Parallelization Strategies for Smith Waterman Algorithm on CellBE

Friman Sánchez∗, Alex Ramirez†, Mateo Valero†

∗ Universitat Politècnica de Catalunya, Barcelona, Spain
† Barcelona Supercomputing Center, Barcelona, Spain

ABSTRACT

The fast growth of bioinformatics field has attracted the attention of computer scientists in the last few years. At the same time the increasing sizes of biology databases require greater efforts to improve the computational performance. From a computer architecture point of view, we are interested in investigating how bioinformatics applications can benefit from current and future multiprocessors architectures. In this poster we present an analysis of several alternatives to implement the Smith-Waterman (SW) algorithm on the Cell BE architecture. We explore different level of parallelism which are available in the algorithm: fine-grained parallelism using SIMD approach and coarse-grained parallelism using blocking and threads working in parallel.

1 Introduction

The SW \cite{SW81} is a dynamic programming algorithm for computing the optimal local-alignment score, which takes two sequences of any length, and at any location determines an optimal alignment between them. To quantify this process, a substitution score matrix is used to indicates the score associated with matching one amino acid with another. Given a query sequence \( A \) of length \( m \), a database sequence \( B \) of length \( n \), a substitution score matrix \( Z \), a gap-open penalty \( q \) and a gap extension penalty \( r \), the optimal local alignment score \( T \) is computed by the following recursion relations:

\[
e_{i,j} = \max \{ e_{i,j-1}, h_{i-1,j} - q \} - r
\]

\[
f_{i,j} = \max \{ f_{i-1,j}, h_{i,j-1} - q \} - r
\]

\[
h_{i,j} = \max \{ h_{i-1,j-1} + Z[A[i], B[i]], e_{i,j}, f_{i,j}, 0 \}
\]

\[
T = \max \{ h_{i,j} \}
\]

Where the overall-maximum local-alignment score involving the first \( i \) symbols of \( A \) and the first \( j \) symbols of \( B \), is represented by \( h\{i,h\} \). The recursion has data dependencies as shown in figure \[\text{II}\] where computation of cells across the anti-diagonals are independent. The final score is reached when all the symbols have been compared.

\[1\] E-mail: fsanchez@ac.upc.edu - \{alex.ramirez,mateo.valero\}@bsc.es

\[2\] This work has been sponsored by the European Commission in the context of the SARC project (contract no. 27648), the Hipeac Network of Excellence and the Spanish Ministry of Education (contract no. TIN2007-60625)
2 Parallelization Strategies on Cell BE

In the SW algorithm, coarse-grain parallelism exists when comparing a query sequence against all the sequences in a database of hundreds of thousands of sequences. Fine-grain parallelism exists when comparing the query against each database sequence. The length of both the query and database sequence is an important issue. Depending on the length, the sequences (and other temporal computations) can fit or not into the Local Store (LS) of the SPU (Only 256KB for code and data). If they fit into the LS, the execution will not make use of the PPU cache to store temporal results. Otherwise, the SPU will have to use buffers, that periodically have to be stored and loaded to and from PPU cache. This is done via DMA transfers which degrade performance and increase the bus utilization.

Figure 1: Available parallelism in the SW execution

2.1 Multiple short comparisons in parallel

Coarse-grain parallelism can be exploited by performing several comparison between sequences in parallel, and fine-grain parallelism is exploited by computing temporal vector of scores of cells parallel to the anti-diagonals, as figure 2 shows. The process starts at the upper left moving from left to right and from top to bottom. Every time a vector anti-diagonal is computed, some temporal results have to be stored because they will be used in the computation of a vector in the next row. This approach is shown in figure 2a, where each SPU is responsible for the comparison between the query and a group of database sequences. Because of its short length, the sequences and temporal computations of the matrix can be kept in the LS of each SPU, which helps to reduce significantly the amount of DMAs operations from LS to main memory and viceversa.

Figure 2: (a) One SPU processes groups of sequences, (b) Several SPUs process each sequence
Table 1: Phases for execution of the SW algorithms on the SPUs

<table>
<thead>
<tr>
<th>Phase Name</th>
<th>Description</th>
<th>Present in the comparison of</th>
</tr>
</thead>
<tbody>
<tr>
<td>wait-for-new-params</td>
<td>SPUs wait for parameters of a new comparison.</td>
<td>short and long sequences</td>
</tr>
<tr>
<td>DMA-get-qpw</td>
<td>SPUs perform DMA operations to get required precomputed profile matrix</td>
<td>short and long sequences</td>
</tr>
<tr>
<td>DMA-put-get-temp-data</td>
<td>SPUs perform DMA put/get operations to store/load temporal data</td>
<td>long sequences</td>
</tr>
<tr>
<td>compute-score</td>
<td>SPUs compute the score of the comparison.</td>
<td>short and long sequences</td>
</tr>
</tbody>
</table>

2.2 Single Comparison of Long Sequences Using one SPU

When sequences are very long, the processing requires that temporal computations (the border between rows in figure 1) have to be stored back in memory instead of using the LS. This is because the amount of data (that depends on the sequence sizes) could not always fit entirely into the LS. As a result, not only the traffic gets increased between memory and the LS, but also the processing of data has to wait for the DMA transfers completion. However, the impact of this limitation can be diminished by using multi-buffering to overlap the SPU processing with DMA transfers of the next data. The important decision here is the choice of appropriate block sizes of data to be computed and transferred using DMA.

2.3 Single Comparison of Long Sequences Using Several SPUs

Another parallel alternative is shown in figure 2b. In this case, several SPUs perform the comparison between the query against a single database sequence. The computation of the matrix is distributed between the SPUs as follows: when 8 SPUs are available, SPU0 computes row 0, row 8, row 16, etc; SPU1 computes row 1, row 9, row 17, and so on. Here, not all the temporal results are written to main memory (the SPUs exchange data in a streaming fashion). Compared with the alternative described in section 2.1, this approach seems to be more scalable when sequence are longer because each SPU is holding smaller pieces of data in the LS. However, additional synchronization between SPUs is required that can potentially degrade the performance.

3 Experimental Methodology and Performance Analysis

We ported the SW algorithm of Ssearch application to the CellBE ISA extension. We use short sequences (length size < 7000 symbols) for evaluating the alternative described in section 2.1. Also, we use a group of longer sequences (length size > 7000) to test the parallel alternatives of sections 2.2 and 2.3. The executions were done on a IBM BladeCenter Cell BE processor. We use PARAVER tool to analyze the execution. This allows us to identify the different phases that the algorithm presents. We summarize the phases of the execution in table 1.

3.1 Phase Analysis of the SW Algorithm

We analyze the phases of the execution using from 1 to 16 SPUs. We quantify how the execution time distributes between the phases. As an example, figure 3 shows the phases when using 2 SPUs to compare long sequences. Here, SPU1 starts waiting for parameters of a new comparison. Then, it performs some DMAAs to get (periodically) some precomputed data used in process (dma-get-qpw). Then, the real computation starts. Periodically, some
temporal data should be stored back to main memory as it was explained before. Figure 4 summarizes, the percentage of time each phase requires for the alternatives of sections 2.1 and 2.2 when using different amount of SPUs. As an example, the bar 8 in the left side, shows that 18% of the time each SPU waits for data to start a new comparison. while around 8% of time, each SPU waits for DMA operations finish. The rest of the time, they do useful work. The results show that PPU is not as fast as it should be to keep all the SPUs busy. The right side of the Figure 4 shows the same data when longer sequences are compared. Here, most of the time is used in computation and the wait for parameters does not take much time: The SPUs do not require data very quickly because they have to perform a lot of work before asking for new parameters to the PPU.

Figure 3: PARAVER Trace of the execution

Figure 4: Time distribution of Phases

4 Conclusions

The SW algorithm and most of the bioinformatics applications are computationally demanding. They combine different levels of parallelism, becoming a very challenging workload that can benefit from the current and future chip multiprocessors architectures. However, it is important to understand which are the challenges these applications propose, in order to define what kind of architectures they require.

References