Discrete Ordinates Solution of the Time Dependent Radiation Transport Equation on a Distributed Network of UNIX Workstations

Timothy J. Tautges and Osman Yasar

January 1991

UWFDM-846

FUSION TECHNOLOGY INSTITUTE

UNIVERSITY OF WISCONSIN

MADISON WISCONSIN

Discrete Ordinates Solution of the Time Dependent Radiation Transport Equation on a Distributed Network of UNIX Workstations

Timothy J. Tautges and Osman Yasar

Fusion Technology Institute
University of Wisconsin
1500 Engineering Drive
Madison, WI 53706

http://fti.neep.wisc.edu

January 1991

UWFDM-846

Abstract

The radiation transport equation is solved using the S_n method on an adaptive mesh. The solution is performed in parallel on a network of workstations, communicating with each other using message passing. Analytical speedup estimates are formulated for the calculations and compared with measured speedups. Speedups of 1.83 and 2.97 are achieved on two and four processors, respectively. Finally, suggestions are given for improving the parallel performance.

Acknowledgements

This work was made possible by the switchboard software provided by Professor Barton Miller of the UW-Madison Computer Science Department. Thanks also go to staff scientist Joe MacFarlane and graduate student Kim Simmons for their enlightening discussions on the subject of radiation hydrodynamics.

This research was made possible in part by an appointment to the Nuclear Engineering and Health Physics Fellowship Program administered by Oak Ridge Associated Universities for the U. S. Department of Energy. The United States Nuclear Regulatory Commission provided support for the purchase of the UNIX workstations used in this research.

Contents

1. Introduction

Radiation transport calculations are used to model the transport of photons in an absorbing medium. This physical process is important in scientific applications, such as the modeling of stellar atmospheres and the simulation of inertial confinement nuclear fusion devices. This study was motivated by the need to simulate the radiation transport occuring in ion beam driven inertial confinement fusion devices. These devices achieve the high temperature and confinement times needed for nuclear fusion by bombarding fuel pellets with large amounts of energy in the form of energetic ion beams.

Delivering a focused energetic beam of light ions to a small fuel target is very difficult in practice. Light ion beams, with their high charge to mass ratio and high current densities, cannot propagate large distances in a vacuum without significant beam divergence. One solution to this problem is to charge and current neutralize the beams so that their mutual electrostatic repulsion and tendency to pinch is removed. This involves filling the channel region (between the cathode, where the beam is produced, and the target) with an ionized gas. One method of doing this, proposed by Yonas [9], is to fill the channel region with a gas and strike a discharge along the beam path just before the beam travels through the channel. The discharge produces an ionized gas plasma inside the channel as well as a magnetic field which confines the beam. The ionized gas plasma charge neutralizes the beam; it also current neutralizes the beam by reacting with a return current in a direction opposite to the beam direction. Figure 1 shows a diagram of a typical light ion beam channel. For a more detailed discussion of light ion beam plasma channels, see [8].

There are many competing physical processes which affect the ability of the plasma channel to confine the ion beam. One very important effect is the initial state of the channel when the ion beam is injected. A short time before beam injection into the plasma channel, the plasma is created by discharging a laser beam along the ion beam path. This

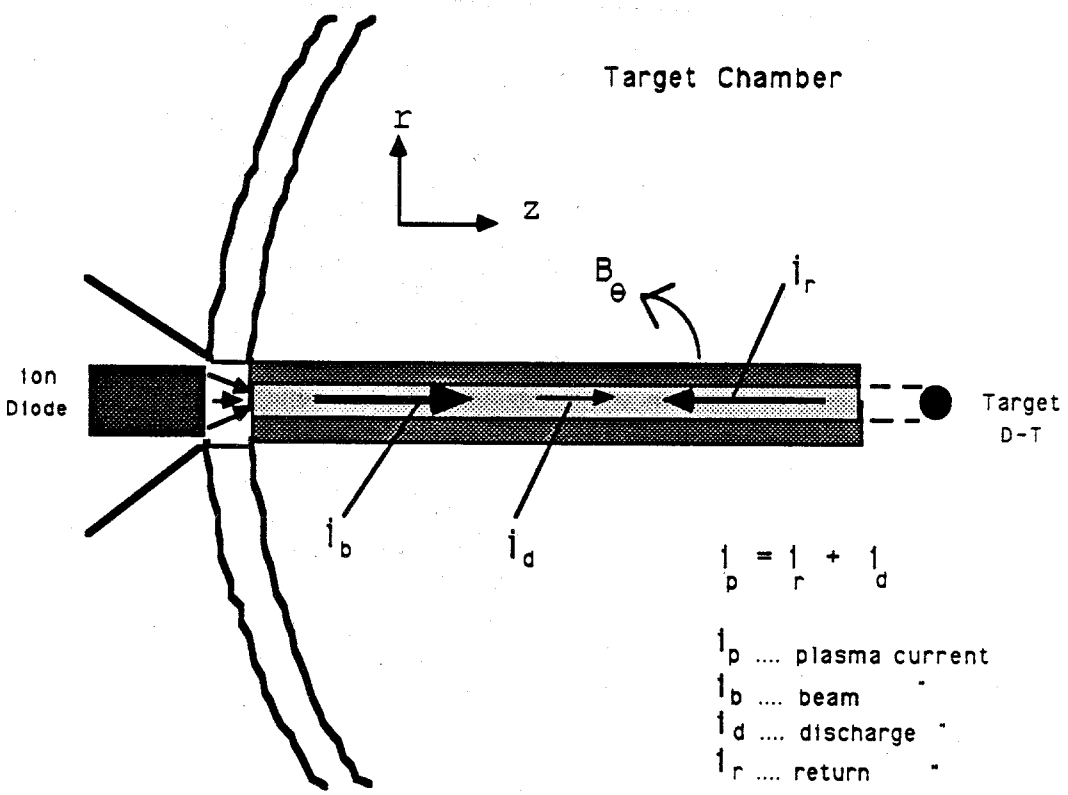

Figure 1. Diagram of beam propagation in a typical light ion beam plasma channel.

produces a relatively hot ionized plasma, which is surrounded by a relatively cool neutral gas. This region of ionized gas surrounding the beam is vital to the beam's stability, since it produces the charge and current neutralization. The behavior of this region is dominated by the radiation of energy to the relatively cold gas on the periphery. It is the modeling of this radiation transport that dominates the execution time of ion beam plasma channel simulations.

Simulation of plasma channels and the propagation of ion beams through them are executed on supercomputer architectures, and take full advantage of vector hardware on these machines. These simulations are extremely expensive in terms of execution time.[8] Yet, they are still limited by the accuracy one can obtain while maintaining reasonable execution and response times[1]. Faster simulations are required for improved accuracy, particularly in the radiation transport modeling. Since present solution techniques are already fully vectorized, the only remaining answer to this problem is parallel processing.

The easiest way to speed up present codes would be to run them in parallel on the CRAY Y/MP, since they are already vectorized for these machines. However, the parallel speedup is limited to 8, the maximum number of processors on these machines. Also, even though the response time would be much lower for a parallel code, the user would likely be charged for the cumulative execution time on all the processors, plus some added overhead cost for parallel processing. This leads us to propose another method of performing parallel radiation transport simulations–one that takes advantage of fixed-cost computers and has a much greater maximum potential speedup.

[1]Both execution time and response time are important. Execution time depends on the program being run and can use up a time allocation too quickly. On the other hand, response time depends on the user load of the computer being used, and can be much larger than the execution time; this makes large simulations (long problem duration or high accuracy) impractical.

This method uses RISC workstations, each capable of around 1 MFLOP execution speed, connected over a high speed local area network. These workstations have become quite common in scientific research establishments, and are powerful tools for code development, data visualization and of course serve as terminals that network to more powerful computers. Yet, these workstations are often underutilized during the daytime, and are seldom utilized at night. In addition, they are a source of fixed-cost computing, where users are charged only for the purchase of the machine and not for actual execution time. Finally, the UNIX operating system that runs on virtually all of the current RISC workstations already has networking software in place which is both relatively easy to use and very powerful. This report will show that these workstations could be used to perform plasma channel simulations in parallel with response times that could rival supercomputer response times, for a much cheaper cost.

The remainder of this report is arranged as follows. Chapter 2 contains a brief description of the physics and derivation of the equations used to model radiation transport. It then describes the numerical methods used to solve these equations. Chapters 3 and 4 outline the technique used to solve these equations in parallel, on a network of distributed workstations, and the parallel software tools used for communication between the workstations. A prediction of the parallel performance is derived in Chapter 5, and compared to actual results in Chapter 6. Chapter 7 contains conclusions and future directions for this research project.

2. Radiation Hydrodynamics Equations

The governing equations for a nonrelativistic fluid in the frame of radiation hydrodynamics (RHD) are described as [8]

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0$$

$$\frac{\partial}{\partial t}(\rho \mathbf{u} + \frac{1}{c^2}\mathbf{F}) + \nabla p + \nabla \cdot (\rho \mathbf{u}\mathbf{u} + \vec{\mathbf{P}}) = 0$$
$$\frac{\partial}{\partial t}(e_p) + \nabla \cdot (\mathbf{q} + (e_p + p)\mathbf{u}) = \int_0^\infty d\nu \int_{4\pi} d\Omega(\chi I - \eta) \tag{1}$$

where e_p is plasma energy density, and $\mathbf{F}$ and $\vec{\mathbf{P}}$ are the radiation flux and pressure tensor. A relation is needed between the pressure, density and temperature to close the system. The pressure can be found using the equation of state relation

$$p = (1 + \bar{Z})nkT$$

where $\bar{Z}$ is the average charge state, n is number density and T is temperature.

The state of the radiation field is found through the radiative transfer equation, which is a mathematical statement of the conservation of photons and is given in the following form [8]

$$(\frac{1}{c}\frac{\partial}{\partial t} + \hat{\Omega} \cdot \nabla) I(\mathbf{r}, t, \hat{\Omega}, \nu) = \eta(\mathbf{r}, t, \hat{\Omega}, \nu) - \chi(\mathbf{r}, t, \hat{\Omega}, \nu) I(\mathbf{r}, t, \hat{\Omega}, \nu) \tag{2}$$

where I is the specific intensity, and η and χ are called emissivity and extinction coefficients, and $\hat{\Omega}$ is the solid angle.

In cylindrical coordinates the multigroup photon conservation equation in one-dimensional (radial) cylindrical coordinates becomes

$$\frac{1}{c}\frac{\partial}{\partial t} I_g(r, t, \hat{\Omega}) + \frac{\mu}{r}\frac{\partial}{\partial r}(r I_g(r, t, \hat{\Omega})) - \frac{1}{r}\frac{\partial}{\partial \omega}(\zeta I_g(r, t, \hat{\Omega})) = \bar{\eta}_g - \bar{\chi}_g I_g(r, t, \hat{\Omega}) \tag{3}$$

where the streaming term [8] in the radiative transfer equation is replaced by

$$\hat{\Omega} \cdot \nabla I_g = \frac{\mu}{r}\frac{\partial}{\partial r}(r I_g) - \frac{1}{r}\frac{\partial}{\partial \omega}(\zeta I_g). \tag{4}$$

The group constants $\bar{\eta}_g$ and $\bar{\chi}_g$ are given as

$$\bar{\eta}_g = \int_{\nu_{g-1}}^{\nu_g} d\nu \eta_\nu$$

and

$$\bar{\chi}_g = \frac{\int_{\nu_{g-1}}^{\nu_g} \chi d\nu}{\int_{\nu_{g-1}}^{\nu_g} d\nu} \tag{5}$$

assuming that one has enough groups so that χ is constant for each group. Here μ, ω and ζ are angular variables as shown in Figure 2 [8]. Cylindrical coordinates are complicated by the fact that even in one spatial dimension two angular variables, ζ and ω, are needed to describe the angular dependency of specific intensity I. As shown in Figure 2, ω is the angle between $\hat{r}$ and $\hat{\Omega}$ and μ and ζ are given as

$$\mu = (1 - \varsigma^2)^{\frac{1}{2}} \cos\omega$$

$$\zeta = (1 - \varsigma^2)^{\frac{1}{2}} \sin\omega. \tag{6}$$

The numerical accuracy of these equations can be improved by solving them on an adaptive grid. That is, the grid does not remain fixed, but moves in such a way as to provide more grid points where the solution has larger gradients. Thus improved accuracy is achieved with little penalty in execution time. The adaptive grid is varied as a function of several dependent variables. This adaption is described using two variables, the mesh metric r_ξ and the mesh velocity r_τ. A transformation is done to convert the governing equations to use these variables as the independent variables; for the radiation transport equation, the result is [8]

$$\frac{1}{c}[rr_\xi I_g]_\tau - \frac{1}{c}[rr_\tau I_g]_\xi + \mu[rI_g]_\xi - r_\xi[\zeta I_g]_\omega - rr_\xi(\bar{\eta}_g B_g - \bar{\chi}_g I_g) = 0. \tag{7}$$

The numerical solution of this equation is more complicated than that of the continuity, momentum, energy and magnetic field because of the degree of discretization it requires. Even though it describes the radiation transport in only the radial direction, by assuming symmetry in azimuthal and axial directions, it includes surprisingly two angular variables ζ and μ besides the radial component r. Therefore more care must be taken in indexing the

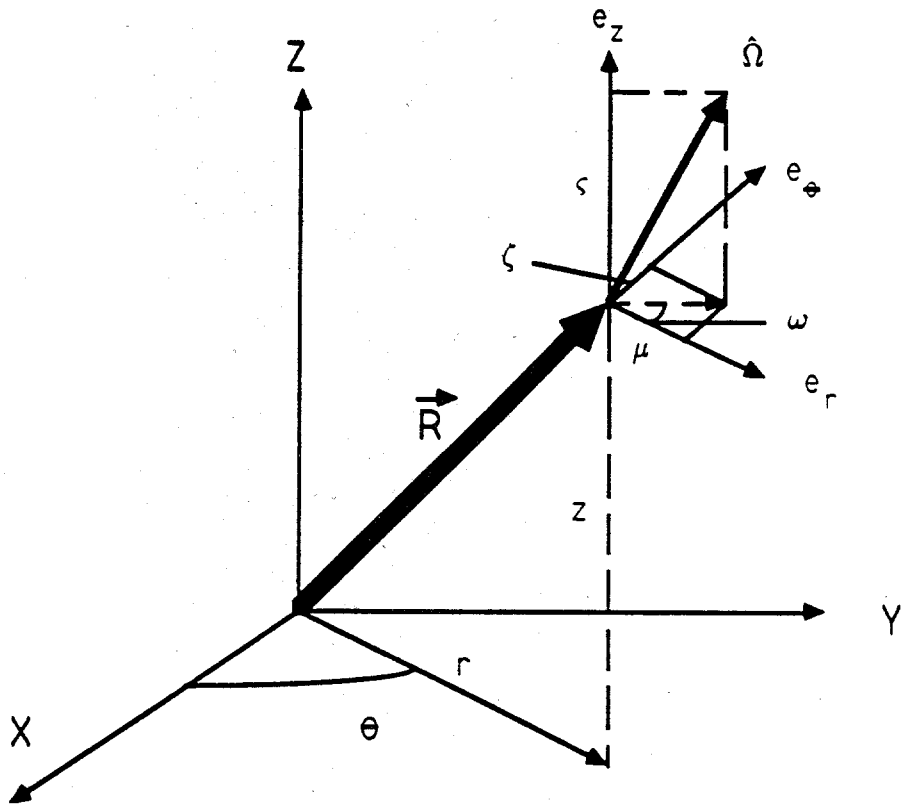

Figure 2. Cylindrical space-angle coordinate system.

discrete directions so that the angular differencing formulas can be applied in a consistent manner.

2.1. Discretization of the Radiation Transport Equation

Equation 7 must now be discretized in time, space and angle. This task is complicated by the fact that even though we are solving the equations in one (radial) dimension, two angular variables were introduced to completely describe the azimuthal symmetry of the geometry. Using the S_n method to discretize the angles and standard finite difference techniques to discretize the space and time coordinates, the final form of the radiation transport equation is (see [8] for details)

$$\begin{aligned}
&\frac{1}{c}\{\ \frac{(rr_\xi I_{g;pq})_i^{n+1} - (rr_\xi I_{g;pq})_i^n}{\Delta\tau}\ \} - \frac{1}{c}\ \{\ (rr_\tau I_{g;pq})_{i+1/2}^{n+1} - (rr_\tau I_{g;pq})_{i-1/2}^{n+1}\ \} \\
&\qquad +\mu_{pq}\ \{(rI_{g;pq})_{i+1/2}^{n+1} - (rI_{g;pq})_{i-1/2}^{n+1}\ \} + \frac{2\alpha_{q+1/2}^p}{w_{pq}}(r_\xi I_{g;pq})_i^{n+1} \\
&-\frac{(\alpha_{q+1/2}^p + \alpha_{q-1/2}^p)}{w_{pq}}(r_\xi I_{g;p,q-1/2})_i^{n+1} - (\ rr_\xi(\bar{\eta}_g B_g - \bar{\chi}_g I_{g;pq})\ )_i^{n+1} = 0. \qquad (8)
\end{aligned}$$

Here, r_ξ and r_τ are mesh metric and velocity and need to be known in advance for the new timestep before solving this equation. Also p, q, α and w_{pq} are all defined in terms of the level-quadrature discrete ordinates as described in more detail by Yasar [8]. To solve this equation, we then start at the outer boundary and march inward for decreasing values of i. Once the solution for negative μ_{pq} is found, the boundary condition at the center of the cylinder can be applied. If there is a reflective boundary (symmetry around the axis) then the I values for positive μ_{pq} can be set by the negative μ_{pq} direction I values which have been calculated by inward marching. Then we solve for positive μ_{pq} and increasing i.

2.2. Coupling of Radiation Transport With Hydrodynamics

The effect of radiation transport on the plasma hydrodynamics is represented by terms for radiation energy, pressure and force in the equations for plasma energy and momentum (Equation 1). Likewise, the plasma conditions affect the radiation transport and are used in the calculation of plasma opacity and emissivity. In addition, this coupling between the hydrodynamics and radiation transport equations replaces coupling that would normally exist between the radiation intensities at different energies. That is, under normal conditions, the plasma radiative properties (opacity and emissivity) would depend on the radiation intensities at all energies, which themselves depend on these radiative properties. This dependence would make the radiation transport equation nonlinear and very difficult to solve. This is the reason for the indirect coupling through the hydrodynamics equations. The elimination of this dependence is also critical to solving the radiation transport calculation in parallel, since it makes the radiation intensities at each energy group completely independent of each other over the course of one timestep.

3. Solution Technique on Distributed Processors

3.1. Using a Distributed Network of Workstations

Distributed parallel processing involves the generation of tasks, or processes, which execute on different processors, or cpu's. There is a certain amount of overhead associated with starting up each task on an individual machine. In addition, these tasks must communicate with each other in order to coordinate the work to be done; in practice, this is called synchronizing the tasks. On a distributed system of processors, tasks are synchronized using message passing. These messages have their associated overhead as well. The overhead of starting up multiple processes and passing messages in between them does not exist in a

serial solution on one processor, and can limit the maximum possible speedup obtainable by a certain algorithm. It is important, then, to minimize this overhead.

The overhead required for starting up multiple processes cannot be eliminated. However, if the problem simulation time is long, the overall execution time will usually dominate and this overhead will be insignificant.

On the other hand, the synchronization or message passing overhead cannot be made insignificant by altering the simulation time. It must be minimized in order to maintain speedup, since this is work that would not normally be done in a single processor solution. For constant problem size, increasing the number of calculations done in each task would reduce the number of tasks and therefore the total number of synchronizations required, since sychronization typically occurs at the end of a task. This is called increasing the granularity of a code.[2] However, reducing the total number of tasks limits the number of processors that can be used and reduces the maximum speedup obtainable. Thus, choosing the granularity size is a tradeoff between reducing overhead while maintaining a large maximum speedup.

In a later section, a relation will be derived between the granularity of a problem and the maximum obtainable speedup. For now though, it will be assumed that increasing the granularity of a code will be most important. The number of tasks will usually be equal to the number of processors, to avoid the overhead of starting up extra processes.

3.2. Problem Partitioning

When considering an algorithm for parallel processing, one must isolate portions of code that can be executed concurrently. These portions are then split up into tasks. This is

[2]The *granularity* is a relative measure of the size of a code's tasks compared to the total amount of work done by the code.

called partitioning the problem or algorithm. It is vital that as much of the total execution time as possible be spent in these parallel sections, since parallel speedup is limited by a well known relation called Amdahl's Law:

$$S_p = \frac{1}{(1 - F_p) + \frac{F_p}{N_p}} \tag{9}$$

where S_p is the maxumum achievable speedup for a code with F_p, the fraction of time spent executing parallel code, and N_p, the number of processors used.

It was established in the previous chapter that the calculation of radiation intensities for any group is completely independent of those for other groups, over the course of a timestep.[3] Without considering the solution cost of the hydrodynamic equations, it is clear that virtually all of the execution time would be spent solving for new radiation intensities, successively over energy groups and mesh points. Even the opacities and emissivities could be calculated in parallel over energy groups (in practice they are obtained from a table using plasma density and temperature as independent variables.) When the solution cost of the hydrodynamics equations is included, the radiation transport solution still dominates the execution time. Thus, the fraction of time spent in the radiation transport solution is close to one, and Eqn. 9 shows large maximum speedups over a wide range of processors.

It is also important that the granularity be as large as possible, in order to minimize synchronization overhead. This will determine the partitioning of work among the processors. The granularity of this problem is limited on the upper end by the total amount of work required to solve the radiation transport equation for one timestep. It is limited on the lower end by the calculations for one energy group, since dependencies between mesh points and angles preclude using more than one processor per energy group. Since the calculations for each energy group are independent of all other groups, each group calculation could be

[3]Between successive radiation transport solutions (one at each timestep), the hydrodynamics equations are solved and the plasma conditions are then used to calculate the plasma opacities and emissivities, which are used in the next transport solution.

a separate task. This limits the maximum number of processors to the number of energy groups used in a calculation. If fewer processors are used, each task solves several groups, to minimize synchronization overhead.

This topic will be discussed again in Chapter 5, when synchronization time and task execution times will be estimated. These estimates will be used to predict the actual speedups obtained for these calculations.

3.3. Control Flow

In a distributed system of processors (workstations, in this case), it is necessary to explicitly specify the control flow between the parallel processors as the calculation progresses. This means specifying when and how parallel processes will be started, and how they will synchronize with each other. For simplicity, we decided to use a host-slave paradigm, where a host controlled the startup of the parallel processes on the remote machines and interfaced with the slaves, while the slaves performed all the actual computations. Data was communicated between the host and the slaves using message passing. This message passing could either be synchronous or asynchronous. In synchronous message passing, the sending or receiving process blocks until the operation completes successfully, while asynchronous message passing operations do not block. In this study, since messages were used to synchronize the distributed processes, there had to be some form of synchronous message passing. This came in the form of message receive operations. When processes attempted to receive a message from other processes, the receive operation did not return until a message had been received successfully. Message sending was assumed to be asynchronous.

As we stated in the previous section, each task calculated new radiation intensities for several energy groups, over all mesh points and angles. This was done once for every problem timestep, with synchronization between the host and slaves between timesteps. Since the

coupling with the hydro calculation was through the plasma conditions, each task was synchronized at the beginning of each timestep by its need for input data. That is, in order to find plasma opacities and emissivities for a new timestep, each task needed the new plasma conditions at that timestep. The adaptive mesh parameters used in this solution method were calculated after the hydro solution; therefore, the new mesh parameters were needed by the slaves as input. Likewise, each task synchronized at the end of each timestep by sending its results back to the host. These results were simply the moments of the radiation intensity, which were used in the plasma energy and momentum equations (Equation 1) for the next timestep.

The actual functions of the host and slave programs are shown in Table 1. It was assumed that the hydro part of the calculation would be performed serially, by the host, at the beginning of each problem timestep. When a "timestep" is mentioned below, it means the radiation transport calculation for that problem timestep, not counting the hydro calculation. Note that any of the workstations being used could act as the host.

There are several things to note from Table 1:

1. As mentioned above, the host and slaves were synchronized by their need for data, the slaves at the beginning of each timestep and the host at the end of each timestep.

2. While the host was still sending input data to slaves, some slaves had already started their calculations. This overlapped computation on some of the slaves with communication on the host. Allowing the first slave to begin computing before all the slaves received data meant that the host had less time to wait before receiving results from the first slave. Performing communication on the host in parallel with computation on some slaves reduced the overall finish time of the calculation, and therefore improved speedup.

3. The host took care of initializing the slaves and coordinating the work between them. Thus, it could be substituted for the radiation transport module in the original hydro code. This was a very simple way to use parallel processing in only the parts of the code that need it.

Table 1. Pseudo code for the host and slaves.

Host Code	Slave Code
1. Initialize problem	**1.** Idle
2. Start slaves and estab. conn.	**2.** Initialize data and connect with host
3. Send initial data to slaves	**3.** Receive initial data
For each timestep, do:	*For each timestep, do:*
4. Perform hydro calc. for this step	**4.** Idle
5. Send new timestep data to slaves	**5.** Receive new timestep data
6. Initialize rad. moments to zero	**6.** Calc. new intensities and moments
7. Receive rad. moments from slaves	**7.** Send radiation moments to host
8. Sum partial rad. mom. into total mom.	**8.** Save new var's into old var's

The only remaining issue was how to terminate the slaves. This could be done by passing a special flag to each slave (which was expecting input data for the next timestep), telling that there would be no next timestep. The slaves, upon receiving that message, simply terminate. This was the only change required in the hydro code outside the radiation transport module, i.e. the radiation transport module had to be called one last time and passed a special terminate flag.

4. Distributed Processing Tools

There were many distributed processing tools in the public domain at the time of this project. The tools fell mainly into two categories: function libraries and distributed operating systems. It was a fortunate situation to be able to choose between so many different packages; however, it was important to decide which features were needed and which weren't. Unneeded features added overhead and, just as important, made packages more difficult to use and maintain. Therefore, before considering all the various distributed processing tools in the public domain, we considered it important to decide which features were and were not important.

4.1. Necessary Features

The features we decided on were:

Remote Job Startup: Using the host slave paradigm, we required the ability for the host to start up slave processes on remote machines.

Message Passing: The ability to pass floating point numbers between processes on remote machines was crucial. The numbers had to be passed in machine independent form, since we wanted to utilize different kinds of UNIX workstations in the department.

Portability: The majority of the UNIX workstations in the department were made by either Sun Microsystems or Digital Equipment Corp. Thus, the software package had to be portable to these architectures. It was desirable to have portability between most other architectures as well, to add flexibility to the code.

Speed: The overhead of the message passing routines determined the minimum granularity of the tasks in the program. If this overhead dominated the largest possible granularity, it would not pay to run this calculation in parallel at all.

User-Friendliness: Last but certainly not least, the package had to be user friendly.

4.2. Unnecessary Features

In addition, there were several features commonly included with distributed processing tools that we did not need. These features usually added overhead and complicated the calling sequences for even the basic functions. These features were:

Fault Tolerance, Security: Although these features are extremely important in a large computing environment, they also add lots of complexity and can give a false sense of security. Since the purpose of this project was to investigate the possible advantages of distributed computing and not to develop a production code, we did not consider the issues of security and fault tolerance. When it is time to develop a production code, however, we recommend strongly that the code be implemented with these issues in mind.

Complicated Slave Control: Again because we chose not to develop a production code, we did not need complete control over processes executing on remote processors. In this case, this means that we would not use the program to kill off any runaway or deadlocked processes. It was assumed that the user would do this. Note that this could be a difficult thing if many (100) processors were used.

Dynamic Load Balancing: The situation could arise where one workstation became overloaded due to other users on the same machine. While it would be desirable to be able to decrease the amount of work required of such machines, building this capability into the first version of the code was beyond the scope of this project. The easiest way to implement this capability would be to have the host adjust the number of energy groups given to each slave. However, it would be necessary to redistribute

the "old" intensity values; this would require a great deal of overhead, and might outweigh the benefits of decreasing the work on the loaded workstation.

4.3. Available Public Domain Tools

In light of the requirements listed above, we examined the various options.

The distributed operating systems were capable of performing very high level functions, such as dynamic load balancing, but the added features came at the price of overhead and added complexity. They also required much more effort to install and maintain. Therefore, they were ruled out from the start.

This left us with several very good function libraries. Among these were the ISIS package from Carnegie Mellon University [1], the Cosmic Environment from Caltech [5], and Sun RPC from Sun Microsystems [7]. These packages had some or all of the following capabilities, bundled with their own calling syntax:

Remote Procedure Call: The ability for a program running on one workstation to call a procedure (create a process) on a different machine.

Authentication, Authorization: The ability to award or deny access to users or users' (remote) processes based on some authorization conditions.

Fault Tolerance: The ability to restart processes that are lost due to machine failures or other reasons.

Process Grouping and Allocation: The user could create an arbitrary grouping of processors and interact with them as a group.

Obviously these packages contained many features that we did not need. Including some of the features such as process grouping and remote procedure call added complexity

to the calling sequence, and all these packages probably incurred overhead by providing these functions. We decided therefore that we needed something simpler and more lightweight.

Fortunately, there were other alternatives. Although we did not find one package with all the desired features, we were able to combine several easy to use packages that suited our needs.

Unix Sockets and Utilities: Virtually all RISC workstations ran the UNIX operating system. Although there were several versions of UNIX, all versions included the Berkeley Socket library, which was built on top of the TCP/IP network protocol. This library provided simple functions to establish connections (called sockets) between processes executing on local or remote machines, and functions to pass messages between processes. Although the sockets were a bit tricky to set up, passing messages through them used the same read and write routines used to read and write disk files in the C programming language.

UNIX also provided a high level command for remote job execution (rsh). Using this command the slave programs could be started up on remote machines; the host would then be started up on the local machine as a normal executing program. All standard input and output was routed through the local machine.

Using the UNIX library calls provided the best portability possible. Since many higher level distributed processing tools were built on top of these library calls, they were also the least expensive in terms of overhead. There were disadvantages, however. The socket connections were difficult to set up, and there were no high level functions for passing floating point numbers between different machines.

Switchboard Package: A process switchboard package [2], written at the University of Wisconsin-Madison, provided a simple way for processes to establish socket connections with each other. As the name implies, processes called up the switchboard (through a

function call), passing it their name and the name of the process with whom they wanted to establish a connection. The function blocked until it found another process making an analogous call, then passed back to each an integer to be used as a file desriptor in normal UNIX read and write calls. This package was used only at the beginning of the program, to initialize the socket connections. The switchboard process ran as a system process on each workstation, and was easily installed.

XDR Package: We still did not have a way to pass machine independent floating point numbers. A simple method would have been to translate the floating point numbers into character strings (using FORTRAN internal files), then pass ASCII strings, retranslating on the other end. However, this would have been prohibitively expensive, since every floating point number would have required 5 bytes plus a byte for each digit of precision, compared to the 4 bytes required to store the floating point number in the computer. A solution was found that used the XDR (eXternal Data Representation) protocol [6]. This was simply a protocol for passing machine independent data between computers. Functions were provided to translate various data types to and from their external representation; floating point was among those data types. This library of functions was installed on the Sun and Dec workstations that were to be used for this project. It was not known what other architectures had XDR implementations, but we assume that most of them did have XDR functions.

Used together, the UNIX function libraries and commands and the Switchboard and XDR packages provided the simple, portable functionality required of this project.

5. Performance Estimates

Before undertaking a project of this type, it is wise to estimate the maximum possible speedup, to ensure that the problem is worth solving in parallel at all. This involves

benchmarking the communication routines and the computation speed, and using these to form analytical speedup estimates.

5.1. Benchmarks

The overhead involved with starting up remote programs and establishing connections between them is a one-time cost. This cost can be made insignificant to the total calculation time by increasing the size or duration of a calculation[4]. However, the message passing overhead for each task is not constant but scales with problem size and duration. Thus, we must ensure that the message passing overhead is insignificant compared to the computations done between synchronization points. This requires us to benchmark the message passing and computation costs.

A program was written to measure the cost of passing floating point numbers between workstations over the network. The cost of a round trip between two workstations was measured, where each leg of the trip involved encoding the numbers, passing them to the remote machine, and decoding the numbers. The cost was represented as the wall clock time required to pass one floating point number to another machine. Measurements were made between machines that were physically close to each other (one or two meters) and between machines that were far away (several tens of meters.) The results are shown in Figure 3.

It is obvious from this figure that there is some "magic" message size that occurs between about 510 and 760 floating point numbers. After investigating further, we found that messages in this range used one chunk of 2048 bytes to pass characters over the network, plus a small chunk to pass the remaining bytes. Larger messages were using 3072 byte chunks, while smaller messages were using 1024 byte chunks. While we did not investigate

[4]Within certain limits, of course. The longer calculation must be representative of those done with serial codes, or at least must be useful for something other than demonstrating the benefits of parallel processing.

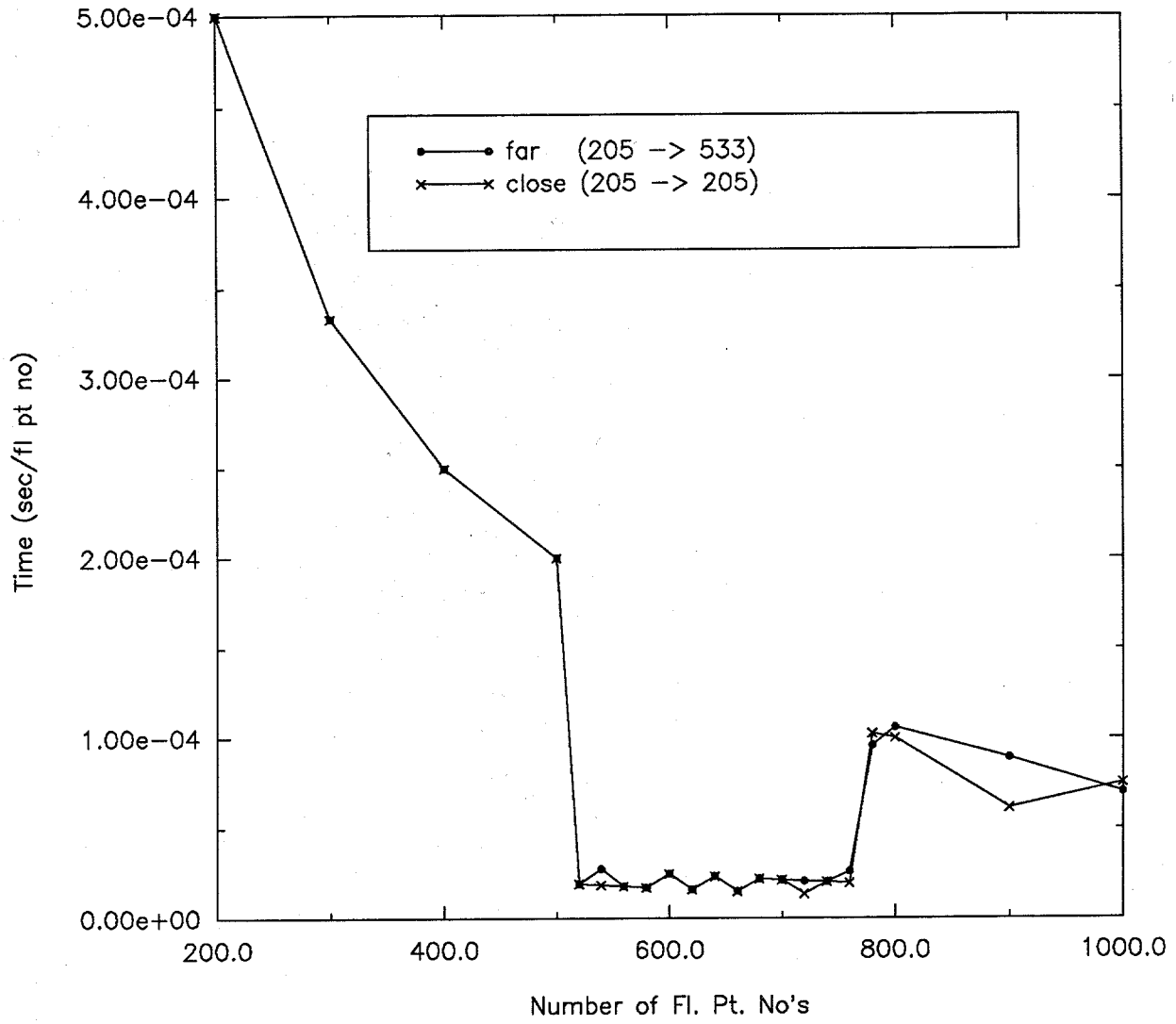

Figure 3. Message passing times for passing floating point numbers between workstations at varying distance (within the same building.) Includes coding and decoding time.

further, it is thought that either the particular machines and/or the network work best with messages of 2048 byte size. This phenomenon was considered when writing the message passing routines. That is, smaller messages were padded to bring them to the optimal size of 2048 bytes.

Several simple loops were run on a Sun Sparcstation 1 to measure the computation cost for each floating point operation. After subtracting the loop overhead cost, the cost for each floating operation was 2.0 microseconds. Note that all the test programs were compiled in debug mode, which produced code that ran at speeds below the benchmark speeds for these machines.

5.2. Communication to Computation Ratio

It is worthwhile solving a program in parallel only if the computations in each task dominate the overhead of running those tasks. We have already stated that the overhead of starting the tasks and establishing the communications between them can be made insignificant by increasing the problem duration. However, the synchronization overhead for each task scales with problem duration just as the number of computations does. This overhead must be estimated and compared to the computations done in each task.

The computation cost per task was estimated simply by counting up the floating point operations done in each task, giving equal weight to all operations (add, subtract, multiply, divide). All other operations (compare, assignment, etc.) were neglected. The communication required for each task was counted as well, in terms of how many numbers had to be passed. Note that for estimating the actual communication cost, the fact that 2048 byte messages were ideal was considered, since the code padded small messages to bring them to 2048 bytes. The computation and communication costs are shown in Table 2.

The ratio of the communication to the computation speed is called the communication to computation ratio, R_{cc}. This is just the ratio of the total communication and computation costs shown in Table 2:

$$R_{cc} = \frac{2 \cdot max[(4 \cdot n_p + 1), 511] \cdot S_{comm}}{[845 \cdot \frac{N_g}{NP} + 1] \cdot n_p \cdot S_{comp}}, \tag{10}$$

where S_{comm} = communication cost in seconds per floating point number passed, and S_{comp} = computation cost in seconds per floating point operation. R_{cc} was calculated using Eqn. 10 with $n_p = 100$ (a typical spatial mesh size) and for varying numbers of processors NP and energy groups N_g. The time required to pass a floating point number, from Figure 3, was taken to be 1.8e-5 s/floating point number, and the computation time was taken to be 2.0 microseconds. The results are shown in Table 3. This data shows that communication cost becomes significant compared to computation cost only when each processor calculates only one or two groups. Otherwise the computation cost dominates.

5.3. Speedup Estimates

The communication to computation ratio gave us an idea of whether it was worthwhile to use parallel processing for our problem. Since that test had positive results, it was instructive to form an estimate of the speedups that would be obtained when actually running the problem in parallel. In our distributed computing paradigm where the host and slave processes are running on distributed machines and communicating using system library calls, the interaction with the host workstation and with the network would not be deterministic, and could vary with network and workstation load from other users. Since these variations would be difficult to quantify, estimates of the upper and lower bounds to the speedup would be both easier and more meaningful.

Under favorable conditions, we assumed that all slaves would perform calculations at the speed measured on a single user system. We also assumed that the messages were passed

Table 2. **a) Computation cost in terms of floating point operations (each addition, subtraction, etc. given equal weight), and b) Communication cost in terms of number of floating point numbers to pass. Here n_p = number of spatial mesh points, NP = number of processors (workstations) used, and N_g = the TOTAL number of energy groups.**

Operation	Cost
Look up opacities and emissivities	$[11 + 13 \cdot n_p] \cdot \frac{N_g}{NP}$
Calculate new rad intensities	$684 \cdot n_p \cdot \frac{N_g}{NP}$
Sum moments	$[161 \cdot \frac{N_g}{NP} + 1] \cdot n_p$
Total computations	$[845 \cdot \frac{N_g}{NP} + 1] \cdot n_p$

a) Computation cost

Operation	Cost
Accept input from host	$max[(4 \cdot n_p + 1), 511]$
Pass output to host	$max[(4 \cdot n_p), 511]$
Total numbers communicated	$2 \cdot max[(4 \cdot n_p + 1), 511]$

b) Communication cost*

*For each vector of length n, the vector length is passed in an extra number. Thus the length of the character string message for a vector of 511 numbers is $(511 + 1) \cdot 4 = 2048$ bytes.

Table 3. Communication to computation ratio R_{cc} for $n_p = 100$, and a) $N_g = 20$, b) $N_g = 100$.

a) $N_g = 20$		b) $N_g = 100$	
NP	R_{cc}	NP	R_{cc}
2	.011	2	.002
4	.022	4	.004
8	.044	8	.009
10	.054	10	.011

from the host to the slaves without delay, and that they all took the same amount of time. The time required for a parallel solution for this case can be estimated using a timeline for the host and the NP'th slave (see Figure 4). For every timestep, the host sends the initial timestep data to each slave; from Table 3, the time in seconds required for each slave is

$$T_{comm} = max[(4 \cdot n_p + 1), 511] \cdot S_{comm}. \tag{11}$$

After receiving the input data, each slave immediately starts its calculations. The last slave receives its message after $NP \cdot T_{comm}$ seconds. This slave performs its calculations in a time of

$$T_{comp} = [845 \cdot \frac{N_g}{NP} + 1] \cdot n_p \cdot S_{comp} \tag{12}$$

seconds. Meanwhile, the other slaves have finished their calculations and are passing results back to the host. Since they all took the same amount of computation time, they should all be finished passing data back to the host by the time the last slave is ready to pass back results. This requires about T_{comm} seconds. Therefore, the overall time required for a

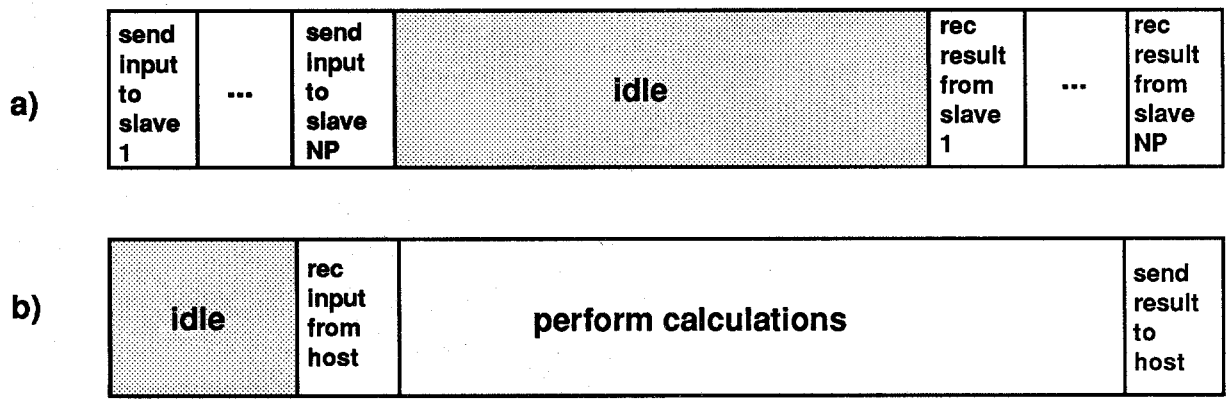

Figure 4. Gantt chart of a) host process and b) NP'th process.

parallel computation on NP processors is

$$T_{NP} = (NP + 1) \cdot T_{comm} + T_{comp} \ . \tag{13}$$

The serial time is just Equation 12 evaluated for NP = 1, without any communication time. Therefore, the speedup on NP processors is

$$S_{NP} = \frac{T_{comp}(NP = 1)}{T_{NP}} \ . \tag{14}$$

Equation 14 was evaluated for various values of NP and N_g, with $n_p = 100$; the results are shown in Table 4.

The estimated speedups in Table 4 represented upper bounds based on the most favorable conditions. Now we must form analogous estimates for unfavorable conditions. For this case, we assumed that all communication had to be finished before slaves could start performing computations for the timestep. We still assumed that each slave was allocated all of the execution time on its particular workstation, and that startup time was insignificant. The computation and communication time for each slave in this case was the same as

Table 4. Upper bound estimated speedups for $n_p = 100$, and a) $N_g = 20$, b) $N_g = 100$.

a) $N_g = 20$			b) $N_g = 100$		
NP	S_{NP}	Proc Eff (%)	NP	S_{NP}	Proc Eff (%)
2	1.97	98.4	2	1.99	99.7
4	3.79	94.8	4	3.96	98.9
8	6.69	83.6	8	7.70	96.2
10	7.69	76.9	10	9.43	94.3

for the favorable case. The order of when functions were done was different, however. Instead of overlapping communication from host to slave with computations being performed in other slaves, no overlap was allowed. Slaves were assumed to start computing when all communication from the host was finished. Since each slave was given the same amount of computations, the slaves all finished at the same time, then waited to pass the results back to the host. The parallel time is just increased by the time required to accept results back from all slaves instead of just the last one. So, the estimated speedup would be

$$S_{NP} = \frac{T_{comp}(NP = 1)}{(2 \cdot NP) \cdot T_{comm} + T_{comp}} \ . \tag{15}$$

This expression was evaluated for the conditions in Table 4; the results are in Table 5.

These lower bounds on estimated speedup are still quite good, especially for a large number of energy groups. Figures 5 and 6 summarize the information in Tables 4 and 5, for $N_g = 20$ and $N_g = 100$, respectively. The upper and lower bounds on estimated speedup are very close together, especially for $N_g = 100$; this is because the communication time was very small and did not affect the overall calculation time very much.

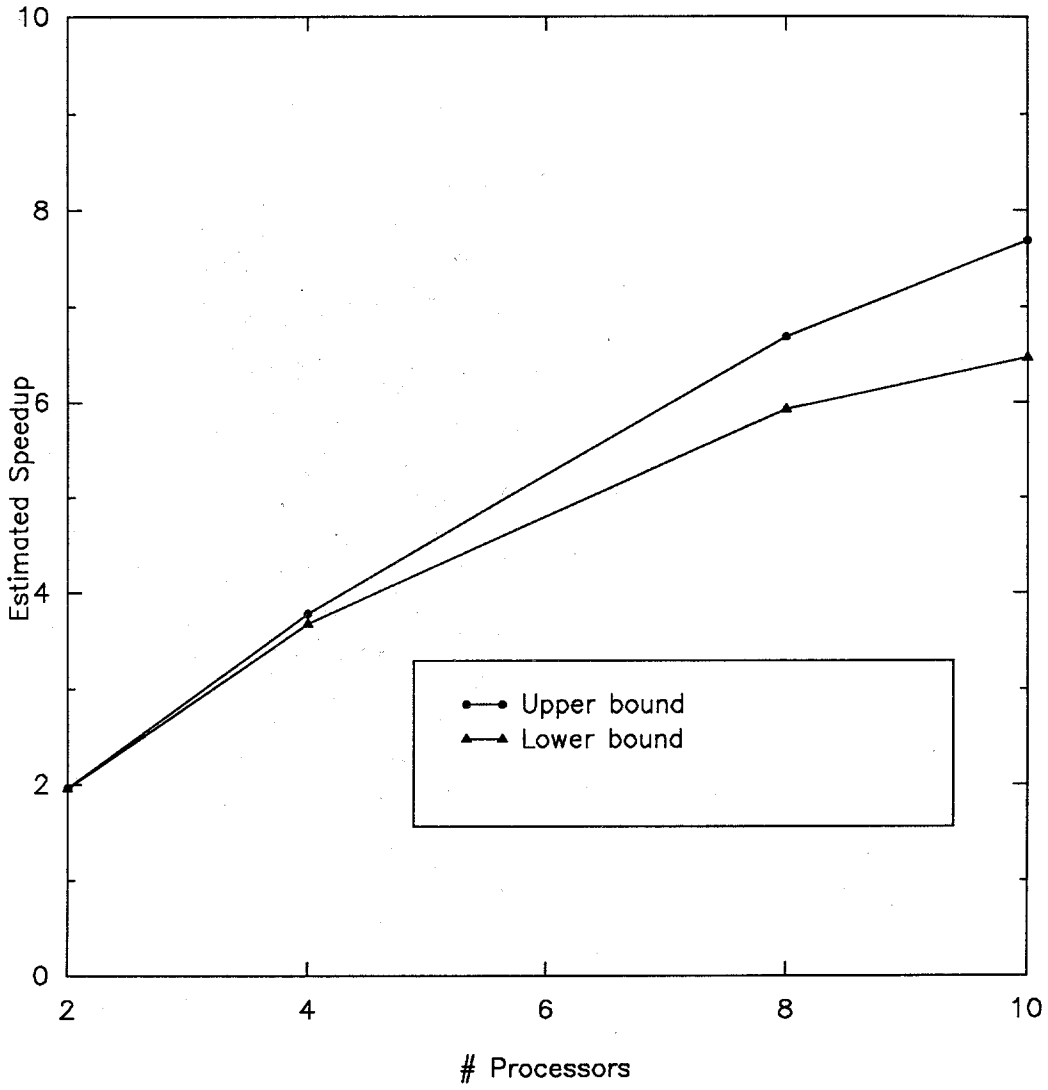

Figure 5. Upper and lower bounds for estimated speedup, for $N_g = 20$.

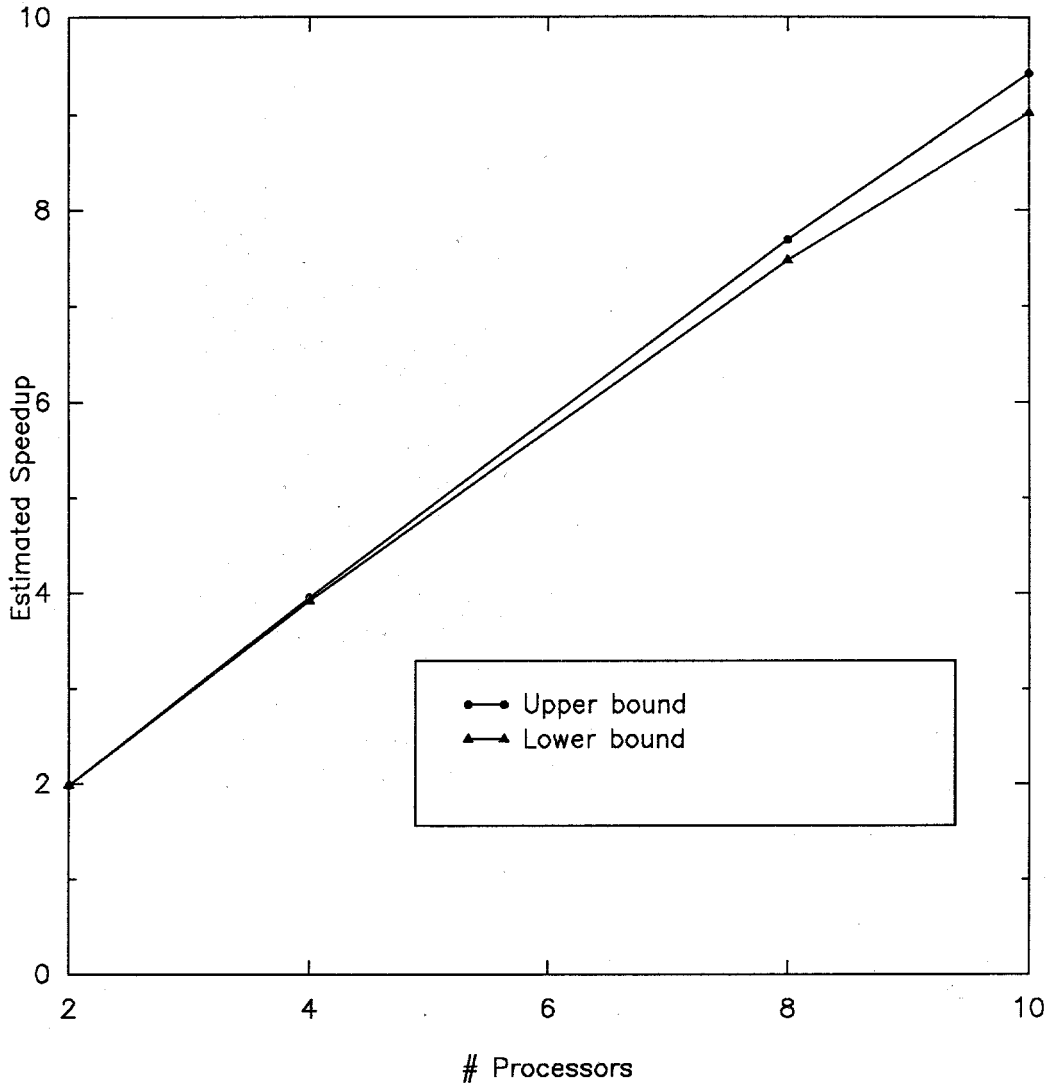

Figure 6. Upper and lower bounds for estimated speedup, for $N_g = 100$.

Table 5. Lower bound estimated speedups for $n_p = 100$, and a) $N_g = 20$, b) $N_g = 100$.

a) $N_g = 20$

NP	S_{NP}	Proc Eff (%)
2	1.96	97.9
4	3.68	92.0
8	5.93	74.1
10	6.47	64.7

b) $N_g = 100$

NP	S_{NP}	Proc Eff (%)
2	1.99	99.6
4	3.93	98.3
8	7.48	93.5
10	9.02	90.2

6. Results

For simplicity, the radiation transport calculation was not implemented within a full hydro calculation, but was run with a driver code instead. This code passed new timestep values of mesh points and plasma conditions to the radiation transport calculation, as if a hydro calculation were actually being performed. A simple steady state problem was used for all the timing tests, where plasma hydrodynamic conditions and mesh point locations were kept constant. The full radiation transport calculation was still performed for each timestep.

A serial version of the driver program was written that simply called the slave code and passed it data through an argument list instead of using messages. The slave performed calculations for all the groups instead of just a subset of groups. This code ran on one workstation (processor) and served as a basis of comparison for the parallel version.

The sample problem used for timing tests performed calculations for 20 energy groups. Although we chose 20 energy groups because of limited group opacity data, we note that

this severely limited the number of processors we could use and the granularity size that we could attain. Since the computation cost scaled with the granularity and the communication cost did not, having smaller granularity increased our communication to computation ratio. This reduced the maximum speedups attained with the parallel code.

6.1. Parallel Speedup For Various Problem Durations

Since we did not attempt to predict the overhead of starting up parallel processes on remote machines, it was necessary to measure the speedup for various problem durations. As the problem duration was increased, the one-time startup cost became insignificant and the measured speedup stopped changing. The goal was to find the minimum problem duration required to achieve as close as possible to asymptotic speedup. Problem duration was measured in terms of the number of timesteps. The length of the timesteps was not important, since it had no effect on the number of calculations done during each timestep[5]. Calculations were run for 1, 10, 100 and 1000 timesteps, on two and four processors.

The parallel speedups for these calculations are shown in Figure 7. This figure shows that for two processors, the speedup leveled off very quickly, so that 100 timestep calculations were adequate for obtaining asymptotic speedup results. In the four processor case, however, this was not the case. The speedup did not level off below 100 timesteps, and even increased somewhat between 100 and 1000 timesteps. Therefore, speedup measurements for four processors had to be made on calculations of 1000 timesteps or more.

[5]This is in contrast to iterative codes, whose accuracy often depends on the timestep length. Increasing the timestep in these codes would cause the calculations to be less accurate, which would require more iterations for an acceptable solution.

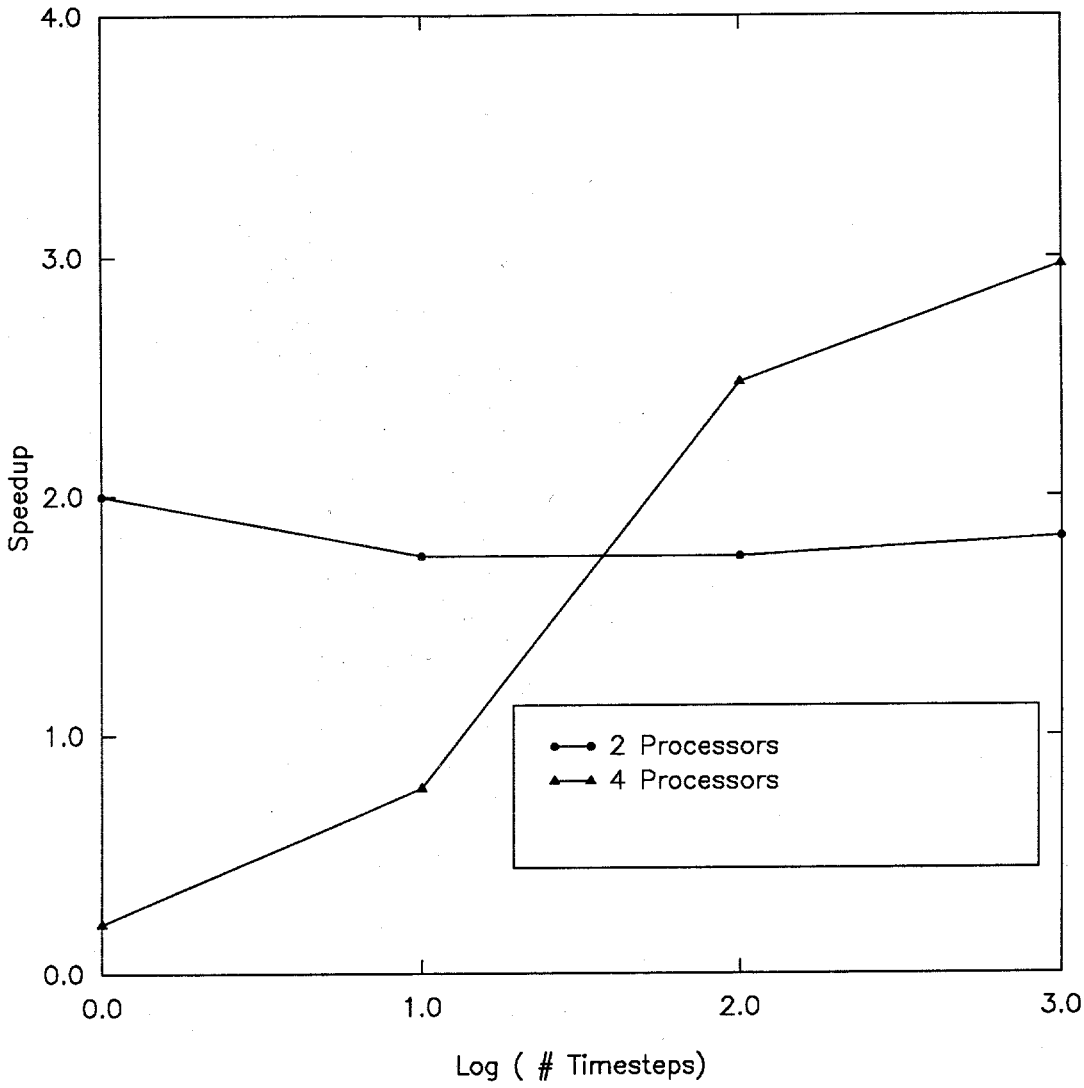

Figure 7. Measured speedups for two and four processor calculations, with respect to problem duration; duration is specified by the number of timesteps calculated in the problem.

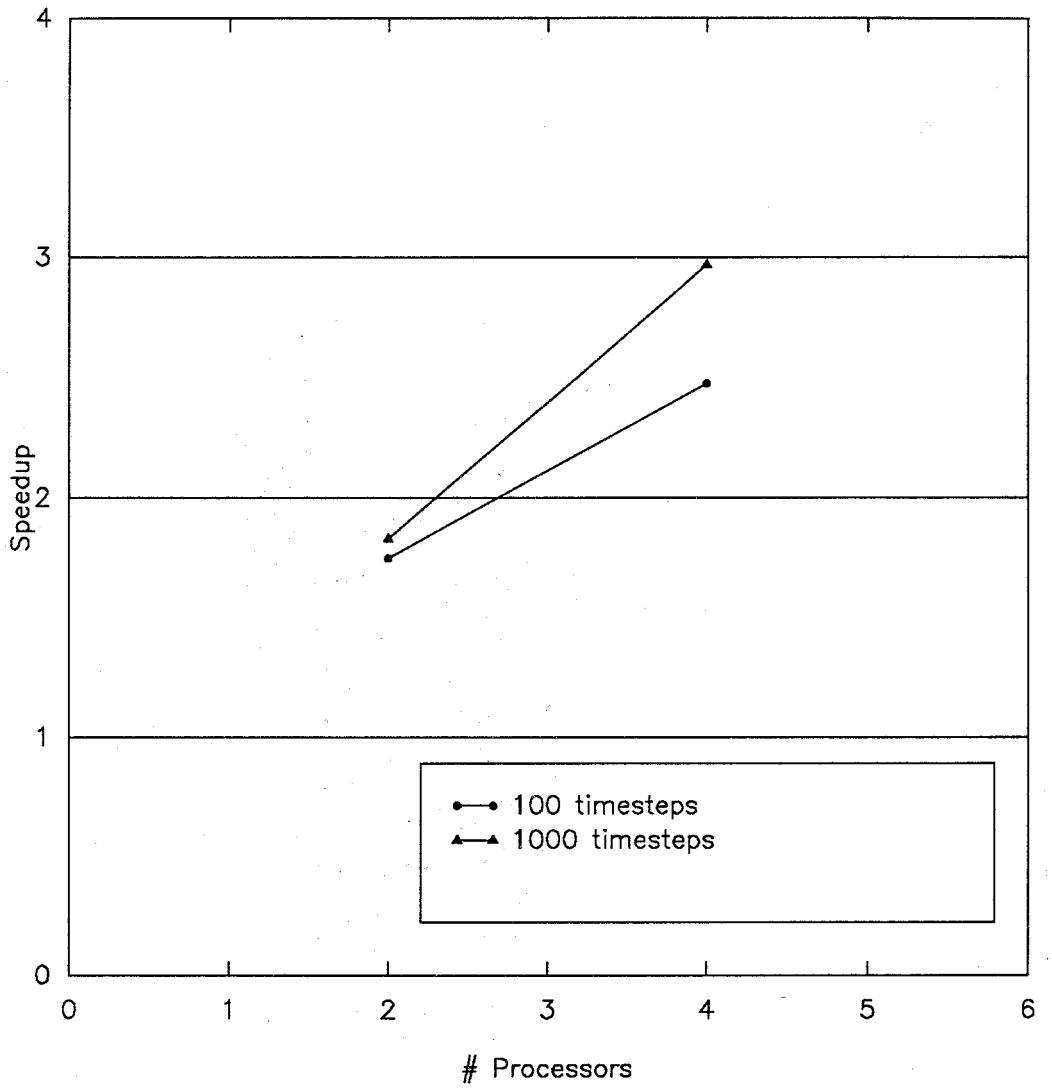

Figure 8. Measured speedups for two and four processor calculations, for $N_g = 20$.

6.2. Measured Parallel Speedup and Processor Efficiency

In Figures 8 and 9 the parallel speedup and parallel efficiency (speedup divided by the number of processors) are plotted with respect to the number of processors. Here we can see that the speedups were not as good as we predicted in Chapter 5. However, they were still very acceptable. This shows that calculations run on 4 workstations in parallel could still run significantly faster than those run in serial fashion on one workstation.

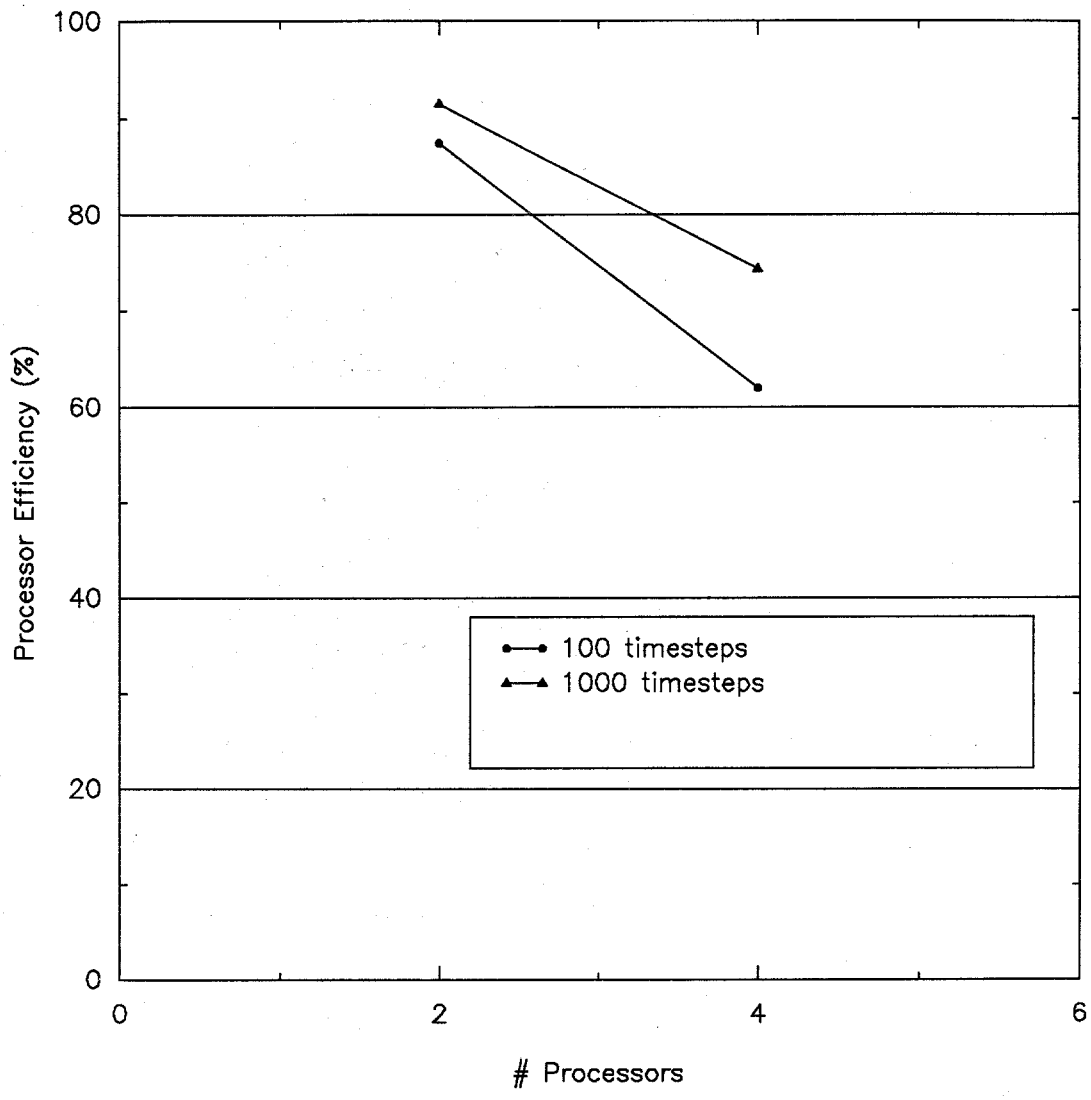

Figure 9. Measured processor efficiencies for two and four processor calculations, for $N_g = 20$.

6.3. Analysis

There were several possible reasons for the measured speedups being lower than those predicted in the last chapter:

1. The slaves running on remote machines may not have been getting exclusive access to the remote machines' resources. When a serial program is run on a local workstation, the user has much more control over the local system load, since much of the normal load is due to terminal interactions. On remote systems, there may be terminal interactions taking place that are beyond our control. Since these tests were performed mainly during daytime hours, there is a good chance that this was happening.

2. Normal network or remote machine activity may have delayed the startup of remote jobs. This would be a non-deterministic effect, and could cause the startup time to be significant in some cases.

3. Non-representative prediction of message passing benchmark times or poor behavior of the network could cause differences between the predicted and actual speedups.

To improve the measured speedup, there were several modifications we could make. The easiest way to improve the speedup would be to perform the calculation for more energy groups. Since the computation time scales with N_g and the communication time does not, this would improve our communication to computation ratio, and would therefore increase speedup. This would improve the accuracy of the radiation transport calculations as well.

As the program was written, the host polled the slaves for results in the order that the slaves were started. This assumed that all slaves required exactly the same time for computations. If this was not the case, the host could end up waiting for a message from

one slave, while there were other messages to be received. Using routines found in the UNIX socket library, one could instead poll several socket connections, and find one that contained an incoming message [2]. That way, the host could accept messages from the slaves on a first come first served basis. This would allow the host to accept messages from slaves that were ready to send them, while the remaining slaves completed their computations.

If we adhered strictly to the host slave paradigm, as we did up to this point, the host performed *all* of the communication with the slaves. This could cause the host to be a bottleneck, especially for large numbers of slaves. This problem could be eliminated by allowing the slaves to take part in the message passing to other slaves. Instead of the host passing the initial data to each slave in succession for each timestep, it could pass the data to a subset of slaves. While the host was still passing the input data to other slaves, the first slaves to receive data could pass it to other slaves, and so on. This is called broadcasting the message, and would reduce the communication time from a scale of NP to a scale of $log_2(NP)$. This would reduce the communication time by a factor of two for four processors, almost three for 8 processors. An analogous method could be used for gathering results, with the slaves accumulating the summed moments along the way. The moment summations could be performed in parallel as well, reducing the computation time of the host process as well.

We did not investigate what was really being done inside the switchboard package, to see whether the package was setting up the optimal socket configuration for our problem. With a little work, the Berkeley socket routines could be called directly from our code. This would reduce overhead slightly and would give us more control over the socket configuration. In addition, it would no longer be necessary to install and maintain the switchboard software on each machine used by this program.

6.4. Improved Speedup Estimates

It is possible that the message passing benchmark program was too simple to accurately represent the message passing behavior in the radiation transport calculation. If this was true, we might be able to more accurately predict the speedups using a different value for the floating point message passing time. The most obvious value we could use could be obtained from the curve in Figure 3. If the curve outside the "magic" range between 511 and 767 numbers was continued inside that "magic range", we could use the value on that curve for 511 floating point numbers. We used this value in Equations 13 and 15 to obtain new speedup estimates; the results are shown in Figure 10, along with the old and measured speedups, for two and four processors.

We see that the new estimated speedups were below the measured speedups, by about the same amount as the old estimates were above the measured speedups. This shows that the actual floating point message passing times were probably somewhere in between the values used in the old and new speedup estimates.

7. Conclusions

The data in Chapter 6 showed that a distributed network of workstations could indeed be used to perform parallel calculations, given a suitable communication to computation ratio for the problem being solved. However, it also showed the sensitivity of these speedups to the cost of passing messages between workstations over the network. Using more than four processors would increase the communication to computation ratio, resulting in lower parallel efficiencies. The existing code would have to be improved before being suitable to run on more than eight processors or so. It would be most important to implement the second and third improvements discussed in Section 6.3. For this calculation it would

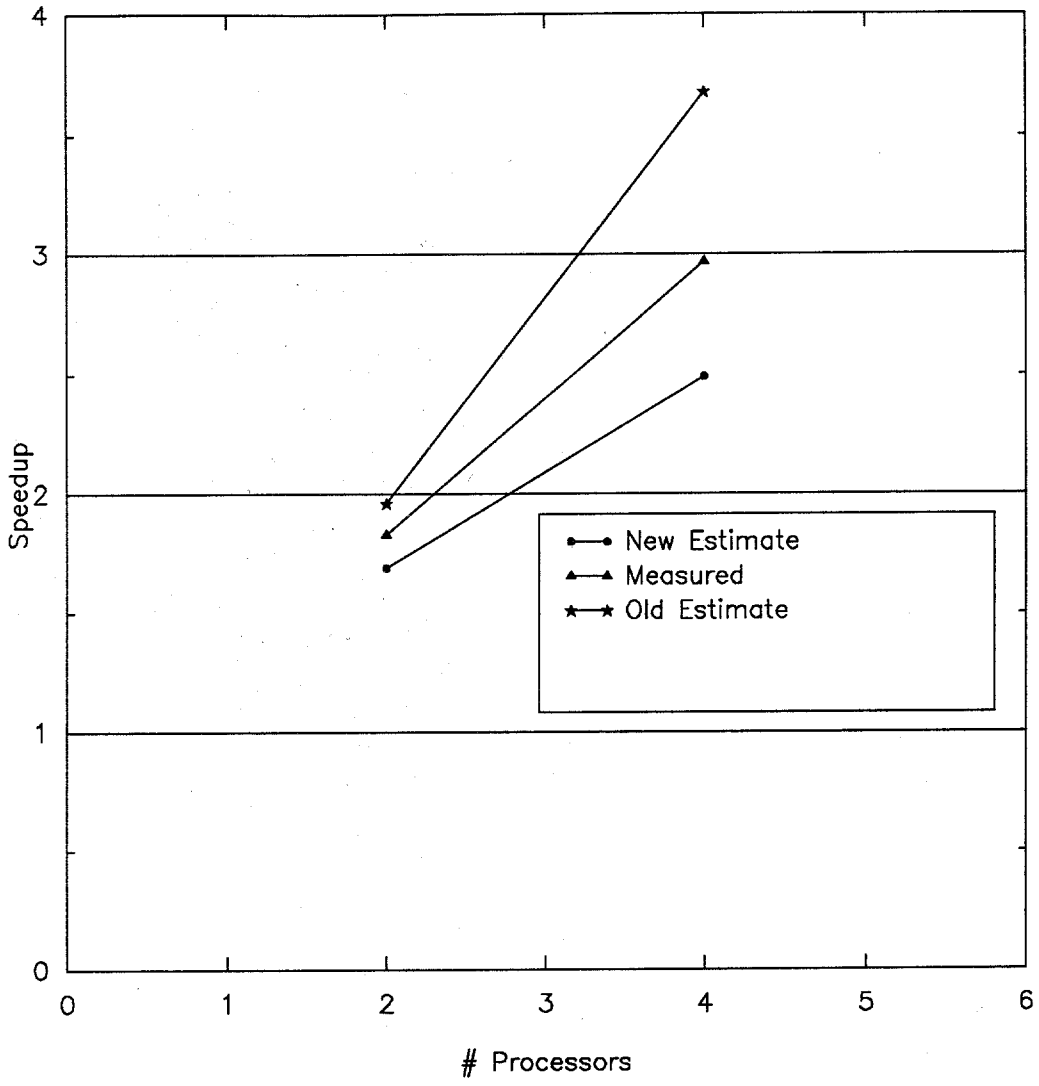

Figure 10. New estimated speedups using a larger floating point message passing time, compared to old estimated and measured speedups, for two and four processors.

also be necessary to increase the number of energy groups, since this severely limited the granularity size and the maximum number of processors.

This research project could proceed in several directions from here. The first thing that would have to be done is to generate opacity data for a larger number of energy groups, and to perform timing studies of the code using that data. That would give better speedup results because of larger granularity, and would allow us to perform tests on more than four processors. The program could then use several different workstation architectures. This was not done in the timing tests in this report, but would be useful to demonstrate the portability of the code.

Since the code was written for a distributed network of processors, it could easily be ported to a more tightly coupled distributed processor machine, such as the Intel hypercube. These machines have message passing routines similar to those written for this code, and of course they have routines for starting up jobs on slave processors [4]. Benchmarks on this machine and on a similar machine, the NCUBE hypercube, have shown the message passing time to be approximately ten microseconds per floating point number and the computation time to be 17-40 microseconds per floating point operation [3]. This would give a communication to computation speed ratio of 0.35, compared to the ratio of 10-100 attained with the network of workstations. This would substantially increase the parallel speedup on these machines. They usually contain 100 - 1000 processors, and so the maximum achievable speedup would also be much higher on these machines, given a problem large enough to keep all the processors busy.

Finally, if this code was to be used for production calculations on the network of workstations, it would have to be more robust. The code would have to be responsible for shutting down runaway slaves, and would have to be able to restart failed calculations.

The principles used in this report to determine the suitability of the radiation transport problem to parallel processing can be used on any code with a similar structure, where most of the run time is spent in loop-type computation and where the loop iterations are independent. The overriding issue for these codes is the communication to computation ratio of the parallel tasks. If there are not enough computations done in tasks to dominate the time required for communication, the tasks simply will not be suitable for running in parallel.

The use of a distributed network of workstations to perform large engineering calculations has the potential for large cost savings. The cost of calculations performed on workstations is not proportional to the amount of execution time used; in fact, there is relatively little cost after the initial purchase. This is in contrast to large computing centers, which recover costs based on cpu usage. Therefore it is to everybody's advantage to get all they can out of workstations that would otherwise stand idle; this report has shown how this can be done. Considering the declining cost of workstations and their recent proliferation into most research institutions, workstations will likely become an alternative to supercomputers for performing scientific computing.

References

[1] K. Birman, R. Cooper, T. Joseph, K. Kane and F. Schmuck, "The ISIS project," *The ISIS System Manual, Version 1.1*, (1989).

[2] D. Draheim, B.P. Miller, S. Snyder, "A Reliable and Secure UNIX Connection Service", Proceedings of the 6th Symposium on Reliability in Distributed Software and Database Systems, Williamsburg, VA (March, 1987).

[3] T. H. Dunigan, "Performance of Three Hypercubes", ORNL-TM-10400, Oak Ridge National Lab, TN (May 1987).

[4] Intel Corp., *Intel Hypercube Fortran Programmer's Manual*, 1989.

[5] Charles L. Seitz, Jakov Seizovic and Wen-King Su, "The C Programmer's Abbreviated Guide to Multicomputer Programming", Caltech Computer Science Technical Report Caltech-CS-TR-88-1, Jan. 1988.

[6] Sun Microsystems, *External Data Representation Protocol Specification, Release 2.0*, 1989.

[7] Sun Microsystems, *Sun Remote Procedure Call Library*, 1989.

[8] Osman Yasar, "A Computational Model For Z-Pinch Plasma Channels", University of Wisconsin Fusion Technology Institute Report UWFDM-823, Feb. 1990.

[9] G. Yonas, "Particle Beam Fusion", Bull. Am. Phys. Soc. 21, 1972, p. 1102.