Performance Analysis of Sequence Alignment Applications

Friman Sánchez, Esther Salamí, Alex Ramirez and Mateo Valero
HiPEAC European Network of Excellence and Barcelona Supercomputing Center (BSC)
Universitat Politècnica de Catalunya (UPC), Barcelona, Spain
{fsanchez, esalami, aramirez, mateo}@ac.upc.edu

Abstract*— Recent advances in molecular biology have led to a continued growth in the biological information generated by the scientific community. Additionally, this area has become a multidisciplinary field, including components of mathematics, biology, chemistry, and computer science, generating several challenges in the scientific community from different points of view. For this reason, bioinformatic applications represent an increasingly important workload. However, even though the importance of this field is clear, common bioinformatic applications and their implications on micro-architectural design have not received enough attention from the computer architecture community. This paper presents a micro-architecture performance analysis of recognized bioinformatic applications for the comparison and alignment of biological sequences, including BLAST, FASTA and some recognized parallel implementations of the Smith-Waterman algorithm that use the Altivec SIMD extension to speed-up the performance. We adopt a simulation-based methodology to perform a detailed workload characterization. We analyze architectural and micro-architectural aspects like pipeline configurations, issue widths, functional unit mixes, memory hierarchy and their implications on the performance behavior. We have found that the memory subsystem is the component with more impact in the performance of the BLAST heuristic, the branch predictor is responsible for the major performance loss for FASTA and SSEARCH34, and long dependency chains are the limiting factor in the SIMD implementations of Smith-Waterman.

I. Introduction

Advances in molecular biology and genomic research have led to a growth in the biological information generated by the scientific community. This information increases continuously in such a way that it has been necessary to create specialized databases to store, organize and index the data [10] [4] [2] [1]. Additionally, the use of specialized tools to view and analyze this information makes computational biology a very multidisciplinary field, including components of mathematics, biology, chemistry and computer science. 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. Because of that, computational biology and its related components in database systems, visualization and analysis tools become an emerging workload that requires high-performance computer systems.

One of the most important tasks in molecular biology is the comparison and alignment of biology sequences. This is basically a problem of finding an approximate pattern matching between two sequences, possibly introducing spaces (gaps) between them. The biological information associated with the similarity between sequences provides important knowledge of the significant relationships between gene sequences (DNA, RNA) or protein sequences of living organisms. A high sequence similarity, for instance, usually implies functional or structural similarity, while sequence differences hold the key information regarding diversity and evolution. For example, consider the alignment between sequences `A=csttpggg` with eight residues (symbols) and `B=csdtnglawgg` with eleven residues:

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

In this 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; and also that $a_5 = p$ 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. This task is computationally intensive and the scenario can be more challenging if it is required to detect all the significant similarities between a single encoded amino-acid sequence and hundred of thousands of protein sequences that are stored in a typical protein database, or when it is necessary to compare complete genomes of a variety of organisms.

Researchers have faced the problem at different levels of complexity. At algorithm level, researchers have proposed several methods for alignment of two sequences using dynamic programming techniques (DP) and heuristic strategies. DP techniques are probably the most important programming method used in sequences alignment [25] and many researchers have proposed different algorithms based on DP to quantify the similarity of a pair of sequences [19] [27] [30] [12]. Among these alternatives, the most important is the Smith-Waterman (SW) algorithm [30], which is generally considered to be the most sensitive. Sensitivity is a measure of how well a method can detect the similarity between sequences. Nevertheless, the computationally intensive task involved in the algorithm is a restrictive factor that prevents its use; its time complexity is $O(m, n)$, where m and n are the lengths of the two sequences respectively.

On the other hand, heuristic strategies have been proposed to speed-up the execution of the search and alignment tasks, such as FASTA [21] and BLAST [7]. These methods reduce the execution time by an order of magnitude compared to the SW algorithm. However, this reduction is obtained at the expense of sensitivity, and due to this loss of sensitivity some related sequences cannot be detected in a search. Some interesting discussion about the sensitivity and selectivity in protein similarity searches can be found in [28]. This scenario shows clearly that new computer systems that can provide high-performance on computational biology applications play an important role in the development of these areas of knowledge.

In this paper, we present a performance analysis of some of the most recognized sequence alignment tools, including the BLAST and FASTA heuristics, and the Smith-Waterman algorithm. Since SIMD extensions have proven very effective at increasing performance of the Smith-Waterman algorithm, we analyze and compare three different implementations: an optimized scalar version (SSEARCH), an Altivec implementation using 128-bit registers, and a futuristic Altivec version that uses 256-bit vector registers. Understanding the characteristics of these applications, detecting performance bottlenecks, and evaluating the different architectural tradeoffs will help designers to tune future processor architectures to increase performance on this emerging class of applications. We present a detailed workload characterization in terms of micro-architecture parameters, like IPC, execution time, instruction distribution, memory behavior, branch prediction, etc.

This paper is organized as follows. Section II discusses the related work, section III presents a brief overview of the sequence alignment task and details some relevant aspects of the studied applications, section IV describes the experimental framework and the evaluated processor configurations. Section V presents the experimental results. Finally, in section VI some conclusions and future work are presented.

II. Related Work

While it is clear that bioinformatic is an important field of research, most of the research effort is dedicated to developing better and faster algorithms, rather than to designing architectures better suited to the existing codes. There has been very little research on how bioinformatic codes exercise the different aspects of a state-of-the-art computer system, and even less have looked at the micro-architecture details of a high performance processor. Since bioinformatic is a new and emerging field, there is not a well defined benchmark. A few proposals have been presented, but they are not widely accepted in the computer architecture community yet. However, all of them recognize the importance of sequence comparison and alignment algorithms.

Most of the reported studies about bioinformatic workloads focus on parallel programming issues and performance evaluations on large high-performance supercomputers [26] [29] [32] [3]. Usually, these studies analyze the impact of the number of processors in the execution time. However, they do not analyze the performance of the individual processors. One interesting work is presented in [11] where a performance profiling of BLAST over a real machine is done (HP zx6000 workstation with dual Itanium-2 processors). Although they analyze some performance aspects of the application, their main goal is to determine performance bottlenecks and evaluate the effect of compiler optimizations on BLAST in a real machine.

One important step-forward to the definition of a representative set of bioinformatic applications is the *BioBench* Suite [6]. It includes a set of applications ranging from sequence comparison to pholygenetic analysis. A second important effort to define a bioinformatic benchmark is *BioPerf* [9]. Compared to *BioBench*, it covers more applications in terms of quantity and diversity and includes parallel codes of the applications where available. Both works have the characteristic of meeting an important group of real and mature bioinformatic tools. These studies characterize the workload behavior using the hardware performance counters on a Pentium 3 processor [6] and Pentium 4 machine [15] or using a combination of the CHUD framework and the IBM Mambo simulator running on an Apple G5 (IBM PowerPC 970) Workstation [9], [24]. The use of performance counters provides insight on the actual architecture, but cannot offer detailed information about micro-architecture parameters, or insight on future designs that change these parameters, like functional unit mix, issue width, cache size, memory latency, and so on.

We contribute with a detailed performance characterization of some of the most recognized sequence comparison tools: BLAST, FASTA, and SSEARCH, the best-known scalar implementation of the Smith-Waterman algorithm. Since previous work has identified SIMD extensions as an excellent way to improve the performance of SSEARCH [31] [22] [24], we also characterize the performance of two Altivec implementations of SSEARCH: one using the existing Altivec instruction set on 128-bit registers, and a futuristic design that uses 256-bit registers, that could exploit higher degrees of data level parallelism. As a difference with previous work, we use a cycle-accurate micro-architecture simulator that offers a higher level of detailed information than hardware performance counters, and allows us to change the configuration to explore alternative processor designs, providing a way to detect which are the most relevant processor features for each different application.

III. Sequence alignment strategies

There are significant differences among the most widely recognized applications for sequence comparison. To begin with, some of them use heuristics, while others use dynamic programming. As we will show, this has severe implications on the way they exercise the processor architecture.

In this section we briefly describe the applications evaluated in this paper. They include 3 different versions of the Smith-Waterman algorithm: a scalar version (SSEARCH), two SIMD versions using 128-bit and 256-bit registers, and 2 heuristic strategies (BLAST, FASTA) that trade off comparison accuracy for speed. Table I summarizes the selected workload and the input parameter used to execute them.

TABLE I
SELECTED WORKLOAD DESCRIPTION

Application	Description	Input Parameters
SSEARCH (SW implementation)	Best known scalar implementation of the SW algorithm. This is part of the SSEARCH program	-q -H -p -b 500 -d 0 -s BL62 -f 11 -g 1
sw_vmx128	Data-parallel implementation presented in the last SSEARCH application that use Altivec SIMD extension	-q -H -p -b 500 -d 0 -s BL62 -f 11 -g 1
sw_vmx256	Data-parallel implementation presented in the last SSEARCH application that use Altivec SIMD extension	-q -H -p -b 500 -d 0 -s BL62 -f 11 -g 1
NCBI BLAST	BLAST program [7]. Heuristic strategies	blastp -d -G 10 -E 1 -b 0
FASTA	FASTA program. It uses heuristic strategies.	-q -H -p -b 500 -d 0 -s BL62 -f 11 -g 1

Listing 1. Typical code in blast

```
...
if(s_off > compressed_wordsize)
  p=*(subject0 + s_off - compressed_wordsize -1);
  if(s_off==compressed_wordsize
      || READDB_UNPACK_BASE_4(p)!=*--q
      || q<query0) {left = 0;}
  else {
    if(READDB_UNPACK_BASE_3(p) != *--q
        || q < query0) {left = 1;}
    else {
      if(READDB_UNPACK_BASE_2(p) != *--q
          || q < query0) {left = 2;}
      else{
        if(READDB_UNPACK_BASE_1(p) != *--q
            || q < query0) {left = 3;}
        else { left = 4;}
  }}}
p = *(subject0 + s_off + extra_bytes_needed);
q = query0 + q_off + 4*extra_bytes_needed;...
```

Listing 2. Typical loops in SSEARCH34 application

```
...
while (1) {
  h = p + *pwaa++;
  p = ssj->H;
  if ((e = ssj->E) > 0 ) {
    if (p == -1) goto next_row;
    if (h < e) h = e;
    else if (h > n_gap_init) {
      e += gap_ext;
      goto transition; }
    e += gap_ext;
    ssj->E = (e>0)? e:0;
    ssj++->H = h; }
  else {
    if ( h > 0) {
      if (h > n_gap_init) {
        e = 0;
        goto transition;
      }
      ssj++->H = h;  }
    else ssj++->H = 0;
}}...
```

Listing 3. Typical loops in SW_vmx128 and SW_vmx256 applications

```
...
/* db_length: length of database sequence*/
/* query_length:length of query sequences*/
for(i=0;i<query_length;i+=8) {
  ...
  for(j=8;j<db_length;j+=8) {
     ...}
}...
```

A. SSEARCH

Current implementations of the SW algorithm benefit from the ability of the processor to exploit Instruction Level Parallelism (ILP). Some implementations also benefit from SIMD extensions to exploit Data Level Parallelism (DLP). Among those exploiting ILP, the SSEARCH tool included in the Fasta Tools is probably the fastest non-parallel implementation of a rigorous alignment algorithm between two sequences [13] [20]. The code sample shown in listing 2, with many `if-then-else` statements shows how challenging this application will be for modern superscalar processors. While this code is more efficient than a direct implementation of the SW algorithm, the branch prediction mechanism may find it difficult to anticipate the path through the complex control flow.

We also analyze the DLP variant of the SSEARCH code present in the Fasta Tools. This code uses the Altivec SIMD extension (with 128-bit wide registers) to implement a variant of the approach presented in [31], that obtains an order of magnitude higher performance than the scalar version, making it practical for real use. To our knowledge, this is the fastest implementation of the SW on a single processor. Listing 3 shows a code example taken from the SIMD implementation. We quickly appreciate two differences with the scalar implementation: first, there is no complex control flow within the loop body; second, the number of loop iterations is fixed, as it depends on the length of the sequences. In this paper, we have labeled this version as SW_vmx128.

Finally, as a novel contribution of this paper, we present a SW implementation using the Altivec SIMD extension, but using 256-bit wide registers. The increased register width should allow us to exploit more DLP, and so increase performance over the 128-bit version. The code looks very similar to the SW_vmx128 version, except that now, each Altivec instruction is operating on twice the number of data elements. We have labeled this version as SW_vmx256.

B. BLAST

BLAST (Basic Local Alignment Search Tool) is probably the most popular bioinformatic tool due to its fast speed. It is an heuristic open-source tool to determine which sequences from a database are most similar to a query sequence. The main strategy of BLAST is to find a pair of segments of identical length from two sequences such that extending or shortening both segments will not improve the similarity score of the segment pair. This heuristic compromises selectivity

TABLE II

QUERY SEQUENCES USED IN THE EVALUATIONS

Protein Family	Accession (ID)	Length (symbols)
Globin	P02232	143
Ras	P01111	189
Glutathione S-transferase	P14942	222
Serine Protease	P00762	246
Histocompatibility antigen	P10318	362
Alcohol dehydrogenase	P07327	375
Serine Protease inhibitor	P01008	464
Cytochrome P450	P10635	497
H+-transporting ATP synthase	P25705	553
Hemaglutinin	P03435	567

(number of matched segments) and sensitivity for speed. A detailed description of the algorithm and other performance improvements can be found in [7] and [8]. A profiling analysis shows that subroutine *BlastNtWordFinder* contributes about 75% of the total execution time. Listing 1, shows a small part of code taken from this function. We can see that although this application is faster than previous ones, also, the `if-then-else` statements and other characteristics like the intensive use of pointer arithmetic, large data structures and control flow statements limit its performance on superscalar processors.

C. FASTA

FASTA [20] is a collection of popular bioinformatic codes that perform a fast protein or nucleotide sequence comparison against a query sequence. The FASTA algorithm is another approximate heuristic that prescreens the database for very short identical matches and then (if optimization of scores is used) extends these matches. Although not as optimal as the SW algorithm, the FASTA algorithm offers a different trade off between comparison accuracy and execution time [5].

IV. EXPERIMENTAL METHODOLOGY

In order to carry out a performance analysis over the mentioned sequence alignment applications, we use a simulation-based methodology. In this section we describe the simulation infrastructure, the selected input working data set, the evaluated processor models and the code generation methodology.

A. Input Working Set and Database

Evaluations were done using a set of 11 different amino-acid query sequences against the SwissProt [4] database. These 11 sequences represent a range of well characterized protein families. The length of these sequences ranges from 143 to 567 amino-acids. It is important to remark that the same set of queries has been previously used by other researchers to evaluate different alignment approaches, like BLAST and ParAlign [7] [23]. Some characteristics of these sequences are listed in table II. Currently, the SwissProt database contains more than 62,615,309 residues grouped into 172,233 protein sequences. All the database searches have been executed with a gap open penalty of 10 and a gap extension penalty of 1. Additionally, the *blosum62* amino-acid substitution score matrix has been used [14]. For space reasons, in this paper we only show the results obtained using the query sequence *Glutathione S-transferase*.

TABLE III

TRACE SIZE

APPLICATION	Instruction count
SSEARCH	319.808.539
SSEARCHVMX128	78.993.134
SSEARCHVMX256	65.570.645
FASTA	27.469.429
BLAST	7.749.725

B. Simulation Framework

We selected the PowerPC architecture with the Altivec SIMD extension as a base of study. Altivec is one of the most complete multimedia extensions in terms of the number of registers, number of operations and support for data reorganization. The simulation process has been carried out using Turandot [17] [18], which is a cycle accurate simulator that models a Power-PC based out-of-order architecture and supports trace-driven simulation. In order to simulate properly the code that contains Altivec instructions (SW_vmx128 and SW_vmx256), we included Altivec support in the simulator. It is important to remark that applications for biological sequence comparison have a very large scale behavior (behavior seen over billions of instructions). For this reason, it is required to generate representative traces of the execution. To do this, we have adapted the Aria tool, which is a framework for doing architecture and micro-architecture studies over the PowerPC architecture based on the MET tools from IBM. Table III summarizes the trace sizes in number of instructions. The traces have been generated taking into account the behavior of the applications described in [24] and in order to do comparisons, the traces belong to the execution on the same sequences of the database. Experiments performed over bigger traces showed similar trends as those presented in this work. To generate the assembler code, we used the GCC compiler and the GNU binutils. We also extended the GCC back-end to include Altivec intrinsics manipulating 256-bit registers.

C. Processor Models

The tools used in this study simulate a fully parameterizable superscalar processor. Manipulating these parameters we can cover a wide range of configurations to represent multiple processor design options. As shown in tables IV, V, and VI we have explored variations of the fetch/issue/commit width, the cache hierarchy, and the branch prediction mechanism. The 4-way configuration represents mainstream superscalar processors such as the PowerPC 970, or Alpha 21264. The 8-way configuration represents more aggressive superscalar designs like a possible Power 6, or the Alpha 21464. The 16-way configuration is presented as a limit situation, where the maximum ILP is exploited.

V. EXPERIMENTAL RESULTS

In this section we present the most relevant results and discuss the main conclusions inferred from them. First, we show the instruction distribution for each application. Then, in order to identify the main factors of performance degradation,

TABLE IV

EVALUATED PROCESSOR CONFIGURATIONS

Resource	Parameter	4-way	8-way	16-way
Width	Fetch	4	8	16
	Rename	4	8	16
	Dispatch	4	8	16
	Retire	6	12	20
	Inflight instrs	160	255	255
Physical Registers	GPR	96	128	128
	VPR	96	128	128
	FPR	96	128	128
Units	LD/ST	2	4	8
	FX	3	6	10
	FP	2	4	8
	BR	2	3	7
	VI	1	2	6
	VPER	1	2	4
	VCMPLX	1	2	4
	VFP	1	2	4
Size Queues	LD/ST issue	20	40	80
	FX issue	20	40	80
	FP issue	20	40	80
	BR issue	20	40	80
	VI issue	20	40	80
	VPER issue	20	40	80
	VCMPLX issue	20	40	80
	VFP issue	20	40	80
	Ibuffer	18	36	72
	Retire	128	180	180
DCache	Read ports (8-byte access)	2	3	7
	Write ports(8-byte access)	1	2	4
	L1 to L2 ports (32 bytes r/w)	1	1	1
	Maximum outstanding misses	4	8	16

TABLE V

EVALUATED MEMORY CONFIGURATIONS

Mem	Param	me1	me2	me3	me4	meinf
I-L1	Size [KB]	32	64	128	128	Inf
	Assoc	1	1	1	1	1
	Line [B]	128	128	128	128	128
	Lat [cycles]	1	1	1	1	1
D-L1	Size [KB]	32	64	128	128	Inf
	Assoc	2	2	2	2	2
	Line [B]	128	128	128	128	128
	Lat [cycles]	1	1	1	1	1
DL2 Shared	Size [MB]	1	2	4	Inf	Inf
	Associ.	8	8	8	8	8
	Line [B]	128	128	128	128	128
	Latency [cycles]	12	12	12	12	12
Main Mem	Latency [cycles]	300	300	300	300	300

we analyze the reason for the processor wasted cycles on the 4-way configuration. With this information, we start the analysis of each subsystem of the processor that is responsible for the lost performance, mainly the memory and branch prediction subsystems.

A. Instruction Breakdown

Figure 1 shows the instruction distribution for the evaluated applications. As it has been reported in previous work, the amount of float point instructions is negligible (these instructions are grouped into the *"other"* group). This confirms that sequence comparison applications have different nature than other scientific applications where the amount of floating point instructions is significant. The instructions distribution for the heuristic strategies and the non-parallel SW implementation (BLAST, FASTA and SSEARCH34) has significant differences compared to the parallel (Altivec) SW implementations (SW_vmx128 and SW_vmx256).

First, the number of control instructions (branches and jumps) in the parallel SW is very small (around 2%). This is due to the fact that these parallel implementations avoid the algorithm optimizations that imply many `if-then-else` sentences. Instead, most of the calculus are packed into the 128- and 256-bit registers. In the other applications, the amount of control instructions is very significant (25% in SSEARCH, 18% in FASTA and 16% in BLAST) because they are based on algorithm optimizations that avoid unnecessary computation based on specific data values. Given the high ratio of control flow instructions we can expect the branch predictor to play a significant role in these applications.

TABLE VI

BRANCH PREDICTOR CONFIGURATION

Description Parameter	Value
Branch Predictor: (Combined branch predictor that selects between gshare and bimodal).	GP
Branch Predictor Table size	16K
Branch Target Table Associativity	4
NFA table size	4K
Latency of a miss in the NFA buffer (BTB)	2 cycles
NFA associativity	4
Maximum number of predicted conditional branches	12
Miss Predicted recovery cycles	3

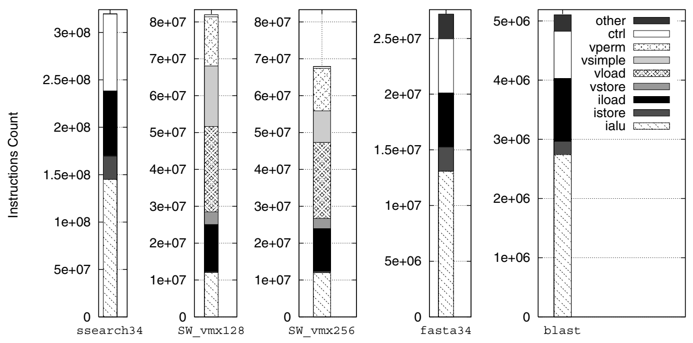

Fig. 1. Instruction breakdown for the different workloads

Second, the results reveal that there is a significant amount of load instructions in all applications. (22% in SSEARCH, 16% in SW_vmx128, 17% in SW_vmx256, 17% in FASTA and 21% in BLAST). However, the number of store instructions is much smaller, specially in the parallel applications. This is due to the fact that all the applications have to load each database sequence from memory and at the end of the computation, they do not store the same amount of data, but only (basically) the score value of the comparison. The data locality will determine the performance of the cache hierarchy.

Third, the largest percentage of the executed instructions are ALU instructions, that is, 44% integer ALU in SSEARCH34, 48% in *FASTA34*, 54% integer ALU in BLAST, 15% integer ALU and 21% integer Altivec in SW_vmx128, and 18% integer ALU and 14% integer Altivec in SW_vmx256. Comparing these results with the instruction distribution for the SPEC 2000 made in [6], where the average ALU instructions is around 40%; we see that these applications put more pressure over the execution units than the SPEC suite (integer or SIMD functional units). As we will show, this causes the applications to be effectively compute-bound. The length of dependency chains will determine the effectiveness of superscalar excution.

B. Pipeline stalls

It is common for a pipeliened processor to stall during many cycles in the execution of an application. There are many possible reasons for such pipeline stall: branch mispredictions, cache misses, full issue queues, result dependecies, etc. Our simulator framework is able to keep track of operations as they flow through the processor pipeline, and records the reason (trauma) for an operation not making forward progress. Traumas recorded by the simulator are grouped into 56 classes , as described in [16]. Table VII sumaryzes the list of traumas in the processor, for space reasons, we do not describe all the possible traumas that can be collected, but only those which have been detected to be important in the execution of our applications.

Figure 2 shows the distribution of traumas in the execution of the applications for a specific 4-way processor configuration. The results show that for BLAST, most traumas are associated with dependencies on integer operation (RG_FIX), followed by accesses to the second level cache (trauma MM_DL2), branch miss-prediction (IF_PRED), and accesses to the first-level data cache (MM_DL1). FASTA has a similar distribution of traumas. However, in SSEARCH34 the relevance of branch mispredictions is more significant than for BLAST, while cache misses play a less important role.

On the other hand, the parallel implementations of the SW algorithm, SW_vmx128 and SW_vmx256, exhibit a very different distribution of traumas, associated to the use of SIMD instructions. On both applications, the most frequent traumas are dependencies on integer SIMD operation (RG_VI) and dependencies on SIMD permutation operation (RG_VPER). However, taking into account that SW_vmx256 is similar to SW_vmx128 (the basic difference is that the first one use Altivec register with 256-bit length), it is interesting to observe that even though this version tries to extract more DLP, at the end, dependecies on SIMD permutation operations (RG_VPER) become more important and other sources of stall emerge. That is the case of accesses to the first- and second-level data cache (MM_DL1 and MM_DL2), and register dependencies with the results from memory operations (RG_MEM).

In the next subsections, we will analyze in a deeper way the impact of the most important sources of performance loss, that is: ILP exploitation on different processor configurations, the memory subsystem, and the branch prediction subsystem.

C. Instructions per Cycle

Figure 3 shows the execution time, measured in CPU cycles of each application for different processor (4-way to 16-way) and different memory configurations (I1/DL1/L2: 32k/32k/1M to infinite). We observe that the SIMD implementations of SW, and specially BLAST, are the only ones that exhibit significant differences in execution time as we change the memory hierarchy. All applications experience approximatey an 8% speedup when going from the 4-wide to the 8-wide configuration. From that point, SSEARCH and BLAST do not show any performance improvement when going to the 8-wide configuration, while the SIMD codes and FASTA still show a moderate speedup.

TABLE VII
LIST OF SOME IMPORTANT TRAUMAS

Name	Description	Stage
IF_NFA	Next Fetch Address miss-pred.	Fetch
IF_PRED	Branch miss-prediction	Fetch
IF_FULL	Instruction Buffer Full	Fetch
FULL_MEM	Too many mem instrs ready	Issue
MM_DL2	L2 cache data miss	Memory access
MM_DL1	L1 D-cache miss	Memory access
RG_FIX	Result dependency in INT units	Register dependencies
RG_MEM	Result dependency in MEM units	Register dependencies
RG_VI	Result dependency in SIMD-int units	Register dependencies
RG_VPER	Result dependency in SIMD-perm units	Register dependencies
OTHER	Miscellaneous reasons	Other reason

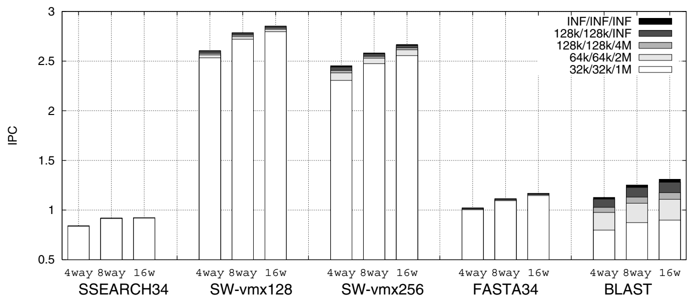

Fig. 4. IPC vs memory configuration

Figure 4 shows the number or instructions completed each cycle for the same set of configurations. The results show that, although the configurations tested can exploit high levels of ILP, they only achieve very modest IPC values. Only the SIMD implementations execute more than 2 instruction in parallel. As will be confirmed in the next subsections, the high regularity of SIMD code and the lack of complex control flow help these application to efficiently schedule instructions and exploit data locality. On the other hand, FASTA, and SSEARCH34 achieve very poor IPC results. Furthermore, the graph shows that going from a small cache configuration to an ideal memory does not improve this poor IPC performance, indicating that the cause must be either data dependencies or poor branch prediction on the complex control flow code they contain. Finally, BLAST obtains moderate IPC results, but only in the presence of an ideal memory configuration. When we use small cache sizes, the performance drops significantly (52% slowdown from ideal caches to 32k L1 caches).

D. Memory hierarchy behavior

Figure 5 shows how the cache size has an effect on (a) the cache miss rate, and (b) the IPC of each application. We simulate L1 cache sizes ranging from 1K to 2M, always using a 2M L2 cache and a 4-way processor configuration. The results show that all applications except SSEARCH34 require at least 4K caches to fit their working set. BLAST is the application with the highest miss rate, for any but the smallest cache sizes. While all others already have a negligible number of misses on a 32K cache, BLAST still has close to

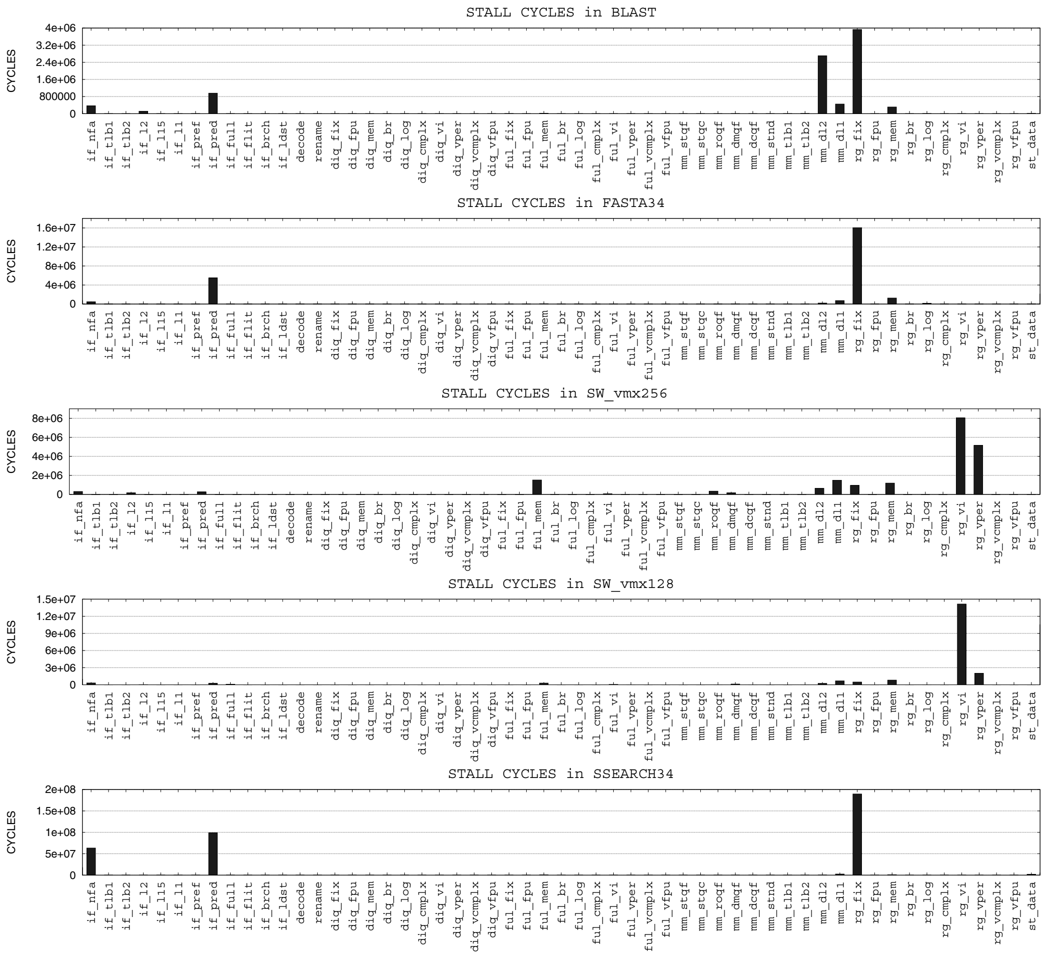

Fig. 2. Histogram of traumas in configuration 4-way, IL1 32K, DL1 32K, L2 1M and real branch predictor

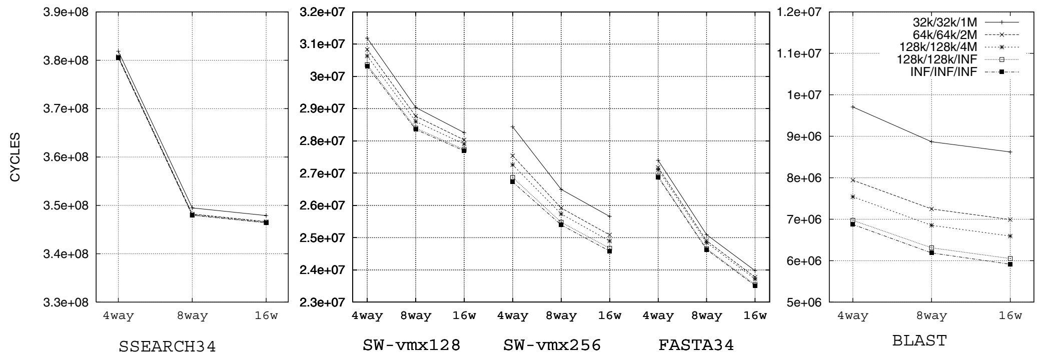

Fig. 3. Cycles vs memory configuration

4% misses. However, as we observe from the IPC figure, it is the SIMD codes that benefit most from the fact that the working set fits in cache, exhibiting more than a 2x speedup as we grow the cache beyond 8K.

Figure 6 shows how the cache associativity impacts the miss ratio and IPC. We set the cache size at 32K and change the associativity from direct-mapped to 8-way set associative. BLAST is the only application that shows a significant decrease of misses when using set associatve caches, as it is the only one that can not fit its working set in the 32K cache. However, as can be observed in the IPC figure, the miss rate reduction obtained does not translate to improvements in the application performance. Clearly, a 32K cache is not enough for BLAST, no matter what associativity is used.

Figure 7 shows the impact of the L1 cache latency on IPC for each application. We fix the IL1/DL1/L2 cache sizes at 32K/32K/1M and the associativity at 1/2/8 set respectively, and change the hit latency from 1 to 10 cycles. The results

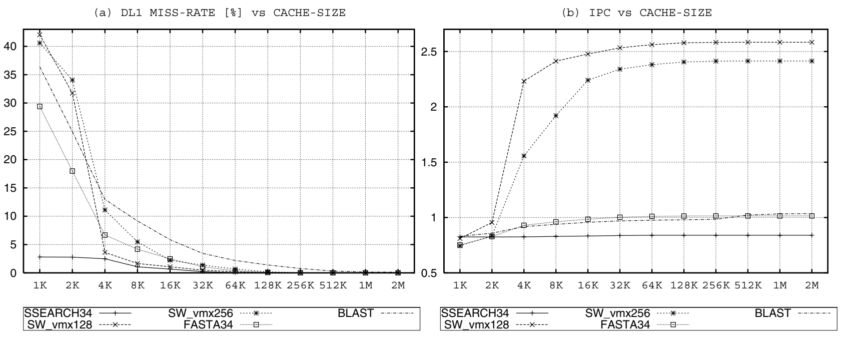

Fig. 5. DL1 cache miss rate vs cache size

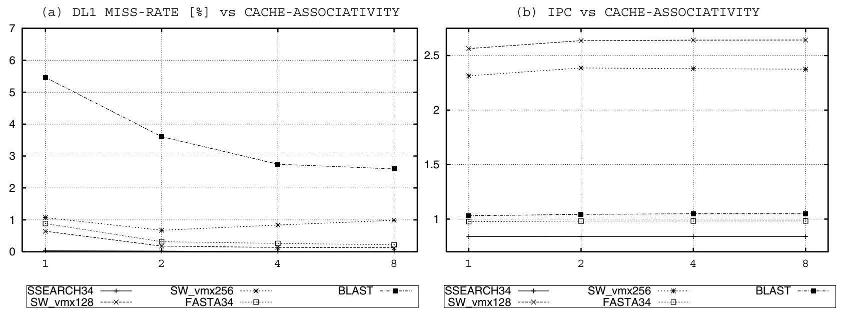

Fig. 6. DL1 cache miss rate vs cache associativity

show that the SIMD codes are the most sensitive to the cache latency, since they were the most compute-bound. Since all load operations (SIMD and scalar) go to the L1 cache, if we increase the time it takes to bring data to the functional units, the whole dependency chain is delayed.

Finally, since SIMD codes are the most sensitive to memory latency, we perform an experiment where we only increase the latency of the 256-bit vector load operations by 1 extra cycle. This is a more fair comparison to the 128-bit version, as it reflects the scenario where both codes have the same load/store bandwidth (the double width is pipelined, and so it requires an addiitonal cycle). The results in Figure 8 show that even with the added cycle latency, the 256-bit version is still 5% faster than the 128-bit version, but the performance advantage has diminished significantly. As we observed in Section V-B, these applications are compute bound, and have long dependency chains, which prevents the 256-bit registers from exploiting twice the amount of DLP.

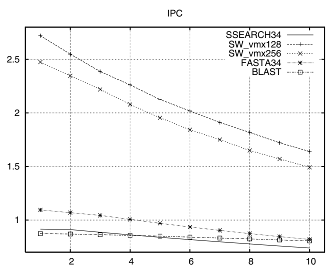

Fig. 7. L1 latency behavior

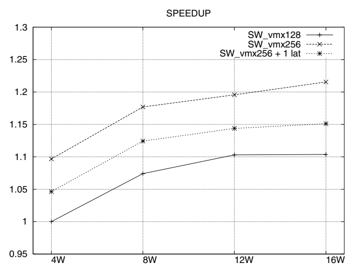

Fig. 8. Speed-up vs L1 latency

E. Branch Prediction Study

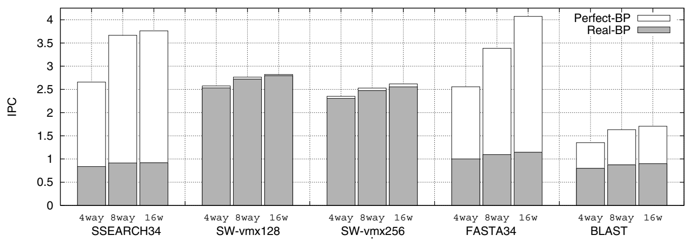

Fig. 9. Perfect and real branch predictor

Figure 9 shows the impact of using a real branch predictor compared to an ideal branch predictor. The real branch predictor is described in table VI. There is an interesting correlation between the number of control flow instructions shown in Figure 1 and the impact of the branch prediction mechanism. As was to be expected given the low number of branches in SIMD instructions, the branch predictor has a negligible impact on these applications. On the other hand, SSEARCH34, FASTA, and BLAST, which have a large number of complex `if-then-else` structures, the branch predictor plays a critical role. BLAST and FASTA benefit from their heuristic approach, which reduces the number of required computations to provide the results, and SSEARCH also reduces the amount of computation avoiding it when not necessary, however, in both cases they become heavily dependent on the accuracy of the branch predictor.

Figure 11 compares the performance of different branch prediction strategies as a function of the predictor size. The

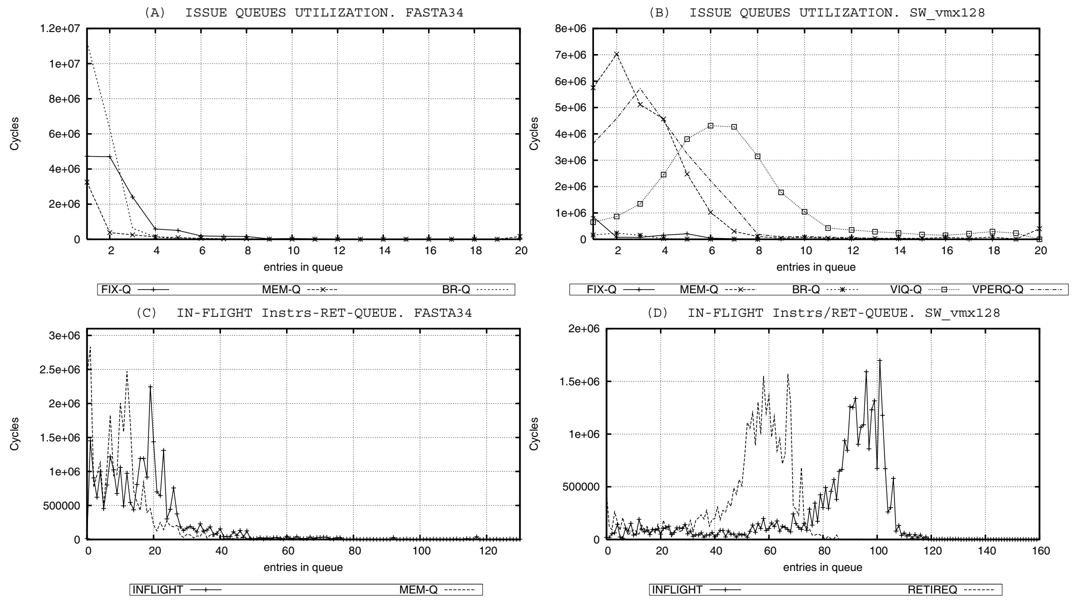

Fig. 10. Utilization of issue queues for FASTA an SW_vmx128

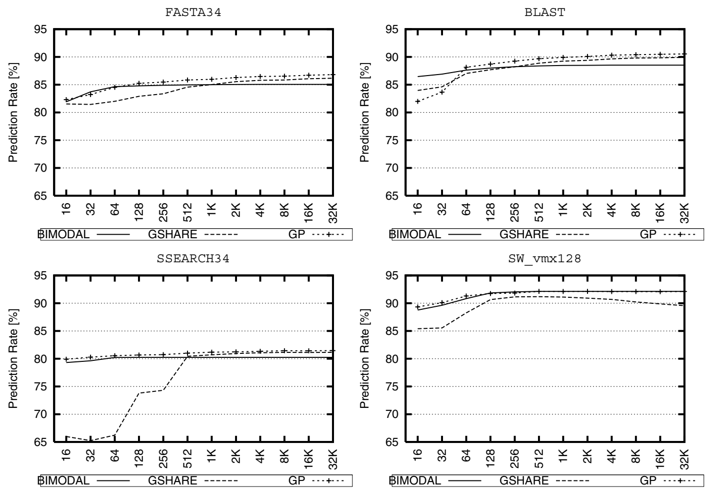

Fig. 11. Branch predictor accuracy

results show that the low prediction accuracy is neither due to the branch predictor size, since near optium accuracy is achieved beyond 512 bytes, not to the prediction strategy, since all 3 compared predictor achieve similar results. We conclude that the low prediction accuracy observed is mainly due to the unpredictable nature of heuristic approaches in BLAST and FASTA, and the data-dependent nature of the computation avoidance in SSEARCH.

F. Resource utilization

Figure 10a,b shows the usage of the different instruction issue queues (integer, load/store, branch) for the FASTA and SW _vmx128 applications. Figure 10c,d shows the number of in-flight instructions, and pending memory operations. We do not show the remaining applications due to space limitations. Also, we use the 32K/32K/1M/ memory configurations and the 4-way processor configuration. The results for FASTA show that the queues are mostly empty, or with little occupation. BLAST and SSEARCH34 (not shown) follow the same trend. This is due to the poor branch prediction accuracy obtained, which causes a high number of pipeline flushes that can be observed in the low number of in-flight instructions, and limits the ILP that can be exploited.

On the other hand, the results for SW_vmw128 show that there is a significant usage of the issue queues, specially the vector integer queue. The high prediction accuracy allows the processor to maintain a high number of in-flight instructions, and exploit ILP. This confirms what we have observed before, that the performance limit is caused by data dependencies and contention on the vector functional units.

VI. Conclusions and Future Work

In this work we have adopted a simulation-based approach to perform a detailed workload characterization of typical applications for sequence comparison and alignment task. We have analyzed architectural and micro-architectural aspects like pipeline configurations, issue widths, functional unit mixes, memory hierarchy, branch prediction and resource utilization.

Although memory accesses are a significant component in all heuristic strategies for sequence comparison, their impact on performance differs significantly between applications. Whereas the memory system is critical for the BLAST application, it has a minimal impact in FASTA, where branch prediction becomes the major responsible for performance degradation.

In the case of the non-parallel implementation of SW algorithm (SSEARCH34), results show that branch prediction has the most important impact in the performance degradation. This situation is basically the result of the algorithm optimizations applied in the implementation, which include many `if-then-else` statements in the source code with a non-simple behavior pattern.

For the parallel implementations of the SW algorithm, we were interested in the evaluation of the impact of SIMD alternatives on performance improvement. To reach this goal, we have used the best reported SIMD implementation of the SW, which is included as an optional tool in the SSEARCH34 toolset. This implementation is based on the use of Altivec SIMD extension presented in current IBM PowerPC processors. In order to explore the performance gains using wider registers, we have also written a new version of this SW implementation using 256-bit registers. The results show that both 128- and 256-bits SIMD implementations have a big component of memory access (scalar and SIMD access). However the impact of the memory system in performance is low. This is basically due to the fact that most of these accesses are contiguous in memory and then, they can take advantage of the data locality. On the other hand, the amount of branch instructions and their prediction are not decisive in the performance of these SW implementations. In these implementations, the critical part is the execution of vector integer and vector permute instructions.

Additionally, the speed-up obtained using wider SIMD register is not as important as it was to be expected. The instruction reduction using 256-bit SIMD (18% on average) does not translate directly into time reduction (9% on average). As we discussed in Section V-B this is mainly due to the true data dependencies and contention on the vector integer and permute functional units.

Future work includes the analysis and performance characterization of other important biological applications, such as multiple sequences analysis, genome-level alignment, phylogenetic analysis, molecular dynamics and so on. These studies will set the bases to define architectural and micro-architectural optimizations in order to speedup the execution of this emerging workload in future high performance processors.

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] Sgi bioinformatics performance report. http://www.sgi.com/industries/sciences/.

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

[5] P. Agarwal and D. J. States. Comparative accuracy of methods for protein sequence similarity search. *BIOINF: Bioinformatics*, 14, 1998.

[6] K. Albayraktaroglu, A. Jaleel, M. Xue Wu, Franklin, B.Jacob, C. Tseng, and D. Yeung. Biobench: A benchmark suite of bioinformatics applications. *Proc. 2005 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS 2005), Austin TX, March 2005.*, March 2005.

[7] 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.

[8] S. F. Altschul, T.L, A. Schffer, J. Zhang, Z. Zhang, W. Miller, and D. Lipman. Gapped blast and psi-blast: a new generation of protein database serach programs. *Nucleic acids research*, 25:3389–3402, 1997.

[9] D. A. Bader, Y. Li, T. Li, and V. Sachdeva. Bioperlf: A benchmark suite to evaluate high-performance computer architecture on bioinformatics applications. *Proc. 2005 IEEE International Symposium on Workload Characterization (IISWC 2005), Austin TX*, Oct 2005.

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

[11] A. Das, J. Lu, H. Chen, J. Kim, P.-C. Yew, W.-C. Hsu, and D.-Y. Chen. Performance of runtime optimization on blast. In *CGO '05: Proceedings of the international symposium on Code generation and optimization*, pages 86–96, Washington, DC, USA, 2005. IEEE Computer Society.

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

[13] P. Green. Swat. www.genome.washington.edu/uwgc/analysistools/swat.htm.

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

[15] Y. Li, T. L. T. Kahveci, and J. Fortes. Workload characterization of bioinformatic applications. *IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2005.

[16] J. H. Moreno, M. Moudgill, J.-D. Wellman, P. Bose, and L. Trevillyan. Trace-driven performance exploration of a powerpc 601 oltp workload on wide superscalar processors. Technical Report RC 20962, IBM Research Report, August 1997.

[17] J.-D. M. J. Moudgill, M. Wellman. Environment for powerpc microarchitecture exploration. *IEEE Micro, 19*, 1999.

[18] J.-D. M. J. Moudgill, M. Wellman. Validation of turandot, a fast processor model formicroarchitecture exploration. *Performance, Computing and Communications Conference,IPCCC. IEEE International*, 1999.

[19] 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.

[20] 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.

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

[22] Rognes and Seeberg. Six-fold speed-up of smith-waterman sequence database searches using parallel processing on common microprocessors. *BIOINF: Bioinformatics*, 16, 2000.

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

[24] F. Sanchez, E. Salami, A. Ramirez, and M. Valero. Parallel processing in biological sequence comparison using general purpose processors. *Proceedings of the IEEE International Symposium on Workload Characterization. IISWC 2005.*, 2005.

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

[26] B. Schmidt, H. Schroder, and M. Schimmler. Massively parallel solutions for molecular sequence analysis. In *IPDPS 02: Proceedings of the 16th International Parallel and Distributed Processing Symposium*, page 201. IEEE Computer Society, 2002.

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

[28] E. Shpaer, M. Robinson, D. Yee, J. Candlin, R. Mines, and T. Hunkapiller. Sensitivity and selectivity in protein similarity searches: A comparison of smith-waterman in hardware to blast and fasta. *Genomics*, 38:179–191, 1996.

[29] S. Smith and J. Frenzel. Bioinformatics application of a scalable supercomputer-on-chip architecture. 1:385–391, 2003. Proceedings of the International Conference on Parallel and Distributed Processing Techniques.

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

[31] A. Wozniak. Using video-oriented instructions to speed up sequence comparison. *Computer Applications in the Biosciences*, 13(1), 1997.

[32] T. K. Yap, O. Frieder, and R. L. Martino. Parallel computation in biological sequence analysis. *IEEE Trans. Parallel Distrib. Syst.*, 9(3):283–294, 1998.