Application Load Imbalance on Parallel Processors

Vasudha Govindan
vasu@hpc.uh.edu
High Perf. Comp. Center
University of Houston
Houston, TX 77204-5502

Mark A. Franklin
jbf@random.wustl.edu
Comp. & Comm. Res. Center
Washington University
St. Louis, MO 63130

Abstract

This paper addresses the issue of dynamic load imbalance in a class of synchronous iterative applications, and develops a model to represent their workload dynamics. Such models of application load dynamics help in more accurate performance prediction and in the design of efficient load balancing algorithms. Our model captures the workload dynamics across iterations, and predicts the workload distribution at any given iteration as the cumulative effect of workload dynamics during the preceding iterations. The model parameters are derived using empirical data from initial runs of the application. The model development is illustrated using data from a parallel N-body simulation application.

1 Introduction

Uneven distribution of workload among processors in a parallel algorithm often results in some processors being idle and underutilized while others are heavily loaded. Such load imbalance often results in poor overall performance. Many important problems in science and engineering have dynamically changing workload distributions. N-body simulations, adaptive meshes and discrete-event simulation are some examples. In these situations, the workload distribution is typically irregular and changes during execution.

This paper focuses on modeling the dynamically changing workload distribution associated with a class of synchronous iterative applications, and deriving the performance of such applications when executed on a parallel computer. A good understanding of the application load dynamics is important for several reasons: (1) It enables more accurate performance prediction, (2) it aids in the design of effective load balancing algorithms, and (3) it provides insight and improvement in application solution algorithms.

Our work is different from other models for workload dynamics\(^1\) in literature\([5, 2, 11]\) in that we do not assume that the task distribution at any iteration is independent of the distribution at its preceding iterations. Several applications like the N-body simulations, particle dynamics, adaptive meshes, etc., have workload characteristics that change with each iteration. Our model captures the workload dynamics across iterations, and predicts the workload distribution at any given iteration as the cumulative effect of workload dynamics during the preceding iterations. The model parameters are derived using empirical data from initial runs of the application. Model development uses concepts from stochastic modeling, and extreme value theory.

Section 2 derives a general model for workload dynamics and, using empirical data from N-body simulation runs, derives parameters specific to the application. Based on this dynamic model, section 3 presents a performance prediction model which permits evaluation of overall performance. Section 4 is concerned with model verification, where, model and measured results for a selected application (N-body simulation) are compared. Section 5 concludes the paper and presents some suggestions for further work.

Most of the papers in literature that deal with modeling workload distribution of parallel applications consider a set of independent tasks, with identical or random execution times, randomly distributed over the available processors. The execution time distribution in these cases can be modeled by a normal distribution\([9, 5, 1]\). In \([2]\) and \([11]\), the workload distribution is derived based on empirical data obtained from the application. Both these models assume that the workload distribution at any iteration is independent of the distribution at all other iterations and that the distributions for all iterations are identical. We focus on a class of iterative applications where this is not true and the workload distribution changes each iteration. As a result, the analysis

\(^1\) Note that workload distribution refers to the distribution of work over the processors and workload dynamics workload dynamics refers to the change in workload distribution.

---

\* Supported by NSF grant CCR-9021041 and ARPA contract DABT-93-C0057. This research was done when Govindan was a graduate student at Washington University.
presented here shows that the workload distribution over the processors follows the log-normal probability distribution. This analysis is verified by comparing model predicted performance.

Given the workload distribution, the total time per iteration for synchronous iterative algorithms is expressed as the maximum of the execution times on each of the processors. Most probability distributions do not have a closed form solution for the maximum order statistic. Most of the performance models developed in the above papers therefore use asymptotic approximations that are accurate for large number of processors [12, 5]. [11] and [2] derive exact expressions for the maximum terms for relatively simple distributions (triangular and binomial distributions respectively). In our development, we evaluate the maximum order statistic directly using numerical integration techniques.

2 Modeling Workload Dynamics

This paper focuses on a class of applications based on synchronous iterative algorithms. The core computation of a synchronous iterative application consists of a sequence of iterations with a set of variables being evaluated at each iteration as a function of some or all of the values from previous iterations. The variables are distributed over multiple processors, with each processor being responsible for evaluating the variables allocated to it. All processors must complete an iteration and exchange variable values before the next iteration can begin. In some synchronous iterative applications, the distribution of variables to processors may not remain constant during execution. This change in distribution may lead to load imbalance in the system. Since the processors synchronize at the beginning (or end) of each iteration, load imbalance degrades the overall performance.

N-body simulations are an example of a synchronous iterative algorithm and are used in this paper for verifying the models developed. Such simulations study a system of particles where there is a force (e.g., gravitational) between every pair of particles. Each iteration (or time-step) of the simulation consists of computing the resultant force on each particle due to all the other particles and determining the change in its velocity and position due to this force. The procedure is repeated for the next iteration using the new velocity and position of the particles. In a parallel implementation, the entire space is generally divided into smaller domains, with each processor assigned to a domain. The processor is responsible for calculating the forces on the particles present in its domain and updating their positions. As the simulation progresses, the particles move in space crossing processor domain boundaries. This movement of particles, coupled with the dynamics of the any force approximation algorithms employed, results in changing workload distribution at each iteration. Thus the workload distribution at any iteration is a function of the workload at the previous iteration and the movement of particles during that iteration.

Task Model: The computation associated with each iteration of a synchronous iterative algorithm can generally be broken into a large number of tasks. The tasks are allocated to the available processors and communicate with each other at the beginning (or at the end) of each iteration. In the N-body simulation example, the resultant force evaluation on a single particle is defined as a task, with each processor having as many tasks as the number of particles allocated to it. To compute the forces on each particle, the position information of other particles is required. At the beginning of each iteration, the processors exchange position information for the particles allocated to them.

We assume that the total number of tasks in the system remains approximately constant for all iterations and is denoted by $N$. Also, tasks are assumed to have equal processing requirements. There are $p$ identical processors in the system, denoted by $P_1, P_2, \ldots, P_p$. The task distribution at iteration $k$ is denoted by the vector $w(k)$ (i.e., $w_i(k)$ is the number of tasks allocated to processor $P_i$ during iteration $k$).

The aim here is to derive a probability distribution function for the task distribution over the processors at a given iteration. The task distribution at iteration $k$ can be expressed as a function of the task distribution at $k - 1$ by setting $k$ equal to the number of tasks present at the start of iteration $k - 1$, plus the number of tasks that arrive at the processor during iteration $k - 1$, minus the number of tasks that leave the processor during iteration $k - 1$:

$$w_i(k) = w_i(k - 1) + \text{tasks arriving during } k - 1 - \text{tasks departing during } k - 1.$$ (1)

In applications that have a slowly and gradually changing workload, the number of tasks moving between processors in any iteration is typically a small fraction of the number of tasks present in the iteration during that iteration. We model the number of tasks leaving or entering a processor during an iteration as a fraction of the number of tasks in the processor. The number of tasks entering $P_i$ in iteration $k - 1$ is denoted by $\alpha_{k-1} \times w_i(k - 1)$, and the number of tasks leaving the processor by $\beta_{k-1} \times w_i(k - 1)$. $\alpha$ and $\beta$ are random variables whose probability distribution is obtained using empirical data from the application. Equation 1 can thus be written as:

$$w_i(k) = (1 + \alpha_{k-1} - \beta_{k-1}) w_i(k - 1)$$ (2)

We assume that $\alpha$ and $\beta$ are independent and identical for all processors, and iterations. This results in all the processors having identical, independent distributions for each iteration. Since the distribution of $w$, $\alpha$, and $\beta$ are independent of $i$, we drop the $i$ subscript in the following equations. The next section indicates how empirical data was obtained and the distributions for $\alpha$ and $\beta$ were derived for
the N-body application. Define another random variable, \( \gamma_{k-1} = (1 + \alpha_{k-1} - \beta_{k-1}) \). The probability distribution of \( \gamma_{k-1} \) can be derived in terms of the distributions of \( \alpha_{k-1} \) and \( \beta_{k-1} \) and equation 2 can now be expressed as:

\[
w(k) = \gamma_{k-1} w(k - 1)
\] (3)

\( \gamma_{k-1} \) represents the change in the load distribution over iteration \( k - 1 \). Applying the above equation recursively, we can express the task distribution at iteration \( k \) as:

\[
w(k) = (\gamma_{k-1} \times \gamma_{k-2} \times \ldots \gamma_0) \times w(0)
\] (4)

If the workload was balanced initially (at \( k = 0 \)), the number of tasks on all processors is equal (i.e., \( w_i(0) = w_j(0) = N/p, \forall i, j \)). Therefore, from Equation 4, the task distribution at any iteration \( k \) can be represented as the product of \( k \) random variables \( \gamma_0 \ldots \gamma_{k-1} \) and the initial distribution \( N/p \).

Since \( \gamma_0 \ldots \gamma_k \) are all identical and independent, the mean and variance of \( w(k) \) can be easily computed. However, for performance prediction models described in a later section, an expression for the probability distribution of \( w(k) \) is required. Deriving the exact probability distribution for \( w(k) \) can be very involved and not practical since it has to be derived for each \( k \), and involves the product of several random variables for large \( k \). We therefore follow an approach that is a good approximation for relatively large values of \( k \).

Let \( z_k \) be the product of the \( k \) random variables, \( \gamma_{k-1}, \gamma_{k-2}, \ldots \gamma_0 \):

\[
z_k = \gamma_{k-1} \times \gamma_{k-2} \times \ldots \gamma_0
\] (5)

From equation 4 with \( w_i(0) = N/p \), we have,

\[
w(k) = z_k \times N/p
\] (6)

Taking logarithms on both sides of Equation 5, we have,

\[
\log z_k = \log \gamma_{k-1} + \log \gamma_{k-2} + \ldots + \log \gamma_0.
\] (7)

For a reasonably large \( k \), the distribution of \( \log z_k \) approaches a normal distribution. The \( \log \gamma \) random variables also have identical, independent distributions. If \( E[\log \gamma] = \mu \) and \( \text{var}[\log \gamma] = \sigma^2 \), the mean and variance of \( \log z_k \) is given by:

\[
E[\log z_k] = E[\log \gamma_{k-1}] + \ldots + E[\log \gamma_0] = \mu k
\]

\[
\text{var}[\log z_k] = V[\log \gamma_{k-1}] + \ldots + V[\log \gamma_0] = \sigma^2 k
\] (8)

\( \mu \) and \( \sigma^2 \) can be derived from the distribution for \( \gamma \).

Since the probability density function of \( \log z_k \) is normal, it can be expressed as:

\[
f_{\log z_k}(z) = \frac{1}{\sqrt{2\pi \sigma^2 k}} e^{-[z-\mu k]^2/2\sigma^2 k}
\] (9)

\( z_k \) is represented by a log-normal distribution whose density function is derived from the normal distribution of equation 9, and can be written as:

\[
f_{z_k}(z) = \frac{1}{\sqrt{2\pi \sigma^2 k}} e^{-(\log z - \mu k)^2/(2\sigma^2 k)}
\] (10)

The mean and variance of \( z_k \) can be written as follows:

\[
E[z_k] = e^{(\mu k - \sigma^2 k/2)}
\]

\[
\text{var}[z_k] = e^{(2\mu k - \sigma^2 k)}(e^{\sigma^2 k} - 1)
\] (11)

The density function for \( w_k(k) \) is derived from equations 6 and 10 and is expressed as follows:

\[
f_{w_k}(w) = \frac{1}{\sqrt{2\pi \sigma^2 k}} \frac{1}{w} e^{-[\log w/(N/p)-\mu k]^2/(2\sigma^2 k)}
\] (12)

This function is used later to predict the overall performance of the algorithm. Note that distributions for \( \alpha \) and \( \beta \), hence \( \gamma \) and \( \log \gamma \) are obtained from empirical data. However, for all applications where \( z_k \) is expressed as a product of several independent random variables, \( z_k \) has a log-normal distribution. Only the mean and variance of the distribution are computed from empirical data from the application.

**Empirical data:** The N-body simulation was run for several particle configurations (1000 to 4000 particles) and several processor populations (2 to 16) and empirical data was collected to determine the relationship between the number of tasks on a processor \( w(k) \), the number of tasks arriving at a processor during that iteration \( \alpha_k w(k) \), and the number of tasks leaving the processor during the iteration \( \beta_k w(k) \). \( \alpha_k w(k) \) and \( \beta_k w(k) \) together represent the movement of particles across processor domains. Figure 1 shows the frequency diagram for \( \gamma_k \) which is derived from data obtained for \( \alpha \) and \( \beta \). \( \gamma \) denotes the change in task distribution over one iteration \( \gamma_{k-1} w(k - 1) \). The distribution for \( \gamma \) in figure 1 indicates that the change in task distribution over one iteration is small, within \( \pm 0.3 w(k) \).

Using standard statistical parameter estimation techniques[13], the probability density function
of $\alpha$ and $\beta$ were estimated as:

$$f_\alpha(\alpha) = k_\alpha e^{\alpha a}, 0 \leq \alpha < \max$$
$$f_\beta(\beta) = k_\beta e^{\beta b}, 0 \leq \beta < \max$$

(13)

where, $a = b = 31.25$ and $\max = 0.3$. Note that $\alpha$ and $\beta$ do not have to be identically distributed, though they were found to be identical in this example. $k_\alpha$ and $k_\beta$ were computed so that $f_\alpha$ and $f_\beta$ are valid probability density functions and $k_\alpha = k_\beta \approx 31.25$. Since $\gamma$ was defined earlier as $\gamma = (1 + \alpha - \beta)$, the density function for $\gamma$ may be computed as the convolution of the density functions for $\alpha$ and $\beta$ and can be written as:

$$f_\gamma(\gamma) = \left\{ \begin{array}{ll}
  \frac{k_\alpha}{2} e^{-a(1 - \gamma)} (1 - \max) & 0 \leq \gamma < 1 \\
  \frac{k_\beta}{2} e^{-(\gamma - 1)} & 1 \leq \gamma < (1 + \max)
\end{array} \right.$$

(14)

The density function for log $\gamma$ is derived from the density function of $\gamma$ (equation 14), and is expressed as follows (see footnote 2):.

$$f_{\log \gamma}(x) = \left\{ \begin{array}{ll}
  \frac{k_\alpha}{2} e^{x - a(1 - e^x)} \log(1 - \max) & x < 0 \\
  \frac{k_\beta}{2} e^{x} e^{-a(e^x - 1)} & 0 \leq x < \log(1 + \max)
\end{array} \right.$$

(15)

We obtain the mean and variance of log $\gamma$ as: $E[\log \gamma] = \mu = -0.00065$ and $V[\log \gamma] = \sigma^2 = 0.0013$. log $z_k$ is a normal variate with mean $= \mu k$ and variance $= \sigma^2 k$ (equation 8). $z_k$ has a log-normal distribution (equation 9) and the density function of $z_k$ is plotted in Figure 2.

$F_{\gamma}$ takes different shapes for different values of $\mu k$ and $\sigma^2 k$. As seen in figure 2, the distribution is skewed to the right, with the tail more pronounced as $\sigma^2 k$ increases.

The mean and variance of $z_k$ was given in equation 11. From our empirical data, we have $\mu = 0.5$ and $\sigma^2 / 2 = 0$, therefore the mean $(E[z_k])$ is always equal to 1 for all $k$. Since $w(k) = z_k \times N/p$ denotes the number of tasks on each processor, the average number of tasks per processor is equal to $N/p$ for all values of $k$. Also, the load distribution is likely to be more spread out for large $k$, as seen in the figure.

3 Performance prediction

In this section, expressions for the execution time per iteration are developed. The time per iteration on each processor can be modeled as the sum of two components. The first, $T_{\text{vary}}$, is sensitive to the task distributions (derived in the prior section), and the second, $T_{\text{other}}$, is an overhead time which is independent of the task distributions and iteration, and is almost same on all the processors. Due to synchronization at each iteration, the total iteration time is the maximum of the iteration times on all processors and can be written as:

$$t_{\text{iter}}(k) = \max_{i \in [1,p]} [T_{\text{vary},i}(k) + T_{\text{other},i}(k)]$$

(16)

$T_{\text{vary},i}(k)$ and $T_{\text{other},i}(k)$ are the corresponding times on $P_i$ for iteration $k$. For example, in the N-body application, the force evaluation time is very sensitive to the task (particle) distribution, and the other components (associated with creating the necessary data structures) are smaller and insensitive to variations in task distribution.

All the tasks are identical and the time taken by each task for the $T_{\text{vary}}$ phase is the same (denoted by $T_{\text{each}}$). Since the work is proportional to the number of tasks, $T_{\text{vary}}$ can be written as:

$$T_{\text{vary},i}(k) = T_{\text{each}} \times w_i(k).$$

(17)

The number of tasks on processor $P_i$, $w_i(k)$ is given by $z_k \times N/p$, where $z_k$ is a random variable defined in the previous section. The total iteration time can now be written as:

$$t_{\text{iter}}(k) = T_{\text{each}} \times \frac{N}{p} \max_{i \in [1,p]} \left[ z_k^{(1)} + z_k^{(2)} + \cdots + z_k^{(p)} \right] + T_{\text{other}}(k)$$

(18)

where, $z_k^{(i)}$ denote independent instances of $z_k$, corresponding to each processor. To evaluate the max term we define the random variable, $y_k$ as follows:

$$y_k = \max(z_k^{(1)}, z_k^{(2)}, \cdots, z_k^{(p)})$$

(19)

$y_k$ is a measure of the load imbalance in the system, and is always greater than or equal to 1. Note that $y_k$ can also be written as the ratio of the maximum time for $T_{\text{vary},i}(k)$ over all processors and the average time as follows:

$$y_k = \frac{\max_{i \in [1,p]} [T_{\text{vary},i}(k)]}{T_{\text{each}} \times \frac{N}{p}} = \frac{\max_{i \in [1,p]} [w_i(k)]}{\frac{N}{p}}$$

(20)

$y_k = 1$ implies that there is no imbalance in the system, and that all the processors have equal number of tasks. Since $y_k$ is the maximum of $p$ independent, identically distributed random variables, its density function is given by[10]:

$$f_{y_k}(y) = p[F_{y_k}(y)]^{p-1} f_{y_k}(y)$$

(21)
where $f_{z_k}$ is the density function and $F_{z_k}$ is the cumulative distribution function for $z_k$ ($F_{z_k}(y) = \int_{-\infty}^{y} f_{z_k}(z) \, dz$). The mean value of $y_k$ is computed as:

$$E[y_k] = \int_{-\infty}^{\infty} y f_{y_k}(y) \, dy$$  

(22)

Deriving an exact expression for $E[y_k]$ may not be possible for all density functions. In particular, no closed form solution can be obtained when $z_k$ is a log-normal variate.

Several asymptotic approximate techniques exist[8] but are valid only as $p \rightarrow \infty$. Therefore, for $2 \leq p \leq 16$, we evaluate equations 22 and 21 directly using numerical integration techniques. From equations 18 and 22, the expected time for iteration $k$ is calculated as:

$$t_{\text{iter}}(k) = t_{\text{eval}} \times N/p \times E[y_k] + T_{\text{other}}(k)$$  

(23)

To complete the performance model and evaluate $t_{\text{iter}}$, expressions for $t_{\text{eval}}$ and $T_{\text{other}}$ are required. These expressions were developed for the Barnes-Hut N-body simulation algorithm[4] employed. They are presented in detail in [6] and are summarized below. $t_{\text{eval}}$ comprises the force evaluation phase for each iteration associated is given by:

$$t_{\text{eval}} = k_{\text{eval}} \log_2 N \times t_{\text{eval}}$$  

(24)

The $\log_2 N$ term is the result of using the Barnes-Hut algorithm. $t_{\text{eval}}$ and $k_{\text{eval}}$ (0.052ms and 5.2 respectively) are measured values relating to the time associated with force calculations for each iteration.

$T_{\text{other}}$ comprises all other calculations for an iteration and includes such items as communications time, updating various data structures etc. and is given by:

$$T_{\text{other},i}(k) = T_{\text{local},i}(k) + T_{\text{comm},i}(k) + T_{\text{update},i}(k)$$

$$T_{\text{local},i}(k) = \sum_{j=1}^{k} [2^{j-1} t_{\text{sort}} N/p \log_2(N/p) + ]$$

$$T_{\text{comm},i}(k) = \sum_{j=1}^{k} [2^{j} N/p + t_{\text{merge}} 2^{j} N/p + t_{\text{msg}}(2) \times n_{\text{byte}}]$$

$$T_{\text{update},i}(k) = t_{\text{update}} N/p + t_{\text{book}}$$

where, measurements yielded $t_{\text{sort}} = 0.004 ms, t_{\text{search}} = 0.05 ms, t_{\text{merge}} = 0.05 ms, t_{\text{update}} = 0.02 ms, t_{\text{book}} = 10 ms, and T_{\text{msg}}(s) = 1.6 [s_{\text{msg}1} + s_{\text{msg}2} + s_{\text{msg}3} + s_{\text{msg}4}]$.

Substituting these expressions back into the model for the iteration time (equation 23) yields the overall performance model. Verifying the task distribution part of the model is considered in the next section.

### 4 Model Verification and Results

In this section, we present experimental results from the N-body application to validate our model predictions. We first show how the task distribution model developed in section 2 is validated. Then model predictions for the overall performance are compared with measured values.

The computing platform used in this experiment is a SparcStation 2000, a shared memory machine with up to 20 identical processors. The application was executed under the PVM[3] environment with the computational tasks viewing the machine as a network of workstations communicating with each other via messages. A problem size of $N = 2000$ was used, with the number of processors, $p \in \{1, 2, 4, 8, 16\}$.

**Verifying the task distribution model:** The task distribution model indicates that for relatively large $k (k \geq 10)$, the task distribution on each processor approaches a log-normal distribution. To verify this, the measured task distribution data from several application runs for various values of $k$ were compared against the log-normal distribution. Figure 3 shows the cumulative distribution function ($F_{z_k}(z)$) for $z_k$ at $k = 10, 100, 500, 1000$ for both empirical data and the log-normal distribution. The empirical distribution is calculated using over 300 samples. In addition, the 95% confidence band for the empirical distribution derived from the Kolmogorov-Smirnov (KS) statistic[13] is also shown.

The log-normal distribution agrees well with the empirical data for all values of $k$, and is within the 95% confidence band. The difference in model and empirical distribution is mainly due to errors in approximating the sum of $k \log z_k$ variables by a normal random variable. The approximation is more accurate for large $k$; model and measured values agree better for large $k (k = 1000)$.

For small $k$, the load imbalance is small and most $z_k$ values are bunched close to 1. This is seen as a steep gradient in $F_{z_k}(z)$ around 1 for $k = 10$. Such a result is to be expected since, after only a few iterations, the system is still near its initial perfect load balance (equation 6).

The gradient is less steep as $k$ increases and the value of $z_k$ is spread over a larger range around 1 (although the mean is always 1 for this example). This indicates that the variance in task distribution is increasing as the number of iterations increase (see equation 11).

**Comparing Overall Performance:** In section 3, we developed a model for the overall performance of a synchronous iterative algorithm using the task distribution model derived in section 2. In this section, the model predictions are compared with measured data.

We first look at the model and measured imbalance values and then compare the model and measured execution times. The measured imbalance in task distribution, $y_k$, is the maximum workload on any processor divided by the average workload over all processors (equation 20). The model estimates for load imbalance are evaluated assuming a log-normal task distribution and are compared with the measured imbalance. Figure 4 shows the measured and pre-
Cummulative Distribution: $F(z) \rightarrow \text{error} = 10\%$ for $p$.

However, since sum of the number of tasks on all processors are identical and distribution over all processors (equation 19) assuming that the processor times are shown for various values of $p$.

Equations 24 and 25 for N-body simulation. The execution predicted execution time is computed using equation 18 and most sensitive to load imbalance in the system. The model predicts iteration time and the force computation phase (which is most sensitive to load imbalance in the system). The model predictions agree better with measured values for the 1000 iterations case due to the approximations (log-normal task distribution) made in deriving the model. Also, the model predictions agree better with measured values for large $p$.

The performance degradation due to load imbalance is significant even for 100 iterations.

Figure 6 shows the speedup obtained with $p$ processors, defined in the standard manner as the ratio of execution time on one processor over the execution time on $p$ processors. Speedups averaged over the first 100 iterations and over 1000 iterations are shown. The speedup obtained for the ideally balanced case is also shown. Speedup for the ideal case is less than linear due to the communications costs. Speedups over the first 100 iterations are much higher than the speedup over 1000 iterations. This is because the load imbalance becomes higher for larger $k$, leading to lower performance. However, the error between measured and model predicted speedups is higher for the 100 iterations case than for the 1000 iterations case due to the approximations (log-normal task distribution) made in deriving the model. Also, the model predictions agree better with measured values for large $p$ than for smaller systems. This is because the assumption that the processor task distributions are independent is more accurate for large $p$. The performance degradation due to load imbalance is significant even for 100 iterations.
ations. The speedup obtained for 100 iterations is less than half the speedup expected for ideal load balance. The performance for 1000 is even smaller than the 100 iterations case.

![Figure 5. Comparing predicted and measured execution time](image)

5 Conclusions

In this paper, models were developed to express the variations in workload distribution, and to predict the overall performance for a class of parallel synchronous iterative applications. The model was verified with measured results from N-body simulation application. Both model and measured results show that the load imbalance in these applications significantly degrades performance. For example, the speedup obtained for 100 iterations is about half the speedup expected with balanced task distribution. Dynamic load balancing is necessary to obtain good performance in such applications. Models such as those developed here are very useful in developing accurate performance prediction schemes, and in designing and evaluating load balancing schemes. For example, the expressions for task arrival and departure rates derived here have been used as inputs to a dynamic load balancing model developed in [7].

References