Simulating the Trajectories of One-Dimensional Pseudoparticles with Parallel Agents

Jaderick P. Pabico
Institute of Computer Science
College of Arts and Sciences
University of the Philippines Los Baños
College 4031, Laguna
(63)49 536-2313/(63)49 536-2302
jppabico@uplb.edu.ph

ABSTRACT
This paper presents a mechanism for multiple agents that dynamically load-balance the parallel computation of quantum trajectories of an one-dimensional free particle using the quantum trajectory method. The agents are able to improve the simulation time by dynamically load-balancing the parallel computation of the trajectories. The load imbalance introduced by the parallel runtime system and by the computationally-intensive trajectories of the pseudoparticles is reduced by the agents at higher number of processors. Results of experiments on a multi-purpose message-passing cluster are presented to confirm that the agents achieve better simulation performance than with the current load balancing techniques at higher number of processors.

Keywords
wavepacket simulation, parallel agents, mechanism design

1. INTRODUCTION
Various observable and measurable physical phenomena in real-world can be described by the time-dependent Schrödinger equation (TDSE). TDSE simulates the dynamics of a particle (usually an electron) represented by the trajectories of pseudoparticles that are riding on a quantum wavepacket. The usual simulation for TDSE includes the space-time grid methods, the basis sets methods, and their various combinations. One method that simulates the trajectories of wavepackets is the quantum trajectory method (QTM) that is based on the hydrodynamic interpretation of quantum mechanics [3]. QTM is a space-time unstructured grid method that solves the quantum equations using a moving weighted least-squares (MWLS) to compute the underlying differential equations. The solutions to the equations of motion give the quantum trajectories for pseudoparticles. In 1999 [14], QTM was implemented for a serial computer. Two years later [4], taking advantage of MWLS' high degree of concurrency brought about by the presence of parallel computation of the trajectories, a version for a shared-memory parallel machine was also implemented. A message-passing parallel implementation with an *adaptive* MWLS was also reported [6]. In terms of simulation time, the parallel machine implementations have obvious simulation performance improvement over the serial computer implementation. In terms of ease of utilizing more processors as the simulation size is increased, the message-passing implementation is more flexible to use than the shared-memory one. In the message-passing implementation, the parallel execution time can be further improved by balancing the loads of participating processors.

In recent years, research advances in dynamic load balancing at the application level have contributed to improving the performance of scientific simulations running under a parallel system with heterogeneous processors. Dynamic load balancing schemes based on scheduling of atomic computations such as *Factoring* (FACT), *Fractiling* (FRAC), *Weighted Factoring* (WFAC), *Adaptive Weighted Factoring* (AWF), and *Adaptive Factoring* (AFAC) have been successfully implemented in the simulation of wavepacket dynamics using the QTM [5, 6]. These schemes are based on probabilistic analyses that take into consideration other variabilities that can make the problem complicated when solved in real time. The variabilities are brought about by systemic factors such as the variance in processor performance during execution and the variance in network latency. The performance of the processors might vary during computation, even when the processing speed is known. For example, the performance of a processor might be affected when a server daemon is woken up or when the processor is interrupted by the hardware.

The above mentioned load balancing schemes greatly improve the performance of QTM but they do not come without inherent problems. This is because they are implemented using a *master-slave* strategy where one processor acts as a master and balancer. As a balancer, the master processor dynamically assigns pseudoparticles to slave processors following a scheduling policy defined by the scheme. For example, in FRAC, the scheduling policy is based on the characteristics of fractals. After being assigned with a *bundle* of pseudoparticles, the performance of the slaves

6th National Conference on Information Technology Education
University of the Philippines Los Baños, 23-24 October 2008

is evaluated by the master. A bundle is some number of pseudoparticles ρ such that $\rho << \lceil n/p \rceil$, where n is the total number of pseudoparticles and p is the number of participating processors in the simulation. The future assignment of new bundles for the slave processor is computed based on its performance of the previous bundle assignment. The performance measure is simply the time it took for the slave to process a given bundle. When a slave is done with its assigned bundle, the slave communicates with the master its processing time for the bundle. The master then communicates to the slave the next bundle of particles that the slave needs to process. The size of the next bundle is defined by the scheduling policy being implemented. If all ρ pseudoparticles have already been assigned and one slave processor finishes early than the other, the master asks the slowest processor to give up *some* of its pseudoparticles to the now idle slave processor. The number of pseudoparticles that is to be given up is again based on the scheduling policy defined by the scheme.

The inherent problem with the master-slave strategy is that when more processors are required by the simulation, the number of slaves that a master has to communicate with increases. The benefits of the dynamic scheduling schemes are greatly offset by the latency due to communication bottleneck with the master, compounded with the type of physical network interconnect the processors are using. When more slaves communicate to the master, the master is overwhelmed with communication requests and might not be able to simulate the trajectories of the psedoparticles assigned to itself. The assigned bundle of pseudoparticles to the master will eventually be reassigned to other slaves requiring more communication. This scenario leaves one processor doing only balancing rather than doing essential computation (i.e., computation that would be performed by the sequential machine for solving the same problem instance). The communication latency greatly reduces the performance of the simulation when p is increased.

The problem with master-slave strategy is that only one processor balances for the rest. The slave processors are dependent on the master processor, are not responsible for balancing, are only tasked to process the assigned particles, and communicate to the master their performance and status. The master on the other hand is tasked to process its assigned[1] bundle of pseudoparticles while it keeps record of the slaves' status. The master also runs the load balancing policy and communicates to $p-1$ slaves[2]. The master-slave strategy can be seen as a multi-agent system where the master is an independent agent while the slaves are dependent yet specialized agents[3]. If all processors will instead act as independent yet truthful agents that follow the same scheduling policy, the communication latency brought about by communicating to only one master will be evenly distributed to the other $p-1$ processors, bringing down the time latency due to communication. Thus, a multi-agent approach to load balancing might be a better strategy than the master-slave one.

[1]The master also does some computation and is considered as its own slave.

[2]The master does not incur communication latency when it communicates to itself.

[3]The slave's specialization is to only compute the trajectories of the pseudoparticles.

In this effort, the scheduling of pseudoparticles in QTM is mapped into a multi-agent strategy. Here, processors are considered autonomous agents with a self-interested goal of finishing its assigned bundle of pseudoparticles at the earliest time. However, the agents are members of a community with a social goal of finishing the simulation at the shortest possible time. The agents are independent in the sense that it can decide for itself. However, the agents are still dependent on other agents such that the social goal of minimizing simulation time is obtained. This dependency requires other agents to communicate with other agents. However, since agents are now independent, they are required to truthfully communicate their status and performance such that the other agents will have a global perspective of the community. This global perspective will help the agents come up with an optimal assignment for themselves such that both their personal goals and the community goals are achieved.

This paper describes a parallel QTM code for a message-passing environment with a multi-agent-based pseudoparticle assignment integrated into the code. To improve the simulation performance, the code incorporates the recommendations of past researchers [4, 5, 6]. The background of QTM and the lessons learned with the parallel implementation of the QTM by other researchers are also discussed. The integration to QTM of parallel agents as a strategy for load balancing is presented. Timing results on a Gnu/Linux cluster which confirm the performance improvements of the QTM from the multi-agent-based load balancing are likewise presented. The summary of the paper and hints on ongoing work are given.

2. QUANTUM TRAJECTORY METHOD

The study of many problems in quantum mechanics is based on finding the solution to the time-dependent Schrödinger equation (TDSE) shown in Equation 1. TDSE describes the dynamics of quantum-mechanical systems composed of a particle of mass m moving in a potential V ($\hbar$ is the reduced Planck's constant, Ψ is the wave function, and $\mathcal{H}$ is the Hamiltonian).

$$i\hbar\frac{\partial\Psi}{\partial t} = \mathcal{H}\Psi, \quad \mathcal{H} = -\frac{\hbar^2}{2m}\nabla^2 + V \tag{1}$$

The hydrodynamic formulation of the quantum mechanics is obtained by substituting the polar form of the system wave-function (Equation 2) into the TDSE [14]. Here, $R(r,t)$ and $S(r,t)$ are the real-valued amplitude and phase functions of space r and time t.

$$\Psi(r,t) = R(r,t)\exp(iS(r,t)/\hbar) \tag{2}$$

Following the work of Lopreore and Wyatt [14], Equation 2 is separated into its real (Equation 3) and imaginary (Equation 4) parts:

$$-\frac{\partial \rho(r,t)}{\partial t} = \nabla\left[\rho(r,t)\frac{1}{m}\nabla S(r,t)\right], \quad (3)$$

$$-\frac{\partial \rho(r,t)}{\partial t} = \frac{[\nabla S(r,t)]^2}{2m} + v(r,t) + Q(\rho;r,t), \quad (4)$$

where $\rho(r,t) = R^2(r,t)$ is the probability density, $v(r,t) = m^{-1}\nabla S(r,t)$ is the velocity, and

$$Q(\rho;r,t) = -\frac{1}{2m}\left(\nabla^2 \log \rho^{1/2} + |\nabla \log \rho^{1/2}|^2\right)$$

is the global quantum potential in logarithmic form. Taking the gradient of Equation 4 and using the Lagrangian derivative $\frac{d}{dt} = \frac{\partial}{\partial t} + v \cdot \nabla$ leads to the equation of motion

$$m\frac{dv}{dt} = -\nabla(V+Q) = f_c + f_q, \quad (5)$$

where f_c and f_q are the classical and quantum force terms, respectively. Rewriting Equation 3 as

$$\left(\frac{\partial}{\partial t} + v \cdot \nabla\right)\rho(r,t) = \frac{d\rho(r,t)}{dt} = \rho(r,t)\nabla \cdot v,$$

and integrating yields the density update equation

$$\rho(r,t+dt) = \rho(r,t)\exp(-dt\nabla \cdot v). \quad (6)$$

A set of n pseudoparticles, each of mass m, is deployed to represent the physical particle. Each pseudoparticle executes a "quantum trajectory" governed by the Lagrangian equations of motion (Equation 5 and Equation 6) and the quantum potential Q. Derivatives of ρ, Q, and v for updating the equations of motion are obtained by curve-fitting the numerical values of these variables using the MWLS method and then differentiating the least squares curves.

The QTM implementation in a serial computer was applied to barrier tunneling [14]. The shared-memory parallel QTM [4] was ported into a message-passing version and restructured to accommodate the implementation of an explicit update scheme [6] and the integration of load balancing. An outline of the code is given in Figure 1.

The MWLS method needed to compute $Q[]$, $f_q[]$, and $dv[]$ solves an overdetermined linear systems of size $n_p \times n_b$. The numerical experiment conducted by [4] suggested $n_b = 6$ and $n_p = n_[p1] = n$ when computing $Q[i]$, and $n_p = n_{p2} = n/2$ when computing $f_q[]$ or $dv[]$. Loops 1 through 3 are computationally bound, consuming the bulk of the CPU time. Loops 2 through 4 may be combined into a single loop but were separated to allow the integration of load balancing.

```
Initialize positions r[], velocities v[],
        and probability densities p[]
for time_step=0 to T-1 do
begin
    for pseudoparticle i = 1 to n (Loop 1)
    begin
       MWLS(i, r[], p[], np1, nb);
       compute quantum potential Q[i];
    end
    for pseudoparticle i = 1 to n (Loop 2)
    begin
       MWLS(i, r[], p[], np2, nb);
       compute quantum force f_q[i];
    end
    for pseudoparticle i = 1 to n (Loop 3)
    begin
       MWLS(i, r[], p[], np2, nb);
       compute derivative of velocity dv[i];
    end
    for pseudoparticle i = 1 to n (Loop 4)
    begin
       compute classical potential V[i]
               and classical force f_c[i];
    end
    Output t, r[], v[], p[], V[], f_c[],
           Q[], f_q[], dv[];
    for pseudoparticle i = 1 to n (Loop 5)
    begin
       update p[i], r[i], v[i];
    end
end
```

Figure 1: The code for simulating the trajectories using the QTM.

3. DYNAMIC LOAD BALANCING ALGORITHMS

In systems that use heterogeneous processors, the difference in processor speeds and memory capacities coupled with the dynamism in processor interconnection can significantly impact the performance of scientific applications. For example, load and resource availability of processors in a grid is unpredictable, and thus it is difficult to know in advance what the effective speed of each machine would be.

For effectively load balancing scientific applications, algorithms that derive from theoretical advances in research on scheduling parallel computations with variable processing times have extensively been studied [13, 16, 17, 9, 15]. As a result, dynamic scheduling based on a probabilistic analysis have been proposed, and successfully implemented in a cluster environment for a number of scientific applications [9, 1, 8, 2, 12, 11, 5, 6].

An example of a dynamic scheduling scheme is the one that uses a scheduling policy based on FACT [9]. In this scheme, a probabilistic analysis allows the formulation of *factoring rules*. Independent computations are assigned to processors in bundles of variable sizes following the factoring rules. The selection of bundle sizes requires that they have a high probability of being completed by the processors before the optimal time. The bundle sizes are dynamically computed during the execution of the application. The larger bundles have relatively little overhead, and their unevenness can be smoothed over by the smaller bundles. Another scheduling policy that improves FACT is FRAC. It combines the scheduling technique that balances processor loads with data locality by exploiting the self-similarity properties of fractals. This method has successfully been implemented for many-body simulations on distributed shared address space and message-passing environments [1, 2].

Other dynamic scheduling schemes that have been successfully implemented and found to improve the performance of FACT are WFAC, AWF, and AFAC. AFAC scheme is a more general model for scheduling. In specific conditions of processor speed and computation workloads, AFAC converts into the other schemes. These schemes are described in details elsewhere [8, 11, 12] and will not be discussed in this paper. Suffice it to say that all these schemes are implemented in a cluster environment using a master-slave strategy which has inherent communication bottleneck for higher p.

4. MECHANISM DESIGN

The master-slave strategy can be regarded as a special case of a multi-agent system. The master is an agent that has total *control* of the $p-1$ slaves. Control here means that the master has all the decision making responsibilities for the community composed of itself and the slaves. Since the master itself is the balancer, it needs to have an updated global information of the status of the system. This means that all slaves will have to periodically[4] communicate to the master. The overhead due to communication increases with the number of slaves present in the community. Thus, at higher values of p, the scheduling schemes with master-slave strategy tend to reduce the performance of the parallel application.

In the multi-agent approach, the processors are considered as autonomous agents that belong to a community. The community is tasked to process the trajectories of the pseudoparticles. Initially, the agents do not know the respective computing speed of the other agents. However, the number of agents that belong to the community and the respective address of each agent are common knowledge. The agents may use the same addressing scheme as the processors' (i.e., $P = \{p_0, p_1, \ldots, p_{m-1}\}$). This addressing scheme is arbitrary and does not favor any agent. What is important here is that every agent knows its address in the community as well as that of the other agents.

4.1 Initial Particle Assignment

The initial job assignment is guided by an assumption that the processing times of all pseudoparticles $\{\rho_0, \rho_1, \ldots, \rho_{n-1}\}$ are the same while processors are assumed to be homogeneous. These assumptions are not common knowledge for the agents but just a simplistic way of initializing the system with a bundle of pseudoparticles to start with. The n pseudoparticles are equally divided to p agents and each agent receives $\lceil n/p \rceil$ pseudoparticles. A trivial assignment would be such that agent p_0 is assigned to pseudoparticles ρ_0 to $\rho_{\lceil n/p \rceil -1}$, agent p_1 to pseudoparticles $\rho_{\lceil N/p \rceil}$ to $\rho_{2\lceil n/p \rceil -1}$, and so on. If the pseudoparticles will originate from a single processor, then a *scatter* or *one-to-all personalized communication* [10] may be utilized. After initialization, the agents will perform a *task sharing* or *task passing* strategy [7] based on a prediction to trajectory computation times.

4.2 Profiling of Particle Computation Time

Each agent will compute the pseudoparticles assigned to it using some statistical sampling order and size. The goal of sampling is for the agent to predict the profile of the processing time of the $\lceil n/p \rceil$ pseudoparticles assigned to it using some sample size $s << \lceil n/p \rceil$. For example, agent p_0 will process pseudoparticles $\{\rho_i | i = 0, \lceil n/p \rceil / s, 2\lceil n/p \rceil / s, \ldots\}$. Based on p_0's processing times on the sampled pseudoparticles, the processing times for all the remaining pseudoparticles assigned to it can be predicted via a curve fitting or interpolation technique (e.g., nonlinear regression, divided difference, etc.). After some time, all agents will have a local knowledge of the predicted processing times of the remaining pseudoparticles assigned to them. If all agents will truthfully exchange this information to all other agents (i.e., via an *all-to-all broadcast* [10]), then each of them will have a global knowledge of the predicted processing times of the remaining pseudoparticles assigned to the community. With this information, the modified bin-packing algorithm may be employed by each agent to reallocate pseudoparticles to processors taking in consideration the pseudoparticles that already have been sampled. This information sharing is what is termed as *result sharing* by Durfee [7].

[4]The period here depends on the scheduling policy being used.

4.3 Normalization with Heterogeneous Agents

In the case of parallel systems with heterogeneous processors, each agent will need to perform a normalization procedure to account for the variability in processor speeds. The normalization involves sampling a pseudoparticle assigned to other agents and which already have been sampled by that agent. For example, if agent p_0 found pseudoparticle ρ_0 to have a processing time of $t_{0,0}$, and if agent p_1 processed the same pseudoparticle ρ_0 and found it to have a processing time of $t_{0,1}$, then the ratio between $t_{0,0}$ and $t_{0,1}$ provides the relative speed of p_1 to p_0, and *vice versa*. If all agents will perform this normalization procedure and will truthfully communicate the information from this procedure to other agents, then all agents will have the global information of their relative speeds compared to the others. An all-to-all broadcast may again be utilized to efficiently exchange the information. Based on this information, the modified bin-packing algorithm for uneven bins may be utilized by each agent to come up with a global reallocation of pseudoparticles. Since all agents will have the same solution, when agent p_i asks for a pseudoparticle from agent p_j, p_j already knows what pseudoparticle to give and must freely give it. The agent behavior is guided by a payoff scheme that is a function of the performance of the community when solving the assigned pseudoparticles.

4.4 Finite Horizon Game

The QTM code fragment presented in Figure 1 presents independent loops embedded in a time-stepping loop, which may be regarded by the p agents as a finite-horizon game with T horizons. Each loop is a computation of the trajectory of the pseudoparticle at a given time-step.

Each game, the agents will cooperatively process n pseudoparticles via normalization, task sharing and result sharing such that the community goal is achieved. The game is formalized by a mechanism design such that the Nash equilibrium is a strategy where agents must truthfully share information and cooperate with other agents.

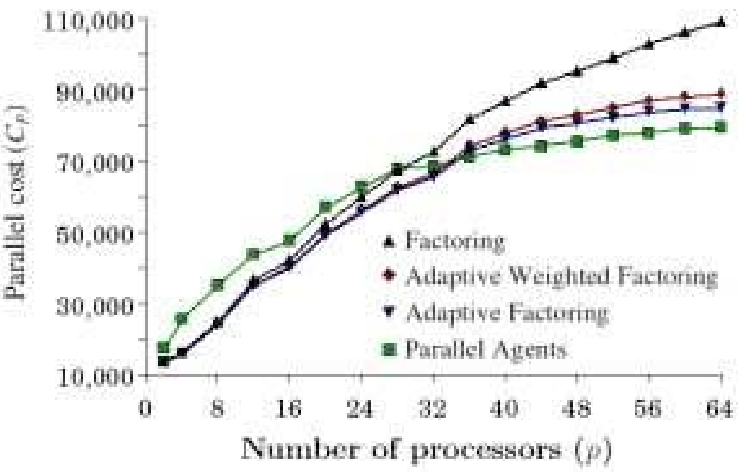

Figure 2: Average parallel cost for wavepacket simulation with 501 pseudoparticles using FAC, AWF, AFAC and multi-agent-based scheduling at increasing number of processors.

5. PERFORMANCE EVALUATION AND RESULTS

To assess the performance of the parallel agents for load balancing, the same QTM code presented in Figure 1 was also integrated with the three scheduling schemes: FACT, AWF and AFAC. All runs were conducted on an adhoc general-purpose cluster at the Institute of Computer Science, University of the Philippines Los Baños. The adhoc cluster is a 256-node, 256-processor cluster with a combined memory of 128 GB. The processors are of Pentium III, Pentium IV and AMD Athlon architectures with processing speeds that range from 800MHz to 2.4GHz. They are interconnected through 10 Mbps Ethernet switches that are interconnected through a 100 Mbps Ethernet uplink. Since this cluster is a general-purpose, adhoc cluster, the network traffic volume is not predictable (i.e., other processes maybe running along with the QTM simulations).

A wavepacket of 501 pseudoparticles was simulated for 10,000 time steps. The performance metrics measured was the parallel cost $C_P = p \times T_p$. Each simulation run was replicated five times to model the variability in C_P induced by the underlying computing architecture.

The average C_ps for running the QTM using FACT, AWF, AFAC and the multi-agent-based scheduling are graphed in Figure 2. The graph indicates that when simulating the trajectory of an one-dimensional particle represented by 501 pseudoparticles:

1. The agent-based scheme is, on the average, less costly at lower p. The graph shows that:
   (a) FACT is actually cost-effective at $p \leq 28$; and
   (b) Both AWF and AFAC are less costly at $p \leq 32$.
2. The agent-based load balancing has lower average C_p at higher p. Graph shows that:
   (a) At $p > 28$, the average C_p for the agent-based scheme is lower than that of FACT.
   (b) At $p > 32$, the average parallel cost for multi-agent-based scheduling is lower than that of AWF and AFAC.

6. SUMMARY

This paper presents an improve performance of the simulation of one-dimensional pseudoparticle trajectories using an agent-based load balancing at higher p. The purpose of load balancing is to improve the simulation performance of the simulation. The static and offline solution to load balancing can be optimally obtained using the bin-packing algorithm for tractable values of n and p. Due to variability in the parallel system, and because the computation times of the trajectories are not known *a priori*, the load balancing problem can only be solved online. Most known dynamic load balancing schemes use a master-slave strategy that is inherently susceptible to communication bottleneck when p is increased. The multi-agent approach regards processors as agents. The approach reduces the bottleneck by distributing the communication to other agents. All agents in the community must truthfully share information via normalization, task sharing, and result sharing. The information exchange may be efficiently performed using the one-to-all personalized communication and the all-to-all broadcast. The behavior of the agents is defined by a payoff function that forces the agents to cooperate. QTM, as a time-stepping method, is regarded by agents as a finite-horizon game. The game formalization has Nash Equilibrium for cooperation and truthful information sharing. The performance of the agent-based QTM is compared to the performance of the QTM with FACT, AWF, and AFAC schemes. When QTM is simulating 501 pseudoparticles, the agent-based QTM is less costly, on the average, at $p > 28$ compared to FACT. When compared to both AWF and AFAC, the agent-based methodology is more cost effective on the average at $p > 32$. Thus, at higher p, the agent-based QTM outperformed the QTM with either FACT, AWF, or AFAC scheme.

As an extension of this work, effort is already underway in applying the parallel agents to the two- and threedimensional versions of the wavepacket simulation. These versions involve significantly more computations than the one treated in this paper. Results of numerical experiments, as well as significant lessons learned, will be reported in the future.

7. ACKNOWLEDGMENTS

This work is funded by the Institute of Computer Science, University of the Philippines Los Baños under Code 2326103.

8. REFERENCES

[1] I. Banicescu and S. Flynn Hummel. Balancing processor loads and exploiting data locality in N-body simulations. In *Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM)*. ACM Press, 1995.

[2] I. Banicescu and R. Lu. Experiences with fractiling in N-body simulations. In *Proceedings of High Performance Computing 98 Symposium*, pages 121–126, 1998.

[3] D. Bohm. A suggested interpretation of the quamtum theory in terms of hidden variables. *Physical Review*, 85:166–193, 1952.

[4] R.G. Brook, P.E. Oppenheimer, C.A. Weatherford, I. Banicescu, and J. Zhu. Solving the hydrodynamic formulation of quantum mechanics: A parallel MLS method. *International Journal of Quantum Chemistry*, 85:263–271, 2001.

[5] R. Cariño and I. Banicescu. A load balancing tool for distributed parallel loops. In *Proceedings of the International Workshop on Challenges of Large Applications in Distributed Environments*, pages 39–46. IEEE Computer Society Press, 2003.

[6] R. Cariño, I. Banicescu, R. Vadapalli, C. Weatherford, and J. Zhu. *Message Passing Parallel Adaptive Quantum Trajectory Method*, pages 127–139. Kluwer Academic publishers, 2004.

[7] E. Durfee. *Distributed Problem Solving and Planning*, pages 118–149. Srpringer-Verlag, 2001. 9th ECCAI Advanced Course, ACAI 2001 and Agent Link's 3rd European Agent Systems Summer School, EASSS 2001 Prague, Czech Republic, July 2001, Selected Tutorial Papers.

[8] S. Flynn Hummel, J. Schmidt, R. Uma, and J. Wein. Load-sharing in heterogeneous systems via weighted factoring. In *Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 318–328, 1996.

[9] S. Flynn Hummel, E. Schonberg, and L. Flynn. Factoring: A method for scheduling parallel loops. *Communications of the ACM*, 35(8):90–101, August 1992.

[10] A. Grama, A. Gupta, G. Karypis, and V. Kumar. *Introduction to Parallel Computing*. Addison-Wesley, New York, 2nd edition, 2003.

[11] I., V., and J. Devaprasad. On the scalability of dynamic scheduling scientific applications with adaptive weighted factoring. *Cluster Computing*, 6(3):215–226, 2003.

[12] I. and V. Velusamy. Load balancing highly irregular computations with the adaptive factoring. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS 2002) - Heterogeneous Computing Workshop*. IEEE Computer Society Press, 2002.

[13] C. Kruskal and A. Weiss. Allocating independent subtasks on parallel processors. *IEEE Transactions on Software Engineering*, 11(10):1001–1016, October 1985.

[14] C. Lopreore and R. Wyatt. Quantum wavepacket dynamics with trajectories. *Physical Review Letters*, 82(26):5190–5193, 1999.

[15] EP. Markatos and T. LeBlanc. Using processor affinity in loop scheduling on shared-memory multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 5(4):379–400, April 1994.

[16] C. Polychronopoulos and D. Kuck. Guided self-scheduling: A practical scheduling scheme for parallel supercomputers. *IEEE Transactions on Computers*, 36(12):1425–1439, December 1987.

[17] T. Tzen and L. Ni. Trapezoid self-scheduling: A practical scheduling scheme for parallel computers. *IEEE Transactions on Parallel Distributed Systems*, 4(1):87–98, January 1993.