SEQUENCE ALIGNMENT APPLICATION MODEL FOR MULTI- AND MANYCORE ARCHITECTURES[1]

Sebastian Isaza, Ernst Houtgast, Georgi Gaydadjiev

Computer Engineering Laboratory, Delft University of Technology
e-mails: s.isazaramirez@tudelft.nl
The Netherlands

Abstract: Exponential growth in biological sequence data combined with the computationally intensive nature of bioinformatics applications results in a continuously rising demand for computational power. In this paper, we propose a performance model that captures the behavior and performance scalability of HMMER, a bioinformatics application that identifies similarities between protein sequences and a protein family model. With our analytical model, the optimal master-worker ratio for any specific user scenario can be estimated. The model is evaluated and is found accurate with error lower than 2%. We applied our model to a widely used heterogeneous multicore architecture, the Cell BE, using the PPE and SPEs as master and workers respectively. Experimental results show that for the current parallelization strategy, the I/O speed to read the database from the disk and the inputs pre-processing are the two most limiting factors in the Cell BE case.

Key words: multicore architectures, bioinformatics, performance models.

1. INTRODUCTION

The rapid development of genetics in recent decades has led to an explosion of genetic information databases. The genetic structure of many species has been sequenced and the resulting sheer size of such data sets makes analysis by hand impossible. In bioinformatics, computer technology is used to enable biological research directions 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

[1] This work was sponsored by the European Commission in the context of the following projects: SARC (#27648) and the HiPEAC NoE. The authors would like to thank the Barcelona Supercomputing Center for the access to the Cell BE blades, tools and their excellent technical support.

similarities between them. Due to the computational complexity of the algorithms used to process these data sets, demand for processing power is soaring. Therefore, it is critical for bioinformatics applications to be efficient and scalable in order to meet this demand. Two popular sequence analysis tools are BLAST [6] and HMMER [7]. Each has its own merits: BLAST is faster; HMMER is more sensitive and also able to find more distant relationships. The adoption of HMMER2 in the SPEC2006 benchmark suite and the recent HMMER3 developments show its significance.

Advancements in microprocessor technology in the past have resulted in steadily increasing computational power, through miniaturization and growing transistor budgets. However, single threaded performance improvement is stagnating because of frequency, power and memory scaling barriers. These “walls” are the reason for the current paradigm shift towards multicore architectures, in an attempt to deliver the expected performance growth. One example of such multicore architecture is Cell BE, a processor with special architectural components and organization that has opened a new path in processor design. In this paper we have used the Cell BE as a case study to validate our proposal, how- ever, we consider that our analysis is applicable to other multicore architectures as well.

The suitability and effectiveness of the multicore paradigm for bioinformatics applications is still an open research question. In this paper, we develop an analytical model of a master-worker parallelization of HMMER for multicore architectures. As a case-study, we use HMMERCELL (a port of HMMER to the Cell architecture) to investigate its scalability behavior. Through profiling on the Cell processor we set the coefficients and finally we are able to validate our model. In essence, main contributions of this paper are:

– Highly accurate performance prediction model (with error within 2%);
– Detailed, quantitative characterization of program phases and their influence on overall performance;
– A careful study of the HMMER scalability bottlenecks.

The remainder of the paper is organized as follows. Section 2 describes the related work. Section 3 briefly introduces HMMER, the Cell BE processor main components and HMMERCELL. In Section 4 we present and validate our performance model. Section 5 describes the experimental methodology used for profiling and for the model construction and validation. Section 6 shows the results, Section 7 discusses our findings and Section 8 presents our conclusions.

2. RELATED WORK

HMMERCELL, the Cell BE port of HMMER, is created by Lu et al. In [9], 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, 1-4 cores) and the Intel Woodcrest platform (3.0 GHz, 1-4 cores), a single Cell BE is reported to be up to thirty times faster than a single core Intel or AMD processor. In contrast, in this paper we build a model of HMMER for a master-worker parallelization scheme and use HMMERCELL as a validation example. Besides, we evaluate HMMERCELL performance in much more detail by breaking down performance into three constituent phases. These are then modeled and profiled in order to analyze their behavior for various workloads. Finally, bottlenecks to scalability are discussed.

HMMER has been ported to various architectures. In [10], 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- comparable to the HMMERCELL performance.

MPI-HMMER was created to take advantage of computer clusters [14]. 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 over-head 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 [15], 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, worker nodes 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 [13] and for computer clusters in HSP-HMMER[11], using MPI. Although our proposed model could also be used for the mentioned HMMER versions, this paper only verifies the model against the HMMERCELL implementation. The reason why HMMER- CELL scales to less cores than other implementations [14, 15] is because of the higher per-core Viterbi performance brought by the Cell's SPEs.

3. BACKGROUND

In this section we start by introducing HMMER functionality. Then we describe the basic features of our implementation platform, the Cell BE processor, and finally we discuss the parallel behavior of HMMERCELL.

3.1. HMMER

HMMER [7] is an open source family of tools often used in biosequence analysis. 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. To perform this comparison, the Viterbi algorithm is used to generate an alignment and a bit score. Based on the bit score, the E-value is calculated, which gives the number of false positives with similar bit score that can be expected for this database size. Larger databases lead to more false positives, so in those cases, an alignment requires a higher bit score to be counted as a significant hit. 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 shows that for all but the simplest workloads, this phase accounts for 98+% of total running time.

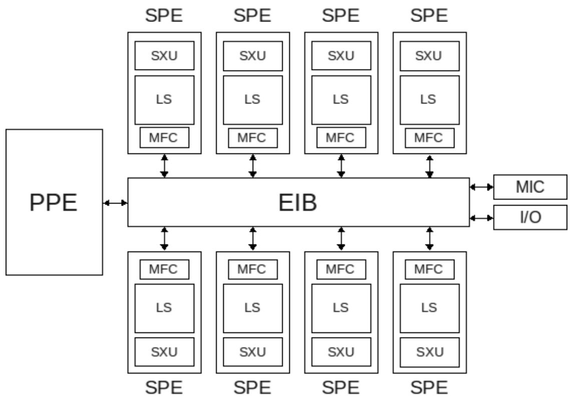

Fig. 1. Cell Broadband Engine architecture block diagram.

3.2. THE CELL BROADBAND ENGINE

The Cell Broadband Engine [8] represents a radical departure from traditional microprocessors design. Figure 1 shows a block diagram of the architecture. The Cell BE features a heterogeneous architecture with 9 computing cores: the Power Processing Unit (PPE), used for general purpose tasks, and 8 Synergistic Processing Elements (SPEs), designed for streaming workloads. SPEs are dual-issue in-order SIMD cores with 256KB Local Stores (LS) and 128 registers, 128-bit wide. The PPE is a 2-way Simultaneous Multithreading dual-issue in-order PowerPC processor. The EIB is a circular ring comprising four 16B-wide unidirectional channels that connects the SPEs, the PPE, two memory controllers and two I/O controllers. The operating system runs on the PPE and software can spawn threads in the SPEs. Data has to be explicitly copied to the SPEs LSs using Direct Memory Access (DMA) commands. The Memory Flow Controller (MFC) in each SPE takes care of these DMA transfers and it does so in parallel to the SPEs' SIMD execution unit.

3.3. HMMERCELL

HMMERCELL [9] is the port of *hmmsearch v2.32* to the Cell BE architecture. Since the execution time of *hmmsearch* is almost exclusively formed by the Viterbi function execution, the parallelization strategy focuses on Viterbi.

In order to optimally utilize the Cell BE architecture, a few key techniques have been used. First of all, parallelism is used at two levels: coarse-grain parallelism by spawning SPE threads and fine-grain parallelism within the SPEs, by using a highly efficient SIMDized version of the Viterbi algorithm [9]. Secondly, due to the small SPE Local Store, the use of small memory footprint version of the Viterbi algorithm is required. Hence, SPEs do not provide a full alignment but only produce an alignment score. High scoring alignments are reprocessed on the PPE to obtain the actual alignment.

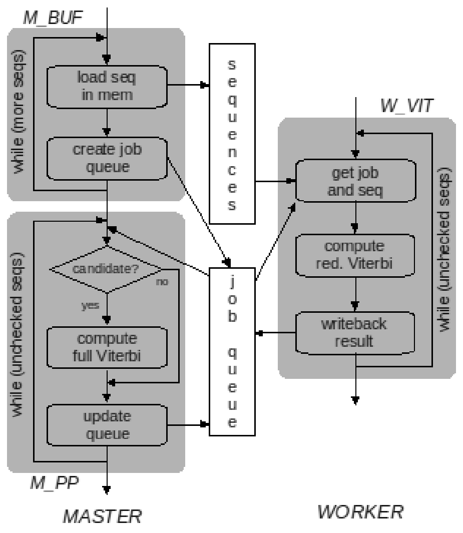

Fig. 2. HMMERCELL program phases.

In Figure 2, an overview is given of the HMMERCELL internal functioning. The PPE and the SPEs assume the master and worker rolls respectively in our case study. The important phases are:

– M-BUF: the master buffers the protein sequences by loading them from disk to main memory and creates tasks for the workers by adding entries in a job queue.

– W-VIT: once a worker gets a job from the queue, it copies the assigned protein from main memory to its LS, performs the reduced Viterbi algorithm and writes the result score back to main memory.

– M-PP: during the post-processing phase, the master runs the full Viterbi algorithm to recover the alignment of proteins that have passed the threshold.

For the sake of clarity, Figure 2 does not show the loading of the HMM as this is done only once, at the beginning, and therefore negligible.

4. HMMER ANALYTICAL MODEL

Here we present an analytical model that estimates the total execution time of a HMMER parallel version that uses the master-worker paradigm. Based on theoretical expectations and code inspection, we model the required time for each program phase separately and then combine these phases together. This results in an accurate model for HMMER performance on multicore platforms.

First, we start with the derivation of the different functions of the model. Then, the model is applied to our implementation platform, the Cell Broadband Engine, to define the numeric values of the different constants. The analytical results are validated and used to show how the model can be used to derive the maximum effectively usable SPE count. More information on the program phases and the profiling results are presented in Section 6.

4.1. MODEL DERIVATION

The following parameters are used in our HMMER model:

– T_M, T_W : master-worker processor time;

– $T_{M\_BUF}, T_{M\_PP}, T_{W\_VIT}$: execution time of phases;

– l_i : length of a specific sequence;

– $\overline{l}$: average length of sequences in the test set;

– m : length of the profile HMM H;

– n : number of sequences in the test set S;

– P_{PP} : chance for protein sequence to score above the threshold and thus requiring post-processing on the master;

– q : number of workers used;

– $\alpha, \beta, \gamma, \delta, C_\alpha, C_\beta, C_\gamma, C_\delta$: model coefficients.

The required time (t) for each phase to process a single sequence is expressed in Equations 1-3 and is based upon expectations from theory and program inspection. Function I_{PP} acts as an indicator, returning 1 when an alignment between a sequence s_i and the model H is significant for a test set of size n and otherwise returning 0. Such a sequence requires post-processing on the master node, which in our case means re-computing the alignment using the full Viterbi algorithm on the PPE.

$$t_{M\_BUF} = \alpha \cdot l_i + C_\alpha \qquad (1)$$

$$t_{W\_VIT} = \beta \cdot m \cdot l_i + C_\beta \qquad (2)$$

$$t_{M\_PP} = \gamma \cdot m \cdot l_i + C_\gamma \qquad (3)$$

Aggregating these equations for individual sequences to the entire test set (containing *n* sequences) results in Equations 4-6. The indicator function I_{PP} has been replaced by the probability function P_{PP} , giving the average chance for a sequence in test set *S* to require post-processing. Predicting the result of indicator function I_{PP} is difficult, as it requires knowledge of the biological match between the protein model and a specific sequence. Probability P_{PP} however, can be estimated based on overall traceback count of a test set. Also, T_{W_VIT} states the time required for the Viterbi computations of all the sequences combined.

$$T_{M\_BUF} = n \cdot (\alpha \cdot \overline{l} + C_\alpha) \qquad (4)$$

$$T_{W\_VIT} = n \cdot (\beta \cdot m \cdot \overline{l} + C_\beta) \qquad (5)$$

$$T_{M\_PP} = n \cdot (\gamma \cdot m \cdot \overline{l} + C_\gamma) \qquad (6)$$

$$\text{with } P_{PP} = \frac{\delta \cdot \ln(n) + C_\delta}{n}$$

To combine the previous equations into an integrated model of HMMER performance, the interrelation between the functions should be taken into account. The dependencies between these three functions are depicted in Figure 3. *W_VIT* starts after *M_BUF*, as at least one sequence should be buffered before processing by the workers can commence. *W_VIT* ends after *M_BUF*, as the last sequence to be buffered must be processed as well. *M_PP* starts when *M_BUF* finishes, as both buffering and post-processing are performed on the master node. *M_PP* ends after *W_VIT*, as the last processed sequence must be checked by the master.

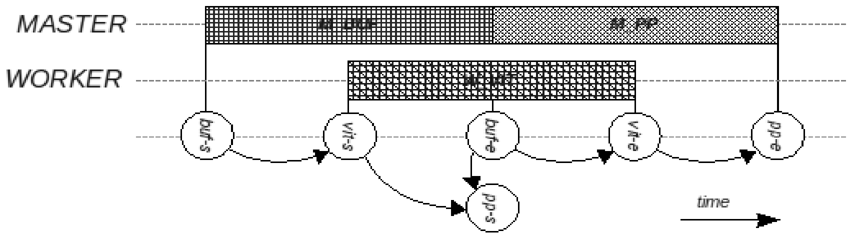

Fig. 3. Relationship of dependence between.

When a test set contains many thousands of sequences, processing time of any individual sequence is insignificant when compared to total execution time. This observation allows for two simplifications: first, the above dependencies between functions can be approximated as follows: *M_BUF* and *W_VIT* can be assumed to start at the same time, *M_PP* starts when *M_BUF* completes, and *M_PP* must finish after *W_VIT*. The model also assumes that when *M_BUF* finishes there are hits already available for it to post-process. This is reasonable considering the fact that *M_BUF* is characterized by long latencies and because hits will usually be randomly distributed in the database.

On the other hand, load balancing between workers is assumed to be perfect, as all processes will finish at approximately the same time. This approximation and hence the accuracy of the model relies on the assumption that the test set contains a large number of sequences, so that the granularity of individual sequence processing becomes very small. This is reasonable, for example a relevant workload such as the SwissProt database contains around half a million sequences. Using these assumptions, execution time is modeled as per Equations 7-9:

$$T_M = T_{M\_BUF} + T_{M\_PP} \tag{7}$$

$$T_W = T_{W\_VIT} / q \tag{8}$$

$$T_{TOTAL} = \max(T_M, T_W) \tag{9}$$

4.2. MODEL PARAMETRIZATION

The previous section shows the generalized form of a performance model for an application parallelized using the master-worker paradigm. The coefficients α-δ, C_α-C_δ and function P_{PP} are specific to the actual implementation of HMMER. Here, we show the actual values for our implementation on the Cell BE architecture. Using linear and logarithmic regression, the parameterized values are derived from the profiling results. The extra processing time required by reprocessing high scoring sequences on the master node is incorporated in the coefficient's values.

$$T_{M\_BUF} = n \cdot \left(\frac{0.19}{10^3} \cdot \overline{l} + \frac{5.52}{10^3} \right) \tag{10}$$

$$T_{W\_VIT} = n \cdot \left(\frac{0.59}{10^3} \cdot \frac{m}{10^2} \cdot \overline{l} + \frac{0.88}{10^3} \right) \tag{11}$$

$$T_{P\_PP} = n \cdot \left(\frac{2.25}{10^3} \cdot \frac{m}{10^2} \cdot \overline{l} + \frac{35.7}{10^3} \right) \cdot P_{PP} \tag{12}$$

$$\text{with } P_{PP} = \frac{21.9 \cdot \ln(n) + 73.2}{n}$$

Combining Equations 7-9 and 10-12, total execution time is approximated by:

$$\max\begin{cases} T_M \Rightarrow n\cdot\left[\left(\frac{0.19}{10^3}+\frac{2.25}{10^3}\cdot\frac{m}{10^2}\cdot P_{PP}\right)\cdot\overline{l}+\left(\frac{5.52}{10^3}+\frac{35.7}{10^3}\cdot P_{PP}\right)\right] \\ T_W \Rightarrow n\cdot\left(\frac{0.59}{10^3}\cdot\frac{m}{10^2}\cdot\overline{l}+\frac{0.88}{10^3}\right)\Big/q \end{cases} \tag{13}$$

4.3. MAXIMUM EFFECTIVE SPE COUNT

Equation 13 can be used to derive the number of workers (or in the case of Cell: SPEs) that can be effectively used in scenarios that are constrained by the master's buffering performance (in our case: the PPE). In such situations, the number of workers that will saturate the master's buffering capability can be estimated by setting T_{M_BUF} equal to T_W, which results in the maximum effective number of workers q:

$$q \approx \frac{T_{W\_VIT}}{T_{M\_BUF}} = \frac{0.59\cdot\frac{m}{10^2}\cdot\overline{l}+0.88}{0.19\cdot\overline{l}+5.52} \tag{14}$$

From this equation, it follows that the number of usable workers is solely dependent on HMM model size. Table 1 gives the maximum number of usable workers for various HMM sizes when using sequences with typical length. Profiling results in Section 6 confirm the data from Table 1.

Table 1. Maximum effectively usable workers.

HMM Length	100	200	300	400	500
q (max worker count)	3	6	9	12	15

4.4. MODEL VALIDATION

To validate our model, additional tests have been performed with new randomly selected data sets of 20.000 and 40.000 sequences (the size is constrained to fit in our blade user quota, but large enough to be significant for the experiments). These test sets have been checked to have an average sequence length near 355 symbols in order to ensure the same behavior as with the full SiwssProt database. Sequences are compared against four different HMMs with length 150 and 450 (two representative lengths as seen in Figure 4). The execution time of each of the HMMER phases is shown in Table 2 for both the empirical execution and the model prediction. The last two columns show the percentage error between prediction and estimation. Our model was able to accurately estimate the execution time of *M_BUF* and *W_VIT*, average deviation between result and expectation was 1.5% and 1.7% respectively. On the contrary, the estimation for *M_PP* was unreliable, as the number of sequences that require post-processing depends on the biological fit between data set and the HMM, and because the time for post-processing varies considerably for each sequence. However, the *M_PP* model inaccuracy will only affect overall performance estimation if the application is constrained by the *M_PP* phase, which only occurs if a high fraction of sequences requires post-processing. However, as traceback count scales logarithmically in test set size, this fraction is marginal for realistic test sizes. Furthermore, for shared memory architectures where the M_PP phase does not need to compute the full Viterbi algorithm, the significance of the phase is even less than in our case-study. Thus, *M_PP* contribution to total execution time is negligible. Overall, the average error of our model was below 2%.

Table 2. Validation results.

Test	Empirical Results			Model Results			Difference	
m,n,q	BUF	VIT	PP	BUF	VIT	PP	BUF	VIT
20k,150a,1	1498	6349	581	1459	6301	176	-2.6%	-0.8%
20k,150a,8	1492	797	581	1459	788	176	-2.2%	-1.2%
20k,150b,1	1442	6345	336	1459	6301	176	1.2%	-0.7%
20k,150b,8	1492	797	335	1459	788	176	-2.2%	-1.2%
20k,450a,1	1441	18440	777	1459	18868	519	1.3%	2.3%
20k,450a,8	1448	2303	776	1459	2359	519	0.8%	2.4%
20k,450b,1	1441	18436	1032	1459	18868	519	1.3%	2.3%
20k,450b,8	1446	2305	1031	1459	2359	519	0.9%	2.3%
40k,150a,1	3071	12747	1031	2934	12673	196	-4.7%	-0.6%
40k,150a,8	3023	1605	714	2934	1584	196	-3.1%	-1.3%
40k,150b,1	2927	12748	427	2934	12673	196	0.2%	-0.6%
40k,150b,8	3026	1603	427	2934	1584	196	-3.1%	-1.2%
40k,450a,1	2925	37021	509	2934	37949	577	0.3%	2.4%
40k,450a,8	2931	4621	507	2934	4744	577	0.1%	2.5%
40k,450b,1	2924	37021	1531	2934	37949	577	0.3%	2.4%
40k,450b,8	2929	4629	1551	2934	4744	577	0.2%	2.4%

5. EXPERIMENTAL METHODOLOGY

Experiments are performed on an IBM QS21 Blade featuring two Cell processors (and hence 16 SPEs) running at 3.2GHz and having 4GB of RAM. The code has been compiled with GCC4.1.1 and -O3 flag. Only one PPE was used in our experiments as we intend to study scalability in the number of SPEs.

For the profiling and model validation tests, profile HMMs from the Pfam database [12] and sequence data sets from the UniProtKB/SwissProt database [5] were used. Figure 4 shows the current model and sequence length distribution for Pfam and SwissProt databases. Only the length of the profile HMMs was taken into account. For the sequence data set, the number of items in the set and the distribution of their lengths was relevant. Based on this information, input test sets have been chosen.

Profiling results were obtained by analyzing runtime traces from an instrumented version of the application. HMMERCELL was manually instrumented using the Extrae tracing library [4]. The generated traces have been inspected with Paraver [3], a visualization environment for trace files. To model application behavior, parametric functions for each phase were created. Their dependence on the input and the choice for linear or logarithmic scaling depends on theory, profiling results and inspection of the HMMER source code. Based on these equations, the formulas were parameterized by fitting the profiled performance data, using linear or logarithmic regression.

6. CELL BE PROFILING RESULTS

This section complements the HMMER scalability analysis by presenting profiling analysis. Besides reporting performance scalability on the number of workers, results also show scaling behavior with respect to input sizes. Profiling results were obtained using 5 distinct HMMs with lengths from 100 to 500 positions. For general performance tests, a sequence set consisting of 20.000 randomly selected sequences with length distribution identical to the SwissProt database was used. Figure 5 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.

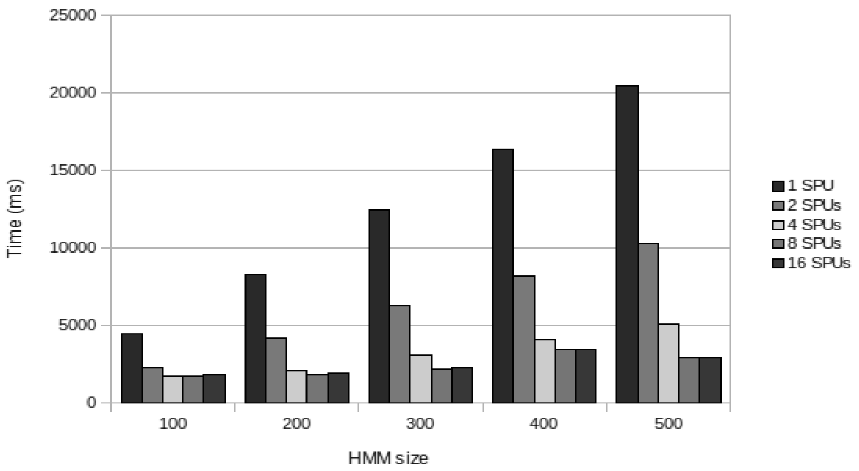

Fig. 5. HMMERCELL execution time overview.

In order to evaluate the scaling capability of HMMERCELL, the behavior of each of the important program phases (*M_BUF, W_VIT, M_PP*) was analyzed in isolation. The goal of these experiments is to understand the behavior of each phase, their dependence on the input parameters, how they contribute to aggregated HMMERCELL performance, and to understand the role and impact of various bottlenecks to scaling capability. In the following subsections, profiling results are discussed for each phase, showing their scaling behavior in HMM size and sequence length. Each phase has two graphs showing its scaling behavior. The figure on the left emphasizes scaling results in sequence length (sequence length on the horizontal axis, lines represent different HMM sizes). The figure on the right emphasizes scaling in HMM model length (HMM size on the horizontal axis, lines represent different sequence lengths). The vertical axes represent execution time.

6.1. THE PPE BUFFERING PHASE

In Figure 6, the scaling behavior of the *M_BUF* phase is shown. This phase of the program, which runs on the PPE, is responsible for loading sequences from disk into main memory, for converting them to HMMER's internal format and for creating jobs (for the SPEs) by adding the corresponding entries to the job queue. From the graphs in the figure, it is clear that the *M_BUF* computation time scales linearly in the sequence length and is independent of the HMM model size. This is in-line with expectations: loading a single HMM (even if it is a long one) takes negligible time compared to loading the many database sequences. In fact, when we talk about the *M_BUF* phase we discard the loading of the HMM. More in-depth profiling revealed that 40% of *M_BUF* time is spent on I/O while most of the rest is spent on formatting the sequences before they can be processed.

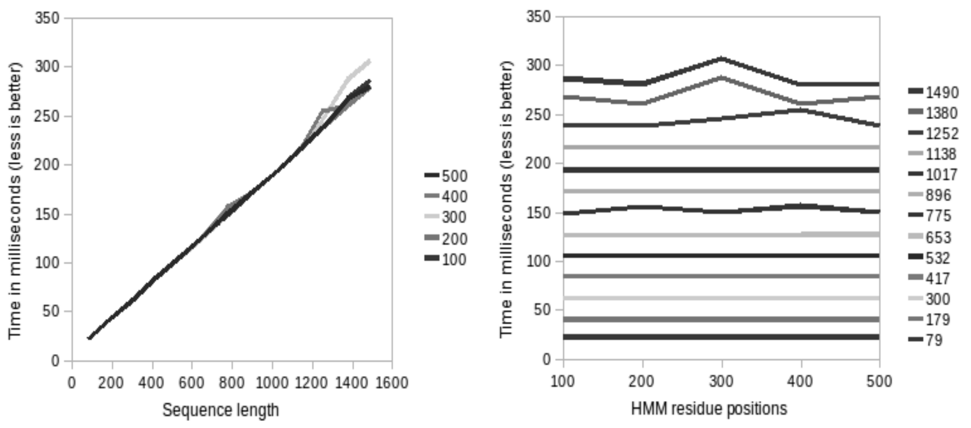

Fig. 6. Scaling characteristics of the PPE buffering phase (M_BUF).

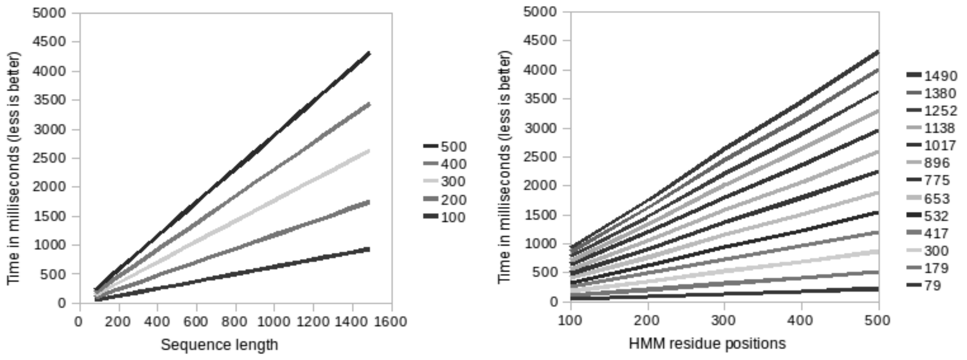

Fig. 7. Scaling characteristics of the SPE Viterbi phase (W_VIT).

Therefore, an increased I/O bandwidth and a faster formatting of sequences (either by parallelization or faster processor) would be the way of speeding this phase up.

6.2. THE SPE VITERBI PHASE

Figure 7 shows the scaling behavior of the most computationally intensive part, the *W_VIT*. During this phase, the SPEs process the PPE-created jobs in *M_BUF*. In each job, a sequence is aligned to the HMM using the Viterbi algorithm. A special version of the algorithm with smaller memory footprint is used so that all data structures fit inside the small SPEs' LS (256KB). In this version, intermediate values are discarded and only the alignment score is produced, which is sent back to the PPE.

From the figures, it is clear that *W_VIT* computation time scales linearly both in the length of the sequence and in the size of the HMM profile. Again, this confirms expectations, as the Viterbi algorithm scales linearly in sequence length and linear for models cast in profile HMM form.

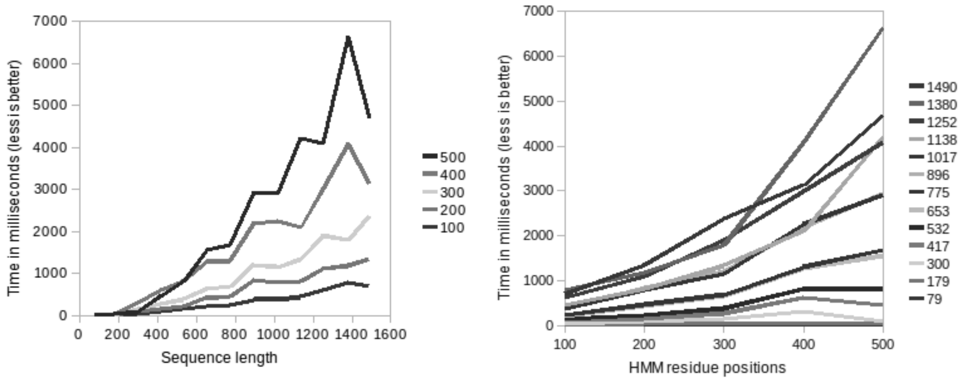

Fig. 8. Scaling characteristics of the PPE Viterbi Traceback phase (M_PP).

6.3. THE PPE TRACEBACK PHASE

Figure 8 depicts the *M_PP* scaling behavior. This phase checks the results that have been produced by the SPEs and performs the full Viterbi algorithm on the PPE for those sequences that have a high alignment score and hence might form a potential match to the model. Compared to the previous phases, *M_PP* behavior is less regular. The reason for this is that whereas *M_BUF* and *W_VIT* are performed for each sequence in the test set, *M_PP* only performs the Viterbi calculations for a subset of sequences, namely those whose alignment score exceeds a certain threshold. The actual number of tracebacks depends on the underlying biological semantics, i.e. how many sequences in the test set fit well to the model.

Behavior of *M_PP* is further analyzed by breaking it down in two components: the number of tracebacks performed for a test set (Figure 9) and the average time required for an individual traceback (Figure 10), given different sequence and HMM combinations. Of course, when the number of tracebacks is multiplied by the time per traceback, the total time spent in the traceback phase is produced.

Of these two components, the results for average time per individual trace-back are as expected: execution time for a single sequence scales more or less linearly in both sequence length and HMM size. The full Viterbi algorithm requires a large data structure in memory and traversing this memory hierarchy is the reason for the observed staggered scaling. Hence, the erratic results in total traceback time are mainly caused by the fluctuations in the number of trace- backs. Some correlation between length and traceback count can be observed: generally speaking, longer sequences result in more hits, since local alignment is performed. Subsections of a sequence are allowed to form a match to the model, hence the longer the sequence the larger the probability of a matching subsection. HMM size has no clear effect on performance.

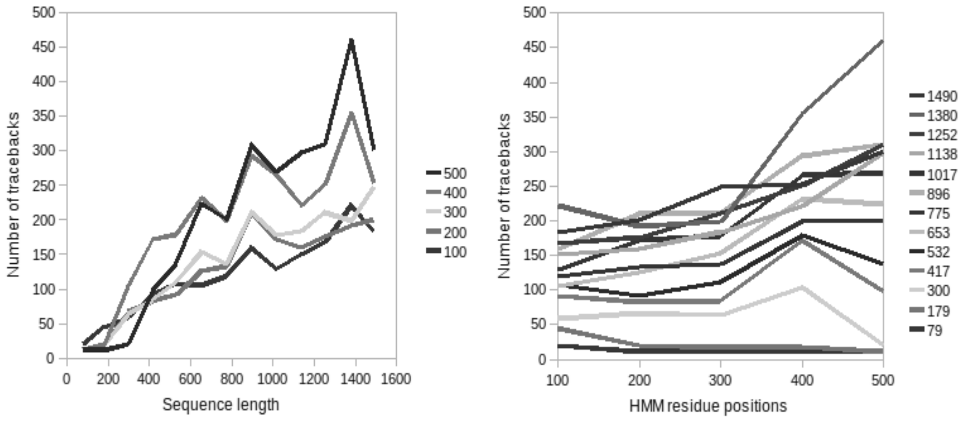

Fig. 9. M_PP analysis: total number of tracebacks.

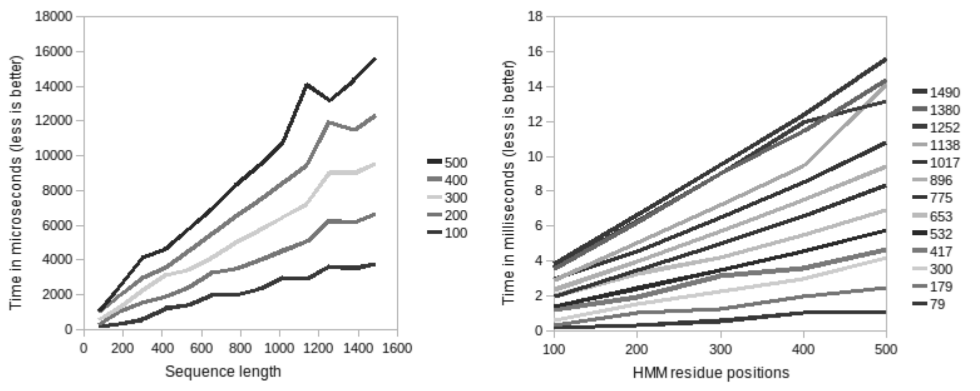

Fig. 10. M_PP analysis: average time per individual traceback.

The number of tracebacks is affected by two factors: the particular combination of HMM and sequence set, i.e. their biological match; and the number of sequences in the test set, as the E-value depends on the test set size. The number of tracebacks required varies largely between HMMs, even for those having identical length. Results between different sequence sets of equal size vary much less. An alignment requires a higher bit score to be counted as significant when comparing against a larger database. A logarithmic relationship between test set size and traceback count is present [1].

7. DISCUSSION

The *hmmsearch* kernel was parallelized according to the master-worker pattern into three stages: buffering, Viterbi processing and post-processing. The phase that performs the Viterbi calculations is the most time-consuming portion of HMMER and is primarily responsible for overall program behavior. Hence, inspection of this part and its parallelization strategy is very important. Offloading the Viterbi calculations onto the workers is effective: the workload is regular, the computation-to-communication ratio is high, and in theory the number of workers that can be efficiently used is only limited by the number of sequences. However, the master should be able to create jobs fast enough. This implies that for any given workload a certain worker count exists that will saturate the master. In this respect, our model shows that HMM model size determines how many workers can be used before the master's buffering capability is exceeded. This is explained by the fact that (as seen in Figs. 6-7) only the HMM size has a different impact on *M_BUF* and *W_VIT*. For short HMMs for instance, worker jobs are small compared to the *M_BUF* phase, resulting in the master not being able to keep up with preparing jobs. Notice that by letting the workers format the input sequences themselves would improve scalability as less work needs to be done by the master in the buffering phase.

The explicit memory architecture of the Cell BE with the SPEs having small LS requires the use of a smaller footprint Viterbi algorithm. As a consequence, the full Viterbi kernel should be included in *M_PP*. This phase is shown to be another potential bottleneck and is shown to introduce inherent uncertainty in the model. However, as the PPE buffering phase and the SPE Viterbi phase are both linearly dependent on the number of sequences in the workload, they are the most influential to overall performance. Because *M_PP* execution time becomes less significant for larger workloads, its impact on the overall model accuracy becomes negligible for realistic test sets. Overall, the model was found to be highly accurate, with only 2% error when compared to execution on real hardware.

Full Viterbi in *M_PP* can be avoided on a shared memory system. A drawback of Cell BE and heterogeneous processors with explicit memory architecture in general, is that there is a direct impact on the parallelization strategy. The advantage is of course that for suitable applications depending on their compute-intensive nature, performance can be very high. However, the ratio between master and workers has to be balanced for the target application. For HMMERCELL, we found that three SPEs saturate the PPE for typical HMM sizes. The proposed model can be used to estimate the optimal ratio between PPE and SPEs for different workloads. In general, modeling the behavioral characteristics is useful: it is a valuable aid for decision-making during design space exploration as it can show the optimal ratio between job creation and job consumption. The proposed model can also be used for scheduling at runtime.

8. CONCLUSIONS

In this paper we presented an analytical model of HMMER aimed at master-worker parallelization schemes. The model was deduced from program inspection and later compared against execution of HMMERCELL on a real Cell processor. The model and the profiling results gave us an insight in the HMMER scalability details. The model prediction for *M_BUF* and *W_VIT* phases was found to be highly accurate, with only 1.5% and 1.7% error on average. Although *M_PP* was not accurately estimated by the model, we showed that for realistic test cases it does not affect the overall prediction. Our total execution time estimation was with and error within 2%.

The findings in this paper are relevant for other bioinformatics applications as well. Most bioinformatics applications contain an abundance of coarse-grained parallelism and the master-worker pattern is a useful strategy to divide the work over multiple cores. For optimal scaling behavior, the master core should be relieved of as many other tasks as possible and control tasks should also be parallelized. In the case of a Cell BE blade, the two PPEs offer together four hardware threads that could be used to divide up the *M_BUF* work. Even better, the SPEs could take care of the sequence formatting work in *M_BUF*. However, parallelizing *M_BUF* would only speedup its sequence formatting part and the I/O bottleneck would still remain.

Although using HMMER and the Cell processor for the experiments, the study presented in this paper has a more general scope. Our ultimate goal is to understand the interaction between bioinformatics workloads and heterogeneous multicore architectures. In our future work we will analyze the new HMMER3 [2] and apply the same methodology. Based on the same core philosophy and algorithms, HMMER3 uses a three stage filtering process similar to HMMERCELL.

REFERENCES

[1] HMMER User's Guide. ftp://selab.janelia.org/pub/software/hmmer3/3.0/Userguide.pdf.

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

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

[4] Barcelona Supercomputing Center, Extrae tool user's Guide, 2011. http://www.bsc.es/ssl/apps/performanceTools/files/docs/extrae-2.1.1-userguide.pdf.

[5] UniProt: Universal Protein Resource, 2011. www.uniprot.org.

[6] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic Local Alignment Search Tool. *Journal of Molecular Biology*, 215:403–410, 1990.

[7] S. R. Eddy. Profile Hidden Markov Models. *Bioinformatics*, 14(9):755–763, 1998.

[8] 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.

[9] 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*, pages 223 –232, 2008.

[10] T. Oliver, L. Yeow, and B. Schmidt. Integrating FPGA Acceleration into HMMER. *Parallel Computing*, 34(11):681–691, 2008.

[11] 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,* pages 766–770, 2009.

[12] E. L. Sonnhammer, S. R. Eddy, and R. Durbin. Pfam: a Comprehensive Database of Protein Domain Families based on Seed Alignments. *Proteins*, 28(3):405–420, 1997.

[13] U.Srinivasan,P.-S.Chen,Q.Diao,C.-C.Lim,E.Li,Y.Chen,R.Ju,andY.Zhang. Characterization and Analysis of HMMER and SVM-RFE Parallel Bioinformatics Applications. *In IEEE International Symposium on Workload Characterization*, pages 87 – 98, 2005.

[14] J. Walters, 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*, page 6 pp., 2006.

[15] J. P. Walters, R. Darole, , and V. Chaudhary. Improving MPI-HMMER's Scalabil- ity with Parallel I/O. *In IPDPS '09: IEEE International Symposium on Parallel & Distributed Processing*, pages 1 –11, 2009..

Information about the authors:

Senastian Isaza – He is a PhD. student in the Computer Engineering Laboratory, at Delft University of Technology, the Netherlands. He graduated in Electronic Engineering at the University of Antioquia, Colombia in 2004. In 2006 he obtained an MSc. In Embedded Systems Design at the University of Lugano, Switzerland. Since 2010 he also holds a faculty position at the University of Antioquia. His research interests include multicore architectures, parallel computing and bioinformatics.

Ernst Houtgast – He obtained his MSc. degree in Computer Engineering at Delft University of technology, the Netherlands. His research interests are parallel computing, bioinformatics and finance. He is currently employed in the financial industry.

Georgi Gaydadjiev – is currently a faculty member in the Computer Engineering Laboratory at Delft University of Technology, the Netherlands. His experience includes more than 25 years in hardware and software research and development in the Industry and Academia in several European countries. His research interests include computer architecture and micro-architecture, reconfigurable computing, hardware/software co-design, embedded systems design, VLSI design, and computer systems testing.