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

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 (Chaichiyoomp et al., 2006; Li, 2003). Recent usage of easily accessible accelerator technologies to improve the computation of ClustalW 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 element (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 independently of each other in parallel, the computation is done in minor diagonal order. In our implementation, distance values d(i, 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 S1 and S2 of lengths l1 and l2 respectively and vector registers vH, vE, vF, vN, vNE and vNF containing the values HA, E, F, NE and NF, respectively. For each iteration \(1 \leq s \leq (l_1 + l_2 - 1)\), the values of \(H(s, j), E(s, j), F(s, j), NE(s, i), NF(s, i)\) and \(vH(s, i), vE(s, i), vF(s, i), vN(s, i), vNE(s, i), vNF(s, i)\) are computed for all \(1 \leq s \leq i \leq l_1\) and \(1 \leq s \leq j \leq l_2\). 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 vNE, vNF and vNF vectors, respectively. The nd score is extracted as \(ND_{(\text{max}, \text{max})}\), where \(\text{max}_{(\text{max}, \text{max})}\) denotes the coordinates of the maximum value in the corresponding matrix \(H_{\text{max}}\). 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...
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 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 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_rmask, spu_or not 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_sub_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 PlayStation3® (PS®3) with Fedora Core 9.0 and the Cell Softwom 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 implementation is 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 PS®3, which involved DMA transfers of required data and sequences between the PPE and the SPE.

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.

Conflict of Interest: none declared.

REFERENCES


<table>
<thead>
<tr>
<th>Number of sequences</th>
<th>Average length</th>
<th>SSE2 with eight threads</th>
<th>Cell/BE on the PS®3</th>
<th>Distance matrix computation of Clustaw 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>1588</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>12000</td>
<td>73</td>
<td>101.12</td>
<td>125.14</td>
<td>4496</td>
</tr>
</tbody>
</table>