TM-0974

Probabilistic Analysis of the Optimal Efficiency of the Multi-Level Dynamic Load Balancing Scheme (Extended Abstract)

by

K. Kimura & N. Ichiyoshi

November, 1990

Probabilistic Analysis of the Optimal Efficiency of the Multi-Level Dynamic Load Balancing Scheme (Extended Abstract)

Kouichi Kimura kokimura@icot.or.jp
Nobuyuki Ichiyoshi ichiyoshi@icot.or.jp
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 the given task into several subtasks, which are distributed to the 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 the dividing costs and the load imbalance among the processors. Thus we obtain the estimates of the maximal efficiency.

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

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

1 Introduction

The purpose of parallel processing is to accelerate the execution of time-consuming tasks by utilizing a number of processors. The *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 others. 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 larger and larger-scale multiprocessors are being developed recently [4], the scalability analysis of the efficiency is becoming more and more important.

In general, the efficiency may deteriorate due to the load imbalance among the processors, the latency of inter-processor communication, or other overheads associated with parallel execution. In particular, if the algorithm is composed of many parts requiring unpredictable amount of computation, as is usual with many combinatorial search problems, load balancing is not an easy task and have a great influence on the efficiency. Thus the load balancing is one of the central issues of parallel processing. A number of dynamic load balancing techniques have been proposed [1, 2, 5].

Furuichi *et al.* [1] proposed *the multi-level dynamic load balancing scheme* for OR-parallel programs, and evaluated its performance using the Multi-PSI, a distributed memory parallel computer with 64 processors. Their basic strategy is to divide a given problem into mutually independent subtasks, and distribute them to the other processors on demand. The on-demand distribution will balance the loads among the processors. However, when the number of processors increases, the number of the subtasks should correspondingly increase. Therefore, if the dividing is entrusted to one processor (*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*). And 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 such a dynamic load balancing scheme. We define a formal model of the single-level dynamic load balancing as a queuing system with several servers, and analyze its behavior using probability theory. We then apply these results to the analysis of the multi-level dynamic load balancing scheme. Among others, we show:

1. With subtasks of random size, the optimal efficiency is worse than that with subtasks of exactly the same size. In terms of isoefficiency function[1], the difference typically appears as an extra $\log p$ factor, where p is a number of processors.

2. The multi-level load balancing scheme is *indeed* more "scalable" than the single-level one. The difference appears as a smaller fractional order of p in the isoefficiency function.

[1] *Isoefficiency function* indicates the rate at which the problem size must grow in order to maintain the efficiency as the number of processors increases [2].

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 *naive* tree configuration with the same degree at all the level 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 [3].

2 Model of the Single-Level Dynamic Load Balancing

We introduce a formal model of the single-level dynamic load balancing, based on which we will discuss the optimal efficiency.

We consider here one of the simplest on-demand load distribution technique: — given a task, one *producer* processor divides it into *mutually independent* subtasks, which are transmitted to the *consumer* processors on demand and then executed. We assume that the granularity of the subtask can be controlled, namely, we can divide the task into more subtasks of smaller size or fewer ones of larger size at will.

For example, let us consider an OR-parallel exhaustive search of a tree : — the producer searches the tree up to depth d and produce subtasks, each of which is the search of a subtree with its root at depth d. Then we can control the granularity of the subtask by choosing the depth d.

In order to attain good efficiency, we have to choose an adequate granularity. With the fewer subtasks of larger size, the load imbalance among the consumers may occur. On the other hand, with the more subtasks of smaller size, the producer may become a bottleneck. Taking these into account, we define our formal model as follows.

Now we begin with preparing the necessary notations and the related assumptions.

- $p > 1$: number of consumers.
  Note that the number of processors is $p+1$ hereafter.
- $T_1 > 0$: random variable, representing the size of the given task (CPU time required for executing it using 1 processor).
- $t_1 = E(T_1)$: expectation of the task size.

We have in mind a family of the "problem spaces" parameterized by t_1, the average size of the instance in each space. We may choose a "problem space" with adequate mean size according to the number of available processors so that we can expect a good speed-up. We assume the task, or the target problem, is given randomly from the space. Note that we do *not* know the exact size of each instance task before execution.

- $N = 1, 2, \ldots$: random variable, representing the number of subtasks.
- $\nu = E(N) \geq 1$: its expectation.

We may choose adequate ν, or adequate granularity of the subtasks, according to the expectation of the task size and the number of processors. Note that we cannot control N itself in general. For example of the above tree search, N, the number of the subtrees with their roots at depth d would not be known before execution, although its expectation may be estimated beforehand.

We divide the task into probabilistically equivalent subtasks. Note that we don't require them to be of exactly equal size.

- R_n : random variable, representing the size of the n-th subtask (CPU time required for executing it).

  $\{R_n\}_{n=1,2,\ldots}$: independent identically distributed

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

- U_n : random variable, representing the CPU time required for producing the n-th subtask.

  $\{U_n\}_{n=1,2,\ldots}$: independent identically distributed

Definition 1 *Given $R_1, \ldots, R_N, U_1, \ldots, U_N$ and p as above, we define the single-level dynamic load balancing model, in which the execution time T_p should be determined as follows :*

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

where Y_n and others represent the time specified below.

$$O_n = \sum_{k=1}^{n} U_k : \textit{ birth of the n-th subtask}$$

$$X_n = \max(O_n, Z_n) : \textit{ onset of executing the n-th subtask}$$

$$Y_n = X_n + R_n : \textit{ completion of executing n-th subtask}$$

$$Z_n = \begin{cases} \displaystyle\min_{1 \leq i_1 < \cdots < i_{p-1} \leq n-1} \; \max_{\substack{1 \leq k \leq n-1 \\ k \neq i_1, \ldots, i_{p-1}}} Y_k & (p < n \leq N) \\ 0 & (1 \leq n \leq p) \end{cases}$$

where Z_n represents the time when at least one of the consumers becomes ready to execute the n-th subtask.

Here we assumed that the subtasks are immediately transmitted from the producer to the consumers when demanded and available. Note that Z_n depends only on $Y_1, \ldots, Y_{n-1}$, and hence these are determined by induction on n. This model can be regarded as a queuing system with p servers.

In the following we define several other characteristics of the model.

- $\dfrac{1}{\lambda} = E(U_n)$: average time for producing a subtask
- $\dfrac{1}{\mu} = \dfrac{t_1}{\nu} = E(R_n)$: mean subtask size
- $\rho = \dfrac{\lambda}{\mu p}$: ratio of the production rate to the consumption rate of the subtasks.

We typically suppose that producing a subtask is just to compute an "address" which specifies the portion of the search space to be assigned. We also assume that this address should be computed in a reasonable amount of time — in a polynomial time of the description length of p. Namely,

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

In tree search, a subtask, *i.e.* a subtree, 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. And we should choose $d = O(\log p)$ in order to produce a polynomial number of subtasks in p (Later we can see that the optimal ν is polynomially bounded in p (Corollary 1)).

Finally, we define:

- $t_p = E(T_p)$: mean execution time
- $s_p = \dfrac{t_1}{t_p}$: mean speed-up
- $\eta = \dfrac{s_p}{p}$: mean efficiency
- $\eta_{\max} = \sup\limits_{\nu} \eta(p, t_1, \nu)$: maximal mean efficiency

Note that the alternative definition of the *mean* speed-up might be $E(T_1/T_p)$ in its literal sense. However, since this quantity is hard to analyze, we did not adopt this definition. We believe that the present definition still serves as a useful indicator of the mean speed-up. The *maximal mean efficiency* $\eta_{\max}$ is the mean efficiency when we choose the optimal ν for given p and t_1.

3 Analysis of the Single-Level Dynamic Load Balancing

We call the model defined in the preceding section *deterministic* when we know the exact size of the given task and can divide it into an arbitrary number of subtasks of exactly the same size using a constant time per subtask :

$$T_1 \equiv t_1 \ .$$

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

Proposition 1 (deterministic model)

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

Here $\simeq$ represents 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 becomes a bottleneck and hence the efficiency will be deteriorated, when and only when $\rho < 1$ [6]. So we assume $\rho \geq 1$. Since each of the p consumer processors completes μ subtasks per unit time.

$$(\text{onset time of executing the last subtask}) \simeq \frac{\nu - 1}{\mu p}$$

The last subtask will require $1/\mu$ computation time, hence

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

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

Choosing $\rho \simeq 1$, we obtain the desired result. ∎

We call the model *exponential* when the size of the task (subtask) has exponential distribution and the number of the subtasks has geometric distribution :

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

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

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

Note that the last two of these equations implies the first. Hence this is a consistent assumption.

Theorem 1 (exponential model)

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

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

INTUITIVE PROOF: (i) Since $\rho > 1$, the onset 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,

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

Therefore,

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

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

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

Lemma 1 *Let $X, X_1, \ldots, X_p$ be mutually independent random variables with identical exponential distribution with mean σ:*

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

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

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

and

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

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

Now, compare the result in the deterministic model with that in the exponential model. In the latter an extra factor of $\log p$ appears, which comes from the load imbalance due to the diversity in the subtask size.

Next let us proceed to the general case. If we admitted an excessive diversity in the subtask size, it would be hopeless to try to achieve reasonable efficiency and the results would be uninteresting to us. Hence we assume the following, which indirectly limits the diversity.

We assume that each submitted subtask should be carried out "steadily" in the following sense: — the expectation of the remaining execution time of a subtask should never exceed the initially expected execution time too much, and the variance of the remaining execution time should stay moderate. Namely,

(Moderate diversity in the subtask size) *For* $\forall n$,

$$E(R_n - a \mid R_n \geq a) \leq \frac{1}{\mu} \cdot (1 + O(\frac{1}{|\log \mu|})) \quad for\ \forall a \geq 0$$

$$E((R_n - a)^2 \mid R_n \geq a) = O(\frac{1}{\mu^2}) \quad for\ \forall a \geq 0$$

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

For example, if R_n is uniformly distributed over $[0, 2/\mu]$, this condition is satisfied.

On the contrary, imagine a typical situation in which the above condition does not hold: search of a highly imbalanced tree. Among the subtrees with their roots at the same depth, only a few of them are exceptionally larger than the others. Accordingly, the average size of the few large subtrees is far larger than the overall average. Here, the above condition is violated; and the few exceptionally large subtasks will induce fatal load imbalance. A remedy for such cases might be to subdivide the subtask, which has been found to be too large. Such a load balancing scheme is beyond the scope of discussion here.

Theorem 2 (general case) *Under the above condition,*

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

INTUITIVE PROOF: According to Lemma 1, we may regard the exponential model 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. ∎

Corollary 1 *Let* ρ *and* c *as above. For* $0 < \forall \epsilon < 1$, *if*

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

then by choosing ν *such that*

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

we have

$$\eta \geq 1 - \epsilon$$

4 Isoefficiency Analysis of the Multi-Level Dynamic Load Balancing Scheme

The multi-level dynamic load balancing scheme is an iterated application of the single-level dynamic load balancing scheme in a hierarchical manner [1]. In our terminology, it is defined as follows by induction on the level ℓ.

The 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 the processors, which executes 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 3 to the isoefficiency analysis [2] of the multi-level load balancing. Namely, we investigate the size of the task required to maintain the efficiency when the number of processors increases.

We begin with a remark on the single-level dynamic load balancing. In addition to the previous assumptions, we assume that the number N of the subtasks should also have moderate diversity in its magnitude. Then it can be shown that both the total task size T_1 and the parallel execution time T_p should also have the same property [3]. Thus we can apply the preceding analysis iteratively to the multi-level load balancing.

Theorem 3 (isoefficiency of the multi-level dynamic load balancing scheme) *With level* ℓ, *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 mean size of the task for which the ℓ-level dynamic load balancing is expected to work with efficiency at least η_0 using p consumer processors, where η_0 is an arbitrarily given positive constant less than 1.

We use induction. Now 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 $\sim$ expresses 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 composed of the root producer and p_1 virtual "consumers", each of which is in fact composed of several 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:

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

$$(2) \qquad \frac{1}{\mu_1} \sim \frac{p_1}{\lambda} \cdot p_2$$

where $1/\mu_1$ denotes the mean 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:

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

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.

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

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.

The next theorem implies that a producer at a higher level should have *more* "consumers" than one at a lower level. In particular, the *naive* 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

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

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

5 Conclusion

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 the scalability over the single-level one.

It would be worthwhile to perform similar analysis of other load balancing schemes, *e.g.* the parallel depth first search algorithms [2], kabu-wake[5].

So far, we did not consider the latency of inter-processor communication, or other overheads associated with parallel execution. How the efficiency suffers from these should be discussed in the future work.

6 Acknowledgments

We would like to thank Vipin Kumar for valuable discussions.

Appendix

A Sketch of the Proof 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, B, C denotes gamma function, beta function, and Euler's constant respectively [7]. Since the last term is negative and greater than $-\frac{1}{2(p+1)}$, equation 1 follows immediately. $\blacksquare$

References

[1] 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).

[2] 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).

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

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

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

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

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