Copernicus: A new paradigm for parallel adaptive molecular dynamics

Sander Pronk
Dept. of Theoretical Physics
Royal Institute of Technology
Stockholm, Sweden

Per Larsson
Dept. of Molecular Physiology and Biological Physics
University of Virginia
Charlottesville, VA, USA

Iman Pouya
Dept. of Theoretical Physics
Royal Institute of Technology
Stockholm, Sweden

Gregory R. Bowman
Dept. of Chemistry
Stanford University
Stanford, CA, USA

Imran S. Haque
Dept. of Computer Science
Stanford University
Stanford, CA, USA

Kyle Beauchamp
Biophysics Program
Stanford University
Stanford, CA, USA

Berk Hess
Dept. of Theoretical Physics
Royal Institute of Technology
Stockholm, Sweden

Vijay S. Pande*
Dept. of Chemistry
Stanford University
Stanford, CA, USA

Peter M. Kasson*
Dept. of Molecular Physiology and Biological Physics
University of Virginia
Charlottesville, VA, USA

Erik Lindahl*
Dept. of Theoretical Physics &
Swedish e-Science Research Center
Royal Institute of Technology
Stockholm, Sweden

ABSTRACT

Biomolecular simulation is a core application on supercomputers, but it is exceptionally difficult to achieve the strong scaling necessary to reach biologically relevant timescales. Here, we present a new paradigm for parallel adaptive molecular dynamics and a publicly available implementation: Copernicus. This framework combines performance-leading molecular dynamics parallelized on three levels (SIMD, threads, and message-passing) with kinetic clustering, statistical model building and real-time result monitoring. Copernicus enables execution as single parallel jobs with automatic resource allocation. Even for a small protein such as villin (9,864 atoms), Copernicus exhibits near-linear strong scaling from 1 to 5,376 AMD cores. Starting from extended chains we observe structures 0.6 Å from the native state within 30h, and achieve sufficient sampling to predict the native state without *a priori* knowledge after 80–90h. To match Copernicus' efficiency, a classical simulation would have to exceed 50 microseconds per day, currently infeasible even with custom hardware designed for simulations.

*pande@stanford.edu, kasson@virginia.edu, erik@kth.se

SC11 November 12-18, 2011, Seattle, Washington, USA

1. INTRODUCTION

Modern high-performance computing spans a wide spectrum of interconnect bandwidths, from remote shared memory links that rival the main memory bus in bandwidth and latency to a complete absence of network interconnect in most distributed computing applications. Large resources with high-bandwidth interconnects tend to be extremely expensive and highly allocated. Distributed computing projects such as Folding@Home can include half a million cores at any time, while 10,000 cores for continuous use would be an extremely large allocation on today's high-end supercomputers. Many interesting real-world applications (all that are not embarrassingly parallel) require some interprocess communication for scaling and are therefore limited both by the availability of this bandwidth as well as the total amount of resources for high absolute performance.

We address the limits of this scaling by using domain-specific knowledge to achieve more efficient parallelization. Biomolecular experiments typically measure averages over billions of molecules undergoing transitions in parallel, while computer simulations have classically tried to reproduce the results by studying sequential transitions for a single molecule. By recognizing that the target problem is fundamentally an ensemble one and incorporating the statistical mechanical sampling into the simulation itself, we can efficiently parallelize using a hierarchical scheme. Top-level communication between members of the ensemble only requires low bandwidth, while communication within an ensemble can utilize high interconnect bandwidth. Having multiple levels of parallelization with different interconnect dependencies al-

lows us to tune the application to the hardware at hand, it achieves state-of-the-art scaling efficiency of simulations at the lowest level, and provides a path for any sampling-based particle simulation to scale to hundreds of thousands of cores (and beyond) on next-generation supercomputers.

In this paper, we present Copernicus[1], a new software platform designed to accomplish this, and its application to molecular dynamics simulations of biomolecules.

Computer simulations of biomolecules, in particular proteins, have yielded fundamental insights into how these building blocks of life work [1], fold into functional form [15] and interact [14]. A distinguishing feature of these systems is that phenomena on the level of atomic detail are important, making individual atoms the smallest units of these simulations. This means that simulations of individual biomolecules usually have 10,000 to 500,000 particles, and it is not possible to improve scaling simply by increasing resolution. The high computational requirements come from the time scales involved in molecular rearrangements, which requires from $5 \cdot 10^5$ (corresponding to 1 ns) to $5 \cdot 10^8$ steps (1 μs) or more.

Molecular dynamics simulations pose significant computational challenges. The systems are big enough to be parallelized, with 100-500 particles assigned to each core in high-performance molecular dynamics (MD) packages such as Gromacs [10, 17] when run on a system with sufficiently low interconnect latency. However, at this point the time required for a step is approaching 100μs of wallclock time, and there simply aren't enough floating-point operations in a single simulation to achieve order-of-magnitude improvements in scalability compared to current state-of-the-art implementations. While molecular dynamics can achieve close to perfect weak scaling, this will put hard bounds in strong scaling and limit the accessible run lengths for biomolecular simulation.

One recent alternative is to use custom hardware with application-specific integrated circuits designed for molecular simulation [18]. This approach is extremely powerful (although expensive) and can extend the accessible timescales by roughly two orders of magnitude, but even this falls far short of most biologically interesting timescales.

Over the last few decades hardware has improved with constantly higher-frequency chips, but the current trend of multi-core chips with *lower* frequencies presents a formidable challenge. This development is likely to continue both in the form of many more traditional cores and the emergence of streaming architectures with simpler processing elements (e.g. GPUs), with trends pointing to an increase in available core counts by several orders of magnitude. This pattern is likely to be similar for custom hardware, which will get more transistors, but not faster ones.

A hypothetical future machine could conceivably simulate a whole cell (weak scaling), but since it would still be subject to the same limits for simulation timescales it would not teach us much about biology. There are several alternatives that try to circumvent the requirement for long simulations. Most rely on the fact the phenomena under study are described by statistical mechanics, and work by splitting up transitions between states into smaller units that can be sampled using shorter individual runs [9]. Examples of this are free energy perturbation-type calculations [4], umbrella sampling [13], or other types of importance sampling methods. Among the most comprehensive of these methods is Markov state modeling (MSM) [16], where the entire free energy landscape of a system under study is split up into individual, short transitions from clusters of similar conformations, where the end result is a collection of clusters and their transition rates. Through adaptive sampling [2] (where new simulations are started from conformations that haven't yet been sampled effectively) it is possible to visit states at a rate many orders of magnitude higher than the physical rate of transition that would otherwise determine the minimum time of a single run [2].

Markov state modeling with adaptive sampling not only explores the landscape of possible states more efficiently, but offers a picture of the *dynamics* of the system being studied while it explores the possible states. An individual molecular dynamics run may or may not see a transition between states, but the ensemble probed by MSM gives the likelihood of transitions and thus the rates at which these transitions happen.

These sampling algorithms give rise to a new level of parallelism, that of the ensemble of individual runs. In effect, the coordination of simulations, an adaptive step, and running new simulations becomes an algorithm of its own. Yet, in practice, this is almost invariably still done by hand, with the user serving as processor for these algorithms. It has also led to a situation where there are essentially two types of simulations: Supercomputers are used for projects with a handful of long trajectories, while throughput-focused resources have been more useful for sampling, often run as a collection of independent jobs.

With Copernicus, we attempt to exploit the inherent parallelism of ensemble simulation and address the difficulties associated with using advanced sampling algorithms, while keeping the performance advantages of massively parallel simulations. A project is executed as a single job, but breaks it up into coupled individual parallel simulations over all available computational resources, with the single simulation as the individual work unit. While the software has been optimized for using multiple high-performance compute clusters, it works equally well with cloud computing instances or even individual workstations.

Copernicus integrates elements of distributed computing, and applies them to more traditional high-performance computing environments by taking advantage of the fast interconnects that may be available on each machine to parallelize individual simulations as far as possible. This makes it possible to use orders-of-magnitude more cores than a single simulation on a supercomputer, enables larger-scale simulations than would be possible with purely distributed computing, and achieves order-of-magnitudes reductions in time-to-solution. In its networked architecture, Copernicus takes a decentralized approach akin to that found in peer-to-peer networks, while at the same time relying on the availability of network resources to do real-time processing of results.

By taking the large-scale sampling jobs as individual units of computation, Copernicus takes a more user-friendly approach to these advanced algorithms, allowing them to be used by non-experts: much in the same way as the emergence of molecular dynamics packages allowed non-experts to start doing high-performance parallel MD simulations.

[1]A first version will be released under an open source license at `http://copernicus-computing.org/` in conjunction with SC11.

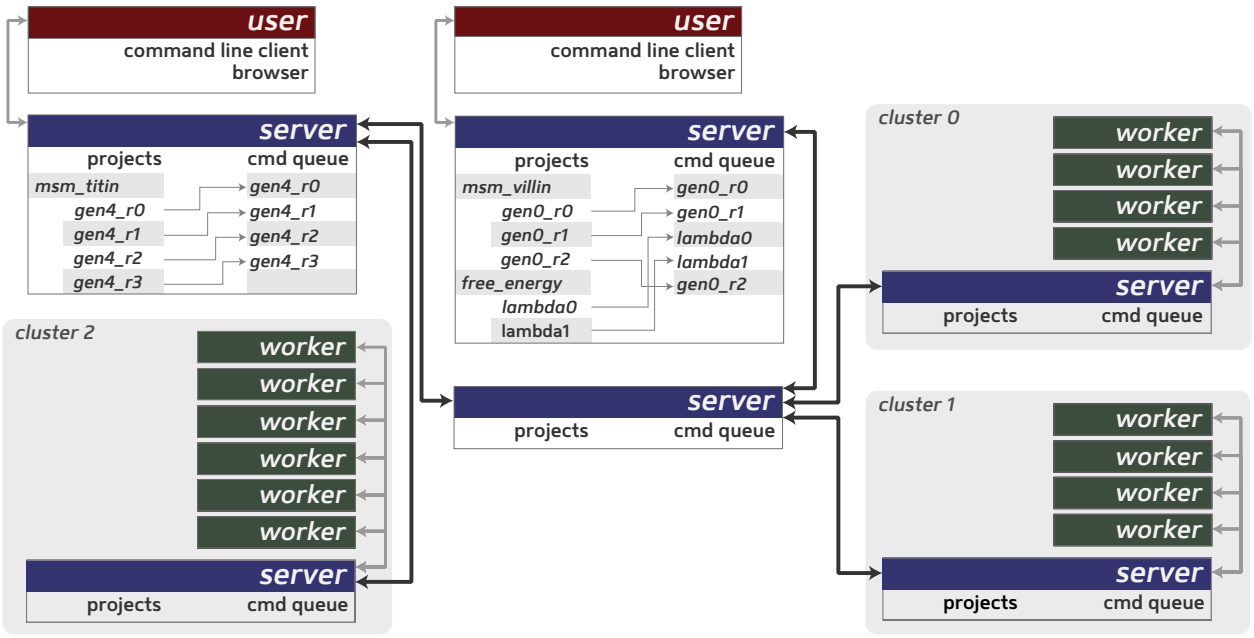

Figure 1: Example Copernicus network architecture with two servers acting as project servers and another four as network servers. Note the simultaneous use of three supercomputers. The overlay network topology is shown in the center, with dark solid lines. In this case, clusters 0 and 1 might be located in Stockholm with a common gateway server, while cluster 2 is in Palo Alto.

The networked approach to running these simulations allows the user to monitor the runs remotely and see results in real time as they become available.

Below, we first describe the architecture of the general Copernicus framework, and then show the benefits of the approach through a specific application in the form of Markov State Modeling for the transition of a small protein (the villin headpiece) from an unfolded to a folded state. Even for this extreme case – the system only contains 9,864 atoms – Copernicus achieves close-to-linear strong scaling up to 5376 cores. We believe this framework combines the strongest aspects of massively parallel high-performance molecular dynamics with the Markov State Model approach to sampling, while making it possible for anybody to deploy on a cluster.

2. COPERNICUS: ARCHITECTURE

The main aim of the Copernicus framework is to efficiently handle large jobs, termed 'projects', consisting of many individual but coupled simulations, and treat those as a single entity. To do this, the system must be able to:

- Match and distribute the individual simulations to the available computational resources.
- Run simulations on a variety of remote platforms simultaneously: HPC clusters, workstations, cloud computing instances, et cetera.
- Parallelize tasks to the maximum extent possible on each resource, and use adaptive coupling beyond this.
- Allow flexibility in the types of projects that can be run.
- Perform real-time analysis of the running project.
- Enable monitoring of running projects.

Because of the networked nature of running remote simulations and the long run time of most projects, Copernicus is set up as a collection of servers connected through an overlay network, with web or command-line clients to control and monitor the projects, and a special class of clients: the 'workers', that run the individual parallel simulations, termed 'commands'. A worker could comprise anything from a single node to hundreds or thousands of nodes running a single simulation with message-passing in a high-end machine.

A typical Copernicus setup is shown in Fig. 1. This illustrates two servers controlling projects and four additional servers acting as communication hosts for the workers. All servers run identical code, and their role is determined solely by their connectivity and whether or not they have running projects. The architecture is fully symmetric in the sense that it can include an unlimited number of sites that both contribute computing resources and initiate projects, subject to authorization. Copernicus makes it possible to combine disparate supercomputers linked by relatively high-latency networks: For the case illustrated in Fig. 1 we can imagine the first two clusters to be located in the same data center with a high bandwidth and low latency connection (behind a firewall with a common gateway server), while the third machine might be on a different continent.

Computational resources are made available to the system by starting workers, e.g. by submitting them to the batch queue of a cluster. Workers present themselves to the system by running a small bootstrap binary to convey the available resources – their architecture, number of cores, and interconnect type – to the closest server. The server then assigns a matching workload of one or more commands to the worker,

including a specification of the program to execute.

The worker completes the commands and returns their output to the closest server, which in turn propagates the results to the server from which the project originated. Projects are maintained by 'controller' servers that maintain job queues and coordinate communication among individual workers and commands. The controller then responds to these finished commands, and new projects or commands can be issued until the project finishes – for example when the standard error estimate of the output result has reached a user-specified minimum value. Progress and results can be monitored in real time through a web interface.

Copernicus can detect and take advantage of shared file systems to reduce communication; future versions will be able to use this to distribute data processing by the controller over the available resources too.

2.1 Plugin-based project control

While much of our work has been focused on biomolecular simulation and tightly integrated with the Gromacs simulation package, a core idea behind Copernicus is to make the framework itself agnostic of the underlying simulation engine and indeed usable for any problem that can be formulated in terms of statistical sampling. In order to be as flexible as possible, even the controllers doing the analysis and deciding what to run are 'plugins' that can be installed (or programmed) by the user. These application-specific controllers break down projects into sub-projects or commands. All knowledge about how to execute a project and how to interpret the resulting command output is contained in these user-installable modules.

Controllers are in essence event handlers that react to a set of conditions: they are called when a project starts, a subproject finishes, a command finishes, etc. In response to these events, controllers can initiate and perform post-processing of data generated by commands. Because the controller can request a specific set of data for each event, controller processing could happen in a decentralized way with project data scattered across several clusters that do not share any storage.

2.2 Overlay network

The design of the Copernicus network was guided by the following considerations:

- Servers and all communications should be secure. Only authorized clients and servers should be able to connect.
- Typically, a compute cluster's individual nodes are on an internal network that is not routed externally, necessitating the presence of an interface server on a cluster's head node.
- The overlay network between servers is small, typically no more than a handful of servers, and is relatively static.
- The network must support routing of requests both to specific servers, and to the first server with available commands.
- User interaction should occur through the network.

To support peer-to-peer usage and almost arbitrary topologies, any Copernicus server can both send and receive commands, either from user clients, worker clients, or other servers. There is no top-level server in the architecture.

The user is free to set up any number of servers to enable communication between a server holding a project and workers; usually a user will set up one server per cluster head node, since compute nodes are typically on a network only accessible from that node. These servers relay workload requests, workloads and simulation data between workers and project-holding servers.

Because most Copernicus network traffic is of a request-response nature, we chose SSL as the basic communication protocol. The encoded routing priority effectively determines the run priority in case of multiple servers with available commands.

In order to set up a new link in the overlay network, the user must initiate the exchange of public keys. Each server has a set of trusted keys which together make up the overlay network and provide authentication between different networks or users. The exchange of keys is, in this case, not an onerous requirement because the number of servers is typically very low, with every server being set up for a specific reason: to act as project server, or to act as network intermediate on a cluster head node. The decentralized nature of this overlay network allows the setup of links to remote clusters on an as-needed basis even while a project is running.

2.3 Resource matching

A workload is matched and presented by a server to a worker using several considerations. Upon startup, a worker gets its platform from the user. This is usually a software platform such as OpenMPI, but could be SMP for a high end shared memory machine. The worker then calls an associated platform plugin. That plugin determines the available resources, such as number of processing cores and amount of RAM, either automatically or through user-defined options. After that, the worker searches for all installed 'executables': descriptions of how to execute specific command types for a specific platform, along with optional binaries to execute. These 'executables' contain the simulation-specific binaries, such as the Gromacs binaries in the example of section 3.

Information about available resources and executables is sent to the server the worker is set up to connect to, where it is relayed to the first available server with commands. This server matches the available executables to commands in its queue, and constructs a workload that maximally utilizes the available resources given the preferred resource requirements of the commands.

When running, workers send 'heartbeats' to the server at specified intervals (120 seconds by default, with a message size typically less than 200 bytes), to report the status of each subpart of the simulation. Heartbeat signals do not get forwarded to other servers; when the worker's server fails to receive a heartbeat signal within twice the heartbeat interval, it will assume that the worker has failed and signal this to the project's server. If the server the worker connects to has access to the same file system as the worker, it will try to return the worker's output data to the project's server, even if the client itself is no longer online. This also allows commands that do checkpointing, such as the Gromacs engine we use in Copernicus, to have another client transparently

continue from the last checkpoint. This makes it possible to schedule Copernicus runs even for very short periods of time on unreliable systems, e.g. during cluster burn-in, and still do useful work.

As described above, each worker client is typically a massively parallel simulation, and the degree of coupling between different worker clients is specified by the controller in terms of the command length. On a high-bandwidth/low-latency machine this could in theory be in the seconds range, provided the controller has time to analyze all the results. In contrast, the interval might be set to several days for high-latency networks used in distributed computing. The decentralized architecture also means we avoid propagating unnecessary data (such as heartbeats) beyond the closest server, which is highly useful even for supercomputers e.g. when the project server and compute resources are located on different continents.

3. FOLDING OF A PROTEIN IN 30 HOURS

The combination of massively parallel and ensemble simulation techniques makes it possible for Copernicus to scale to hundreds of thousands of cores, but its main strength is that parallel adaptive molecular dynamics on supercomputers can achieve extremely high absolute performance, as described in the performance benchmark section. To illustrate the scaling, speedup, and drastically shorter time-to-solution for a real problem that is achieved with Copernicus, we have simulated protein folding of the villin headpiece mutant 35-NleNle. This is a small single-domain protein that has been used as a benchmark for recent high-performance simulation platforms [7, 8, 19]. The MSM plugin in Copernicus enables automated parallel adaptive simulation given a user-specified number of starting structures. To optimize estimation of folding rates and mechanism, data from running simulations are periodically clustered, simulations in well-explored regions of phase space terminated, and new simulations spawned from under-explored regions. The automation and parallelization of this process on HPC resources enables us to reduce the time-to-solution to hours or days rather than weeks or months. In addition, it enables anybody with access to moderate amounts of supercomputing resources – the present example used merely 500,000 core-hours – to rapidly perform protein folding simulations, which is a substantial advance.

3.1 Simulation setup

The run was started by specifying nine unfolded conformations of villin[2]. Simulations were evaluated against the native-state structure with PDB accession number 2F4K [12], but no information from this structure was used in Copernicus. Each unfolded conformation was solvated in a cubic box containing 3036 TIP3P water molecules, and tasks were run for 50 ns between clustering steps. Simulation parameters were used as previously published [7]. Initial velocities were drawn from a Maxwell-Boltzmann distribution, and the temperature was kept at 300 K with a Nosé-Hoover thermostat with an oscillation period of 0.5 ps. A conservative simulation timestep of 2 fs was used. Long-range electrostatics were treated with a reaction field, using a continuum dielectric constant of 78. Coordinates were saved every 50 ps, giving 1000 conformations for each individual trajectory. The simulation engine employed inside Copernicus was based on Gromacs version 4.5.3 [10] and the Amber03 force field [6, 20].

[2]available at `https://simtk.org/home/foldvillin`

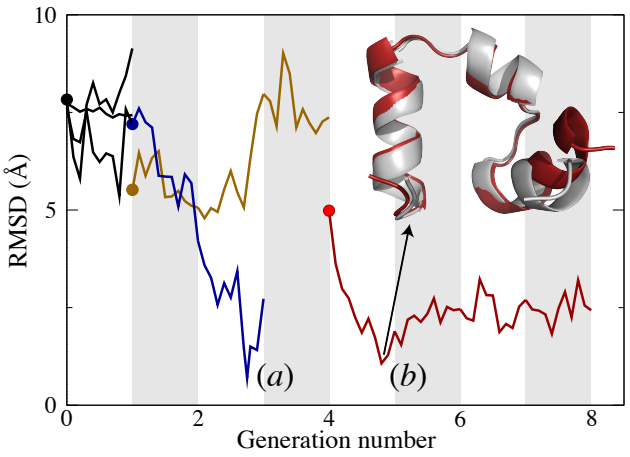

Figure 2: Per-generation evolution of a selection of villin trajectories in Copernicus, compared to the native structure. Three of the starting configuration trajectories are shown in black, followed for one generation. Based on the first generation clustering, new tasks were started. One of these is shown in orange, but it did not yield any low RMSD-conformations. A second trajectory resulted in the first folded conformation (a, as low as 0.7 Å RMSD from the native conformation). The red trajectory was started from the clustering after four generations, and is the one from which it was first possible to predict the folded conformation without prior knowledge of it (b, at 1.4 Å RMSD). This structure is shown in blue in the inset, and the native state in red, with the phenylalanine core shown explicitly in sticks.

3.2 Markov State Modeling

Markov State Models provide a natural way of understanding the nature of the free energy surface for any flexible molecule, such as a protein. The theory behind Markov State Models (or MSM's) have been outlined in detail elsewhere [16], but we repeat a few of the key concepts. Markov State Modeling is a kinetic clustering technique, in that it groups together conformations that can interconvert quickly. Thus, construction of a MSM requires a division of the high-dimensional free energy landscape into metastable states: a state partitioning.

In early phases of the simulation, the main task is identifying these metastable states. Especially when starting from unfolded conformations, much of conformation space is initially unexplored. Later in the simulation, the main task becomes accurate statistical estimation of the transitions between metastable states – determining the folding rate and mechanism. For this reason, one of the user-settable parameters for the Copernicus MSM controller is whether to use even or adaptive weighting in spawning new trajectories at each clustering step. Even weighting consists of running a uniform number of trajectories from each confor-

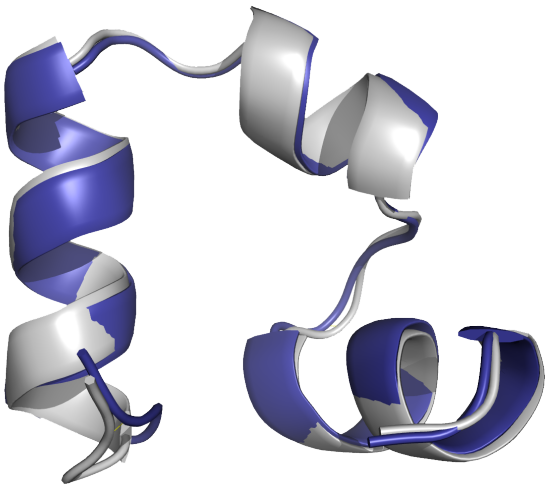

Figure 3: Superposition of a frame from the simulation trajectory that yielded the first observed folded villin structure (blue) with the experimentally determined native structure (light grey). The C_α RMSD between the structures is 0.7 Å.

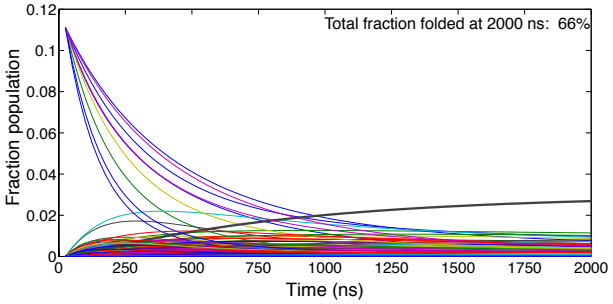

Figure 4: Time evolution of cluster populations in the microstate MSM. At 0 ns, all conformations are in one of the nine unfolded states. The emergence of a folded state is visible as a thicker black line. By 2 μs, a total 66% of the population has folded (within 3.5 Å of the native state).

mational cluster in the state partitioning. This is desirable when state partitioning is highly unstable, because then the limiting uncertainty is the state definitions themselves. As the state partitioning stabilizes, it becomes more advantageous to use adaptive weighting. This weights the number of trajectories started from each cluster by the uncertainty in the transitions between clusters. Once a set of stable state definitions has been achieved, adaptive weighting optimizes convergence of the kinetic properties of the model, which can boost sampling efficiency twofold compared to even weighting.

25 individual simulation tasks were generated for each unfolded configuration, giving a total of 225 50-ns trajectories in the first iteration. As soon as one trajectory finishes, it is reported back to the controller. The controller then extends the run by another 50 ns. After a sufficient number of trajectories have finished, the MSM controller performs clustering and adaptive sampling, marking trajectories for termination and spawning new trajectories as indicated. The number of new trajectories and the frequency of clustering and adaptive sampling are user-settable parameters, as is the number of clusters to generate and the time separation between structural snapshots. In this work we used 10,000 clusters and snapshots taken every 1.5 ns in each trajectory. Future versions will allow the values to be changed dynamically, since the optimal settings depend on the available compute resources, a fluctuating quantity on multi-user systems.

We tested how well the parallel adaptive molecular dynamics performed in a number of ways. Historically, obtaining a native-like conformation from an unfolded starting structure has been used as the measure of success in protein-folding [5]. After three generations of sampling (taking 30 h on our system), we obtain trajectories with conformations as low as 0.6–0.7 Å C_α RMSD of the native state, as shown in Fig. 3.

Such a metric, although used in many studies, of course requires prior knowledge of the native state and is somewhat artificial. To show the power of our adaptive simulation approach, we also identify the native state in a "blind" fashion. In Copernicus, the lowest free energy conformation can be predicted from the largest-population cluster at equilibrium, which is computed by examining distribution of the microstate transition matrix after it has achieved a stationary state. This cluster is scored by measuring the RMSD between the cluster and the native state, estimated as the average of five random samples. After eight generations of runtime (corresponding to 80-90 hours), the method predicts an equilibrium folded state approximately 1.4 Å from the native state (Fig. 2). A more detailed analysis shows that this cluster derived predominantly from a trajectory spawned in the fourth generation, which was then extended over three more generations (150 ns). It should be noted that by this time some other clusters were closer to the native state than the one predicted by this method; however, they could not be identified in a blind fashion.

An important strength of a converged kinetic model is that it allows prediction not only of the equilibrium distribution of states but also folding rates, mechanism, and any kinetic or thermodynamic quantities that we can compute on the individual structures. Here we are primarily concerned with computing this model as fast as possible and performing basic validation; more detailed examples of mechanistic insight from MSMs and ensemble simulation are given elsewhere [7, 11, 21, 3]. To validate the villin folding model, we examine the kinetics of folding. Starting from the unfolded states, we can calculate time evolution of the Markov State Model using the transition probability matrix $\mathrm{T}(\tau)$ calculated with a Markovian lag time τ,

$$p(t+\tau) = p(t)T(\tau). \quad (1)$$

For the microstate MSM built by Copernicus, we observe the $t_{1/2}$ for formation of a folded state to be approximately 500–600 ns (Fig. 4), in good agreement with the experimentally characterized folding time for this 35-NleNle variant of villin, which is around 700 ns. To do this, we constructed a Markov State Model with a lag time of 25 ns (a sensitivity analysis showed that the system became Markovian for lag times of 20 ns or greater). Analysis was performed on the largest connected subset of the Markovian transition matrix. Conformations within 3.5 Å RMSD of the native state were considered folded.

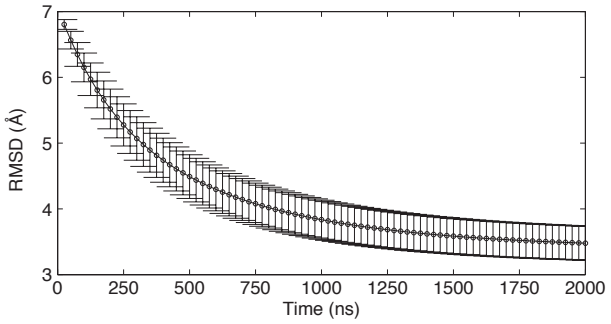

Figure 5: Time evolution of the average C_α RMSD from native for the villin ensemble. Error bars represent one standard deviation.

Despite the visual appeal of watching a single protein folding event, there is an increasing consensus in the field that quantitatively robust comparisons to experiment are needed. Indeed, the main strength of parallel adaptive molecular dynamics is its efficiency for computing ensemble properties such as are measured in experiments. As an example, Figure 5 illustrates the time evolution of the ensemble average of the C_α RMSD, including the statistical error.

4. PERFORMANCE

The all-encompassing goal for Copernicus has been to significantly reduce biomolecular simulation time-to-solution beyond the previous state-of-the-art. This requires good relative scaling, but we cannot afford to compromise absolute performance: in the real world, computational resources are scarce and have to be shared between projects and users. Copernicus has been designed to use Gromacs for molecular dynamics, with either multicore nodes or GPUs being used to maximize performance of individual simulations. As illustrated in Fig. 6, this provides Copernicus with several levels of parallelization. For the core nonbonded interactions we have manually written single-instruction multiple-data (SIMD) assembly kernels for a number of architectures (x86, PowerPC, BlueGene, IA64, etc.), and the kernels can also be executed on GPUs. On the intermediate level, multithreading is employed for multi-core nodes, and explicit message-passing (MPI) used for inter-node communication. This provides unmatched performance for a fixed number of cores, and when combined with parallel adaptive molecular dynamics implemented by message passing on the highest Copernicus level, it also scales to several orders of magnitude more cores than what has been possible before.

One reason for picking villin as an example application is the small size of the molecular system, a challenge for achieving strong scaling. At 9,864 atoms, Gromacs would reach a performance around 200 ns/day with 100 cores (or even 0.5 μs/day with long time steps, which were not used here), but this is roughly the limit of strong scaling on general-purpose hardware. If parallel adaptive molecular dynamics can employ thousands of cores for this system, it will scale to hundreds of thousands of cores for larger biomolecules.

Two different hardware resources were used for the villin benchmark: A Supermicro cluster of 180 24-core 2.1 GHz AMD Istanbul (4 6-core CPUs) nodes connected through QDR Infiniband (at roughly 2.7 GB/s), and a Cray XE6

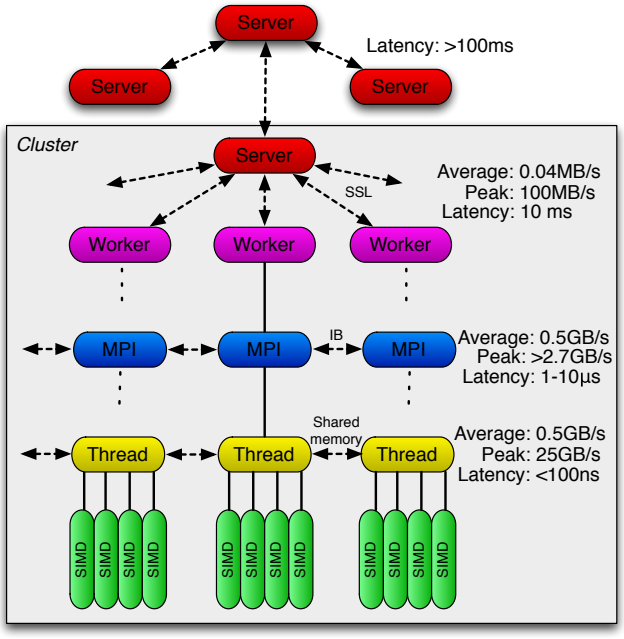

Figure 6: Scaling through multi-level parallelism in Copernicus. The nonbonded kernels use handtuned single-instruction multiple-data assembly, threads are used within nodes that share memory, and each Copernicus task is a massively parallel message-passing simulation that typically communicates over Infiniband. The average as well as peak bandwidth used for the villin example project is indicated. Beyond the point of efficiently scaling individual simulations, we employ hundreds of worker tasks on a typical cluster or supercomputer, and these in turn communicate with other resources through additional servers. Top-level servers interact with controllers to determine what tasks to execute. This hierarchical architecture adapts to successively higher latency, for instance when clusters on multiple continents are contributing to a project.

with 1516 24-core 2.1 GHz AMD Magny Cours nodes (2 12-core CPUs) connected with a Gemini 3D-torus (9.6 GB/s).

The complete project required only 100 hours of wallclock time, using 64–80 nodes on the Infiniband system and 96–144 nodes on the Cray simultaneously: a total of 3840–5376 nodes during that time. Successive MSM generations took 10-11 hours each, and the first folded villin conformation (see Fig. 3) was observed after roughly 30 hours of run time.

In our view, the most interesting property is the absolute performance – measured as time-to-solution – as a function of the total amount of hardware resources used. To compute this, we additionally benchmarked simulations with different numbers of cores and then simulated the controller's activity given different numbers of cores per task and total resources allocated. We used the time to observation of the first folded conformation as a stop criterion; the more realistic case of predicting the native state without *a priori* knowledge of it corresponds to roughly a factor 2.5 more time.

Fig. 7 shows the scaling efficiency of the villin run for different numbers of cores assigned to individual simulations.

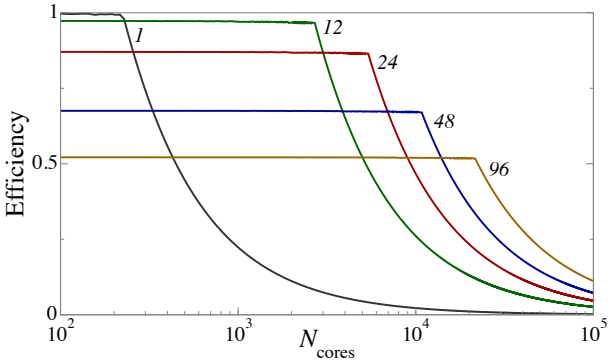

Figure 7: Scaling efficiency of the villin folding run as a function of total number of cores. The different lines represent different numbers of cores assigned to the individual Gromacs simulations, ranging from 1 to 96. While the relative efficiency of a single task is lower when it is parallelized over more cores, it significantly increases the total number of cores we can use use for Copernicus simulations.

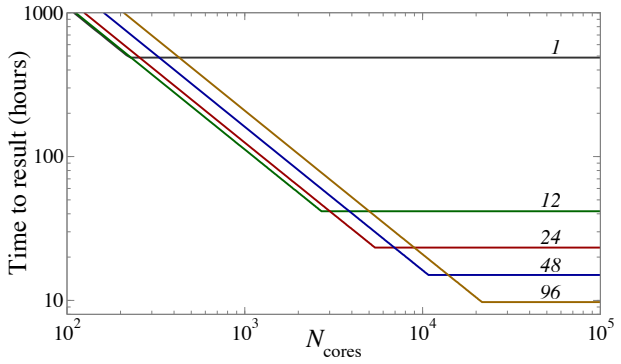

Figure 8: Total time to solution for folding of villin as a function of the total number of cores, with different number of cores assigned to the lowest-level simulations (colored lines). As long as there are idle commands in the queue it is more efficient to start additional simulations, but when this point is reached it becomes advantageous to further decompose individual simulations at a slight cost in relative scaling efficiency. The project reported in the text was run with 5,000 cores; using 20,000 cores the time to solution would have been just over 10h.

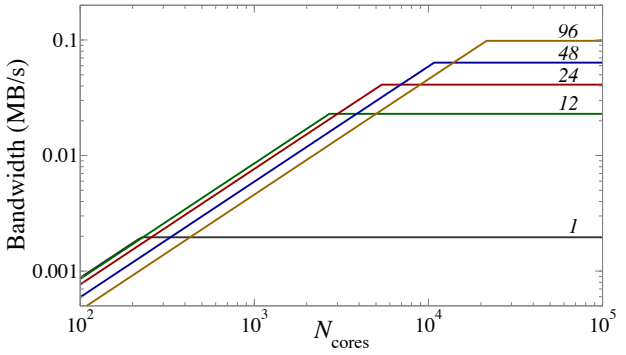

Figure 9: Average ensemble-level bandwidth use as a function of the total number of cores, for the same runs as Figs. 7 and 8.

There is a limit to how many simulation tasks we can use efficiently for parallel sampling of small molecules (we only used 225 tasks for villin), which causes the parallel efficiency to drop rapidly once this point is reached. The scaling efficiency is calculated as $t_{\text{res}}(1)/[Nt_{\text{res}}(N)]$, where $t_{\text{res}}(n)$ is the time it takes for n cores to run the entire set of MSM commands ($t_{\text{res}}(1) = 1.1 \cdot 10^5$ hours).

The corresponding wallclock time-to-solution for the project is shown in Fig. 8. This illustrates the trade-off involved in the scaling of individual commands vs. that of the run as a whole to achieve high absolute performance. For a given number of cores, the efficiency will drop slightly as commands are parallelized over more cores for the small villin system. However, the overall runtime will hit a limit imposed by the number of individual commands per generation that can meaningfully lead to a Markovian regime with MSM sampling. When the number of commands exceeds the number of commands per generation, the time to result ceases to decrease as a function of total number of cores. An easier way to think about this might be to compare to classical scaling: By parallelizing over more cores we might reduce efficiency slightly, but since it enables us to use more resources it will reduce the time-to-solution. Seen this way, Copernicus still achieves 53% scaling efficiency when using 20,000 cores to simulate folding of a 9,864-atom system.

Bandwidth and latency requirements are modest on the ensemble level. While the massively parallel part of a single villin simulation requires 500–2900 MB/s for 24 to 96 cores, the average bandwidth used for ensemble synchronization typically does not exceed 0.1 MB/s (Fig. 9). Because output data is transmitted to the server controlling the project, latency and available bandwidth do affect total run time slightly since each worker must wait until it receives a new workload. However, we estimate this time be no more than 30 seconds per day of running. In terms of the processing algorithm, this delay is hidden because the data transfers occur in parallel with project processing.

For bigger problem sizes, the strong scaling regime for Copernicus will typically increase more than proportionally to the system size. First, the underlying molecular dynamics implementation has close to ideal weak scaling [17], and the number of cores in each simulation can thus increase in proportion to the system size. Second, a larger system will typically have significantly more complex dynamics, which makes it possible to use many more parallel commands. We therefore expect studies of realistic normal-size protein dynamics to effectively use at least an order of magnitude more cores.

5. CONCLUSIONS

The parallel adaptive approach to molecular dynamics presented here is a powerful combination of the strongest aspects of parallel simulation, distributed computing, and Markov State Models. We have shown that the Copernicus framework effectively increases the scale at which biomolecular molecular dynamics simulations can be performed from a few hundred cores to many thousands and beyond, likely

reaching millions of cores for large molecules. In contrast to custom hardware that parallelizes a single simulation, Copernicus achieves this by using domain-specific knowledge to focus on the desired results of biomolecular simulations: quantitatively robust estimation of molecular reaction parameters. Our approach uses statistical mechanics knowledge to drive a message-passing parallelization scheme, and adapts this scheme to match the available computational resources by maximizing CPU and bandwidth use simultaneously.

Copernicus provides an open – but authenticated – peer-to-peer architecture for ensemble simulations in high performance computing. Currently, Copernicus comes with plugins to run Markov-State-Model-driven sampling and Bennett Acceptance Ratio free energy perturbation calculations [4]. In addition to the architectural improvements outlined above, further development will focus on making available controllers optimized for different sampling algorithms and questions in statistical mechanics modeling of biomolecules.

By using Copernicus, we have demonstrated efficient use of 5,376 cores to fold a small protein and predict the folded state without *a priori* knowledge within 3.5 days of wallclock time. As illustrated in the performance section, this could be pushed significantly further with more available resources. To obtain a similarly converged statistical model, a classical simulation would have to exceed 50 μs/day for villin, which is currently infeasible even with special-purpose hardware.

6. ACKNOWLEDGEMENTS

This work was supported by grants from the European Research Council (209825), the FP7 ScalaLife project, and the Swedish Research Council (2010-5107) to EL, as well as NSF MCB-0954714 and NIH R01-GM062868 to VSP. Supercomputing resources were provided by the Swedish National Infrastructure for Computing (014/10-31).

7. ADDITIONAL AUTHORS

8. REFERENCES

[1] S. Berneche and B. Roux. Energetics of ion conduction through the k+ channel. *Nature*, 414:73–77, 2001.

[2] G. R. Bowman, K. A. Beauchamp, G. Boxer, and V. S. Pande. Progress and challenges in the automated construction of markov state models for full protein systems. *J. Chem. Phys.*, 131:124101, Sep 2009.

[3] G. R. Bowman, V. A. Voelz, and V. S. Pande. Atomistic folding simulations of the five helix bundle protein lambda 6-85. *J. Am. Chem. Soc.*, 133:664–667, 2011.

[4] C. Chipot and A. Pohorille. *Free energy calculations*. Springer, Berlin, 2007.

[5] Y. Duan and P. A. Kollman. Pathways to a protein folding intermediate observed in a 1-microsecond simulation in aqueous solution. *Science*, 282:740–744, 1998.

[6] Y. Duan, C. Wu, S. Chowdhury, M. Lee, G. Xiong, W. Zhang, R. Yang, P. Cieplak, R. Luo, T. Lee, et al. A point-charge force field for molecular mechanics simulations of proteins based on condensed-phase quantum mechanical calculations. *J. Comp. Chem.*, 24(16):1999–2012, 2003.

[7] D. L. Ensign, P. M. Kasson, and V. S. Pande. Heterogeneity even at the speed limit of folding: Large-scale molecular dynamics study of a fast-folding variant of the villin headpiece. *J. Mol. Biol.*, 374:806–816, 2007.

[8] P. L. Freddolino and K. Schulten. Common structural transitions in explicit-solvent simulations of villin headpiece folding. *Biophys. J.*, 96:3772–3780, 2009.

[9] D. Frenkel and B. Smit. *Understanding Molecular Simulation*. Academic Press, London, second edition, 2002.

[10] B. Hess, C. Kutzner, D. van der Spoel, and E. Lindahl. Gromacs 4: Algorithms for highly efficient, load-balanced, and scalable molecular simulation. *J. Chem. Theory Comput.*, 4(3):435–447, 2008.

[11] P. M. Kasson, N. W. Kelley, N. Singhal, M. Vrljic, A. T. Brunger, and V. S. Pande. Ensemble molecular dynamics yields submillisecond kinetics and intermediates of membrane fusion. *Proc. Natl. Acad. Sci. U.S.A.*, 103:11916–11921, 2006.

[12] J. Kubelka, T. Chiu, D. Davies, W. Eaton, and J. Hofrichter. Sub-microsecond protein folding. *J. Mol. Biol.*, 359:546–553, 2006.

[13] D. P. Landau and K. Binder. *A Guide to Monte Carlo Simulations in Statistical Physics*. Cambridge University Press, Cambridge, 2000.

[14] D. L. Mobley, A. P. Graves, J. D. Chodera, A. C. McReynolds, B. K. Shoichet, and K. A. Dill. Predicting absolute ligand binding free energies to a simple model site. *J. Mol. Biol.*, 371(4):1118–34, Aug 2007.

[15] V. Pande, I. Baker, J. Chapman, S. Elmer, S. Khaliq, S. Larson, Y. Rhee, M. Shirts, C. Snow, E. Sorin, et al. Atomistic protein folding simulations on the submillisecond time scale using worldwide distributed computing. *Biopolymers*, 68(1):91–109, 2003.

[16] V. S. Pande, K. Beauchamp, and G. R. Bowman. Everything you wanted to know about markov state models but were afraid to ask. *Methods*, 52:99–105, Sep 2010.

[17] R. Schulz, B. Lindner, L. Petridis, and J. C. Smith. Scaling of multimillion-atom biological molecular dynamics simulation on a petascale supercomputer. *J. Chem. Theory Comput.*, 5(10):2798–2808, 2009.

[18] D. E. Shaw, R. O. Dror, J. K. Salmon, J. Grossman, K. M. Mackenzie, J. A. Bank, C. Young, M. M. Deneroff, B. Batson, K. J. Bowers, E. Chow, M. P. Eastwood, D. J. Ierardi, J. L. Klepeis, J. S. Kuskin, R. H. Larson, K. Lindorff-Larsen, P. Maragakis, M. A. Moraes, S. Piana, Y. Shan, and B. Towles. Millisecond-scale molecular dynamics simulations on anton. In ACM, editor, *Proceedings of the Conference on High Performance Computing, Networking, Storage and Analysis (SC09)*, New York, NY, 2009.

[19] D. E. Shaw, P. Maragakis, K. Lindorff-Larsen, S. Piana, R. O. Dror, M. P. Eastwood, J. A. Bank, J. M. Jumper, J. K. Salmon, Y. Shan, and W. Wriggers. Atomic-level characterization of the structural dynamics of proteins. *Science*, 330:341–346, 2010.

[20] E. Sorin and V. Pande. Exploring the helix-coil transition via all-atom equilibrium ensemble

simulations. *Biophys. J.*, 88(4):2472–2493, 2005.

[21] V. A. Voelz, V. R. Singh, W. J. Wedemeyer, L. J. Lapidus, and V. S. Pande. Unfolded state dynamics and structure of protein l characterized by simulation and experiment. *J. Am. Chem. Soc.*, 132:4702–4709, 2010.