Sequence Analysis

Multi Threaded Vectorized Distance Matrix Computation on the Cell/BE and x86/SSE2 Architectures

Adrianto Wirawan1,* , Kwoh Chee Keong1 and Bertil Schmidt1
1School of Computer Engineering, Nanyang Technological University, Singapore

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 paper, we introduce an implementation that accelerates the distance matrix computation on x86 and Cell Broadband Engine (BE), a homogeneous and heterogeneous multi-core system, respectively. By taking advantage of multiple processors as well as SIMD vectorization, we are able to achieve speedups 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

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 more than 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., 2009).

In this paper, 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 are able to achieve significant speedups 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 SPEs 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 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 dynamic programming (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 \(l_i\) and \(l_j\), respectively and vector registers \(vH, vE, vF, vN_S, vN_L\) and \(vN_T\) containing the values \(H_{i,j}, E_{i,j}, F_{i,j}\) and \(N_{S(i,j)}, N_{L(i,j)}\) and \(N_{T(i,j)}\) are computed for all \(1 \leq i \leq l_i\) and \(1 \leq j \leq l_j\). 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 \(vN_S, vN_L\) and \(vN_T\) vectors, respectively. The \(mid\) score is extracted as \(N_{l_{max}}(i,j)\), where \(l_{max}\) denotes the coordinates of the maximum value in the corresponding matrix \(H_{l}l\). The algorithm used in our implementations is described in more detail in (Wirawan, et al., 2009).

*To whom correspondence should be addressed.
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 lookup are replaced with sequential ones to the query. For the Cell/BE implementation, the query profile computation is done in the PPE and is distributed to the respective SPEs using 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 spu_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 mailbox functions pthreads are utilized 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 dataset are sorted, the dataset are divided into roughly equal size workload for each thread to process.

Cell/BE does not support saturation arithmetic which are needed in the calculations to anticipate overflow problems. Hence, we utilized several spu intrinsics, i.e. spu sel, spu splats, spu rmaska, 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 intrinsics support saturation arithmetic. Hence, saturated subtractions and additions 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 has 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 standalone 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, 12GB RAM running Linux Fedora 10. The sequential ClustalW application, available online at http://www.bia-star.edu.sg/achievements/applications/clustalw/, was benchmarked on an Intel Quad-Core i7-920 2.66GHz CPU, 12GB RAM running Linux Fedora 10.

<table>
<thead>
<tr>
<th>Number of Sequences</th>
<th>Average Length</th>
<th>SSE2 with 8 threads</th>
<th>Cell/BE on the PS3®</th>
<th>Distance Matrix Computation of ClustalW on Intel Quad-Core i7-920</th>
</tr>
</thead>
<tbody>
<tr>
<td>400</td>
<td>856</td>
<td>21.41</td>
<td>16.79</td>
<td>1566</td>
</tr>
<tr>
<td>1000</td>
<td>858</td>
<td>127.91</td>
<td>101.21</td>
<td>9747</td>
</tr>
<tr>
<td>2000</td>
<td>266</td>
<td>58.95</td>
<td>56.83</td>
<td>2099</td>
</tr>
<tr>
<td>4000</td>
<td>247</td>
<td>200.63</td>
<td>173.26</td>
<td>9280</td>
</tr>
<tr>
<td>8000</td>
<td>83</td>
<td>35.19</td>
<td>39.04</td>
<td>1097</td>
</tr>
<tr>
<td>8000</td>
<td>73</td>
<td>101.12</td>
<td>125.14</td>
<td>4496</td>
</tr>
</tbody>
</table>

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 speedup from the accelerator presented in this paper.

Conflict of Interest: none declared.

REFERENCES


