An Experimental Study of Optimizing Bioinformatics Applications

Guangming Tan[1,2], Lin Xu[1,2], Shengzhong Feng[1], Ninghui Sun[1]
[1] Institute of Computing Technology Chinese Academy of Sciences
[2]Graduate School of Chinese Academy of Sciences
{tgm,xulin,fsz,snh}@ncic.ac.cn

Abstract

As bioinformatics is an emerging application of high performance computing, this paper first evaluates the memory performance of several representative bioinformatics applications so that some appropriate optimization methods can be applied. Based on the computational behavior of these bioinformatics applications, we propose two optimized algorithms on high performance computer architectures. 1) For the data(I/O) intensive program, MegaBlast, we overlap computation with I/O to produce an improved high-throughput algorithm with reduced time and memory requirements. 2) For a CPU-intensive RNA secondary structure prediction algorithm, we propose a fine-grain parallel $O(N^3)$ algorithm based on reconfigurable arrays (FPGAs). In order to optimize the FPGA architecture, we evaluate the performance in different architectures using cycle-by-cycle simulator.

1. Introduction

Bioinformatics applications are attracting great effort from high performance computing. Sequence alignment [1] and structure prediction [2] in bioinformatics are exerting great pressure on the processing ability of current computer systems[3]. Sequence alignment is a process of scanning gene and protein sequence database. It is a common and often repeated task in molecular biology. The need for speeding up this process comes from the exponential growth of the biosequence banks: every year their size increases by a factor of 1.5 to 2. Comparison algorithms, whose complexities are quadratic with respect to the length of the sequences, detect similarities between the query sequence and subject sequence. One frequently used approach to speed up this time consuming operation is to introduce heuristics in the search algorithms, such as BLAST [4] and FASTA [5]. The main drawback of this solution is that the more time efficient the heuristics is, the worse is the quality of the results. RNA secondary structure prediction algorithm shows high time complexity, the most widely used minimum free energy method [2] is an $O(N^4)$ dynamic programming algorithm [6]. By analyzing the energy rules or restricting the size of loop, the time complexity of algorithm is reduced to $O(N^3)$, but it is still time consuming when the length of sequence becomes long.

Memory access overhead is becoming more significant and has been widely investigated in scientific computing[8]. On the one hand, it is necessary to develop high efficient algorithms to overcome the memory wall. On the other hand, an important trend in high performance computing is to develop an innovative architecture [9][10]. As an emerging computer architecture, FPGA based reconfigurable computing [11] has shown a magnitude speedup over the performance of standard microprocessor CPUs owing to its intrinsic spatial parallel and integrated local memory [11]. In order to exploit high performance algorithms in bioinformatics, we select several popular applications and investigate their memory system performances. The rest of this paper is organized as follows. Section 2 provides the study of memory system performance of some bioinformatics applications. In section 3 and 4, we propose two optimized algorithms and evaluate their performance. Section 5 concludes this paper.

2. Memory System Performance

We use hardware performance counters (Oprofile [15] and PAPI [16])as well as executive-driven simulation in this study. In order to study cache behavior under different configurations, we performed extensive executive-driven simulation experiments, using SandFox [17], which is Vmips [18] based executive-driven simulator developed by one group in our institute. The

1-4244-0054-6/06/$20.00 ©2006 IEEE

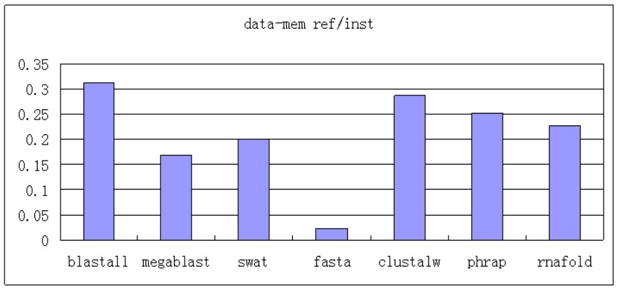

Figure 1. The average number of memory references per instruction is 0.224 for bioinformatics applications

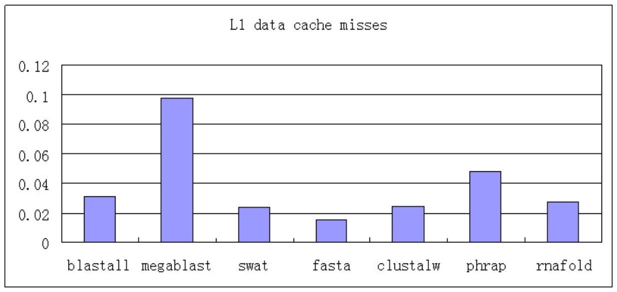

Figure 2. L1 D-cache performance seven bioinformatics applications

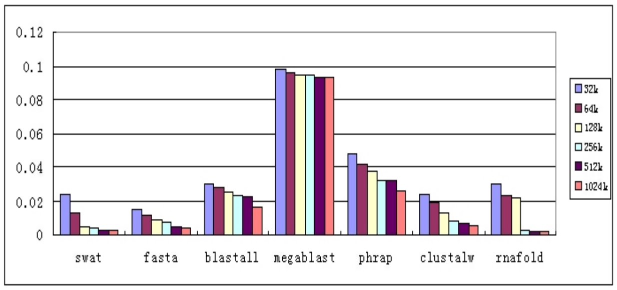

Figure 3. L1 D-cache performance for bioinformatics applications with different cache size

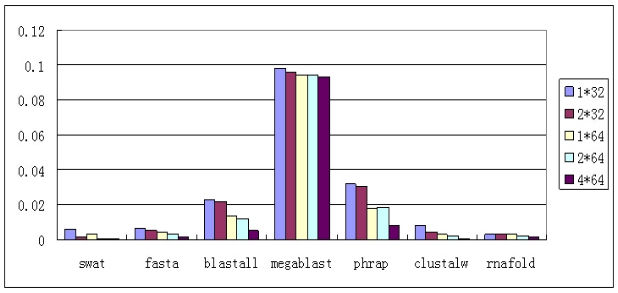

Figure 4. L1 D-cache performance for bioinformatics applications with different cache line size and associativity (associativity*line size). The size of cache is 256KB

used application set comprises six sequence alignment programs (swat[6], fasta[5], blastall[4], megablast[4], phrap[19], clustalw[7]) and one RNA secondary structure prediction program ViennaRNA/rnafold [20]. At the same time, Bader et al. [12][13][14] gives more comprehensive evaluations of bioinformatics applications.

We noted that bioinformatics applications used here spend more than 99 percent of program execution time in user mode. We ignore the OS effects without affecting the accuracy. Figure 1 indicates that bioinformatics applications have a low number of average memory references per instruction (0.224). In the three heuristic sequence alignment programs, the number of average memory references per instruction is not the highest, but both the number of memory references and instructions are the largest because of its great memory requirement. An important question is how well the cache performs under this traffic. Figure 2 shows the measured results for all applications. Figure 2 indicates the cache miss rate of bioinformatics applications is very low. In seven bioinformatics applications, megablast has the highest data cache miss rate. In section 3, we propose an optimized algorithm to improve memory and cache performance.

In order to comprehensively study the cache performance for bioinformatics applications, we then used SandFox to simulate the cache behavior for these applications. Figure 3 and 4 show the simulation results for seven bioinformatics applications, with different cache configurations. We observed that except for megablast, the other six applications show sensitive to the different cache configuration to some extent.

In this section, we performe a comprehensive study of the memory requirements of a group of representative bioinformatics applications. Our observations suggest that all seven bioinformatics applications except megablast are CPU-bound. For megablast, we will focus on the improvement of memory system by software optimization methods. While the memory system improvement of other five programs is necessary, it is difficult to improve the cache performance through software algorithm optimization technology. We concentrate more on increasing processing power on some specific architecture.

3. Improving Memory Efficiency of Megablast

3.1. Original Megablast Algorithm

Because of batch processing and greedy algorithm in extending significant similar segments [21], MegaBlast is the fastest and high-throughput program in the NCBI BLAST toolkits [22]. The basic flow of MegaBlast is the same with other programs in NCBI BLAST. Assume that a set of query sequences $Q = \{q_1, q_2, ..., q_m\}$ and subject sequences $S = \{s_1, s_2, ..., s_n\}$, the length of word or hit is w. MegaBlast algorithm is described as Algorithm 1(For detail, refer to [5]).

Algorithm 1. MegaBlast

```
Build_hash_table(Q); /*build a hash table for all
                       query sequences, which is
                       considered as one sequence*/
for $s_i$ in S /*all sequences in database*/
    Find_seed(hash_table, $s_i$); /*find hits*/
    Extend_filter_seed( ); /*Extending hits and get
                             HSPs*/
    Sort_hsp( ); /*insert HSPs into some priority
                   queue*/
endfor
Output( ); /*output alignment results*/
```

There are two problems in MegaBlast algorithm. First, because HSPs are selected from the alignment results, which are generated by aligning one sequence with all database sequences, all alignment results are kept in memory until it finishes searching all subject sequences. In the worst case, the memory cost is proportional to the product of the sizes of two sequence sets being compared.

$$Memory_1 = O(c_1 * |Q| * |S|) \tag{1}$$

where $|Q|$ and $|S|$ is size of two sets of sequences, c_1 is relative with similarity between two sequences. We note that c_1 is larger when the similarity is higher. When size of sequences set is large and similarity is high, the memory cost will increase. Second, MegaBlast is a computation and I/O intensive program. For the large size of sequences, I/O almost occupies half of the overall time. The performance of the I/O subsystem also affects overall performance of application programs. In MegaBlast, the CPU is idle when the I/O subsystem is outputting the alignment results to disk file. Since high memory and I/O overhead are the two main bottlenecks of MegaBlast algorithm, we propose a solution to those bottlenecks.

3.2. High Spatial-temporal Efficient Megablast Algorithm

We note that the query sequences and subject sequences are symmetric and exchanging query and database sequence promise the correct alignment results for finding hits. We exchange the set of query sequences with the set of database sequences and build a hash table based on the database sequences. The optimized algorithm is described Algorithm 2.

Algorithm 2. ste_blast

```
Build_hash_table(S) /*build a hash table for all
                      subject sequences, which is
                      considered as one sequence*/
for $q_i$ in Q /*all query sequences*/
    Find_seed(hash_table, $q_i$) /*find hits*/
    Extend_filter_seed( ); /*Extending hits and get
                             HSPs*/
    Sort_hsp( ); /*insert HSPs into some priority
                   queue*/
    Output( ); /*output the alignment of $q_i$*/
endfor
```

It places **Output** into the for loop, which actually eliminates the accumulation of memory requirement. The cost of memory is proportional to the size of subject sequences set, instead of the product of the sizes of two sequence sets:

$$Memory_2 = O(c_2 * |S|) \tag{2}$$

where $|S|$ is the size of subject sequence set and c_2 is a similar parameter as c_1 in equation (1). The proportion of memory requirement is:

$$Memory_1/Memory_2 = O(c_1 * |Q|/c_2) 0 < c_1/c_2 \leq 1 \tag{3}$$

When each alignment generate the same size of results, $c_1/c_2 = 1$ and the maximum of two algorithms is the same and the optimized algorithm achieves the best performance. When the size of one sequence alignment results is far larger than other sequence, the memory requirement of the optimized algorithm is determined by the maximum cost of memory and achieves the lowest performance. The optimized MegaBlast algorithm outputs alignment results as soon as one query sequence finishes aligning. The task of results output

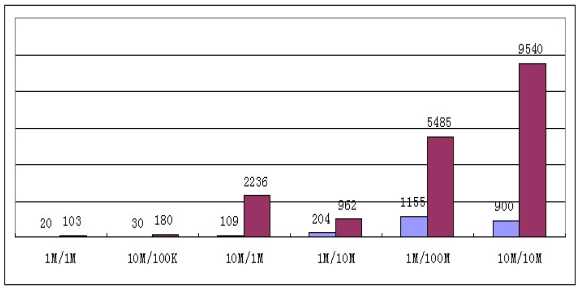

Figure 5. The comparison of runtimes in seconds of two BLAST algorithms. Each right column represents the original Megablast algorithm. For 1MB/1MB, the optimized algorithm is 20 seconds and Megablast algorithm is 103 seconds. For 10MB/100KB, the time is 30 seconds and 180 seconds, respectively.

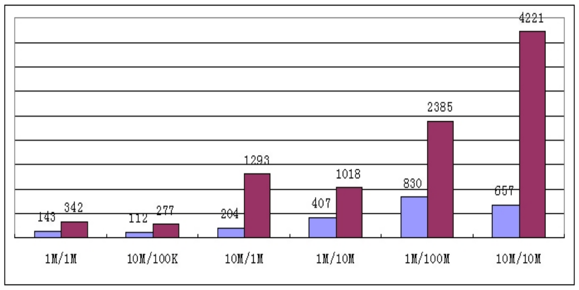

Figure 6. The comparison of memory in MB of two BLAST algorithms. Each right column represents the original Megablast algorithm. For 1MB/1MB, the optimized algorithm is 143MB and Megablast algorithm is 342MB. For 10MB/100KB, the memory usage is 112MB and 277MB, respectively.

to disk is managed by I/O controller and CPU continues aligning next query sequences. The optimized algorithm achieves the overlap of computation with I/O, decreases the overall time.

3.3. Performance Evaluation

We selected mouse embryo EST sequences to measure the performance of the optimized algorithm on a platform with 1.6GHz Opteron, 4.5GB memory. As shown in Figure 5 and Figure 6, the optimized MegaBlast greatly reduces the running time and memory usage.For the alignment between 10MB query sequences set and 10MB database sequences set, when the sizes of two sequences sets are comparative, the optimized algorithm shows higher superiority. From equation 12, the value c_1/c_2 is less than 1. However, when the size of the subject sequences set is much larger than the size of the query sequences set, the hash table in the optimized algorithm is larger than in the original Megablast algorithm. In the case of 1MB query sequences set and 10MB database sequences set, the original Megablast algorithm needs 129MB memory and the optimized algorithm needs 389MB memory for building hash table. Because the whole hash table is kept in memory, the value $Memory_1/Memory_2$ is not linearly proportional to the size of query sequences set. Another factor determining the value $Memory_1/Memory_2$ is the memory-access mode in program. Both algorithms read their own subject sequences set through memory-map. When the size of the subject sequences is larger than the size of query sequences set, the original Megablast algorithm only allocates physical memory for part of the whole database sequence, but the whole database sequences are kept in physical memory as a hash table. Furthermore, because building hash table needs more time, these factors also make the running time of the optimized algorithm longer. For example, with the alignment of 1MB query and 10MB subject sets, the original Megablast algorithm only spends 0.05 seconds (where overall time is 962 seconds) and the optimized algorithm spends 88 seconds (where overall time is 204 seconds). So the optimized algorithm cannot achieve ideal performance.

Although the optimized algorithm can promise that the hits(seed) are the same with that of the original MegaBlast, the final alignment results should be filtered by a statistic score, which is based on the size of searching space. However, the search space is determined by effective length of subject sequences in NCBI BLAST. So exchanging query and database sequence changes the searching space, thus the final results are different from that of the original MegaBlast. One solution is to develop a new score calculation scheme from a viewpoint of mathematics. In fact, we note that the difference between the results of two algorithms is only the number of final alignment. The bars in Figure 7 show the comparison of the numbers of successful extensions, which are determined by the statistical score. The results difference is only the number of final alignments, the statistics results describe the percentage of results that are different for the different queries (See the line in Figure 7). We notice that the optimized algorithm can find more results than the original MegaBlast in some cases.

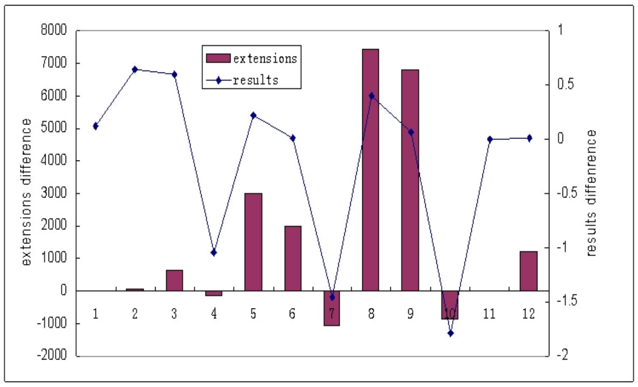

Figure 7. The comparison of the number of successful extensions. The sizes of sequence sets are 100KB, 1MB, 5MB, 10MB, respectively. The plus represents that the number of results of the optimized algorithm is larger than the original MegaBlast, The minus represents that the number of results of the optimized algorithm is smaller than the original MegaBlast.

4. Exploiting Fine-grain Parallelism for RNA Folding

4.1. Zuker's Algorithm

In Zuker's dynamic programming algorithm(The detail and notation refer to [2]), the procedure to compute the energy of the optimal structure of an RNA sequence is a nested dynamic programming. The energy of the optimal structure of subsequence $r_1, r_2, ..., r_i$ is W_i:

$$W(i) = min\{W(i-1), min_{1<j\leq i}\{W(j-1) + V(j,i)\}\} \quad (4)$$

$V(i,j)$ is the energy of the optimal structure of subsequence $r_i, r_{i+1}, ..., r_j$, where $(r_i,r_j)$is pair.

$$V(i,j) = f(V(i+1,j-1), VBI(i,j), VM(i,j)) \quad (5)$$

$VBI(i,j) = \{eL(i,j,i',j') + V(i',j')\}$ is the energy of an optimal structure of the subsequence from i through j where (i,j) closes a bulge or an internal loop. VM is the energy of an optimal structure of the subsequence from i through j where (i,j) closes a multibranched loop. $VM(i,j)$ can be computed as:

$$VM(i,j) = min\{WM(i+1,k-1) + WM(k,j-1)\} \quad (6)$$

$$WM(i,j) = min\{V(i,j), min\{WM(i,k-1) + WM(k,j)\}\} \quad (7)$$

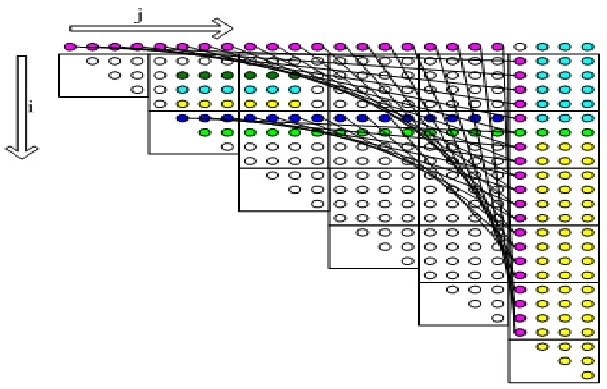

Figure 8. The data dependence graph and stripped matrix. Each entry depend on the entry along the same row and column

Matrix V is the core which all equations depend on. In the computing procedure of VBI, $VBI(i,j)$ depends on all the left-bottom region of $V(i,j)$. $VM(i,j)$ depends on the bottom line and left line of $WM(i,j)$, they are $WM(i+1,k-1)(i+1 < k \leq j-1)$ and $WM(k,j-1)(i+1 < k \leq j-1)$. $WM(i,j)$ depends on the corresponding cell in V, that is $V(ij)$, and the left cells and the bottom cells of $W(i,j)$, they are $WM(i,k)(i < k \leq j)$ and $WM(k,j)(i < k \leq j)$.

4.2. Fine-grained Parallel Algorithm

The order of serial computing V is from bottom to top line by line, from left to right in each line. The computing procedure of adding $WM(i,i...j-1)$ with the corresponding $WM(i+1...j,j)$ can be pipelined. We can find that both $WM(i,j)$ and $WM(i,j+1)$ depend on $WM(i,i...j-1)$. Therefore we can utilize the line $WM(i+1...j,j)$ repeatedly. That forms a parallel computing. Figure 8 and 9 demonstrate that each cell in block can be computed parallel by FPGA array. Each processing unit in FPGA executes an adding operation. Then the minimum of each line is calculated and the result is output. In the procedure the matrix V is saved to backtrack. Suppose a sequence which length is n can be divided into M strips, where $M = \lceil n/DEPTH \rceil$, each data block is $Block(i,j)(1 \leq i \leq j \leq M)$. Taking $Block(i,j)$as an example we explain the procedure of our parallel algorithm as follows: After PE array's computing procedure, only the element at the left-bottom in $Block(i,j)$ is the final result, the remainders depend on the elements in $Block(i,j)$. Therefore the complementary computing is to finish computing the

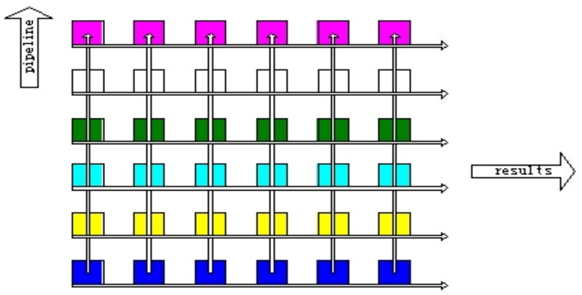

Figure 9. The FPGA arrays

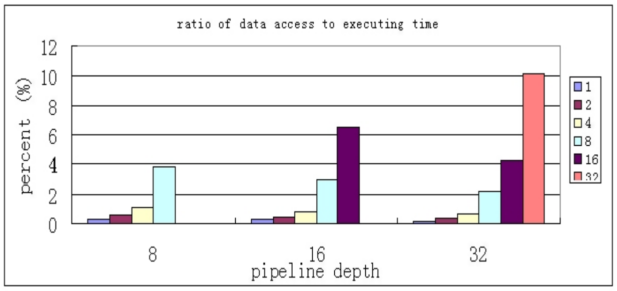

Figure 10. Sequence length: 880bps. The ratio of data transmission time to executing time. The parallel width is 1, 2, 4, 8, 16 and 32. The maximum parallel width is 8 for pipeline depth 8 and the maximum parallel width is 16 for pipeline depth 16

whole block. In order to determine a proper parallel and pipeline level, we implement a cycle-by-cycle simulator for FPGA and map the fine-grained parallel algorithm to the FPGA simulator. The performance analysis focuses on memory access, pipeline bubble and executing time.

Algorithm 3.

```
for($j = 1; j < J; j++$) {
    Mapping each cell in $Block(i, j)$ into the register
    input1 of the FPGA array's corresponding PE;
    for($i = I + 1; i < M; j++$)
        for($k = 0; k < DEPTH; k++$) {
            Input a row element of $Block(i, j)$ from left to
            right row by row into registers input2 in the
            bottom line of PE and put register input2 in
            each line into the forward line;
            Get the sum of the elements in each line, saved
            in output, and get the minimum of the outputs
            in each line, output the minimum into the
            corresponding position in WM;
        }
        $Block(i, j)$'s complementary computing;
}
```

4.3. Performance Evaluation

Because of the continuous updating of computation cell and many memory cells needing to save the temporal values, memory access is important to the performance of the fine-grain parallel algorithm. The simulator assumes the results can be denoted by a 20bit-length data. We use D as pipeline depth and P as parallel width. The data bus width is 128bits and the frequency is 133Mhz. The data of initializing PE array is D*P*20bits. The initializing procedure time is $\lceil \frac{D*P*20}{128} \rceil / 133\mu$s. Updating array in pipeline, that is updating register input2, needs P*20bits data and the time is $\lceil \frac{P*20}{128} \rceil / 133\mu$s. The data of output result is DEPTH*20bits and the output time is $\lceil \frac{D*20}{128} \rceil / 133\mu$s. Figure 10 indicates that the time of memory access increases with increasing the pipeline depth and parallel width. If the pipeline depth is constant, the number of reused data will be constant. However, the number of input data every time and the clock cycle of transmitting data will increase when the parallel width increases. Thus the ratio of memory access to overall executing time will increase. If the parallel width is constant, the increasing pipeline depth will reduce the ratio of memory access to overall executing time because of the increasing number of reused data for the same number of input data. In Figure 10, a parallel width of 8 is used as an example.

Bubble means the proceeding unit is idle at a time. From Figure 11 we can conclude that the bubble is increasing with the pipeline depth and the increase is close to linear one. When the pipeline depth and parallel width are fixed, the bubble is increasing with the sequence length. Figure 12 demonstrates that the bubble ratio in different sequence length. We can conclude that the array's utilizing ratio is increasing with the sequence length.

The simulator assumes adding and min operation can be accomplished in 1 cycle clock, and VBI and WM was synchronously computed. We chose two different RNA sequences of different length, which are 880bps and 2313bps. The executing time in different pipeline depth and parallel width is demonstrated in Table 1 and 2 (the FPGA's main frequency is 100MHz). The results demonstrates for the same RNA sequence for a fixed parallel width speedup increases with the pipeline depth. Similarly, for a fixed pipeline depth the speedup increases with the parallel width. From the simula-

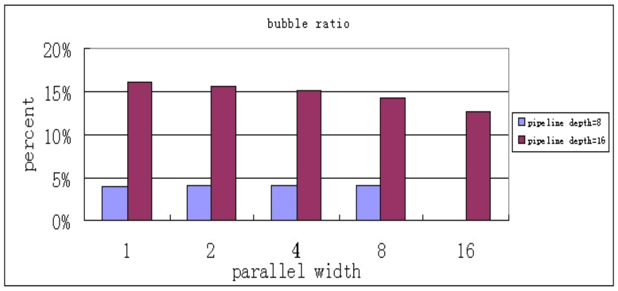

Figure 11. Sequence length: 880bps.The bubble distribution with pipeline depth and parallel width

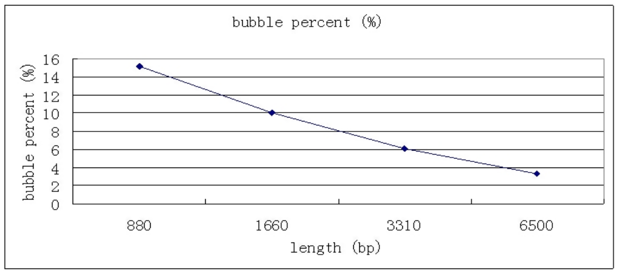

Figure 12. The bubble percent to sequence length. The pipeline depth is 16 and parallel width is 8

Table 1. Length=880bps the base of speedup is the executing time (=2280ms) under Xeon2.4GHz. Time: second. D: Pipleline depth, P: Parallel width, T: Time, S: Speedup

D	8		16		32	
P	T	S	T	S	T	S
1	154.2	14.8	82.7	27.6	59.35	38.41
2	83.2	27.4	47.2	48.3	27.7	82.3
4	47.7	47.8	29.4	77.6	19.3	118.1
8	30.0	76.0	20.6	110.7	15.1	151.07
16			16.1	141.7	13.0	175.4
32					12.0	190.0

Table 2. Length=2313bps, the base of speedup is the executing time (=20376ms) under xeon 2.4ghz. Time: second. D: Pipeline depth, P: Parallel width, T: Time, S: Speedup

D	8		16		32	
P	T	S	T	S	T	S
1	2626.7	7.8	1325.5	15.4	617.58	32.97
2	1327.7	15.3	676.2	30.1	357.4	57.0
4	678.2	30.0	351.5	58.0	191.9	106.2
8	353.5	57.6	189.2	107.7	109.1	186.8
16			108.0	188.7	67.7	301.0
32					47.0	433.5

tion result we can conclude: under the same pipeline depth, the executing time is decreasing with the increasing parallel width; but the memory access data is increasing and the bubble is increasing. Under the certain pipeline depth and parallel width, first, because the width of data bus is 128bits and the data width is 20bits, when the parallel width is 8, the bus can be utilized fully and decreased the delay of memory access. Then, under the parallel width 8, the pipeline depth can be 8, 16 and 32. From the executing time figure we can conclude when the pipeline depth is 8 we can get a better speedup. Although 32 can get a better speedup than 16, it will consume more space in FPGA and is difficult to realize. Balancing the whole performance and the width of data bus in FPGA, we can decide that the configuration in which pipeline depth is 16 and parallel width is 8 is a better one. In that configuration we can get a better speedup and the data transfer demands are satisfied.

5. Conclusions

For the first time, we perform a comprehensive memory performance analysis for the main applications in bioinformatics: sequence alignment. Based on their different memory performance characteristics and computing behaviors, we proposed specific algorithm optimization methods. Eliminating memory usage accumulation and hiding I/O cost by overlapping computation with I/O, memory performance is improved, furthermore, the running time is reduced. For relatively CPU bound applications, we try exploiting instruction level parallelization and fine-grain parallelization. Based on the emerging reconfigurable computing technology, a fine-grain parallel RNA secondary structure prediction algorithm is mapped to FPGA arrays.

The performance analysis on the cycle-by-cycle simulator indicates that the parallel algorithm achieves huge speedup. The instruction level parallelization and fine-grain parallelization technology improves the arithmetic efficiency for CPU-bound application. The optimization methods in this paper are commonly used in high performance computing, however, to find a suitable optimization method for a specific algorithm is a difficult and complex work because of the lack of the instruction of some basic framework. Through the study of computational behavior of bioinformatics application, especially the memory access behavior because of the memory wall which is argued in recent research works, we demonstrate an approach to high performance algorithm optimization.

The optimization techniques in our work are often used in high performance computing (HPC) applications [23]. In fact, a current important trend of HPC is to developing some efficient methods to improve the utilization of computers. The research work in this paper is our first step to develop high efficient programs in bioinformatics. In our future work, we will focus on abstracting some performance critical kernel programs in bioinformatics and intend to develop a systematic method to optimize these kernels to be adaptive to different architecture. When the performance critical kernels are standardized, it is easy to design a specific hardware accelerator to implement the kernel library.

6. Acknowledgments

This work is supported by knowledge innovative project of Chinese Academy of Sciences under the contact no. KSCX2-SW-233, 863 Grid node of Hongkong University under the contact no. 2002AA104530, National Natural Science Foundation of China Project under Grant 60503060.

References

[1] D. Gusfield. Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. Cambridge University Press, 1999.

[2] R. B. Lyngso, M. Zuker. Fast evaluation of internal loops in RNA secondary structure prediction. Bioinformatics. 1999, Vol. 15(6), pp. 440-445.

[3] N. Camp, H. Cofer, R. Gomperts. High-Throughput BLAST. SGI White Paper, September 1998

[4] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, D. J. Lipman. Basic local alignment search tool. Journal of .Molecular Biology, 1990, Vol. 215, pp. 403-410

[5] F. Galison. The Fasta and Blast programs. 2000, http://bioweb.pasteur.fr/seqanal/blast/

[6] M.S. Waterman, T.F. Smith. Rapid dynamic programming methods for RNA secondary structure. Adv. Appl. Math., 1987, Vol. 7, pp. 455-464

[7] J.D.Thompson, D.G.Higgins and T.J.Gibson CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, positions-specific gap penalties and weight matrix choice. Nucleic Acids Research, 1994, Vol. 22, pp. 4673-4680.

[8] W. Wulf S. McKee. Hitting the memory wall: Implications of the obvious. ACM Computer Architecture News, 1995.

[9] M. B. Taylor. The Raw Microprocessor: A Computational Fabric for Software Circuits and General-Purpose Programs. IEEE Micro, March-April 2002

[10] K.Mai. Smart Memories: A Modular Reconfigurable Architecture. Proceedings of the 27th Annual International Symposium on Computer Architecture, June 2000.

[11] B. Kiran and K. P. Viktor. Reconfigurable computing: Architecture, models and algorithms. Current Science, Vol. 78, No. 7, 10 April 2000.

[12] D.A. Bader, V. Sachdeva, V. Agarwal, G. Goel, A.N. Singh. BioPerf: A Benchmark Suite to Evaluate High-Performance Computer Architecture on Bioinformatics Applications. IEEE International Symposium on Workload Characterization, October 2005.

[13] K. Albayraktaroglu, A. Jaleel, X. Wu, B. Jacob, M. Franklin, C.-W. Tseng, and D. Yeung. BioBench: A Benchmark Suite of Bioinformatics Applications. IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'05), Austin, TX, March 2005.

[14] Y. Li, T. Li, T. Kahveci, J. A. B. Fortes. Workload Characterization of Bioinformatics Applications. IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2005.

[15] http://oprofile.sourceforge.net

[16] http://icl.cs.utk.edu/papi/

[17] http://www.ncic.ac.cn/ hpcog/

[18] http://www.dgate.org/vmips/

[19] http://www.phrap.org/phredphrapconsed.html

[20] http://www.tbi.univie.ac.at/ ivo/RNA/

[21] Z. Zhang, S. Schwartz, L. Wagner, W. Miller. A greedy algorithm for aligning DNA sequence. Jounal of Computational Biology,2000,Vol. 7(12 1/2), pp.203-214

[22] www.ncbi.nlm.nih.gov

[23] J. Demmel, J. Dongarra, V. Eijkhout, E. Fuentes, A. Petitet, R. Vuduc, C. Whaley and K. Yelick. Self adapting linear algebra algorithms and software. In Proceeding of the IEEE, vol. 93, no. 2, pp. 293-312, special issue on Program Generation, Optimization and Adaptation, 2005.