Cuarto Congreso Colombiano de Computación 4CCC
Sociedad Colombiana de Computación S(Co)2
Universidad Autónoma de Bucaramanga UNAB
© 2009 Los Autores. Todos los Derechos Reservados.

Exploiting Different Levels of Parallelism in the Biological Sequence Comparison Problem

Friman Sánchez Castaño
Technical University of Catalonia, Computer Architecture Department.
Barcelona, Spain
fsanchez@ac.upc.edu

Alex Ramirez
Technical University of Catalonia and Barcelona Supercomputing Center-CNS
Barcelona, Spain
alex.ramirez@bsc.es

Mateo Valero Cortés
Technical University of Catalonia and Barcelona Supercomputing Center-CNS
Barcelona, Spain
mateo.valero@bsc.es

ABSTRACT

In the last years the fast growth of bioinformatics field has atracted the attention of computer scientists. At the same time, de exponential growth of databases that contains biological information (such as protein and DNA data) demands great efforts to improve the performance of computational platforms. In this work, we investigate how bioinformatics applications benefit from parallel architectures that combine different alternatives to exploit coarse- and fine-grain parallelism. As a case of analysis, we study the performance behavior of the Ssearch application that implements the Smith-Waterman algorithm (SW), which is a dynamic programing approach that explores the similarity between a pair of sequences. The inherent large parallelism of the application makes it ideal for architectures supporting multiple dimensions of parallelism (thread-level parallelism, TLP; data-level parallelism, DLP; instruction-level parallelism, ILP). We study how this algorithm can take advantage of different parallel machines like the SGI Altix, IBM Power6, IBM Cell BE and MareNostrum machines. Our study includes a qualitative analysis of the parallelization opportunities and also the quantification of the performance in terms of speedup and execution time. These measures are collected taking into account the specific characteristics of each architecture. As an example, our results show that a share memory multiprocessor architecture (SMP) like the PowerPC 970MP of Marenostrum machine can surpasses a heterogeneous multiprocessor machine like the current IBM Cell BE.

Categories and Subject Descriptors

C.1.2 [**Processor Architectures**]: Multiprocessors; D.2.8 [**Metrics**]: [complexity measures, performance measures]; J.3 [**Life and Medical Sciences**]: Biology and genetics.

General Terms

Algorithms,Measurement,Performance

Keywords

Parallel Architectures, Multiprocessor Architectures, Bioinformatics Applications, Sequence Comparison.

1. INTRODUCTION

Bioinformatics is a very multidisciplinary field including components of mathematics, biology, chemistry, computer sciences, software engineering, processor architecture, hardware design, etc. Currently, bioinformatics is considered as one of the fields of computing technology with fastest growth and development [5]. Additionaly, the vast amount of biological data that has become available since the early 1990s has made necessary to create specialized databases to store, organize and index data [7] [4] [2] [1], and also has led the advance in research and development of specialized tools to view and analyse this biological information. Because of that, computational biology and its related components in database systems, visualization and analysis tools has become an emerging workload that requires high-performance computing systems. This emerging importance is also reflected in research and development of several algorithmic methodologies to process biology data efficiently. However, due to the immense quantity of information, performing even a simple analysis on genome-scale data quickly turns into a computationally difficult and time consuming problem.

Bioinformatics allow researchers to process the massive biological data (e.g., nucleic acid and protein sequences, structures, functions, pathways and interactions) and identify information of interest. Bioinformatics is a very diverse field that that include different kind of tasks performed on biological data. Some of these tasks are: alignment and comparison of biology sequences, phylogenetic analysis, multiple sequence alignment, sequences profiling searching, genome-level alignment, sequence assembly, protein structure prediction, protein docking and so on. However, regarding the variety of existing tasks, the most commun is the comparison and alignment of biological sequences (DNA, proteins, RNA), which is basically the problem of finding an approximate pattern matching between two or more sequences. Molecular biologists usually compare sequences to find similarities between them in order to define whether one sequence is similar to another. Generally, such comparisons involve aligning sections of the two sequences in a way that exposes the similarities between them. For example, consider the sequences `A=csttpggg` with eight residues (sym-

bols) and `B=csdtnglawgg` with eleven residues. One possible alignment could be:

```
A = c s - t t p g - - - g g
    | |   |   | |
B = c s d t - n g l a w g g
```

In the above alignment, we say that $b_3 = d$ is inserted into the first sequence or it is deleted from the second one, depending on the point of view. Also, $a_5 = p$ is substituted by $b_5 = n$ or b_5 by a_5. Consecutive dashes in the sequences represent a gap, for example, there is a gap of length one between a_2 and a_3 and a gap of length three between a_6 and a_7 in the first sequence. In order to compare sequences a similarity score between the individual residues must be established. Biologists have defined *substitution scoring functions* that assign a similarity score between all possible pairs of residues. In the case of proteins, these functions quantify whether the substitution of one residue for another is likely to conserve the chemical and physical properties of the protein or is more likely to disrupt essencial structural and functional features of the protein. For these reasons, the definition and evaluation of these score functions is an important scientific discussion in the biology community. Numerous approaches have been used to create such quantifications, referred to as *substitution score matrix*. Some of these matrices are PAM, PAM100, BLOSUM45, BLOSUM62, etc [10] [14] [13]. An interesting discussion about the characteristics of these matrices can be found in [17]. As an example, figure 1 shows the BLOSUM62 amino acid subsitution score matrix.

Researchers have proposed several methods for alignment of two biological sequences. However, the dynamic programming (DP) is probably the most important programming method in sequences alignment [30] and many researchers have proposed different algorithms based on DP to quantify the similarity of a pair of sequences [23] [31] [32] [12]. Between these alternatives, the most important sequence-search algorithm is that of Smith-Waterman (SW) [32], which is generally considered to be the most sensitive. Sensitivity is a measure of how well a method can detect the actual similarity between two sequences. Nevertheless, the computationally intensive task involved in the algorithm is a very restrictive factor that prevents the use of these proposals. The time complexity of this algorithm is $O(mn)$, where m and n are the lengths of the two sequences respectively.

On the other hand, many heuristic strategies have been proposed and developed to speedup the execution of the search and alignment tasks, such as FASTA [25], BLAST [6] and ClustalW [15]. These methods reduce the running time by several factors compared with the SW algorithm. However, this reduction is obtained at the expense of sensitivity, and due to this loss of sensitivity some related sequences can not be detected in a search.

For these reasons, in order to perform both sensitive and fast searches in biological databases, researchers have studied and proposed several strategies to produce a fast implementation of the SW algorithm under different computational platforms such as special purpose hardware, specific

	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W
C	9																			
S	-1	4																		
T	-1	1	5																	
P	-3	-1	-1	7																
A	0	1	0	-1	4															
G	-3	0	-2	-2	0	6														
N	-3	1	0	-2	-2	0	6													
D	-3	0	-1	-1	-2	-1	1	6												
E	-4	0	-1	-1	-1	-2	0	2	5											
Q	-3	0	-1	-1	-1	-2	0	0	2	5										
H	-3	-1	-2	-2	-2	-2	1	-1	0	0	8									
R	-3	-1	-1	-2	-1	-2	0	-2	0	1	0	5								
K	-3	0	-1	-1	-1	-2	0	-1	1	1	-1	2	5							
M	-1	-1	-1	-2	-1	-3	-2	-3	-2	0	-2	-1	-1	5						
I	-1	-2	-1	-3	-1	-4	-3	-3	-3	-3	-3	-3	-3	1	4					
L	-1	-2	-1	-3	-1	-4	-3	-4	-3	-2	-3	-2	-2	2	2	4				
V	-1	-2	0	-2	0	-3	-3	-3	-2	-2	-3	-3	-2	1	3	1	4			
F	-2	-2	-2	-4	-2	-3	-3	-3	-3	-3	-1	-3	-3	0	0	0	-1	6		
Y	-2	-2	-2	-3	-2	-3	-2	-3	-2	-1	2	-2	-2	-1	-1	-1	-1	3	7	
W	-2	-3	-2	-4	-3	-2	-4	-4	-3	-2	-2	-3	-3	-1	-3	-2	-3	1	2	11

Figure 1: Score associated with the replacement of one amino acid with another in the Blosum62 Substitution score matrix

purpose coprocessors, general purpose multiprocessors and general purpose uniprocessors.

This paper anaylises the implementation of the SW algorithm on diverse parallel architectures that combine multiple dimensions of parallelism (TLP, DLP and ILP). We study how the SW algorithm can take advantage of different parallel machines like the SGI Altix, IBM Power6, IBM Cell BE and MareNostrum machines. Our study includes a qualitative analysis of the parallelization opportunities and also the quantification of the performance in terms of speedup.

This paper is organized as follows: Section 2 details some aspects of the Smith-Waterman algorithm as a solution for the sequence comparison problem. Section 3 presents a brief description of the multiprocessor architectures we use in this work. Next, section 4 analyses several parallel alternatives of implementing the SW algorithm on the described architectures. Then, section 5 describes the experimental methodology used in this work. Sections 6 presents the experimental results and the performance analysis. Section 7 discusses the related work and finally, section 8 summarizes the paper and describes some future work directions.

2. THE SMITH-WATERMAN ALGORITHM AS A SOLUTION FOR SEQUENCE COMPARISON PROBLEM

The SW is a sequence comparison algorithm developed by T.F. Smith and M.S. Waterman [32]. It implements dynamic programming techniques that take two sequences of any length, and at any location determine an optimal alignment between the two sequences. In the process, scores or weights are assigned to each character-to-character comparison: positive for exact matches/substitutions, negative for insertions/deletions. In weight matrices, scores are added together and the highest scoring alignment is reported. A short description of the algorithm follows.

2.1 The Smith-Waterman Algorithm

Given a query sequence A of length m, a query sequence B of length n, a substitution score matrix S, a gap-open penalty q and a gap extension penalty r, the optimal local alignment score T can be computed using the following

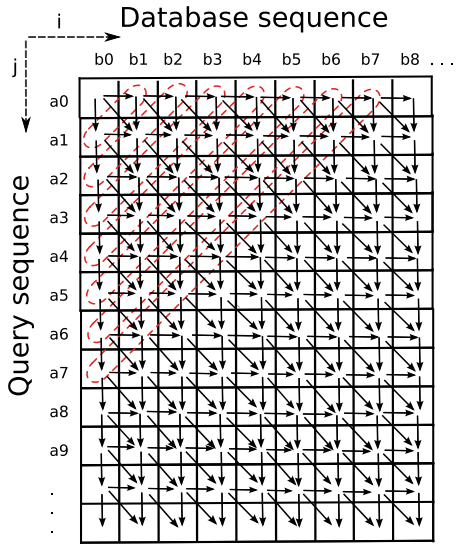

Figure 2: Data-dependency graph in the execution of Smith-Waterman algorithm.

recursive equations:

$$e_{i,j} = max\{e_{i,j-1}, h_{i-1,j} - q\} - r \quad (1)$$
$$f_{i,j} = max\{f_{i-1,j}, h_{i,j-1} - q\} - r \quad (2)$$
$$h_{i,j} = max\{h_{i-1,j-1} + S[A[i], B[i]], f_{i,j}, e_{i,j}, 0\} \quad (3)$$
$$T = max\{h_{i,j}\} \quad (4)$$

Where, $e_{i,j}$ and $f_{i,j}$ are the maximum local-alignment score involving the first i symbols of A and the first j symbols of B, and ending with a gap in sequence B or A, respectively. And $h_{i,j}$ represents the overall-maximum local-alignment score involving the first i symbols of A and the first j symbols of B. The recursion should be calculated with i going from 1 to m and j from 1 to n, with the initial conditions $e_{i,j} = f_{i,j} = h_{i,j} = 0$ for all $i = 0$ or $j = 0$. The order of the computation of the values in the alignment matrix $h_{i,j}$ is strict, that is, the value of any cell cannot be computed before the value of all cells to the left and above it has been computed. Figure 2 shows the data dependencies in the calculation of $h_{i,j}$. From the figure, it can be observed that the computation of $h_{i,j}$ is independent across the antidiagonals. This characteristic has been observed by several researchers and based on it, they have proposed several special purpose architectures that exploit this parallelism [9].

As we will see in this work, an efficient way of taking advantage of the existing parallelism of the algorithm, basically depends on the two following issues: First, the targeted computational platform (the machine) where the algorithm is executed. Second, the efficient implementation of the algorithm on the selected computing platform. As a starting point of our study, we use an efficient implementation of the SW algorithm which is available in the Ssearch [24] application. We modify the application in order to obtain the different versions of the SW algorithm that are used in this work.

An important characteristic of the sequence comparison is the intrinsic embarrasing parallelism of the SW algorithm, as shown in figure 2. However, this is not the only source of parallelism of the sequence comparison problem. Another important factor is the scenario of execution in which it is required to perform sequence comparison. Usually, biologists require to perform the comparison of some *query sequences* against several *databases* that store hundreds of thousands of sequences. Before doing a detailed description of these scenarios, we introduce some discussion about the typical databases used by biologist.

2.2 DNA and Protein Sequence Databases

When the nucleotide sequence of a gene or an entire genome has been determined it is often deposited in larges public sequence databases. The three most important public nucleotide databases are GenBank [7], EMBL [2] and DDBJ [1]. GenBank is handled by the National Center for Biotechnology Information (NCBI) in the United States. The EMBL is handled by the European Bioinformatics Institute (EBI) that makes part of the European Molecular Biology Laboratories (EMBL). The DDBJ database is handled by the DNA Data Bank of Japan. These databases are organized in a specific way and they can be accessed and downloaded freely from the webpages of the institutes.

Similarly, potential protein coding regions in the DNA sequences are also traslated and collected in protein databases which also include proteins identified by other methods. SwissProt [4] is a public protein database handled by the Swiss Institute of Bioinformatics and the EBI. As an example of the size, the last release (release 56.6) of this database of 16-Dec-08 contains 405506 sequences, comprising 146166984 amino acids (residues).

2.3 Scenarios of Execution and Levels of Parallelism of the SW algorithm

Usually, biologists require to perform the comparison of some *query sequences* against several *databases* of sequences. To perform the sequence comparison in this scenario, the Smith-Waterman algorithm of the Ssearch application works as figure 3 shows. Here, the parallelism exists at different levels. First, a coarse-grain parallelism exists because queries can be compared independently against each database. Even for each database it is posible to compare concurrently the query against the sequences held in the database. This coarse-grain parallelism can be exploited by using threaded architectures, massively parallel manchines, specific parallel hardware or multicore architectures.

On the other hand, fine-grain parallelism can be exploited in each sequence-to-sequence comparison using the SW algorithm. Here, the basic goal is the computation of a score matrix (part 2 of figure 3). The computation of this matrix contains a data dependency in an antidiagonal fashion as was explained in section 2.1. This parallelism can be exploited by using machines with Single Instruction Multiple Data units (SIMD units), systolic arrays, specific hardware or even multicore architectures with SIMD capabilites like the IBM Cell processor.

The pseudocode shown in listing 1 is a very brief description of the way the sequence comparison is performed in the mentioned scenario by the Ssearch application, that is,

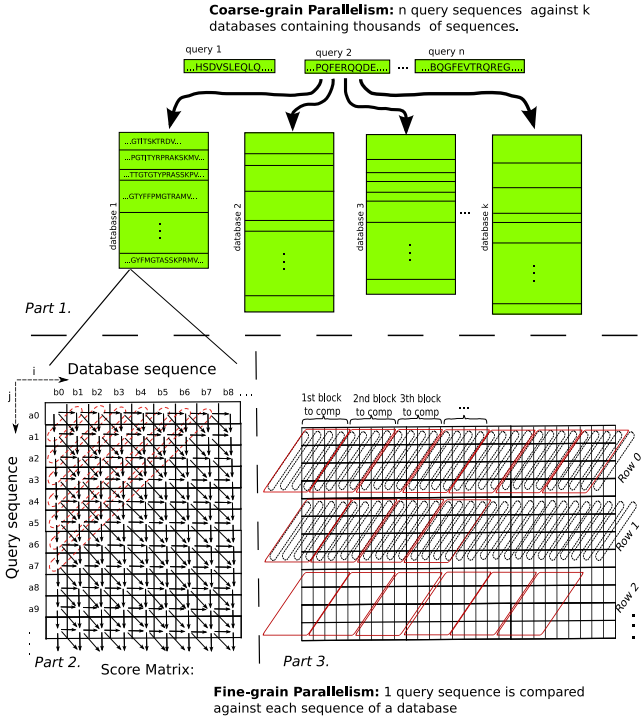

Figure 3: Scenarios of SW execution on the Ssearch application and type of parallelism

the comparison between the query sequences and the different databases. Here, we can distinguish three different `for` loops; the first one takes each query of the total `number_of_query_seqs`, extracts some information of this (pointer to memory, length of the query, etc). Then, the second `for` loop opens each database of the total `number_of_dbs`. After this, the third `for` loop accesses each sequence of the database previously opened, and also, extract some information of this sequence (`db_seq` that is the pointer to sequence of a database, length of the sequence, etc). Finally, with all these information of a query and a database sequence, the SW algorithm is performed by the function `perform_SW_between()`, which is the algorithm described by the equations of the section 2.1, and depicted in figures 2 and 3. After the SW algorithm is done, the results of these comparisons are used to prepare the output of the execution. The output depends on the parameters defined by the user, that commonly requires a list of the most similar sequences of the databases to the query sequences, the score of each comparison is also reported and sometime the alignment between the sequences, (as described in section 1).

As we can see in figure 3 and in the pseudocode of listing 1, there are various oportunities to extract coarse- and fine-grain parallelism in this scenario of execution of the application. From the point of view of performance, an efficient alternative depends on taking into account the existing differences between the targeted architecture. For example, the thread implementation for Altix machine differs from the IBM Cell BE implementation due to the different programming model that the latter architecture allows. In the next section, we describe different thread implementations of the Ssearch application depending on the target machine.

Listing 1: Brief Example of the Typical code in Sequence Comparison problem

```
  ...
/* number_of_query_seqs: Number of query sequences
   to compare against all the databases */
/* number_of_dbs: Number of databases used in
   the comparison */
/* number_of_seqs_in_db: Number of sequences
   stored in a SPEcific database */
/* query_seq: Pointer to a query sequence */
/* query_length:Length of a query sequences */
/* database: Pointer to a database of sequences */
/* db_seq: Pointer to a sequence from a db */
/* db_length: Length of database sequence */

main{
...
 /* First Loop (external loop) */
 for(i==0; i<number_of_query_seqs) {
  ...
  query_seq =get_pointer_to_the_query_seq(i);
  query_length = get_length_of(query_seq);
  ...
  /* Second Loop (internal loop) */
  for(j==0; j< number_of_dbs) {
   ...
   database= get_pointer_to_the_db(j);
   ...
   /* Third Loop (a more internal loop) */
   for(k=0; k<number_of_seqs_in_db(database)) {
    ...
    db_seq= get_pointer_to_the_seq(k,database);
    db_length = get_length_of(k,db_seq);

    score=perform_SW_between(query_seq,
                             query_length,
                             db_seq, db_length);
    process_result(score);
   }
  }
 }
}
```

3. ARCHITECTURES DESCRIPTION

In order to analyse how the SW algorithm takes advantage of different levels of parallelism, we select a broad spectrum of modern parallel architectures that are able to exploit different levels of parallelism. Most of them can exploit ILP, DLP and TLP concurrently. This section describes these architectures paying special attention to the capability of each one to extract the mentioned levels of parallelism, which impact positivelly the performance of the SW algorithm.

3.1 SGI Altix architecture

SGI Altix is a shared memory machine, with a Non-uniform Memory Access (cc-NUMA) architecture with 64 dual core Montecito(IA-64) processors. Each one of the 128 cores works at 1,6 GHz, with a 8MB L3 cache and 533 MHz Bus, and the system has a total 512 GB RAM. Each processor core maintains context for two threads of execution (hardware multithreading). This architecture also includes advanced microarchitetural mechanisms for exploiting ILP like predication, speculation, branch prediction, register renaming, etc. Each 128-bit instruction word contains three instructions, and it can fetch up to two instruction words per cycle from the L1 cache. Then, processor can execute up to six instructions per cycle. There are thirty functional execution units organized in eleven groups.

3.2 Power6 Architecture

The Power6 is a two-way simultaneous multithreaded (SMT) dual-core processor, where each core has two integer units, two binary FP units, one decimal floating-point unit and one SIMD unit. Each core has a L1 cache of 128 KB (64 for KB data + 64 KB for instructions) and a 4 MB semi-shared L2 cache, where the cache is assigned a specific core, but the other has access to it. The two cores share a 32 MB L3 cache which is off die. An interesting observation is that Power6 architecture is an in-order design (Previous Power5 machine was an out-of-order execution.) The machine we use in this work is a IBM Bladecenter JS22, which combines four 4.0 GHz Power6 cores. So it is possible to to use the machine as a eight SMT processor.

As mentioned, each Power6 core includes an AltiVec SIMD unit. This Altivec unit contains seven distincts execution subunits: two load subunits, one store subunit, one simple arithmetic subunit (XS), one vector FP unit (VFP), one complex unit (XC) and one vector permutation unit (VP). However, despit the number of SIMD units, the instruction issue of the core is done by group of 6 instruction and each group can have up to two SIMD instruction. The execution latencies of the SIMD units are: eight cycles for VFP, three cycles for XS, seven cycles for XC and four cycles for VP operations.

3.3 Marenostrum Architecture

MareNostrum [3] is a large parallel computer built with 10240 IBM PowerPC 970MP processors at 2.3 GHz (2560 JS21 blades). Each 970MP processor is a dual core PowerPC RISC. Each core is equipped with 64 KB instruction/32 KB data L1 cache and a a 1 MB L2 memory. Each node of Marenostrum has two 970MP, then, each node can sustain up to four threads running concurrently.

The 970MP core includes advanced microarchitetural mechanisms for exploiting ILP like branch prediction for up to two branches per cycle, out-of-order issue of up to 10 operations into 10 execution pipelines, register renaming. It can fetch up to eight instruction per cycle, and contain 10 execution pipelines: Two load/store units, two fixed-point units, two floating-point units, one branch units, one Condition register operation unit, one vector permute unit, one vector arithmetic/logic unit (for SIMD operations).

The execution latencies of the SIMD units are: eight cycles for VFP, three cycles for XS, seven cycles for XC and four cycles for VP operations. VPU execution pipelines: Vector simple fixed: 1-stage execution, Vector complex fixed: 4-stage execution, Vector floating point: 7-stage execution, VPERM: 1-stage execution. It is important to remark that in our experiments, we only use one node of Marenostrum, that is, we use up to 4 thread working simultaneosly. Then, when we refer to "Marenostrum" we mean one node of it.

3.4 Cell BE Architecture

Cell BE architecture [18] includes a general purpose processor (PowerPC Processing Unit PPU) with in-order execution, connected to 8 128-bit SIMD cores called Synergistic Processing Elements (SPEs). Each SPE has a 256KB scratch pad memory called Local Storage (LS). The nine cores are connected through the Element Interconnect Bus (EIB) which is a circular bus made of two channels in opposite directions each. It is also connected to the L2 cache and the memory controller.

The PowerPC processor is a two-way simultaneous multithreaded core which includes one Altivec SIMD support and acts as a master for the eight SPEs. The PPE works with conventional operating systems, while the SPEs are RISC processing units that can be seen as accelerators and execute a SIMD ISA with 128 128-bit SIMD registers, but they can not run an operating system. The PPE contains a 32 KB instruction and a 32 KB data L1 cache and a 512 KB L2 cache. An SPE can operate on 16 8-bit integers, 8 16-bit integers, 4 32-bit integers, or 4 single precision floating-point numbers in a single cycle, It is important to remark that SPEs cannot directly access system memory. The 64-bit virtual memory addresses formed by the SPE must be passed from the SPE to the SPE memory flow controller (MFC) to set up a DMA operation within the system address space.

Our experiments run on a IBM BladeCenter QS22 system that includes two Cell BE. As a result, it is possible to have up to four thread running on PPEs and up to sixteen SPEs can be used concurrently. The machine run at 3.2 GHz.

4. PARALLEL STRATEGIES AND IMPLEMENTATION ON DIFFERENT MACHINES

Taking into account the parallel characteristics of the application that have been described in sections 2.1 and 2.3, and the computing platforms where the application runs, it is possible to define several parallel strategies for implementing the algorithm. This section describes several implementations that extract the coarse-grain parallelism using multiple threads (TLP), where each thread works on a specific processing unit of determined computing machine; also, some implementations extract the fine-grain parallelism making use of SIMD capabilities of the different machines. Here, the main idea is to extract parallelism as figure 3 (part 2 and 3) shows. That is, the antidiagonal parelellism.

There are several ways of implementing a parallel version of Ssearch. From the point of view of performance, an efficient alternative depends on taking into account the existing differences between the targeted architecture. For example, the thread implementation for a share memory machine (like the Altix machine) differs from the distributed memory machine (like the IBM Cell BE processor) implementation due to the different programming model that the latter architecture allows. In the next section, we describe different thread implementations of the Ssearch application depending on the target machine.

4.1 Shared Memory Implementations

One TLP alternative is shown in figure 4. Here, a master thread is in charge of taking each database sequence and put it into a work queue (buffer). It will be called a task to process. Each task in the buffer contains all the information about the database sequence that will be compared against the query. There is a worker thread that processes the tasks of a buffer, makes the comparison of the sequences and every time a comparison finishes, the worker returns the score to the buffer. When the worker thread finishes the

processing of all the tasks of a buffer, it tries to own another buffer that is already filled by the master and repeats the process until all the buffers have been processed (when all the database sequences are processed). To avoid unnecessary stalls of the workers, this implementation needs twice buffers than worker threads. The master is also in charge of processing the resulting scores, taking into account, the input parameter given by the user.

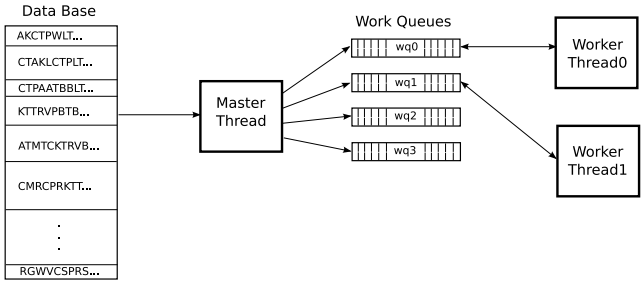

Figure 4: Thread implementation for a share memory machine (like Altix, Power6 and Marenostrum machines)

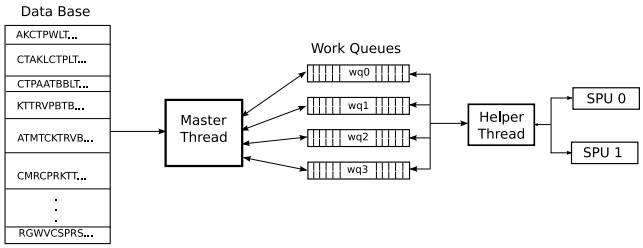

Figure 5: Alternative 1, for a thread implementation on a distributed memory machine (like the IBM Cell BE multiprocessor)

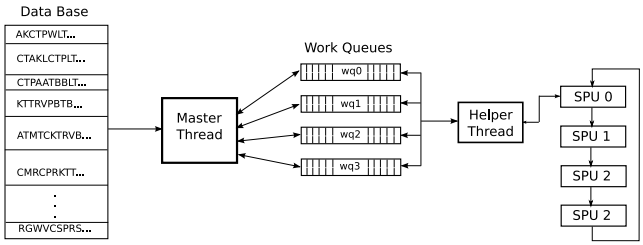

Figure 6: Alternative 2, for a thread implementation on a distributed memory machine (like the IBM Cell BE multiprocessor)

We use this approach for the shared memory machines (that is, SGI Altix, Power6 and MareNostrum architectures). Additionally, for Power6 and Marenostrum we combine the exploitation of this coarse-grain parallelism with the SIMD approach to extract fine-grain DLP. Then, we use a version of the SW algorithm that performs the sequence-to-sequence comparison taking advantage of the Altivec SIMD extension of PowerPC architecture that works with 128-bit wide registers. This is done calculating temporal vector of scores of cells parallel to the antidiagonals, as it is shown in figure 3 (part 3). In the computation of the matrix, some temporal results have to be stored in memory because they will be used in the computation of a vector in the next row.

4.2 Distributed Memory Implementations

Multiprocessor architectures on a single chip, like Cell BE, combine the capabilities of extracting coarse-grain parallelism, with the capability of extracting fine-grain parallelism using SIMD architecture accross the hetereogenous processing units that are part of it. Due to the diverse oportunities that Cell BE architecture offers to the programers, it is important to consider how the algorithm can be implemented and which characteristics of the Cell BE are more relevant in terms of performance. Here, we study 2 different implementations of the SW algorithm on Cell BE. Porting the SW implementation of Ssearch to Cell BE ISA has relevant details that impact the performance. Some relevant issues are related to the fact that each SPE has its own LS memory to be shared by data and instructions, the synchronization between PPU and SPEs, the length of the sequences to compare. Some of these aspects are analysed in the next sections.

4.2.1 Parallel Ssearch on Cell BE, version 1

In our first approach, to extract TLP, we implement the mechanism described in figure 5, where a master thread performs basically the same work described in section 4.1.

A helper thread takes tasks from a buffer and sends them to a specific SPE that performs the comparison, which is the heaviest part of the application. There is only one helper thread feeding all the SPEs that perform comparsions. This helper is responsible for interacting with the SPEs. This interaction includes synchronization, sending parameters and receiving results, etc. The helper thread sends tasks to the SPEs in a round robin fashion. When all the SPEs are working, the helper waits for the finalization of one to asign more tasks to it. When a SPE has finished a comparison, it sends the result of the computation to the helper thread that update this value into the correspondent entry in the buffer. The advantage of this scheme is that there are only two threads working in the PPE and there can be many SPEs working in parallel on independent data (independent sequences of a database).

4.2.2 Parallel Ssearch on Cell BE, version 2

Another parallel alternative is shown in figure 6. In this case, all the available SPEs perform the comparison between the query and a single sequence of a database. In this case, the computation of each matrix is distributed between the SPEs, that is, each SPE is responsible for computing several rows of the matrix. For example, when 8 SPEs are used, SPE0 computes row 0, row 8, row 16, etc; SPE1 computes row 1, row 9, row 17, and so on. There are some possible advantages of this alternative and the following issues are important: better bandwidth utilization and scalability than the previous approach. Improved bandwidth utilization can be achieved since not all the temporal results are written to main memory (the SPEs exchange data in a streaming fashion). This approach is more scalable with increasing sequence sizes because each SPE is holding smaller pieces of data in its local memory as compared to version 1 above. This alternative, however, requires additional synchronization between SPEs that can potentially degrade the performance. This alternative will be very efficient when very large and few sequences are compared. Although this alternative is discussed here, it is important to remark that currently we are working on

having a final version of code for this implementation. For this reason, there is not evaluation results for this alternative yet.

Table 1 summarizes the studied SW implementations of the Ssearch application, the machines used and some description of the capability to extract coarse- and fine-grain parallelism of each machine.

5. EXPERIMENTAL METHODOLOGY

As a starting point of our study, we execute all the SW implementations and take results accross different number of processing units. We discuss these results in terms of execution time and reached speedups.

In the sequence comparison problem, the length of sequences is an important factor that compromise the performance. We perform an evaluation of the SW algorithm implementations with two sets of sequences with different length that lead to two scenarios: First, when comparisons are performed between sequences with a relatively short length, that is, sequence lengths < 2500 symbols. Second, when comparisons are performed between longer sequences (length size > 4000). This issue is especially important for the application implementation on the IBM Cell BE machine because due to the fact that the 256KB local memory space of each SPE should be shared between data and program. For this reason, we distinguish the following two different scenarios of the SW execution:

- *Comparing Short Sequences:* Following the process explained in figure 3 (part 3), when implementing the SW algorithm on Cell BE, every time a vector antidiagonal is computed, some temporal results have to be stored because they will be used in the computation of a vector in the next row. When the query and database sequences are short enough, all the temporal values of a row can be kept in the Local memory of the SPE, which helps to reduce significatively the amount of DMAs operations from LS to main memory and viceversa. We say that a sequence is short when its lenght is < 2500 symbols. With this limit, we ensure that data and program fit in the LS without having to send them to memory via DMA.

- *Comparing Long Sequences:* When sequences are longer, the process requires that temporal computations (the border between rows in figure 3) have to be stored back in memory instead of using the local store memory (LS). This is because the amount of data (that depends on the sequence lengths) 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 SPE 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.

For the Ssearch inputs, we use several protein query sequences against the SwissProt database [4]. These queries represent a range of well characterized protein families used in other works to evaluate different alignment approaches [26]. The SwissProt database contains around 405506 sequence entries. We used the *blosum62* amino-acid substitution score matrix [13]. The applications are implemented in C. For Cell BE, the code segments running on the PPE side were compiled with ppu-gcc 4.1.1 with -O3 -maltivec options. The code in the SPE side was compiled using SPE-gcc with -O3 option. For the rest of the platforms we use the gcc compiler 4.2.1 with -O3 optimization level.

6. EXPERIMENTAL RESULTS AND PERFORMANCE ANALYSIS

This section discusses the results of the performed experiments. We have looked mostly at parallel execution issues due to the parallelism exiting in the SW algorithm.

6.1 Execution Time on Different Architectures

Figures 7 and 8 show results when comparing short and long sequences respectively. As expected, scalar version of Ssearch, (that is, the version that runs on the Altix machine) is less efficient than the remaining alternatives. Although this version is an implementation of the SW algorithm that exploit TLP, it also can benefit from the ability of the processor to exploit ILP. However, these two levels of parallelism are not enough for the SW algorithm.

An interesting observation comes from the comparison of the execution of Power6 and PowerPC 979MP of Marenostrum machine. Both machines can exploit TLP and fine-grain DLP using SIMD execution, however, the results show that Marenostrum surpasses Power6; for example, using two worker threads in Marenostrum and four worker threads in Power6 machine the application requires the same time to be executed. Although the Power6 is a very especialized architecture and its operation frequence is greater than the PowerPC 970MP, Power6 is an in-order machine while PowerPC 970MP is an out-of-order one, that is, the latter can combine the capabilities to exploit TLP and DLP with ILP also. Additionally, as explained in section 3, the execution latencies for SIMD units in Power6 are longer than the latencies in the PowerPC 970MP.

On the other hand, Cell version delivers a greater performance than Power6 machines, but less than Marenostrum. This is an interesting result because it shows that the PowerPC 970MP architecture can overcome SPEs of Cell. As an example, for short sequence comparisons, when two worker threads are used in Marenostrum, it is 18% faster than the `Cell-opt1` version with two SPEs; while the same PowerPC 970MP is only 16% slower than Cell with three SPEs. Additionally, using four worker threads in Marenostrum has equal performance than using Cell with seven SPEs. Similar behavior is presented when longer sequences are compared.

6.2 Speedup

Figures 9 and 10 show the results of speedup for the evaluated architectures (for short and long sequences respectively). The X axis means number of worker threads for Altix, Marenostrum, and Power6 architectures. It means

Table 1: Evaluated SW implementations

Name	Machine	Strategy	Coarse-grain parallelism (maximun number of threads)	Fine-grain parallelism
altix-scalar	SGI Altix	Figure 4	1 master + 16 workers	ILP in the Itanium superscalar machine
Marenostrum-1	MareNostrum, 1 node	Figure 4	1 master + 4 workers	DLP with Altivec SIMD extension
Power6	Power6 machine	Figure 4	1 master + 8 workers	DLP with Altivec SIMD extension
Cell-opt1	PPU + n SPEs	Figure 5	1 master + 1 helper + 16 workers	DLP with SIMD Cell ISA

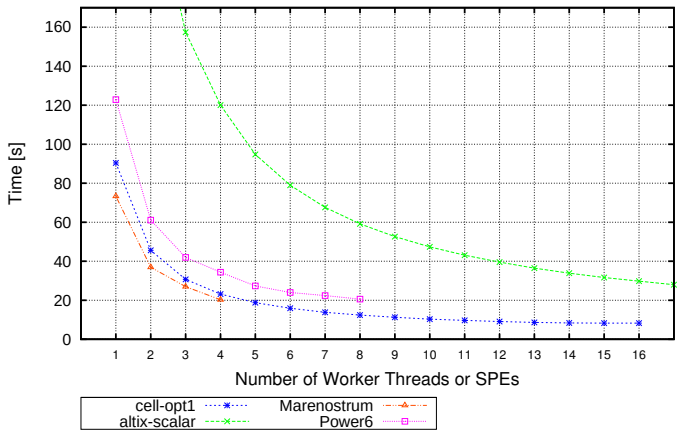

Figure 7: Execution Time for short sequences

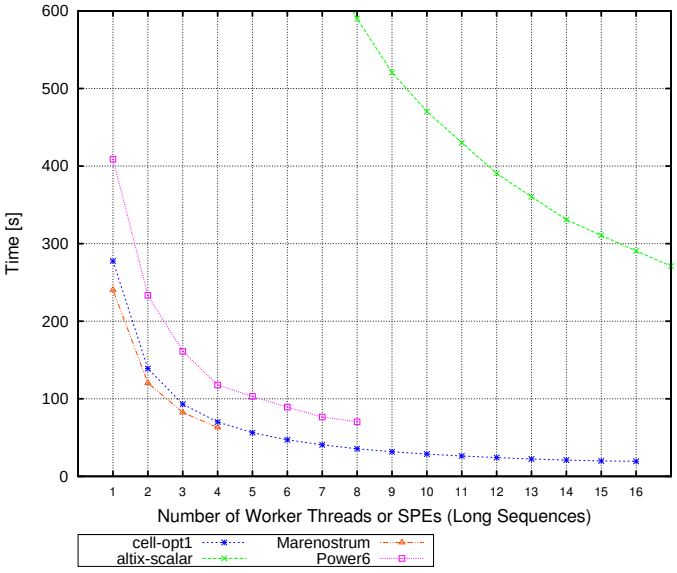

Figure 8: Execution Time for long sequences

number of SPEs for Cell BE. The Y axis means the speedup for each architecture. As can be seen, the scalar implementation of the algorithm (in the Altix machine) can reach a speedup very near to the ideal speedup, (however, remember that the execution time was the worst among all the architectures). This is due to the fact that the scalar algorithm (which only exploit ILP) is too slow, and most of the time is wasted in the basic kernel of the algorithm, making that not other issues of the execution become relevants like communication, waits of the threads, etc.

Another important observation is related to the speedup that the Power6 and Marenostrum machines reach. Both machines can take advantages of the SIMD approach to exploit fine-grain DLP. The speedup is almost lineal when less than 4 worker threads are used. After this, the behavior is not lineal. The same observation is done for the Cell version of the application. Here, we can see that the Cell machine can exploit both fine- and coarse-grain parallelism in a very efficient way when using no more than 7 or 8 workers (in this case, X axis represents the number of workers SPEs as we explained in sections 4.2.1). After that, for short sequences the speedup does not increase linearly when more SPEs are used. However the contrary behavior is observed when long sequences are compared. Additional experiments have shown that when short sequences are compared and many processing units are used, the buffers are consumed faster and the master thread spends a significant percentage of time reading the sequences of the database, becoming the bottleneck of the application. That is, reading the database becomes critical. On the other hand, each worker thread is spending most of the time performing computation.

7. RELATED WORK

A wide range of proposals aimed at reducing the execution time of the SW algorithm targeting parallel machines with multiple processing units, has been developed and evaluated. There are basically, two parallel strategies to perform sequence comparison on large databases. The first strategy involves the use of several processors that cooperate between them to compute each comparison [8] [20] [16] [33]. This is a way to exploit fine-grain parallelism among processing units. The other strategy consists in distributing the whole database between the available processors and then, each processor performs comparisons independently [11] [22]. The first solution requires that communication between processors is fast enough to perform all the computation coopearatively. Additionally, the synchronization between processor plays an important roll to obtain significant performance. The second solution uses the processing units of a parallel machine in such a way that each processor performs its comparisons independently. This solution does not require very fast communication mechanisms between processors. Also, the synchornization between them is not very relevant. Here, the most important issue is the capacity of each processor

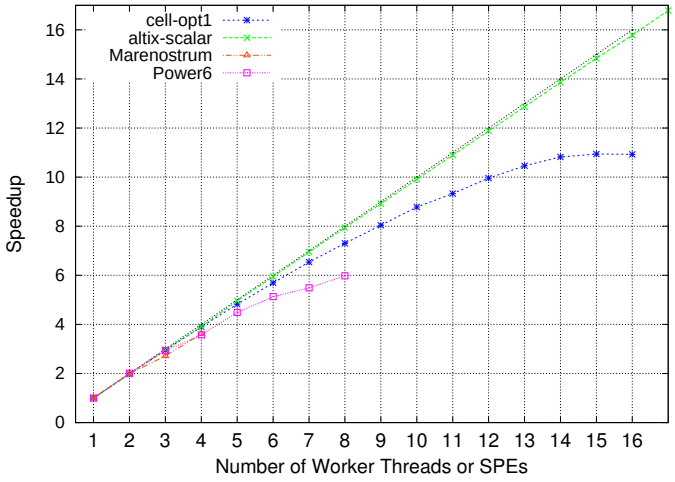

Figure 9: Speedup for short sequences

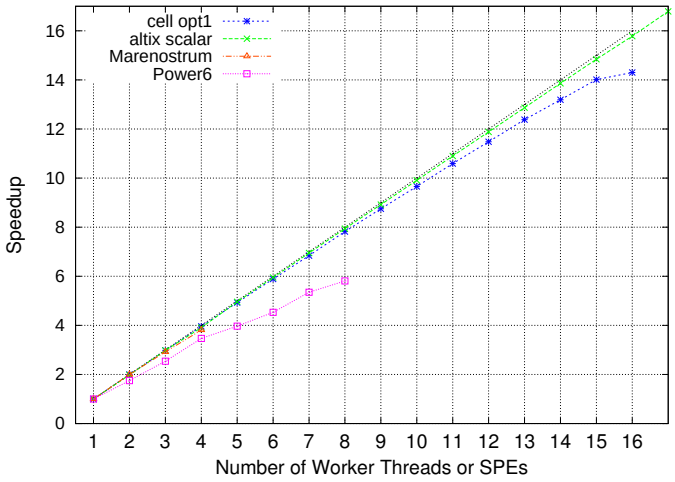

Figure 10: Speedup for long sequences

to perform computations efficiently. The load balancing is another important aspect in order to assign approximately the same amount of work to each processor.

On the other hand, there are evaluations of the SW algorithm that combine superscalar processing and Single-Instruction Multiple-Data processing on augmented general purpose processors [26, 29] to exploit fine-grained parallelism in the sequence alignment problem. However, due to the permanent and almost exponential growth of the amount of biological data, it is clear that this solution alone does not satisfy the performance demands imposed by this field.

Combining the SIMD approach inside a processing unit, with multiple processing nodes to distribute the work is an alternative that combines the advantages of the mentioned strategies. Some modern multi-processor architectures on a single chip, e.g, the Cell BE [18], CUDA [27], Power6 [19]; combine the parallelism benefits of multiprocessor systems, with the higher speed interconnects of the systems on a chip and the lower power consumption. However, some of these alternatives have not been studied in depth as a solution for the sequence alignment problem and in general, for bioinformatics applications. Sachdeva et. al [28] present some results on the viability of Cell BE for bioinformatics applications, all performing sequence alignment. In this case, the authors describe the use of several SPEs for a pairwise alignment of only 8 sequence pairs that fit entirely in the LS, however the analysis does not include more complex scenarios where large databases and long sequences are compared. Svetlin [21] describes a SW implementation on the last-generation Graphics Processing Units (GPU) G80 of Nvidia, but again, the main goal of the work is the mapping of the algorithm on the architecture of the mentioned GPU.

8. CONCLUSIONS AND FUTURE WORK

In this paper we studied the performance behavior of SW algorithm, which is one of the most well-known sequences comparison algorithm, on different computational platforms. We have observed that due to the embarrasing parallelism of the algorithm and the possible scenarios of use, it requires parallel architectures that can exploit both coarse- and fine-grain parallelism efficiently and even also ILP.

We see that Cell BE architecture is very efficient exploiting the first two mentioned level of parallelism that the application needs. However, with a deeper performance analysis, we realized that one of the main reasons for performance degradation when Cell BE is used, is the limited performance that the PPE delivers. The results shows that even though the SPEs are very efficient accelerators for sequences comparison problem, the modest characteristics of the PPE leads as a consequence that the threads working in this part of the Cell BE, cannot execute their tasks as fast as SPEs needs. In the SW implementation on Cell, the part of the code that runs in PPE is basically a scalar code which could benefit from the ability of a processor that can exploit ILP more efficiently than the current PPE. Results for PowerPC 970MP architecture of Marenosturm manchine show that combining the benefits of exploiting TLP, DLP and ILP is needed to speedup the SW algorithm. On the other hand, a machine like Altix which can only extract TLP and ILP is not enough to satisfy the challenges that sequence comparison problem represents.

However, additional experiments are being performed in order to investigate which algorithmic and architectural improvements can be carried out to increase the speedup even more. Also, we are investigating other possibles scenarios where sequence comparison is requiered and how these scenarios present challenges to these architectures. Also, our current and future work involves the usage of architecture simulation techniques in order to evaluate possible solutions to the identified limitations of the Cell BE architecture. Our final goal is to use this research as guidance for the architecture design of future multi-core systems targeting bioinformatics applications. We also intend to widen our study to other applications of the same bioinformatic field. This work is one step towards the definition of a future multi-core architecture incorporating domain specific bio-accelerators. We believe that multi-core architectures able to exploit multiple dimensions of parallelism are a valid option that will play an important role in the future of bioinformatics.

9. ACKNOWLEDGEMENTS

This work was sponsored by the European Commission in the context of the SARC Integrated Project #27648 (FP6), the HiPEAC Network of Excellence, the FEDER funds under contract TIN2007-60625 and by the Spanish Ministry of Science. The authors would like to thank the Barcelona Supercomputing Center (BSC) for supplying the computing resources for our research.

10. REFERENCES

[1] Dna data bank of japan. http://www.ddbj.nig.ac.jp/.

[2] Embl nucleotide sequence database. http://www.ebi.ac.uk/embl/.

[3] The marenostrum machine. http://www.bsc.es.

[4] Swissprot, universal protein database. http://www.expasy.org/sprot/.

[5] Bioinformatics market study for washington technology center, June 2003. www.altabiomedical.com.

[6] S. F. Altschul, W. Gish, W. Miller, M. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.

[7] D. Benson, I. Karsch, D. Lipman, J. Ostell, B. Rapp, and D. Wheeler. Genbank. *Nucleic acids research*, 28(1):15–18, 2000.

[8] A. Boukerche, A. C. M. A. de Melo, M. Ayala-Rincon, and T. M. Santana. Parallel strategies for local biological sequence alignment in a cluster of workstations. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 15*. IEEE Computer Society, 2005.

[9] E. T. Chow, J. C. Peterson, M. S. Waterman, T. Hunkapiller, and B. A. Zimmermann. A systolic array processor for biological information signal processing. In *ICS 91: Proceedings of the 5th international conference on Supercomputing*, pages 216–223, 1991.

[10] M. Dayhoff, R. Schwartz, and B. Orcutt. A model of evolutionary change in proteins. *Atlas of Protein Sequence and structure*, 1978.

[11] A. Deshpande, D. Richards, and W. Pearson. A platform for biological sequence comparison on parallel computers. *Comput. Appl. Biosci.*, 1991.

[12] O. Gotoh. An improvement algorithm for matching biological sequences. *Journal on Molecular Biology*, 162:705–708, 1982.

[13] J. Henikoff, S. Henikoff and S. Pietrokovski. Blocks+: a non-redundant database of protein alignment blocks derived from multiple compilations. *Bioinformatics*, 15, 1999.

[14] S. Henikoff and J. Henikoff. Amino acid substitution matrices from protein blocks. *Proceding in Natural Academic Science*, 89, 1992.

[15] D. Higgins, J. Thompson, T. Gibson, and J. Thompson. Clustal w: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22:4673–4680, 1994.

[16] R. Hughey. Parallel hardware for sequence comparison and alignment. *Proceedings of Int. Conf. Application Specific-Array Processors. IEEE Computer Society*, Sep 1996.

[17] B. Huhg, W. David, and J. Alexander. Strategies for searching sequence databases. *BioTechniques*, 28(6), 2000.

[18] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, and D. Shippy. Introduction to the cell multiprocessor. *IBM Systems Journal*, 49(4/5):589–604, 2005.

[19] H. Q. Le, W. J. Starke, J. S. Fields, F. P. O'Connell, D. Q. Nguyen, B. J. Ronchetti, W. M. Sauer, E. M. Schwarz, and M. T. Vaden. Ibm power6 microarchitecture. *IBM J. Res. Dev.*, 2007.

[20] W. Liu and B. Schmidt. Parallel design for computational biology and scientific computing applications. *IEEE International Conference on Cluster Computing (CLUSTER 03)*, 2003.

[21] S. A. Manavski and G. Valle. Cuda compatible gpu cards as efficient hardwarer accelerator for smith-waterman sequence alignment. *BMC Bioinformatics*, 9, 2008.

[22] P. L. Miller, P. M. Nadkarni, and W. R. Pearson. Comparing machine-independent versus machine-specific parallelization of a software platform for biological sequence comparison. *Comput. Appl. Biosci.*, 1992.

[23] S. Needleman and C. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.

[24] W. R. Pearson. Searching protein sequence libraries: comparison of the sensitivity and selectivity of the smith-waterman and FASTA algorithms. *Genomics*, 11:635–650, 1991.

[25] W. R. Pearson and D. J. Lipman. Improved tools for biological sequence comparison. In *Proc. Natl. Acad. Sci (USA).*, pages 2444–2448, Apr. 1988.

[26] T. Rognes. Rapid and sensitive methods for protein sequence comparison and database searching. *Phd Thesis, Institue of Medical Microbiology. University of Oslo*, 2000.

[27] S. Ryoo, C. I. Rodrigues, S. S. Baghsorkhi, S. S. Stone, D. B. Kirk, and W. mei W. Hwu. Optimization principles and application performance evaluation of a multithreaded gpu using cuda. In *PPoPP '08: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*. ACM, 2008.

[28] V. Sachdeva, M. Kistler, E. Speight, Tzeng, and T.H.K. Exploring the viability of the cell broadband engine for bioinformatics applications. *Proceedings of the 6th Workshop on High Performance Computational Biology*, pages 1–8, 2007.

[29] F. Sanchez, E. Salami, A. Ramirez, and M. Valero. Performance analysis of sequence alignment applications. *Proceedings of the IEEE International Symposium on Workload Characterization. IISWC 2006.*, 2006.

[30] D. Sankoff. The early introduction of dynamic programming into computational biology. *Bioinformatics*, 16:1:41–47, 2000.

[31] P. Sellers. On the theory and computation of evolutionary distances. *SIAM J. Appl. Match*, 26:4:787–793, 1974.

[32] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[33] G. Tan, N. Sun, and G. R. Gao. A parallel dynamic programming algorithm on a multi-core architecture. In *SPAA '07: Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures*, 2007.