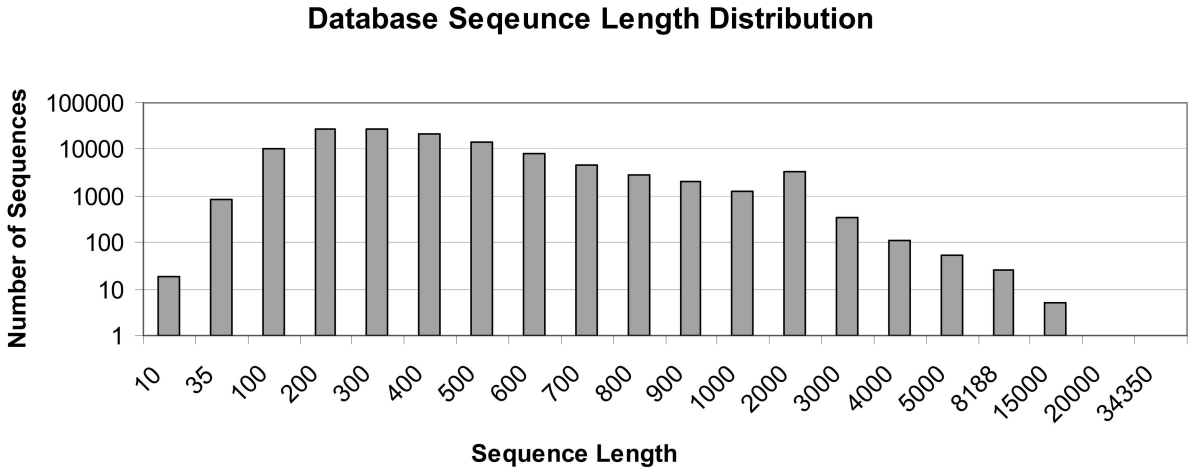


Fig. 8. Distribution of sequence length.

conventional processor can process sequences of any length. However, due to restrictions on the limited size of FPGA on-board RAM, the FPGA coprocessor (Note that this is the restriction due to the size of the FPGA we have used. Similar constraints will exist for other FPGAs.) can process query sequences no more than 1,420 characters long and database sequences no more than 8,188 characters long.

When the master node detects an over-sized query sequence, a simple approach is to assign all sequences to conventional processors and avoid invoking FPGA coprocessors. But we clearly lose the advantages of reconfigurable computing. The master node is designed to automatically segment the query into multiple subqueries, each no longer than 1,420 characters, thereby enabling a similarity search for the FPGA processing for arbitrary length of queries. If the master node detects an oversized query, query segmentation splits a long query sequence with an overlap, and puts segmented subquery sequences into a query pool such that each subquery can fit into the FPGA chip. In order to maintain the original sensitivity and avoid a loss of accuracy in the results, the master node is to apply automatic segmentation to the query sequence using a 1,420 character window with an overlap. First, the master node uses a dispatcher to send the first subquery to the FPGA for processing. Upon receiving the search results, the master assigns the next subquery to the FPGA coprocessor from the pool. This process is repeated until the pool is empty. Then the master node uses a result merger to sort and summarize all subresults into one unified result based on statistical significance. Finally, it prints the top alignments using the original query versus the top scored database sequences. The accuracy analysis of query segmentation is presented in Section 7.

The oversized database sequences can be resolved within a worker node. First, the worker node presorts the length of sequences when loading the database. If there is any sequence that is over the limit of 8,188, then it creates an additional thread to search these long sequences using the conventional processor on the host machine, and the majority of database sequences are still processed using the FPGA coprocessor at the same time.

6 Hardware and Software Platforms

The hardware platform used for the computing platform is based on a SunFire x2100 cluster running the Linux operating system. The cluster consists of one master node, with a dual Core AMD Opteron 275 processor, and 10 worker nodes with gigabit Ethernet connections between nodes. Each worker node has a dual core AMD Opteron 175 processor with 2 GB of RAM.

The FPGA devices are programmable logic devices, which have a matrix of logic cells connected by an interconnection network, surrounded by I/O for data transfer. The FPGA board used for this study was an ADP-WRC-II PCI-board with a Xilinx Virtex II XC2V6000 and 1 GB SDRAM from Alpha Data [2]. Besides the PCI interface (PLX PCI9656) and FPGA device, a clock generator creates the base clock frequency for the board. The FPGA board can be plugged into any PCI-based host computer system. Since the PCI Express interface of worker nodes is not compatible with the standard PCI interface of the FPGA board, we added a 1.9 GHz Pentium IV node with 768 MB memory into the cluster. This node was used as the host machine of the FPGA board.

A modification of the open resource FASTA sequence comparison package (release 3.4) [9] was ported to the above computing platform. The communication layer between nodes is based on MPICH2 [4].

7 Performance Evaluation

We downloaded the protein databases from NCBI [19] and EBI [28]. The yeast.aa database is a FASTA format database containing 2,974,038 characters in 6,298 sequences. The length of sequence in the yeast.aa is between 20 and 4,910 characters. Uniref50 contains 586,687,758 characters in 846,716 sequences and 74 sequences are longer than 8,188. The month.aa database contains 122,650 sequences and 43,531,487 characters, and the sequences range in length from 6 to 34,350 characters. In the month.aa there are six database sequences that are greater than the FPGA processing limit. Fig. 8 shows the segment distribution of sequence lengths for the month.aa database.

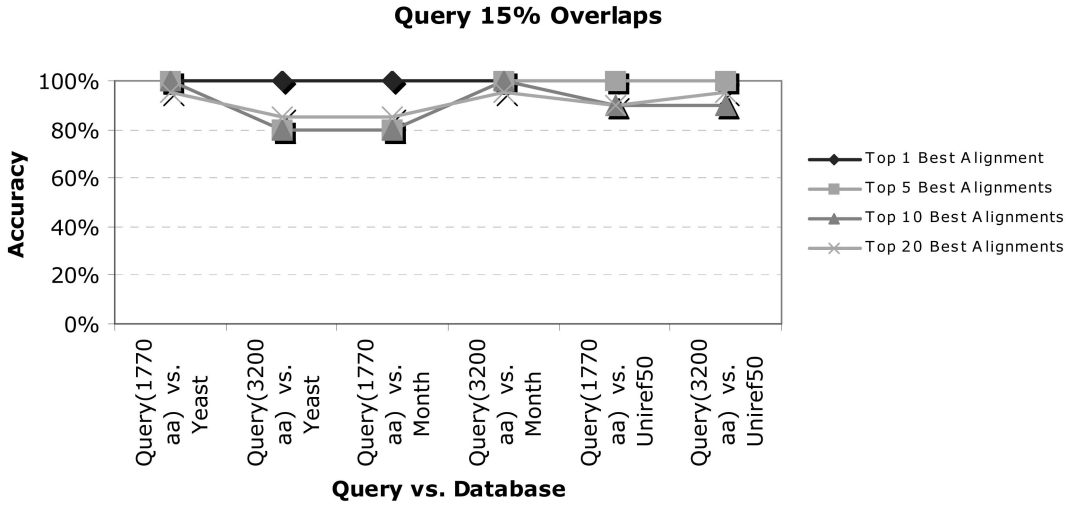


Fig. 9. Accuracy analyses of query segmentation with 15 percent overlaps.

7.1 Query Segmentation Accuracy Analysis

Our implementation splits the query sequences into smaller ones with an overlap to allow examination of the area at the split (Section 5.4.2). The sequence can be split into subsequences of maximum length of 1,420 and each one can be processed on the FPGA coprocessor. The drawback lies in the need to introduce overlaps between the sections of sequence and to manage the results obtained from each subsequence.

Work presented at the level of query sequence segmentation with an overlap exclusively concerns the alignment accuracy [6], [10], [32]. The amount of overlap reflects a trade-off between the system performance and the result accuracy. In our experiments using the query segmentation model, splitting of a single sequence into overlapping smaller subsequences is contrasted with the original sequence against the same database. The long query sequence was split into subsequences of 1,420 characters with an overlap from 0 percent to 40 percent between the subsequences. To better analyze the query segmentation accuracy, we did a best alignment comparison of segmented query with various amounts of overlaps to the original query search Accuracy comparison of query segmentation with 15 percent overlaps is shown in Fig. 9.

As we increase the proportion of overlaps from 0 percent to 25 percent, the accuracy of the results increases. A 100 percent accuracy is observed for the top 1 best alignment sequence by searching the overlapped query segments against the databases. The query segmentation with an overlap can obtain a higher pairwise alignment accuracy over the same operation without any overlaps. Our tests show that 15 percent to 25 percent overlaps can achieve over 80 percent accuracy and sensitivity from top 1 to top 20 best alignments. In particular, if the query sequence hits many high similarity sequences in the database, the overlapped query segmentation method can achieve 90 percent-100 percent accuracy with the 5 percent to 15 percent overlaps. Users can specify the size of overlap depending on the number of top alignments they require. Computing redundancy only exists for the pairs in the overlaps of subsequence. Compared to the nonoverlapping segmentation, the additional computing time grows marginally with the increasing size of overlaps.

7.2 Overall System Performance

Due to the limited availability for FPGA boards, only one worker node was enhanced with an FPGA coprocessor. The FPGA board was configured to contain 119 affine PEs. We performed a comprehensive performance evaluation based on the serial implementation on a single Pentium IV 1.9 GHz processor without the SSE2 acceleration, the serial implementation on a single Opteron processor without the SSE2 acceleration, the serial implementation on a single Pentium processor with the SSE2 acceleration, the serial implementation on a single Opteron processor with the SSE2 acceleration, and the MPI implementation with a single FPGA coprocessor. Fig. 10 plots the heterogeneous computing system speedups of various configurations, comparing the above five different configurations.

With various query sequences against the month.aa database in Fig. 10, the peak performance is obtained for the query of length 1,223, yielding a speedup factor of nearly 60 using 1 FPGA coprocessor over a Pentium processor without the SSE2 acceleration. We also notice that the performance decreases when the length of query sequences is over 1,420. For the sample query length of 1,770, it has to be preprocessed and split into two subqueries, then sent to the FPGA for processing. The FPGA coprocessor loads the database twice, and therefore, increases the overhead and decreases the performance. Nevertheless, it is still over $54\times$ faster than the Pentium version.

In Fig. 11, we can see the heterogeneous computing platform performance by comparing the three varied-length sequences to two databases for various configurations and numbers of worker processors. All of MPI tests were done using one master processor or one worker processor or one FPGA coprocessor with multiple workers. As the number of worker processors increases from 1 to 16, the MPI speedup increases as well, and the FPGA+SSE2+MPI acceleration achieves up to $110\times$ speedups over a serial C implementation on a single Pentium IV 1.9 GHz processor. We also did a performance comparison between homogeneous and heterogeneous implementations in Fig. 12. The experimental results demonstrate the effectiveness of our heterogeneous computing platform, which accelerates the

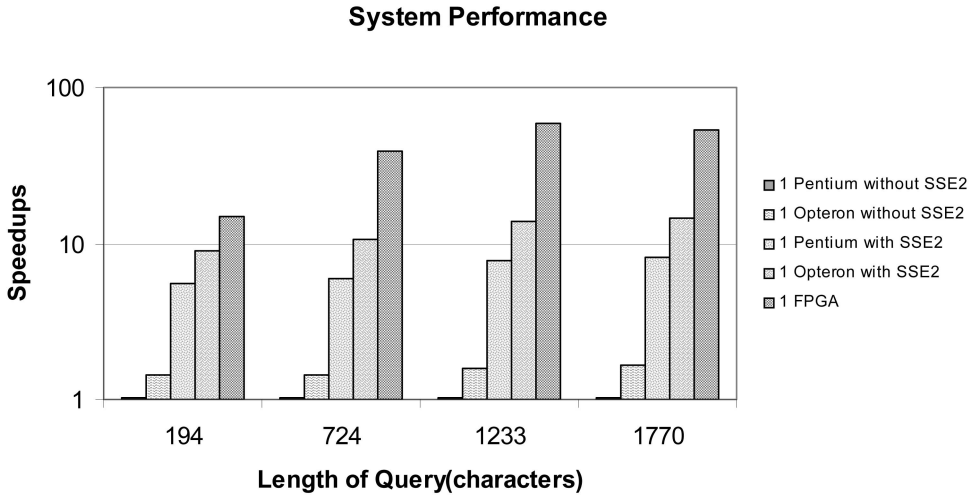
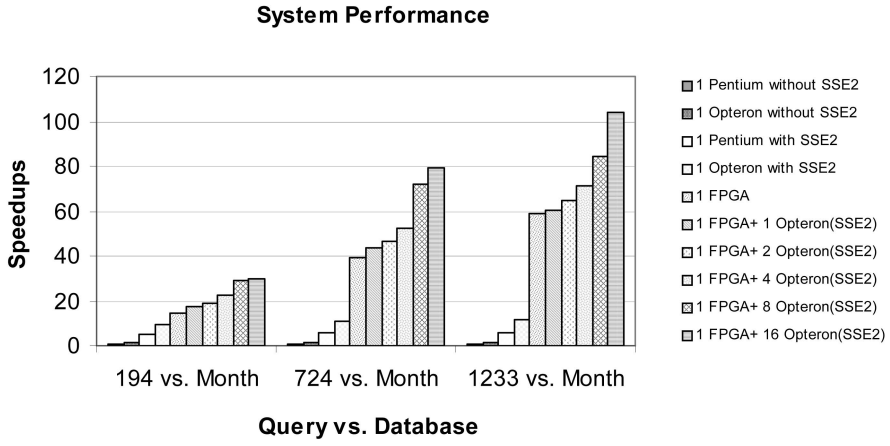
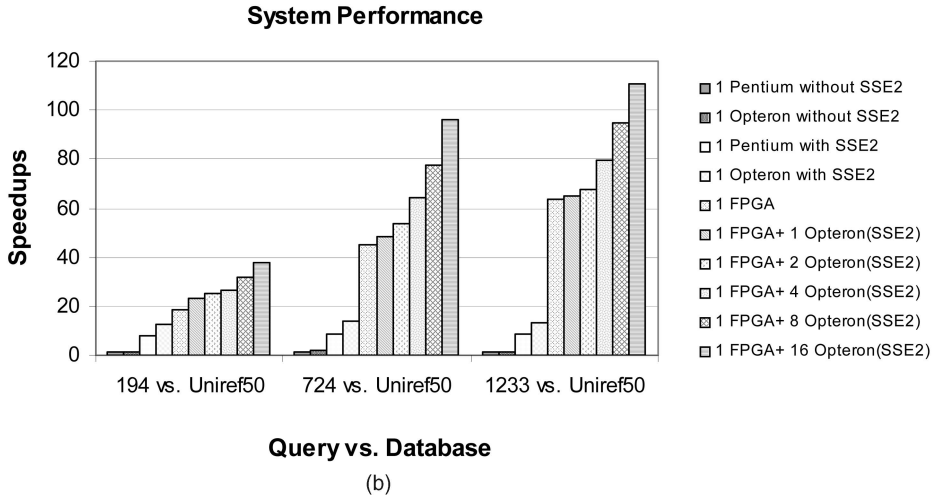


Fig. 10. System speedup of various query lengths for various configurations.



(a)



(b)

Fig. 11. Speedup comparison for various configurations.

Smith-Waterman biological sequence database search by at least 16-fold leap with one hardware accelerator. However, the homogeneous search shows a maximum of 13× speedups while using 16 worker processors without any acceleration. The hybrid computing effectively boosts the computation, in comparison with the performance of the heterogeneous implementation.

A performance measure commonly used in computational biology is million cell updates per second (MCUPS). A MCUPS represents the time for a complete computation of one million entries of the Smith-Waterman computational matrix, including all comparisons, additions and maxima computations. We tested the performance that was calculated as the total number of cells divided by searching time.

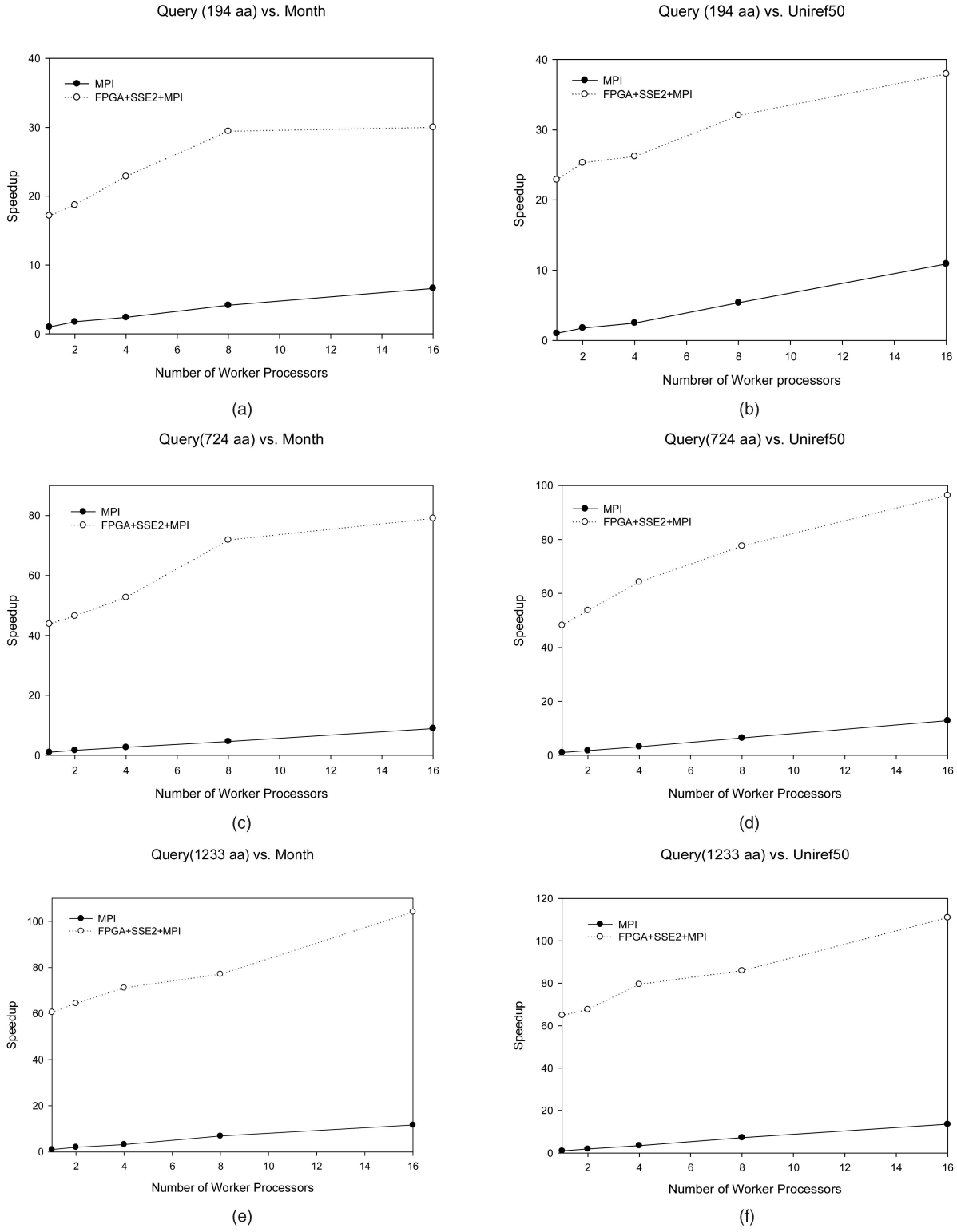


Fig. 12. Performance comparison of heterogeneous and homogeneous computing.

For the comparison of other state-of-the-art implementations, we have evaluated performance of different massively parallel machines [8], [10], [15], [23]. Our implementation achieved the real performance, a total of 11,129 MCUPS, which is about four times faster than Farrar's solution [8]. CELL BE, GPU, and Kestrel are solutions based on one- or two-board accelerators. CELL BE [23] is 4 times slower, GPU [15] is 6 times slower, and Kestrel [10] is 28 times slower than our system.

Overall, we see that long query sequence and large database sizes exhibit better performance. For instance, the longer query sequence outperforms the shorter query. This is due to the high utilization of the FPGA coprocessor resource. It clearly demonstrates that the FPGA coprocessor

has the ability to exploit the huge amount of inherent parallelism of the Smith-Waterman algorithm and perform hundreds or even thousands of operations simultaneously. The hybrid system architecture builds high-performance reconfigurable computing platform, which is based on a combination of conventional processors and FPGA coprocessors through the MPI-enabled coherent enterprise network infrastructure. It also illustrates that the hybrid-computing platform is a more efficient architecture for the high throughput sequence similarity database searches.

8 Conclusions

In this paper, we have shown that the FPGA and MPI-enabled heterogeneous computing platforms can achieve supercomputer performance. The experimental results demonstrate the effectiveness of our heterogeneous computing platform, which accelerates the Smith-Waterman biological sequence database search by exploiting reconfigurable logic, fine-grained instruction parallelism, and coarse-grained data parallelism within a given architecture. Our proposed hybrid architecture allows us to exploit coarse-grain parallelism through traditional parallel processing on conventional processors as well as fine-grain parallelism through direct hardware execution on FPGAs. Moreover, the flexibility of the heterogeneous computing framework can quickly adapt to the next generation of accelerators, like GPUs [15] and Cell BE [1], [23], [31], for higher level of performance.

We expect to obtain better performance by adding multiple FPGAs into the enterprise computing infrastructure. Furthermore, our proposed heterogeneous computing platform has the potential to deliver unprecedented performance for accelerating additional data and computation intensive bioinformatics applications, such as Multiple Sequence Alignment, while overcoming the barriers that can limit conventional processors.

Acknowledgments

The authors would like to thank Progeniq Pte. Ltd. [22] for providing them with the FPGA software in this study. They would also like to thank Peter Chen for helping them with the hardware setup.

References

[1] A.M. Aji, W. Feng, F. Blagojevic, and D.S. Nikolopoulos, "Cell-SWat: Modeling and Scheduling Wavefront Computations on the Cell Broadband Engine," *Proc. Fifth ACM Int'l. Conf. Computing Frontiers,* pp. 13-22, May 2008.
[2] Alpha-Data, http://www.alpha-data.com, 2010.
[3] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman, "Basic Local Alignment Search Tool," *J. Molecular Biology,* vol. 215, pp. 403-410, 1990.
[4] Argonne National Laboratory, MPICH2—A Portable Implementation of MPI, http://www-unix.mcs.anl.gov/mpi/mpich/, 2010.
[5] Cray XD1, Cray Inc., http://www.cray.com, 2010.
[6] O. Creţ et al., "FPGA-Based Scalable Implementation of the General Smith-Waterman Algorithm," *Proc. IASTED Parallel and Distributed Computing Symp.,* pp. 410-415, 2004.
[7] A. Darling, L. Carey, and W. Feng, "The Design Implementation, and Evaluation of mpiBLAST," *Proc. ClusterWorld Conf. & Expo,* 2003.
[8] M. Farrar, "Striped Smith-Waterman Speeds Database Searches Six Times over other SIMD Implementations," *Bioinformatics,* vol. 23, pp. 156-161, 2007.
[9] FASTA, http://fasta.bioch.virginia.edu/, 2010.
[10] L. Grate, M. Diekhan, D. Dahle, and H. Hughey, "Sequence Analysis with the Kestrel SIMD Parallel Processor," *Pacific Symp. Biocomputing,* vol. 6, pp. 263-274, 2001.
[11] O. Gotoh, "An Improved Algorithm for Matching Biological Sequences," *J. Molecular Biology,* vol. 162, pp. 705-708, 1982.
[12] P. Guerdoux-Jamet and D. Lavenier, "SAMBA: Hardware Accelerator for Biological Sequence Comparison," *Bioinformatics,* vol. 13, pp. 609-615, 1997.
[13] D.T. Hoang, "Searching Genetic Databases on Splash 2," *IEEE Workshop FPGAs for Custom Computing Machines,* pp. 185-191, 1993.
[14] IA-32 Intel Architecture Software Developer's Manual, Volume 1: Basic Architecture, 2004.
[15] S. Manavski and G. Valle, "CUDA Compatible GPU Cards as Efficient Hardware Accelerators for Smith-Waterman Sequence Alignment," *BMC Bioinformatics,* vol. 9, suppl 2, p. S10, 2008.
[16] X. Meng and V. Chaudhary, "Bio-Sequence Analysis with Cradle's 3SoC Software Scalable System on Chip," *Proc. ACM Symp. Applied Computing,* pp. 202-206, 2004.
[17] X. Meng and V. Chaudhary, "Optimized Fine and Coarse Parallelism for Sequence Homology Search," *Int'l J. Bioinformatics Research and Applications,* vol. 2, no. 4, pp. 430-441, 2006.
[18] X. Meng and V. Chaudhary, "Improving Data Throughput for FPGA-Based Sequence Database Similarity Searches Using an Adaptive Buffering Scheme," *J. Parallel Computing,* vol. 35, pp. 1-11, 2009.
[19] NCBI FTP site, ftp://ftp.ncbi.nih.gov/blast/db/FASTA/, 2010.
[20] T. Oliver, B. Schmidt, and D. Maskell, "Hyper Customized Processors for Bio-Sequence Database Scanning on FPGAs," *Proc. ACM/SIGDA 13th Int'l Symp. Field-Programmable Gate Arrays,* pp. 229-237, 2005.
[21] W.R. Pearson, "Searching Protein Sequence Libraries: Comparison of the Sensitivity and Selectivity of the Smith-Waterman and FASTA Algorithms," *Genomics,* vol. 11, pp. 635-650, 1991.
[22] Progeniq Pte. Ltd., http://www.progeniq.com/, 2010.
[23] V. Sachdeva, M. Kistler, E. Speight, and T. Tzeng, "Exploring the Viability of the Cell Broadband Engine for Bioinformatics Applications," *Proc. Parallel and Distributed Processing Symp.,* pp. 1-8, 2007.
[24] B. Schmidt, H. Schroder, and M. Schimmler, "Massively Parallel Solutions for Molecular Sequence Analysis," *Proc. Int'l Parallel and Distributed Processing Symp.,* pp. 186-193, 2002.
[25] I. Sharapov, "Computational Application for Life Sciences on Sun Platforms: Performance Overview," White Paper, 2001.
[26] T.F. Smith and M.S. Waterman, "Identification of Common Molecular Subsequences," *J. Molecular Biology,* vol. 147, pp. 195-197, 1981.
[27] TimeLogic Corporation, www.timelogic.com, 2010.
[28] Uniref Database, http://www.ebi.ac.uk/uniref, 2010.
[29] J. Walter, X. Meng, V. Chaudhary, T. Oliver, L. Yeow, B. Schmidt, D. Nathan, and J. Landman, "MPI-HMMER-Boost: Distributed FPGA Acceleration," *J. VLSI Signal Processing Systems for Signal, Image, and Video Technology,* vol. 48, no. 3, pp. 223-238, 2007.
[30] M. Warren, E. Weigle, and W. Feng, "Green Density: A 240-Node Beowulf in One Cublic Meter," *Proc. Supercomputing (SC),* 2002.
[31] A. Wirawan, C.K. Kwon, N.T. Hieu, and B. Schmidt, "CBESW: Sequence Alignment on the Playstation 3," *BMC Bioinformatics,* vol. 9, p. 377, 2008.
[32] Y. Yamagucchi, T. Maruyama, and A. Konagaya, "High Speed Homology Search with FPGAs," *Proc. Seventh Pacific Symp. Biocomputing,* vol. 7, pp. 271-282, 2002.
[33] C.W. Yu, K.H. Kwong, K.H. Lee, and P.H.W. Leong, "A Smith-Waterman Systolic Cell," *Proc. 13th Conf. Filed-Programmable Logic and Applications,* pp. 375-384, 2003.

Xiandong Meng received the MS and PhD degrees in computer engineering from Wayne State University, Detroit, Michigan. He is currently a senior staff member in the Supercomputing Facility at Texas A&M University, College Station. He did extensive research in accelerating the bioinformatics algorithms and published papers in prestigious journals and conferences. His research interests include high-performance computing, high-throughput bioinformatics computing, and reconfigurable computer architecture. He is a senior member of the IEEE.

Vipin Chaudhary received the MS and PhD degrees in computer science and electrical and computer engineering from the University of Texas at Austin. He is an associate professor in the Department of Computer Science and Engineering, NYS Center of Excellence in Bioinformatics and Life Sciences at University at Buffalo, the State University of New York. He has authored/coauthored more than 100 publications. His current research interests include chip multiprocessor architectures, computer assisted surgery, medical imaging, biomedical engineering, and high-performance bioinformatics. He is a member of the IEEE.