2011 International Conference on Complex, Intelligent, and Software Intensive Systems

Scaling HMMER Performance on Multicore Architectures

Sebastian Isaza*, Ernst Houtgast*, Friman Sanchez†, Alex Ramirez†‡ and Georgi Gaydadjiev*
*Computer Engineering Laboratory, Delft University of Technology
†Computer Architecture Department, Technical University of Catalonia
‡Barcelona Supercomputing Center

***Abstract*—In bioinformatics, protein sequence alignment is one of the fundamental tasks that scientists perform. Since the growth of biological data is exponential, there is an ever-increasing demand for computational power. While current processor technology is shifting towards the use of multicores, the mapping and parallelization of applications has become a critical issue. In order to keep up with the processing demands, applications' bottlenecks to performance need to be found and properly addressed. In this paper we study the parallelism and performance scalability of HMMER, a bioinformatics application to perform sequence alignment. After our study of the bottlenecks in a HMMER version ported to the Cell processor, we present two optimized versions to improve scalability in a larger multicore architecture. We use a simulator that allows us to model a system with up to 512 processors and study the performance of the three parallel versions of HMMER. Results show that removing the I/O bottleneck improves performance by $3\times$ and $2.4\times$ for a short and a long HMM query respectively. Additionally, by offloading the sequence pre-formatting to the worker cores, larger speedups of up to $27\times$ and $7\times$ are achieved. Compared to using a single worker processor, up to $156\times$ speedup is obtained when using 256 cores.**

I. Introduction

The discovery of the DNA structure in 1953 has drastically altered the field of biology. In the following decades, improvements in sequencing technology has led to an explosion in availability of genetic information. Figure 1 shows the exponential growth of one of the most important global resources for protein data: the Swiss-Prot database [1]. Other genetic databases are also experiencing similar growth trends. Nowadays, the genetic structure of many different species has been sequenced and the resulting sheer size of such data sets makes analysis by hand impossible. Bioinformatics is the discipline that uses computer technology to enable types of biological research that would be unfeasible otherwise.

Within bioinformatics, sequence alignment is a primary activity. Fragments of DNA or protein sequences are compared to each other in order to identify similarities between them. Due to the computational complexity [2] of the algorithms used to process these data sets, demand for processing power is soaring. It is then critical for bioinformatics applications to be efficient and scalable in order to meet this demand. Two of the most popular sequence analysis tools are BLAST [3] and HMMER [4]. Although BLAST is faster, HMMER is more sensitive and is able to find more distant relationships. The inclusion of HMMER2 in the SPEC2006 and the recent development of HMMER3 show its significance.

In computer architecture, single threaded performance

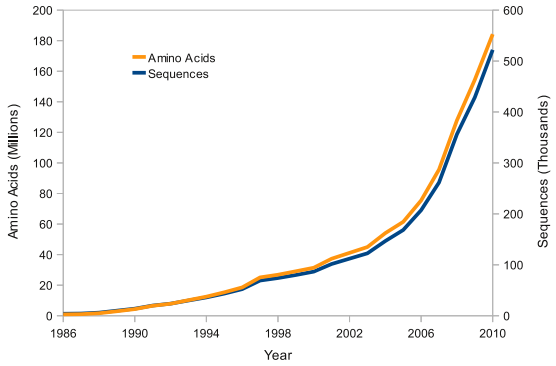


Fig. 1. Swiss-Prot database growth.

growth is stagnating because of frequency, power and memory scaling issues. As a response, technology is currently shifting towards multicore architectures. This paradigm shift however, imposes a critical challenge as performance scaling will only be achieved by the efficient parallelization of applications. Performance bottlenecks need to be found and tackled in order to keep up with the soaring demand for computational power in the bioinformatics domain.

The effectiveness of the multicore paradigm for bioinformatics applications is still an open research question. In this paper we study the scaling and parallel behavior of HMMER. We use HMMERCELL (a port of HMMER to the Cell architecture [5]) as baseline for the analysis and propose two optimized versions based on the bottlenecks found. We simulate a multicore with up to 512 processors in order to measure performance. We compare the three parallel versions, report their relative speedup and their performance scalability with the number of cores. In concrete, the main contributions of this paper are:

- The analysis of HMMERCELL bottlenecks for larger scale multicores;
- Two optimized HMMER versions that improve performance by up to $3\times$ and $27\times$ compared to HMMERCELL;
- The performance analysis of a system with up to 512 cores that is able to speedup HMMER by $156\times$.

The rest of the paper is organized as follows. Section II describes other work with similar aim and how ours is different. Section III describes HMMER program phases and the

978-0-7695-4373-4/11 $26.00 © 2011 IEEE
DOI 10.1109/CISIS.2011.102

parallel versions. In Section IV, we describe the experimental methodology used to obtain the results. In Section V we present and discuss the simulation results. Section VI draws the conclusions.

II. Related Work

HMMER has been ported to various architectures in order to accelerate its execution. In [6], an FPGA implementation of HMMER is investigated. As in HMMERCELL, the computationally intensive kernel of the Viterbi algorithm is the main focus. Similar to HMMERCELL, the FPGA is used as a filter: the sequences with a promising score require reprocessing on the host machine. A thirty fold speedup over an AMD Athlon64 3500+ is reported. This result is comparable to the performance of HMMERCELL.

HMMERCELL, the Cell port of HMMER, is created by Lu et. al. In [7], detailed information on the implementation and parallelization strategy is provided, along with raw performance data where it is benchmarked against commodity x86 architectures. Compared to the AMD Opteron platform (2.8 GHz) and the Intel Woodcrest platform (3.0 GHz), a single Cell is reported to be up to thirty times faster than those Intel and AMD processors using a single core. In this paper HMMERCELL performance is evaluated by manually instrumenting the code to obtain runtime traces and map it onto a larger multicore architecture (SARC [8]) to study performance scalability. After bottlenecks are identified, two optimized versions are proposed and evaluated.

Further parallel versions of HMMER for different architectures have been proposed by researchers. MPI-HMMER was created to take advantage of computer clusters [9]. Similar to HMMERCELL, one node is assigned a manager role and the rest of the machines are workers over which the workload is distributed. To cope with overhead from message passing, sequences are grouped in larger bundles and sent as one message. Through double buffering, communication latency is minimized. An eleven-fold speedup is reported when using sixteen machines. In [10], MPI-HMMER is analyzed and found to be scalable up to 32-64 nodes, depending on workload. PIO-HMMER is introduced, addressing I/O-related bottlenecks through the use of parallel I/O and optimized post-processing. The manager distributes an offset file with sequences to each node and they read the sequences from their local database. Furthermore, nodes only report significant results back to the manager. The resulting scaling capability is much improved, as up to 256 machines can be used effectively. Other authors have parallelized HMMER *hmmpfam* kernel for shared-memory machines [11] and for computer clusters in HSP-HMMER[12], using MPI. In [13], HMMER is reported to scale up to 1024 processors on a Blue Gene/L system. Although that paper does not report kernel performance (GCUPS), we can safely assume it is significantly slower. The Blue Gene/L platform uses PowerPC 440 cores that, compared to the Cell SPEs [5] we use, do not support SIMD processing and run at much lower clock frequency rates. Having a superior performance per worker, poses a higher pressure on the shared resources like the master processor and thus the scalability is more difficult to achieve. The aim of our work is to study the parallel performance of HMMER for on-chip multicores with high throughput workers. Furthermore, the use of on-chip multicores provides great energy savings compared to discrete multiprocessors and clusters.

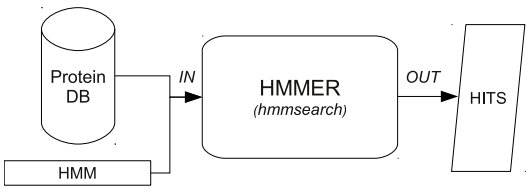


Fig. 2. Input/Output diagram of HMMER *(hmmsearch)*.

III. Application's Description

In the following we present our target application HMMER and describe the parallel versions used in our experiments.

A. HMMER

HMMER is an open source family of tools often used in biosequence analysis [4]. It is aimed specifically at protein sequence analysis. Groups of protein sequences thought of as belonging to the same family are modeled with profile Hidden Markov Models (HMMs). This paper focuses on one tool within the HMMER suite: *hmmsearch*. With this program, an HMM can be compared to a protein sequence database (see Figure 2). The goal is to find the proteins from the database that are most similar to the protein family represented by the HMM query. To perform this comparison, the Viterbi algorithm is used to generate an alignment score. The *hmmsearch* output is a list of high scoring sequences and their alignment to the HMM. Execution time is dominated by the Viterbi decoding phase, which is performed once for each sequence in the database. Profiling of the original HMMER code running on a single processor shows that for all but the simplest workloads, this phase accounts for 98+% of total running time.

B. Parallel HMMER

This paper uses HMMERCELL [7] as baseline: HMMER v2.3.2 ported to the Cell processor [5]. Since the execution time of *hmmsearch* is almost exclusively formed by the execution of the Viterbi function, the parallelization strategy focuses on this program phase and follows a master-worker scheme. The master processor creates parallel jobs that are consumed by the workers. Parallelism is used at two levels: coarse-grain parallelism by spawning threads and fine-grain parallelism by using SIMD processing within the Viterbi kernel. Due to memory limitations caused by the small Local Stores, the Viterbi algorithm is modified to use a smaller memory footprint. Hence, worker processors do not provide a full alignment but only produce an alignment score. High scoring alignments are reprocessed (by running the full Viterbi) on the master processor to obtain the actual alignment (more details on HMMERCELL can be found in [7]).

After extensive profiling and tracing of HMMERCELL we present here two optimized versions that are more scalable and target a larger and more general multicore architecture

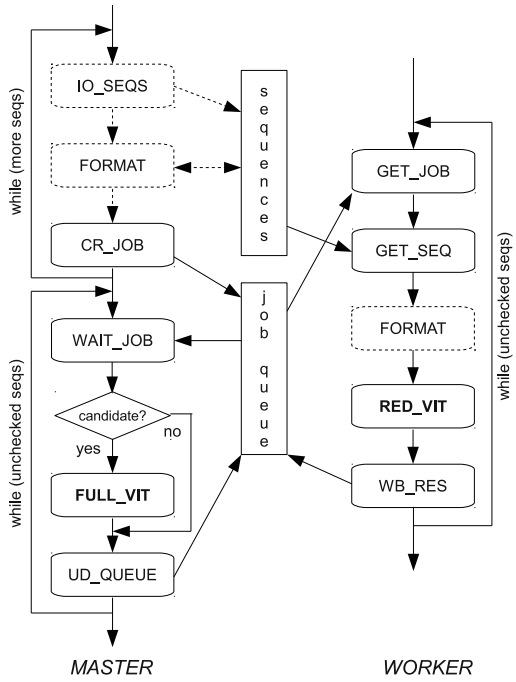


Fig. 3. Parallel HMMER diagram.

(see SectionIV-A). Figure 3 shows a diagram of the parallel HMMER functioning and Table I lists the phases involved. For every sequence in the database, two steps are involved at first. A sequence is load from the disk into main memory (IO_SEQS) and then it is formatted (FORMAT) for appropriate Viterbi processing. In HMMERCELL, both phases are sequentially handled by the master processor which creates a centralized bottleneck. Since the Swiss-Prot database amounts to 234MB, we notice that in a realistic scenario, the database can be permanently hold in memory. Thus, the application is relieved from the I/O bottleneck and should be able to efficiently use more processors. The second observation is that the FORMAT phase is independent for every sequence in the database and therefore can be performed by the workers in parallel. The three parallel versions described are listed here for reference:

- **BASE**: Baseline HMMERCELL parallelization with I/O reads and sequential formatting of sequences.
- **M_FORMAT**: Removing the I/O bottleneck by holding database in main memory. FORMAT is performed by the master.
- **W_FORMAT**: Besides removing I/O, the FORMAT phase is performed by the workers in parallel.

The processing done in FORMAT consists in replacing every amino acid by its index in the alphabet. Hence, moving this simple operation to the workers side only implies a negligible increase in code size and no extra memory for the sequence as it can be overwritten when formatted.

For the sake of clarity, Figure 3 does not show the loading of the HMM as this is done only once, at the beginning.

TABLE I
LIST OF PROGRAM PHASES.

Phase name	Description
IO_SEQS	Read sequences from disk.
FORMAT	Format sequence for proper processing.
CR_JOB	Create an entry in the job queue.
WAIT_JOB	Wait for an available job to process.
FULL_VIT	Compute full Viterbi algorithm.
UD_QUEUE	Update entry status as done.
GET_JOB	Get a job entry from the queue.
GET_SEQ	DMA sequence from memory to SPM.
RED_VIT	Compute reduced Viterbi algorithm.
WB_RES	Write-back result in queue.

IV. EXPERIMENTAL METHODOLOGY

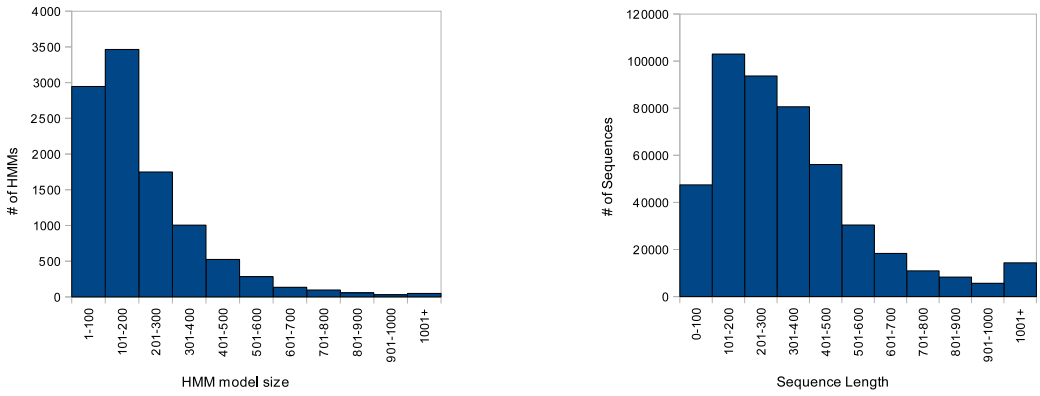
To study the impact of the different parallelizations we have used the *TaskSim* simulator [8], [14] that models the parameterizable multicore architecture as sketched in Figure 5 (see Section IV-A). The simulator uses a hybrid trace-driven high-level/cycle-accurate technique, that is, some components are simulated very accurately while others are abstracted out, depending on the aspects one wants to analyze. The simulator inputs 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 HMMERCELL using a Cell adapted version of the MPItrace tracing library [15]. The generated traces have been inspected with Paraver [16] in order to visualize the bottlenecks. The instrumented code was run on an IBM QS21 Blade, with a Cell processor running at 3.2GHz and 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 one SPE in Cell) 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.

Protein sequence data sets are from the Swiss-Prot database [1] and HMMs from Pfam [17]. Figure 4 shows the current model and sequence length distribution for Pfam and Swiss-Prot databases. Based on this information, input test sets have been chosen. Two HMMs of 100 and 500 states are used to study a typical and a demanding execution scenario. In order to keep traces under a reasonable size, we have used a randomly selected subset of Swiss-Prot for the simulations (20,000 sequences). Every sequence creates a job and so we have enough jobs to stress our multicore system.

A. Multicore Architechture

In this paper we have studied the performance scalability of HMMER on the SARC architecture [8], shown in Figure 5. It is a clustered (or tiled) multicore interconnected with a system of hierarchical buses. Clusters of 8 (P)rocessors 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.

SARC is a heterogeneous architecture given that different processor types coexist. The (M)aster processor is a powerful



(a) Pfam (b) Swiss-Prot

Fig. 4. Entry length histogram of protein databases.

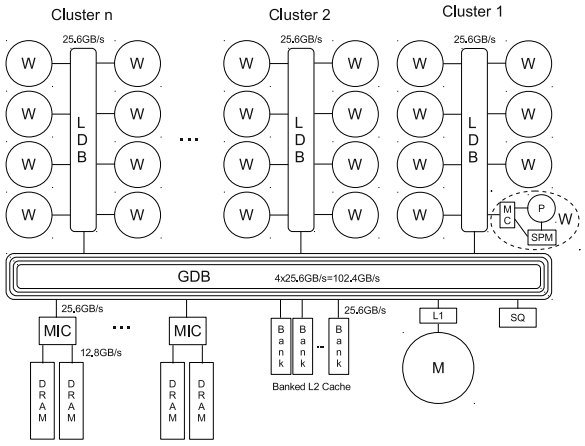


Fig. 5. The SARC multicore architecture.

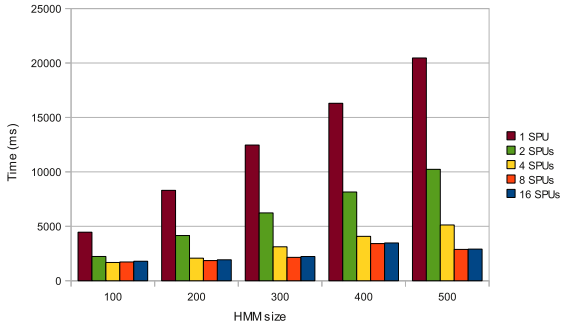


Fig. 6. HMMERCELL execution time overview.

out-of-order superscalar processor that can efficiently run the operating system and the control sections of applications. 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 types of memory: its own scratchpad memory (SPM), the global shared memory and the other worker's SPM. For the last two cases, it is required to program a DMA engine on the memory controller (MC) which takes care of the data transfer, decoupling it from the worker processor execution.

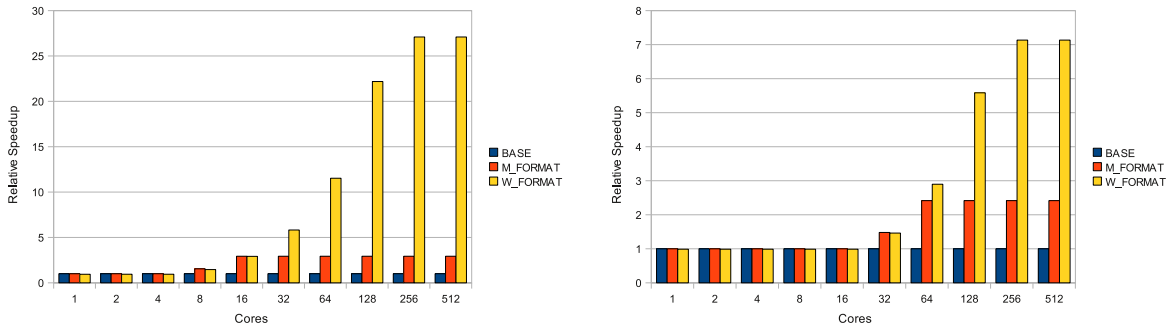
Each LDB is a bus with a ring topology, able to transmit 8 bytes per cycle at 3.2GHz (that is, 25.6 GB/s). Since the GDB is expected to handle more traffic, its bandwidth is increased with 4 rings. Every ring provides an independent communication channel between any pair of components in the system. Main memory is composed of two off-chip DRAMs controlled by one MIC providing up to 25.6GB/s. The DRAMs are DDR3-1600 with a peak bandwidth of 12.8GB/s. Fine-grain interleaving at the DRAM channels is used, that is, consecutive memory lines change DRAMs so that large DMA requests of consecutive memory ranges can be processed in parallel accessing several DRAMs concurrently.

The L2 cache size is 1MB and it is organized in 4 banks, each able to provide 25.6GB/s. As in the DRAMs 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) is composed by 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.

V. Results and Discussion

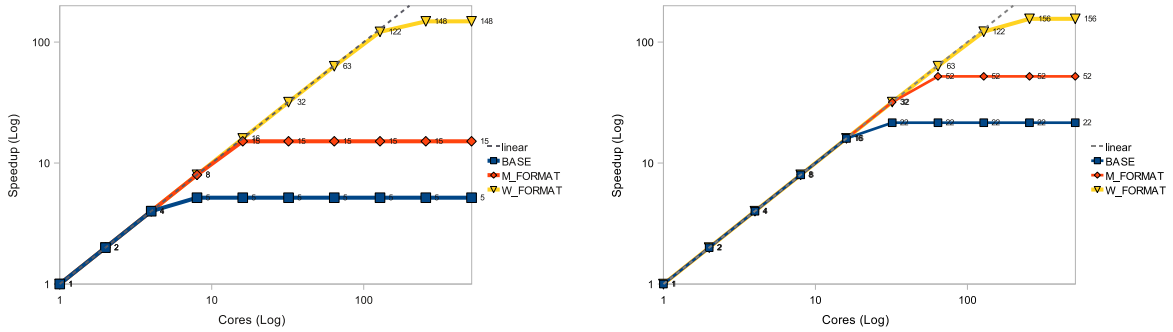
In this section we first show profiling results of HMMERCELL in order to have an overview of performance scalability



(a) Short HMM (100 states)

(b) Long HMM (500 states)

Fig. 7. Speedup relative to the BASE version.



(a) Short HMM (100 states)

(b) Long HMM (500 states)

Fig. 8. HMMER performance scalability with a single worker core as baseline.

with various types of inputs. We use 5 distinct HMMs with lengths from 100 to 500 positions, thus covering the average and demanding scenarios as follows from Figure 4(a) (larger HMMs need more processing). Then we create a sequence set consisting of 20.000 randomly selected sequences with length distribution identical to the Swiss-Prot database. Figure 6 gives an overview of HMMERCELL performance where some basic characteristics on how HMMERCELL reacts to changes in input parameters are revealed: the use of a longer HMM model size requires correspondingly longer execution time; in general, the use of additional SPEs leads to shorter execution times; and only a certain number of SPEs can be used effectively, depending on the workload. Due to management overhead, using more SPEs results in identical or even deteriorated performance.

Afterwards, we investigate the performance of the two optimized versions (see Section III) and compare it with BASE on the SARC architecture. Figure 7 shows the speedup provided by the M_FORMAT and W_FORMAT versions over BASE, for various processor counts. We analyze both the average case (100 states HMM) and a more demanding one (500 states HMM). On Figure 7(a) we see that for the short HMM there is no benefit when using 4 or less cores. For few cores, the I/O overhead does not affect performance because the execution time is largely dominated by the computation of the Viterbi kernel (RED_VIT). On the other hand, a large HMM imposes an even higher demand on the RED_VIT processing and since it is executed in parallel, more cores (16) can be efficiently used (Figure 7(b). By holding the database in main memory and thus avoiding the I/O bottleneck, a $3\times$ and $2.4\times$ speedup is obtained for short and long HMMs respectively. Finally parallelizing the FORMAT phase brings the largest improvement. Speedups of up to $27\times$ and $7\times$ are achieved for each case, compared to the BASE version. For short HMMs, the impact is significantly larger due to the fact that the sequential parts amount to a bigger portion of the execution. Compared to the M_FORMAT version, W_FORMAT reaches $9.3\times$ and $3\times$ speedups for the short and long HMMs respectively.

Figure 8 shows the performance scalability in the number of cores using *Log-Log* axes. For the short HMM (Figure 8(a)), the BASE version can only take advantage of 8 workers for a maximum speedup of $5\times$. M_FORMAT increases maximum speedup to $15\times$ with 16 cores and W_FORMAT increases the number of utilizable cores to 256 to achieve $148\times$ speedup. Figure 8(b) shows that for a large HMM, performance scales a bit more (up to $156\times$). This is because a longer HMM makes the parallel portion of work larger.

In Figure 9, we plot the cumulative time distribution of phases in the best parallel version (W_FORMAT). This, in order to study the performance saturation that occurs when using more than 256 cores. We look at four scenarios: using

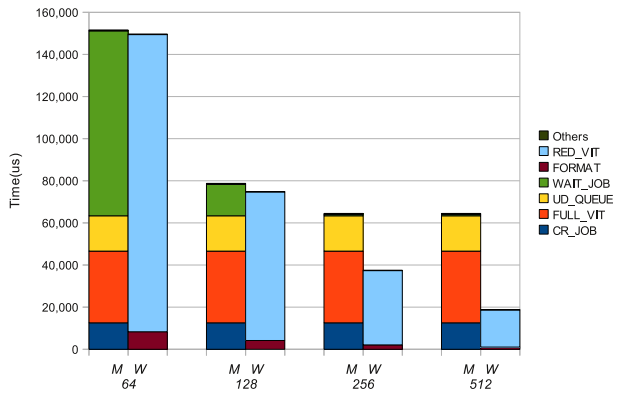


Fig. 9. Time share of HMMER program phases for a short HMM.

64, 128, 256 and 512 cores. For each case, one bar represents the master and the other one the average behavior of the workers. First of all, we see that on the master side, the WAIT_JOB time gets reduced with more cores that are able to finish the jobs faster. The other phases in the master stay unchanged. On the workers side, most of the time is spent on RED_VIT and FORMAT phases, indicating a high utilization rate. With the increase in the number of cores, phases get reduced proportionally. The most important trend to observe in Figure 9 is that the reduction in total execution time (determined by the master), stops at 256 cores due to fixed duration of three phases: CR_JOB, FULL_VIT, UD_QUEUE. These, stay unchanged in all cases as they only depend on the input data and not on the number of cores being used, as in Amdahl's law.

Figure 9 also reveals that workers spend most of the time in computing the Viterbi kernel (and formatting) and do not manage to saturate the memory nor the buses. This was confirmed by running further simulations with different system configurations. These experiments showed that increasing the GDB bandwidth, adding more cache and cache banks, and adding more MICs to increase the memory bandwidth did not change the time it takes the workers to complete all jobs.

VI. Conclusions

This paper analyzed the performance scalability of several parallel versions of the HMMER software for protein sequence alignment. We started by analyzing HMMERCELL, a Cell port of HMMER. By manually instrumenting the code and looking into the execution traces, we were able to find out the bottlenecks when the application is ported to a larger multicore like SARC. Based on our findings, we proposed two optimized versions that were aimed at alleviating the bottlenecks: holding the database in memory to avoid I/O access and parallelizing the pre-formatting of sequences. Simulation results showed that the M_FORMAT version was up to $3\times$ faster while the W_FORMAT parallelization achieved up to $27\times$ speedup, compared to HMMERCELL. Compared to using a single worker processor, W_FORMAT scaled performance up to $156\times$ with 256 cores. Adding more cores or increasing the memory bandwidth did not improve performance. This was because the system is then bottlenecked by the master processing the FULL_VIT phase and collecting results from all workers. Parallelizing FULL_VIT in a system with larger scratchpad memories is the way to increase scalability further.

We are convinced that our study presents very useful insights in the design of future multicore chips targeting bioinformatics workloads.

Acknowledgments

This work was carried out under the COMCAS project (CA501), a project labeled within the framework of CATRENE, the EUREKA cluster for Application and Technology Research in Europe on NanoElectronics. This work was also sponsored by the HiPEAC2 Network of Excellence. The authors would like to thank Felipe Cabarcas and the Barcelona Supercomputing Center for the access to the Cell blades and the TaskSim simulator.
Sebastian Isaza is also affiliated with the Deparment of Electronic Engineering, University of Antioquia, Colombia.

References

[1] "Uniprotkb/swiss-prot protein knowledgebase release 2010-11 statistics," November 2010, http://www.expasy.org/sprot/relnotes/relstat.html.
[2] R. Edgar, "MUSCLE: a multiple sequence alignment method with reduced time and space complexity," *BMC Bioinformatics*, vol. 5, no. 1, pp. 113+, August 2004.
[3] S. F. Altschul, W. Gish, W. Miller, M. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, pp. 403–410, 1990.
[4] S. R. Eddy, "Profile hidden markov models," *Bioinformatics*, vol. 14, no. 9, pp. 755–763, 1998.
[5] J. Kahle, M. Day, H. Hofstee, C. Johns, and D. Shippy, "Introduction to the cell multiprocessor," *IBM Systems Journal*, vol. 49, no. 4/5, pp. 589–604, 2005.
[6] T. Oliver, L. Yeow, , and B. Schmidt, "Integrating fpga acceleration into hmmer," *Parallel Computing*, vol. 34, no. 11, pp. 681–691, 2008.
[7] J. Lu, M. Perrone, K. Albayraktaroglu, , and M. Franklin, "Hmmer-cell: High performance protein profile searching on the cell/b.e. processor," in *ISPASS '08: IEEE International Symposium on Performance Analysis of Systems and software*, 2008.
[8] 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*, vol. 30, no. 5, pp. 16–29, 2010.
[9] J. P. W. B. Qudah and V. Chaudhary, "Accelerating the hmmer sequence analysis suite using conventional processors," in *AINA '06: International Conference on Advanced Information Networking and Applications*, 2006.
[10] J. P. Walters, R. Darole, , and V. Chaudhary, "Improving mpi-hmmer's scalability with parallel i/o," in *IPDPS '09: IEEE International Symposium on Parallel & Distributed Processing*, 2009.
[11] U. Srinivasan, P.-S. Chen, Q. Diao, C.-C. Lim, E. Li, Y. Chen, R. Ju, and Y. Zhang, "Characterization and analysis of hmmer and svm-rfe parallel bioinformatics applications," in *IEEE International Symposium on Workload Characterization*, 2005.
[12] B. Rekapalli, C. Halloy, and I. Zhulin, "Hsp-hmmer: A tool for protein domain identification on a large scale," in *ACM Symposium on Applied Computing*, 2009.
[13] K. Jiang, O. Thorsen, A. Peters, B. Smith, and C. P. Sosa, "An efficient parallel implementation of the hidden markov methods for genomic sequence-search on a massively parallel system," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, pp. 15–23, January 2008.
[14] 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 University of Catalonia, Tech. Rep., 2010.
[15] "Mpitrace tool user's guide," http://www.bsc.es/media/1379.pdf.
[16] "Paraver," Sep. 2010, http://www.bsc.es/paraver.
[17] E. L. Sonnhammer, S. R. Eddy, and R. Durbin, "Pfam: a comprehensive database of protein domain families based on seed alignments," *Proteins*, vol. 28, no. 3, pp. 405–420, 1997.