On the Migration of the Scientific Code Dyana from SMPs to Clusters of PCs and on to the Grid

Michela Taufer[1], Thomas Stricker[1], Gérard Roos [1], Peter Güntert [2]

[1] Department of Computer Science
ETH Zentrum
CH-8092 Zurich, Switzerland
stricker,taufer@inf.ethz.ch

[2] Institute for Molecular Biology and Biophysics
ETH Hoenggerberg
CH-8092 Zurich, Switzerland
guentert@mol.biol.ethz.ch

Abstract

Dyana is a molecular biology code used in the study of infectious prion proteins. Like many other scientific codes, Dyana was migrated successfully from vector supercomputers to a more cost-effective cluster of commodity PCs. A further migration to a widely distributed grid computing platform looks very tempting because many of these platforms promise the use of nearly free compute-cycles on the Internet.

Not all codes are equally suited for all platforms. Even embarrassingly parallel codes might require a significant re-engineering effort for a migration from one platform to another. A better understanding of the performance characteristics of a code is required before a migration is attempted.

To address this problem, we present a systematic method to study the viability of a code migration from one platform to another. We construct an analytic performance model of the application. We use the previous migration from SMPs to commodity clusters of PCs to validate and calibrate the model. Finally, we extrapolate the performance of Dyana to new platforms including widely distributed computing on the grid and we suggest optimizations in the process of migration.

We demonstrate our method with the molecular biology code Dyana. In particular, our general model predicts that Dyana can efficiently use up to 42000 processors with its current workload and is therefore well suited for grid computing on the Internet.

Keywords: *performance evaluation and modeling, computation-intensive applications, widely distributed supercomputing, migration of scientific codes, software engineering, grid architectures.*

1 Introduction

Rapid changes in technology lead to ever cheaper and more powerful platforms for high performance computing. The computational power and the machine characteristics found in supercomputers of yesterday are already available in high-end workstations today and will be available in consumer devices in every household tomorrow. This trend leads to the vision of a computational grid comprising millions of computing devices worldwide, connected by the Internet.

The focus in high performance computing has shifted from installations with maximal performance to installations with the better price-performance ratios. At the same time, with the rapid expansion of the Internet, many powerful PCs are networked by some fairly good interconnects. In theory, these machines can supply thousands of Megaflops in compute performance to computational scientists, if they are used during their idle times and a mode of operation to deliver their spare compute cycles is created. Besides early academic prototypes [13, 14], several companies are developing middleware, toolkits and business models for *widely distributed computing on the Internet* [1, 4]. In some cases a newly proposed business model of "good cause computing" suggests that users donate the spare compute cycles to science.

There exists a fair number of useful computations that were once brand-marked as embarrassingly parallel and therefore uninteresting. Indeed, the performance characteristics of those applications are quite boring if they are run on high-end supercomputers. Still, those applications do computations that are highly useful to the advancement of science once they can be deployed more widely and executed on ever cheaper resources. Furthermore, a closer look to some examples may quickly reveal that there are quite different levels of embarrassingly parallel and that not all of those codes can be mi-

grated equally well to newer and cheaper platforms.
Code migrations from vectorizable codes to message passing programs require some well-known transformations. In most cases, a successful migration involves some significant re-engineering of the code. The control and data transfer primitives need to be changed from multi-threading with shared memory primitives to calls for message passing libraries or even back to calls handling TCP/IP streams for widely distributed computing on the Internet. Unfortunately, these transformations are often done without a precise cost-benefit model in mind and without a prior performance characterization of the code.
In this paper, we propose a systematic way to check the viability of a migration from clusters of PCs to a framework of widely distributed computing well in advance and demonstrate it with the highly parallel molecular biology code, Dyana [6]. Dyana computes three-dimensional protein and nucleic acid structures by energy minimization in a simulated annealing process and is highly valuable to the investigation of Prion related diseases like the Bovine Spongiform Encephalopathy (BSE), the mad cow disease. The planned use of Dyana fits the idea of good cause computing and so it could be a good candidate to exploit free compute resources donated by Internet users.
For our investigation we rely on the performance data gathered during a successfully completed migration from SMPs to clusters of commodity PCs [11]. The performance data of those experiments is used to calibrate a detailed analytical model describing the precise performance characteristics of the code (CPU dependency, memory system usage, I/O requirements) for a variety of node architectures and networks, like uni-processor and dual-processor Pentium III nodes at 400, 800 and 1000 MHz. We also included several different memory systems (i.e. PC motherboards) and one completely different architecture using a DEC Alpha based cluster with a dedicated high speed interconnect [3]. Once our performance model is calibrated, it extends to the planned migration to widely distributed computing on the Internet.
In Section 2 we take a closer look at the Dyana application code and its functions. In Section 3 we discuss the migration path and the different mechanisms for control and data transfers used on the different platforms. In Section 4 we establish and calibrate the performance model used to guide the migration of Dyana. This leads to the discussion about viability and the expected performance on the new platforms in Section 5.

2 The Protein Structure Calculation Code Dyana

The application code Dyana was written at the Institute of Molecular Biology and Biophysics of ETH Zurich [6]. It calculates three-dimensional protein and nucleic acid structures from distance constraints and torsion angle constraints collected by nuclear magnetic resonance (NMR) experiments [5]. The computation is based on simulated annealing driven by the fast torsion angle dynamics (TAD) algorithm [7].
The simulated annealing is repeated several times, each time starting with different random initial values of the degrees of freedom, the torsion angles. This independent generation of many conformers introduces a high degree of inherent parallelism into the structure calculation with Dyana.
Dyana is written for a master-slave setting. The master coordinates the parallel distribution and calculation. The slaves run the simulations and return the results (i.e., the resulting conformer) to the master that collects and analyzes them.

2.1 Task Parallelism in the SMP Version

The original version of Dyana runs on shared-memory multiprocessors (SMPs). The parallelism in the computation of the conformers has been reached by means of multiple independent processes spawned by a UNIX fork() system call. The master process creates the slave processes calling fork() and implicitly sends the data to them, according to the standard UNIX semantics for process creation. Since no information is exchanged during the computation of conformers, no additional shared memory data structures are necessary.
Each slave computes exactly one conformer. Once a slave has finished its computation, it sends the conformer back to the master and exits. The slaves return their results to the master through the file system.
The master is informed by the operating system upon the termination of a slave (child-termination signal). Although the master has to know when a slave exits, it does not need to know which particular slave has finished. A counter, which increases upon forking and decreases upon receiving a child-termination signal tells the master when a parallel part of the computation is finished. Once all conformer simulations have been completed, the master gathers the data about protein structures by reading the corresponding files and continues with the sequential execution of a subsequent evaluation part.
As long as there are more structures to compute, the master spawns slaves. At the same time, the master ensures that the number of slaves does not exceed the number of available processors.

2.2 Managing Computation Steps and Tasks with INCLAN

The Dyana software system provides a collection of functions implementing the algorithms for NMR structure calculation. The user arranges these functions to obtain a sequence of commands that computes a protein

structure from its experimental NMR data. This high degree of flexibility has been achieved by integrating INCLAN (INteractive Command LANguage) [5] into Dyana. INCLAN is an interpreted command language, offering control logics like if-then-else commands, loop constructs, ordinary variable assignment and arithmetic operations to control the search for the molecule structure with minimal energy. Therefore, all common strategies for the generation of conformers (work-units or tasks) can implemented in Dyana.

While the flexibility of INCLAN allows several different tasking and scheduling models of the computation, we used for our modeling uses a fairly rigid tasking model. Each computation phase starts with an initial conformer structure and generates a series of random variations that are considered for energy minimization at this step. Each variation of the conformer is handled by INCLAN like a work queue in the master.

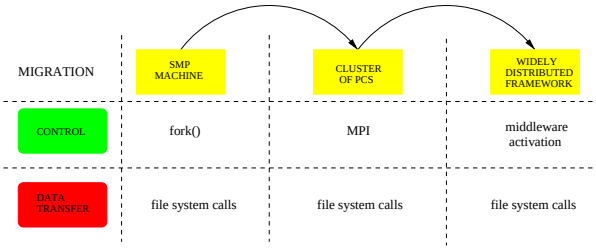

Figure 1. Control and data transfer mechanisms used in the migration path from SMP machine, to cluster of PCs and on to a framework for widely distributed computing. While in the migration we re-engineer the control mechanisms from fork() calls on SMPs, to MPI commands on clusters and middleware library calls on frameworks for widely distributed computing. We maintain the same data transfer mechanism (file system calls) for all architectures.

2.3 Characteristics of the SMP Version

The fork() system call used for task creation simplifies the code implementation, since the entire address space of the master is replicated to the slave. The process on the slave only lives until its computation completes and there is no need to reset some state or keep the state of the master and the slaves consistent.

On the other hand, forking a process can become an expensive task unless shared memory with copy on write is used. If we look at the fork() system call as an implicit send command that transmits, the whole address space of a process the potential cost becomes obvious. Furthermore, the distributed operating systems on cluster nodes do not provide the replication of an entire address space across different machines and the fork() call cannot be used for task creation. Therefore, the original SMP version of Dyana needs some re-engineering to run on clusters or widely distributed platforms.

3 Migration of the Code from SMPs to Clusters and to Distributed Platforms

The migration of a code from one platform to another could be done with a minimal amount of re-engineering. For a migration of Dyana from SMPs to strongly interconnected cluster of PCs and further on to frameworks for widely distributed computing, we separate the control mechanism (i.e. the task activation and task synchronization) from the data transfer mechanism (i.e. the exchange of the initial molecular state and the computed 3D structure at the end). Control and data transfers are adapted separately to the new platform. The options chosen for each platform in the migration path are shown in Figure 1.

In the migration from an SMP to a cluster, we change the control mechanisms from fork() calls implemented in the command language INCLAN to MPI primitives for task activation [12]. In frameworks for widely distributed computing, the task activation will be delegated to the appropriate middleware primitive using the library calls provided for this purpose. The additional synchronization primitives for the coordination of the initial and final data transfers are handled in a similar manner.

To manage the data transfers, we use the file system calls provided by the underlying memory mapped files on SMP machines, by the networked file system NFS for clusters of PCs or by the calls of the toolkit in widely distributed computing. Due to the small amount of data communicated, this can be done with a reasonable efficiency on all platforms considered. We acknowledge that there must be an appropriate security concept for each platform. Controlling the access to the slave computers and the integrity of data transfers is of a lesser interest to a performance study. Good solutions have been proposed in the past literature and we assume that the problem is solved in the middleware packages mentioned in the introduction [1, 4].

For the migration to a widely distributed computing platform, the tasking model of a typical Dyana computation has to be reviewed critically. To provide the necessary parallelism the generation of new conformers must be done automatically and span more than one generation of alternatives. As with all optimization problems, the problem of getting trapped in local minima must be addressed carefully. The code needs to be adapted to a proper workstealing paradigm as used in CILK [2].

The job queuing in the master and the work stealing scheduler can easily handle different speeds of different machines and the model can be extended to handle different classes of machines. For the total execution time only the total of compute power remains relevant, provided that the speed of a machine justifies the cost of networking it to a grid. Dyana computations are highly

fault tolerant by their nature. Since variations of the conformers are generated by a random generator the infrequent loss of a task can be tolerated easily.

4 Performance Model of Dyana

The purpose of our study is to get an idea whether Dyana can run on on a more cost-effective platform in an efficient way. To answer this question we design and validate a simple but effective analytical model which lets us roughly calculate the execution time for different application parameters, different number of processors and different platforms.

4.1 Model Design

As outlined in Section 2, the code Dyana simulates and evaluates the different alternatives of three-dimensional molecular structures, called conformers. For each of them, Dyana performs:

- the creation of an initial structure with random values for the dihedral angles,
- the minimization of violations of the conformational constraints by simulating annealing using torsion angle dynamic,
- the evaluation of the three dimensional structure resulted by checking it against spectroscopic data.

For a run of n conformers on p processors, the total execution time, t_{dyana}, can be split into three terms:

$$t_{dyana} = t_{init} + t_{siml} + t_{comm} \quad (1)$$

where:

- t_{init} is the time spent to create and initialize the parameters needed during the whole simulation,
- t_{siml} is the time used to run the simulations of annealing using torsion angle dynamics,
- t_{comm} is the communication time during which the several slaves communicate with the master and vice versa.

Since t_{init} is constant for the whole simulation and irrelevant if compared with the other time components, we ignore its contribution.
The simulation is done in two phases. First, a random structure is created, then the simulation of annealing takes place. t_{siml} for n conformers on p processors is given by:

$$t_{siml} = \left\lceil \frac{n}{p} \right\rceil \left(t_{create\_structure} + t_{conf\_siml} \right) \quad (2)$$

where:

- $t_{create\_structure}$ is the time spent for creating the random structure,
- t_{conf_siml} is the time spent minimizing the violations of the conformational constraints by simulating annealing.

Since t_{conf_siml} is the most time consuming and $t_{create\_structure}$ is less than 5% of t_{siml}, we consider:

$$t_{siml} \approx t_{conf\_siml} \quad (3)$$

t_{conf_siml} is given by:

$$t_{conf\_siml} = \frac{FlOp_{conf}}{FlPt_{speed}} \quad (4)$$

where:

- $FlOp_{conf}$ is the total amount of floating point operations per conformer. $FlOp_{conf}$ is mostly determined by the protein structure considered, but it can also be a code factor and can depend on the compiler technology. Table 1 reports the amount of $FlOp_{conf}$, for both the molecules considered in our study, the prion hPrP(R220K) and the ER2 proteins, on Pentium and Alpha architectures. The different number of floating point operations on the two platforms is related to the different compilers used and the different standard libraries called for transcendal functions.

Protein	Architecture	Compiler	$FlOp_{conf}$
hPrP	Pentium	PGI Compiler	6.5 GFlOp
hPrP	Alpha	DEC Compiler	5.3 GFlOp
ER2	Pentium	PGI Compiler	1.6 GFlOp
ER2	Alpha	DEC Compiler	1.4 GFlOp

Table 1. Total number of observed floating point operations ($FlOp_{conf}$) per conformer simulation for a prion hPrP and a ER2 on Pentium and Alpha platforms. The different number of floating point operation on the two different platforms is due to the different compilers.

- $FlPt_{speed}$ is the amount of floating point operations per second. It depends on architecture parameters like CPU rate and memory characteristics.

In our model, $FlPt_{speed}$ is approximated as:

$$FlPt_{speed} = \frac{CPU\_clock\_rate}{clocks\_per\_FlOp} \quad (5)$$

In Formula 5, the CPU_clock_rate is the frequency of the CPU. The clocks per floating point operation, $clocks\_per\_FlOp$, are further related to the CPU frequency and the memory characteristics:

$$clocks\_per\_FlOp = f(CPU\_speed, memory\_characteristics) \quad (6)$$

The CPU is characterized simply by its clock speed. Despite all architectural differences like super-scalar execution, different handling of hazards and dependencies interfering with the pipelines, we observed that on most current microprocessors one floating point instruction can be successfully completed for every clock cycle. To characterize the memory system, we use the Extended Copy Transfer Characterization (ECT) [10, 9]. Our model considers the average amount of cycles that are spent by one floating point operation when it has to fetch its data from memory. The total number of cycles used in an application run is estimated based on a detailed memory characterization (MBytes per second) taken from the ECT micro-benchmarks. We consider two different kind of memory accesses: contiguous and non-contiguous/strided access. We calculate the coefficients of the approximation by a linear regression model:

$$clocks\_per\_FlOp = a\alpha + b\beta + c\gamma \qquad (7)$$

In Formula 7, α is the amount of cycles needed for a floating point operation without any memory access, β is the amount of cycles needed for a floating point operation which has to load a word from memory continuously, while γ is the amount of cycles needed for a floating point operation which loads a word from memory using strided access. Table 2 shows the values of α, β, γ estimated using the ECT model for the several platforms considered.

CPU Type	α	β		γ	
		CPUs		CPUs	
		1	2	1	2
Pentium III 933 MHz	1	13	18	33	75
Pentium III 800 MHz	1	17	21	26	50
Pentium II 400 MHz	1	11	14	29	40
DEC Alpha 667 MHz	1	7	7	34	34

Table 2. Values of α, β and γ estimated using the ECT memory system model for Pentium and Alpha platforms.

The parameters *a*, *b* and *c* characterize Dyana and are chosen using the least-squares criterion [8] for all different machines, molecular structures and runs of Dyana. So on average, *a* represents the amount of operations which do not need any memory load, *b* is the number of floating point operations which access memory blocks continuously, while *c* is the amount which access the memory non-continuously.

At the end of each conformer simulation, the slaves send the computed three-dimensional molecular structure back to the master writing the data using file system calls and remain ready for a new simulation. In case of symmetrical networks, the time for the communication between master and slaves, t_{comm_conf}, depends on:

- the size of the data, $size_{data}$, that is exchanged,
- the number of processors, p,
- the bandwidth, $bandwidth_{master}$, that the master can sustain serving many slaves and
- the bandwidth, $bandwidth_{slave}$, that a single slave handle.

$$t_{comm\_conf} = max\left(\frac{size_{data}}{bandwidth_{slave}}, \frac{p \cdot size_{data}}{bandwidth_{master}}\right) \qquad (8)$$

The time for the whole communication becomes:

$$\begin{aligned} t_{comm} &= \left\lceil \frac{n}{p} \right\rceil t_{comm\_conf} = \\ &\left\lceil \frac{n}{p} \right\rceil max\left(\frac{size_{data}}{bandwidth_{slave}}, \right. \\ &\left. \frac{p\, size_{data}}{bandwidth_{master}}\right) \end{aligned} \qquad (9)$$

On strongly interconnected cluster platforms which normally consists of a small number of nodes connected and a large bandwidth (1000BaseT), the communication does not play any relevant role for the overall performance. On frameworks for widely distributed computing, it is either the available bandwidth of the slaves or the bandwidth of the master which limits the scalability of a parallel computation. The analytical model becomes:

$$\begin{aligned} t_{dyana} &= \left\lceil \frac{n}{p} \right\rceil \left(\frac{FlOp_{conf}}{CPU\_clock\_rate}(a\alpha + b\beta + c\gamma) + \right. \\ &\left. max\left(\frac{size_{data}}{bandwidth_{slave}}, \frac{p\, size_{data}}{bandwidth_{master}}\right)\right) \end{aligned} \qquad (10)$$

The model can easily be adapted for asymmetric networks as found in home-computers connected to the Internet by asymmetric ADSL or CableTV links.

4.2 Model Validation

Since we have a running implementation of Dyana for SMPs and clusters, we can use application measurements as the most reliable way to validate our analytical model. We compare the execution time measured with the application behavior computed using the analytical model for a few selected scenarios and for different kind of molecular structures.

Figure 2a and Figure 2b compare respectively the execution time versus the estimated time for the Prion protein on a Pentium II cluster (400MHz) with one active processor per node (single processor) and two active processors per node (dual processors). Figure 3a and Figure 3b display the same comparisons with the ER2 protein. The graphs point out a good level of accuracy of the model. Figure 4 compares the execution time versus the estimated time for the Prion on a cluster of 16 Alpha (667Mzh) dual processors.

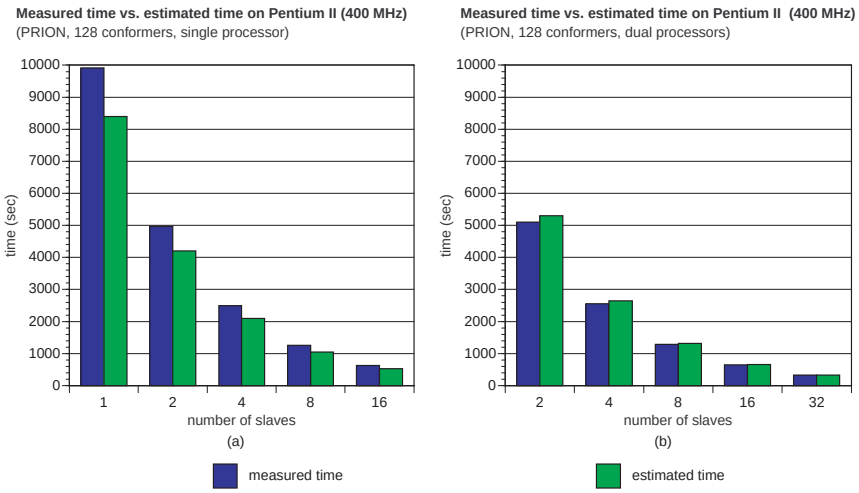

Figure 2. Comparison of the measured execution time to the modeled execution time of Dyana for a Prion protein on a cluster of dual Pentium II (400MHz). A different number of slaves with one active processor per node (a) and with two active processors per node (b) is considered.

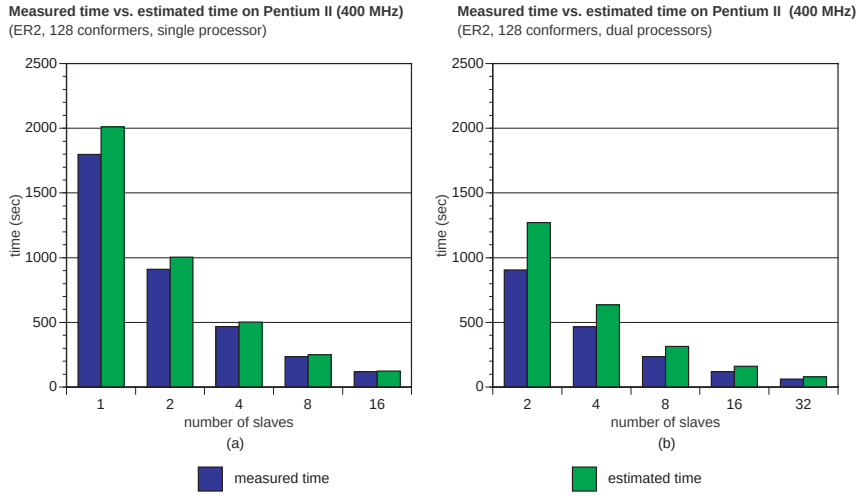

Figure 3. Comparison of the measured execution time to the modeled execution time of Dyana for a ER2 protein on a cluster of dual Pentium II (400MHz). A different number of slaves with one active processor per node (a) and with two active processors per node (b) is considered.

5 Performance Prediction of Dyana on Different Platforms

In the past sections we established and calibrated an analytical performance model for Dyana based on the migration from SMP platforms to cluster of PCs. In this section we will use our model to extrapolate the performance of Dyana to different compute platforms at both ends of the performance spectrum:

- a framework for widely distributed computing (cluster of 400 MHz Pentium Celeron Processors) interconnected by a highly heterogeneous network connecting home users with standard modems to a master with a high bandwidth connection,
- a strongly interconnected high-end cluster whose nodes, all 1 GHz Pentium III processors, are interconnected by high bandwidth networks (1000BaseT).

In Dyana, the master performs only short calculations when it receives a request by a slave. The amount of the data exchanged between master and slave depends on the molecule but in general it is not large. For a typical Prion protein, the communication size, including all the synchronization signals, is less than 20 KBytes.

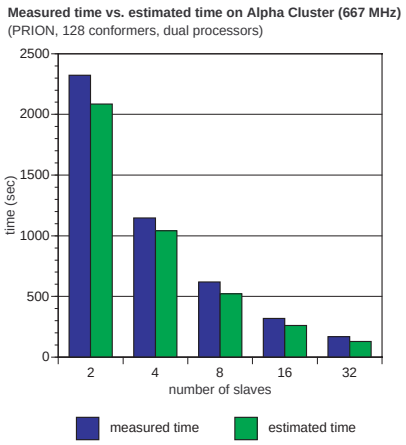

Figure 4. Comparison of the measured execution time to the modeled execution time of Dyana for the Prion protein on a cluster of DEC Alpha (667MHz) for different number of slaves with two active processors per node.

5.1 Performance Prediction of Dyana in Widely Distributed Computing

Although Dyana does not require much computation of the master when it receives a service request, this task may become a bottleneck as soon as the master can not cope with an exceptionally large amount of requests it receives. This is not an unrealistic scenario when thousands of computers on the Internet become involved. With our simple model and the profiling data of the application on clusters, we can try to estimate this cross-over point when a master can no longer serve the amount of requests received without causing a limitation in performance.

On frameworks for widely distributed computing, the master normally resides on a strongly interconnected platform on the back-bone using fully switched 100BaseT or 1000BaseT Ethernet, while the slaves remain connected to this framework by voice modems, ISDN or ADSL connection, at a speed of just 7 KBytes/sec (worst case). With these numbers and our model, we can estimate the communication time:

$$t_{comm} = \left\lceil \frac{n}{p} \right\rceil max\left(\frac{size_{data}}{bandwidth_{slave}}, \frac{p \cdot size_{data}}{bandwidth_{master}}\right) \quad (11)$$

With communication time and duty cycle of the master (master computation time over slave computation time), we can easily derive the maximal number of slaves a single master can handle.

$$p > p_{max} = \frac{bandwidth_{master}}{bandwidth_{slave}} \approx 1500 \quad (12)$$

In the scenario in which the bandwidth of the slaves limits the communication, a request needs approximately 3 seconds to be served:

$$t_{comm\_conf} = \frac{size_{data}}{bandwidth_{slave}} \approx 3\ sec \quad (13)$$

According to our prediction for the case of a slave with a low-cost node (Pentium II 400 MHz), the computation of a conformer required an average 85 seconds. During this time, the master can serve other requests and the duty cycle of Dyana becomes:

$$duty\_cycle = \frac{t_{conf\_siml}}{t_{comm\_conf}} \approx \frac{85}{3} \approx 28 \quad (14)$$

The total number of slaves per master that can be served on this platform is related to the maximum number of slaves which can be served at the same time, p_{max}, and the duty cycle of the application, $duty\_cycle$. In a situation of perfect load balancing, the maximal number of slaves becomes approximately:

$$p_{max} \cdot duty\_cycle = 1500 \cdot 28 \approx 42000 \quad (15)$$

For a good scalability beyond this point, the master must be replicated as well adding a second level of parallelization. Multiple masters can be used to do the communication phases and prevent related bottlenecks on computation involving more computers.

The parameter values observed in Dyana and similar codes indicate that the typical network latency of a few hundreds milliseconds in a grid is far less than the computation time of a task and even an order of magnitude less than the total communication time required for the execution of the task. The computations are therefore communication bandwidth and not latency limited.

Figure 5a shows the total time for the computation of 256 conformers while Figure 5b shows the prediction of the conformer number per hour for the Prion protein on a cluster of Pentium II 400 MHz Celeron. A few hundred processors are still a small number for use in widely distributed computing and for the final paper we would like to gather more numbers and extrapolate our predictions more aggressively to 100'000 nodes.

5.2 Performance Prediction of Dyana on Strongly Interconnected High-End Clusters

For the verification of the model we also consider new platforms with higher performance than the current Beowulf clusters. On a strongly interconnected high-end cluster, the communication rate requirements of Dyana are easily met. Therefore, the communication time can be assumed zero. Figure 6a shows the total time for the computation of 256 conformers for a strongly interconnected high-end cluster whose nodes, Pentium III 1 GHz, are interconnected by high bandwidth networks, while Figure 6b show the prediction of the number of conformers per hour.

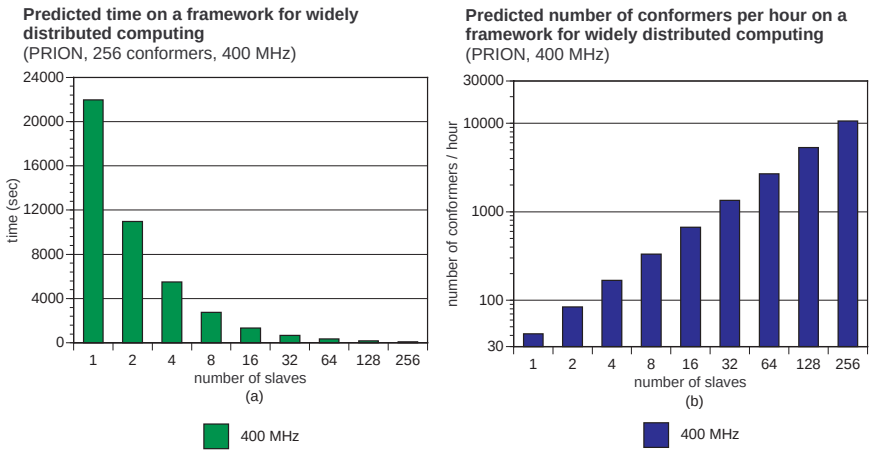

Figure 5. Predicted execution time of Dyana for the computation of 256 conformers (a) and predicted compute rate in conformers per hour (b) of Dyana for a Prion protein on a framework for widely distributed computing. The predictions are computed by our analytical model from basic performance characteristics of the new target platform.

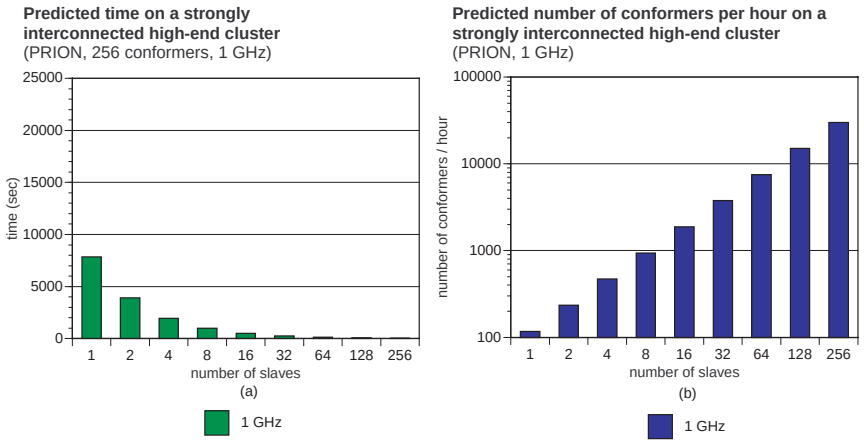

Figure 6. Predicted execution time of Dyana for the computation of 256 conformers (a) and predicted compute rate in conformers per hour (b) of Dyana for a Prion protein on a strongly interconnected high-end cluster. The predictions are computed by our analytical model from basic performance characteristics of the new target platform.

6 Extending our Approach to other Codes and Future Work

An entire class of scientific codes (OPAL, CHARMM, GAMESS, AMBER) can distribute their work-units among different processors in a cluster or on the grid setting. Due to a similar master-slave setting, some computation intensive phases are followed by a communication phase.

In addition to Dyana, we did also look at CHARMM (Chemistry at HARvard Macromolecular Mechanics). The resource usage of CHARMM at the architectural and the operating system level are highly similar to Dyana. Therefore, the modeling of CHARMM can be achieved by the set of formulas given in this paper for modeling the performance of Dyana. The application parameters that determine the clock-per-floating-point-operation (i.e. a specialization of better known CPI) are very similar. So the usage of the floating point units and of the memory system in CHARMM and DYANA are very alike. The total number of floating point operations per work-unit is fairly different, but can easily be determined by benchmarking the sequential version of the code. The amount of communication to start and complete a work-unit can be characterize in the parameters used in our model. Therefore, the same model can be used to characterize the code and predict the performance of CHARMM for grid environments. A precise characterization of CHARMM is given in [15].

Despite the highly positive outlook given by our prediction for Dyana on a grid platform, we recently focused our attention and our engineering capacities to the migration of CHARMM to some well known grid platforms for highly distributed computation [1, 4].

7 Conclusion

A successful migration of the molecular dynamics code Dyana from SMP workstations to a more cost-effective cluster of commodity PCs drastically increases the compute resources available to biologists for this kind of calculations. Because of a clean separation of control and data transfer, the amount of code re-engineering needed for the migration is kept at a minimum. While we readily adapt the control transfer mechanism to the best tasking paradigm for a particular platform, we left all bulk data transfers with file system calls, using the appropriate underlying mechanism of shared memory in SMPs, NFS in clusters or the corresponding toolkit functions in grid computing environments. This programming shortcut is properly justified by the small impact of the data transfers on the overall performance of Dyana.

To further increase the number of computers available to Dyana, we study the viability of a next migration step from clusters to some newly developed frameworks for widely distributed grid computing on the Internet. The effectiveness of such a migration is studied with an analytic performance model of Dyana. The model predicts performance on a broad range of different computing platforms including SMP servers, clusters of PCs and novel infrastructures for widely distributed computing on the grid. The model incorporates application parameters like protein size and machine parameters like CPU clock frequency, memory system type or network speed and the degree of parallelism used. The parameters are properly fitted to different experiments involving 1 to 32 processors as 32bit Intel Pentium IIIs and 64bit Compaq Alphas with clock rates between 400 MHz to 1 GHz.

The use of the model is calibrated and validated during a previous migration from SMPs to cluster of PCs. The model reaches an excellent fit with a high accuracy in its predictions on average and a worst case of just of 19% deviation from experimental data. The per processor performance range considered is 82 MFlop/s to 180 MFlop/s per processor. The aggregate performance in our most parallel system reaches up to 4.9 GFlop/s.

After calibration, we use our performance model to extrapolate the application runs to a widely distributed computing infrastructure incorporating thousands of processors on the Internet and are predicting excellent scalability up to approximately 42000 processors. At this number of slaves the communication to the master becomes the bottleneck. Replication of the master in a multilevel hierarchical setup can resolve this.

The performance model of this study indicates that Dyana can be successfully migrated to frameworks for widely distributed systems on the Internet and is, therefore, well suitable for grid computing. Since Dyana is crucial to the ongoing research of Prion infections like BSE, the code would be an ideal candidate for a “good cause” computing campaign.

References

[1] D. Anderson and et al. United Devices - Building the worlds largest computer, one computer at a time. http://www.ud.com/.

[2] R. Blumofe, C. Joerg, B. Kuszmaul, C. Leisserson, K. Randall, and Y. Zhou. Cilk: An efficient multithreaded runtime system. In *Proc. 5th ACM Symp. on Principles and Practice of Parallel Prog. (PPoPP)*, pages 207–216, Santa Barbara, July 1995. ACM.

[3] S. Brauss, M. Lienhard, J. Nemecek, A. Gunzinger, Näf M., M. Frey, M. Heimlicher, A. Huber, P. Müller, and R. Paul. An Efficient Communication Architecture for Commodity Supercomputers. In *Proceeding of the SC99 Conference*, Portland, Oregon, USA, Nov 13-19 1999.

[4] A. Chien and et. al. Entropia - High Performance Internet Computing. http://www.entropia.com.

[5] P. Güntert. Structure calculation of biological macromolecules from NMR data. *Quart. Rev. Biophys.*, 31:145–237, 1998.

[6] P. Güntert, C. Mumenthaler, and K. Wüthrich. Torsion angle dynamics for NMR structure calculation with the new program Dyana. *J. Mol. Biol.*, 273:283–298, 1997.

[7] A. Jain, N. Vaidehi, and G. Rodriguez. A fast recursive algorithm for molecular dynamics simulation. *J. Comp. Phys.*, 106:258–268, 1993.

[8] R. Jain. *The Art of Computer Systems Performance Analysis*. Wiley Professional Computing, 1996.

[9] Ch. Kurmann and T. Stricker. ECT - Extended Copy Transfer Characterization. http://www.cs.inf.ethz.ch/CoPs/ECT/.

[10] Ch. Kurmann and T. Stricker. Characterizing memory system performance for local and remote accesses in high-end SMPs, low-end SMPs and clusters of SMPs. In *7th Workshop on Scalable Memory Multiprocessors ACM Trans. Computer Systems. held in conjunction with ISCA98*, Barcelona, Spain, June 1998.

[11] G. Roos. Computation of protein structure with dyana on a cluster of pcs. Technical report, ETH Zurich, 2000.

[12] M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra. *MPI - The complete Reference*. MIT Press, 1997.

[13] P. D. Stout. WAX: A Wide Area Computation System. Technical report, School of Computer Science, Carnegie Mellon University, 1994. PhD thesis. Available as Technical Report CMU-CS94-230.

[14] W. T. Sullivan, D. Werthimer, S. Bowyer, J. Cobb, D. Gedye, and Anderson D. A new major SETI project based on Project Serendip data and 100,000 personal computers. In *Proc. of the Fifth Intl. Conf. on Bioastronomy.*, 1997.

[15] M. Taufer, E. Perathoner, A. Cavalli, and T. Caflish, A. Stricker. Performance Characterization of a Molecular Dynamics Code on PC Clusters. Is there any easy parallelism in CHARMM? In *Proc. of IPDPS, IEEE and ACM International Parallel and Distributed Processing Symposium*, Fort Lauterdale, FL, USA, Apr 15-19 2002.