ICOT Technical Report: TR-0814

TR-0814

Probabilistic Analysis of the Optimal Efficiency of the Multi-Level Dynamic Load Balancing Scheme

by
K. Kimura & N. Ichiyoshi

October, 1992

ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan

(03)3456-3191 ~ 5
Telex ICOT J32964

Institute for New Generation Computer Technology

Probabilistic Analysis of the Optimal Efficiency of the Multi-Level Dynamic Load Balancing Scheme

Kouichi Kimura Nobuyuki Ichiyoshi

Institute for New Generation Computer Technology
1-4-28 Mita, Minato-ku, Tokyo 108, Japan

Abstract

This paper investigates the optimal efficiency of the multi-level dynamic load balancing scheme for OR-parallel programs, using probability theory.

In *the single-level dynamic load balancing scheme*, one processor divides a given task into a number of subtasks, which are distributed to other processors on demand and then executed independently. We introduce a formal model of the execution as a queuing system with several servers. And we investigate the optimal granularity of the subtasks to attain the maximal efficiency, taking account of dividing costs and load imbalance between the processors. Thus we obtain estimates of the maximal efficiency.

We then apply these results to analysis of the efficiency of *the multi-level dynamic load balancing scheme*, which is the iterated application of the single-level scheme in a hierarchical manner. And we show how the scalability is thereby improved over the single-level scheme.

Key words. parallel processing, load balancing, efficiency, scalability, isoefficiency, queuing theory.

1 Introduction

The purpose of parallel processing is to accelerate the execution of time-consuming tasks by utilizing a number of processors. *Efficiency*, defined by the speed-up divided by the number of processors, indicates the performance of parallel processing. It depends not only on the algorithm itself, but also on the number of processors, the problem size (the amount of computation required by the best sequential algorithm for solving it), and other factors. In general, for a given task, the efficiency decreases as the number of processors increases, as Amdahl's law illustrates. Therefore it is important to analyze how the efficiency depends on these factors. In particular, as increasingly larger-scale multiprocessors are now being developed [9], the scalability analysis of the efficiency is becoming more and more important.

Various measures of the scalability of a parallel algorithm have been proposed for different situations [6]. Among these, the notion of "isoefficiency" [5] succinctly captures the characteristics of scalability of a parallel algorithm. The efficiency usually decreases with an increasing number of processors, but recovers again with a larger problem. *Isoefficiency function* indicates how much the problem size should be increased with the number of processors so as to maintain a constant efficiency. A parallel algorithm with a lower order isoefficiency function is supposed to be more scalable.

Ideally the efficiency should be one, however, it may deteriorate due to various reasons: the load imbalance between the processors, inter-processor communication latency, speculative computations, or other overheads associated with parallel execution. In particular, if the algorithm is composed of many parts requiring unpredictable amounts of computation, as is usual with many combinatorial search problems, load imbalance between the processors is likely to occur and may have a great influence on the efficiency. Thus the load balancing is one of the central issues of parallel processing.

Furuichi *et al.* [2] proposed *the multi-level dynamic load balancing scheme* for OR-parallel programs, and evaluated its performance using the Multi-PSI, a distributed memory MIMD machine with 64 processors. Their basic strategy is to divide a given problem into mutually independent subtasks, and distribute them to other processors on demand. This on-demand distribution balances the load between the processors. However, when the number of processors increases, the number of subtasks should increase correspondingly. Therefore, if the dividing is entrusted to one proces-

sor (*the single-level dynamic load balancing scheme*), it will become a bottleneck. So they proposed to divide the problem iteratively in a hierarchical manner (*the multi-level dynamic load balancing scheme*). Their experiments show that the latter is in fact more "scalable" than the former.

The purpose of this paper is to theoretically investigate the optimal efficiency of these dynamic load balancing schemes. We define a formal model of the single-level dynamic load balancing as a queuing system with several servers. The unpredictable amount of computation required by each subtask is probabilistically treated. The results on the single level scheme are then applied to the analysis of the multi-level scheme. Among others, we show:

1. With subtasks of random sizes, the optimal efficiency is worse than with subtasks of exactly the same size. The isoefficiency function in a typical former case is $\log p$ times larger than that in the latter case, where p is the number of processors.

2. The multi-level load balancing scheme is indeed more "scalable" than the single-level one. The isoefficiency function for the former has a smaller fractional order of p than the latter.

3. In the tree configuration of the processors in the multi-level dynamic load balancing scheme, a processor at a higher level should have a larger fan-out degree than one at a lower level. The order of their ratio is a fractional power of $\log p$. In particular, the uniform tree configuration with the same degree at all levels is *not* optimal.

For conciseness, we make a compromise in rigidity and give only intuitive proofs in this paper. A rigorous treatment will appear in [4].

2 Dynamic Load Balancing Scheme

In this section we describe the multi-level dynamic load balancing scheme, proposed by Furuichi *et al.* [2]. It can be applied to many parallel programs on MIMD machines.

2.1 Assumptions for problems

We assume that our target problem can be divided into many subproblems such that:

(i) The subproblems can be solved independently of one another.

(ii) The amount of computation required by each subproblem is unpredictable before it is solved.

These assumptions seem natural in the OR-parallel exhaustive search procedures for many combinatorial problems. For instance, consider an exhaustive search of a tree, which represents the search space of a combinatorial problem. This tree can be divided into many subtrees at any depth d, and the search of the entire tree will be reduced to the searches of these subtrees. The latter searches will be independent of one another, as long as we don't employ any special pruning strategies. And, as is inherent in combinatorial problems, the entire tree will be irregular and the size of each subtree will be unpredictable before we search it. Thus both assumptions are satisfied.

Now, in general, assumption (i) implies that the problem can be solved efficiently in parallel: — different processors solve different subproblems simultaneously. However, assumption (ii) makes it difficult to *statically* balance the load between the processors. This prompts us to employ a *dynamic* load balancing strategy.

2.2 Single-level load balancing scheme

In the following, we will refer to the problem to be solved as a *task*, and similarly, to a subproblem as a subtask.

We consider here one of the most naïve on-demand load distribution techniques: given a task, one *producer* processor divides it into a number of mutually independent subtasks, which are transmitted to the *consumer* processors on demand and then executed.

This on-demand load distribution will balance the load between the processors. We refer to this load balancing strategy as *the single-level dynamic load balancing scheme* [2].

2.3 Tuning the granularity

In order to make this load balancing scheme work efficiently, we have to tune the granularity of the subtasks. With a few large subtasks, load imbalance between the consumer processors is likely to occur. As an extreme case, if the number of subtasks is less than the number of consumer processors, some of the processors will never be used. On the other hand, with many small subtasks, the producer is likely to become a bottleneck.

So we assume that:

(iii) The granularity of the subtasks can be controlled.

and we will optimize it. Namely, we assume that the task can be divided into more subtasks of smaller sizes or less subtasks of larger sizes, at will.

For example, let us consider the OR-parallel exhaustive search of a tree again. Each subtask is the search of a subtree with its root at depth d. So by choosing an appropriate depth d, we can control the granularity of the subtasks.

2.4 Bottleneck in speedup

The single-level dynamic load balancing scheme has an apparent drawback: it does not scale.

Suppose that the number of processors is increased. Then, as we just saw in the last subsection, the granularity of the subtasks should be tuned accordingly. In this case, we have to increase the number of subtasks (to a certain extent larger than the number of the consumer processors). But if we increase the number of subtasks, the producer will become a bottleneck, since it is in charge of producing all of these subtasks. Thus the efficiency will inevitably drop.

So, in order to improve the scalability, we should remove such a producer bottleneck.

2.5 Multi-level load balancing scheme

The *multi-level dynamic load balancing scheme* alleviates the producer bottleneck by hierarchical load distribution.

In the *2-level dynamic load balancing scheme*, we divide a given task at *a root producer* into many subtasks. They are distributed to the *second level producers* on demand, and then are divided again into smaller *subsubtasks*. These subsubtasks are further distributed to the *leaf consumers* on demand, and are finally carried out.

Schemes of more than two levels can be defined similarly. By increasing the number of levels, we can improve the scalability as we will see later.

3 Model of Dynamic Load Balancing

We introduce a formal model of the single-level dynamic load distribution. The aim of our analysis is to study how the efficiency is deteriorated due to the load imbalance.

3.1 Model of single-level dynamic load balancing

The dynamic behavior of the on-demand load distribution can be naturally expressed as a *queueing system with several servers* — customers arrive one after another and wait in a queue for service at any one of the counters.

Here a subtask corresponds to a customer in the queueing system, and a consumer processor corresponds to a server. Hence the production of a subtask corresponds to the arrival of a customer and the execution of a subtask corresponds to the service to a customer.

Motivated by this analogy, we define the parallel execution time by the single-level dynamic load balancing scheme as follows.

Definition 1 *Let N be the number of subtasks and p be the number of consumer processors. We define the size of a subtask as the CPU time required for executing it. We denote the size of the n-th subtask by R_n, and the CPU time required for producing it by U_n. We define the parallel execution time T_p as:*

$$T_p = \max_{1 \le n \le N} Y_n$$

where Y_n as well as O_n, X_n and Z_n are defined as follows by induction on n.

$$O_n = \sum_{k=1}^{n} U_k \quad (1 \le n \le N)$$

$$X_n = \max(O_n, Z_n) \quad (1 \le n \le N)$$

$$Y_n = X_n + R_n \quad (1 \le n \le N)$$

$$Z_n = \begin{cases} \min\limits_{1 \le i_1 < \cdots < i_{p-1} < n} \ \max\limits_{\substack{1 \le k < n \\ k \ne i_1, \ldots, i_{p-1}}} Y_k & (p < n \le N) \\ 0 & (1 \le n \le p) \end{cases}$$

Here O_n represents the creation time of the n-th subtask, X_n (Y_n) represents the starting (ending) time of its execution, and Z_n represents the time when at least one of the consumers becomes ready to execute the n-th subtask.

Here we ignored inter-processor communication latency and assumed that a subtask is immediately transmitted from the producer to a consumer when demanded (at time Z_n) and available (at time O_n). According to assumption (i) in Section 2.1, we assumed that no suspension occurs in executing each subtask and that Y_n is simply given by the sum of X_n and R_n. Note that Z_n depends only on $Y_1, \ldots, Y_{n-1}$. Hence X_n, Y_n and Z_n are determined by induction on n.

This model can be regarded as a queuing system with p servers. R_n represents the service time for the n-th customer and U_n represents the interval of arrival between the $(n-1)$-th and n-th customers.

3.2 Basic assumptions

In terms of assumption (ii) in Section 2.1, the exact values of N, R_n and U_n defined above will not be known beforehand. One of the worthwhile approaches is the probabilistic analysis of algorithms initiated by R. Karp [3] and others.

We suppose that a problem is given randomly from a set of similar problems (*a problem space*) and we will engage in an average case analysis. Here N, R_n and U_n as well as T_p are regarded as random variables.

The size of a task (problem) is defined as the CPU time required for executing it using one processor. It is also a random variable, and we denote it by T_1 and its expectation by

$$t_1 = E(T_1) : \text{ average task size}$$

We assume that there is a family of problem spaces of different average task sizes. For example, a family corresponds to a general question — *e.g.*, color the vertices of a graph so as to satisfy a certain condition. Each problem space consists of problem instances with the same description length — *e.g.*, graphs with the same number of edges. Problem instances with a larger description length will require a larger amount of computation on average. Namely, they will have a larger average task size.

In terms of assumption (iii) in Section 2.3, we assume that we can control the granularity of the subtasks in the average sense. Namely, we assume that we can control

$$\nu = E(N) : \text{ average number of subtasks}$$

but not N itself.

And we assume that the subtasks are probabilistically equivalent and that there is no correlation between the subtask sizes. Namely,

$$\{R_n\}_{n=1,2,\ldots} : \text{ i.i.d.}$$

$$\{U_n\}_{n=1,2,\ldots} : \text{ i.i.d.}$$

(i.i.d. stands for *independent identically distributed*.)

If the subtask sizes were not probabilistically equivalent, namely, if we knew that some of the subtasks were expected to be larger than the others, they should be distributed in a different manner. If there was a correlation between the subtask sizes, for example, if we could predict the size of a subtask based on that of another, we could balance the load better based on such prediction. However, we will not discuss such cases here. They will suggest other load distribution strategies, which might be complicated, strongly problem-dependent and hard to analyze in a general setting.

We define several other characteristic values:

$$\frac{1}{\lambda} = E(U_n) : \text{ average time for producing a subtask}$$

$$\frac{1}{\mu} - \frac{t_1}{\nu} = E(R_n) : \text{ average subtask size}$$

In other words, λ represents the *production rate* of the subtasks and μ represents the *consumption rate* of the subtasks at each consumer processor. We denote the ratio of the production rate to the overall consumption rate of the subtasks by:

$$\rho = \frac{\lambda}{\mu p}$$

A small value of ρ implies fine-grained granularity, while a large value of ρ implies coarse-grained granularity. Later we will see that ρ is a normalized indicator of the granularity of the subtasks.

We assume that the total amount of the subtask sizes is equal to the size of the whole task:

$$R_1 + \cdots + R_N = T_1$$

and regard U_n as the overheads associated with parallel processing. We assume that these overheads are moderate in the following sense. Informally, producing a subtask is just computing an "address" which specifies the portion of the search space assigned to it. We assume that this address should be computed in a reasonable amount of time — in a polynomial time of $\log p$, the description length of p. Namely, we assume:

$$\frac{1}{\lambda} = O((\log p)^k) \text{ for some } k \geq 0$$

For example, in the exhaustive tree search, a subtask, *i.e.*, a subtree with its root at depth d, can be specified by a path of length d from the root of the entire tree to its own root. If the nodes of the tree have bounded degrees, the "address" of a subtask can be written down in $O(d)$ time. We should choose $d = O(\log p)$ in order to produce a polynomial number (in p) of subtasks. Later we will see that the optimal ν is in fact polynomially bounded in p (Corollary 1).

We denote the *mean execution time* by $t_p = E(T_p)$ and adopt as a proper definition of the *mean speed-up*:

$$s_p = \frac{t_1}{t_p} : \text{ "collective" mean speed-up}$$

instead of the *arithmetic* mean speed-up $E(T_1/T_p)$. A "collective" mean represents a *weighted* mean of ratios

according to their denominators and is calculated by dividing the sum of their numerators by the sum of their denominators. For instance, suppose that a task is chosen at random uniformly from the problem space with $T_1 \in \{t_1^{(1)}, \ldots, t_1^{(I)}\}$ and $T_p \in \{t_p^{(1)}, \ldots, t_p^{(I)}\}$. Then

$$s_p = \frac{t_1^{(1)} + \cdots + t_1^{(I)}}{t_p^{(1)} + \cdots + t_p^{(I)}} = \sum_{i=1}^{I} \frac{t_p^{(i)}}{t_p^{(1)} + \cdots + t_p^{(I)}} \cdot \frac{t_1^{(i)}}{t_p^{(i)}}$$

where $t_1^{(i)}/t_p^{(i)}$ is the speed-up for the i-th problem instance and is weighted in proportion to its parallel execution time $t_p^{(i)}$.

Finally, we define:

$$\eta = \frac{s_p}{p} \quad : \quad \text{mean efficiency}$$

$$\eta_{\max} = \sup_{p} \eta(p, t_1, \nu) \quad : \quad \text{maximal mean efficiency}$$

The *maximal mean efficiency* $\eta_{\max}$ is the mean efficiency when we choose the optimal granularity for given p and t_1.

4 Analysis of the Single-Level Dynamic Load Balancing

4.1 Deterministic case

Firstly we study a special case that corresponds to uniform programs, in which we know the exact size of a given task and can divide it into an arbitrary number of subtasks of exactly the same size in a constant time for each:

$$T_1 \equiv t_1 \ ,$$

$$N \equiv \nu, \quad R_n \equiv \frac{1}{\mu} = \frac{t_1}{\nu}, \quad U_n \equiv \frac{1}{\lambda} \quad \text{for } \forall \nu = 1, 2, \ldots$$

We refer to such a special case as *the deterministic case*, since everything is deterministic here, *i.e.*, all random variables are distributed according to δ-distributions. In this case, optimizing the granularity, we obtain the following estimate of the efficiency.

Proposition 1 (deterministic case)

$$\frac{1}{\eta_{\max}} \simeq 1 + \frac{p^2}{\lambda t_1}$$

Here $\simeq$ means that the left-hand side and the right-hand side differ only in a lower order term as $p \to \infty$.

INTUITIVE PROOF: The producer will become a bottleneck and hence the efficiency will be deteriorated, when and only when $\rho < 1$ [8]. So we assume $\rho \geq 1$. Since each of the p consumer processors completes μ subtasks per unit time, the execution of the last subtask will start at time:

$$X_N \simeq \frac{\nu - 1}{\mu p}$$

And since the last subtask will require $1/\mu$ computation time,

$$T_p \simeq X_N + \frac{1}{\mu} \simeq \frac{\nu - 1}{\mu p} + \frac{1}{\mu} \simeq \frac{t_1}{p} + \frac{\rho p}{\lambda}$$

$$\frac{1}{\eta} = \frac{p t_p}{t_1} \simeq 1 + \frac{\rho p^2}{\lambda t_1}$$

Choosing $\rho \simeq 1$, we obtain the desired result. (Note that ρ may not be exactly one since ν takes only discrete values.) ∎

Here the optimal granularity of the subtasks is characterized by $\rho \simeq 1$, which implies that the production rate and the consumption rate of the subtasks should be balanced. This is intuitively convincing.

4.2 Exponential case

Next we study another special case, *the exponential case*, in which the task (subtask) size is distributed according to an exponential distribution, and the number of subtasks is distributed according to a geometric distribution:

$$P(x < T_1 \leq x + dx) = \frac{1}{t_1} \exp(-\frac{x}{t_1}) dx \quad \text{for } \forall x \geq 0$$

$$P(x < R_n \leq x + dx) = \mu e^{-\mu x} dx \quad \text{for } \forall x \geq 0, \ \forall n$$

$$P(N = n) = \frac{1}{\nu} \left(1 - \frac{1}{\nu}\right)^{n-1} \quad \text{for } \forall n = 1, 2, 3, \ldots .$$

Note that the last two equations imply the first one. Hence this is a consistent assumption.

For example, if a subtask consists of a simple loop and there is a constant probability of termination at each iteration, the size of a subtask is distributed exponentially. Such a case is referred to as *Markov service time* in queueing theory.

Theorem 1 (exponential case)

(i) *For* $\forall \rho > 1$, $\quad \frac{1}{\eta} \simeq 1 + \frac{\rho p^2}{\lambda t_1} \cdot \log p$

(ii) *For* $0 \leq \forall \rho \leq 1$, $\quad \frac{1}{\eta} \gtrsim 1 + \frac{p^2}{\lambda t_1} \cdot \log p$

INTUITIVE PROOF: (i) Since $\rho > 1$, the starting time of executing the last subtask is the same as in the proof of Proposition 1. We may assume that all the p consumer processors are busy at this time, otherwise the "producer bottleneck" must have occurred in spite of $\rho > 1$. Hence, from the following Lemma 1,

$$(\text{remaining execution time}) \simeq \frac{1}{\mu} \cdot \log p$$

Therefore,

$$t_p \simeq \frac{\nu - 1}{\mu p} + \frac{1}{\mu} \cdot \log p \simeq \frac{t_1}{p} + \frac{\rho p}{\lambda} \cdot \log p$$

$$\frac{1}{\eta} = \frac{\rho t_p}{t_1} \simeq 1 + \frac{\rho p^2}{\lambda t_1} \cdot \log p$$

(ii) can be proved similarly and we omit the details. ∎

Lemma 1 *Let $X, X_1, \ldots, X_p$ be i.i.d. according to the exponential distribution with mean σ:*

$$P(x < X \leq x + dx) = \frac{1}{\sigma} \exp(-\frac{x}{\sigma})dx \quad \text{for } \forall x \geq 0$$

Then, for $\forall\, a \geq 0$,

$$E(X - a \mid X > a) = \sigma$$

$$E((X - a)^2 \mid X \geq a) = 2\sigma^2$$

$$E(\max_{1 \leq i \leq p} X_i) \simeq \sigma \cdot \log p \qquad (1)$$

PROOF: The first two claims can be verified directly. The proof of the last claim is sketched in Appendix. ∎

Now, compare the result in the deterministic case with that in the exponential case. An extra factor of $\log p$ appears in the latter, which stems from the load imbalance between the consumers due to the diversified subtask sizes. (In the former case, all the subtasks have the same size.) Thus the scalability in the latter case is poorer by this factor.

4.3 General case

Now let us consider a general case. However, in order to ensure reasonable efficiency, we need to make some more assumptions.

Remember that once a subtask is entrusted to one consumer processor it is never distributed to or shared with other processors. So a subtask should be "steadily" executed. To be more precise, the expectation of the remaining execution time (*life expectancy*) of a subtask should never blow up in the following sense.

(Moderate diversity in the subtask size) *For $\forall n = 1, 2, \ldots$ and $\forall a \geq 0$,*

$$E(R_n - a \mid R_n \geq a) \leq \frac{1}{\mu} \cdot (1 + O(\frac{1}{1 + |\log \mu|}))$$

$$E((R_n - a)^2 \mid R_n \geq a) = O(\frac{1}{\mu^2})$$

as $p \to \infty$ and $\mu \to 0$. Here $E(X|C)$ represents the conditional expectation of X under condition C.

As for the first inequality, it is sufficient to assume that the life expectancy of a subtask ($E(R_n - a \mid R_n \geq a)$) should never exceed its initial life expectancy ($1/\mu$). For example, in the deterministic case, the life expectancy decreases by the amount of elapsed time a. This is the "steadiest" case. In the exponential case, the life expectancy remains constant and never blows up (Lemma 1). This is the marginal case.

On the contrary, when the size of a subtask is distributed too divergently, this condition may be violated and the efficiency may be poor. For example, suppose that most of the subtasks are small (of size a) and only a few are exceptionally large (of size $A \gg a$). Then the few subtasks that have survived the initial small period (a) will have a much longer life expectancy ($A - a$) than the overall initial life expectancy ($\approx a$). They will incur a fatal load imbalance and the efficiency will be damaged. A remedy for such cases might be to subdivide the subtasks that have been found to be exceptionally large. Such a load balancing scheme is beyond the scope of discussion here.

Thus, in order to ensure reasonable efficiency, we should assume that the subtask sizes are not too diverse. The above assumption in terms of life expectancy indirectly limits the diversity of the subtask size. We can show that the scalability in this general case is not worse than the exponential case:

Theorem 2 (general case)

$$\forall \rho > 1 \quad \exists c > 0 \quad s.t. \quad \frac{1}{\eta} \leq 1 + \frac{\rho p^2}{\lambda t_1} \cdot (\log p + c)$$

INTUITIVE PROOF: According to Lemma 1, we may regard the exponential case as one of the worst cases under our assumption. Hence the efficiency in the general case would not be worse than the exponential case, for which Theorem 1 provides the desirable bound. ∎

A rigorous proof of this theorem is given in [4].

Corollary 1 (isoefficiency function for the single-level scheme) *Let ρ and c be as above. For*

$0 < \forall \epsilon < 1$, *if we have a task large enough that:*

$$t_1 = \frac{1}{\epsilon} \cdot \frac{\rho p^2}{\lambda} \cdot (\log p + c)$$

then by choosing ν *as*

$$\nu = \frac{1}{\epsilon} \cdot p \cdot (\log p + c) \quad \left(\Leftrightarrow \frac{1}{\mu} = \frac{\rho p}{\lambda} \right)$$

we can maintain the efficiency as: $\eta \geq 1 - \epsilon$

5 Isoefficiency Analysis of the Multi-Level Dynamic Load Balancing

The multi-level dynamic load balancing scheme is an iterated application of the single-level scheme in a hierarchical manner (Section 2.5). In our terminology, it is defined as follows by induction on the number of levels ℓ.

1-level dynamic load balancing is nothing but the single-level dynamic load balancing. For $\ell \geq 1$, the $(\ell + 1)$-level dynamic load balancing is the single-level dynamic load balancing with each consumer substituted by a number of processors, which execute each subtask using the ℓ-level dynamic load balancing scheme. We assume that the production rate λ is common to all levels.

We now apply the results in Section 4 to the isoefficiency analysis of the multi-level load balancing.

We begin with a remark on the single-level dynamic load balancing. In addition to the previous assumptions, we assume that N, the number of the subtasks, should also be moderately diverse. Namely, for $\forall n = 1, 2, \ldots$,

$$E(N - n \mid N \geq n) \leq \nu \cdot (1 + O(\frac{1}{\log \nu}))$$

$$E((N - n)^2 \mid N \geq n) = O(\nu^2)$$

Then it can be shown that both the total task size T_1 and the parallel execution time T_p also have the same property [4]. Thus we can apply the preceding analysis iteratively to the multi-level load balancing and obtain the following results.

Theorem 3 (isoefficiency function for the multi-level scheme) *With the ℓ-level scheme, for arbitrary $\epsilon > 0$, there exists $c > 0$ such that*

$$t_1 \geq \frac{c}{\lambda} \cdot p^{\frac{\ell+1}{\ell}} \cdot (\log p)^{\frac{\ell+1}{2}} \implies \eta_{\max} \geq 1 - \epsilon$$

for infinitely many p, where p is the number of consumer processors.

INTUITIVE PROOF: Let $\tau_\ell(p)$ denote the isoefficiency function of the ℓ-level dynamic load balancing: — the average size of the task for which the ℓ-level dynamic load balancing is expected to work with efficiency of at least η_0 using p consumer processors, where η_0 is an arbitrarily given positive constant less than 1.

We use induction. Assume that

$$\tau_\ell(p_2) \sim \frac{1}{\lambda} \cdot p_2^{\alpha_\ell} \cdot (\log p_2)^{\beta_\ell}$$

for some $\ell \geq 1$ and $p_2 > 1$. Here $\sim$ shows that the left-hand side and the right-hand side have equivalent magnitude as $p \to +\infty$. Note that Corollary 1 implies this for $\ell = 1$ with $\alpha_1 = 2$, $\beta_1 = 1$.

Regard the $(\ell + 1)$-level dynamic load balancing as being composed of the root producer and p_1 virtual "consumers", each of which is in fact composed of a number of processors using the ℓ-level dynamic load balancing scheme with real p_2 consumers, where $p = p_1 \cdot p_2$. Then the overall efficiency is maintained if and only if both the single-level dynamic load balancing at the root producer and the ℓ-level dynamic load balancing inside each "consumer" maintain their efficiency.

Corollary 1 gives the condition for the former:

$$\tau_{\ell+1}(p) \sim \frac{1}{\lambda} \cdot p_1^2 \cdot \log p_1 \cdot p_2$$

$$\frac{1}{\mu_1} \sim \frac{p_1}{\lambda} \cdot p_2 \tag{2}$$

where $1/\mu_1$ denotes the average subtask size carried out on the "consumers". Note that each "consumer" is accelerated in proportion to p_2, when the ℓ-level load balancing inside it works efficiently.

On the other hand, the induction hypothesis above gives the condition for the latter:

$$\frac{1}{\mu_1} \sim \frac{1}{\lambda} \cdot p_2^{\alpha_\ell} \cdot (\log p_2)^{\beta_\ell} \tag{3}$$

From these equations we obtain

$$\tau_{\ell+1}(p) \sim \frac{1}{\lambda} \cdot p^{\alpha_{\ell+1}} \cdot (\log p)^{\beta_{\ell+1}}$$

$$\alpha_{\ell+1} = 2 - \frac{1}{\alpha_\ell}, \qquad \beta_{\ell+1} = 1 + \frac{\beta_\ell}{\alpha_\ell}$$

Hence we obtain the following as desired.

$$\alpha_\ell = \frac{\ell + 1}{\ell}, \qquad \beta_\ell = \frac{\ell + 1}{2} \qquad \blacksquare \tag{4}$$

This theorem implies that the multi-level dynamic load balancing scheme is *indeed* more scalable than the

single-level one in the sense of isoefficiency. Scalability is improved with the number of levels.

The next theorem implies that a producer at a higher level should have *more* "consumers" than one at a lower level. In particular, the uniform tree configuration of processors, in which a producer at every level has equally $p^{1/\ell}$ consumers, is *not* optimal.

Theorem 4 (optimal processor configuration) *In order to attain $\eta_{\max}$ above we should have*

$$(\textit{degree of the root producer}) = O(p^{\frac{1}{\ell}} \cdot (\log p)^{\frac{\ell-1}{2}})$$

INTUITIVE PROOF: Note that the claim is trivial for $\ell = 1$. From Equation (2),(3) and (4) above, we obtain

$$p_1 \sim p^{\frac{\ell-1}{\ell}} \cdot (\log p)^{\frac{\ell}{\ell}} = p^{\frac{1}{\ell+1}} \cdot (\log p)^{\frac{\ell}{2}}$$

This is the desired result for $(\ell+1)$-level with $\ell \geq 1$. ∎

6 Conclusions

We have investigated the optimal efficiency of the multi-level dynamic load balancing scheme for OR-parallel programs, using probability theory. In particular, we showed how the multi-level dynamic load balancing scheme improves scalability over the single-level one in terms of isoefficiency function.

It would be worthwhile to perform similar probabilistic analysis of other load balancing schemes, *e.g.*, the parallel depth first search algorithms [5], kabu-wake [7].

So far, we have not considered inter processor communication latency, or other overheads associated with parallel execution. How the efficiency suffers from these should be treated in future works.

7 Acknowledgments

We would like to thank Vipin Kumar for taking part in valuable discussions

Appendix

A Proof Sketch of Equation (1)

For simplicity we may assume $\sigma = 1$. Then,

$$\begin{aligned} E(\max_{1\leq i\leq p} X_i) &= p\int_0^\infty x e^{-x}(1-e^{-x})^{p-1}dx \\ &= -p\int_0^1 (1-y)^{p-1}\log y dy \\ &= -p\cdot\frac{\partial}{\partial q}B(p,q)|_{q=1} \\ &= -\Gamma'(1)+\frac{\Gamma'(p+1)}{\Gamma(p+1)} \\ &= C+2+\log(p+1)-\frac{1}{2(p+1)} \\ &\quad -\int_0^\infty \frac{2t}{t^2+(p+1)^2}\cdot\frac{dt}{e^{2\pi t}-1} \end{aligned}$$

where Γ denotes gamma function; B, beta function; and C, Euler's constant [1]. Since the last term is negative and greater than $-\frac{1}{2(p+1)}$, equation (1) follows immediately. ∎

References

[1] L. V. Ahlfors, *Complex Analysis*, McGraw-Hill (1966).

[2] M. Furuichi, K. Taki, and N. Ichiyoshi, "A Multi-Level Load Balancing Scheme for OR-Parallel Exhaustive Search Programs on the Multi-PSI," *Proc. of the 2nd ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming*, pp.50–59 (1990).

[3] R. M. Karp, "The Probabilistic Analysis of Some Combinatorial Search Algorithms," in J. F. Traub (ed.), *Algorithms and Complexity: New Directions and Recent Results*, Academic Press (1976).

[4] K. Kimura and N. Ichiyoshi, "Probabilistic Analysis of the Optimal Efficiency of a Dynamic Load Balancing Scheme," *in preparation, to appear in ICOT Technical Report.*

[5] V. Kumar and V. N. Rao, "Load Balancing on the Hypercube Architecture," *Proc. of the 1989 Conference on Hypercubes, Concurrent Computers and Applications*, pp.603–608 (1989).

[6] V. Kumar and A. Gupta, "Analysis of Scalability of Parallel Algorithms and Architectures: A Survey," *to appear in the 1991 International Conference on Supercomputing* (1991).

[7] K. Kumon, H. Masuzawa, A. Itashiki, K. Satoh, and Y. Sohma, "Kabu-wake: A New Parallel Inference Method and Its Evaluation," *ICOT Technical Report* 150 (1986).

[8] N. U. Prabhu, *Queues and Inventories*, John Wiley (1965).

[9] P. C. Treleaven, "Parallel Architecture Overview," *Parallel Computing* 8, pp.71–83 (1988).