From Proceedings of High-Performance Distributed Computing
(HPDC-3) – August 1994

Falcon – Toward Interactive Parallel Programs: The On-line Steering of a Molecular Dynamics Application

Greg Eisenhauer
Karsten Schwan
Weiming Gu
Niru Mallavarupu
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332

Abstract

This paper focuses on the opportunities and costs of on-line steering as applied to a substantial parallel application. We demonstrate potential performance improvements through the use of the Falcon system, an experimental system for the on-line monitoring and steering of parallel programs. The visual presentation of program output along with animated displays of program performance information via Falcon's monitoring system enables the on-line capture, analysis, and display of program information required for program steering. Falcon also provides the mechanisms for the manipulations of program state that accomplish this online steering.

1 Interactive parallel programs

Future parallel applications will run on a variety of parallel machines linked with high performance networks. The primary purpose of our research is to exploit these modern systems' capabilities to offer human-interactive interfaces that can execute simultaneously with a parallel application's computational and storage tasks. Specifically, we are exploring the potential for increases in performance and functionality gained by the on-line interaction of end users with their supercomputer applications on single and on networked parallel machines. Unlike the efforts of the scientific visualization community which are often targeted towards interaction with the *data* produced by parallel programs, we are concerned with interaction with the programs themselves. Such on-line interactions have already been identified as highly desirable by both end users and program implementors, and these interactions are collectively referred to as *interactive program steering.*

The program steering addressed by our work encompasses rapid, on-line program changes that require built-in, custom monitoring and on-line algorithms determining and enacting such changes (e.g., in research addressing configurable operating systems[MS93]). It also includes gradual changes to long-running real-time programs[BS91] and scientific applications able to interact at human speeds via user interfaces, such as the application presented in this paper. Regardless of the speed and nature of interaction, program steering is based on the on-line capture of information defining current program and configuration state[Bru91, MW91, OSS93], and it assumes that human users or algorithms inspect such information and manipulate it to make steering decisions.

This paper focuses on the opportunities and costs of on-line steering as applied to one substantial parallel application, a molecular dynamics simulation (MD) used by physicists for exploration of elementary properties of lubricants. We demonstrate the potential of improving performance by on-line steering with experimental results attained on a 64-node KSR-1 supercomputer. An experimental system for the on-line monitoring and steering of parallel programs – the Falcon system – is used in interactive runs of the MD code. Falcon improves program performance substantially because it allows the user to manipulate the program's data decomposition in order to balance the load among different parallel execution threads. The visual presentation of program output along with animated displays of program performance information via Falcon's monitoring system enables the on-line capture, analysis, and display of program information required for program steering. Additionally, the Falcon system is portable across multiple parallel computing platforms, including the KSR machine, SGI machines, and other shared memory architectures.

2 Steering a parallel application

2.1 Why program steering?

The on-line manipulation of parallel and distributed programs has been shown to increase performance in many domains. These performance increases generally result from one of three types of manipulation. System- or program-specific on-line algorithms select and enact program or system changes to provide one type of performance advantage. Examples of this include the rapid automatic configuration of small program fragments for maintaining real-time response in uniprocessor systems[MP89], the on-line adaptation of functional program components for realizing reliability/performance tradeoffs in parallel and real-time applications[GS89], and load balancing or program configuration for enhanced reliability in distributed systems[MW91].

Performance gains can also be realized from the manipulation of specific program attributes during execution. Examples of program attribute manipulation include changes to:

- changes to the implementation of critical program components to balance execution speeds against resource requirements,
- changes to program solution algorithms that affect convergence, and
- changes to replicated program components' states to enhance or maintain reliability.

On-line program manipulations may also change a program's execution environment. Examples of environment changes include the balancing of system loads in order to guarantee acceptable rates of progress to different program components, or the configuration of operating system or even communication protocol components to tailor their functionality to different phases of a program's execution[MS93].

A final form that program steering may take is to have users direct their computations towards "interesting" or "meaningful" subdomains. Many current large scale computations create immense quantities of data, but only a fraction of that data may be relevant to the phenomenon being studied. Interactive direction of the program could reduce data handling problems as well as provide opportunities for performance improvements through calculating "uninteresting" areas with less than full fidelity.

In this paper we examine a molecular dynamics application in which domain decompositions can be changed interactively in order to adjust processor loads and improve performance under changing experimental conditions.

2.2 The parallel MD code

MD is an interactive molecular dynamics simulation used to explore the statistical mechanics of complex liquids[XORL92]. The specific physical system modeled is a time-stepped simulation of the behavior of hydrocarbon chains on a fixed substrate. The dominant computational requirement is the calculation of long-range forces between particles, but other required computations with different characteristics also affect the application's structure and behavior. These computations include finding the bond forces within the hydrocarbon chains, determining system-wide characteristics such as atomic temperature, and performing analysis and on-line visualization.

The implementation of the MD application attains parallelism by domain decomposition. MD assumes that decomposition changes only slowly over time and that computations in different sub-domains are independent outside some cutoff radius. Inside this cutoff radius, information must be exchanged between neighboring particles, so that different processes must communicate and synchronize between simulation steps. The resulting overheads are moderate for fairly coarse decompositions (e.g., 100-1000 particles per process) but unacceptable for finer grain decompositions (e.g., 10 particles per process).

Our simulated system is used by physicists at Georgia Tech[XORL92] to investigate evolutionary trends of the structure and dynamics of *n*-hexadecane (C_{16}-H_{34}) films on a crystalline substrate modeling $Au(001)$. In the simulation the alkane system is described via intramolecular and intermolecular interactions between pseudoatoms (CH_2 and terminal CH_3 segments) and the substrate atoms. The calculational cell is a square cylinder which is periodically repeated in the x-y directions. Temperature is controlled via infrequent scaling of the particles' velocities. The base system contains 4800 particles in the alkane film and 2700 particles in the crystalline base. For purposes of experimentation, the sample system may be scaled in size resulting in good ratios of access to local vs. remote data for individual processes even on larger scale parallel MIMD architectures (e.g., machines with more than 100 processors).

Domain decomposition of the MD code takes advantage of the molecular structure of the alkanes and substrate. Specifically, the physical system is divided into spatial domains, and each domain is allocated to a virtual processor. Molecules are assigned to domains by their centers of mass. The processor on which a domain is placed is responsible for calculating the forces on the domain's alkanes. In order to calculate the forces on its alkanes due to the presence of alkanes in other domains, each processor must obtain information from all spatial domains whose particles might affect its own, called *neighbor domains*. The number of neighbor domains is determined by the geometry of the domains, the sizes of the domains, and the size of the cutoff radius. The information flow between domains represents the main communication requirement for this class of MD applications, and it must take place once for each simulation timestep. Other communication requirements include the occasional need to calculate a global temperature value to control the temperature of the simulation.

While the simplest decompositions are slices over one or more dimensions, the MD code is quite flexible regarding decomposition geometries. This flexibility is important because a spatial domain and its molecules comprise the unit of allocation to a processor, so that the spatial decomposition strongly affects the resulting communication patterns, computational load balancing, and subsequently, the performance of the parallel code. MD is written so that, given the set of domains and their neighbor lists, any style of decomposition can be supported. This is also of considerable importance when simulating oddly-shaped or irregular systems. Section 3 will consider some performance effects of different spatial decompositions over the sample system.

Because the nature of the decomposition has a dramatic effect upon system performance and because the disposition of the particles in the system changes over time, it is important to consider the possibility of changing the decomposition dynamically while the simulation is running. While support for such dynamic changes was not part of the original MD design, the mechanisms in place to handle molecules moving from one domain to another support such simple decomposition changes such as shifting the domain boundary. If the domain boundary changes between iterations, the "transferred" molecules are simply treated like molecules whose motion has carried them from one domain to another. This property of MD made decomposition steering an obvious early target for experimentation.

2.3 On-line steering of the MD code

Steering opportunities. In general, desirable online interactions of human users with the parallel MD code can occur both during the program's use for research in computational physics and during debugging or performance tuning. We are most concerned with the interpretation and use of program output by end users and with the use of performance attributes by program developers. We are less concerned with interactive debugging where developers arbitrarily start, stop, or incrementally execute their programs.

The following on-line interactions with the MD code have potential for improving the computational performance of the application:

- Decomposition geometries could be modified to match physical systems, where a slab-based geometry is useful initially and a pyramidal geometry would become useful when a probe is lowered into the physical system.
- End users could provide high-level guidance concerning possible computational shortcuts, such as the reuse of data from previous iterations in next steps.
- The interactive modification of cutoff radius could improve solution speed by computing uninteresting time steps with some loss of fidelity, again requiring the involvement of end users.

The increased performance also enhances the usefulness of the code for performing materials science research because scientists using programs like the MD simulation can benefit from the on-line interactions described above by gaining more rapid insights into material properties and behaviors. Program performance can also be increased or tuned by on-line program steering performed by algorithms that automatically evaluate program information in order to adjust selected program attributes. For example, algorithms could automatically perform the following steering functions:

- The boundaries of spatial decompositions could be shifted for dynamic load balancing among multiple processes operating on different subdomains. Simple shifts could be performed by automatic algorithms, whereas more radical geometry shifts would require human interaction.
- Global temperature calculations reduce simulation speed because they require a globally consistent state. Such calculations could be replaced by less accurate local temperature control. On-line analysis could determine how often global computations must be performed based on the temperature stability of the system.

Although we have not yet completed our evaluation of on-line program steering for the MD code, our results have already demonstrated the utility as well as the performance advantages of on-line steering. Specifically, we can demonstrate the performance advantages derived from direct end user involvement in changing decomposition boundaries for dynamic improvements in load balancing among different computational execution threads (see Section 3.2).

Steering assumptions and issues. Several assumptions have been made concerning program steering. First, the programmers must understand the costs and benefits derived from on-line steering. This understanding is necessary because we must rely on the programmers to implement the actual configurable program components required for steering. For example, consider data structures used by MD's execution threads for 'lookup' of molecules. If such structures are not designed to permit the addition or deletion of molecules from individual threads' work lists, then the on-line changes in domain decomposition suggested in the previous section are not possible.

We also assume that the MD application should be minimally affected in performance by the presence of program steering. Accordingly we cannot use overly expensive mechanisms for program monitoring, such as trace files containing complete program state information. We also cannot use simple approaches to steering such as including into application code 'wait' statements seeking user input before the code proceeds with further computations. Such highly intrusive approaches to program steering would not be adopted by physical scientists who are already seeking to maximally utilize available computation time for solving their scientific and engineering problems. Furthermore, steering performance increases would not be achieved if the steering itself were overly intrusive.

One last issue with program steering is the decision as to whether certain on-line program interactions should be part of or outside of a parallel application program. This must be decided on a case by case basis. Certainly, most parallel codes routinely contain specific algorithms adjusting selected application parameters in order to maintain reasonable convergence of iterative solution methods, to ensure the validity of experiment parameters, etc. Such algorithms are rightly part of the parallel program. At the same time, many other steering algorithms should be separately stated and changed from the application program's code, including steering algorithms that respond to changes in the target hardware or in the program's execution environment, adjust operating system level abstractions used by the program, change the values of input/output program parameters, or adjust program parameters under user control.

3 Parallel MD and steering

For the purposes of evaluating the practical utility of program steering, we consider the MD application running on a 64-node Kendall Square Research KSR-1 machine. The MD application itself is built upon Cthreads, a user-level threads facility that is built on top of the kernel-level threads facility provided by the KSR OS. The advantages of Cthreads include increased performance compared to Pthreads for thread creation, context switches, and synchronization. Cthreads also gives users the flexibility to design and use thread primitives customized to their applications. The Falcon monitoring and steering system is strongly, but not inseparably, tied to Cthreads.

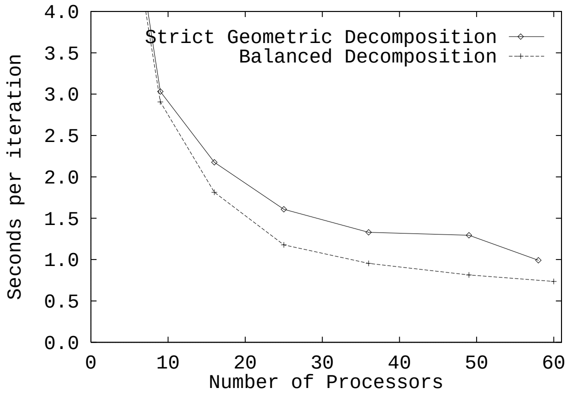

Figure 1: **Iteration Times for the Strict and Balanced Decompositions**

3.1 General performance

Normative comparisons of the KSR-1 MD code and basic speedup results are presented in [ES94]. They demonstrate performance similar to what is attained with comparable Fortran codes running on a Cray Y-MP and the KSR-1 itself. Speedup results show continuing improvements even for large numbers of processors (up to 60 on our KSR-1 machine). More germane to this paper is the observation that *computational costs* change in MD when running different systems or using different domain decompositions. The factors that affect performance are both characteristics of the simulated system (such as density and cutoff radius) as well as the parameters which can be specified to MD (such as the number of processors employed and the decomposition scheme).

As an example of how sensitive program behavior can be to variations of these parameters, consider the performance differences between two slightly different styles of establishing the boundaries between spatial domains. To achieve the better performance numbers in [ES94], we did *not* rely strictly on geometry to establish the domain boundaries. Instead we allowed the boundaries to shift slightly in either direction in order to balance the number of molecules assigned to each domain. We call this *balancing* the decomposition.

Even with a relatively uniform distribution of molecules, a strict geometric decomposition can result in a poorly balanced set of domains. Allowing the domain boundaries to shift slightly can result in a decomposition where the molecules are more evenly distributed. The number of molecules in a domain greatly influences time it takes to calculate an iteration for that domain. The overall time will be lower if all the domains require about the same amount of time.

Figure 1 illustrates the beneficial effects of balancing the domains in a slab decomposition of our sample system. The balanced and geometric decompositions perform similarly if there are few processors. However, the probability of an unbalanced decomposition increases as the work is more thinly spread. As a result, the performance of the strict geometric decomposition is somewhat erratic and generally worse than the balanced decomposition. These results also demonstrate the importance of dynamic load balancing as molecules move during the course of the simulation. The domain boundary shifts necessary to do dynamic rebalancing are among the class of simple decomposition changes that can be easily handled by mechanisms already in place in MD.

3.2 Performance with program steering

The previous section demonstrates the importance of load balancing to MD performance. Program steering can make such load balancing easy to perform, on-line and continuously. For simplicity, the program run with steering is performed with only 4 threads computing distinct slabs of the material being simulated. One additional thread is collecting and computing global state like total energy and performing visualization processing (ie., displaying current particle and molecule positions). The Falcon tool uses a second additional thread, called a *local monitor* thread, which collects all monitoring outputs from the running program. The local monitor filters output as much as possible and then forwards it to any interested parties for such needs as on-line steering, display, or long term storage for postmortem analysis.

A decomposition into horizontal slabs is interesting for steering because obvious load balancing heuristics give fairly poor results. We use application steering to adjust the slab boundaries at run-time to yield better speedups. In particular, consider the initial decomposition depicted in Figure 2. In this configuration, decomposition boundaries are determined by simply dividing the vertical extent of the alkane layer evenly into 4 ranges. Note that the top-most domain has fewer particles and the bottom-most domain will have more interactions with the fixed substrate. This situation yields poor load balancing for the initial decomposition. Using Falcon the user can visualize this load

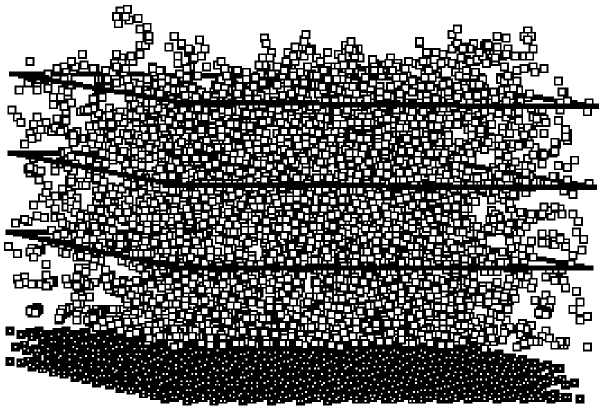

Figure 2: **Initial Decomposition of the Steered System.**[1]

imbalance with a simple bar-graph view.

Load imbalances can be corrected by adjusting the positions of the boundaries among particles assigned to execution threads. However, without an exceptionally accurate model of the behavior of the parallel application, there is no a priori way to know what boundary positions will best balance loads. User-directed steering during program execution can be used instead, thereby avoiding the construction of complex models for particle and thread mapping. We use online steering to improve on the results of simple initial mapping models during program execution. Specifically, in the MD code, application steering can adjust boundary positions after or during each simulation time step, monitor the resulting program performance, then readjust, etc.

Figure 3 shows the performance effects of such steering for several timesteps taken by the MD code (indicated by speedup as compared to a single processor). Each of the steering events is actually a manual adjustment of one or more of the partition boundaries by the user. Guided by the information contained in the Load Balancing bar-graph, the user can gradually adjust the system toward a more appropriate set of boundaries. For dynamically changing systems (e.g., descending probes), similar techniques can be used for continuous boundary adjustment. Once system performance properties and the associated boundary adjustments are understood, the end user can write 'scripts' or configuration algorithms that capitalize on the knowledge about load balancing gained from earlier system runs.

[1] The horizontal frames mark the boundaries between processor domains. The dark particles are the fixed substrate while the lighter particles are the alkane chains.

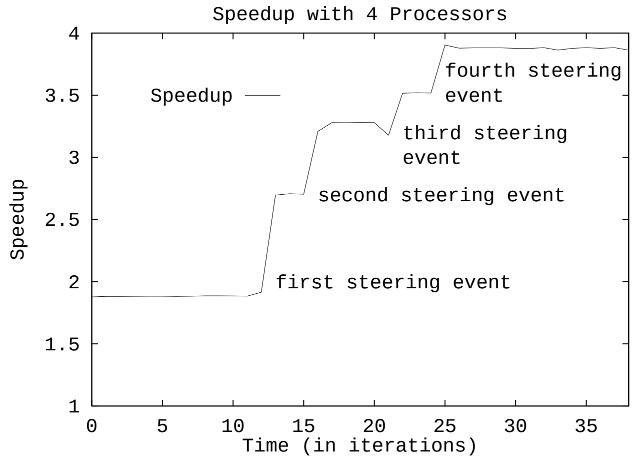

Figure 3: **The Effect of Steering on Performance Over Time with 4 Processors.**

4 The Falcon system

Programs cannot be steered without input from both program users and implementors. End users must formulate steerable program attributes and determine which program attributes must be monitored based on the program steering decisions to be made. They must also determine appropriate visual displays of monitored program state and steering attributes. To support end users in these tasks, tool developers need to provide mechanisms for program instrumentation and steering as well as tools for the construction of visual displays. In addition, default program monitoring may be provided in order to enhance a programmer's understanding of the application's performance on specific target architectures. Falcon provides an on-line monitoring and steering system with the following elements:

- language constructs for specification of application-specific monitoring as well as monitoring facilities supplied by default for arbitrary threads-based parallel programs,
- mechanisms (e.g., local monitor threads) for the efficient on-line capture, analysis, and display for program information,
- built-in displays for understanding the performance of threads-based parallel programs, and
- a growing library of facilities that permit end users and program implementors to steer their parallel code using on-line monitoring data.

4.1 Specification, capture, and analysis of program information

The main contribution of Falcon is its ability to analyze and display on-line both application-specific program information (e.g., values of specific program variables like 'particle energy') as well as generic, useful performance information (e.g., processor utilization)[KS91]. The technical approach chosen for specification of program monitoring is language and target machine independent. Specifically, we assume that implementors explicitly specify both *monitoring* and *configuration* program *attributes* either statically and/or dynamically. Falcon permits the on-line inspection of such attributes as well as the association of display primitives with such attributes. By viewing the attributes end users can understand the captured program state. Falcon supports program steering by permitting the on-line alteration of *configuration attributes*. Future work will permit the association of monitoring attributes with steering algorithms that automatically change configuration attributes when triggered by monitoring events.

We previously developed language primitives for stating high-level monitoring specifications[KSO90]. The current implementation of Falcon focuses on developing runtime library support in order to investigate effective parallelizations of monitoring activities as well as using alternative methods for monitoring. The library offers the ability to adhere to specific timing constraints regarding the on-line execution of monitoring and visualization code with respect to the execution of an application's core computations. In addition, Falcon offers a *sensor* language[KS91] with which program-dependent monitoring attributes may be stated and implemented. A higher-level *view* language and configuration language and library support are being developed for support of large-scale program monitoring and display. An easy-to-use system for inclusion of custom sensors with user code will also be part of the Falcon tool.

A major advantage of Falcon over other monitoring systems[LMC88, MCH+90] and tools like Paragraph is its ability to accept user-specifications that state what and how monitoring should be performed for a specific user program. As a result, users can easily construct just the monitoring views they desire[KS91], in addition to or instead of the default views provided with the system. This capability is especially important for end users who wish to debug and steer their parallel applications using views concerning their output data, then adjusting attributes like 'time step size', etc. Another difference of Falcon from other work is that users can also define their own graphics or use existing graphical displays for displaying their programs' results. As a result, programmers can construct performance monitors and displays that illuminate specific aspects of their parallel programs, such as displays of performance-relevant data structures and

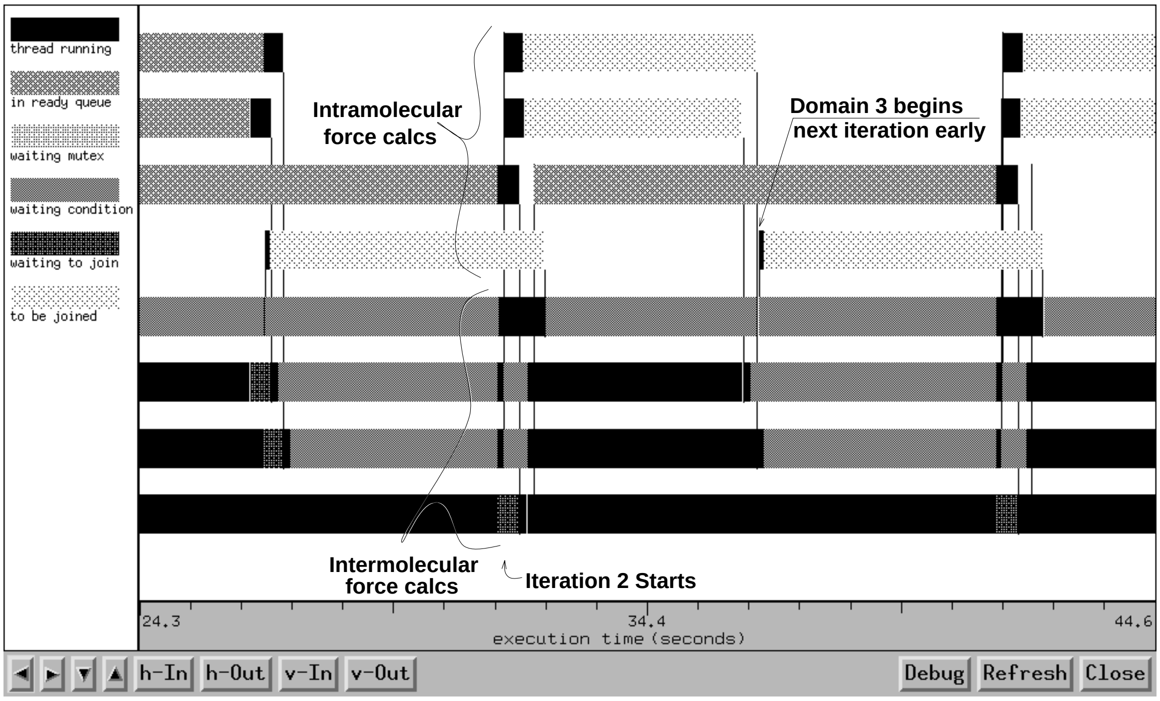

Figure 4: **An annotated lifetime view. This color display is rendered in monochrome for presentation.**

their access patterns. Toward this end, we use standard graphical tools, such as X windows and Motif for Falcon's graphical displays, and we use existing tools such as Polka [Sta91] for program animation.

4.2 Performance understanding

Falcon offers a default *"thread lifetime" view* of a thread program that is invaluable for understanding and evaluating the performance of the MD code. This 'lifetime' view of a threads program relies on the fact that the Cthreads library has instrumented all calls to its constructs, including thread fork, thread exit, lock, unlock, etc. Given this information, the threads lifetime view shows the different states of threads over time. Figure 4 depicts a typical thread lifetime view for the steered system presented in the previous section. (Please note that the real lifetime view is an active color display that can be only poorly represented as a static monochrome figure.) The figure presented has been annotated with text and arrows for purposes of illustration. In the actual display, threads are identified by color and thread states by fill patterns. Thread names and colors are matched in an accompanying display. In the monochrome figure executing threads show as relatively even shades. Threads waiting for a condition are shown as a dashed box in the timeline. Threads ready to run but waiting the the run queue have a diagonal crossed pattern. This lifetime view has been invaluable for gaining insight into aspects of program behavior such as the relative waiting times, co-execution properties between threads, etc.

5 Conclusions

Falcon has been shown capable of supporting on-line program steering by providing users with run-time performance and application-specific information about their parallel programs. Falcon's monitoring mechanisms are implemented with the Cthreads parallel programming library for a number of hardware platforms, including the KSR-1, SGI machines, and SUN SPARCstations. The MD code's performance is shown to improve significantly when using on-line steering, in this case due to the difficulties in domain decomposition of the simulated system.

The next steps in our research address scalability, ease of use, and program steering:

- We are investigating how threads-based monitoring views and graphical displays may be scaled to programs with thousands or at least, hundreds of execution threads.
- We are developing high-level specifications of monitoring views for large-scale parallel programs, in conjunction with 'views' that concern

program steering.

- We are developing general mechanisms for program steering, much like the runtime library now underlying the Falcon monitoring system.

In the long term, we will embed the Falcon system into a complete environment for parallel programming, called the LOOM system. LOOM will provide an integrated environment for creating, understanding, tuning, and steering parallel programs. It will offer a uniform model for the representation and access to program and performance information, libraries for efficient on-line and postmortem access to such information by multiple programming tools (e.g., the performance monitor vs. the steering interface), and graphical displays of parallel programs useful for program restructuring and tuning.

World Wide Web Access

Full color screen captures of the figures in this paper as well as additional information on the Falcon project can be found on WWW at the URL: **http://www.gatech.edu/systems/pardist/papers/hpdc3-steering.html**.

References

[Bru91] Bernd Bruegge. A portable platform for distributed event environments. In *ACM/ONR Workshop on Parallel and Distributed Debugging, Santa Cruz*, pages 184–193, May 1991.

[BS91] T. Bihari and K. Schwan. Dynamic adaptation of real-time software. *ACM Transactions on Computer Systems*, 9(2):143–174, May 1991.

[ES94] Greg Eisenhauer and Karsten Schwan. Md -– a flexible framework for high-speed parallel molecular dynamics. In *High Performance Computing, Proceedings of the 1994 SCS Simulation Multiconference*, pages 70–75. Society for Computer Simulation, Society for Computer Simulation, April 1994.

[GS89] Prabha Gopinath and Karsten Schwan. Chaos: Why one cannot have only an operating system for real-time applications. *SIGOPS Notices*, pages 106–125, July 1989.

[KS91] Carol Kilpatrick and Karsten Schwan. Chaosmon – application-specific monitoring and display of performance information for parallel and distributed systems. In *ACM Workshop on Parallel and Distributed Debugging*, pages 57–67. ACM SIGPLAN Notices, Vol. 26, No. 12, May 1991.

[KSO90] Carol Kilpatrick, Karsten Schwan, and David Ogle. Using languages for describing capture, analysis, and display of performance information for parallel and distributed applications. In *International Conference on Computer Languages '90, New Orleans*, pages 180–189. IEEE, March 1990.

[LMC88] T. Leblanc and J. Mellor-Crummey. Debugging parallel programs with instant replay. *IEEE Transactions on Computers*, C(36), April 1988.

[MCH+90] Barton P. Miller, Morgan Clark, Jeff Hollingsworth, Steven Kirstead, Sek-See Lim, and Timothy Torzewski. Ips-2: The second generation of a parallel program measurement system. *IEEE Transactions on Parallel and Distributed Systems*, 1(2):206–217, April 1990.

[MP89] Henry Massalin and Calton Pu. Threads and input/output in the synthesis kernel. In *Proceedings of the 12th Symposium on Operating Systems Principles*, pages 191–201. SIGOPS, Assoc. Comput. Mach., Dec. 1989.

[MS93] Bodhisattwa Mukherjee and Karsten Schwan. Improving performance by use of adaptive objects: Experimentation with a configurable multiprocessor thread package. In *Proc. of Second International Symposium on High Performance Distributed Computing (HPDC-2)*, pages 59–66, July 1993.

[MW91] Keith Marzullo and Mark Wood. Making real-time systems reactive. *ACM Operating Systems Review*, 25(1), Jan. 1991.

[OSS93] D.M. Ogle, K. Schwan, and R. Snodgrass. Application-dependent dynamic monitoring of distributed and parallel systems. *IEEE Transactions on Parallel and Distributed Systems*, 4(7):762–778, July 1993.

[Sta91] John T. Stasko. Using direct manipulation to build algorithm animations by demonstration. In *Proceedings of the ACM SIGCHI '91 Conference on Human Factors in Computing Systems, New Orleans*, pages 307–314, May 1991.

[XORL92] T. K. Xia, Jian Ouyang, M. W. Ribarsky, and Uzi Landman. Interfacial alkane films. *Physical Review Letters*, 69(13):1967–1970, 28 September 1992.