Parametrizing Multicore Architectures for Multiple Sequence Alignment

Sebastian Isaza
Computer Engineering Laboratory, Delft University of Technology, The Netherlands
s.isazaramirez@tudelft.nl

Friman Sanchez
Computer Architecture Department, Technical University of Catalonia, Spain
fsanchez@ac.upc.es

Felipe Cabarcas
Technical University of Catalonia and Barcelona Supercomputing Center, Spain
felipe.cabarcas@bsc.es

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

Georgi Gaydadjiev
Computer Engineering Laboratory, Delft University of Technology, The Netherlands
g.n.gaydadjiev@tudelft.nl

ABSTRACT

Sequence alignment is one of the fundamental tasks in bioinformatics. Due to the exponential growth of biological data and the computational complexity of the algorithms used, high performance computing systems are required. Although multicore architectures have the potential of exploiting the task-level parallelism found in these workloads, efficiently harnessing systems with hundreds of cores requires deep understanding of the applications and the architecture. When incorporating large numbers of cores, performance scalability will likely saturate shared hardware resources like buses and memories. In this paper we evaluate the performance impact of various configurations of an accelerator-based multicore architecture with the aim of revealing and quantifying the bottlenecks. Then, we compare against a multicore using general-purpose processors and discuss the performance gap. Our target application is ClustalW, one of the most popular programs for Multiple Sequence Alignment. Different input data sets are characterized and we show how they influence performance. Simulation results show that due to the high computation-to-communication ratio and the transfer of data in large chunks, memory latency is well tolerated. However, bandwidth is critical to achieving maximum performance. Using a 32KB cache configuration with 4 banks can capture most of the memory traffic and therefore avoid expensive off-chip transactions. On the other hand, using a hardware queue for the tasks synchronization allows us to handle a large number of cores. Finally, we show that using a simple load balancing strategy, we can increase performance of general-purpose cores by 28%.

Categories and Subject Descriptors

C.1.3 [**Processor Architectures**]: Other Architecture Styles —*Heterogeneous (hybrid) systems*; C.1.4 [**Processor Architectures**]: Parallel Architectures; C.4 [**Performance of Systems**]: [Design studies]; J.3 [**Life and Medical Sciences**]: [Biology and genetics]

General Terms

Algorithms, Measurement, Performance

Keywords

multicore architectures, hardware accelerators, ClustalW, multiple sequence alignment, bioinformatics

1. INTRODUCTION

Bioinformatics is the discipline that applies computational techniques to solve biology problems [15]. Due to the huge and steadily growing amount of data involved, high performance computers are used to assist biologists. In recent years, many algorithms and software tools have been proposed to analyze large amounts of biological data. Multiple Sequence Alignment (MSA) is one of the essential tasks in bioinformatics. When having a group of DNA sequences an MSA can reveal sections that are common to most of the sequences in the group. This information has many uses, for example, it can indicate evolutionary relationships between species or, in the case of protein analysis, it may determine a protein's function based on other known functionalities in the reference group.

ClustalW [29] is a widely used application to perform multiple sequence alignment. Unlike pairwise sequence alignment tools like BLAST [5] and FASTA [2], ClustalW aligns a set of sequences together as a group to produce an MSA. Given the intractability of computing an optimal MSA, ClustalW uses a heuristic known as *progressive alignment.* The program is divided in three main phases, namely: Pairwise Alignment (PW), Guide Tree (GT) and Progressive Alignment (PA). This paper focuses on ClustalW-PW, the most time consuming application phase (see Section 3).

Improvements in sequencing technologies have led to an exponential growth of biological databases. Figure 1 shows

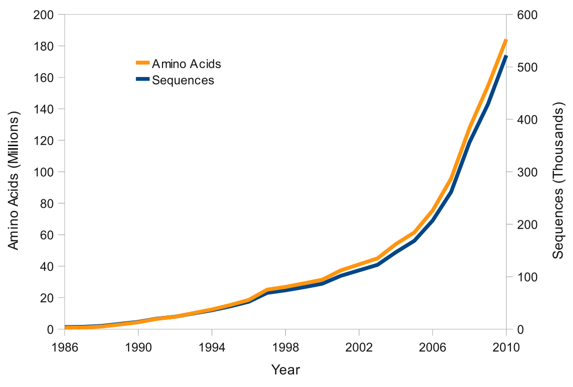

Figure 1: SwissProt database growth.

the SwissProt [8] database size over the years, currently containing 184,241,293 amino acids in 522,019 protein sequences. Gene databases such us the EMBL-Bank [9] contain significantly more sequences, i.e. 195,241,608 in 1TB of data. As these databases grow, analyses become more ambitious and the need for computational power increases. On the other hand, the use of expensive multiprocessor machines to meet the performance demands has two main limitations: only few can afford buying them and their energy consumption is extremely high. In the search for efficient solutions, previous studies have proposed accelerating ClustalW using off-the-shelf FPGAs, GPUs and chip multiprocessors (CMP). Most of the reported results show significant performance improvements with expected energy and cost reductions. However, future systems are very likely to contain large numbers of cores that impose pressure on the shared hardware resources like buses, caches and memories. Although the trend to steadily increase the number of cores seems to be certain, the characteristics of the cores that should be used is still an open issue. While the general-purpose stream is considering application-specific improvements to achieve efficiency, the embedded world leans towards a more general-purpose style to increase flexibility.

In this paper, we investigate various multicore architecture configurations to study performance scalability. We map ClustalW onto a multicore that uses either general-purpose processors or application-specific accelerators. This allows us to project state-of-the-art processors and accelerators performance into scaled future systems and determine the requirements in terms of shared hardware resources. Although the fundamental pros and cons of the two approaches are known (flexibility and ease of programming vs. higher throughput and energy efficiency), this paper aims at investigating the performance consequences of using each of them and analyzing the sensitivity to various hardware components (e.g. memory, cache, bus). Experimental results show that an accelerator-based multicore can still achieve high efficiency when using hundreds of cores. Furthermore, because those accelerators can be run at lower frequencies, they will be able to stay within the allowed power budget (and will dramatically reduce energy consumption as well).

In any case, performance will be largely determined by the available parallelism in the application, which in some cases may depend on the input data characteristics. In this paper we take input data sets from BioPerf [11] and characterize them against three additional input sets that we build ourselves. Conclusions are made on the performance implications when exploiting the application's parallelism.

The main contributions of this paper are:

- The characterization of various types of input data sets and their impact on performance when exploiting parallelism;
- The quantification of bandwidth and synchronization requirements;
- The performance scalability comparison of a multicore based on general-purpose processors against one based on application-specific accelerators;
- The load balancing improvement for high number of cores processing a small input data set.

The use of a hybrid high-level/cycle-accurate simulator setup allows us to abstract out the parts of the system that do not affect our analysis. This, in turn, enables high speed simulations enabling the modeling of systems with hundreds of cores. The content of this paper is organized as follows. Section 2 gives an overview of the related work. Section 3 describes the target application and its parallelization. In Section 4, the simulation setup and the multicore architecture are explained. Finally, Section 5 analyzes the results and Section 6 draws some conclusions.

2. RELATED WORK

Various works are found in literature with respect to ClustalW parallelizations. They have used different implementation platforms: from multiprocessor machines and GPUs to FPGAs and custom made ASICs. In [14, 13], results of distributed memory parallelizations (using *MPI*) of ClustalW are presented. The PW and PA stages were parallelized targeting computer clusters and reported speedups without any analysis. In [14], the performance scalability is fairly good up to 8 processors while in [13] it lasts up to 16 processors. The main reason for this difference is that the input sequences used in [13] are about 4 times longer on average. This results in the PW phase taking much longer than the other phases. As a consequence and since PW is the most scalable ClustalW phase, the overall performance scalability improves. The main performance bottleneck is the long computation time needed for each PW alignment because of the use of sequential codes running on a rather old processor (Pentium III). More recently, other works [27, 30, 19, 20] have studied ClustalW's performance on the Cell BE [21]. Results show that despite of a non-trivial programming model, specific Cell BE features like the Local Stores (LS) and the SIMD processing can achieve significant performance improvements for bioinformatics applications, including ClustalW. Besides, ClustalW-PW is shown to scale well up to the 8 SPE cores available in a Cell BE. Our work aims to extend the analysis of parallelism in the bioinformatics domain to future architectures containing hundreds of cores. While previous works report that ClustalW-PW has no scalability issues (for tens of cores), we show that when using hundreds of processors (in a CMP), performance can be saturated.

On the other hand, there have also been efforts on optimizing sequence alignment kernels at a finer grain on software

Table 1: Performance comparison of various SW and HW implementations of Smith-Waterman.

Reference	Performance	Speedup
Cell BE [19]	0.4GCUPS	1×
Cell BE [18]	2GCUPS	5×
GPU [23, 22]	5GCUPS	2.5×
FPGA [12]	14GCUPS	35×
CGRA [23]	76.8GCUPS	192×
FPGA [7]	688GCUPS	1720×

and hardware. In [19], the SIMD capabilities of the Cell BE are exploited, achieving 0.4GCUPS per SPE. GCUPS (Giga (10^9) Cell Updates Per Second) is a common metric used for Smith-Waterman implementations that allows us to compare with other reported results. An optimized version that avoids misaligned memory accesses is presented in [18, 17], achieving 2GCUPS. GPU implementations [23, 22] have reported 5GCUPS and FPGA designs [12] 14GCUPS. Liu et.al. presented in [23] a careful comparison of his coarse-grain reconfigurable architecture (CGRA) performance (76.8 GCUPS) with the mentioned GPU and FPGA results. Lastly, an even higher performance (688GCUPS) has been recently reported by a commercial FPGA-based product [7], likely using more hardware resources though. In this paper we study a multicore architecture where each core has the performance of either a general-purpose processor [19, 18, 17] or an application-specific accelerator [23, 22, 12, 7]. Table 1 shows the difference in throughput of various implementations of the kernel targeted in this paper.

Results reported in most of the previous work focus on describing application mapping decisions onto commercial hardware and present speedups compared to the competitor (e.g. GPU vs. Cell BE). However, all lack in-depth analyses of hardware resource utilization that are needed in order to find the real bottlenecks. We instead, aim at not only efficiently mapping ClustalW to a scaled multicore but also at finding the true bottlenecks that would limit performance scalability. This analysis and the related findings form the main contribution of this paper. We envision this knowledge as very useful for designers of future multicore systems targeting applications from the bioinformatics domain.

The SARC architecture used in our study is presented in [25], where results are reported for the comparison of one pair of very long sequences, using FASTA-SSEARCH34 [24, 2]. We complement those experiments by looking at a different problem: the multiple sequence alignment with ClustalW [29]. Our results show that although ClustalW-PW also tolerates high memory latencies (similar to the observations in [25]), the bandwidth requirements are quite different. Moreover, we study a manycore based on ASIC accelerators while previous work [25] used ASIPs instead.

3. APPLICATION'S DESCRIPTION

ClustalW [29] is a widely used software (above 30,000 citations) to perform MSA, developed by the European Bioinformatics Institute. Its source code is freely available and various web servers exist [1, 4] where users can submit their alignment jobs. ClustalW takes as input a file containing a set of DNA or protein sequences that are expected to have similarities. The program output (the MSA) contains the same set of sequences but with inserted gaps that place similar regions from different sequences, one on top of each other. In order to compute an MSA, ClustalW uses a heuristic known as *progressive alignment*. As mentioned before, ClustalW implements this heuristic in three steps: PW, GT and PA. PW is responsible for computing similarity scores for all sequence pairs and GT uses these scores to cluster sequences in a tree structure (a phylogenetic tree is produced). PA computes the final alignment by walking the tree, adding (aligning) one sequence (or profile) to the MSA at a time. In PW, a similarity score is computed for all sequence pairs. This score is computed by aligning the two sequences (without actually recovering the alignment but just the score), based on the Smith-Waterman [28] algorithm. In [16], the computational complexity of the different ClustalW phases is computed. In Table 2, profiling results of the original (sequential) ClustalW code are compared with its theoretical complexity, where N is the number of sequences and L is the average sequence length. We see that even for a large[1] N, PW takes most of the time in a uni-processor execution. For an input of N sequences, PW must compute $N(N-1)/2$ alignments, called tasks here. On the other hand, GT only grows significantly for the case of many short input sequences. For most of the cases, including the test inputs used in our experiments, GT time is negligible.

3.1 ClustalW-PW Parallelization

Tasks in PW are independent of each other which enables the parallel use of, in principle, as many processors as the available number of tasks. We have parallelized ClustalW-PW following a master-worker scheme widely accepted. One master processor is in charge of submitting tasks to a queue by taking pairs of sequences as they come from the database. Each job queue entry contains an ID and a memory pointer to the task parameters: the pointers to the input sequences, their lengths and the gap penalties. Worker processors poll the queue until they obtain a job and then proceed to fetch the input data from main memory (using DMA). Once data is loaded in the Scratchpad Memory (SPM), a worker core performs the computations and writes back results to main memory on completion (see Section 4.2 for details on the architecture). Most of the communication traffic in the system will be due to workers fetching input data. During the write-back of results, only a score needs to be transferred, which is negligible compared to the input sequences. The master must wait for the completion of all the scheduled jobs before the next phase in ClustalW (GT) can start.

Table 2 shows the profiling results for various input data sets and the corresponding theoretical speedup limit for the parallelization of PW, as obtained from applying Amdahl's law [10]. Although PA can be parallelized to some extent as well [13, 23], its tasks have dependencies that limit the achievable degree of parallelization. Hence, the total application's speedup will be largely determined by the characteristics of the input dataset.

4. EXPERIMENTAL METHODOLOGY

For our analysis we have used the *TaskSim* simulator

[1]Although databases contain millions of sequences, current MSA programs are not reliable for aligning more than a few thousands.

Table 2: Characterization of various ClustalW input sets.

		Profiling $N(L)$*							
		P13569		P10091		NM_000249	BioPerf-A	BioPerf-B	BioPerf-C
	$O(Time)$	500(940)	318(1,035)	**500(355)**	318(355)	83(1,641)	50(52)	66(1,083)	318(1,045)
PW	$O(N^2L^2)$	98.5%	97.7%	98.5%	97.9%	90.5%	82%	61%	82%
GT	$O(N^4)$	Negl.	Negl.	Negl.	Negl.	Negl.	Negl.	Negl.	Negl.
PA	$O(N^3 + NL^2)$	1.5%	2.3%	1.5%	2.1%	9.5%	18%	39%	18%
Score		8,809,293	16,144,863	9,785,331	12,803,740	9,778,088	120,625	1,298,083	-4,528,287
Norm. Std. Dev.		0.44	0.41	0.04	0.05	0.49	0.15	0.71	0.75
Max. Speedup		66.67×	43.48×	66.67×	47.62×	10.53×	5.56×	2.56×	5.56×

* N is the number of sequences in a set and L is the average length.

[25, 26] that models a parametrizable multicore architecture as sketched in Figure 3 (see Section 4.2). The simulator uses a hybrid trace-driven high-level/cycle-accurate technique, that is, some components are simulated very accurately while others are abstracted out, depending on the aspects one wants to analyze. The simulator input are trace files that describe the application functioning by providing three types of information: data transfers, computational bursts and synchronization signals. To obtain these traces we have manually instrumented a ClustalW (v.1.83) port to the Cell BE [21]. Thereafter the instrumented code was run on an IBM QS21 Blade, running at 3.2GHz with 4GB of RAM. The code has been compiled with GCC4.1.1 and -O3 flag. Only the master and one worker processor (that is, a PPE and a single SPE in Cell BE) were used to generate the trace. This in order to create a single pool of tasks that can be dynamically scheduled to any of the simulated worker processors.

Since the processor's load and store operations are served by the SPM (without polluting any caches), the execution time of processing a given task (excluding synchronizations or DMA waits) is independent of the activity in the rest of the system. Therefore, the simulator does not "waste" time on accurately simulating the details of the internal execution of a core. On the other hand, TaskSim does simulate in great detail (cycle-accurate) the data transfers (that is, buses contention, caches and memory) and the synchronization signals exchanged among processors.

To ensure our experimental simulations' accuracy, we ran ClustalW-PW on the Cell BE Blade using 8 SPEs and compared the execution time with that obtained from simulating a system with equal configuration. This allowed accuracy verification when simulating multiple cores from a trace generated using only one. Results showed that the simulation error is under 2%, considered adequate for the purpose of our study.

The simulation output also contains traces (like the ones graphically shown in Section 5.2) that we analyze using the performance visualization tool Paraver [6].

4.1 Input Data Sets

Various inputs with different characteristics are used in our simulations (see Table 2). First, we take the three input data sets (A, B and C) provided in BioPerf [11] that have been used in some of the related work [27, 30, 19, 20]. Although not clearly stated by the BioPerf authors, the sequences seem to be randomly selected from the databases. As a result, sequences are highly dissimilar as confirmed in the low alignment scores obtained with ClustalW. Table 2 also shows the sequence length standard deviation (normalized with the average length) as a measure of how different sequence lengths are within each input set.

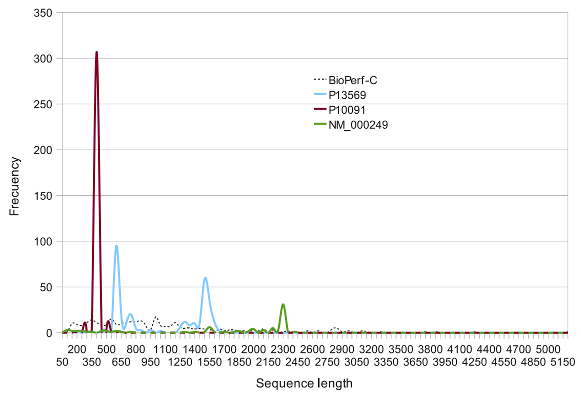

Figure 2: Sequence length histogram for different input test sets.

A different methodology is followed to generate additional (more realistic) test sets, as reported in previous work [13]. In our study we use protein P10091 CYSA_MARPO (365 amino acids) as query to perform a BLAST search against the SwissProt database. Since ClustalW is commonly used to align sequences that are expected to have some similarity or evolutionary relationship, we take the first 500 protein hits returned by the search to use as input for the MSA. We apply the same methodology to protein P13569 (1480 amino acids) and to DNA sequence NM_000249 (2662 nucleotides) with the GenBank database.

Figure 2 shows a sequence length histogram of input data sets analyzed. The taller the peaks in a given set, the more homogeneous the sequences' lengths are. Data from BioPerf is clearly the most heterogeneous in sequences' length and, as confirmed by the alignment scores from Table 2, also dissimilar in the biological sense. By looking at Table 2 and Figure 2, a correlation can be observed between alignment scores, length standard deviation and profiling results. Test sets with a low score and high length variation spend less time on the most parallel part of the program, i.e. PW. The reason for this phenomena has to do with the calculation of boundaries in the recursive structure of the PA alignment algorithm, which is out of the scope of this paper. Although in a uniprocessor scenario this does not matter, in a multicore it has great performance implications as the parallelizable portion is smaller. The last row in Table 2 shows the maximum application speedup achievable if PW times is reduced to zero.

In our attempt to simulate a realistic scenario, we use 500 sequences corresponding to protein P10091 with lengths most similar to the average sequence length of the SwissProt database (355 amino acids). In PW, a set of 500 sequences produces 124,750 coarse grain tasks that can be run in parallel, which is enough to stress a system with up to 1024 cores as it is the case for our study. Only for the experiments in Section 5.2 we use sequences from BioPerf in order to show an issue that arises in such execution scenario.

4.2 Multicore Architecture

The architecture considered, Figure 3, is a clustered (or tiled) multicore interconnected with a system of hierarchical buses. Clusters of 8 (P)rocessors (similar to a single Cell BE [21]) are grouped together through a local data bus (LDB) and all clusters are connected by the global data bus (GDB). The memory interface controllers (MIC) to access off-chip DRAM memory and the L2 cache banks are also connected to the GDB and shared by all processors.

It is a heterogeneous architecture given that different processor types coexist. The (M)aster processor is a powerful out-of-order superscalar processor that can efficiently run the operating system and the applications control sections. It accesses memory through a private L1 instruction/data cache and is connected to the rest of the system through the GDB. (W)orker processors perform data processing, that is, the bulk work of the targeted applications. Each worker can access three different memory types: its own SPM, the global shared memory and the other worker's SPM. For the last two cases, the memory controller (MC) DMA engine is used for the data transfers, decoupling them from the worker processor execution.

Each LDB is a bus with a ring topology, able to transmit 8 bytes per cycle at 3.2GHz (that is, 25.6 GB/s). Since the GDB is expected to handle more traffic, its bandwidth is increased by adding more rings. Every ring provides an independent communication channel between any pair of components in the system. Main memory is composed of several off-chip DRAMs, controlled by one or more MICs each providing up to 25.6GB/s. The DRAMs are DDR3-1600 with a peak bandwidth of 12.8GB/s, two of them are connected to every MIC. Fine-grain interleaving at the MIC level and then of DRAM channels is used, that is, consecutive memory lines change MICs and DRAMs so that large DMA requests of consecutive memory ranges can be processed in parallel accessing several DRAMs in parallel.

The L2 cache is organized in banks, each able to provide 25.6GB/s. As in the MICs case, fine-grain interleaving in accessing L2 banks provides high bandwidth for accesses to consecutive addresses. Since data transfers in this architecture mostly rely on DMAs, fine-grain interleaving enables the cache banks to serve multiple parts of a single DMA request in parallel. As a consequence, the effective bandwidth observed by the request can be higher than that of a single bank. The L2 is meant to take advantage of data reuse among different workers. However, since the data set of one task easily fits on the worker's SPM, there is no need to have per worker L1 caches. Having them would imply a lot of extra area, complexity to support coherency and power.

The synchronization queue (SQ) has two hardware FIFOs implemented in an independent memory module connected to the GDB. One queue is for the master to submit tasks and the other for the workers to report task completion.

The intention of the experiments presented here is to evaluate the sensitivity of the application's performance to various critical architectural parameters and to determine the minimum configuration that achieves the maximum speedup.

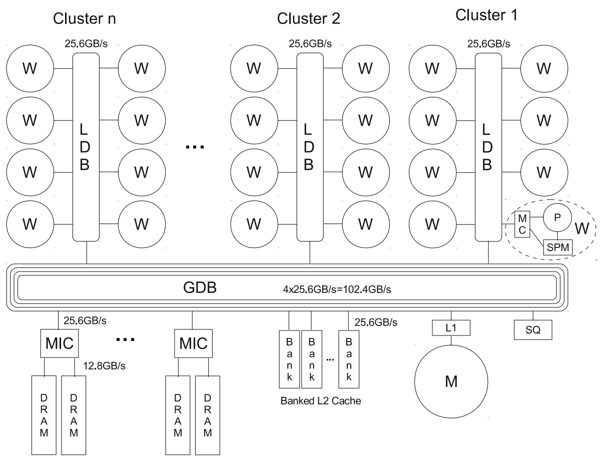

Figure 3: Simulated multicore architecture.

Table 3: List of parameter names and value ranges.

Parameter name	Range of values
Number of workers	1,2,4... 1024
Workers' throughput	1×,100×
DRAM latency (cycles)	1,64,128,256,512,1024,2048,4096
Number of MICs	1,2,4,8,16,32
L2 cache size	0KB,64KB,128KB,512KB,1MB
L2 cache banks	1,2,4,8
GDB rings	1,2,4,8,16,INF
SQ latency (cycles)	1,32,128,256,512

Table 3 lists the parameters along with the value ranges used in the simulations. For every figure shown in Section 5, only one parameter is varied. All other parameters are set to appropriate values so as to emphasize the effect of the one being analyzed.

4.3 Workers

One special parameter that needs discussion is the *workers' throughput*. One of the key aspects that determines system performance is obviously the data processing power of an individual worker. However, increasing workers' throughput does not always result in an equal increase in application's performance. Shared hardware resources like the buses or the memory may become the bottleneck.

As explained before, the hot-spot in ClustalW-PW uses the Smith-Waterman algorithm. Because that kernel has been the focus of many research papers and even commercial products in recent years, it is possible for us to compile an overview of the wide range of achievable single-core performances. These are described in Section 2 and Table 1. We compare simulation results of a multicore based on general-purpose processors (1× throughput) with one using accelerators (100× throughput). These accelerators are dedicated hardware structures that implement a *systolic array* architecture to exploit the parallelism available in the Smith-Waterman processing [23]. Based on Table 1 we use 100× as a conservative value. We run experiments with different architectural parameters in order to find out the

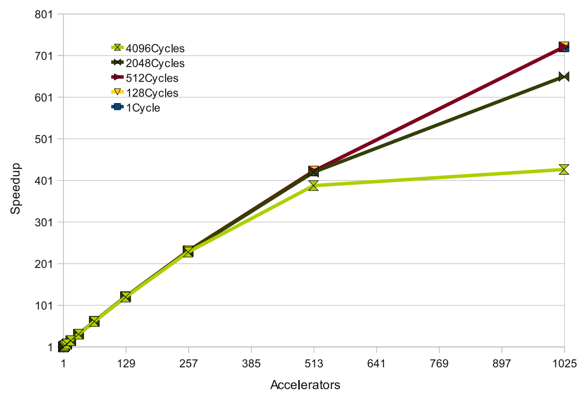

Figure 4: Sensitivity to memory latency. Other parameters are: 32 MICs, No cache, INF Rings.

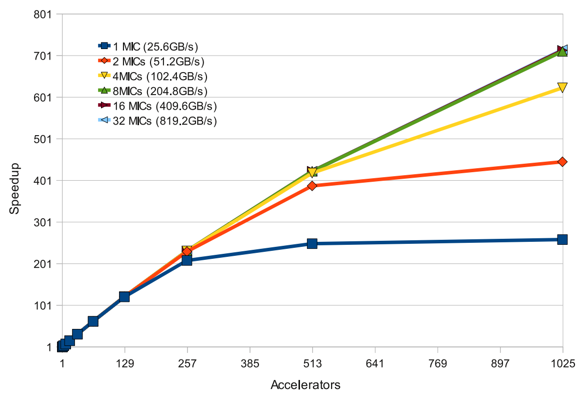

Figure 5: Sensitivity to memory bandwidth. Other parameters are: No cache, INF Rings.

minimal configuration that achieves maximum performance using 100× accelerators.

5. SIMULATION RESULTS

In this section we discuss our simulation results. First, we study the sensitivity of various architectural parameters for the accelerator-based multicore. Then, we look at the load balancing problem of a general-purpose-based many-core that arises when the input set is relatively small. Finally, we run a set of simulations where we compare the performance achieved by the accelerators with that of the general-purpose cores.

5.1 Accelerator-based Multicore

Tasks in ClustalW-PW are very coarse and have a high computation-to-communication ratio, that is, a large amount of processing is done on a relatively small data set. Despite of this, the use of accelerators manages to lower this ratio and the shared resources (e.g. memory, buses, SQ) requirements increase. In this section we analyze the impact of the architectural parameters from Table 3, using up to 1024 accelerators (100×). In Figures 4 - 9, the baseline is a single accelerator core.

Figure 4 shows the impact of varying the latency to access main memory. This simulation has an artificial setup in which we replace the shared cache and the DDR memory system with an ideal conflict-free memory having a configurable latency between 1 cycle and 4096 cycles. For reference purposes, notice that the average DDR3 latency is between 150 and 250 cycles. On top of the raw memory latency, the time to traverse the buses has to be taken into account as well. Results show that the application tolerates well even large latencies. Only by increasing latency beyond 2048 cycles, performance degradation is observed. This result is expected given that data transfers are mostly done in large chunks (close to 1KB) to fetch the sequences to the accelerators' SPMs.

Figure 5 shows the sensitivity to memory bandwidth. We vary the number of MICs between 1 and 16. Every MIC provides 25.8GB/s with two DRAM channels of 12.8GB/s each. We disable the cache and set the GDB rings to infinite in order to isolate the effect of the off-chip memory bandwidth accesses. Results show that for 1024 accelerators, 8 dual-channel MICs (204.8GB/s) are needed to sustain the traffic

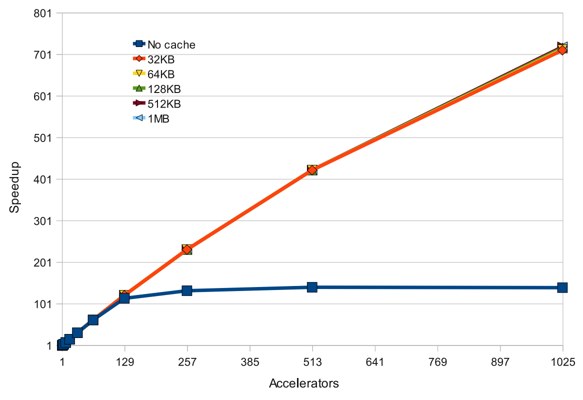

Figure 6: Sensitivity to cache size. Other parameters are: 4 cache banks, 1 MIC, 1 DRAM, INF Rings.

and achieve nearly the maximum performance. For 512 and 256 cores, 4 MICs (102.4GB/s) and 2 MICs (51.2GB/s) are enough though. Below 128 cores, bandwidth is not an issue as the application is still compute bound. With 1024 accelerators, going from 1 to 4 MICs increases the speedup from 259× to 623×.

In Figure 6 we investigated different L2 cache sizes and compare against having no cache. For accessing off-chip memory, only a single-channel MIC (12.8GB/s) is configured so that the benefit of the cache size can be better appreciated. The GDB is set to infinite rings for the same reason. We can see that a small cache of only 32KB is enough to achieve close to the maximum speedup. This is because in aligning all-to-all sequences, workers reuse a lot of the already small dataset. When increasing the cache size from 32KB to 1MB, the hit rate improves from 91% to 99%. However, 32KB are enough to achieve 98.8% of the 1MB cache performance (for 1024 cores).

However, having no caches dramatically degrades performance. Notice that the no-cache curve in Figure 6 has lower performance than the one MIC curve in Figure 5. This is because in Figure 5, MICs are always set to have two DRAM channels, as opposed to the single channel in Figure 6. The performance gain of using caches comes mostly from the

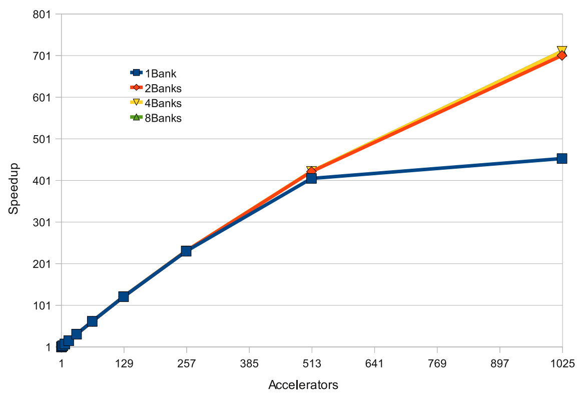

Figure 7: Sensitivity to cache banks. Other parameters are: 1MB cache, 1 MIC, 1 DRAM, INF Rings.

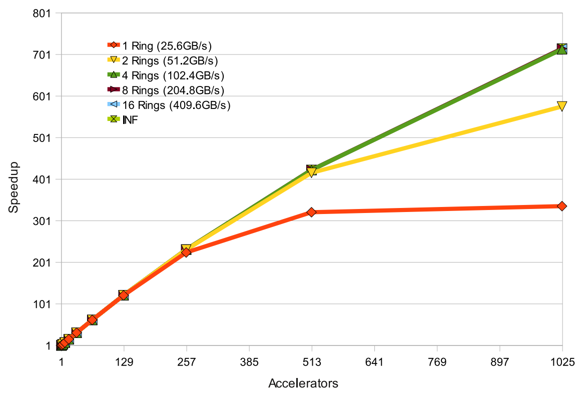

Figure 8: Sensitivity to GDB bandwidth. Other parameters are: No cache, 32 MICs.

bandwidth provided by the 4 cache banks, for a total cache bandwidth of 102.4 GB/s. This is confirmed in Figure 7 where we can see that the sensitivity is higher with respect to the number of banks, rather than the cache size. Notice that results from Figure 6 have important implications. They tell us that a tiny 32KB cache can capture most of the memory traffic that otherwise would have to go off-chip. From Figure 5 we can see that around 8 dual-channel MICs are needed to achieve similar speedups. Therefore, a small cache can save a lot of hardware complexity and more important, a lot of energy.

Next we simulate several GDB ring numbers in order to see the bandwidth sensitivity from the bus perspective. Figure 8 shows that the configuration with one ring (25.6GB/s) can only serve 256 accelerators at most. We can also see that all the traffic can be captured with an interconnect able to provide 102.4GB/s (4 rings) of aggregate bandwidth. Figure 8 shows that the maximum speedup can improve from 336× with one ring to 713× with 4 rings.

Afterwards, Figure 9 shows the tolerance to synchronization latency. Because it is a centralized and shared resource, it is a potential bottleneck when using many accelerators. Results show that up to 128 cycles are tolerated by the application. If we consider an SQ that can hold all the total tasks in our setup, a latency below 128 cycles is quite possible.

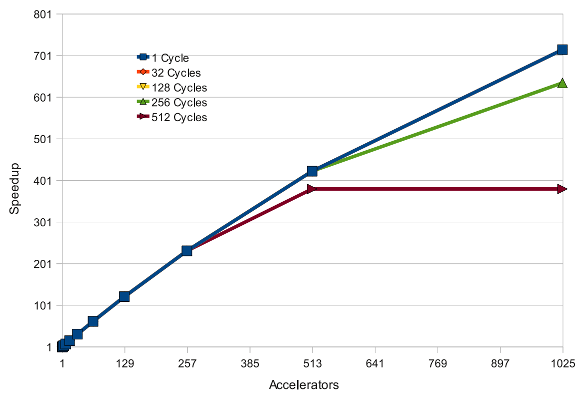

Figure 9: Sensitivity to SQ latency. Other parameters are: No cache, 32 MICs, INF rings.

There are 124,750 tasks with each entry taking 8 Bytes (an ID and a pointer). This multiplied by the two queues needed makes a total size of 1MB, small enough to have a latency smaller than 128 cycles (the Cell BE LS has 256KB with 6 Cycles latency). When having smaller tasks, the SQ bottleneck can be reduced by adding queues per cluster or per core. This result shows that using main memory to synchronize (increasing latency) could significantly affect performance.

Finally, due to low pressure imposed on shared hardware resources by the general-purpose cores, the minimal system bandwidth configuration (1 MIC, 1 ring, 1 cache bank) was able to sustain the maximum performance.

5.2 Load Balancing with GP Workers

In ClustalW-PW, tasks' duration directly depends (quadratically) on the length of the two processed sequences. When the input set is heterogeneous in sequence length, as it is the case with the BioPerf sets (see Table 2 and Figure 2), the variation in tasks' duration will be very high, up to two orders of magnitude. On the other hand, if many processors are involved and the input set is not large enough, each processor will end up with only a few tasks. With the default task scheduling policy, this leads to a few processors finishing much later than others because they had to process much larger tasks.

To look into this issue, we took 200 sequences from input C in BioPerf (due to their heterogeneity) and run simulations with up to 1024 cores. This results in 19,900 tasks, every core getting roughly 19 tasks to process on average. The lower part of Figure 10 shows how the default distribution of tasks behaves. It is a screenshot obtained with Paraver that allows us to see the behavior of all processors in time. Every horizontal line represents one core's state. The load unbalancing is appreciated in the right part of the plot.

In order to improve the load balancing we applied a simple task sorting strategy. We noticed that we can roughly estimate all tasks durations in advance just by looking at the sequence lengths. Using that information, we sort the tasks so that longer ones are scheduled first, distributed among different workers. The upper part of Figure 10 shows a significant improvement in the load balancing and consequently in performance. However, some time should be spent in the

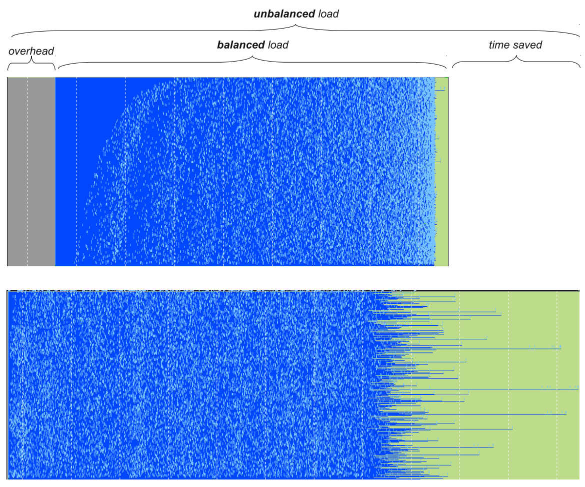

Figure 10: Load balancing with 200 sequences as input, with 1024 general-purpose workers. The vertical axis represents the cores and the horizontal axis represents time. Dark (blue) segments are computations and light ones (gray and green) show idleness.

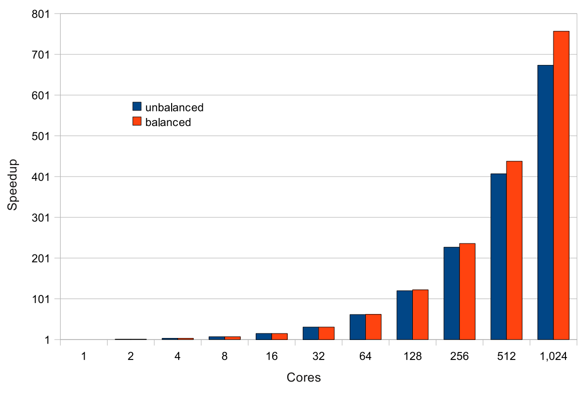

Figure 11: Performance scalability with and without task sorting using general-purpose cores. 200 sequences from BioPerf are used as input set.

sorting computation, creating an overhead [2] (see gray segment in left part of upper Figure 10). Figure 11 shows the scalability improvement of task sorting for this type of input set. These results correspond to general-purpose cores. For accelerators, because the sorting is not parallel nor accelerated, the sorting overhead becomes comparable with tasks computation and hence it is not worth. In the end, task sorting can accelerate processing time by 28%. The maximum speedup of 1× workers (Figure 11) increases from 674× to 758×.

5.3 Accelerators vs. General-purpose

Figure 12 shows the performance scalability of ClustalW-PW with respect to the number of cores used and compares the case of using accelerators with that of general-purpose

[2]The overhead time was measured on an Intel Core 2 Duo running the *qsort* function from the Standard C Library. Using a parallel sorting algorithm can reduce this overhead.

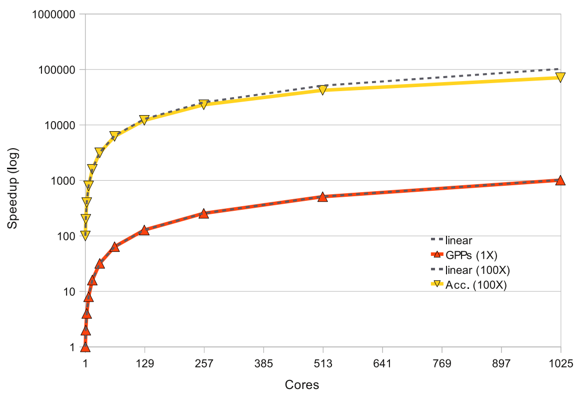

Figure 12: Performance comparison for up to 1024 workers of two types: general-purpose cores (1×) and application-specific accelerators (100×). The vertical axis is in *Log* scale and the baseline is a single general-purpose core.

cores. For both simulation sets we put the configuration that achieved the maximum performance, as discussed in Section 5.1. Speedup curves use one master and a single general-purpose worker as baseline. The vertical axis uses logarithmic scale in order to fit both performance results in a single plot while still appreciating their performance gap. Notice that due to the logarithmic scale, the flattening of the curves does not mean saturation. Dashed lines show the linear ideal scaling as a reference. Results show that even when using accelerators, we can scale performance significantly and efficiently use the 1024 cores.

Table 2 shows that for the input set used in the simulations, the theoretical maximum speedup of ClustalW is 66.67× when only PW is parallelized. Since the maximum speedup achieved for PW was 720×, this translates into 61.1× total application speedup. However, PA parallelism has been reported [13, 23] to achieve around 4× speedup on average. This would increase the speedup impact of our accelerator approach from 61.1× to 195×.

Although power analysis is out of the scope of this paper, it is important to notice that the performance gap that appears in Figure 12 is likely to be higher when considering a real chip. The simultaneous functioning of hundreds of general-purpose cores running at the typical frequencies of about 3GHz will not fit within the power budget of current CMOS technologies. Such cores will have to be run at lower frequencies, with a direct impact on performance. On the other hand, application-specific accelerators are designed to run at lower frequencies wile still providing high computational performance. In fact, their advantage comes from the fact the circuits are designed to efficiently match a particular application or application domain, not from running complex hardware structures at very high frequencies.

6. CONCLUSION

This paper presented a study of a manycore architecture targeting the multiple sequence alignment problem. A hybrid high-level/cycle-accurate simulator has been used to model a system with up to 1024 cores. We started by characterizing various input datasets and found out that they have a significant influence in the effectiveness of parallelizing ClustalW.

We have simulated a multicore architecture composed of one master processor and 1024 application-specific accelerators for ClustalW-PW. Results show that while high latencies are tolerated by the application, the system bandwidth determines the performance. This is mostly due to the use of DMAs to fetch chunks of data from main memory to the accelerator' SPMs. Simulations demonstrated that a small shared cache, as long as it is partitioned in (four) banks, is able to capture most of the memory traffic and provides the necessary bandwidth. Instead of using main memory to synchronize, we also showed that a realistic latency for the SQ is able to handle all the accelerators and all the available tasks in our already large input set.

On the other hand, the general-purpose cores have much lower throughput than the accelerators and therefore, they do not put pressure on the rest of the system. A minimalistic configuration was able to provide the maximum performance. However, we showed that when using heterogeneous inputs of small to medium size, the distribution of the load gets unbalanced. Estimating the duration of tasks in advance, allowed us to sort them and schedule long ones first. This simple strategy improved performance by 28% for general-purpose cores. However, because the task sorting is performed on a single processor, its overhead may become too big and hence not worth when using accelerators.

Simulations showed the "minimum" performance gap between general-purpose and accelerator cores for our case study. We make the observation that if implemented in real chips, the power wall will have the effect of further increasing the performance gap. This due to the fact that accelerators run at much lower frequencies, while general-purpose cores will be forced to apply frequency scaling, directly affecting performance. Lastly, we estimated that with PA parallelization, the total application speedup would increase from 61.1$\times$ to 195$\times$ using accelerators.

In the future work we will apply the same methodology to other bioinformatics applications. In particular we will study the available parallelism in ClustalW-PA and the one in HMMER [3]. We will also investigate in a more quantitative fashion, the impact of frequency scaling in performance scalability.

7. ACKNOWLEDGMENTS

This work is sponsored by the COMCAS project (CA501), a project labeled within the framework of CATRENE, the EUREKA cluster for Application and Technology Research in Europe on NanoElectronics. The SARC (#27648) and ENCORE (#249059) EU projects and the HiPEAC2 Network of Excellence have also supported our work. The authors would like to thank the Barcelona Supercomputing Center for the access to the Cell BE blades and the tools.

Sebastian Isaza and Felipe Cabarcas are also affiliated with the Department of Electronic Engineering, University of Antioquia, Colombia.

8. REFERENCES

[1] European Bioinformatics Institute, ClustalW Web Server. http://www.ebi.ac.uk/Tools/clustalw/.

[2] European Bioinformatics Institute, FASTA Web Server. http://www.ebi.ac.uk/Tools/sss/fasta/.

[3] Howard Hughes Medical Institute, HMMER Web Site. http://hmmer.janelia.org/.

[4] Kyoto University Bioinformatics Center, GenomeNet Bioinformatics Tools. http://align.genome.jp/.

[5] National Center for Biotechnology Information, BLAST Web Server. http://blast.ncbi.nlm.nih.gov.

[6] Barcelona Supercomputing Center, Paraver, Sep. 2010. http://www.bsc.es/paraver.

[7] Convey Computer Announces Record-Breaking Smith-Waterman Acceleration of 172X, May 2010. http://www.conveycomputer.com/Resources/.

[8] Swiss Institute of Bioinformatics, UniProtKB/Swiss-Prot Protein Knowledgebase Release 2010-11 Statistics, November 2010. http://www.expasy.org/sprot/relnotes/relstat.html.

[9] The European Molecular Biology Laboratory, Nucleotide Sequence Database, 2010. www.ebi.ac.uk/embl/.

[10] G. M. Amdahl. Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. In *Proceedings of the Spring Joint Computer Conference*, AFIPS '67 (Spring), pages 483–485. ACM, 1967.

[11] D. Bader, Y. Li, T. Li, and V. Sachdeva. BioPerf: A Benchmark Suite to Evaluate High-Performance Computer Architecture on Bioinformatics Applications. In *Proceedings of the IEEE International Workload Characterization Symposium.*, pages 163 – 173, Oct. 2005.

[12] K. Benkrid, Y. Liu, and A. Benkrid. A Highly Parameterized and Efficient FPGA-Based Skeleton for Pairwise Biological Sequence Alignment. *IEEE Trans. on VLSI Systems*, 17(4):561 –570, 2009.

[13] K. bin Li. ClustalW-MPI: ClustalW Analysis Using Distributed and Parallel Computing. *Bioinformatics*, 19:1585–1586, 2003.

[14] J. Cheetham, F. K. H. A. Dehne, S. Pitre, A. Rau-Chaplin, and P. J. Taillon. Parallel CLUSTAL W for PC Clusters. In *Proceedings of International Conference on Computational Science and Its Applications (ICCSA)*, pages 300–309, 2003.

[15] J. Cohen. Bioinformatics-An Introduction for Computer Scientists. *ACM Computing Surveys*, pages 122–158, 2004.

[16] R. Edgar. MUSCLE: a Multiple Sequence Alignment Method with Reduced Time and Space Complexity. *BMC Bioinformatics*, 5(1):113+, August 2004.

[17] M. Farrar. Striped Smith-Waterman Speeds Database Searches Six Times over other SIMD Implementations. *Bioinformatics (Oxford, England)*, 23(2):156–161, January 2007.

[18] M. Farrar. Optimizing Smith-Waterman for the Cell Broadband Engine. http://sites.google.com/site/farrarmichael/smith-watermanfortheibmcellbe, 2008.

[19] S. Isaza, F. Sanchez, G. Gaydadjiev, A. Ramirez, and M. Valero. Preliminary Analysis of the Cell BE Processor Limitations for Sequence Alignment Applications. In *Proceedings of the 8th International Workshop on Embedded Computer Systems:*

Architectures, Modeling, and Simulation, SAMOS '08, pages 53–64, 2008.

[20] S. Isaza, F. Sanchez, G. Gaydadjiev, A. Ramirez, and M. Valero. Scalability Analysis of Progressive Alignment on a Multicore. In *Proc. of IEEE Int. Workshop on Multicore Computing Systems*, pages 889–894, 2010.

[21] J. Kahle, M. Day, H. Hofstee, C. Johns, and D. Shippy. Introduction to the Cell Multiprocessor. *IBM Systems Journal*, 49(4/5):589–604, 2005.

[22] L. Ligowski and W. Rudnicki. An Efficient Implementation of Smith Waterman Algorithm on GPU using CUDA, for Massively Parallel Scanning of Sequence Databases. In *IPDPS '09: Proceedings of the 2009 IEEE International Parallel&Distributed Processing Symposium*, pages 1–8, 2009.

[23] P. Liu and A. Hemani. A Coarse Grain Reconfigurable Architecture for Sequence Alignment Problems in Bioinformatics. In *SASP: 8th IEEE Symposium on Application Specific Processors*, pages 50–57, 2010.

[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] A. Ramirez, F. Cabarcas, B. H. H. Juurlink, M. Alvarez, F. Sanchez, A. Azevedo, C. Meenderinck, C. B. Ciobanu, S. Isaza, and G. Gaydadjiev. The SARC Architecture. *IEEE Micro*, 30(5):16–29, 2010.

[26] A. Rico, F. Cabarcas, A. Quesada, M. Pavlovic, A. Vega, C. Villavieja, Y. Etsion, and A. Ramirez. Scalable Simulation of Decoupled Accelerator Architectures. UPC-DAC-RR-2010-10. Technical report, Technical University of Catalonia, 2010.

[27] V. Sachdeva, M. Kistler, E. Speight, and T.-H. K. Tzeng. Exploring the Viability of the Cell Broadband Engine for Bioinformatics Applications. *Parallel Computing*, 34(11):616–626, 2008.

[28] T. F. Smith and M. S. Waterman. Identification of Common Molecular Subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[29] J. Thompson, D. Higgins, and T. Gibson. 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.

[30] H. Vandierendonck, S. Rul, M. Questier, and K. D. Bosschere. Accelerating Multiple Sequence Alignment with the Cell BE Processor. *The Computer Journal*, 53(6):814–826, 2010.