

# MDSIMAIID: From Fast Electrostatics to Parallel Scaling

Michael S. Crocker

Department of Computer Science and Engineering  
Fall Semester 2004  
CSE 498K Computational Biology

**Abstract**—MDSimAid, an automated recommender system, was initially developed to optimize parameters of fast electrostatic algorithms in Molecular Dynamics (MD) simulators. It has since been enhanced to optimize the parallel execution of MD simulators by perfecting load balancing and reducing communication time between parallel processors.

## I. OPTIMIZATION FOR PARALLEL EXECUTION

Up to this point, MDSimAid [1] has been focused on optimizations for the MultiGrid (MG) [2] and Particle Mesh Ewald (PME) [3] fast electrostatic algorithms. This approach is limited because the algorithms themselves can only be improved up to a point. Even if the parameters for these algorithms are tweaked to perfection, there is still a limit to how much the runtime speedup can be improved. Even if the calculations can be efficiently and evenly distributed to the various grid points for improved speed, those calculations still need to be done.

There are however, different ways to speed up Molecular Dynamics (MD) simulations. Optimizing parameters for the electrostatic methods is only one way. More recently, we have been looking for other ways that MDSimAid can aid in decreasing the runtime of these simulations. The most obvious is through parallelization. Many of the MD simulators, including NAMD [4] and ProtoMol [5], are designed to run on parallel processors. In most cases, running in parallel requires compiling the simulator with a slightly different configuration.

When optimizing a parallel simulation, there are some qualities to consider. First, how many processors are available for a parallel run? Second, how well do the simulator and the algorithms used by the simulator scale to multiple processors? Finally, how can the simulation be influenced in terms of splitting the workload between the multiple processors, and how can the communication be optimized? MDSimAid deals with these issues in much the same way that it did with the parameters for the fast electrostatic algorithms.

First consider the number of processors available. If there are  $N$  processors available for the MD simulator to use, then in a perfect world with no communication overhead and an ability to equally divide the computational work into  $N$  parts, the simulation should experience an  $N$ -fold speedup. Sixteen processors should provide simulation results six-

teen times faster than with only one. This, however, is not the case.

There are many factors that keep parallel scaling efficiency down. Some algorithms are better at being split into different threads of execution. In general, the MG method can be scaled to more processors than the PME method before becoming highly inefficient. Communication between threads is another critical issue. Some methods require communication more often. Different types of physical interconnect between processors are faster than others. It is important when running parallel simulations that the communication between processors be as fast as possible. There are also different parameters that dictate how communication is handled between the parallel processes.

While MDSimAid could never optimize parallel execution for perfect speedup, there are still many improvements that can be obtained by searching the parameter space as it pertains to parallel processing. The most important parameter to optimize is the number of processors. Initial tests performed to observe the possibilities of optimizing for parallelization were done using ProtoMol. See figures 1, 2(a), and 2(b).

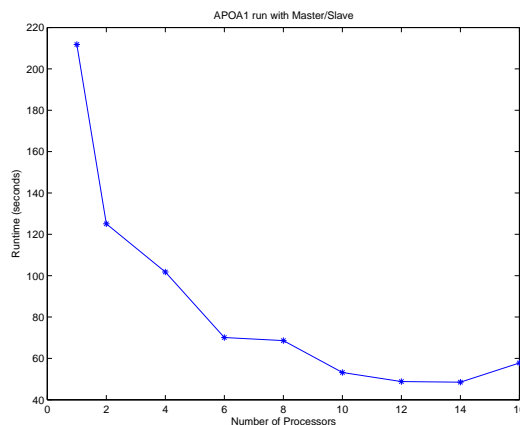
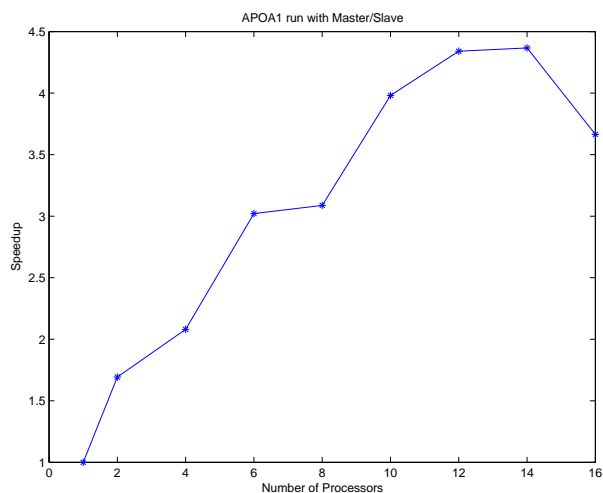
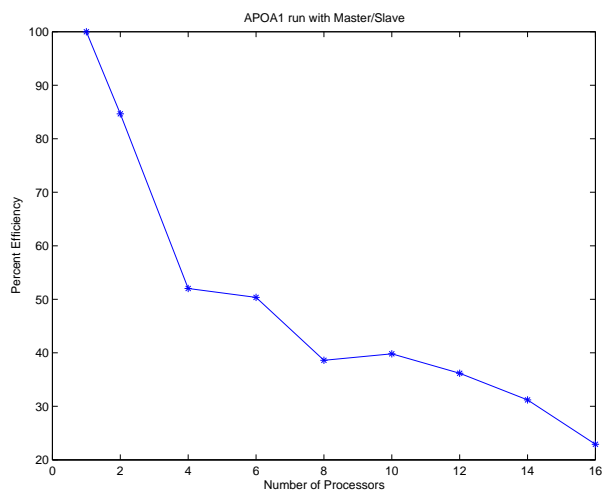


Fig. 1. Parallel runtime with different numbers of processors.

Excluding the number of processors, the goal for MDSimAid's optimization of parallel runs has to do mostly with load balancing. When the computation from the MD simulator's algorithms are divided between processors, the



(a)



(b)

Fig. 2. (a) Speedup and (b) Efficiency for N processors.

amount of work that each must do should be equal. If the load balancing is not equal, some processors will finish before others and have to wait until all the others are done. As the balance becomes better and better, the overall runtime will improve because very little time will be wasted waiting for everyone to finish. There are four parameters that are used to optimize the communication and load balancing of parallel runs for ProtoMol: masterslave, usebarrier, maxpackages, and parallelpipe.

The masterslave and usebarrier parameters are basically flags. If masterslave is set to true, then the parallel execution is set up in a master/slave format. This means that one processor acts as the master of the rest of the processors. It handles all of the communication so that the loads can be balanced more evenly. This means, however, that the

master processor does no actual data calculation. When N processors are called for under master/slave, then N+1 processors are actually used. If masterslave is set to false, then the parallel execution is set up in a static format. The static format uses all of the processors at the same time. Communication and organization of data between processors is handled individually by each of the processors. Static uses exactly N processors, but will usually not have the same load balancing that a master/slave configuration will have. If barrier is true, then a processor will wait for full synchronization before communicating globally. If not, the processor will communicate its results before full synchronization. The parallelpipe parameter defines the depth of the pipe used to hold work for each slave. This will allow the individual processors to buffer data if it is lagging behind other processors. The maxpackages parameter has to do with the amount of data passed between processors in a single communication.

## II. TESTS AND RESULTS

We did various tests isolating each of these parameters for the purpose of getting a general idea how they influence parallel runtime. In considering the output from ProtoMol, we looked at three quantities: overall runtime, time spent computing force calculation (for each processor), and time spent communicating (for each processor). Based on this data we can tell what configuration gives the best performance, as well as what factors contributed to that runtime, whether it came from good load balance or quick communication.

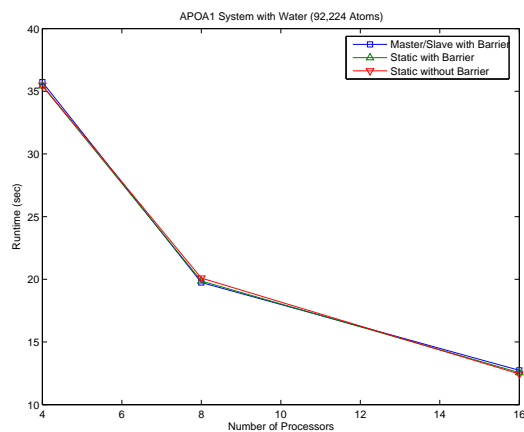
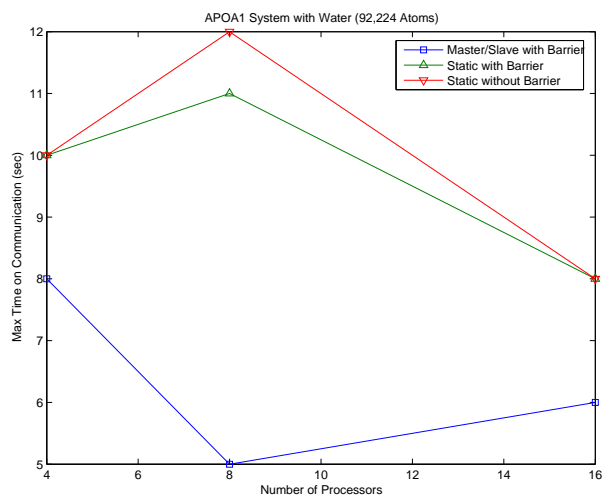


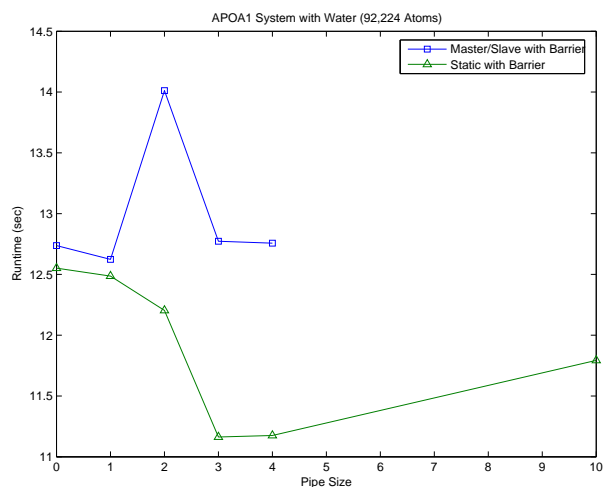
Fig. 3. The effects of master/slave vs. static and barrier vs. no barrier on runtime.

## III. DISCUSSION

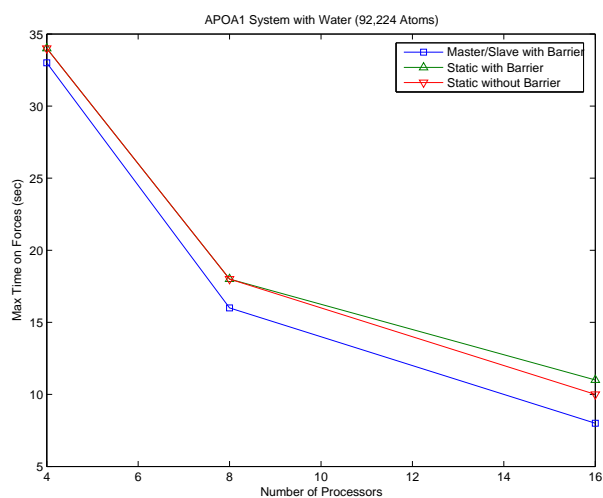
The data collected so far comes from only one molecular system. Clearly more data needs to be analyzed to get a better overview of how these parameters influence performance. There are some promising results, especially in the maxpackages and parallelpipe parameters. These two could be used in a script similar to the one developed for the MG and PME algorithm optimization scripts. Once issues



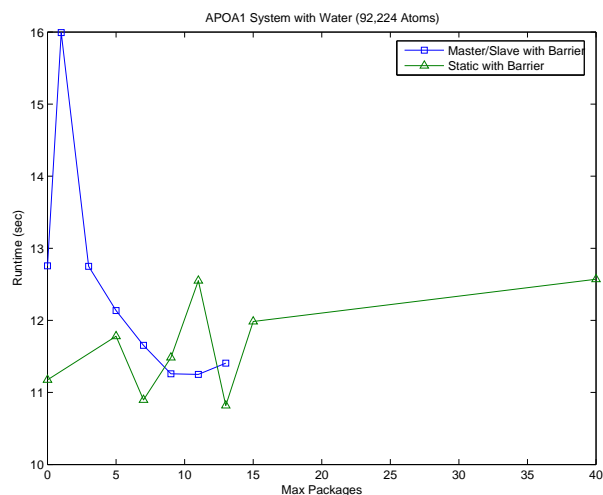
(a)



(a)



(b)



(b)

Fig. 4. The effects of master/slave vs. static and barrier vs. no barrier on: (a) communication, (b) force calculation.

dealing with the parallel cluster are resolved (we do not want to crash iss anymore), an automated script should follow quickly. In addition, many other tests are underway to expand the knowledgebase that MDSimAid has to work with.

#### ACKNOWLEDGMENT

The authors would like to thank Scott Hampton for performing various tests and Dr. Jesus Izaguirre for help with the project and the report.

#### REFERENCES

[1] M. Crocker, S. S. Hampton, and J. A. Izaguirre, "MDSimAid: A recommender system for automatic parameter selection in fast electrostatic algorithms," *J. Comp. Chem.*, 2004, submitted. Preprint available at <http://www.nd.edu/~shampton/CrHI0x.pdf>.

Fig. 5. How values for (a) pipe depth and (b) max packages influence runtime.

[2] J. A. Izaguirre and T. Matthey, "Parallel multigrid summation for the N-body problem," *J. Paral. Distrib. Comp.*, 2004, submitted.

[3] T. Darden, D. York, and L. Pedersen, "Particle mesh ewald: An  $N \log(N)$  method for Ewald sums in large systems," *J. Chem. Phys.*, vol. 98, no. 12, pp. 10 089–10 092, 1993.

[4] L. Kalé, R. Skeel, M. Bhandarkar, R. Brunner, A. Gursoy, N. Krawetz, J. Phillips, A. Shinozaki, K. Varadarajan, and K. Schulten, "NAMD2: Greater scalability for parallel molecular dynamics," *J. Comput. Phys.*, vol. 151, pp. 283–312, 1999.

[5] PROTOMOL, "PROTOMOL: An object oriented framework for molecular dynamics," <http://sourceforge.net/projects/protomol/>, Aug. 2004, 554 downloads from Sept. 2003 - Aug. 2004.