Combining Task- and Data Parallelism to Speed up Protein Folding on a Desktop Grid Platform

Is efficient protein folding possible with CHARMM on the United Devices MetaProcessor?

B. Uk[1], M. Taufer[1], T. Stricker[1]
[1] Department of Computer Science
ETH Zurich
CH-8092 Zurich, Switzerland
buk,taufer,stricker@inf.ethz.ch

G. Settanni[2], A. Cavalli[2], A. Caflisch[2]
[2] Department of Biochemistry
University of Zurich
CH-8057 Zurich, Switzerland
settanni,cavalli,caflisch@bioc.unizh.ch

ABSTRACT

The steady increase of computing power at lower and lower cost enables molecular dynamics simulations to investigate the process of protein folding with an explicit treatment of water molecules. Such simulations are typically done with well known computational chemistry codes like CHARMM.
Desktop grids such as the United Devices MetaProcessor are highly attractive platforms, since scavenging for unused machines on Intra- and Internet delivers compute power that is almost free. However, the predominant programming paradigm for current desktop grids is pure task parallelism and might not fit the needs for protein folding simulations with explicit water molecules. A short overall turn-around time of a simulation remains highly important for research productivity, but the need for an accurate model and long simulation time-scales leads to tasks that are too large for optimal scheduling on a desktop grid.
To address this problem, we introduce a combination of task- and data parallelism as a well suitable computing paradigm for protein folding investigations on grid platforms. As a proof of concept, we design and implement a simple system for protein folding simulations based on the notion of combined task and data parallelism with clustered workers. Clustered workers are machines grouped into small clusters according to network and CPU performance criteria and act as super-nodes within a desktop grid, permitting the utilization of data parallelism in addition to the task parallelism.
We integrate our new paradigm into the existing software environment of the United Devices MetaProcessor. For a test protein, we reach a better quality of the folding calculations than we reached using just task parallelism on distributed systems.

Keywords

Protein folding, computational grid, best-first search, data and task parallelism, CHARMM, United Devices MetaProcessor.

1. INTRODUCTION

Protein folding is a research area in biology that could greatly benefit from the vast amount of almost free computational power provided by widely distributed computation on the Internet. Accurate molecular dynamics (MD) simulations of proteins are time consuming. In particular, the explicit treatment of water and the Particle Mesh Ewald Method (PME) method limit the accessible time scale to less than one microsecond of simulated time, while proteins in real life take microseconds to seconds for the folding process. The computational resources must probably be increased by a million to deal with this kind of application.
Protein folding is essentially a search for the best fitting conformer (molecule state) and there is a lot of parallelism in the application, but the need for an accurate and long running simulation with extended time-scales leads to tasks that are too large for optimal scheduling on a widely parallel computing platform like a desktop grid. Moreover, a short overall turn-around time of a simulation remains highly important for research productivity. To address the need for accuracy and effectiveness in protein folding, we propose to use task- and data parallelism within the CHARMM code in this technical report. With this new paradigm the number of CHARMM applications suitable for running on widely distributed platforms and grids can be increased significantly.
As a proof of concept, we introduce a simple distributed computation model based on *clustered workers* forming small clusters of PCs within the grid environment of our widely distributed platform. The use of data parallelism is delegated to the clustered workers, that are able to act as supercomputer nodes and speed up the turn-around time of a

partial simulation, despite the need for accurate and time consuming methods for computation. Clustered nodes rely on proximity and strong interconnections among their processors. We find such situations everywhere, i.e. in office workgroups sharing a single Ethernet switch, in machine rooms with smaller clusters or in student computing rooms at a university. The suitability and the precise characteristics of CHARMM for task parallelism on computational grids is established in previous work of our group [1, 2] in which we successfully migrated the CHARMM code for simple but fairly coarse grain protein folding simulations (e.g. with implicit solvent and no PME method) to the widely distributed platform of United Devices (UD): the UD MetaProcessor [3]. We also collected experiences with data parallel CHARMM using accurate methods like PME and described an extensive workload characterization in [4].

In this technical report, we also consider the UD MetaProcessor as a good platform for task parallel computation, but extend the previous work by combining the two paradigms of task- and data parallelism in one single system. This requires some small engineering changes to the distributed computing infrastructure which can be accommodated with minor changes to the UD MetaProcessor platform. The discovery of suitable candidates of processing nodes (PCs) for clustered workers in general remains a challenging issue subject to past and future research in the grid community. Focusing on the feasibility of an application in this technical report, we go for the more suitable candidates, taking advantage of a well-mapped and well-known campus computing infrastructure, but in general, many more factors of the infrastructure for such a system must be taken into account - most notably the available network bandwidth and latency between the nodes, but also the heterogeneity caused by different CPU clock rates and the dynamic load behavior due to other processes on the nodes. Most issues related to the discovery of network topologies have been discussed and implemented in ReMoS [5]. An algorithm for dynamically building clusters from network mapping information is given in [6].

To demonstrate the feasibility of large scale and accurate distributed folding simulation in this technical report, we quantify the increase of the quality of the folding simulations due to clustered workers and report our experience with clustered workers implemented on to the commercial United Devices MetaProcessor platform. We can comment on the communication requirements, the handling of failures and the security.

Previous research addresses the issue of protein folding simulations on distributed platforms. CHARMM on the distributed platform Legion has been studied in [7], but their implementation is limited to high-performance, strongly interconnected clusters and a restricted number of tasks, while our work applies more broadly to commodity desktop PCs connected by a wide variety of network technologies. The *Folding@home* project conducted by the Pande group at Stanford University [8, 9] relies on the TINKER molecular dynamics package. This project is based on task parallelism and an exhaustive search using a generate-and-test method on PCs connected via Internet in which systematic generations of possible solutions for the protein folding is done without any support of heuristic functions.

In Section 2, we briefly introduce the protein folding process using the well-known computational code CHARMM. We also identify the algorithmic approach of protein folding as a best-first search algorithm with particular depth-first and breadth-first components. In Section 3, we state the limitations of protein folding on current widely distributed systems and address the solution to these problems by presenting a combination of task- and data parallelism as a very natural computing paradigm for protein folding or similar simulations in computational chemistry. In Section 4, we present our prototype of a system that uses task- and data parallelism on the United Devices MetaProcessor, a platform for widely distributing computing that we regularly use in our desktop grid. In Section 5, we evaluate our system for speed and accuracy and present experimental results combining task- and data parallelism for a specific protein, the GSGS domain. In Section 6, we summarize what we learned from our experiment and conclude.

2. PROTEIN FOLDING

2.1 The Computational Code CHARMM

CHARMM is a code for simulating the structure of biologically relevant macromolecules (proteins, DNA, RNA) [10]. The package uses classical mechanical methods to investigate potential energy surfaces derived from experimental and "ab initio" quantum chemical calculations [11]. We use molecular dynamics (MD) simulations at constant temperature to investigate the protein folding process. In these simulations the Newton equation of motion of the system (protein + thermal bath) is discretize and solved by an integration procedure (Verlet algorithm). The force applied to the atoms in each step is calculated by the negative gradient of the CHARMM potential energy [11].

2.2 Modeling Protein Folding as a Search Algorithm

To simulate protein folding, we start from an unfolded conformation with random torsion angles. A heuristic function guides the search for the folded conformation. This function provides an estimate on how closely a simulated conformation matches to the experimental structure observed by x-ray crystallography or NMR spectroscopy. The function maps each simulated conformation into a measure of desirability called the quality factor. The quality factor used in this technical report is the Root-Mean-Square Deviation (RMSD) of the atomic coordinates (in Angstroms, Å) from the folded structure determined by NMR.

The search for a trajectory from an unfolded to a folded conformation can be seen as a search through a search tree enumerating a large number of different conformations of the molecule under investigation. To find trajectories leading to conformations close to the native (folded) conformation of a protein, a well guided search procedure is used to find a path through the tree, starting from a conformation representing the initial state, then going through the successors so far until either a fixed number of simulation steps is done or a given quality factors is reached. The full search tree with all the possible conformation may become extremely large and might exceed the computational capability of all computers available in a grid. However, like in many brute force searches, a large number of possible paths in the search tree can be pruned and do not need to be generated nor to be explored. In our approach to the folding process, the search

process represents the tree implicitly as a set of production rules (production of the nodes of the tree).
In our search for the optimal match of a conformations, we do a best-first search trough the large tree of simulated molecule conformations. Such a search combines the benefits of the depth-first search (hill climbing) and the breadth-first search (exhaustive enumeration) procedures.

2.2.1 Molecular dynamics simulation as depth-first search

A MD simulation along a certain number of time steps is the basic quantum of our protein folding computation. Such a simulation goes way down into the tree of possible conformations and can be seen as a depth-first search. We depict the trajectory of such a simulation as a single arc in our tree. A simulation is the basic work-unit (minimal scheduling quantum) in our task parallel version of protein folding. A work-unit is a linear chain of simulation time steps without any branches. For cost reasons, we evaluate the quality factor only periodically at certain snapshots along a work-unit simulation, typically around every 100 simulation steps. The result returned by a work-unit is the conformation with the lowest quality factor.

2.2.2 Randomizing conformations as breadth-first search

The protein folding process is explored by processing multiple work-units simultaneously and in parallel. This process leads to breadth-first searches through a tree of possible conformations calculated by the work-units through MD simulation. The branching of the search tree is achieved by the introduction of some disturbance by randomize changes to the best conformation found so far. The random factor is introduced by changing the velocities of the atoms to random values while preserving the temperature. The effectiveness of this kind of algorithms in molecular dynamics simulations has been discussed in [12].
The branching factor is determined by the size of the work-pool, the set of work-units in process and the set of work-units waiting to be assigned to a worker. The size of the work-pool can be either fixed or can change dynamically during the folding process. Fitting the relatively static configuration of our desktop grid and considering the better reproducibility, we fix the size of the work-pool at the beginning of the folding process in this technical report. For larger and more dynamic desktop grid configurations, a dynamic work-pool might be more suitable. New work-units are generated at regular, fixed intervals during the folding process. The new generated units fill the empty spaces in the work-pool and substitute the work-units which have terminated with a result.
Figure 1 shows a partial search tree. Each nodes of the tree corresponds to a conformation and is characterized by a quality factor (red numbers close to the nodes). From each node of the tree, a set of arcs leads to further nodes. An arc stands for an entire MD simulation (search in depth) that starts from the conformation in this node but is based on random velocities of the atoms (label on the arc). The successors of a node are the conformations found in the completed work-units starting from this node and are possible candidate for the generation of new work-units at the next interval. In the picture, we show three update phases for which new work-units are generated and queued in the work-pool. In Figure 1, the starting protein conformation (Mol_1) at the top is a best conformation known at the time. We also assume an empty work-pool in that state and see that for each empty spot a new work-units with a different random seed is generated ($r1$, $r2$, etc). In the next update (level below) only two work-units have terminated their simulation, while two others are either in progress at the workers or waiting to be assigned to a worker. Among the work-units that reached a result, the conformation with best quality factor (Mol_2) is chosen and two new work-units are generated from that conformation using random seeding and they are added to the work-pool (server queue). Note that the work-units always remain in the work-pool queue until finished. During the third update, all the four work-units are finished with their simulation. At this time the best conformation (Mol_3) is used to add four new work-units to the work-pool.
If a phase results in no improvement of the quality factor, the newly calculated conformations are discarded and more randomized work-units are generated from the current best conformation. If this situation occurs repetitively, the protein folding simulation is stuck in a local minimum, and a different strategy is needed to escape this minimum. To deal with local minimum, we do backtrack to earlier conformations that are still maintained in the work-pool. It is extremely unlikely that the protein folding simulation process results in the same conformation over and over again and this virtually excludes the possibility of cycles in the search path.

3. COMBINING TASK- AND DATA PARALLELISM IN PROTEIN FOLDING

3.1 Limitations of Protein Folding on Existing Systems

The steady increase of compute power has enabled the use of more advanced molecular dynamics simulations techniques to investigate the 3-D structure of biological macromolecules [13]. The basis of these calculations relies upon several models of varying complexity and accuracy that range from *ab initio* calculations, based on the approximate numerical solution of the Shrödinger equation for the electrons of the system [14] to highly simplified models where only few degrees of freedom are taken into account [15, 16]. Obviously the computational cost of each model is directly related to its complexity and accuracy. In addition there is a relationship between simplicity and ease of distributed calculation. Highly advanced and complex calculation methods are much harder to carry out in a highly parallel and distributed environment.
The simulated time scales for the biological phenomena in question range from picoseconds (e.g. for an enzymatic reaction) to seconds for the most advanced processes of protein folding. For the study of aggregation and fibril formation, the time scales can extend to multiple days. On a microscopic side, the physical laws governing each phenomenon are only captured in part by the resolution of the available models. Thus, depending on the phenomenon we want to describe, we have two requirements: we need to simulate enough time for the phenomenon to occur and we need a sufficient resolution for the model to be accurate. In short we need as much computing power as we can get for those

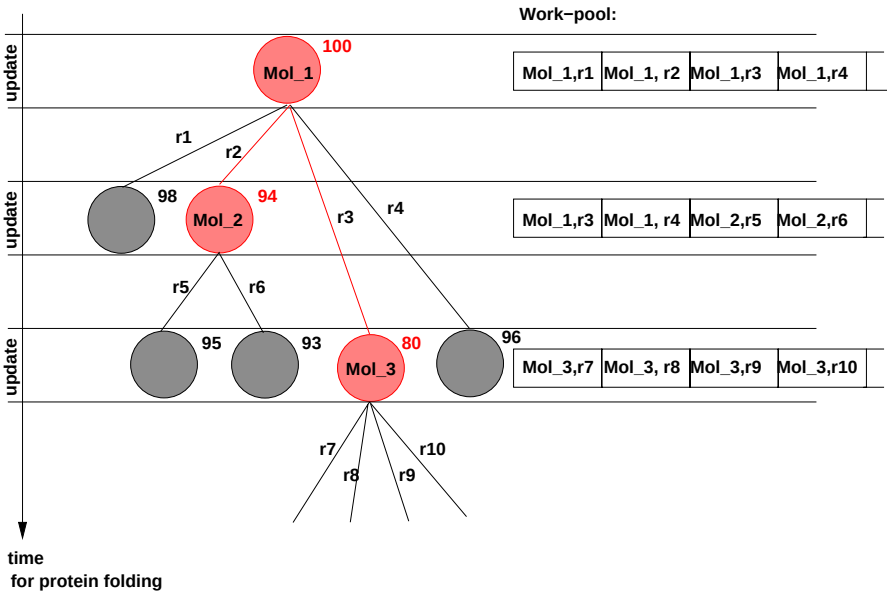

Figure 1: Example of partial search tree for a protein folding simulation. In each update phase (level) some new work-units are generated from the conformation with the best quality factor observed so far and queued in the work-pool.

calculations.

3.1.1 *Demand for large spectrum of folding conformations*

In addition to a good resolution and an appropriate time scale, the simulations require the test of many randomized conformations in the search of an optimal fit to the observed folded structure of the protein. The search is difficult due to the enormous number of different 3-D conformations that a protein or a structured peptide can assume. Although the precise number of states in the search space can not be counted, it becomes clear that protein folding investigation is an extremely resource demanding, time consuming process and therefore the most advanced distributed compute platforms should be considered.

3.1.2 *Benefits of more accurate calculation models*

For the folding of small proteins and structured peptides, which naturally occurs in the μs to ms time scale, several models are being used. We will focus our attention on the empirical models with implicit or explicit treatment of solvent [10, 17]. In the implicit solvent models, all atoms of the solute are taken into account, while the effects of solvent (charge screening, free energy of solvation etc.) are approximated with expressions that depend on the coordinates of the solute atoms [18]. The main advantage of this approach consists of integrating out the degrees of freedom of the solvent, thus dramatically reducing the computational cost of the calculation [19]. On the other hand, for systems where the molecular nature of water plays a relevant role, this approximation does not hold and may lead to inaccurate results.

A step forward in the direction of more accurate (and more complex) descriptions is made by models that explicity represent water molecules. The atoms of the solute are immersed in a box of water molecules with periodic boundary conditions. The water box should be large enough to give negligible interactions between images of the solute in neighboring boxes. This approach leads usually to a 10-fold increase of the number of atoms of the system and consequently to a two order of magnitude increase of the computational cost of the simulation, if no cutoff on the maximal distance between interacting atoms is used (usually two body interactions are the rate limiting step of the calculation). The use of a cutoff allows reducing the computational expenses but introduces some distortions and inaccuracies that may be partly corrected using Particle Mesh Ewald method [20]. Opposite to implicit water simulations, where the small number of atoms poses strong limitations to the parallelization of the calculation (because of the large amount of communication needed between processors), explicit water simulations with the use of cutoffs allow to take advantage of parallelization on parallel systems.

3.1.3 *Significance of short turn-around time*

Biological studies involving protein folding often require a fair amount of manual computational steering by an expert understanding the biological meaning of the simulation results. Therefore simulation environments with short turn-around times are preferred.

The scalability of the MD code with explicit water molecules on parallel machines is fundamental for the reduction of turn-around time of simulations. PME, despite the use of the Fast Fourier Transform, was shown to scale well, at least on some architectures [21] providing some significant reduction of the turn-around time.

3.1.4 *Computational requirements*

A protein folding simulation results in a search through a very large conformation tree in which high parallelism in

breadth is needed in order to limit the total turn-around time of the folding investigation. To cope with this problem, large number of CPUs is required. Accurate models cause long turn-around times for work-units, making the search in depth a high time consuming process.

3.2 Data Parallel CHARMM

The amount of data parallelism in CHARMM and consequently the scalability of the parallel execution of a work-unit was studied in previous work by others [22] and in our group [4, 23]. In our previous study we have proven that on small clusters of PCs, the data parallel CHARMM scales well up to 8-16 nodes using message passing systems like SCore [24] on Gigabit Ethernet (similar results have been measured for Fast Ethernet in [23]). Marginally better scalability is achieved using more cost-demanding technologies like Myrinet (up to 32 nodes).

3.3 Task Parallel CHARMM

In [1, 2], we have investigated the protein folding process based on task parallelism and best-first search for a desktop computational grid, the United Devices MetaProcessor (a detailed description of the platform is reported in Section 4). In that previous study, we took into account folding processes with implicit treatment of water and no PME model so that the turn-around times were short while the results missed some accuracy. The grid platform was able to provide a very large range of investigation using a scheduling approach for the work-units (tasks) based on master-worker setting and *eager scheduling*. The protein folding algorithm used has turned out to be inherently *fault tolerant*: because the work-units generated from a certain conformation are identical except from a random number seed that is used to assign new atom velocities, the results are only marginally affected by the infrequent loss of an online work-unit. We also proved that the calculation of protein folding using the best-first search on the chosen grid platform is robust against heterogeneous environments and against limited communication capabilities.

3.4 Extending the Scope of Calculations

Looking at the demands of protein folding simulations listed above, it appears to us that the combination of task- and data parallelism is a natural paradigm for protein folding and similar calculation in computational chemistry. The high level of task parallelism provided in task parallel CHARMM [1, 2] can be used for large searches in breadth required by the protein folding process. To limit the turn-around time of the work-units characterized by accurate computation methods (i.e. explicit treatment of water, PME method), the data parallelism present in data parallel CHARMM [4, 23] can be used for the speed-up of the search in depth.

4. A SOFTWARE SYSTEM COMBINING TASK- AND DATA PARALLELISM

4.1 The United Devices MetaProcessor Platform

The United Devices MetaProcessor platform (MP platform) [3, 25] provides an environment for running compute-intensive tasks in a highly distributed way over many desktop-class machines in an Intranet (corporate-wide) or in the Internet (worldwide). The MP platform is well established for applications with low communication-to-computation ratio and involving coarse-grain parallelism and with no dependencies between work-units. Figure 2 shows the MetaProcessor platform architecture and components.

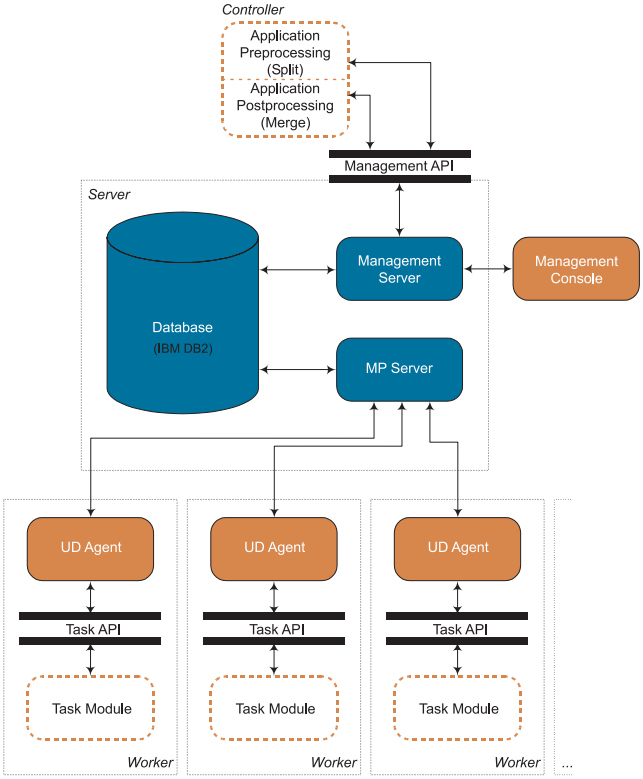

Figure 2: MetaProcessor platform overview.

The MP server is the centerpiece of the software system that links the participating agents to the MP platform. It is responsible for scheduling, as well as for the distribution of task modules, resident data and work-units to the agents, and for receiving results returned by the agents. The management server allows the internal data structures to be accessed via the *Management API* (Application Programming Interface), allowing for instance the submission of work-units and retrieval of results. The database contains all information relevant to the MP platform. The database, MP server and Management server form the server side of the MP platform.
The UD agent is a small program which runs on each participating device (or *worker*). The agent communicates with the MP server to request work-units, resident task- and data modules as needed, executes the task module on the participating device, and returns the task's results to the server. The management console provides a web-based interface to the MP platform, allowing administrative tasks to be performed. *Work-units* are submitted to the server via the Management API by a *controller process* which performs the generation of work-units and the retrieval of results.

4.2 Limits of the UD MetaProcessor

The UD MetaProcessor was originally designed for environ-

ments in which each node operates as an isolated, independent worker. As such, the MP platform works well for situations involving pure task parallelism where, for instance, users let their computers at home participate in a calculation. The MP platform currently does not support any parallel computing environments like MPI or OpenMP which allow data parallel computations. On the other hand, installations of groups of PCs connected by switched 100 MBit/s Ethernet network interconnections are commonplace nowadays and are powerful enough to support data parallel calculations in small clusters. Such clusters could be integrated with the MP platform to perform data parallel computation inside the cluster in addition to the task parallelism of the MP platform. In our prototype, we implement an extension to the MP platform to be able to accommodate such combined parallel models of computation using MPI without any support from the MP platform. In the future, we would like to improve the integration of such an environment within the MP platform to facilitate future applications of combined task- and data parallelism.

4.3 The Architecture of Clustered Workers

A combination of task- and data parallelism is achieved by extending the MP platform by our implementation of *clustered workers* in which a single compute node is built statically from a small PC cluster instead of a single CPU. Alternatively, we could also see a clustered worker as several CPUs that dynamically team up to form a small cluster of PCs within the distributed computing platform.

4.3.1 Static clustering of CPUs into a clustered worker

A first, simple solution extends the UD agent software to be able to control a clustered worker of 2-16 nodes. As the MP platform can handle different agents and task executables for different operating systems, it can easily handle clustered workers with different node architectures: uniprocessor PCs, SMP machines, small clusters of PCs. The agent registers itself with the MP server and identifies itself either as a uniprocessor PC or a clustered workers communicating its node architecture along with corresponding information like the OS and the CPU clock rates.

The architecture of the agent is fixed and cannot change during the lifetime of the agent. The server sends the proper task executable to the workers and does not need significant re-engineering since the necessary functionality is already provided by the present MP server. The main disadvantage of this approach is that we largely rely on static configuration tables and network maps to determine the structure of the platform. However, this solution was fairly easy to implement and delivered the experimental results presented in this technical report for widely distributed protein folding with CHARMM.

4.3.2 Dynamic clustering of CPUs into a clustered worker

A more advanced system allows clustered workers to form dynamically at run-time. Such a software system requires a significant number of extensions to the MP server software and to the agent software. At the time of registration with the server or even later at the time of scheduling a work-unit, the UD agent tries to find other agents in its local network which could be possible candidates to form a clustered worker. The process is achieved easily by means of a simple resource discovery protocol. The server can keep track of this information and can be enabled to find suitable groups of nodes for forming clustered workers whenever a scheduling decision is made. The controller process in the UD system defines the performance requirements (either dynamically or statically) for the specific work-units of an application and communicates these requirements to the server. The server software suggests a clustered worker configuration as a combination of several nodes together. The server also instructs the participating node to establish efficient message passing or shared memory communication among the CPUs to form the clustered worker.

4.3.3 Engineering issues of clustered workers

The communication requirements for task parallelism (i.e. the communication between the server and the workers) and for data parallelism (i.e. the communication within a clustered worker) are quite different. In the first case, a regular Internet-style, best-effort TCP/IP communication is sufficient while in the latter case a low-latency, high-bandwidth communication facility is required to link the CPUs of a clustered worker. Note that the PCs for clustered workers are selected in the system because they have a good network connection available. Further candidates for such communication support are shared memory nodes in the case of a SMP worker, or nodes in PC clusters with high performance interconnects for the MPI message passing library in the case of a clustered worker. To ensure that a clustered workers can function correctly, it must be ensured that the necessary communication libraries are available on each node of the clustered worker and that they do not affect the PCs in a negative way. Resolving the engineering details of such an architecture is quite a challenge since most high performance communication systems require kernel extensions that are hard to distribute in Intranet or in Internet and remain difficult to maintain for a large variety of different PCs.

The system with clustered workers must still be able to handle failures of nodes in a reasonable manner. The current MP platform does not actively react to failures of workers but shifts the fault tolerance problem to the scheduling of the application. In a clustered worker setting, reacting to faults must be proactive since all related nodes would have to react accordingly to the fault of a single CPU, for example by aborting the work-unit after a certain timeout and - in the dynamic allocation case - by re-registering for single-node operations.

Furthermore, incorporating clustered workers to a platform for widely distributed computing must be leveraged against the efforts to provide security and proper encapsulation of the agent of a worker. In the current MP platform, all communication between the server and the workers is encrypted. For performance reasons, encrypting the communication inside a clustered worker is a performance limiting approach and the nodes of a clustered worker must rather trust each other as well as the network they are sharing. While these aspects could easily be ignored in our test-case with static allocation, they remain a challenge for future work on dynamic allocation of clustered workers.

4.3.4 The prototype of clustered workers for the MP platform

To test our concept of clustered workers, we worked around

the limitations of the current MP platform to be able to allow a single UD agent to act as the master node in a single CHARMM calculation task involving a small cluster of 2-8 nodes. To do this successfully without actually making changes to the MP server platform, we required tight control over the client machines involved in the computations. For this first prototype, we manually determined which machines would act as master nodes, and which machines would join them to form clusters. We made use of the fact that task executables are always started in a subdirectory directly beneath the working directory in which the UD agent is started. To be able to start MPI jobs, we designed the task module to invoke a certain shell-script which would then start the actual task as an MPI job on the predefined nodes. This script takes a parameter which specifies the actual task executable to be invoked. First, a suitable CHARMM executable binary was compiled with the MPICH cluster library. To avoid library dependency problems, this binary was statically linked with all required libraries. This executable was then wrapped using the UNIX "shar" tool, and this shar archive was modified to invoke the shell-script mentioned above after extracting the CHARMM binary, passing the CHARMM binary's file name as a parameter. On each master node, the UD agent was installed in an empty directory. In addition, the MPICH tools had to be installed on each machine participating. In the working directory of the UD agent, a short shell-script was installed which invokes "mpirun" with the actual task binary, the correct number of nodes and the machine-file containing a list of the machines belonging to the respective clustered worker.

5. EVALUATION WITH A PROTEIN FOLDING CALCULATION

Our prototype system using task- and data parallelism for highly distributed CHARMM calculation on the UD MetaProcessor is carefully evaluated with test calculations for protein folding simulations of the GSGS as an experimental molecular structure.

5.1 The Peptide used in this Test Case

The GSGS is a 20-residue synthetic peptide with a stable three-stranded antiparallel β-sheet fold. It has been studied experimentally in aqueous solution by nuclear magnetic resonance (NMR) and theoretically with an implicit solvation model [26, 27]. Experimental and theoretical studies indicate that at temperatures around 300 K, the GSGS populates a single structured form (see Figure 3) in equilibrium with a random coil. Due to its small size, the GSGS peptide is an ideal system to test theories and algorithms dealing with protein folding.

5.2 The Experimental Set-up

The initial conformation of the GSGS is random with a quality factor of 7.5 Å (RMSD from folded structure). In the folding investigation, we can either go through a fixed number of steps in a fixed amount of simulation time and look at the end the quality factor obtained or we can run the simulation until an aimed quality factors is reached.
For the analysis of the performance and the quality of results achieved with pure task parallelism and with the combined task- and data parallelism, we fixed the total time of folding simulation to about 60 hours. We compare the quality

Figure 3: The folded structure of the GSGS, a 20-residue synthetic peptide.

factors reached at the end of the 60 hours on a traditional task parallel UD platform (*testbed with task parallelism*) using only isolated workers (each machine is considered as isolated node of the grid) with the quality factors reached on an innovative platform that comprises the same number of machines but with some of them grouped into clustered workers (*testbed with task- and data parallelism*). Table 1 reports the two different testbed configurations.

Testbed with task parallelism	Testbed with task- and data parallelism
6 workers with 1GHz	6 workers with 1GHz
6 workers with 933MHz	6 workers with 933MHz
6 workers with 600MHz	6 workers with 600MHz
6 workers with 500MHz	6 workers with 500MHz
18 workers with 400MHz	6 clustered workers each with 3 x 400MHz

Table 1: Different CPU clock rates and number of workers for the two different testbed configurations considered, "purely task parallel" and "combined task- and data parallelism"

The grouping of the CPUs into clustered workers is based on network speed and CPU performance criteria. With a map of our network at hand, we look at the kind of interconnection between the machines and estimate their turn-around time per work-unit. Figure 4 reports the average turn-around time for work-unit on each kind of a machine. Since the older machines with just a 400 MHz CPU clock rate lead to a larger turn-around time per work-unit, we group three of these older machines into a clustered workers (3 x 400) and achieve a more powerful super node corresponding to an above 1GHz CPU. Figure 4 confirm that such clustered workers (3 x 400) can significantly shorten the turn-around time of the work-units.
Clustering weaker CPUs into clustered workers results in a reduced variability of the average turn-around times for different kind of workers as reported in Figure 4. We still maintain the heterogeneity in the grid by choosing an appropriate

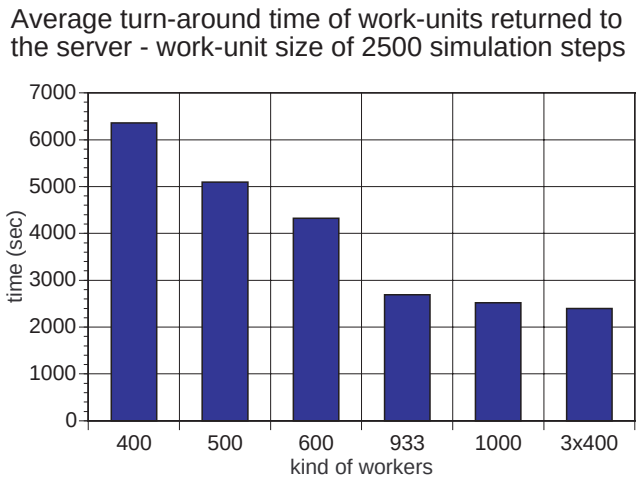

Figure 4: Average turn-around time for work-unit on the several kind of workers taken into account. Besides the isolated workers (1000 MHz, 933 MHz, 600 MHz, 500 MHz and 400 MHz), we look at clustered workers composed by three machines each with 400 MHz CPU clock rate (3 x 400)

scheduling interval length between two renewal processes of the work-pool. We report in [2] that the remaining heterogeneity, contributing some additional randomness, helps to get better results for the folding investigation. For the folding simulation test case, the number of work units and their sizes is roughly the same for both platform candidates. The simulation comprises 2500 steps per work-unit and the work-pool size is 120.

5.3 Differences in Scheduling Characteristics

Figure 5 reports the number of work-units returned (left y axis) to the server by the different kind of workers in the *testbed with pure task parallelism*, in which each node of the grid is an isolated worker, for a simulation of 60 hours. The figure also graphs the number of work-units (accepted work-units) which effectively contribute to an improvement of the quality factor (right y axis). In Figure 6 we report the same information for the *testbed with task- and data parallelism* which groups the slower machines (400 MHz CPU clock rate) into six clustered workers (each one with three nodes of 400 MHz, 3 x 400). The left y axis of Figure 6 displays the number of returned work-units for each class of workers, the right y axis shows the number of accepted work-units, which improve the quality factor. For both Figure 5 and Figure 6, while the left y axis tells us about the overall effort in the search process, the right x axis tells us about the successful work-units that actually help to improve the quality factor in the folding calculation.

For a investigation of the architectural differences in the platform, we use the same amount of machines (42 machines) for the same amount of total simulation time (60 hours). At the end of the experiment with several simulations, both testbed configurations completed a similar amount of work-units returned to the server (compare Figure 5 left y axis and Figure 6 left y axis). However, clustered workers combining task- and data parallelism provides larger number of work-units that act on the quality factor improving it (i.e. the accepted work-units) than workers with just task parallelism. This aspect is clearly visible if we compare the number of work-units accepted for the 18 400 MHz nodes (18 x 400) on the testbed with plain task parallelism (see Figure 5 - right y axis) with the number of work-units accepted for the six clustered workers that are made up with three 400 MHz nodes each (see 6 x (3 x 400) in Figure 6 - right y axis).

5.4 Differences in the Quality Factors

A higher number of conformations accepted leads most likely to a better quality factor in the overall results. For the testbed with clustered workers, we get indeed better quality factor than for the testbed with with only isolated workers - and this despite the same computational resources involved and the same amount of computation done, just due to better scheduling and better guidance of the best-first search. Figure 7 and Figure 8 show respectively the best quality factor reached for several protein folding simulations of the GSGS domain on the testbed with plain task parallelism (with only isolated workers) and on the testbed with task- and data parallelism (with clustered workers). As reported in Figure 7, in a fixed amount of time of 60 hours, we get a quality factor of 3.4 Å using a testbed with plain task parallelism. On the other hand, in Figure 8 for a testbed with task- and data parallelism, we get a quality factor of 2.5 Å which is an exceptionally good result that has never been reached before for this test case with the folding method used in this study, not even with a significant amount of additional computational power in a homogeneous cluster of PCs.

In summary, with the same amount of simulation time, the platform with a carefully choice of clustered workers is able to reach better quality factors for a protein folding simulation with explicit treatment of water (accurate computation) by shortening the turn-around time of the slower work-units significantly. The simulations were repeated several times and the scheduling data as well as the quality factors were found within small variations of the data reported in this chapter. The observations also appear to be robust, since this tendency of the quality factors is clearly visible on similar setups other that the testbed configurations described here.

6. CONCLUSION

Protein folding calculations with accurate molecular dynamics methods and explicit treatment of water require a large amount of compute cycles. Accurate simulations with an acceptably short turn-around time can not be carried out in widely distributed systems using task parallelism alone. To speed up the calculations and to incorporate a large number of machines in a desktop grid, we must also consider data parallelism in addition to task parallelism.

We conjecture that combining data and task parallelism is a natural paradigm for protein folding or similar calculations in computational chemistry. The folding process on a distributed system can be viewed as a combination of breadth-first and depth-first searches through a tree of protein conformations. The need to simulate enough time-steps for the relevant phenomenon to occur in protein folding requires a linear search in depth. The randomized consideration of dif-

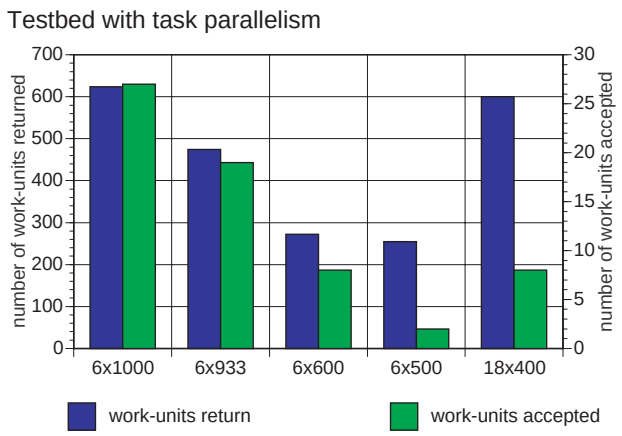

Figure 5: Number of work-units returned (left y axis) to the server by the different kind of workers and number of work-units accepted for the improvement of the quality factor (right y axis) on the testbed with plain task parallelism during a folding simulation of 60 hours.

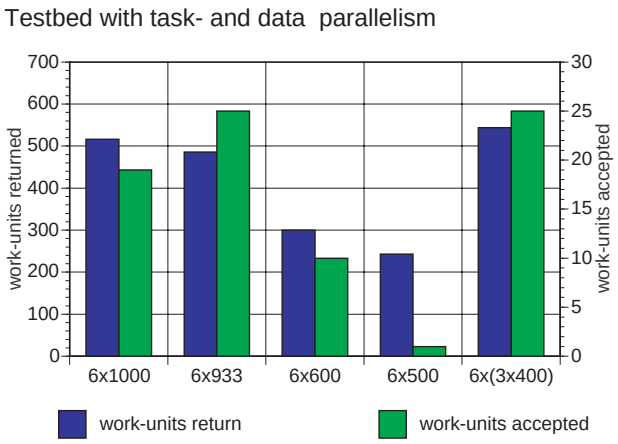

Figure 6: Number of work-units returned (left y axis) to the server by the different kind of workers and number of work-units accepted for the improvement of the quality factor (right y axis) on the testbed with task- and data parallelism during a folding simulation of 60 hours.

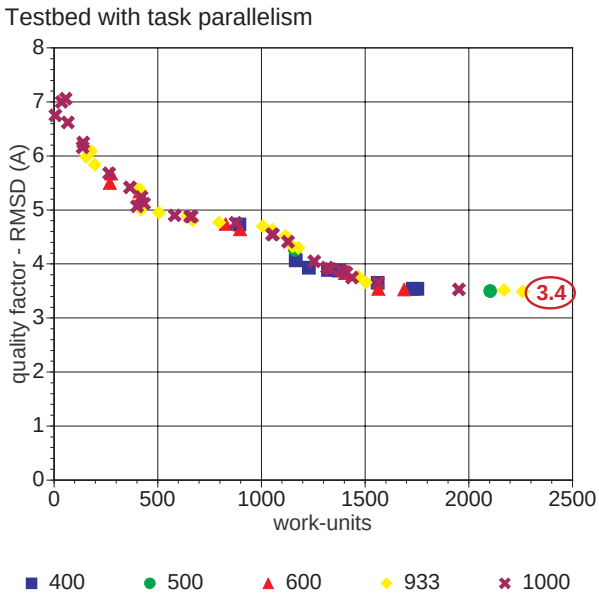

Figure 7: Plot of the quality factor along the most successful protein folding simulation of the GSGS domain on the testbed with plain task parallelism with isolated workers for a 60 hours simulation.

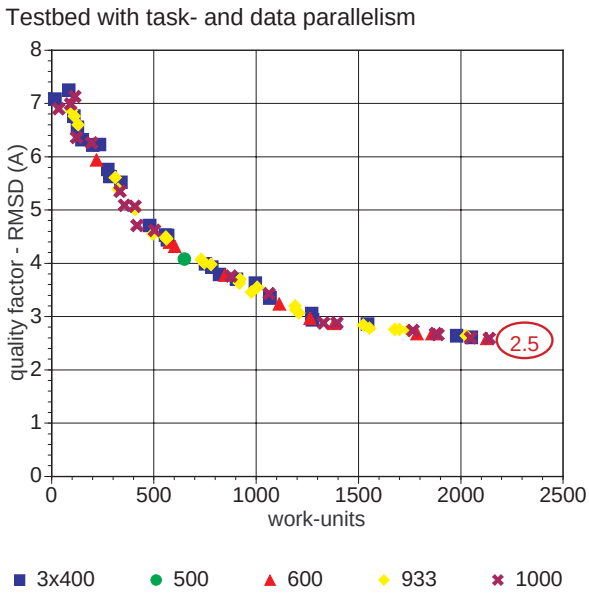

Figure 8: Plot of the quality factor along the most successful protein folding simulation of the GSGS domain on the testbed with task- and data parallelism with clustered workers for a 60 hours simulation.

ferent paths corresponds to the breadth of the search, which increases the demand for computational power beyond the linear simulation.

To cope with the demand for more power and more parallelism, we develop new strategies for generating and distributing computations on widely distributed systems. Our new paradigm of computational systems with clustered workers combines task- and data parallelism. With both forms of parallelism involved a higher number of processors can be effectively used, and this makes protein folding simulations with CHARMM much more suitable for desktop grid platforms.

We design and implement a simple simulation system based on our notion of clustered workers. A clustered worker

groups multiple isolated processors of a desktop grid into a small PC cluster. The candidates for clustering are selected based on performance criteria, like clock rates and interconnect bandwidth. We demonstrate the feasibility and benefit of the paradigm using the commercial software environment of the United Devices MetaProcessor on a desktop grid provided by the student computer rooms of our technical university.

We run a small test protein on two desktop grids of 42 processors each but with a different configuration and look at the computational work done and the quality factor reached after roughly 60 hours of protein folding simulation. Comparing a configuration with isolated workers to a configuration with clustered workers, we observe that the clustered workers have a much higher fraction of work-units that contribute to the improvement of the quality factor, with approximately the same number of work-units completed. As a result of this improvement to the search, we reach a better quality factor for a simulation with clustered workers than with isolated workers. For our test protein and 60 hours with 42 machines, the quality factor jumped from 3.4 to 2.5 which is a significant improvement.

Using the commercial software environment of the UD MetaProcessor and the widespread molecular dynamics code of CHARMM, we show that desktop grids can be a highly efficient computational platform for protein folding. The experimental evaluation of our system shows further that combining data and task parallelism leads to better results than task parallelism alone and is therefore a highly promising approach.

7. REFERENCES

[1] B. Uk. Migration of the Molecular Dynamics Application CHARMM to the Widely Distributed Computing Platform of United Devices. Diploma Thesis, ETH Zurich, Switzerland, 2002.

[2] B. Uk, M. Taufer, T. Stricker, G. Settanni, and A. Cavalli. Implementation and characterization of protein folding on a desktop computational grid - is charmm a suitable candidate for the united devices metaprocessor? Technical Report 385, ETH Zurich, Institute for Comutersystems, October 2002.

[3] United Devices, Inc. Edge Distributed Computing with the MetaProcessor Platform, 2001. http://www.ud.com/products/documentation/.

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

[5] P. Dinda, T. Gross, R. Karrer, B. Lowekamp, N. Miller, P. Steenkiste, and D. Sutherland. The Architecture of the Remos System. In *Proc. of the 10th IEEE Symposium on High-Performance Distributed Computing (HPDC-10)*, San Francisco, California, Aug 2001.

[6] T. DeWitt, T. Gross, B. Lowekamp, N. Miller, P. Steenkiste, J. Subhlok, and D. Sutherland. ReMoS: A Resource Monitoring System for Network-Aware Applications. Technical Report CMU-CS-97-194, Carnegie Mellon School of Computer Science, 1997.

[7] A. Natrajan, M. Crowley, N. Wilkins-Diehr, M. Humphrey, A. Fox, A. Grimshaw, and C. Brooks. Studying Protein Folding on the Grid: Experiences using CHARMM on NPACI Resources under Legion. In *Proc. of the 10th IEEE Symposium on High-Performance Distributed Computing (HPDC-10)*, San Francisco, California, Aug 2001.

[8] Folding@home Project Page. http://folding.stanford.edu.

[9] B. Zagrovic, E. Sorin, and V. Pande. Beta Hairpin Folding Simulations in Atomistic Detail Using an Implicit Solvent Model. *Journal of Molecular Biology*, 317(4), 2002.

[10] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus. CHARMM: A program for macromolecular energy, minimization, and dynamics calculations. *J. Comput. Chem.*, 4:187–217, 1983.

[11] A.D. MacKerell Jr. and et al. All-atom empirical potential for molecular modeling and dynamics studies of proteins. *J. Phys. Chem. B*, 102:3586–3616, 1998.

[12] M.R. Shirts and V.S. Pande. Mathematical Analysis of Coupled Parallel Simulations. *Phys Rev Lett*, 86(22):4983–7, May 2001.

[13] M Karplus and J.A. McCammon. Molecular Dynamics Simulations of Biomolecules. *Nature Structural Biology*, 9(9):646–652, September 2002.

[14] R. Car and M. Parrinello. Unified Approach for Molecular-dynamics and Density-functional Theory. *Physical Review Letters*, 55(22):2471–2474, 1985.

[15] C. Clementi, P.A. Jennings, and J.N. Onuchic. How native-state topology affects the folding of dihydrofolate reductase and interleukin-1 beta. *Natl Acad Sci USA*, 97(11):5871–5876, May 2000.

[16] G. Settanni, A. Cattaneo, and A. Maritan. Role of Native-state Topology in the Stabilization of Intracellular Antibodies. *Biophys J.*, 81(5):2935–2945, November 2001.

[17] J. Gsponer and A. Caflisch. Molecular Dynamics Simulations of Protein Folding from the Transition State. *Proc Natl Acad Sci U S A.*, 99(10):6719–24, May 2002.

[18] B. Roux and T. Simonson. *Biophysical Chemistry*, 78(1-2):1–20, April 1999.

[19] P. Ferrara, J. Apostolakis, and Caflisch A. Evaluation of a fast implicit solvent model for Molecular Dynamics Simulations. *Proteins*, 46(1):24–33, 2002.

[20] U. Essmann, L. Perera, M.L. Berkowitz, T. Darden, H. Lee, and L.G. Pedersen. A smooth Particle Mesh Ewald Method. *J. chem. phys.*, 103(19):8577–8593, November 1995.

[21] M.F. Crowley, T.A. Darden, T.E. Cheatham, and D.W. Deerfield. Adventures in Improving the Scaling and Accuracy of a Parallel Molecular Dynamics Program. *J. Supercomputing*, 11(3):255–278, 1997.

[22] Y. S. Hwang, R. Das, J. H. Saltz, M. Hodoscek, and B. R. Brooks. Parallelizing Molecular Dynamics Programs for Distributed Memory Machines. *IEEE: Computational Science and Engineering*, 2:18–29, 1995.

[23] E. Perathoner. Performance Driven Migration an Optimization of a Common Molecular Dynamics Code (CHARMM) on Different Cluster Platforms. Diploma Thesis, ETH Zurich, Switzerland, 2001.

[24] Y. Ishikawa, H. Tezuka, A. Hori, S. Sumimoto, T. Takahashi, F. O'Carroll, and Harada H. RWC PC Cluster II and SCore Cluster System Software – High Performance Linux Cluster. In *Proc. of the 5th Annual Linux Expo*, pages 55–62, 1999.

[25] United Devices, Inc. MetaProcessor Platform, Version 2.1 Application Developer's Guide, 2001.

[26] P. Ferrara and Caflisch A. Folding Simulations of a three-stranded antiparallel β-sheet Peptide. *Proc. NAtl. Acad. Sci.*, 97(20):10780–10785, September 2000.

[27] P. Ferrara and Caflisch A. Native Topology or Specific Interactions: What is More Important for Protein Folding? *J. Mol. Biol.*, 306:837–850, 2001.