Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Boosting data throughput for sequence database similarity searches on FPGAs using an adaptive buffering scheme

X. Meng [a,*], V. Chaudhary [b]

[a] Supercomputing Facility, Texas A&M University, College Station, TX 77843, USA
[b] Department of Computer Science and Engineering, NYS Center of Excellence in Bioinformatics and Life Sciences, University at Buffalo, SUNY, Buffalo, NY 14260, USA

A R T I C L E   I N F O

Article history:
Received 5 December 2006
Received in revised form 7 February 2008
Accepted 16 September 2008
Available online xxxx

Keywords:
FPGA
PCI
Direct memory access
Smith–Waterman algorithm
Data prefetching

A B S T R A C T

Searching on DNA and protein databases using sequence comparison algorithms has become one of the most powerful techniques to better understand the functionality of particular biological sequences. However, the requirements to process the biological data exceed the ability of general-purpose processors. FPGAs (Field Programmable Gate Arrays) connected to server processors have been used to accelerate similarity searches. However, reconfigurable computing platforms have utilized an external I/O bus as the communications channel, limiting the communication speed between the host processor and the FPGA. This communication bottleneck often offsets the application speedup enabled by FPGAs. In this paper we present an adaptive data prefetching scheme to avoid reconfigurable processing coprocessor stalls due to data unavailability through profiling methodologies and quantitative analysis. Experimental results on various query sequences show that the proposed scheme can effectively eliminate a major portion of the data access penalty, increase throughput of the FPGA implementation by up to 42%, and achieve a speedup of 110 for affine gap penalties over a standard PC implementation.

Published by Elsevier B.V.

1. Introduction

The amount of biological sequences available in databases has been growing exponentially over the past several years. Although computing power has been growing at an exponential pace, the requirements to process the biological data far outstrips the ability of traditional computing to meet the challenge of converting the data into information or knowledge. At this point, sequential computing, that is, a single general-purpose processor, can allow only a small part of the massive, multi-dimensional biological information to be processed. Under this scenario, the process of data analysis and understanding of the data-described biological processes could remain incomplete, causing us to lose vast quantities of valuable information because CPU-power and time constraints could fail to follow critical events and trends.

The algorithms for biological sequence database search can be implemented to run efficiently on various types of hardware with the ability to perform several operations simultaneously. There is a wide range of different hardware available on which the algorithms have been implemented. Hughey [12] has reviewed various types of hardware that can be used and its performance. The hardware can be divided into general-purpose processors, which can be used for many different types of computations, and hardware devices specifically designed for performing sequence alignments and database searches. Reconfigurable computing hardware, a special-purpose processor, takes advantage of programmable logic devices such as FPGAs that can be reconfigured or reprogrammed to implement specific functionality more suitably and more efficiently than on a general-purpose processor.

* Corresponding author.
E-mail addresses: x-meng@tamu.edu (X. Meng), vipin@buffalo.edu (V. Chaudhary).

0167-8191/$ - see front matter Published by Elsevier B.V.
doi:10.1016/j.parco.2008.09.010

Bio-sequence database searches based on FPGA reconfigurable hardware platforms have gained popularity recently. The hyper customized FPGA implementation [17], which performs the Smith–Waterman [23] database searches for both linear and affine gap penalties, has reported tremendous speedup as compared to a standard desktop PC. However, the data communications between the FPGA board and the host system with the large volume of data involved in these kinds of applications impact the performance. We have developed an adaptive parallel data prefetching scheme [16] for the execution of Smith–Waterman sequence database searches to alleviate the communication bottleneck leading to a substantial performance improvement.

The rest of the paper is organized as follows: the next section lists related work; Section 3 gives some basic background information on the Smith–Waterman algorithm and FPGA coprocessing, along with an analysis of impacts of the latency and bandwidth between the host processor and the FPGA; Section 4 describes the design of the adaptive data prefetching scheme and discusses the implementation of the reconfigurable hardware platform; Section 5 analyzes the adaptive prefetching performance results and explores how to eliminate the side effects that prefetching may identify; a summary and future work are presented in Section 6.

2. Related work

A number of different designs for special-purpose hardware for performing sequence alignments and database searches have been proposed and implemented. Their advantage over general-purpose computers is that they can be tailored specifically to perform sequence comparisons at a high speed, while their disadvantage is their high cost. Special-purpose hardware is usually built using either FPGA or custom ASIC (application specific integrated circuit) technology. The advantage of FPGA is that they are reprogrammable and work with different algorithms [17,18], while ASIC is customarily designed to a very specific purpose and cannot be changed.

Because each implementation is designed based on different hardware and uses different parameter setting, direct performance comparison is quite difficult. We list a number of parallel architectures that have been specifically designed or taken advantage of their unique features for sequence database searches.

Paracel [20] used a custom ASIC approach to do the sequence alignment. Their system used 144 identical custom ASIC devices, each containing approximately 192 PEs (processing elements) [26]. Starbridge [24] has developed a reconfigurable computing system using FPGAs, which can deliver 10–100 times or greater improvement in computational efficiency. The system employs a dual processor motherboard and a single Hypercomputer board with nine Xilinx XC2V6000-BG1152 Virtex-II FPGAs and two XC2V4000-BG1152 Virtex-II FPGAs. Splash-2 [7] and SAMBA [11] compute simple edit distances for both 2 and 4 bit alphabet based upon FPGA technology; however, they are not flexible enough for a general case of protein sequence searches with full 8-bit substitution matrix. Solutions based on systolic array parallelization and the SIMD (single instruction multiple data) concept are presented in Kestrel [3] and Fuzion [22]. The FPGA implementation [6,10,25] can provide a compact and high performance design. But the above solutions allow for neither affined gap penalties nor local alignment.

The Cray XD1 [5] supercomputer is a high performance computing platform combining scalar processing with reconfigurable computing technology. The Cray XD1 Smith–Waterman solution has performed 28 times faster when tested against other 64-bit AMD Opteron processor-based solutions running the same application.

Some other examples are MMX implementation [19] on common microprocessors, DSP implementation [14] on a Cradle 3SoC chip, and SSE2 implementation [15] on a Linux cluster using Intel Pentium processors. These implementations exploit SIMD instruction sets of general-purpose architectures or a combination of SIMD and cluster computing to achieve parallelism.

3. Smith–Waterman algorithm and FPGA

3.1. Smith–Waterman database searches

The comparison of biological sequences is a fundamental task in molecular biology. The Smith–Waterman algorithm [23] is perhaps the most widely used local similarity algorithm for biological sequence comparison. In 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. In Eqs. (1) and (2), A, B are the query and database sequences being compared, respectively; ai is the ith letter in A, bj is the jth letter in B; matrices H, E and F are of size $(|A|+1)(|B|+1)$, and row and column 0 are initialized to 0 in H; $SS(ai,bj)$ is the similarity of ai and bj as read from an amino acid substitution score matrix. Matrix entries are calculated using the following recurrences:

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

where

$$E[i,j] = \max\{H[i-k,j] - g(k)\}, \quad \text{for } 0 < k < i,$$
$$F[i,j] = \max\{H[i,j-l] - g(l)\}, \quad \text{for } 0 < l < j. \tag{2}$$

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 end at the same position but contain a gap in sequence A or B, while $g(k)$ and $g(l)$ are the gap penalty functions. The computations should proceed with i going from 1 to m and j going from 1 to n.

The Smith–Waterman algorithm exhibits strong data dependency between neighboring cells in the computation matrix because the value of H in any cell in the alignment matrix cannot be computed before all cells to the left or above it have been computed. The algorithm uses memory space proportional to the product of the lengths of the two sequences, mn, and the computing time complexity is $O(mn(m+n))$. Gotoh [9] reduced the time needed by the algorithm to $O(mn)$ when affined gap penalties of the form $g(k) = q + rk$; ($q \geqslant 0, r \geqslant 0$) are used, where q is the gap open penalty and r is the gap extension penalty. The overall optimal alignment score is equal to the maximum value of $H[i,j]$.

In the biological sequence database searches, we are particularly interested in the four standard nucleotide bases adenine (A), thymine (T), guanine (G), and cytosine (C), and the 20 amino acids, which are A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, and Y. Fig. 1 shows an example of the Smith–Waterman alignment between two DNA sequences ACACACTAG and AGCACACA. The similarity matrix uses the linear gap penalty $q = r = 1$, and a substitution score +2 if two characters match, and -1 otherwise. The arrows show a trace-back path, which leads to the corresponding the best local alignment output, starting from the highest score to zero.

3.2. FPGA coprocessing

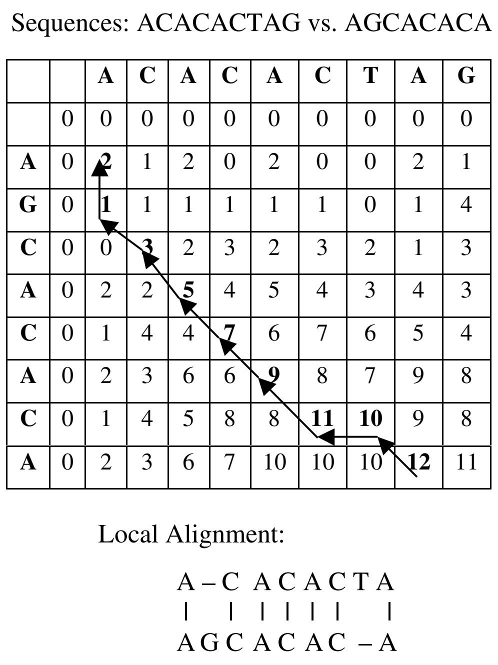
The FPGA device acts as a highly flexible coprocessor, which has to be integrated into a host system. The FPGA device is utilized for the parallel execution of the Smith–Waterman algorithm by a connection between the FPGA board and the processor and memory of the host computer. Database sequences are transferred from the host memory to the FPGA board and processed alignment scores are written back to the host memory for output or further processing.

The Smith–Waterman algorithm has been mapped to a linear systolic array [26] of PEs. The systolic array is composed of identical PEs. The characters of the query sequence are preloaded into each PE, while the database characters are fed serially into the array in Fig. 2. Thus, a comparison between pairs of characters, which applies the algorithm's equations, is done on each machine cycle. A database array of length m can be compared to an input query sequence of length n in $O(m + n - 1)$ steps.

3.3. System analysis

We assume the system has no data prefetching and results post-processing, thus, the real time, T_{total}, is

$$T_{total} = T_{fpga} + T_{dbloading}. \tag{3}$$



Sequences: ACACACTAG vs. AGCACACA

		A	C	A	C	A	C	T	A	G
	0	0	0	0	0	0	0	0	0	0
A	0	**2**	1	2	0	2	0	0	2	1
G	0	**1**	1	1	1	1	1	0	1	4
C	0	0	**3**	2	3	2	3	2	1	3
A	0	2	2	**5**	4	5	4	3	4	3
C	0	1	4	4	**7**	6	7	6	5	4
A	0	2	3	6	6	**9**	8	7	9	8
C	0	1	4	5	8	8	**11**	**10**	9	8
A	0	2	3	6	7	10	10	10	**12**	11

Local Alignment:

```
A – C  A C A C T A
|   |  | | | |   |
A G C  A C A C – A
```

Fig. 1. Example of Smith–Waterman alignment of two sequences.

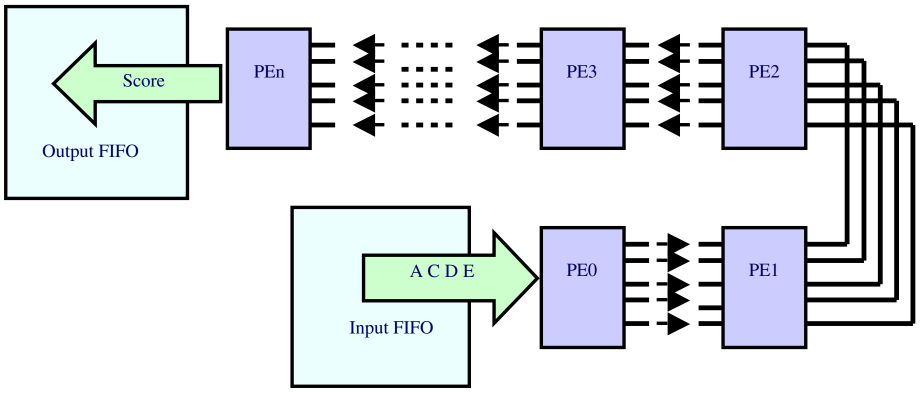


Fig. 2. Diagram for linear systolic processor array figure.

As aforementioned in Section 3.1, the FPGA coprocessor has been utilized to calculate only the optimal alignment score *H*[*i*,*j*]. The theoretical FPGA hardware execution time, T_{fpga}, is

$$T_{fpga} = T_{init} + DB_{size}/PCI_{bandwidth} + (M_{db} + N_{query} - 1) \times t_{processing} + T_{close}, \tag{4}$$

where

T_{init} is the hardware initialization latency which takes approximately 80 ms over the PCI bus,
DB_{size} is the total size of database in bytes,
$PCI_{bandwidth}$ is the bandwidth of the PCI bus,
M_{db} in the total number of database characters,
N_{query} is the number of query characters,
$t_{processing}$ is the hardware processing time for a comparison between a pair of characters, and
T_{close} is the final hardware close latency.

We now analyze the database sequence transfer. The average sequence length in the nr database is 342 amino acids. From the perspective of the data transfer, the average input-to-output ratio is 342 to 1 for processing a single database sequence. Due to the large size of databases, the database loading time, which transfers database characters from the disk to the application buffer on the host, takes a major portion of the total execution time. Database loading time, $T_{dbloading}$, can be decomposed into four parts:

$$T_{dbloading} = T_{db\_open} + T_{unpack\_db} + T_{pack\_fifo} + T_{db\_close}, \tag{5}$$

where

T_{db_open} is the database sequence file open latency,
T_{unpack_db} is the time of unpacking the database sequences,
T_{pack_fifo} is the time of packing the database sequences into the FIFO buffer, and
T_{db_close} is the database sequence file close latency.

4. Adaptive data prefetching scheme design and implementation

Data prefetching [2,4,8] has been receiving considerable attention in high performance prcessors as a potential means of boosting performance. Prefetching can also be an effective means of accelerating system time for acceleration based computing.

4.1. Execution profile analysis

The total execution time of an FPGA-based biological sequence database search can be de decomposed into three primary parts: database sequence load time on the host machine, Smith–Waterman processing time on FPGA, and other time including data initialization and result output. In order to analyze how each part contributes to the total program execution time, we carried out a timing profile analysis, as shown in Fig. 3, by comparing various query sequences to a 1 GB section of the compressed nr protein database [21], which contains 936,896,903 characters in 2,739,534 sequences. The measurements were taken on a 1.9 GHz Pentium IV processor with 768 MB memory, and an ADP-WRC-II FPGA PCI-board (Fig. 4) with a

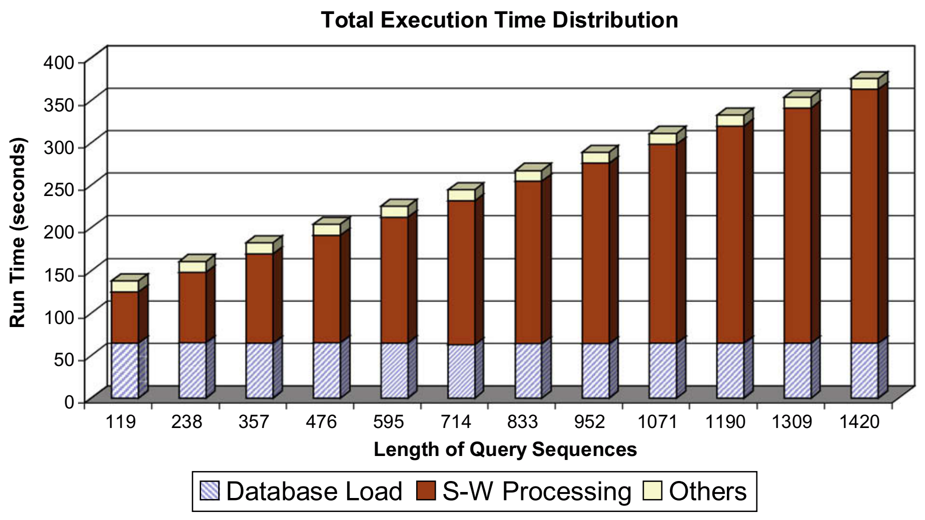


Fig. 3. Total execution time of various query sequence lengths.

Xilinx Virtex II XC2V6000 from Alpha Data [1]. The board contains 119 affine PEs and accepts the query sequence up to 1420 characters. All experimental results we obtained were based on the above hardware configuration in this study.

Based on these measurements, we conclude that the execution time is dominated by FPGA processing for most cases. Fig. 3 shows that the FPGA processing time increases proportionally with the increase in query sequence size. Furthermore, the database sequence load time remains constant and independent of the query sequence size. However, the database loading time decreases from 56% to 20% of the total execution time when the query size increases from 119 to 1420. This is very promising for absorbing the database load time using current processing and is exploited in our implementation as shown later.

4.2. Data pipe architecture

FPGAs are programmable logic devices which have a matrix of logic cells connected by an interconnection network, surrounded by I/O for all kinds of data transfer. Besides the PCI interface (PLX PCI9656) and FPGA device (Xilinx Virtex II XC2V6000) in Fig. 4, a clock generator creates the base clock frequency for the board. Another important component on an FPGA board is the local memory (SDRAM) that stores data. The board has a 64-bit 66 MHz PCI bus, which provides data transfer rates of up to 528 MB/s. The FPGA coprocessor board can be plugged into any PCI-based host computer system.

DMA (direct memory access) is an efficient way to transfer a block of data between the host computer's memory and FPGA's memory with as little burden on the CPU as possible. Bus-mastering PCI devices contain dedicated logic for performing DMA transfers. To perform a DMA transfer, the CPU first programs the PCI device's registers instructing it on where to transfer the data, how much data to transfer and which direction the data should travel in. It then initiates the DMA transfer, and typically, the CPU is interrupted by the device once the transfer has been completed. The advantage of DMA then, is that the CPU can perform other tasks while the PCI device performs the data transfer.

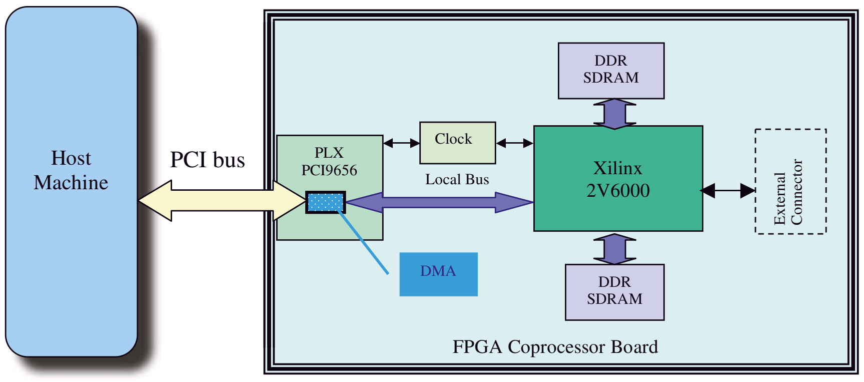
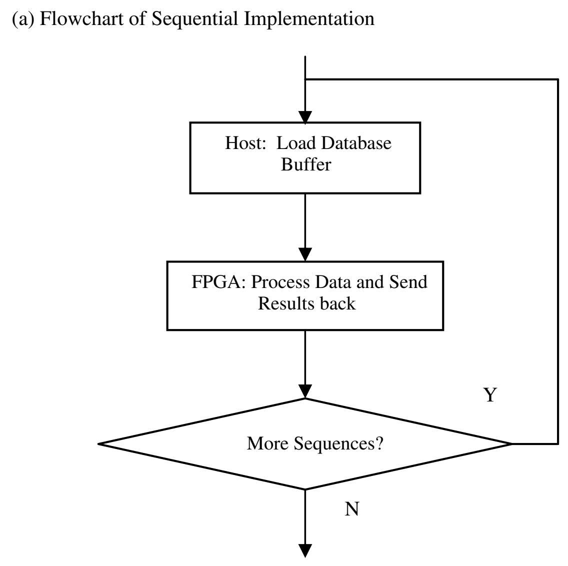


Fig. 4. Reconfigurable computing platform structure.

The local bus protocol of a DMA-initiated burst is efficient for throughput. DMA controls both bus interfaces during data transfer. It has two independent channels providing a flexible prioritization scheme and each channel has its own bi-directional 256 byte deep FIFO. Using DMA enables the data transfers and the processing to be accomplished in


(a) Flowchart of Sequential Implementation
Host: Load Database Buffer
FPGA: Process Data and Send Results back
More Sequences?
Y
N

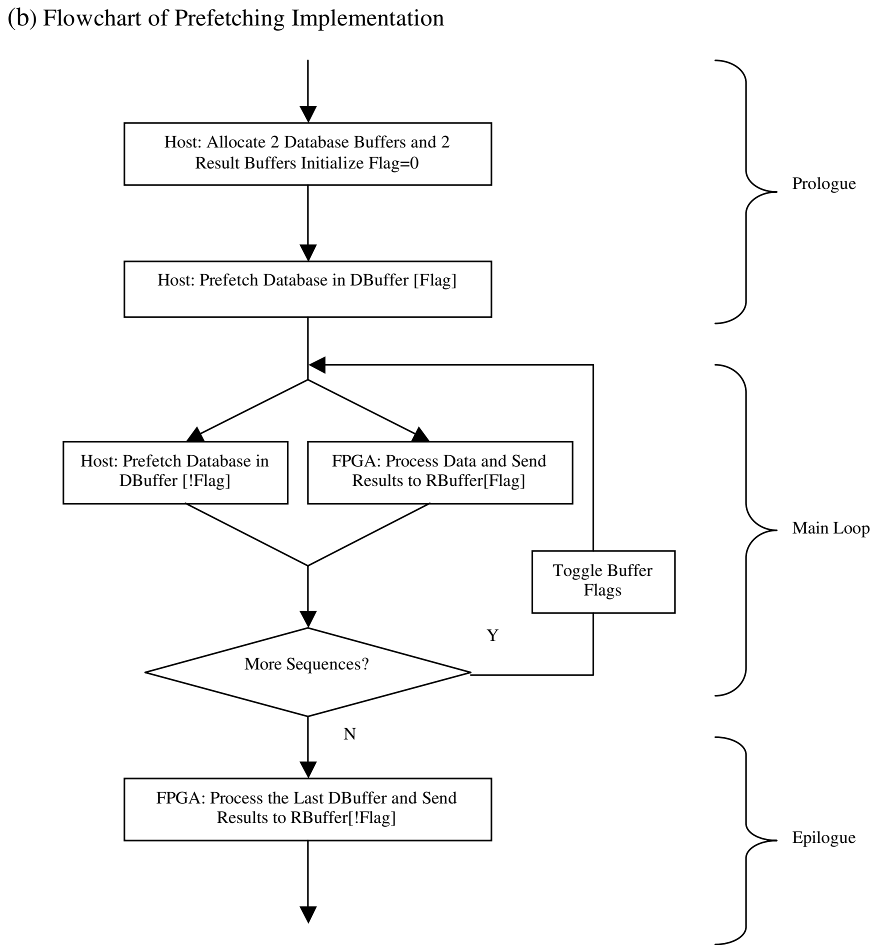

(b) Flowchart of Prefetching Implementation
Host: Allocate 2 Database Buffers and 2 Result Buffers Initialize Flag=0
Host: Prefetch Database in DBuffer [Flag]
Prologue
Host: Prefetch Database in DBuffer [!Flag]
FPGA: Process Data and Send Results to RBuffer[Flag]
Toggle Buffer Flags
Main Loop
More Sequences?
Y
N
FPGA: Process the Last DBuffer and Send Results to RBuffer[!Flag]
Epilogue

Fig. 5. Flowcharts of sequence database search.

parallel. The sequence transfer is organized in such a way that the use of DMA resources is optimized to achieve the shortest possible communication time. Furthermore, it is easier to obtain reliable and reproducible execution time with DMA data transfers.

4.3. Data prefetching scheme

The data transfer requires low latency and high bandwidth to ensure that the communication does not adversely affect the processing time. Biological sequence searches on an FPGA-based coprocessor board require database sequences to be stored in an application buffer of the host machine, and then transferred to the FPGA board for processing, because the amount of on-board memory is usually very limited. Loading entire sequence databases, especially with the large volume of biological sequences, into the memory of the host machine causes significant performance drawbacks. Even though the existing implementation adopted the DMA transfer, database load on the host side and computation on the FPGA side are still in serial order.

To overcome this drawback, we designed a double buffering parallel implementation using DMA on the FPGA board and Pthreads [13] on the host machine. Pthreads is to realize potential program performance gains by exploiting parallelism on the conventional processor. 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 simultaneously in the other application buffer utilizing the full FPGA power. Once processing is done in a local buffer, the FPGA coprocessor sends results back to the host and retrieves new sequences from another buffer and starts computation immediately with the least idle time. The host sequence database buffers are filled out by the host machine during the FPGA processing. The comparisons of original design and our double buffering design flowcharts are shown in Fig. 5.

Our implementation takes advantage of DMA transfers, which are performed by the PCI device for large blocks of data, between the host CPU and the FPGA board. The local bus bridge, PCI9656, in an FPGA board contains multiple DMA engines. Application software running on the host system can control these DMA engines for the rapid data transfer and processing to and from the FPGA using send/receive threads. At the same time, the host software creates additional threads to load database sequences into application buffers in the background. Thus, the transformation overlaps the communication time with the computation time in parallel loops to effectively hide the latency of the database sequence transfer time.

4.4. Scheduling prefetches and data prefetching

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. The scheduling solution is able to resolve the imbalance between data load and processing time by providing a more flexible prefetching mechanism, e.g., the adaptive prefetching buffer size. It also seeks to minimize the chance that a buffer will be prefetched falsely, overwriting the data buffer that is actually used next.

Since the proportions of database load and FPGA processing time keep changing with the varied query inputs, the choice of how to select a proper buffer size for data transfer could be critical to performance. A static or improper buffer size may decrease performance and bring additional communication overhead to the system. A data prefetching scheme would be ineffective unless a proper buffer size is selected.

In order to exploit the seamless transformation between loading and processing, we designed and implemented a query-based data prefetching adaptation algorithm based on the length of the query sequence. The idea is that the short query would require small buffers to overcome database loading delay, and the long query sequence would require large buffers. The communication overhead would be eliminated as much as possible when buffer size reaches a certain point. We discuss experimental results on how to dynamically determine the appropriate buffer size in the next section.

5. Performance analysis

5.1. FPGA processing time

If m and n are the lengths of the database sequences and the query, respectively, then as we explained in Section 2, the time complexity of a serial implementation of the Smith–Waterman algorithm is $O(mn)$. For a fixed database, the total PC execution time shows a linear increase with the query size m, as shown in Fig. 6. With the FPGA processing, the total execution time changes to $O(f(m)n)$, where $f(m)$ is a stair function related to query size m. It takes almost the same amount of clock cycles for FPGA coprocessor to process database sequences if the length of query sequence falls into one particular length range [17]. The reason for this is that all the PEs work simultaneously even though some PEs have no assignments.

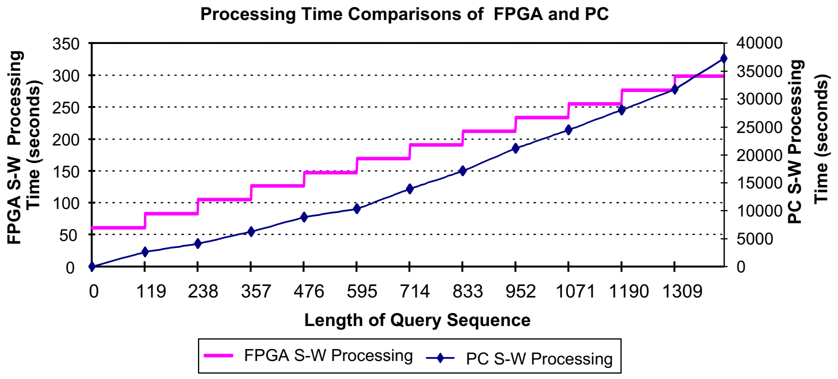


Fig. 6. FPGA vs. PC processing time comparisons (S–W denotes Smith–Waterman).

5.2. Adaptive prefetching performance

Experimental results, which are shown in Fig. 7, give us the comparative performance results with various double database sequence buffer sizes in MB. The performance is evaluated over the range of buffer size from 13 MB to 325 MB, which can hold approximately 10,000–250,000 database sequences. Particularly, compared to the original design which uses a static single buffer regardless of the lengths of the input queries, the performance of the double buffer implementation is enhanced for most cases, e.g., from a buffer size of 52–286 MB. But, the improvement in prefetching performance saturates at a certain point. Thus, the appropriate buffer size can be determined at this point. The best performance is reached at the buffer size of 221 MB for short query sequence size of 119. The buffer sizes of 260 MB and 273 MB are excellent for query sizes of 714 and 1420, respectively.

However, with the various data buffer sizes, the additional overhead in the pre-fetching implementation incurred by the buffer management may result in an idle FPGA coprocessor. Below a buffer size of 52 MB or above 286 MB, the performance is not improved, but rather the delay increases. In our experiments, the worst performances were observed at the boundaries, i.e., buffer sizes of 13 MB and 325 MB, due to the overhead such as frequent switching between small buffers or filling out a very large buffer.

As we have explained earlier in Section 5.1, the FPGA Smith–Waterman processing time is a stair function rather than a linear function. The database buffer load time on the host machine should be fairly close to the FPGA processing time. The prefetching buffer size, S_{buffer}, required to compare a query sequence to a database of sequences on a single FPGA board is given as

$$S_{\text{buffer}} = \lambda \times g\left[\text{ceil}\left(\frac{m}{N+1}\right)\right], \tag{6}$$

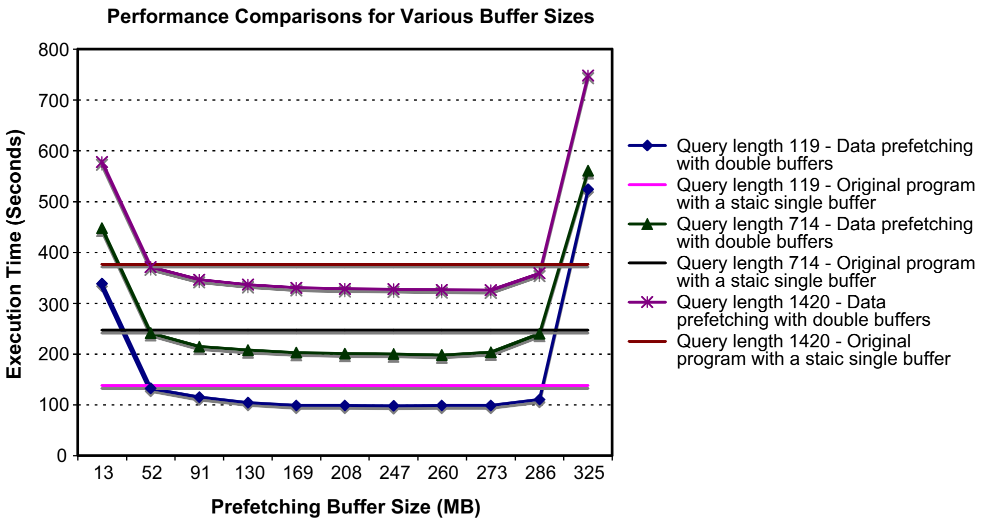


Fig. 7. Data prefetching performance comparisons for various queries against various double buffer sizes.

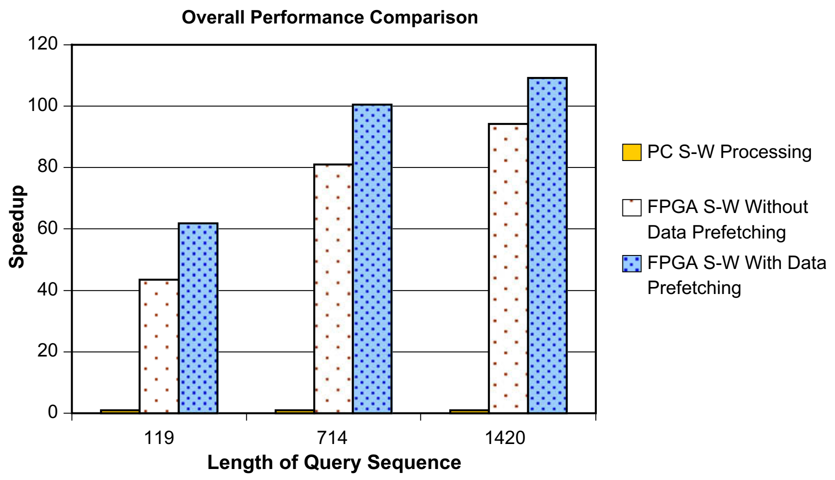


Fig. 8. Performance of adaptive data prefetching.

where S_{buffer} is the prefetching buffer size in megabytes, λ is the coefficient of host machine speed, m is the size of query sequence, N is the number of PEs, and $g[x]$ is a stair function for buffer size based on our experiments, and is summarized as

$$g(x) = \begin{cases} 221, & x = 1, \\ 234, & x = 2, \\ 247, & x = 3, \\ 260, & 4 \leqslant x \leqslant 6, \\ 273, & 7 \leqslant x \leqslant 12. \end{cases} \tag{7}$$

Fig. 8 shows the performance gains with various query sequences searched against the nr database. While using the adaptive data prefetching on FPGA, we achieved a 42% overall performance increase on a short query sequence of length 119 over the version without data prefetching. And there are 21% and 16% improvements for query sizes of 714 and 1420, respectively. For performance comparison, the adaptive data prefetching scheme achieves a speedup of 110 over a desktop computer with a 1.9 GHz Pentium IV running the serial program.

5.3. Prefetching performance analysis

In a data prefetching implementation, the data transfer is double buffered. Overlapping communication and execution prevents the host processor from stalling while it is waiting for the computation to finish, and hides the communication time

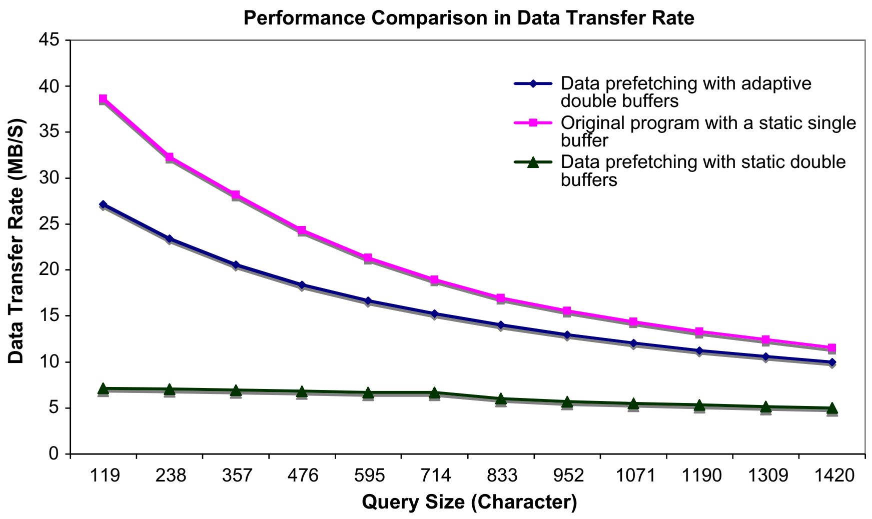


Fig. 9. Comparisons of database sequences transfer rate for various query sizes.

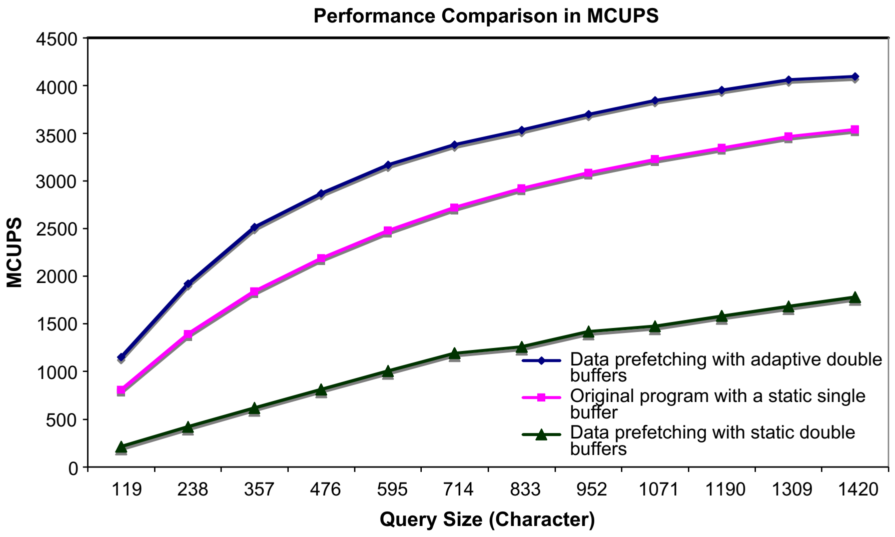


Fig. 10. Comparisons of the overall performance in MCUPS for various query sizes.

from the program execution. The host database load and the FPGA processing are two independent stages so that the total execution time drops to the maximum of the time to transfer data and the time to perform the computation. From Fig. 9, we know that $T_{overhead}$ is associated with the sequence buffer size S_{buffer}.

$$T_{total\_static\_prefetch} = MAX(T_{fpga}, T_{dbloading} + T_{overhead}[S_{buffer}]). \tag{8}$$

Ideally, with the appropriate selection of S_{buffer}, $T_{overhead}$ can be completely eliminated by applying the adaptive scheme. The optimal performance can be obtained by adaptive data prefetching as shown in Eq. (9).

$$T_{total\_adaptive\_prefetch} = MAX(T_{fpga}, T_{dbloading}). \tag{9}$$

Fig. 9 shows the impact of data prefetching operations and the achievable data transfer rate over the non-prefetching operations and the double buffered prefetching with a static buffer size of 325 MB. Here, we picked the worst case of the static double buffer size in order to highlight the importance of the adaptive strategy. Especially from a query size of 714 to 119, the bandwidth is increased from 21% to 42% with the adaptive data prefetching over the non-prefetching. Due to the complexity of dynamic programming, database sequence transfer rate decreases with the longer query. That is, the long query sequence requires more time to perform computation than the short sequence on the same data set (Fig. 3). The data prefetching scheme enables the pipelining of the communications such that it not only absorbs the database sequences transfer time on the host machine, but also absorbs the latency of the result transfers between the FPGA and the host. This approach greatly improves the bandwidth without upgrading the widely used legacy PCI communication interface, which usually limits the effectiveness of the reconfigurable computing.

Fig. 10 shows the overall performance comparison in MCUPS (mega cell updates per second). MCUPS is a commonly used performance comparison measure in computational biology and represents the number of times in a second that an FPGA coprocessor can update 1,000,000 Smith–Waterman cells (Eqs. (1) and (2)). MCUPS provides a good usability metric for sequence database search system implementations.

The performance plot demonstrates a linear growth in processing speed as the input query size increases. The communication pipeline significantly improves the overall performance and can achieve nearly 4000 MCUPS real performance depending on the query size. With data prefetching, the query sequence size of 833 could obtain almost the same MCUPS speed as the query size of 1420 without the prefetching.

6. Summary and future work

As the computing demands of the bioinformatics applications has continued to increase, and various applications from drug design to forensic DNA analysis come to rely on this technology. Many solutions, especially FPGA-based reconfigurable computing, to the problem of biological sequence database searches have been studied, but almost all implementations focus on the processing engine design: FPGA reprogrammable logics. Unfortunately the impact of data transfer between the host computer and the FPGA coprocessor is sometimes ignored by researchers. Recent technological advances have accelerated the imbalances between data transfer speed and processor speed. Techniques to reduce communication latencies become essential for achieving high FPGA utilization. We investigated the possibility of improving the communication efficiency between the host computer and the FPGA coprocessor for further accelerating the existing database search systems.

In this research, we implemented a software data prefetching scheme by exploiting the overlap between computation and communication. Performance is improved when the communication is overlapped with computations performed by the FPGA coprocessor, because loading the database sequences requires a significant amount of time to accomplish. However, the communication overhead of the inappropriate size of data buffer may decrease the prefetching performance gains. An approach that dynamically determines the size of prefetched data has been shown to be very effective for reducing the communication latency with the minimal overhead. The data prefetching design can yield up to 42% improvement in performance compared to the non-prefetching implementation.

Our pre-fetching scheme can also be applied to any other PCI-based FPGA co-processing system. This technique can effectively achieve significantly higher levels of FPGA efficiency by reducing the existent processor-to-coprocessor communication latency and making it useful for enhancing the performance of even those data intensive applications in which the required bandwidth is high. The performance results indicate that a smaller and cheaper FPGA with the data prefetching could deliver all of the performance that a bigger and more expensive one will meet. The high bandwidth and low latency I/O, and high FPGA density would be critical to the performance, but the appropriate communication strategy can be even more critical.

As a next step, we will apply the data prefetching scheme for a network-based heterogeneous parallel and distributed biological database sequence search platform. We expect our strategy to be beneficial in reducing the network communication latency too. The limitation of the current Smith–Waterman implementation on FPGA, such as the length limitations of query and database sequences, will also be solved via heterogeneous parallel computing.

Acknowledgements

The authors would like to thank Progeniq [21] for providing the FPGA software and sequence databases for the research and the anonymous reviewers for their detailed comments.

References

[1] Alpha-Data. <http://www.alpha-data.com>.
[2] J. Baer, T. Chen, An evaluation of hardware and software data prefetching, Supercomputing (1991) 176–186.
[3] Di Bias et al, The kestrel parallel processor, IEEE Transactions on Parallel and Distributed Systems 16 (1) (2005) 80–92.
[4] Tien-Fu Chen, J.L. Baer, A performance study of software and hardware data prefetching schemes, IEEE Transactions on Computers 44 (5) (1995) 609–623.
[5] Cray Inc. <http://www.cray.com>.
[6] S. Dydel, P. Bala, Large scale protein sequence alignment using FPGA reprogrammable logic devices, LNCS 3203 (2004) 23–32.
[7] D.T. Hoang, Searching genetic databases on Splash 2, in: IEEE Workshop on FPGAs for Custom Computing Machines, IEEE Computer Society Press, 1993, pp. 185–191.
[8] E.H. Gornish, A. Veidenbaum, An integrated hardware/software data prefetching scheme for shared-memory multiprocessors, International Journal of Parallel Programming 27 (1) (1999) 35–70.
[9] O. Gotoh, An improved algorithm for matching biological sequences, Journal of Molecular Biology 162 (1982) 705–708.
[10] S.A. Guccione, E. Keller, Gene matching using JBits, field-programmable logic and applications, LNCS 2438 (2002) 1168–1171.
[11] P. Guerdoux-Jamet, D. Lavenier, SAMBA: Hardware Accelerator for Biological Sequence Comparison CABIOS, vol. 13, Oxford University Press, Oxford, 1997. pp. 609–615.
[12] R. Hughey, Parallel hardware for sequence comparison and alignment, Computer Applications in the Biosciences 12 (1996) 473–479.
[13] S. Kleiman, D. Shah, B. Smaalders, Programming with Threads, first ed., Prentice Hall, 1996.
[14] X. Meng, V. Chaudhary, Bio-Sequence Analysis with Cradle's 3SoC Software Scalable System on Chip, in: Proceedings of the ACM Symposium on Applied Computing, 2004, pp. 202–206.
[15] X. Meng, V. Chaudhary, Optimized fine and coarse parallelism for sequence homology search, International Journal of Bioinformatics Research and Applications 2 (4) (2006) 44–430.
[16] X. Meng, V. Chaudhary, An adaptive data prefetching scheme for biosequence database search on reconfigurable platforms, Proceedings of the ACM Symposium on Applied Computing (2007) 140–141.
[17] T. Oliver, B. Schmidt, D. Maskell, Hyper customized processors for bio-sequence database scanning on FPGAs, IEEE Transactions on Circuits and Systems II 52 (12) (2005) 851–855.
[18] T.F. Oliver, B. Schmidt, Y. Jakop, D.L. Masskell, Accelerating the Viterbi algorithm for profile hidden markov models using reconfigurable hardware, LNCS 3991 (2006) 522–529.
[19] T. Rognes, E. Seeberg, Six-fold speedup of Smith–Waterman sequence database searches using parallel processing on common microprocessors, Bioinformatics 16 (8) (2000) 699–706.
[20] Paracel. <http://www.paracel.com>.
[21] Progeniq Pte. Ltd. <http://www.progeniq.com/>.
[22] B. Schmidt, H. Schroder, M. Schimmler, Massively parallel solutions for molecular sequence analysis, in: International Parallel and Distributed Processing Symposium: IPDPS'02 (HiComB'02), Ft Lauderdale, FL, IEEE, 2002, pp. 186–193.
[23] T.F. Smith, M.S. Waterman, Identification of common molecular subsequences, Journal of Molecular Biology 147 (1981) 195–197.
[24] Starbridge Systems Inc. <http://www.starbridgesystems.com>.
[25] Y. Yamagucchi, T. Maruyama, A. Konagaya, High speed homology search using run-time reconfiguration, LNCS 3438 (2002) 281–291.
[26] C.W. Yu, K.H. Kwong, K.H. Lee, P.H.W. Leong, A Smith–Waterman systolic cell, LNCS 2778 (2003) 375–384.