BIOINFORMATICS APPLICATIONS NOTE Vol. 26 no. 10 2010, pages 1368–1369
doi:10.1093/bioinformatics/btq135

Sequence analysis

Advance Access publication March 26, 2010

Multi-threaded vectorized distance matrix computation on the CELL/BE and x86/SSE2 architectures

Adrianto Wirawan*, Chee Keong Kwoh and Bertil Schmidt

School of Computer Engineering, Nanyang Technological University, Singapore

Associate Editor: Limsoon Wong

ABSTRACT

Summary: Multiple sequence alignment is an important tool in bioinformatics. Although efficient heuristic algorithms exist for this problem, the exponential growth of biological data demands an even higher throughput. The recent emergence of multi-core technologies has made it possible to achieve a highly improved execution time for many bioinformatics applications. In this article, we introduce an implementation that accelerates the distance matrix computation on x86 and Cell Broadband Engine, a homogeneous and heterogeneous multi-core system, respectively. By taking advantage of multiple processors as well as Single Instruction Multiple Data vectorization, we were able to achieve speed-ups of two orders of magnitude compared to the publicly available implementation utilized in ClustalW.

Availability and Implementation: Source codes in C are publicly available at https://sourceforge.net/projects/distmatcomp/

Contact: adri0004@ntu.edu.sg

Received on December 11, 2009; revised on March 5, 2010; accepted on March 23, 2010

1 INTRODUCTION

Multiple sequence alignment (MSA) of many nucleotides or amino acids is an important tool in bioinformatics. It can identify patterns or motifs to characterize protein families, and is therefore utilized to detect homology between sequences as well as to perform phylogenetic analysis. Many MSA heuristics have been proposed to reduce the exponential complexity of computing optimal MSAs. Heuristic MSA implementations include MSA (Lipman *et al*., 1989), ClustalW (Thompson *et al*., 1994), T-Coffee (Notredame *et al*., 2000), MUSCLE (Edgar, 2004) and PRALINE (Simossis and Heringa, 2005). ClustalW is considered to be one of the most popular MSA tools. It is based on the progressive alignment method. Although not optimal, this method can produce reasonably good alignments at a good efficiency. However, the exponential growth of biological data demands an even higher throughput. Thus, software approaches to improve the performance of ClustalW have been introduced, including caching (Catalyurek *et al*., 2003) and parallel processing (Chaichoompu *et al*., 2006; Li, 2003). Recent usage of easily accessible accelerator technologies to improve the ClustalW algorithm include FPGAs (Oliver *et al*., 2005) and GPUs (Liu *et al*., 2007; Liu *et al*., 2009).

Profiling of ClustalW reveals that the distance matrix computation is the most time-consuming phase and takes typically >90% of the overall runtime. Therefore, accelerating this phase would greatly improve the performance as a whole.

Implementations through the use of Single Instruction Multiple Data (SIMD) parallelism can achieve increased performances by performing operations on multiple values in parallel. The recent emergence of multi-core technologies, either homogeneous or heterogeneous, makes it possible to achieve an improvement in execution time for many bioinformatics applications (Szalkowski *et al*., 2008; Wirawan *et al*., 2008).

In this article, we introduce our implementation that accelerates the distance matrix computation on the Cell Broadband Engine (Cell/BE) and the commonly used Intel x86 architecture. By taking advantage of multiple processors as well as SIMD vectorization, we were able to achieve significant speed-ups of two orders of magnitude compared to the distance matrix computation used in ClustalW.

2 METHODS

Our Cell/BE implementation takes advantage of the 128-bit SIMD vector registers of each synergistic processing elements (SPE) and uses half-word values (16 bits) for the computation, which is the smallest element supported by the Cell/BE instruction set. This allows eight cells to be processed per vector register. SPU intrinsics (IBM, 2007) are used to improve the efficiency of the program. The SSE2 instructions support 8- and 16-bit elements in the vector registers. Our implementation utilizes 16-bit elements, allowing eight cells to be processed per vector register. Since all elements in the same minor diagonal of the dynamic programming (DP) matrix can be computed independent of each other in parallel, the computation is done in minor diagonal order. In our implementation, distance values $d(S_i, S_j)$ are computed without computation of the actual traceback using DP.

Since all elements in the same minor diagonal of the DP matrix can be computed independent of each other in parallel, the computation is done in minor diagonal order. Given are sequences S_i and S_j of lengths l_1 and l_2, respectively and vector registers vH, vE, vF, vN_A, vN_E and vN_F containing the values H_A, E, F, N_A, N_E and N_F, respectively. For each iteration $c\left[1 \le c \le \left(l_1+l_2-1\right)\right]$, the values of $H_A(i,j)$, $E(i,j)$, $F(i,j)$, $N_A(i,j)$, $N_E(i,j)$ and $N_F(i,j)$ are computed for all $1 \le i \le l_1$ and $1 \le i \le l_2$. Vector masks are computed during the calculations of the vE, vF and vH vectors, which are then used to determine the value of the corresponding vN_E, vN_F and vN_A vectors, respectively. The *nid* score is extracted as $N_A(i_{\max}, j_{\max})$, where $(i_{\max}, j_{\max})$ denotes the coordinates of the maximum value in the corresponding matrix H_A. The algorithm used in our implementations is described in more detail in Wirawan *et al*. (2009).

To speed-up the computation, a query profile is pre-computed. The query profile is indexed by the query sequence position and the database sequence symbol and is stored in a column-based manner. Therefore, random accesses to the substitution matrix due to table look up are replaced with sequential ones to the query. For the Cell/BE implementation, the query

*To whom correspondence should be addressed.

profile computation is done in the power processor element (PPE) and is distributed to the respective SPEs using direct memory access (DMA) transfer. For the SSE2 implementation, each thread contains its respective query profile information needed to complete the computation.

Our Cell/BE implementation utilizes the additional instructions of the PPE relating to control of the SPEs to implement the multi-threading. Unlike SPEs, the PPE can read and write the main memory and the local memories of SPEs through the standard load/store instructions. The PPE reads the input dataset, pre-processes it and divides the dataset into equal size blocks for each SPEs to process. Since the blocks are independent of each other, no thread synchronization is necessary during the calculations. The mailbox functions *spe_in_mbox_write* and *spu_read_in_mbox* are used to ensure that all the SPEs obtain their respective contexts in their local memory. Using the context data, each SPE then transfers any required information and necessary sequences. To improve transfer efficiency, the database sequences in main memory and in the local storage are aligned within the cache line and data structures are initialized during the transfer of the sequence. Our SSE2 implementation uses *pthreads* to implement the multi-threading. The input dataset are pre-processed and sorted according to length. Each thread contains a copy of the database sequence, query sequence and its respective query profile. Since the datasets are sorted, the datasets are divided into roughly equal size workload for each thread to process.

Cell/BE does not support saturation arithmetic that are needed in the calculations to anticipate overflow problems. Hence, we utilized several SPU intrinsics, i.e. *spu_sel*, *spu_splats*, *spu_rlmaska*, *spu_nor* and *spu_and* in conjunction with the existing *spu_add* and *spu_sub* to handle saturated additions and saturated subtractions, respectively. Unlike Cell/BE, Intel's SSE2 instructions support saturation arithmetic. Hence, saturated subtraction and addition functions, *_mm_subs_epu16* and *_mm_adds_epu16*, respectively, are utilized to ensure that the values of the vector are within valid range.

3 RESULTS

A set of performance evaluation experiments have been conducted using six protein sequence datasets consisting of sequences selected from the human immunodeficiency virus (HIV) dataset downloaded from NCBI. The experimental datasets represent datasets of small number of long sequences, medium number of medium-length sequences and large number of short sequences, respectively.

Our Cell/BE implementation is benchmarked on a stand-alone PlayStation®3 (PS®3) with Fedora Core 9.0 and the Cell Software Development Kit (SDK) 3.1. Our SSE2 implementation is benchmarked on an Intel Quad-Core i7-920 2.66GHz CPU, 12 GB RAM running Linux Fedora 10. The sequential ClustalW application, available online at http://www.bii.a-star.edu.sg/achievements/applications/clustalw/, was benchmarked on an Intel Quad-Core i7-920 2.66GHz CPU, 12 GB RAM running Linux Fedora 10.

Table 1 shows the performance evaluation of our implementations using the above-mentioned datasets on different architectures. The Cell/BE implementation shows a better performance for datasets with fewer but longer sequences, while the SSE2 implementation shows a better performance for datasets with more but shorter sequences. This is due to the overhead for the PS3®, which involved DMA transfers of required data and sequences between the PPE and the SPEs.

Since the first stage of several other MSA tools, e.g. T-Coffee (Notredame *et al*., 2000) and MUSCLE (Edgar, 2004) is also based on the computation of pairwise distances, these tools will also benefit from the speed-up from the accelerator presented in this article.

Table 1. Comparison of run-times (in seconds) of our Cell/BE implementation running on a PS®3, our SSE2 implementation on Intel Quad-Core i7-920 2.66GHz CPU and distance matrix computation of ClustalW running on Intel Quad-Core i7-920 2.66GHz CPU using six datasets of variable number of sequences and average length

Number of sequences	Average length	SSE2 with eight threads	Cell/BE on the PS3®	Distance matrix computation of ClustalW on Intel Quad-Core i7-920
400	856	21.41	16.79	1566
1000	858	127.91	101.21	9747
2000	266	58.95	56.83	2099
4000	247	200.63	173.26	9280
4000	83	35.19	39.04	1097
8000	73	101.12	125.14	4496

Conflict of Interest: none declared.

REFERENCES

Catalyurek,U. *et al*. (2003) A component-based implementation of multiple sequence alignment. In *Proceedings of the 2003 ACM Symposium on Applied Computing*, ACM, NY, New York, pp. 122–126.

Chaichoompu,K. *et al*. (2006) MT-ClustalW: multithreading multiple sequence alignment. In the *20th International Parallel and Distributed Processing Symposium, IPDPS 2006*. Availabvle at http://www.scopus.com/inward/record.url?eid=2-s2.0-33847126624&partnerID=40&md5=4132316f74c84188034bbf1ae38c80d1

Edgar,R.C. (2004) MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Res.*, **32**, 1792–1797.

IBM (2007) Software Development Kit 2.1 Accelerated Library Framework Programmer's Guide and API Reference, Version 1.1. *IBM developerWorks.*

Li,K.-B. (2003) ClustalW-MPI: ClustalW analysis using distributed and parallel computing. *Bioinformatics*, **19**, 1585–1586.

Lipman,D.J. *et al*. (1989) A tool for multiple sequence alignment. *Proc. Natl Acad. Sci. USA*, **86**, 4412–4415.

Liu,W. *et al*. (2007) Streaming algorithms for biological sequence alignment on GPUs. *IEEE Trans. Parallel Distributed Syst.*, **18**, 1270–1281.

Liu,Y. *et al*. (2009) MSA-CUDA: multiple sequence alignment on graphics processing units with CUDA. In *Proceeding of the 20th IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP 2009)*. IEEE Press, Boston, MA, pp. 121–128.

Notredame, C. *et al*. (2000) T-coffee: a novel method for fast and accurate multiple sequence alignment. *J Mol. Biol.*, **302**, 205–217.

Oliver,T. *et al*. (2005) Multiple sequence alignment on an FPGA. In *11th International Conference on Parallel and Distributed Systems (ICPADS 2005)*. IEEE Computer Society Press, Fukuoka, Japan, pp. 326–330.

Simossis,V.A. and Heringa,J. (2005) PRALINE: a multiple sequence alignment toolbox that integrates homology-extended and secondary structure information. *Nucleic Acids Res.*, **33**, W289–W294.

Szalkowski,A. *et al*. (2008) SWPS3 - fast multi-threaded vectorized Smith-Waterman for IBM Cell/B.E. and x86/SSE2. *BMC Res. Notes*, **1**, 107.

Thompson,J.D. *et al*. (1994) CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res.*, **22**, 4673–4680.

Wirawan,A. *et al*. (2008) CBESW: sequence alignment on the playstation 3. *BMC Bioinformatics*, **9**, 377.

Wirawan,A. *et al*. (2009) Pairwise distance matrix computation for multiple sequence alignment on the cell broadband engine. *Lect. Notes Comput. Sci.*, **5544**, 954–963.