Network-on-Chip Hardware Accelerators for Biological Sequence Alignment

Souradip Sarkar, *Student Member*, *IEEE*, Gaurav Ramesh Kulkarni, *Student Member*, *IEEE*, Partha Pratim Pande, *Member*, *IEEE*, and Ananth Kalyanaraman, *Member*, *IEEE*

Abstract—The most pervasive compute operation carried out in almost all bioinformatics applications is pairwise sequence homology detection (or sequence alignment). Due to exponentially growing sequence databases, computing this operation at a large-scale is becoming expensive. An effective approach to speed up this operation is to integrate a very high number of processing elements in a single chip so that the massive scales of fine-grain parallelism inherent in several bioinformatics applications can be exploited efficiently. Network-on-Chip (NoC) is a very efficient method to achieve such large-scale integration. In this work, we propose to bridge the gap between data generation and processing in bioinformatics applications by designing NoC architectures for the sequence alignment operation. Specifically, we 1) propose optimized NoC architectures for different sequence alignment algorithms that were originally designed for distributed memory parallel computers and 2) provide a thorough comparative evaluation of their respective performance and energy dissipation. While accelerators using other hardware architectures such as FPGA, General Purpose Graphics Processing Unit (GPU), and the Cell Broadband Engine (CBE) have been previously designed for sequence alignment, the NoC paradigm enables integration of a much larger number of processing elements on a single chip and also offers a higher degree of flexibility in placing them along the die to suit the underlying algorithm. The results show that our NoC-based implementations can provide above 10^2-10^3-fold speedup over other hardware accelerators and above 10^4-fold speedup over traditional CPU architectures. This is significant because it will drastically reduce the time required to perform the millions of alignment operations that are typical in large-scale bioinformatics projects. To the best of our knowledge, this work embodies the first attempt to accelerate a bioinformatics application using NoC.

Index Terms—Network-on-chip, bioinformatics, DNA/protein sequence alignment, on-chip parallelism, hardware acceleration.

1 Introduction

The bioinformatics community faces a daunting challenge today because the rate of data generation is rapidly outpacing the rate at which it can be computationally processed. Propelled by recent technological breakthroughs in high-throughput DNA and protein sequencing, experimentalists are generating data at unprecedented rates. For example, the GenBank database [1], which is the largest public repository for molecular sequence data, is continuing to double in size every 18 months since its inception in the early 1990s. Sequence data are subsequently used in further computational analysis that can ultimately lead to the discovery and fundamental understanding of the genetic composition in organisms.

The most predominant compute operation that is carried out in nearly all sequence analysis projects is pairwise sequence alignment, which aims at measuring the similarity between two DNA or protein "sequences" (represented as strings over a fixed alphabet). The operation is performed using a dynamic programming (DP) algorithm [2], [3] that computes a two-dimensional table, with rows and columns representing the character sequence of the two strings being compared. The operation is used almost on a daily basis by molecular biologists, and also in all genome sequencing projects of any scale. While the task of carrying out a single pairwise sequence comparison is in itself computationally lightweight (in milliseconds) on traditional machines, performing millions to billions of such comparisons could become easily prohibitive. For example, a recent analysis that computed pairwise alignments for over 28 million metagenomic sequences [4] took an aggregate of 10^6 CPU hours—a task that took months to complete after parallelization at the coarse level using a combination of 2,300 processors and high-end memory systems. Our experiments show that running the multiple sequence alignment tool ClustalW [5] even on hundreds of sequences requires several hours on state-of-the-art workstations.

To speed up the data processing, several hardware accelerators have been proposed recently including, but not limited to, [6], [7], [8], [9], [10]. Among these, the use of FPGA-based reconfigurable hardware platforms, Graphics Processing Unit (GPU), and Cell Broadband Engine (CBE) is notable. The principal advantages of using FPGA-, GPU-, or CBE-based systems are fast prototyping and ease of implementation. These systems primarily rely on software and use an existing hardware platform to map algorithms. In the aforementioned schemes, as the basic hardware is not tailor-made for the applications under consideration, the achievable speedup is typically limited—the accelerators built for sequence alignment based on FPGAs, GPUs, and

• *The authors are with the School of Electrical Engineering and Computer Science, Washington State University, PO Box 642752, Pullman, WA 99164-2752.*
E-mail: (ssarkar, gkulkarn, pande, ananth)@eecs.wsu.edu.

Manuscript received 10 Nov. 2008; revised 11 Mar. 2009; accepted 16 Apr. 2009; published online 10 Sept. 2009.
Recommended for acceptance by R. Marculescu.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2008-11-0565.
Digital Object Identifier no. 10.1109/TC.2009.133.

CBE provide a speedup of around 100[1] [6], [7], [8], [9] over standard serial implementation. But such generic hardware-based systems can be used for multiple applications with only software modifications.

For large-scale deployment of a data-intensive application, performance and scalability are of major concerns, and therefore, it is desirable that the hardware implementation is optimized to suit the exact computation and on-chip communication patterns that the application code generates. With biological databases already having reached hundreds of billions of bytes and continuing to swell at exponential rates, a simple search operation against a database of known sequences involves an unprecedented number of sequence alignment operations—at a magnitude proportional to the size of the underlying database in the worst case. To tackle this challenge and thereby advance the state of biocomputing research to the next level, larger magnitudes of speedups are necessary, which can only be achieved by assimilating the latest breakthroughs in the Integrated Circuit (IC) design. An effective way to address this would be to integrate huge number of processing elements on a single chip for exploiting the massive scales of fine-grain parallelism inherent in bioinformatics applications. Computational systems based on multiple cores on a single chip are emerging as a viable method to continue the quest for higher performance in various application domains [11], [12]. Network-on-Chip (NoC) is viewed as an enabling methodology to obtain such high degree of integration in a single chip [13], [14]. It is the digital communication backbone, which interconnects the components on a multicore System-on-Chip (SoC). Power and wire design constraints are forcing the adoption of this new paradigm for designing large multicore chips, which incorporates modularity and explicit parallelism.

In this paper, we propose the design of NoC accelerators for performing the sequence alignment operation. As part of this effort, we designed and characterized the performance of NoC architectures that implement two of the most effective techniques for computing sequence alignment in parallel—1) using the parallel prefix operation to compute the DP table one row at a time and 2) using a systolic array-based technique that exploits the task parallelism that can be generated by processing one antidiagonal (same as "minor diagonal") of the DP table at a time. These techniques have been implemented in algorithms primarily designed for distributed memory parallel computers [15], [16], [17], [18]. The parallel prefix-based approach guarantees optimal parallel time complexity [15] and the antidiagonal guarantees optimal space complexity [17]. Here, we map these two techniques onto the chip using NoC such that the resulting architectures are optimized to suit the underlying computational and communication patterns. A subsequent comparative analysis of these two optimized NoC implementations reveals the performance and energy dissipation trade-offs. In addition, the results show that the NoC-based accelerators can outperform other hardware accelerators (based on FPGA, GPU, and CBE) by factors ranging from 60 to 4,000, and our serial implementation by factors ranging from 10^4 to 2.5×10^4. The several orders of magnitude in speedup observed are consistent with our design-level expectations, and demonstrates the high potential of NoC-based architectural design for biological sequence analysis.

It is to be noted that the scope of the designs described in this paper is restricted to the comparison of DNA or protein sequences whose lengths can be fit into the on-chip memory. Assuming that each character in a DNA or protein sequence occupies a byte, this length can range from a few hundreds to a few tens of thousands of characters. Biologically, this range is sufficient to include most realistic inputs—for example, DNA fragments generated in genome sequencing projects contain anywhere between 30 and 1,000 nucleotides (or character positions); most genes contain a few thousand nucleotides; and protein sequences contain typically 200-500 amino acid residues. For comparing full-length genomes that could span millions to even billions of characters in length, coarse-grain parallelization is more suited. In other words, the NoC architectures proposed in this paper provide the fine-grain parallelism that is required to ideally suit analysis wherein an abundant number of relatively small alignment tasks are computed.

The rest of the paper is organized as follows: Section 2 reviews the literature on the set of hardware accelerators that has been previously designed for sequence alignment. Section 3 presents the main ideas behind the serial and parallel algorithms for sequence alignment. In Section 4, two different NoC architectures are proposed. Section 5 presents the experimental results of testing and comparing our implementations. Finally, Section 6 concludes the paper. Throughout the paper, we will refer to pairwise sequence alignment as "PSA."

2 Related Work Involving Other Hardware Accelerators

Several hardware accelerators have been previously developed for PSA. These accelerators are based on FPGAs [6], [7], GPUs [8], or Cell Broadband Engine [9], [10]. Interestingly, all the above accelerators except the CBE implementation in [10] use the antidiagonal-based technique for parallelizing the computation of the DP table, as it can be implemented using a simpler layout of processing elements. However, unless the lengths of the two input sequences are approximately equal, the time complexity of the underlying parallel algorithm is suboptimal. From this perspective, it is imperative to implement and study the effectiveness of the parallel prefix-based technique, which guarantees an optimal runtime as well. Such an implementation would require a more complex layout of processing elements and support for arbitrary network topologies. Consequently, the purpose of this paper is to explore performance of NoC as a more powerful and flexible methodology for sequence alignment.

As for performance results, Weiguo et al. report that the GPU hardware GeForce 7800 GTX can perform up to 700 million DP cells per second, implying an overall time of 1.428 milliseconds for aligning two sequences of length ~1K each. The GPU cores are deployed directly without being optimized to implement the sequence alignment algorithm.

1. In direct comparison, the speedup based on the NoC designs presented in this paper exceeds 10^4 over serial implementations.

The FPGA implementation by Oliver et al. reduces the time to 1 millisecond. The CBE implementation by Sachdeva et al. using 16 Synergistic Processing Units (SPUs) runs in 0.65 millisecond. The other CBE implementation [10] achieves a runtime of $\sim$17 ms using eight SPUs. Once again these SPUs are not optimized for bioinformatics application suites. As a reference, our own serial implementation of the Smith-Waterman algorithm [2] took 100 milliseconds for aligning two 1K sequences on a 2.3 GHz Xeon CPU. This observation is consistent with the near 100-fold speedups reported by the authors of the aforementioned accelerators. The goal of *this* paper is to drastically improve on the speedup by orders of magnitude using custom-built optimized NoC architectures.

3 Algorithms for Sequence Alignment and Parallelization

Sequence alignment is a way of measuring the similarity between two sequences. Algorithmically, comparing two sequences (or strings) is modeled as a combinatorial optimization problem. Characters of one sequence are "aligned" against characters of the other in an order-preserving manner, and only a selected set of operations is permitted at each aligning position: 1) "match" one character with another, 2) "mismatch" (or substitute) one character with another, and 3) align one character with an empty (called "gap" and denoted by "-") symbol on the other string. Through a scoring scheme, a positive score is rewarded for similarities (match) and negative scores are assigned for differences (mismatches and gaps). The task of computing an optimal alignment is therefore a task of identifying an alignment with the maximum possible overall score.

Computing an optimal PSA is a well-studied problem [2], [3], [19]. While there are several variants of the problem, the complexities of the underlying DP algorithms are identical. Given two sequences s_1 and s_2 of lengths m and n, respectively, an optimal PSA can be computed sequentially using dynamic programming in $O(mn)$ time and $O(m+n)$ space. This is achieved by computing an $(m+1) \times (n+1)$-sized table T such that T[i,j] contains the optimal score for aligning the prefixes $S_1[1..i]$ against $S_2[1..j]$. For example, the global alignment[2] score of aligning prefixes $S_1[1..i]$ and $S_2[1..j]$ is given by

$$T[i,j] = \max \left\{ \begin{array}{l} T[i-1,j-1] + \sigma(s_1[i], s_2[j]) \\ T[i-1,j] - g \\ T[i,j-1] - g \end{array} \right\}, \qquad (1)$$

where $g > 0$ corresponds to the gap penalty and $\sigma()$ to the score for substituting $s_1[i]$ with $s_2[j]$ or vice versa. As can be noted, the value at T[i, j] depends on the cells T[i-1, j-1], T[i-1, j], and T[i, j-1]. Sequentially, this dependency constraint can be met during computation through a "forward pass" of the table in which the table is computed one row at a time starting from the first row, and within each row computing column by column starting from the first column. At the end of the forward pass, the optimal score is available at T[m, n]. The next step is a "backward pass" in which an optimal path (or equivalently, an optimal alignment) that led to the optimal score is retraced from T[m,n] to T[0,0]. Fig. 1 illustrates an example DP table along with its optimal path.

2. There are other commonly used formulations such as the local and semiglobal alignments Algorithmically, the recurrences are slightly modified variant of (1), with no impact on the overall complexities.

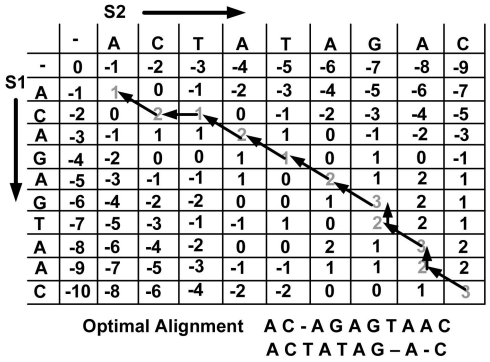


Fig. 1. Example of computing the global alignment between two sequences using Needleman and Wunsch algorithm [3]. The arrows show the optimal path. The following scoring scheme was used: match score $= 1$, mismatch penalty $= 1$, and gap penalty $= 1$.

Parallelization. There are two main challenges in parallelizing DP algorithms for PSA: 1) meeting the dependency constraint without affecting the parallel speed-up during forward pass and 2) computing the optimal retrace without storing the entire DP table during forward pass. To meet these challenges, several coarse-grain parallel algorithms have been previously developed [15], [16], [17], [18], [20]. These algorithms offer varying degrees of computational complexities and ease of implementation. The algorithm by Huang [17] develops on ideas proposed by Edmiston and Wagner [16] by using the antidiagonal approach during forward and backward passes. The guiding observation is that the cells along the same antidiagonal of the DP table are not interdependent, and therefore, can be computed in parallel. If p denotes the number of processors, then this algorithm requires $O(\frac{(m+n)^2}{p})$ time and $O(\frac{m+n}{p})$ space. Aluru et al. [15] devised an alternative strategy that overcomes the dependency constraint by reformulating the problem of computing the scores within a row in parallel using the parallel prefix operation. This algorithm requires $O(\frac{m \times n}{p})$ time and $O(m + \frac{n}{p})$ space, assuming that $m = O(n)$. The algorithm by Rajko and Aluru [17] uses a combination of these ideas to arrive at a more complex albeit time- and space-optimal solution—i.e., $O(\frac{m \times n}{p})$ time and $O(\frac{m+n}{p})$ space.

For mapping onto the NoC architecture, we based our choice on the following factors: 1) parallel runtime and space complexities, 2) relative ease of adoption to the on-chip framework, and 3) the potential to fully benefit from the on-chip communication network. Based on these factors, we selected the algorithms by Aluru et al. [15] and Huang [17] for implementations in this work. While it will be ideal to also evaluate the optimal algorithm by Rajko and Aluru [18], it is highly complex for implementation. It is to be noted that none of the previously proposed hardware accelerators implemented the parallel prefix approach.

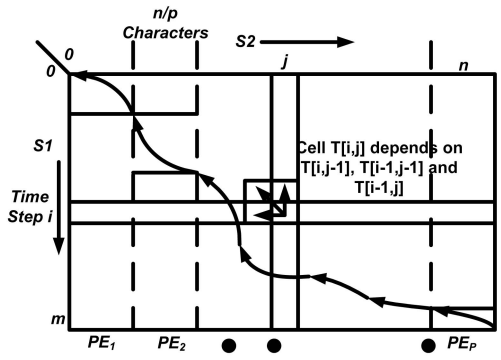


Fig. 2. Computation of the DP table in parallel using p processors in the parallel prefix approach.

In what follows, we briefly outline the main ideas behind these two algorithms. For convenience, we will refer to the algorithm by Aluru et al. as the "PP algorithm" (for parallel prefix), and the algorithm by Huang as the "AD algorithm" (for antidiagonal).

We implemented three of the most popular variants of the alignment problem—global [3], local [2], and semiglobal [21]. To best reflect practical application, we implemented the affine gap penalty function model [19] in which the gap penalty function grows linearly with the length of the gap in an alignment. Algorithmically, this is achieved by computing three DP tables (T_1, T_2, and T_3) instead of one DP table. However, the underlying runtime and memory complexities for computing alignments based on the affine gap model are exactly the same as that of the single-table constant gap model. The actual time and memory costs in practice are expected to only increase by a factor of 3. Because of this algorithmic equivalence and for ease of exposition, we will describe the parallel algorithms below for the single-table constant gap model, even though we implemented the more generic affine gap model.

The PP approach. The PP algorithm partitions the input sequence s_2 into p pieces such that each PE receives roughly n/p characters (as shown in Fig. 2). The other input sequence s_1 is made available to all the PEs one character at a time. Throughout, we will assume p to be a power of 2 although the algorithm can be easily extended to arbitrary processor sizes through a virtual padding scheme. The $(m+1) \times (n+1)$-sized table T computation is divided into p roughly equal parts such that PE $\mathrm{p_i}$ is assigned the responsibility of computing the ith block of $O(n/p)$ columns. The forward pass in table computation proceeds iteratively, row by row, such that at any given iteration i, all the PEs participate in computing row i in parallel. We identify each iterative step as one "time step." Within each row, the algorithm reduces the problem of computing the recurrence in (1) to the problem of computing the following recurrence:

$$X[j] = \left\{ \begin{array}{c} w[j] + jg \\ X[j-1] \end{array} \right\}, \tag{2}$$

where $0 \leq j \leq n$ and w[j] is obtained by local computation (without any need for communication). Computing this recurrence is equivalent to the problem of finding $\mathrm{n}+1$ prefix maximums, which can be easily accomplished using the PP operation as follows: Since each row is

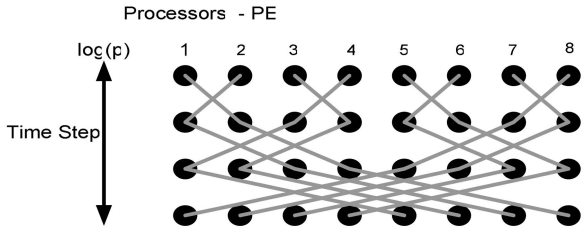


Fig. 3. Communication pattern generated by the PP algorithm.

partitioned into roughly $O(n/p)$ blocks and assigned to p PEs, the prefix maximums can be computed by first computing p local maximums in an $O(n/p)$ computation step, and then using $O(\log p)$ communication steps to update their local prefix maximums into global prefix maximums. More specifically, at communication step k (for $0 \leq \mathrm{k} < (\mathrm{logp}))$, PE_i exchanges its most recent global maximum with PE_j such that $\mathrm{j} = \mathrm{i} + 2^{\mathrm{k}}$ and the kth least significant bits in the binary representations of i and j are 0 and 1, respectively. For example, in a 4 processor system and at $\mathrm{k} = 0$, PE_0 exchanges with PE_1, and PE_2 exchanges with PE_3; at $\mathrm{k} = 1$, PE_0 exchanges with PE_2, and PE_1 exchanges with PE_3. The time steps of the inter-PE communication scenario for eight PEs are shown in Fig. 3. Consequently, each time step can be completed by performing: 1) an $O(n/p)$ local computation and 2) an $O(\log p)$ parallel prefix communication. After the last row is fully computed, the PEs reverse their computation by progressing from last row to top row and retrace an optimal alignment path that yielded the optimal score at cell T[m,n]. However, this would require that the entire table be stored, implying an $O(mn)$ aggregate space complexity. To allow retracing an optimal alignment in just $O(\frac{m+n}{p})$ space, each PE stores all the entries in the last column of its block of n/p columns and then uses this information to retrace. This step can be achieved in $O(n/p)$ computation time and $O(p)$ communication time. In the interest of space, we omit the details of analysis and proofs, and refer the reader to the original paper [15]. It has been proved that the algorithm has an overall computation complexity of $O(mn/p)$ and a communication complexity of $O(m \log p)$ on a distributed memory parallel computer [15].

The AD algorithm. This requires that both the input sequences s_1 and s_2 are made available to all the PEs. For the forward pass, the algorithm proceeds iteratively by computing one antidiagonal of DP table at each "time step", where an "antidiagonal" is defined as the subset of cells [i, j] that has the same $\mathrm{i} + \mathrm{j}$ value. For an $(m+1) \times (n+1)$ DP table, there are a total of $\mathrm{m} + \mathrm{n} + 1$ antidiagonals with values $0, 1, 2, \ldots, \mathrm{m} + \mathrm{n}$, and the algorithm computes the tth antidiagonal at time step t (as shown in Fig. 4). All the cells within an antidiagonal can be computed in parallel because the value at any T[i, j] depends on the values already computed and available from previous two time steps. The next question is to identify the PE that will work on each cell of an antidiagonal. It turns out that the assignment of PEs to cells does not matter for the overall complexity as long as the number of PEs working on an antidiagonal is maximized. Therefore, in this paper, we adopt a strategy that will assign PE_k to the cells in the antidiagonal that have the form [i, j] such that $(\mathrm{i} \bmod \mathrm{p}) = \mathrm{k}$. This is shown

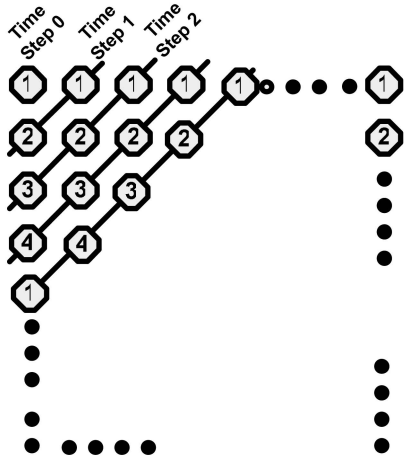


Fig. 4. Antidiagonal table computation. The numbers within the cells represent the PE responsible for computing it.

in Fig. 4. We selected this cyclic allocation scheme because the communication pattern that emerges from such a setting is a simple neighborhood communication—i.e., the data that $\mathrm{PE_k}$ needs to compute a cell (i, j) are present either within itself or in $\mathrm{PE_{k-1}}$ (as shown in Fig. 4).

For the backward pass, the main challenge is to reconstruct an optimal path in the absence of the entire DP table stored at the end of the forward pass, as otherwise the space complexity will be Θ(mn). The algorithm by Huang uses a variation of the Hirschberg technique for space reduction [22], by identifying the cell $(\mathrm{i}', \mathrm{j}')$ in the middle antidiagonal through which an optimal path must have passed. Once such a cell is found, the problem space can be recursively subdivided into reconstructing the paths on the left-top and right-bottom sides of $(\mathrm{i}', \mathrm{j}')$. In this paper, we developed another variant that directly uses the Hirschberg technique. In this scheme, the special cell $(\mathrm{i}', \mathrm{j}')$ is defined to be $(\lceil m/2 \rceil, j)$ through which an optimal path is guaranteed to pass. We achieve this by propagating all possible "candidate" cells during the forward pass such that the candidate that propagates to the last cell $(\mathrm{m}+1) \times (\mathrm{n}+1)$ is the winner. Once such an $(\mathrm{i}', \mathrm{j}')$ is found, the original problem of retracing the DP table from $(\mathrm{m}+1,\ \mathrm{n}+1)$ back to (0, 0) reduces to two disjoint subproblems: 1) retrace from $(\mathrm{m}+1,\ \mathrm{n}+1)$ back to $(\lceil m/2 \rceil, j)$ and 2) retrace from $(\lceil m/2 \rceil, j)$ back to (0, 0). These two subproblems can be solved using recursion. To maintain parallel efficiency during these recursive steps, we partition the processor space into two subsets such that the number of PEs in each subset is proportional to the number of cells for computation in the corresponding recursive step. The AD approach also requires $O(\frac{(m+n)^2}{p})$ time and $O(\frac{m+n}{p})$ space.

4 NoC Implementation

4.1 Mapping the Sequence Alignment Algorithms to NoC

Instead of depending on FPGAs or any other existing processing platforms, we designed tiny PEs operating on a particular segment of the table T, as explained in Section 3, and integrating them using an NoC.

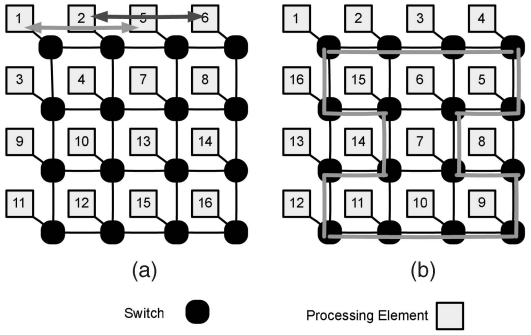


Fig. 5. (a) Mapping of the PP algorithm into a mesh. (b) Mapping of the AD algorithm into a mesh with an embedded ring.

4.1.1 The PP Implementation

The communication is always point-to-point and the PEs are required to exchange a single integer number among them. As an example, the interprocessor communication steps for a system with eight PEs are shown in Fig. 3. Consequently, instead of building a full-blown packet-switched network, a simpler circuit-switched NoC is designed. The total time required to complete a sequence alignment operation depends on the computation time taken by each PE and the communication time needed to exchange data among the PEs. The algorithm is structured such that at every time step, there is an $O(n/p)$ local computation phase followed by a communication phase. The communication carries out a parallel prefix operation in $O(\log p)$ stages, but the overall communication time will depend on the architecture of the NoC and placement of the PEs. Therefore, it is important to place the PEs in the NoC in such a way so as to reduce both the latency and energy dissipation in communication. While there are multiple NoC architectures [14], the hypercubic topology is best suited for the parallel prefix operation. But a physical realization of the NoC is limited by the layout dimension of the chip, which is predominantly 2D in practice. For a system size of p, if we construct a $\log_2$p-dimensional hypercube, then the number of hops between any two PEs is always 1. But as shown in Fig. 5a, if we embed a $\log_2$p-dimensional hypercube into a D-dimensional mesh, which is more realistic from an implementation perspective, then the number of hops in the ith communication step is given by (3):

$$L_i = 2^{\lfloor \frac{i-1}{D} \rfloor}, \tag{3}$$

where i can range from 1 to $\log_2$p. Hence, the maximum communication latency (in number of hops) between any two PEs is given by (4):

$$L_{\max} = 2^{\lfloor \frac{\log_2 p - 1}{D} \rfloor}. \tag{4}$$

Given that there are exactly $\lceil \log_2 p \rceil$ time steps within one parallel prefix operation, the total communication time T (in hops) of the parallel prefix operation is given by (5):

$$T = \sum_{i=1}^{\log_2 p} 2^{\lfloor \frac{i-1}{D} \rfloor}. \tag{5}$$

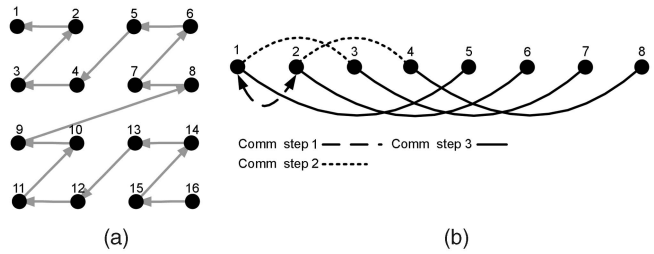


Fig. 6. (a) The communication pattern for backward pass in PP. (b) The communication pattern for backward pass in AD.

For 2D, the above series, and hence, T evaluate to $(2\sqrt{2} \times p - 2)$ if log p is even, and $((\frac{3}{\sqrt{2}})p - 2)$ otherwise. Given that there are $O(\log p)$ time steps, the above results yield an average of $O(p/\log p)$ number of hops per time step for both cases.

It is not practical to implement any arbitrarily higher dimensional hypercube. For example, a system with 64 PEs would necessitate construction of a 6D hypercube. Therefore, for investigation, we designed a 2D mesh-based NoC for carrying out sequence alignment. When the communication pattern shown in Fig. 3 is mapped to a 2D mesh-based NoC, even a data exchange between two hypercubic neighbors may cost several hops. But a well-defined property of the communication pattern in parallel prefix algorithm is that PEs do not communicate arbitrarily. As an example, in a system with 16 PEs, if the placement of Fig. 5a is followed, then the worst-case communication latency arises while communicating between PEs separated by two hops. With increasing system size, this worst-case communication latency will be more. For a system with 64 PEs, the worst-case latency in a communication step will be four hops.

As explained above, the forward pass is followed by a backward pass operation. This step is implemented using p-1 neighbor PE communication exchanges, as the PEs regenerate the path from cell [m, n] to [0, 0]. We modeled this communication pattern as a Z space filling curve as shown in Fig. 6a.

4.1.2 The AD Implementation

In the AD approach, the forward pass requires only a neighborhood PE communication pattern, as explained in Section 3. More specifically, the three values required to compute T[i, j] are available from the previous two antidiagonals, and because of the cyclic allocation strategy that we used to map PEs onto the antidiagonals (see Fig. 4), these cells are either present in the same PE or the previous PE. The exception is the first PE which will depend on the last PE due to the cyclic allocation. Therefore, it suffices to use a ring topology. In our implementation, we achieve this by embedding a ring into the mesh, and following the Moore space filing curve, which is similar to the Hilbert curve [23] for PE numbering. The placement of PEs is shown in Fig. 5b.

This interconnection enables single-cycle communication among the neighboring nodes. At every time step, each PE works on one antidiagonal of the DP table. If the length of an antidiagonal is greater than the number of PEs, then the cells are computed in multiple stages. The number of such stages is given by

$$\begin{aligned} &Stages\ per\ Anti\ Diagonals \\ &= \frac{Average\ Anti\text{-}Diagonal\ length}{\#PEs} \approx \frac{m}{2p}, \end{aligned} \tag{6}$$

where $m \leq n$ without loss of generality. The communication steps follow each of the computation steps at each cell. This implies that the total number of data exchanges is proportional to $(m+1) \times (n+1)$. Note that this result is unlike the PP approach; the communication volume is independent of the number of PEs during the forward pass.

In our backward pass implementation, we partition the processors into two subgroups, and the number of processors in each of the subgroups depends on the number of cells in the two partitions. The processor grouping requires a broadcast operation, as shown in Fig. 6b, to propagate the new partitioning cell coordinate to all the PEs, which takes $O(\log p)$ time (where p is the number of PEs).

4.2 NoC Switch Design

Due to the deterministic pattern of communication in case of both the PP and AD techniques, we designed simple pass transistor-based switch boxes [24] to forward the data from one PE to the other, instead of designing network routers for data communication.

In the PP approach, data exchanges between two nonadjacent hypercubic neighboring PEs give rise to higher communication delay. To reduce the delay, instead of building a multihop or pipelined communication link between two nonadjacent PEs, the switch boxes are designed to establish a direct communication path (unpipelined or single hop) between the PEs [25]. As an example, for the mesh shown in Fig. 5a, a particular communication step requires that PE_1 communicates with PE5, and simultaneously, PE_2 communicates with PE_6. In this situation, the switch box connected to 2 should be configured in such a way that a direct communication link is established between 1 and 5 as shown in Fig. 7. The architecture of a switch is shown in Fig. 8. Commensurate with all the time steps in the parallel prefix operation (shown in Fig. 3), the switch is

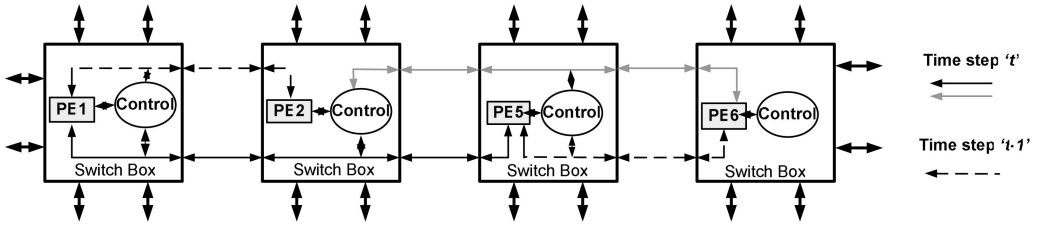


Fig. 7. Establishment of communication links facilitating bypass during parallel prefix.

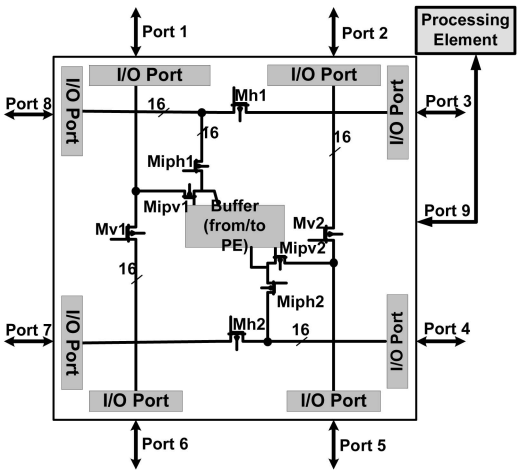


Fig. 8. Generic switch architecture for both PP and AD approaches.

designed to establish direct path between any two communicating neighbors in the vertical and horizontal directions. For a system with 16 PEs, the communication steps within the parallel prefix operation are shown in Fig. 9. In "Time Step 1" of parallel prefix, the neighboring processing elements communicate (like 1-2, 3-4, 5-6, etc.). For example, to exchange data between PE_1 and PE_2, the following pass transistors will be on: Miph1 and Mh1 of the switch connected to PE_1, and Miph1 of the switch connected to PE_2. The same switch setup facilitates data transfer among PE_3-PE_4, PE_5-PE_6...PE_{15}-PE_{16}. In the subsequent time steps, other switches involved in the communication are configured accordingly (by turning on suitable pass transistors in the switches). With increasing system size, the number of ports in the switches needs to increase to facilitate single-hop or unpipelined communication. On the contrary, the number of ports in the multihop scenario does not increase as the message ripples through the intermediate switches. During the backward phase, the data transfer is serial, and hence, no new modification is required in the existing communication infrastructure.

For the AD technique, during the forward phase, only neighboring PEs communicate simultaneously. The backward pass requires broadcasting of information, which is implemented as shown in Fig. 6b. To achieve the simultaneous communication among multiple nonadjacent PEs during this phase, the bypass strategy of the PP implementation is adopted here too.

5 Experimental Results

Input data. Given that the complexities of the PSA algorithms discussed above depend only on the lengths of the sequences being compared (and not on the sequences content), we used two arbitrary DNA sequences with lengths 1,024 characters each in all our experiments. In practice, the length 1,024 represents the length range for sequences that can be experimentally generated (or "sequenced") using a traditional Sanger sequencer [26]. These sequences constitute a typical input in genome sequencing projects, where a massive number of pairwise alignments are computed over millions of such sequences. However, we also note that there are DNA sequences of a vast length ranges in public databases—from tens to hundreds of characters (e.g., short reads from new generation sequencing), to thousands of characters (genes), to tens of thousands to millions and even billions of characters (fully assembled whole genomes). Even though we selected 1,024 for our input tests, our NoC implementations can be used to any of these length ranges as long as the sequences fit in an on-chip memory. Fixing the input size in all the experiments allowed us to conduct a fair comparative evaluation of the different NoC architectural topologies in our implementations.

Experimental setup. We present here experimental results of our NoC implementations for the two algorithms described in Section 3: PP and AD. We studied the timing requirements and the energy dissipation for both cases. Each character of a string was represented using 3 bits. This is because the alphabet size of DNA sequences typically used in practice is 5 ({a, c, g, t, n}). In all our implementations, the PEs need to exchange only integer data among them. Each integer number used during the communication was represented by 16 bits. Each PE was designed to perform the required character comparison, which is the primary unit operation in string alignment. The PEs were implemented in RTL and then synthesized using the 90 nm standard cell library from CMP (http://cmp.imag.fr/). The PEs communicate with each other with the help of switches. The switches mainly consist of pass transistor logic and were designed using Cadence Spectra tools.

We considered two types of NoC implementations: 1) a "Pipelined" communication scheme, where all the nonadjacent PEs communicate in multiple hops step by step and 2) an "Unpipelined" communication scheme, where the nonadjacent hypercubic neighboring PEs communicate in a single hop with the help of bypass. Performances of both

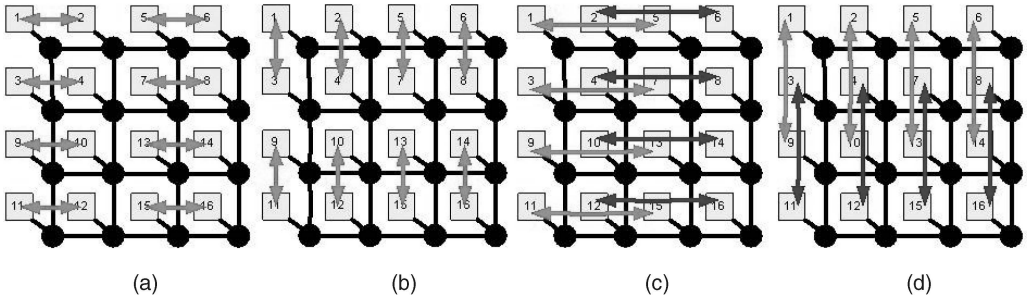
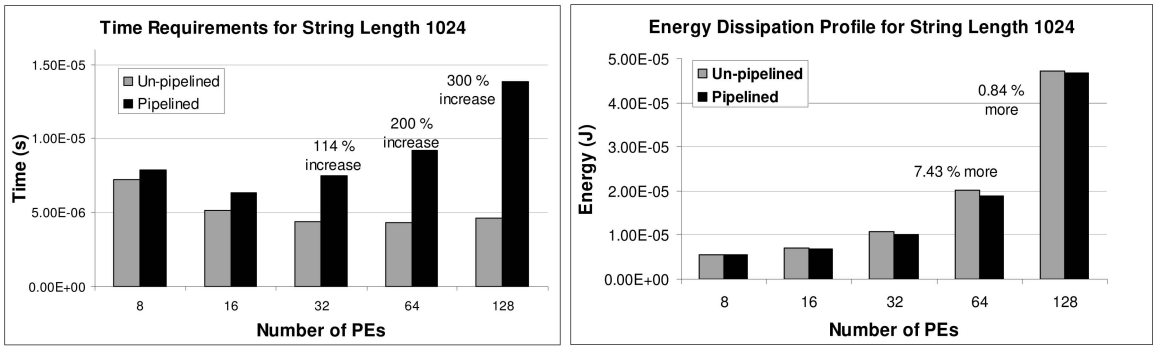


Fig. 9. Different time steps in communication during parallel prefix (PP). The shaded arrows represent the different simultaneous communications taking place for each parallel prefix step. (a) Time step 1. (b) Time step 2. (c) Time step 3. (d) Time step 4.



(a) (b)

Fig. 10. (a) Time requirements comparison for PP. (b) Energy dissipation profiles for PP.

schemes were compared in terms of energy and timing. The energy dissipation and the total time required in the sequence alignment operation depend on the PEs and the communication infrastructure. The energy dissipation and delay of the communication infrastructure, in turn, depend on two components, the switch blocks and the interswitch wires. The energy dissipation and delay of the switch blocks were determined using the CADENCE Spectra tool. The clock frequency of operation was 1.667 GHz. The delay and energy dissipation of the interswitch wires depend on their capacitance, which was calculated by taking into account each interswitch wire's specific layout by the following expression:

$$C_{\text{interswitch}} = C_{wire} \cdot w_{i+1,i} + n \cdot m \cdot (C_G + C_J), \qquad (7)$$

where C_{wire} represents the capacitance per unit length of the wire, $w_{i+1,i}$ is the wire length between two communicating switches, n is the number of repeaters, m represents the size of those repeaters with respect to minimum size devices, and finally, C_G and C_J represent the gate and junction capacitance, respectively, of a minimum size inverter. The energy dissipation and delay incurred by each PE are obtained using Synopsys Design Vision.

5.1 Pipelined versus Unpipelined Implementation

As a first step, we compare and contrast the performance of the NoC for the pipelined and the unpipelined implementations. We conduct this analysis only for the PP implementation. For the AD implementation, only neighborhood PEs communicate (in a ring topology), and therefore, all data transfers will be inherently single hop, no bypass strategy is needed.

Figs. 10a and 10b show the timing and energy dissipation profile of the NoC with varying system size. As can be expected from Fig. 10a, the pipelined implementation takes much longer time to complete than the unpipelined implementation. The total time has two parts: communication time and computation time. With increase in system size, the communication time increases because there will be more number of time steps within each parallel prefix operation in both the multihop and single-hop scenarios. However, the computation time of each PE decreases due to reduction in the substring length. But in the pipeline scenario, the communication time dominates over the computation time significantly (as an example, 14:1 ratio for a system size of 64). This is true for the unpipelined case as well, but the ratio is smaller (6:1 for the same system). As a result, in the pipelined case, the rate of increase of overall time with increasing system size is much higher than the corresponding unpipelined implementation. Contrary to the timing characteristics, the unpipelined case dissipates more energy (as shown in Fig. 10b). This can be attributed to various factors: 1) the increase in the interconnect length traversed in one communication cycle, 2) buffers along the path, and 3) increase in switch complexity.

It can be observed in Fig. 10 that the unpipelined implementation provides significantly more savings in time compared to the pipelined one, while resulting in very little penalty in energy consumption for all system size. As an example, for a system size of 128 PEs, the unpipelined implementation achieves more than 300 percent improvement in time while consuming only 0.84 percent more energy. This result indicates the value added by the bypass strategy. As a result of this analysis, we adopted the unpipelined strategy in the final implementation of PP, and all the corresponding results presented henceforth are for the unpipelined implementation.

5.2 Energy and Timing Characteristics of the PP Approach

Fig. 11 shows the energy and timing characteristics of our NoC implementation of the PP approach. Fig. 11a shows the energy dissipation profile with varying the number of PEs for the PP operation. The two contributing factors are the communication and computation energy. The increase in the communication energy with system size is attributed to the increase in total number of communication steps. On the contrary, the computation energy reduces very slowly. With doubling the system size, the work performed by each PE reduces by half. Hence, the energy per PE also reduces. It is observed that with doubling system size (halving the string length handled by each PE), the factor of energy reduction per PE is slightly more than 2. Consequently, the total computation energy has a slow decreasing trend with doubling system size. Overall, the net energy trend is dominated by the communication energy.

The timing characteristics of our PP implementation are shown in Fig. 11b. For a system with p PEs, each parallel prefix operation involves $O(\log p)$ communication steps. As

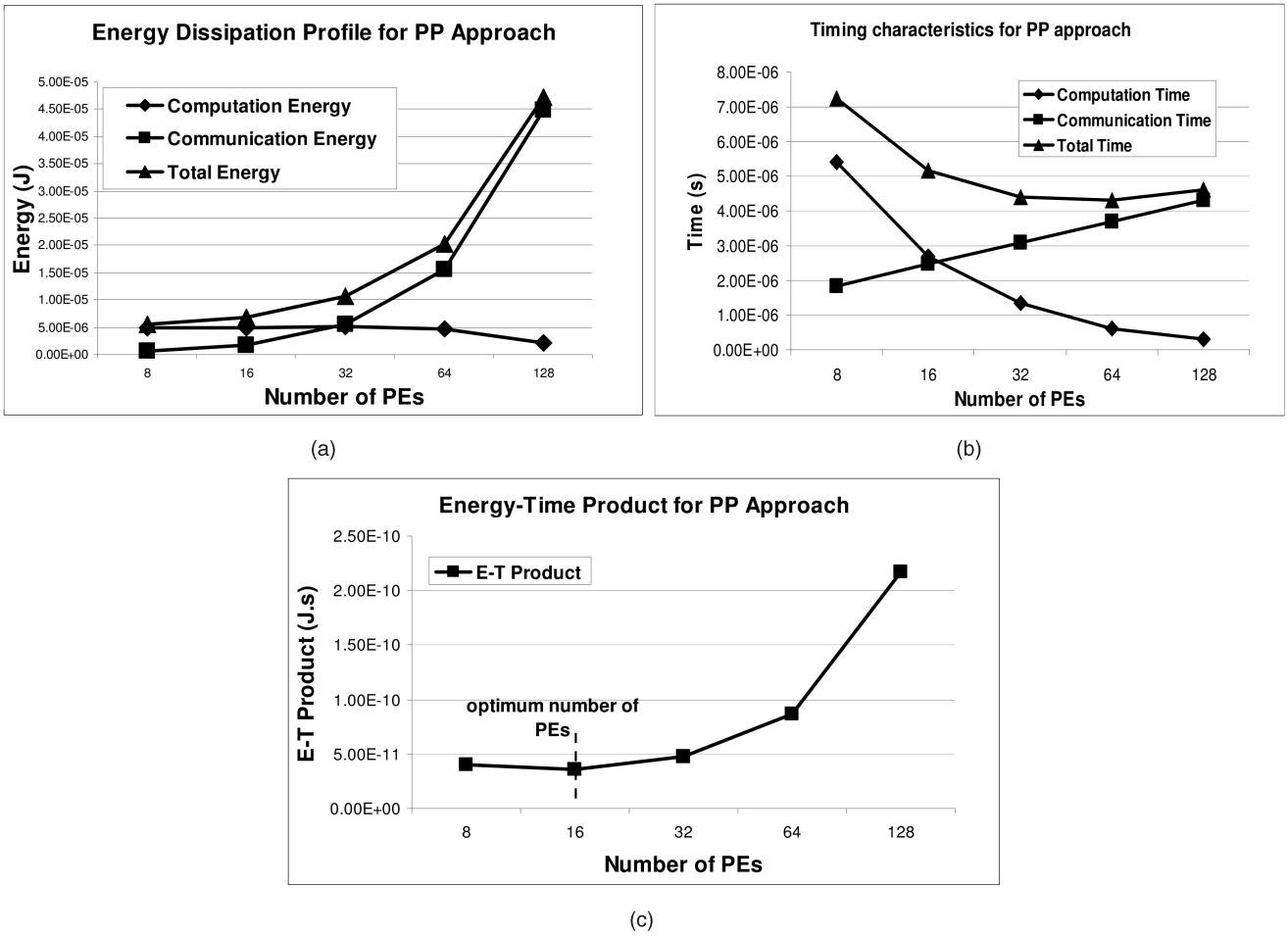


Fig. 11. (a) System energy profile. (b) Timing requirements. (c) The E-T product for PP approach.

there are m rows, where m denotes the length of string s_1, the total communication time for the entire alignment operation is $O(m \log p)$. Consequently, with increasing number of PEs, the communication time increases. At the same time with increasing system size, the number of columns in the DP table, and hence, the overall workload handled by each PE decrease. In fact, the computation time of each PE almost halves with doubling the system size until the input size becomes too small for the system size. This explains the observed trend of the computation time in Fig. 11b. Consequently, the total time needed to perform the alignment operation, which is the sum of the computation and the communication time, first decreases and reaches a valley, but beyond a certain number of PEs, it again starts increasing. In all our experiments, we observed that more than 90 percent of the overall time was spent on the forward pass of the algorithm.

To determine the optimum number of PEs, we consider the variation of the energy-time product with respect to the system size. In Fig. 11c, it can be observed that for comparing two strings of length 1,024, for the PP algorithm, the optimum number of PEs turns out to be 16.

5.3 Energy and Timing Characteristics of the AD Approach

Fig. 12 shows the energy and timing characteristics of our NoC implementation of the AD approach. Fig. 12a shows the energy characteristics. In the AD approach, the total number of communication steps during forward pass is $O(mn)$, irrespective of the system size, where m and n are the lengths of the two strings. This is because there is a data exchange at every cell of the DP table. Consequently, the communication energy in the forward pass remains unchanged with increase in the system size. However, during the backward pass, the communication energy increases with the system size due to the broadcast operation. This explains the slow rise in the total communication energy shown in Fig. 12a.

The computation energy is given by

$$Comp.Energy_{AD} = E_p \times \#PE \times \#Comp.cycles, \quad (9)$$

where E_p is the energy of a single processor per computation cycle. As the system size doubles, the number of total computation cycles per processor halves. Therefore, the product of these two factors is an invariant for a given input size. Reduction in E_p with increase in system size can be explained as follows: At any given time step, all the PEs are working on one antidiagonal. If the length of the antidiagonal is greater than the number of PEs, then the antidiagonal has to be computed in more than one parallel step. Consecutive parallel steps involve a certain number of computation stages performed by each PE as given in (6), which is inversely proportional to the system size. As a

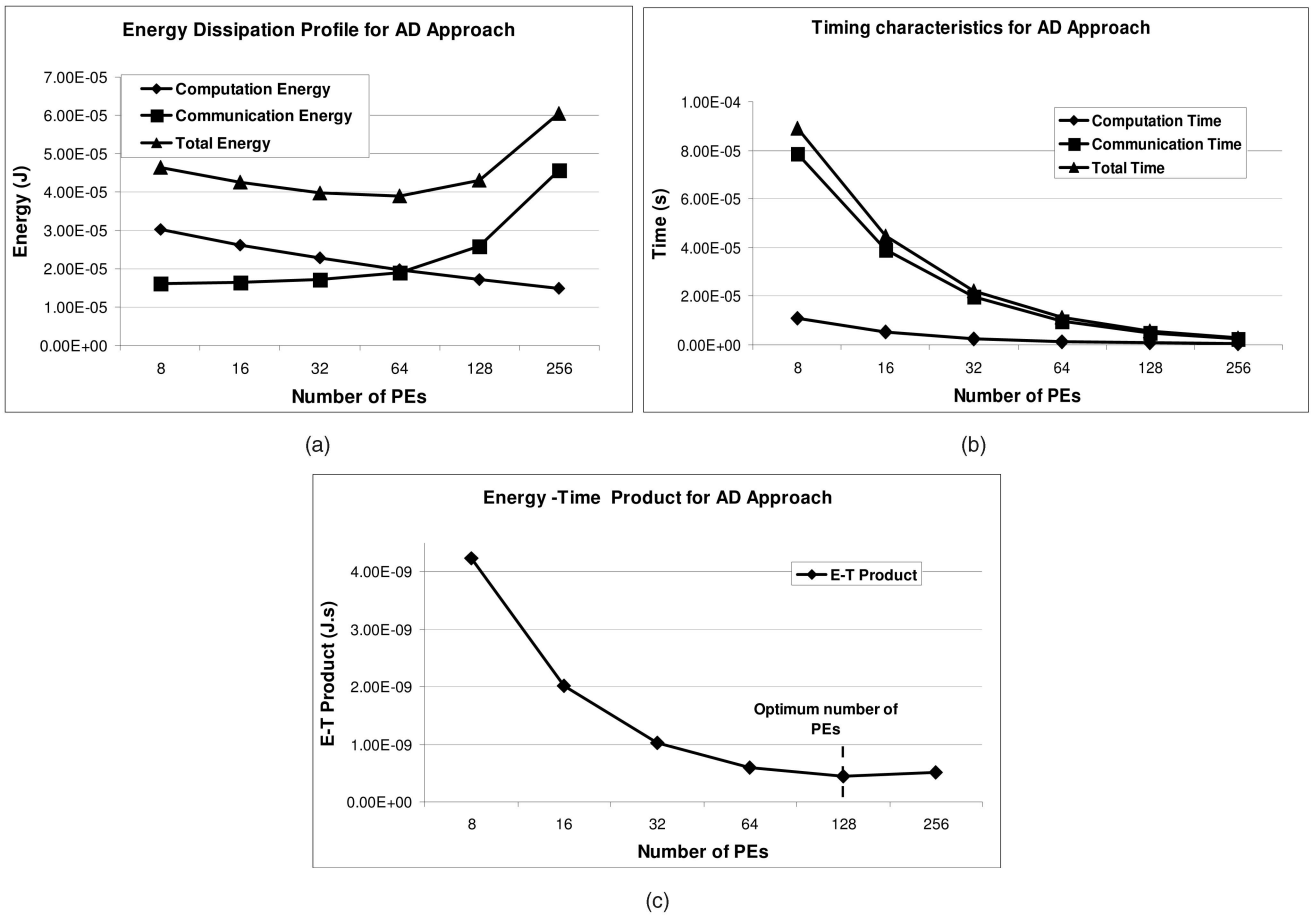


Fig. 12. (a) System energy profile. (b) Timing requirements. (c) The E-T product plot for AD approach.

result, if the number of PEs increases, the number of stages per antidiagonal computation reduces. Consequently, the number of times each PE is activated reduces, contributing to lower overall computation energy. This is confirmed in the results shown in Fig. 12a. Consequently, the total energy first reduces (following the decrease in computation energy) and then increases as communication energy starts to dominate, as shown in Fig. 12a.

Both communication and computation time complexities of the AD approach are $O(\frac{(m\times n)^2}{p})$. Since in our experiments, $\mathrm{m} = \mathrm{n}$, this is equivalent to $O(\frac{(m\times n)}{p})$. Therefore, as the system size doubles, the overall time halves as shown in Fig. 12b. Fig. 12c shows the energy-time product for the AD approach, and as can be observed, the optimum number of PEs is 128.

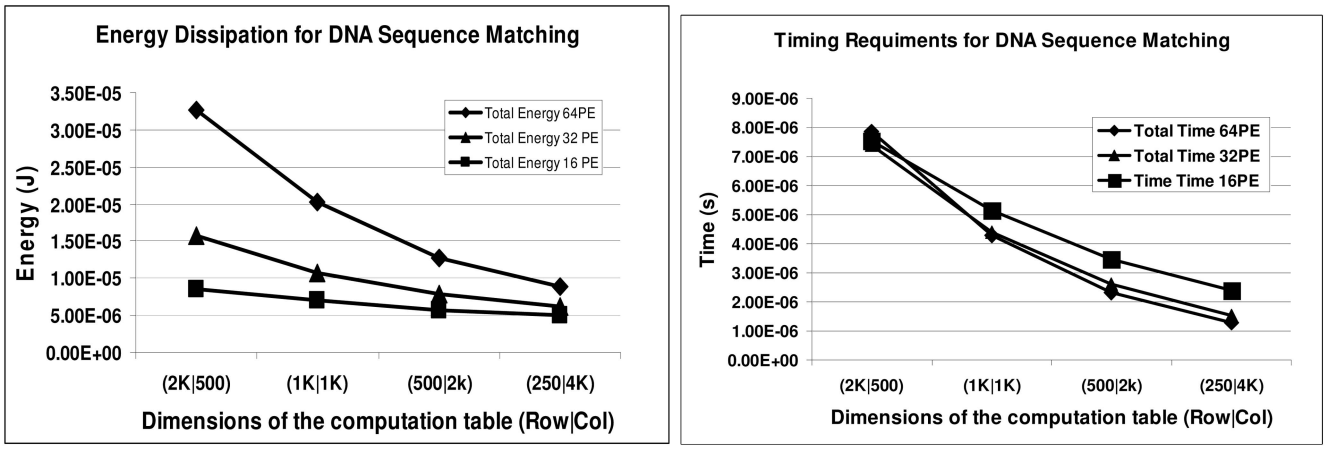
Table 1 presents a comparative performance evaluation between the PP and AD algorithms in terms of energy dissipation and timing. It is evident that the PP approach outperforms the AD both in terms of time and energy when the system size is less than or equal to 64. But, as we increase the number of processors beyond 64, the sharp rise in the communication energy in PP attributes to rise in its total energy. Thus, for large system sizes, AD approach outperforms PP in terms of energy dissipation, though it still takes more time.

5.4 Impact of Varying String Sizes

Initially, in our analysis, the lengths of the two strings were maintained equal. Here, we study the effect of comparing two strings of different lengths. Fig. 13 shows the impact of varying the string sizes on the timing and energy dissipation for the PP implementation. In order to allow a fair comparison, we varied the string sizes but keeping the total work (i.e., number of cells in the DP table) same. We considered four different combinations for s_1 and s_2. As the number of rows is decreased and/or the number of columns increased, the total time and energy both decrease. This can be explained by the decrease in the number of communication steps with decreasing number of rows, though the total amount of computation still

TABLE 1
Comparative Evaluation of PP and AD Schemes for 1KX1K Data

System Size	Energy (J)		Time (s)	
	PP	AD	PP	AD
8	5.57E-06	46.4E-06	7.24E-06	91.3E-06
16	7.01E-06	42.6E-06	5.15E-06	47.4E-06
32	10.7E-06	39.8E-06	4.41E-06	25.7E-06
64	20.3E-06	38.9E-06	4.30E-06	15.3E-06
128	47.2E-06	43.2E-06	4.61E-06	10.4E-06



(a) (b)

Fig. 13. (a) System energy profile. (b) Timing requirements plot obtained by varying the lengths of the strings using the PP implementation. The rows correspond to the characters in s1 and the columns correspond to the characters in s2.

remains the same. For the AD approach, the computation is always along the antidiagonal and the communication is only with the neighboring PE. Hence, there is no change in either time or energy when the string lengths were varied keeping the area of the DP table the same.

Next, we increase the number of bits from 3 to 4 for representing each of the characters of the DNA sequence to accommodate standard ambiguous character encoding. Tables 2 and 3 show the timing and energy statistics, respectively, using the PP implementation. It can be observed that the total time and total energy increase only marginally from 3 to 4 bit representation.

5.5 Parallel Prefix Implementation on Protein Sequence Data

We also undertook another case study with protein sequences, which contain one of 20 amino acid residues at each character position. In Fig. 14, we present the energy and timing results on amino acid sequences of length 256×256 (to reflect the average length of a protein sequence). We adopted the PP algorithm (5 bit representation for each protein character) and used the PAM substitution matrix [27] for assigning the score. The trend is very similar to that for the DNA sequences of length 256×256 except for an increase in time and energy. The energy increase has been a result of the increase in computation energy due to table lookup. There is no change in the communication energy. The increase in time is also due to the increase in computation time. The same trend as DNA sequence matching is expected while implementing using AD algorithm.

5.6 Discussion

To assess the real benefit that can be realized using our NoC implementation, we compared our runtimes against other hardware accelerator implementations and our own serial implementation on a 2.3 GHz Xeon CPU. The results are tabulated in Tables 4 and 5. The time needed to transfer the sequences from main memory and writing back the resultant path back to the main memory depends on the adopted interface mechanism. If the NoC-based chip is used as a coprocessor on the same mother board as the main processor, then the total time was about 0.096 μs. This number is derived using a bus width of 128 and a bus speed of 1,333 MHz. On the other hand, if the PCI express 3.0 is chosen as the interfacing standard, then the timing requirement is higher, which is around 2 μs. The coprocessor-based implementation overhead is included in the timing data for our NoC implementation presented in Tables 4 and 5. As can be observed, our PP-based NoC implementation provides

TABLE 2
Timing Comparison for 1KX1K Data

Number of PEs	3 bit (s)	4 bit (s)
64	4.30E-06	5.27E-06
32	4.41E-06	5.92E-06
16	5.15E-06	6.09E-06

TABLE 3
Energy Comparison for 1KX1K Data

Number of PEs	3 bit (J)	4 bit (J)
64	20.3E-06	21.9E-06
32	10.7E-06	11.4E-06
16	7.01E-06	7.50E-06

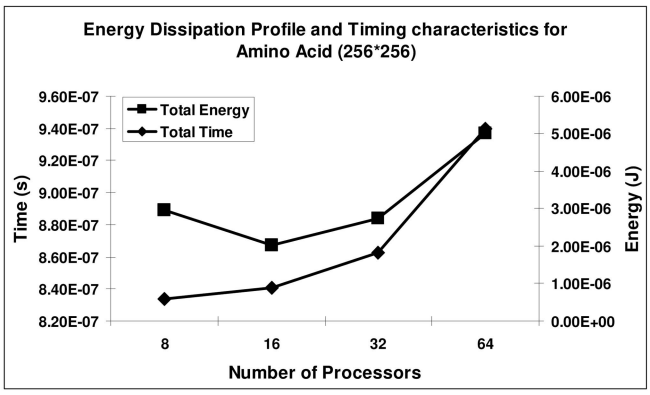


Fig. 14. Energy dissipation profile and timing requirements for AA data.

TABLE 4
Speedup of Various Accelerators over Our Serial Implementation

	Intel 2.3GHz Xeon CPU	GPU [8]	CBE [9]	CBE [10]	FPGA [7]	OUR NoC IMPLEMENTATION	
						PP	AD
Time (ms)	100	1.43	0.65	17.5	1	0.00439	0.01054
Speedup over serial implementation	1	69.93	153.85	5.7	100	22779.04	9487.667

between 150 and 4,000-fold speedup over other existing accelerators and more than well over 10^4-speedup over the serial implementation. If the PCI-based interface is used, then our implementation also achieves between 100 and 2,700-fold speedup over the existing accelerators.

These estimates go well beyond demonstrating the paradigm-shifting potential of NoC architectures over bioinformatics applications. For example, the analysis of over 28 million metagenomic sequences, which took months to complete after parallelization at the coarse level [4], can be completed in a matter of days using our NoC-based hardware accelerator. The NoC architecture can not only provide such high performance improvements but also, more importantly, enable solving much larger problems than was ever possible before under practical experimental settings.

Our research has also laid out a design template for the future development of new acceleration models for other related applications in bioinformatics. For instance, the BLAST algorithm [28], which is an approximation method for computing sequence alignments, is a popular tool for detecting performing sequence database searches. Due to large sequence database sizes, accelerating BLAST search operations is performance-critical. Nevertheless, the underlying algorithm in BLAST also uses the Smith-Waterman algorithm, while also implementing a prefiltering process prior to computing alignment using a string lookup table data structure. Consequently, an offshoot of our research could be that NoC can be explored as a viable means for acceleration for BLAST as well. Before such a project is undertaken, however, a feasibility study should be conducted to assess both the quality degradation, which is possible due to approximation, along with the performance impact due to implementing additional string data structures.

TABLE 5
Speedup over Existing **H**ardware **A**ccelerators

Other Accelerators		FPGA [7]	CBE [9]	CBE [10]	GPU [8]
Our Implementation	PP	227.79	148	3986.3	325.74
	AD	94.87	61.67	1660.3	135.67

6 Conclusion

Computational biology research has reached a critical juncture, where the rate of data generation is rapidly outpacing the rate at which it is processed. To close this gap, hardware accelerators that can achieve significant speedups are needed. Efficient accelerators can be designed by integrating high number of processing cores in a single die to exploit the coarse-grain parallelism inherent in bioinformatics applications. The NoC paradigm enables this high degree of integration. In this paper, we presented the first optimized NoC architecture to efficiently parallelize sequence alignment on chip. Design of the NoC is optimized to significantly reduce the latency in communication between the processing elements. By achieving significant speedups over serial implementations and other state-of-the-art hardware accelerators, we demonstrate the high impact and transformative potential that NoC architectures are capable of producing in bioinformatics research.

References

[1] Benson et al., "GenBank," *Nucleic Acids Research,* vol. 35, pp. D21-D25, 2007.

[2] T.F. Smith and M.S. Waterman, "Identification of Common Molecular Subsequences," *J. Molecular Biology,* vol. 147, pp. 195-197, 1981.

[3] S.B. Needleman and C.D. Wunsch, "A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins," *J. Molecular Biology,* vol. 48, pp. 443-453, 1970.

[4] S. Yooseph et al., "The Sorcerer II Global Ocean Sampling Expedition: Expanding the Universe of Protein Families," *Public Library of Science Biology,* vol. 5, no. 3, 2007, doi:10.1371/journal.pbio.0050016.

[5] J. Thompson et al., "CLUSTALW: Improving the Sensitivity of Progressive Multiple Sequence Alignment through Sequence Weighting, Position-Specific Gap Penalties and Weight Matrix Choice," *Nucleic Acids Research,* vol. 22, pp. 4673-4680, 1994.

[6] S. Dydel and P. Bala, "Large Scale Protein Sequence Alignment Using FPGA Reprogrammable Logic Devices," *Proc. Conf. Field-Programmable Logic and Its Applications,* pp. 23-32. 2004.

[7] T. Oliver et al., "Using Reconfigurable Hardware to Accelerate Multiple Sequence Alignment with ClustalW," *Bioinformatics,* vol. 21, no. 16, pp. 3431-3432, 2005.

[8] W. Liu, B. Schmidt, G. Voss, and W. Mueller-Wittig, "Streaming Algorithms for Biological Sequence Alignment on GPUs," *IEEE Trans. Parallel and Distributed Systems,* vol. 18, no. 9, pp. 1270-1281, June 2007.

[9] V. Sachdeva et al., "Exploring the Viability of the Cell Broadband Engine for Bioinformatics Applications," *Parallel Computing,* vol. 34, no. 11, pp. 616-626, 2008.

[10] A. Sarje and S. Aluru, "Parallel Biological Sequence Alignments on the Cell Broadband Engine," *Proc. IEEE Int'l Parallel and Distributed Processing Symp.,* 2008.

[11] S. Vangal et al., "An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS," *IEEE J. Solid State Circuits,* vol. 43, no. 1, pp. 29-41, Jan. 2008.
[12] I.A. Khatib et al., "A Multiprocessor System-on-Chip for Real-Time Biomedical Monitoring and Analysis: Architectural Design Space Exploration," *Proc. IEEE Design Automation Conf. (DAC),* pp. 125-130, July 2006.
[13] L. Benini and G. De Micheli, "Networks on Chips: A New SoC Paradigm," *Computer,* vol. 35, no. 1, pp. 70-78, Jan. 2002.
[14] P.P. Pande et al., "Performance Evaluation and Design Trade-Offs for Network on Chip Interconnect Architectures," *IEEE Trans. Computers,* vol. 54, no. 8, pp. 1025-1040, Aug. 2005.
[15] S. Aluru et al., "Parallel Biological Sequence Comparison Using Prefix Computations," *J. Parallel and Distributed Computing,* vol. 63, pp. 264-272, 2003.
[16] E.W. Edmiston and R.A. Wagner, "Parallelization of the Dynamic Programming Algorithm for Comparison of Sequences," *Proc. Int'l Conf. Parallel Processing,* pp. 78-80, 1987.
[17] X. Huang, "A Space-Efficient Parallel Sequence Comparison Algorithm for a Message-Passing Multiprocessor," *Int'l J. Parallel Programming,* vol. 18, no. 3, pp. 223-239, 1989.
[18] S. Rajko and S. Aluru, "Space and Time Optimal Parallel Sequence Alignments," *IEEE Trans. Parallel and Distributed Systems,* vol. 15, no. 12, pp. 1070-1081, Dec. 2004.
[19] O. Gotoh, "An Improved Algorithm for Matching Biological Sequences," *J. Molecular Biology,* vol. 162, pp. 705-708, 1982.
[20] A. Apostolico et al., "Efficient Parallel Algorithms for String Editing and Related Problems," *SIAM J. Computing,* vol. 19, no. 5, pp. 968-988, 1990.
[21] D. Gusfield, *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology.* Cambridge Univ. Press, 1997.
[22] D.S. Hirschberg, "A Linear Space Algorithm for Computing Maximal Common Subsequences," *Comm. ACM,* vol. 18, no. 6, pp. 341-343, 1975.
[23] A.R. Butz, "Alternative Algorithm for Hilbert's Space Filling Curve," *IEEE Trans. Computers,* vol. 20, no. 4, pp. 424-426, Apr. 1971.
[24] J.M. Rabaey et al., *Digital Integrated Circuits: A Design Perspective.* Prentice Hall, 2003.
[25] J. Kim et al., "Flattened Butterfly Topology for On-Chip Networks," *IEEE Computer Architecture Letters,* vol. 6, no. 2, pp. 37-40, July-Dec. 2007.
[26] F. Sanger et al., "Nucleotide Sequence of Bacteriophage Lambda DNA," *J. Molecular Biology,* vol. 162, pp. 729-773, 1982.
[27] M.O. Dayhoff, R.M. Schwartz, and B.C. Orcutt, "A Model of Evolutionary Change in Proteins," *Atlas of Protein Sequence and Structure,* vol. 5, no. 3, pp. 345-352, Nat'l Biomedical Research Foundation, 1978.
[28] S.F. Altschul et al., "Basic Local Alignment Search Tool," *J. Molecular Biology,* vol. 215, pp. 403-410, 1990.

Souradip Sarkar received the MS degree in computer engineering from Washington State University in 2007 and the MTech degree in computer science from Indian Statistical Institute, Kolkata, in 2006. He is currently working toward the PhD degree at the School of Electrical Engineering and Computer Science at Washington State University in Pullman. His research interests include network-on-chip and system-on-chip implementations of algorithms for computational genomics applications, reconfigurable system architectures, and synchronization issues in multiple clock domain platforms. He is a student member of the IEEE.

Gaurav Ramesh Kulkarni received the bachelor's degree in computer science and engineering from the University of Pune, India, in 2007. He is currently working toward the master's degree at Washington State University, Pullman. His primary area of interest includes computational biology and high-performance computing. He is a student member of the IEEE.

Partha Pratim Pande received the MS degree in computer science from the National University of Singapore in 2002 and the PhD degree in electrical and computer engineering from the University of British Columbia in 2005. He is an assistant professor at the school of Electrical Engineering and Computer Science, Washington State University, Pullman. His primary research interests lie in the areas of design and test of networks on chip, fault tolerance and reliability of multiprocessor SoC (MP-SoC) platforms, 3D integration, and on-chip wireless communication network. He is serving on the program committees of different international conferences like IOLTS, ATS, MWSACS, DELTA, and NoCs. He is a member of the IEEE.

Ananth Kalyanaraman received the BE degree in computer science and engineering from Visvesvaraya National Institute of Technology in Nagpur, India, in 1998, and the MS and PhD degrees in computer science and computer engineering from the Iowa State University, in 2002 and 2006, respectively. He is an assistant professor at the School of Electrical Engineering and Computer Science, Washington State University (WSU) in Pullman. He is also an affiliate faculty in the Molecular Plant Sciences graduate program and the Center for Integrated Biotechnology at WSU. Subsequently, he joined the Computational Biology and Scientific Computing Laboratory at the Iowa State University. His research interests include computational biology and bioinformatics, parallel algorithms and applications, and combinatorial pattern matching. The primary focus of his work has been on developing high-performance computing algorithms and software for data-intensive problems in computational genomics. He is a recipient of the Best Paper Awards at IPDPS 2006 and CSB 2005, and doctoral fellowships from IBM Research and Pioneer Hi-Bred International, Inc. He has developed several parallel programs, which are collectively used in more than 100 research and corporate organizations for large-scale computational genomics. He has served on several program committees including SC 2008, ICPP 2007, IPDPS 2007, HiPC 2008, and HiCOMB 2008. He is a member of the ACM, the IEEE, the ISCB, the LSS, and the SIAM. He is a member of the IEEE.