Exploiting Fine-Grained Parallelism in the Phylogenetic Likelihood Function with MPI, Pthreads, and OpenMP: A Performance Study

Alexandros Stamatakis[1] and Michael Ott[2]

[1] The Exelixis Lab, Teaching and Research Unit Bioinformatics, Department of Computer Science, Ludwig-Maximilians-University Munich
stamatakis@bio.ifi.lmu.de, http://icwww.epfl.ch/~stamatak/

[2] Department of Computer Science, Technische Universität München
ottmi@in.tum.de, http://www.lrr.in.tum.de/~ottmi/

Abstract. Emerging multi- and many-core computer architectures pose new challenges with respect to efficient exploitation of parallelism. In addition, it is currently not clear which might be the most appropriate parallel programming paradigm to exploit such architectures, both from the efficiency as well as software engineering point of view. Beyond that, the application of high performance computing techniques and the use of supercomputers will be essential to deal with the explosive accumulation of sequence data. We address these issues via a thorough performance study by example of RAxML, which is a widely used Bioinformatics application for large-scale phylogenetic inference under the Maximum Likelihood criterion. We provide an overview over the respective parallelization strategies with MPI, Pthreads, and OpenMP and assess performance for these approaches on a large variety of parallel architectures. Results indicate that there is no universally best-suited paradigm with respect to efficiency and portability of the ML function. Therefore, we suggest that the ML function should be parallelized with MPI and Pthreads based on software engineering criteria as well as to enforce data locality.

1 Introduction

Emerging parallel multi- and many-core computer architectures pose new challenges not only for the field of Bioinformatics, since a large number of widely used applications will have to be ported to these systems. In addition, due to the continuous explosive accumulation of sequence data, which is driven by novel sequencing techniques such as, e.g., pyrosequencing [1], the application of high performance computing techniques will become crucial to the success of Bioinformatics. Applications will need to scale on common desktop systems with 2–8 cores for typical everyday analyses as well as on large supercomputer systems with hundreds our thousands of CPUs for analyses of challenging large–scale datasets. While many problems in Bioinformatics such as BLAST searches [2], statistical tests for host-parasite co-evolution [3], or computation of Bootstrap

M. Chetty, A. Ngom, and S. Ahmad (Eds.): PRIB 2008, LNBI 5265, pp. 424–435, 2008.
© Springer-Verlag Berlin Heidelberg 2008

replicates [4] for phylogenetic trees are embarrassingly parallel [5], they might nonetheless, soon require the introduction of an additional layer of parallelism, i.e., hybrid [3,6] or multi–grain [7] parallelism to handle constantly growing dataset–sizes. Moreover, for large embarrassingly parallel problems, hybrid parallelizations can potentially allow for more efficient exploitation of current computer architectures by achieving super-linear speedups due to increased cache efficiency (see Section 4 and [8]). To this end, we focus on fine–grained loop–level parallelism, which is typically harder to explore than embarrassing parallelism. We study performance of MPI-, OpenMP-, and Pthreads-based loop–level parallelism by example of RAxML [9] which is a widely used program (2,400 downloads from distinct IPs; over 5,000 jobs submitted to the RAxML web-servers) for Maximum Likelihood-based (ML [10]) inference of phylogenetic trees. The program has been used to conduct some of the largest phylogenetic studies to date [11,12].

Apart from considerable previous experience with parallelizing RAxML and mapping the phylogenetic ML function to a vast variety of hardware architectures that range from Graphics Processing Units [13], over shared memory systems [8] and the IBM Cell [7], to the SGI Altix [14] and IBM BlueGene/L [15] supercomputers, RAxML exhibits properties that make it a well-suited candidate for the proposed study: *Firstly,* the communication to computation ratio can easily be controlled by using input alignments of different lengths; *secondly* the computation of the ML function requires irregular access of floating point vectors that are located in a tree; *thirdly* the parallelization strategies described here are generally applicable to *all* ML-based programs for phylogenetic inference, including Bayesian methods.

The current study represents the first comparison of MPI, Pthreads, and OpenMP for the phylogenetic ML function, which is among the most important statistical functions in Bioinformatics. It is important to note that, despite a more demanding development process, MPI naturally enforces data locality, which might significantly improve performance on NUMA architectures and ensures portability to systems such as the IBM BlueGene.

1.1 Related Work

A previous study on the comparison of OpenMP, MPI, and Pthreads [16] focused on performance for sparse integer codes with irregular remote memory accesses. Other recent papers [17,18] conduct a comparison of OpenMP versus MPI on a specific architecture, the IBM SP3 NH2, for a set of NAS benchmark applications (FT, CG, MG). The authors show that an OpenMP-based parallelization strategy, that takes into account data locality issues, i.e., requires a higher MPI-like programming effort, yields best performance. However, such an approach reduces portability of codes. In a parallelization of a code for analysis of Positron Emission Tomography images [19] the authors conclude that a hybrid MPI-OpenMP approach yields optimal performance.

Shan *et al.* [20] address scalability issues of a dynamic unstructured mesh adaptation algorithm using three alternative parallel programming paradigms

(MPI, SHMEM, CC-SAS) on shared and distributed memory architectures and report medium scalability for an MPI-based parallelization which however provides a high level of portability.

Our study covers a larger diversity of current architectures than the aforementioned papers, in particular with respect to multi-core systems, and assesses performance of three common programming paradigms for the ML function. To the best of our knowledge, this is the first comparative study, that provides a comparison of the three programming paradigms for loop-level parallelism on multi-core architectures, cluster, and SMP architectures.

2 General Fine–Grained Parallelization Scheme

The computation of the likelihood function consumes over 90-95% of total execution time in all current ML implementations (RAxML [9], IQPNNI [21], PHYML [22], GARLI [23], MrBayes [24]). Due to its intrinsic fine-grained parallelism, the ML function thus represents the natural candidate for parallelization at a low level of granularity. Though the ML method also exhibits a source of embarrassing parallelism at a significantly more coarse–grained level [15], in our study, we exclusively focus on fine–grained parallelism, which will become increasingly important for multi–gene analyses (see [11,25] for examples) or even larger whole–genome phylogenies that can have memory requirements exceeding 16–32GB.

To compute the likelihood of a fixed *unrooted* tree topology with given branch lengths, initially one needs to compute the entries for all internal likelihood vectors that essentially reflect the probabilities of observing an `A,C,G,` or `T` at an inner node for each site of the input alignment, bottom-up towards a virtual root that can be placed into any branch of the tree.

Note that, all computations of the partial likelihood vectors towards the virtual root can be conducted independently. As outlined in Figure 1 synchronization is only required before reduction operations that are conducted by the functions that compute the overall log likelihood score of the tree or optimize branch lengths (branch length optimization not shown in Figure 1). The computation of partial likelihood array entries consumes about 75% of total execution time. Once the partial likelihood arrays have been computed, the log likelihood value can then be calculated by essentially summing up over the likelihood vector values to the left and right of the virtual root. This means, that a reduction operation is required at this point.

In order to obtain the *Maximum* Likelihood value all individual branch lengths must be optimized with respect to the overall likelihood score. For a more detailed description please refer to [5] and [10]. Note that, most current search algorithms such as GARLI, RAxML, or PHYML, do not re-optimize *all* branch lengths and do not re-compute all partial likelihood arrays after a change in tree topology but rather carry out local optimizations as outlined in Figure 1 in the neighborhood of the tree region that is affected by the change. The main bulk of all of the above computations consists of `for`-loops over the length m of the multiple sequence input alignment, or more precisely over the number m' of

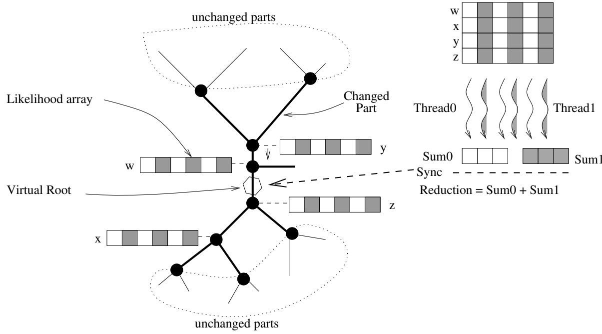


Fig. 1. Outline of the parallelization scheme used for Pthreads and MPI

distinct patterns in this alignment. The individual iterations of the `for`-loops of length m' in the functions that are used to calculate the phylogenetic likelihood function are independent and can therefore be computed in parallel, i.e., the maximum degree of parallelism is m'. This property is due to one of the fundamental assumptions of the ML model which states that individual alignment columns evolve independently [10].

3 Parallelization with OpenMP, Pthreads, and MPI

3.1 OpenMP

The parallelization of RAxML with OpenMP, is straight–forward, since only a few `pragma`'s have to be inserted into the code. Note that, in the current RAxML release (available as open-source code at `http://icwww.epfl.ch/~stamatak/`, version 7.0.4) we only parallelized the standard GTR model of nucleotide substitution [26] under the Γ model of rate heterogeneity [27]. The parallelization scheme is analogous to the concept presented in [8], however there is one fundamental difference: OpenMP might induce serious numerical problems, because the order of additions in reduction operations is non-deterministic. Given, e.g., four partial likelihood scores $l_0, ..., l_3$ from 4 threads $t_0, ..., t_3$ the order of additions to compute the overall likelihood is unspecified and can change during the inference. This behavior might cause two —otherwise exactly identical— mathematical operations to yield different likelihood scores. This has caused serious problems in RAxML with 4 threads on a large multi-gene alignment, i.e., it is not only a theoretical problem. To this end we modified the straight–forward OpenMP parallelization of the `for`-loops to enforce a guaranteed addition order for reduction operations.

While OpenMP clearly requires the lowest amount of programming overhead, it is less straight-forward to identify and resolve issues that require a higher degree of control over mechanisms such as thread affinity, memory locality, or reduction operation order.

3.2 Pthreads and MPI

The basic parallelization concept for Pthreads and MPI is analogous to the strategy for the BlueGene/L as outlined in [15]. While this parallelization mainly focused on proof–of–concept aspects and only implements the GTR+Γ model (see above) for a single version of the RAxML search algorithm, the parallelization presented here represents a complete re-implementation that covers the full functionality and plethora of models provided by RAxML (please consult the RAxML Manual for details [28]). The main goal of this re-engineering effort was to develop a single code that will scale well on multi-core architectures, shared memory machines, as well as massively parallel machines. Since the concepts devised for the Pthreads- and MPI-based parallelizations are conceptually similar we provide a joint description.

In a distributed memory scenario each of the p worker processes allocates a fraction m'/p of memory space (where m' is the number of unique columns in the alignment) required for the likelihood array data–structures which account for $\approx$ 90% of the overall memory footprint. Threads will just use an analogous portion of a global data structure. Thus the memory space and computational load for likelihood computations is equally distributed among the processes/threads and hence the CPUs. Moreover, the vector fractions m'/p are consistently enumerated in all processes, either locally (MPI) or globally (Pthreads).

The master thread/process orchestrates the distribution or assignment of data structures at start-up and steers the search algorithm as well as the computation of the likelihood scores. Thus, the master simply has to broadcast commands such as, e.g., compute likelihood array entries, given certain branch lengths, for vectors w, x, y, and z (see example in Figure 1) for the respective fraction m'/p and compute the likelihood score. In the Pthreads-based version the master thread also conducts an equally large part m'/p of the likelihood computations.

Global reduction operations, which in both cases (likelihood computation & branch length optimization) are simply an addition over m' double values, are performed via the respective MPI collective reduction operation while jobs are distributed with `MPI_Broadcast`. The Pthreads version is implemented accordingly, i.e., threads are generated only once at program start and then synchronized and coordinated via a master-thread. Job distribution and reduction operations in the Pthreads-based version are less straight-forward than with MPI, since Pthreads lack an efficient barrier method. Therefore, we implemented a dedicated function that uses a busy-wait strategy.

In contrast to the branch length optimization and likelihood computation operations, the computation of partial likelihood arrays frequently consists of a series of recursive calls, depending on how many vectors must be updated due to (local) changes in the tree topology or model parameters (see Figure 1). In order to reduce the communication frequency such series of recursive calls are transformed into a single iterative sequence of operations by the master. The master then sends the whole iterative sequence of inner likelihood vector updates that are stored by their vector numbers in an appropriate tree traversal data structure via a single broadcast to each worker or makes it available in

shared memory. Note that, in contrast to the "classic" fork-join paradigm used in OpenMP, this approach explicitly makes use of a dependency analysis of the algorithm and reduces the number of synchronization points.

An important change with respect to the previous version is the striped assignment of alignment columns and hence likelihood array structures to the individual threads of execution (see Figure 1). The rationale is that this allows for better and easier load distribution, especially for partitioned analyses of multi–gene datasets. Using a striped allocation every processor will have an approximately balanced portion of columns for each partition. Moreover, this also applies to mixed analyses of DNA and protein data, since the computation of the likelihood score for a single site under AA models is significantly more compute-intensive than for nucleotide data.

An important observation since the release of the Pthreads-based version in January 2008 is that the parallel code is used much more frequently than the previous OpenMP–based version since it compiles "out-of-the-box" on Unix/Linux and Macintosh platforms with `gcc` and allows for explicit specification of the number of threads via the command line. Such considerations are important for tools whose users are mainly non–experts. Development and maintenance experience over the last years has shown that potential users quickly abandon a tool if it requires installation of additional software and compilers. The programming effort to re-engineer RAxML and implement the Pthreads- as well as MPI-based parallelizations amounted to approximately 6 weeks.

4 Experimental Setup and Results

4.1 Test Datasets, Experimental Setup and Platforms

In order to test scalability of the three parallel versions of RAxML we extracted DNA datasets containing 50 taxa with 50,000 columns (d50_50000, 23,285 patterns) and 500 taxa with 5,000 columns (d500_5000, 3,829 patterns) from a 2,177 taxon 68 gene mammalian dataset [29]. In addition, we extracted DNA subsets with 50 taxa and 500,000 base-pairs (d50_500000, 216,025 patterns) as well as 250 taxa and 500,000 base-pairs (d250_500000, 403,581 patterns) from a large haplotype map alignment [14].

We used the Intel compiler suite version 10.1 for all three program versions on all platforms. Additionally, the platform-specific compiler optimizations flags used where identical for each of the three versions with only one exception: On the Altix interprocedural optimizations (IPO) caused a performance degradation by a factor of 3 if applied to the sequential version. Therefore we disabled these optimizations in that case. For all other compilations we enabled IPO as it slightly improved performance.

To measure the speedup we started RAxML tree searches under the GTR+Γ model on a fixed Maximum Parsimony starting tree (see [28] for details) on all platforms. Note that, we only report the best speedup values for every number of cores used on multi-core platforms with respect to the optimal thread to CPU assignment/mapping. Due to architectural issues, an execution on two cores

that are located on a single socket, can be much slower than an execution with two cores, located on two distinct sockets (see [30] for a more detailed study of thread-to-core mapping effects on performance). For instance we observed execution time differences of around 40% on the Intel Clovertown system for different assignments of two threads to the 8 cores of the system and over 50% for distinct mappings of four threads.

As test platforms we used a 2-way quad-core AMD Barcelona system (8 cores), a 2-way quad-core Intel Clovertown system (8 cores), an 8-way dual-core Sun x4600 system (16 cores) that is based on AMD Opteron processors. We measured execution times for sequential execution as well as parallel execution on 2, 4, 8, and 16 (applies only to x4600) cores. In addition, we used a cluster of 4-way SMP (4 single cores) 2.4 GHz AMD Opteron processors, that are interconnected via Infiniband, to test scalability of the Pthreads-, OpenMP-, and MPI-based versions up to 4 CPUs, and up to 128 CPUs (127 worker processes) for the MPI-based version. Finally, we used an SGI Altix 4700 system with a total of 9,728 Intel Itanium2 Montecito cores, an aggregated peak performance of 62.3 Teraflops, and 39 Terabyte of main memory (the HLRB2 supercomputer at the Leibniz Rechenzentrum, `http://www.lrz-muenchen.de/services/compute/hlrb`). On the SGI Altix we assessed scalability of the Pthreads- and OpenMP-based versions up to 32 CPUs and up to 256 CPUs for the MPI-based version.

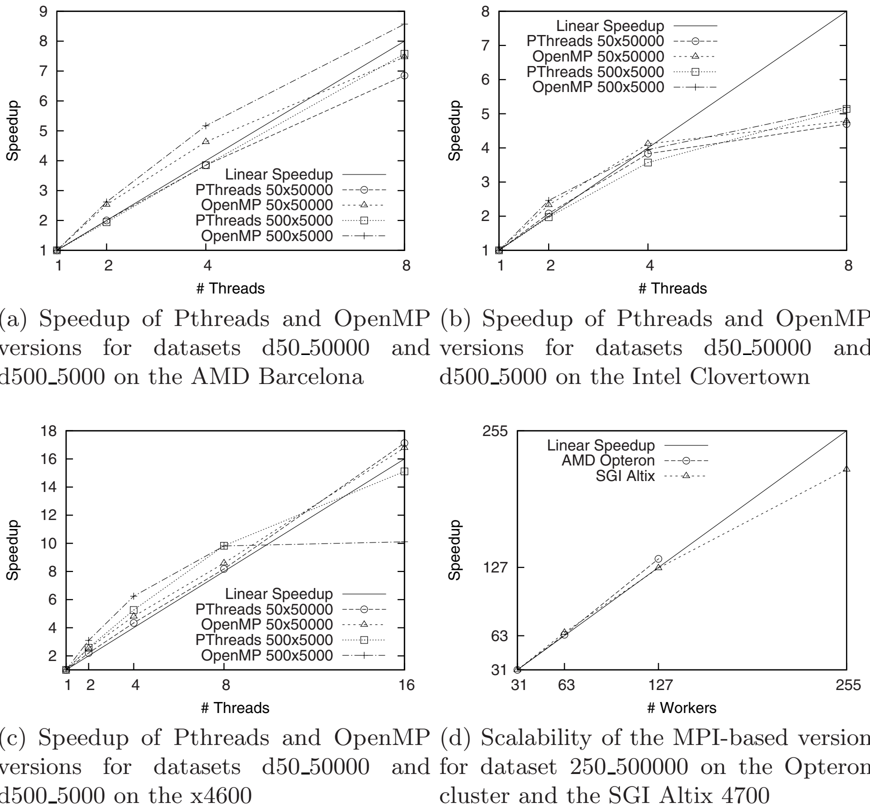
As outlined above we directly compare MPI, Pthreads, and OpenMP on the SGI Altix and AMD Opteron cluster. In addition, we provide comparisons between OpenMP and Pthreads on the Barcelona, Clovertown, and x4600 systems.

4.2 Results

In Figures 2(a) through 2(c) we indicate speedup values for the Pthreads and OpenMP versions on datasets d50_50000 and d500_5000 on the three multi-core systems: Barcelona, Clovertown, and x4600. We show results for these two datasets because they differ significantly in their computation to communication ratio, i.e., this ratio is approximately 100 times less favorable for dataset d500_5000. Note that, the memory footprint of dataset d500_5000 is about twice as high as for d50_50000. On the Barcelona both versions scale almost linearly up to 8 cores, while there is a significant decrease in parallel efficiency on the Clovertown. This is due to the UMA architecture and the L2 cache which is shared between each two cores: the memory bandwidth can be saturated by only 4 threads and the cache available to each thread is halfed if all cores are utilized. Both Pthreads and OpenMP scale well up to 16 cores on the x4600. However, there is a significant decrease in parallel efficiency for the OpenMP-based version on the more communication-intensive dataset d500_5000 above 8 cores. Thus, the Pthreads-based communication mechanisms we implemented, yields significantly better speedups for the full 16 cores on this system.

In Figure 2(d) we provide the *relative* speedups (relative with respect to a run with 31 worker processes [1]) for the MPI version on the large and memory-intensive

[1] A sequential execution was not possible due to run-times and memory requirements.



(a) Speedup of Pthreads and OpenMP versions for datasets d50_50000 and d500_5000 on the AMD Barcelona

(b) Speedup of Pthreads and OpenMP versions for datasets d50_50000 and d500_5000 on the Intel Clovertown

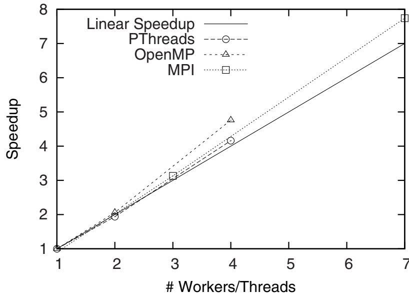
(c) Speedup of Pthreads and OpenMP versions for datasets d50_50000 and d500_5000 on the x4600

(d) Scalability of the MPI-based version for dataset 250_500000 on the Opteron cluster and the SGI Altix 4700

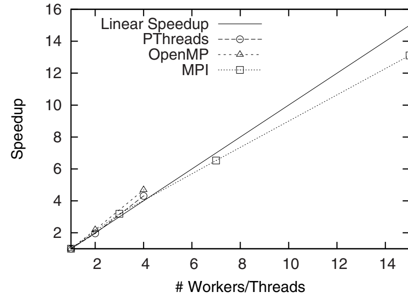
Fig. 2. Scalability of Pthreads, OpenMP, and MPI versions on various architectures

d250_500000 dataset up to 128 CPUs of the AMD Opteron cluster and up to 256 cores on the SGI Altix 4700. This plot demonstrates that the entirely re-designed MPI version for production runs achieves similar parallel efficiency as the previous proof-of-concept implementation [14,15].

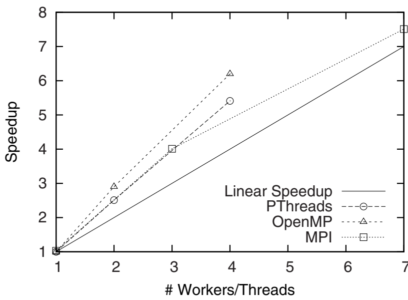
In Figures 3(a) to 3(d) we provide a direct comparison of the Pthreads, OpenMP, and MPI versions for the AMD Opteron cluster and the SGI Altix 4700. Speedup values for dataset d50_50000 on the Opteron cluster (Figure 3(a)) are super-linear due to increased cache efficiency for all three programming paradigms. On the larger d50_500000 dataset (Figure 3(b)) scalability of OpenMP and Pthreads are similar, while the MPI-based version yields slightly sub-linear speedups on the Opteron cluster for 7 and 15 worker processes. This is due to the fact that execution times in these cases become relatively short, such that the initial sequential portion of the code (striped data distribution) has an impact on performance. The more communication–intensive dataset d500_5000 also scales well for all three paradigms on the AMD Opteron system (Figure 3(c)). In general, the OpenMP version scales best on this system. However, this is not the case on all architectures, especially for more communication-intensive datasets sizes (see Figure 2(c)).



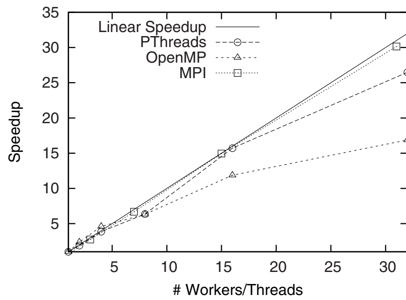
(a) Scalability of the Pthreads, OpenMP, and MPI versions for dataset d50_50000 on the Opteron cluster



(b) Scalability of the Pthreads, OpenMP, and MPI versions for dataset d50_500000 on the Opteron cluster



(c) Scalability of the Pthreads, OpenMP, and MPI versions for dataset d500_5000 on the Opteron cluster



(d) Scalability of the Pthreads, OpenMP, and MPI versions for dataset d50_50000 on the SGI Altix 4700

Fig. 3. Performance Comparison of Pthreads, OpenMP, and MPI versions

Both MPI and Pthreads also yield super-linear speedups in most cases. Finally, in Figure 3(d) we provide performance data for all three parallel versions on the SGI Altix for dataset 50_50000 up to 31 worker processes/threads. The MPI and Pthreads versions scale significantly better than OpenMP for more than 7 threads/workers which is also consistent with the observations on the x4600 (see Figure 2(c)). For more than 15 threads/workers, MPI outperforms Pthreads as the MPI version only accesses local memory while the Pthreads version has to access most of its data structures remotely at the master – with lower bandwidth and higher latency.

5 Conclusion and Future Work

We have conducted a detailed performance study of parallel programming paradigms for exploitation of fine–grained loop–level parallelism, by example of the widely used phylogenetic ML function as implemented in RAxML on a broad variety of current multi-core, cluster, and supercomputer architectures. Results

indicate that none of the three paradigms outperforms the others across all architectures. We thus conclude that the selection of programming paradigms should be based on software engineering and portability criteria.

One important aspect is that Bioinformatics applications are typically used by non-experts such that the easier to compile Pthreads option should be preferred over OpenMP and MPI for shared memory architectures. The usage of MPI on shared memory machines could lead to serious performance degradations in the case that MPI implementations are used that have not been optimized for communication via shared memory.

In terms of portability, we argue in favor of the usage of both Pthreads and MPI, since programs can easily be compiled for massively parallel machines such as the BlueGene as well as for shared memory architectures. Note that, we do not consider hybrid parallelism here because, as already mentioned, ML-based inferences exhibit embarrassing parallelism at a more coarse-grained level. In addition, Pthreads allow for explicit allocation of local memory, i.e., to distribute the data structures and thus facilitate the joint development and maintenance of the MPI and Pthreads versions. In this case, synchronization and communication can be handled via one single generic interface that can then be mapped to appropriate MPI or Pthreads constructs and greatly reduce the complexity of the code. Moreover, such an —in principle— distributed memory Pthreads-based parallelization can improve performance on NUMA architectures. A striped distribution of alignment sites, which is required to achieve load–balance on concatenated DNA and AA (Protein) data would induce a significant programming overhead in OpenMP as well. Our experiments show that the performance of the Pthreads-based and OpenMP-based implementations is platform–specific, such that one should opt for the more generic approach. Finally, the Pthreads-based version can be further improved by removal of some synchronization points and exploitation of data locality.

Thus, future work will cover the performance analysis and profiling of data locality impact for a Pthreads–based version that allocates and uses local instead of global likelihood vector data structures.

Acknowledgements

We are grateful to Olaf Bininda-Emonds, Dan Janies, and Andrew Johnson for providing us their alignments for performance tests. We would like to thank Dimitris Nikolopoulos for useful discussions on previous comparative performance studies for different programming paradigms. The Exelixis Lab (AS) is funded under the auspices of the Emmy-Noether program by the German Science Foundation (DFG).

References

1. Hamady, M., Walker, J., Harris, J., Gold, N., Knight, R.: Error-correcting barcoded primers for pyrosequencing hundreds of samples in multiplex. Nature Methods 5, 235–237 (2008)

2. Darling, A., Carey, L., Feng, W.: The Design, Implementation, and Evaluation of mpiBLAST. In: Proceedings of ClusterWorld 2003 (2003)
3. Stamatakis, A., Auch, A., Meier-Kolthoff, J., Göker, M.: Axpcoords & parallel axparafit: Statistical co-phylogenetic analyses on thousands of taxa. BMC Bioinformatics (2007)
4. Felsenstein, J.: Confidence Limits on Phylogenies: An Approach Using the Bootstrap. Evolution 39(4), 783–791 (1985)
5. Bader, D., Roshan, U., Stamatakis, A.: Computational Grand Challenges in Assembling the Tree of Life: Problems & Solutions. In: Advances in Computers. Elsevier, Amsterdam (2006)
6. Minh, B.Q., Vinh, L.S., Schmidt, H.A., von Haeseler, A.: Large maximum likelihood trees. In: Proc. of the NIC Symposium 2006, pp. 357–365 (2006)
7. Blagojevic, F., Nikolopoulos, D.S., Stamatakis, A., Antonopoulos, C.D.: Dynamic Multigrain Parallelization on the Cell Broadband Engine. In: Proceedings of the 12th ACM SIGPLAN symposium on Principles and practice of parallel programming, pp. 90–100 (2007)
8. Stamatakis, A., Ott, M., Ludwig, T.: RAxML-OMP: An Efficient Program for Phylogenetic Inference on SMPs. In: Malyshkin, V.E. (ed.) PaCT 2005. LNCS, vol. 3606, pp. 288–302. Springer, Heidelberg (2005)
9. Stamatakis, A.: RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. Bioinformatics 22(21), 2688–2690 (2006)
10. Felsenstein, J.: Evolutionary trees from DNA sequences: a maximum likelihood approach. Journal of Molecular Evolution 17, 368–376 (1981)
11. Dunn, C.W., Hejnol, A., Matus, D.Q., Pang, K., Browne, W.E., Smith, S.A., Seaver, E., Rouse, G.W., Obst, M., Edgecombe, G.D., Sorensen, M.V., Haddock, S.H.D., Schmidt-Rhaesa, A., Okusu, A., Kristensen, R.M., Wheeler, W.C., Martindale, M.Q., Giribet, G.: Broad phylogenomic sampling improves resolution of the animal tree of life. Nature (2008) (advance on-line publication)
12. Robertson, C.E., Harris, J.K., Spear, J.R., Pace, N.R.: Phylogenetic diversity and ecology of environmental Archaea. Current Opinion in Microbiology 8, 638–642 (2005)
13. Charalambous, M., Trancoso, P., Stamatakis, A.: Initial Experiences Porting a Bioinformatics Application to a Graphics Processor. In: Bozanis, P., Houstis, E.N. (eds.) PCI 2005. LNCS, vol. 3746, pp. 415–425. Springer, Heidelberg (2005)
14. Ott, M., Zola, J., Aluru, S., Johnson, A.D., Janies, D., Stamatakis, A.: Large-scale Phylogenetic Analysis on Current HPC Architectures. Scientific Programming (Submitted, 2008)
15. Ott, M., Zola, J., Aluru, S., Stamatakis, A.: Large-scale Maximum Likelihood-based Phylogenetic Analysis on the IBM BlueGene/L. In: Proceedings of IEEE/ACM Supercomputing Conference 2007 (2007)
16. Berlin, K., Huan, J., Jacob, M., Kochhar, G., Prins, J., Pugh, B., Sadayappan, P., Spacco, J., Tseng, C.: Evaluating the Impact of Programming Language Features on the Performance of Parallel Applications on Cluster Architectures. In: Rauchwerger, L. (ed.) LCPC 2003. LNCS, vol. 2958. Springer, Heidelberg (2004)
17. Cappello, F., Etiemble, D.: MPI versus MPI+ OpenMP on the IBM SP for the NAS Benchmarks. In: Proc. Supercomputing 2000, Dallas, TX (2000)
18. Krawezik, G., Alleon, G., Cappello, F.: SPMD OpenMP versus MPI on a IBM SMP for 3 Kernels of the NAS Benchmarks. In: Zima, H.P., Joe, K., Sato, M., Seo, Y., Shimasaki, M. (eds.) ISHPC 2002. LNCS, vol. 2327. Springer, Heidelberg (2002)

19. Jones, M., Yao, R.: Parallel programming for OSEM reconstruction with MPI, OpenMP, and hybrid MPI-OpenMP. Nuclear Science Symposium Conference Record, 2004 IEEE 5 (2004)
20. Shan, H., Singh, J., Oliker, L., Biswas, R.: A Comparison of Three Programming Models for Adaptive Applications on the Origin2000. Journal of Parallel and Distributed Computing 62(2), 241–266 (2002)
21. Minh, B.Q., Vinh, L.S., von Haeseler, A., Schmidt, H.A.: pIQPNNI: parallel reconstruction of large maximum likelihood phylogenies. Bioinformatics 21(19), 3794–3796 (2005)
22. Guindon, S., Gascuel, O.: A Simple, Fast, and Accurate Algorithm to Estimate Large Phylogenies by Maximum Likelihood. Systematic Biology 52(5), 696–704 (2003)
23. Zwickl, D.: Genetic Algorithm Approaches for the Phylogenetic Analysis of Large Biological Sequence Datasets under the Maximum Likelihood Criterion. PhD thesis, University of Texas at Austin (April 2006)
24. Ronquist, F., Huelsenbeck, J.: MrBayes 3: Bayesian phylogenetic inference under mixed models. Bioinformatics 19(12), 1572–1574 (2003)
25. McMahon, M.M., Sanderson, M.J.: Phylogenetic Supermatrix Analysis of GenBank Sequences from 2228 Papilionoid Legumes. Systematic Biology 55(5), 818–836 (2006)
26. Tavar, S.: Some Probabilistic and Statistical Problems in the Analysis of DNA Sequences. Some Mathematical Questions in Biology: DNA Sequence Analysis 17 (1986)
27. Yang, Z.: Maximum likelihood phylogenetic estimation from DNA sequences with variable rates over sites. Journal of Molecular Evolution 39, 306–314 (1994)
28. Stamatakis, A.: The RAxML 7.0.4 Manual, The Exelixis Lab. LMU Munich (April 2008)
29. Bininda-Emonds, O., Cardillo, M., Jones, K., MacPhee, R., Beck, R., Grenyer, R., Price, S., Vos, R., Gittleman, J., Purvis, A.: The delayed rise of present-day mammals. Nature 446, 507–512 (2007)
30. Ott, M., Klug, T., Weidendorfer, J., Trinitis, C.: Autopin - Automated Optimization of Thread-to-Core Pinning on Multicore Systems. In: Proceedings of 1st Workshop on Programmability Issues for Multi-Core Computers (MULTIPROG) (January 2008)