# Randomized Static Load Balancing for Tree-Shaped Computations

Peter Sanders
LS Informatik für Ingenieure und Naturwissenschaftler
Universität Karlsruhe, 76128 Karlsruhe
E-mail: sanders@ira.uka.de

**Abstract**

Parallelizing a problem by statically assigning a fixed number of subproblems to each processor is very popular due to its simplicity and low communication overhead. In many cases it can be proved to be sufficient to randomly assign $O(\log N)$ subproblems to each of $N$ processors in order to smooth out load imbalance due to varying subproblem sizes. However, this is not true for the *tree structured computations* considered in this paper because the subproblem sizes get less uniform when the number of generated subproblems is increased. Even under moderate assumptions, a polynomial number of subproblems needs to be assigned to each processor. Still, for machines with slow communication and applications with good splitting functions, a carefully designed randomized static load balancer can be a competitive alternative to dynamic load balancing schemes. The results also help to explain the impact of static load balancing as an initialization method for a subsequent dynamic load balancing phase.

## 1 Introduction

One of the key tasks in parallel programming is load balancing, i.e., evenly distributing subproblems over the individual processors (PEs). One of the simplest approaches is to statically assign an equal number of subproblems to each PE. Even if we assume that interactions between subproblems are not important, load imbalance due to varying subproblem sizes may limit the efficiency of this approach. If, in addition, subproblem sizes are hard to predict, there seems to be no way to get good load balance. But this dilemma can be solved by randomization. For example, in [8] it is proved that in many cases it is sufficient to randomly assign $O(\log N)$ subproblems[1] to each of $N$ PEs in order to smooth out load imbalance on the average.

In this paper we want to consider randomized static load balancing for tree shaped computations (RaTSLoB). In *tree shaped computations* subproblems are generated by repeatedly splitting a root problem. This is a natural model for many algorithms based on tree search like backtracking or depth first branch-and-bound. We show that even under moderate assumptions, a polynomial number of

[1] Throughout this paper log stands for the logarithm base 2.

subproblems needs to be assigned to each PE in order to get good performance. The degree of this polynomial depends on the quality of the splitting function. Still, for good splitting functions and machines with slow communication static load balancing is a competitive algorithm.

Section 2 introduces the machine and application model under consideration. Then, Section 3 explains how RaTSLoB can be implemented with low communication overhead. Section 4 summarizes previous work in this area. The main part of this paper is the analysis of RaTSLoB in Section 5. Finally, Section 6 summarizes the results and mentions some interesting questions for future work. In particular, two ideas about the impact of static load balancing as an initialization scheme for dynamic load balancing are introduced.

# 2 Definitions

We consider a MIMD computer consisting of $N = 2^n$ identical PEs[2] which interact by exchanging messages through a network of diameter $d(N)$. A message of length $s$ can be broadcast to all PEs in time $O(d(N) + s)$.

The problem *size* is measured by the sequential execution time $w$ required to solve the entire problem (the root problem). The root problem can be represented by a message of length $r(w)$. The only things an algorithm can do with a problem is to work on it, to check whether it is exhausted and to split it into two parts. Splitting takes time $s(w)$. Nothing is guaranteed about the relative size of the two generated subproblems.

For the purpose of the analysis, we model the quality of the splitting function as follows: A subproblem of size $v$ is split into two parts of size $Xv$ and $(1-X)v$ where $X$ is a random variable with range $[0, 1]$, a constant variance $\mathbf{Var}X = \sigma^2 \in [0, \frac{1}{4}]$, mean $\mathbf{E}X = \frac{1}{2}$ and $\mathbf{P}[X = x] = \mathbf{P}[X = (1-x)]$. The last two conditions can be assumed without loss of generality because they can always be met by randomly exchanging the generated subproblems with probability $\frac{1}{2}$. Different applications of the splitting functions are assumed to be independent. The fact that the size of the generated subproblems adds up to the original problem size is not as evident as it appears because many tree shaped computations have to risk some speculative computation in order to be able to split the problem at all.[3] We neglect this effect because we want to concentrate on the load balancing aspect and because there are many applications where speculative computation is not a severe problem.

We assess the performance of a load balancing algorithm by analyzing a problem class without interactions between the subproblems so that load balancing is the only source of parallelization overhead. This can even be justified for applications wich do have interactions between subproblems if we want to concentrate of the load balancing aspect.

A natural performance measure is the parallel execution time $T_{\text{par}}(N, w)$ or the *efficiency* $E(N, w) = \frac{w}{NT_{\text{par}}(N,w)}$. The awkwardness of discussing these bivariate functions can be avoided by fixing $E$ and solving for $w$ yielding the *isoefficiency function* $w(N)$. This function is a convenient measure for the degree of scalability

[2]The algorithms and analysis presented here can also adapted to PE numbers which are not an integer power of two.

[3]The reverse effect is also possible and can lead to "superlinear" speedup.

```
(* Input: *)
R : Problem (* Root problem *)
k : N (* Number of subsequent splits to be performed *)

Determine a random permutation π : {1, ..., 2^k} → {1, ..., 2^k}
Broadcast R to each PE
FOR PE i := 1 TO 2^n DOPAR (* asynchronously *)
    FOR j := (i − 1)2^(k−n) + 1 TO i2^(k−n) DO
        P := R (* Problem under consideration *)
        let (b_0, ..., b_(k−1)) be the bit representation of π(j) − 1
        FOR l := 0 TO k − 1 DO
            IF b_l = 0 THEN
                P := left part of splitting P into two parts
            ELSE
                P := right part of splitting P into two parts
        process P
collect results
```

Figure 1: Generic Algorithm for randomized static load balancing for tree shaped computations.

of an algorithm. The smaller the isoefficiency function, the smaller the problem size required for cost optimal parallel execution. For a more detailed discussion refer to [9].

# 3 Implementation issues

Figure 1 shows the basic algorithm for randomized static load balancing for tree shaped computations. Conceptually, the root problem $R$ is split into $2^k$ parts which are equally but randomly distributed to the PEs using the permutation $\pi^{-1}$. But instead of actually transmitting subproblems, every PE directly generates the subproblems allocated to it. In order to do this, a single broadcast of the root problem is sufficient. No further communication is necessary until the results are collected[4]. This approach trades $k$ redundant split operations for a global transmission of a subproblem.

Splitting problems into exactly two parts seems to be somewhat impractical on the first glance. On the one hand, many applications are more naturally expressed in terms of splitting functions which generate a variable number of successors (e.g. the successors of a depth first search tree node). But the successors can always be grouped into two subsets. Heuristics which try to divide a problem as evenly as possible are often easier to express with exactly two subproblems even if the underlying application has a variable number of successors [16, 20]. On the other hand, a split operation might produce an empty subproblem. Due to the

[4] For many applications, collecting results can be implemented as efficiently as distributing work. For example, if the overall result is combined from the partial results by applying an associative and commutative operation (e.g. $+$, min or $\cup$), the partial results can first be combined locally and then globally using a single reduction operation.

randomization we can hope that the empty subproblems generated by subsequent vain split operations are equally distributed over the PEs.

The only nontrivial part of Algorithm 1 is the random permutation $\pi$. The classical algorithm for generating a random permutation [7, Section 3.4.2, Algorithm P] is sequential and requires explicit storage of the function table. Using this algorithm would ruin the storage and communication economy we wanted to gain with our approach. Fortunately, we do not really need to be able to generate every permutation with equal likelyhood. We only need some permutation which is sufficiently uncorrelated to the behavior of the splitting function.

A quite simple approach is based on the theory of finite fields [13]. Let $p$ be a primitive polynomial modulo 2 of degree $k$ (e.g. as tabulated in [15]). Then the polynomial $x^l \bmod p$ ($l$ relatively prime to $2^k - 1$) is a generating element of the multiplicative group of $GF(2^k)$, i.e., the sequence $((x^l)^1, \ldots, (x^l)^{2^k-1})$ enumerates the nonzero polynomials modulo $p$. By interpreting the coefficients of the polynomials as bits of a computer word, we can efficiently enumerate the numbers between 1 and $2^k - 1$. If we insert the number $2^k$ at some random place in this sequence we get the desired pseudo-random permutation. Computing the next sequence element takes $O(k)$ word operations (multiplication of binary polynomials modulo $p$ using shifts and `XOR`s) and computing the starting element on each processor can be done with $O(k^2)$ word operations using a square-and-multiply exponentiation algorithm.

A somewhat more complicated but related idea are Sobol sequences [15] which also enumerate the numbers between 1 and $2^k - 1$ using primitive polynomials modulo 2. They are popular for applications like Monte Carlo integration because they sample the integration domain more uniformly than a true random sequence. Whether this special feature is also useful for our application remains to be seen, but one advantage is that the next sequence element can be generated using only a constant number of word operations.

# 4 Related work

The idea to parallelize the traversal of large trees by statically decomposing it into identical subtrees is quite old. In [1] trees are decomposed without the detour over bisection of the tree.[5] However, since every PE gets only one subproblem the efficiency is quite limited. Later, this and similar approaches were used as an initialization scheme for dynamic load balancing algorithms [12, 11, 19, 5, 20]. In [18, 6] PEs are initialized with several subproblems of work (but without using tree splitting). Quite some work has been done on decomposing trees where finding any solution branch will finish the search. See [2] for a recent survey.

Our model of tree splitting is inspired by the work of Rao and Kumar [17, 9]. In our terminology they assume that the distribution of the splitting factor $X$ is $\mathbf{P}\left[X = \frac{1}{2} - \sigma\right] = \mathbf{P}\left[X = \frac{1}{2} + \sigma\right] = \frac{1}{2}$ where $\sigma$ is some arbitrary but fixed constant smaller than $\frac{1}{2}$. An important difference is that in our approach the actual value of $\sigma$ turns out to be quite important. On the other hand we can model applications where the result of the splitting function is occasionally very low. (For example consider a splitting event where a subproblem of size $w$ is split

[5]The formulas given in the paper contain small errors. This can be considered to be one additional argument for our simpler approach of always bisecting the tree.

into subproblems of size $c$ and $w-c$. Depending on $w$, the splitting factor can get arbitrarily close to 0 here.)

For other load models, it is a quite common technique to assign several pieces of work with independent sizes to each processor [8, 4, 14]. Often it is tried to justify the independence assumption by assigning subtasks to a PE which are in some sense as far apart as possible [14]; [18, 6] use this rule for trees and it works well for them but in general true randomization is safer because it is easy to construct plausible looking trees where the maximum distance strategy is catastrophically bad. A general problem for the analysis of the algorithms considered here is that assumptions about statistical properties are required which cannot really hold because the underlying computation is unknown but quite deterministic. (With the notable exception of probabilistic simulations [4].)

# 5 Analysis

In Section 3 we did not specify how the parameter $k$ in Algorithm 1 should be chosen in order to get good performance. This is a crucial question. If $k$ is too small, the load will turn out to be unevenly distributed. If $k$ is too large, the algorithm spends most of its time generating tiny subproblems of work which are not worth this effort. For this purpose, Section 5.1 derives basic results which make it possible to give bounds on the optimal value of $k$ in Section 5.2. Section 5.3 uses these results to assess the scalability of the algorithm.

## 5.1 Properties of the work splitting process

Since we assumed that different split operations are independent, it is easy to derive the expected value and the variance of the size of a subproblem that has been split $k$ times:

**Theorem 1** *Let $Y$ be the size of a subproblem generated by $k$ splits of a root problem of size 1. Then, $\mathbf{E}Y = 2^{-k}$ and $\mathbf{Var}Y = \left(\sigma^2 + \frac{1}{4}\right)^k - 4^{-k}$.*

**Proof:** Let $X_1, \ldots, X_k$ be the independent identically distributed multiplication factors for the $k$ splitting operations performed (i.e. $Y = \prod_{i=1}^{k} X_i$). We know that $\mathbf{E}X_i = \frac{1}{2}$ and $\mathbf{Var}X_i = \mathbf{E}{X_i}^2 - (\mathbf{E}X_i)^2 = \sigma^2$ (or $\mathbf{E}{X_i}^2 = \sigma^2 + \frac{1}{4}$). Therefore,

$$\mathbf{E}Y = \mathbf{E}\prod_{i=1}^{k} X_i = \prod_{i=1}^{k} \mathbf{E}X_i = 2^{-k}$$

and

$$\mathbf{Var}Y = \mathbf{E}\left(\prod_{i=1}^{k} X_i\right)^2 - (\mathbf{E}Y)^2 = \prod_{i=1}^{k} \mathbf{E}{X_i}^2 - 4^{-k} = \left(\sigma^2 + \frac{1}{4}\right)^k - 4^{-k}.$$

■

An interesting question is how large subproblems can get. A simple way to produce one of the largest (leaf-)subproblems is to always retain the largest subproblem after a split operation. Although the algorithm does not know which one is the larger subproblem, there will always be exactly one subproblem generated in this way.

**Theorem 2** *Let $M$ be the size of a subproblem generated by $k$ splits of a root problem of size 1 where after a split the larger subproblems are retained. Then,* $\mathbf{E}M = \left(\sigma + \frac{1}{2}\right)^k$.

**Proof:** Define $X_i$ as in the proof of Theorem 1. Let $M_i := \max(X_i, 1-X_i)$, then

$$\mathbf{E}M_i = \mathbf{E}\max(X_i, 1-X_i) = \mathbf{E}\left(\frac{1}{2} + \max(X_i - \frac{1}{2}, \frac{1}{2} - X_i)\right) = \frac{1}{2} + \mathbf{E}\left|X - \frac{1}{2}\right| = \frac{1}{2} + \sigma.$$

Since the $M_i$ are independent we get

$$\mathbf{E}M = \mathbf{E}\prod_{i=1}^{k} M_i = \prod_{i=1}^{k}\mathbf{E}M_i = \left(\sigma + \frac{1}{2}\right)^k.$$

■

Ultimately we want to know how much the load of a PE (i.e. the sum of the sizes of the subproblems allocated to it) varies. Answering this question is complicated by the fact that the sizes of subproblems derived from the same root problem are not independent. However, the variance of the sum of subproblem sizes is not larger than the sum of their variances. (If the subproblem sizes were independent, equality would hold.)

**Theorem 3** *Let $Y_1, \ldots, Y_m$ be the sizes of $m$ subproblems which are randomly selected from a collection of $2^k$ subproblems generated from the same root problem of size 1, and let $Z_{km} = \sum_{i=1}^{m} Y_i$. Then,*

$$\mathbf{Var}Z_{km} \leq m\left[\left(\sigma^2 + \frac{1}{4}\right)^k - 4^{-k}\right].$$

**Proof:** (By induction over $k$.)
**Case $k = 0$:** $\mathbf{Var}Z_{0m} = 0 \leq 0 = m(1-1) = m\left[\left(\sigma^2 + \frac{1}{4}\right)^0 - 4^{-0}\right]$
**Case $k - 1 \longrightarrow k$:** We prove the equivalent proposition

$$\mathbf{E}Z_{km}^2 \leq m\left[\left(\sigma^2 + \frac{1}{4}\right)^k - 4^{-k}\right] + (\mathbf{E}Z_{km})^2 = m\left[\left(\sigma^2 + \frac{1}{4}\right)^k - 4^{-k}\right] + m^2 4^{-k}.$$

Let $I$ denote the number or subproblems (out of the $m$ subproblems considered) wich are generated from the left child problem of the root problem, let $X$ denote the splitting factor for the root problem and let $XZ_{k-1,I}$ be the sum of the sizes of these $I$ subproblems and $(1-X)Z'_{k-1,m-I}$ the sum of the sizes of the remaining problems:

$$\mathbf{E}Z_{km}^2 = \sum_{i=0}^{m}\mathbf{P}\left[I = i\right]\mathbf{E}\left(XZ_{k-1,i} + (1-X)Z'_{k-1,m-i}\right)^2$$

Using the independence of $Z_{k-1,i}$ and $Z'_{k-1,m-i}$ and the facts $\mathbf{E}(1-X)^2 = \left(\sigma^2 + \frac{1}{4}\right)$, $\mathbf{E}X(1-X) = \left(\frac{1}{4} - \sigma^2\right)$, $\mathbf{E}Z_{jl} = l2^{-j}$ yields

$$\mathbf{E}Z_{km}^2 = \sum_{i=0}^{m}\mathbf{P}\left[I = i\right]\left(\left(\sigma^2 + \frac{1}{4}\right)\left(\mathbf{E}Z_{k-1,i}^2 + \mathbf{E}{Z'}_{k-1,m-i}^2\right) + 2\left(\frac{1}{4} - \sigma^2\right)i(m-i)4^{-(k-1)}\right).$$

By the induction hypothesis we have

$$\mathbf{E}Z_{k-1,i}^2 \leq i\left[\left(\sigma^2+\frac{1}{4}\right)^{k-1} - 4^{-(k-1)}\right] + i^2 4^{-(k-1)}$$

and

$$\mathbf{E}{Z'}_{k-1,m-i}^2 \leq (m-i)\left[\left(\sigma^2+\frac{1}{4}\right)^{k-1} - 4^{-(k-1)}\right] + (m-i)^2 4^{-(k-1)}$$

or

$$\begin{aligned}
&\left(\sigma^2+\frac{1}{4}\right)\left(\mathbf{E}Z_{k-1,i}^2 + \mathbf{E}{Z'}_{k-1,m-i}^2\right)\\
&\leq \;\; m\left(\left(\sigma^2+\frac{1}{4}\right)^{k} - \left(\sigma^2+\frac{1}{4}\right)4^{-(k-1)}\right) + \left(\sigma^2+\frac{1}{4}\right)(i^2+(m-i)^2)4^{-(k-1)}.
\end{aligned}$$

Using $\left(\sigma^2+\frac{1}{4}\right) \geq \frac{1}{4}$, $i^2+(m-i)^2 \leq \frac{m^2}{2}$, $i(m-i) \leq \frac{m^2}{4}$ and $\sum_{i=0}^{m} \mathbf{P}\left[I=i\right] = 1$ we get

$$\begin{aligned}
\mathbf{E}Z_{km}^2 &\leq m\left(\left(\sigma^2+\frac{1}{4}\right)^{k} - 4^{-k}\right) + \left(\left(\sigma^2+\frac{1}{4}\right)\frac{m^2}{2} + \left(\frac{1}{4}-\sigma^2\right)\frac{m^2}{2}\right)4^{-(k-1)}\\
&= m\left(\left(\sigma^2+\frac{1}{4}\right)^{k} - 4^{-k}\right) + m^2 4^{-k}
\end{aligned}$$

■

## 5.2 Lower and upper bounds on $k$

In order to achieve good load balance on the average, the large subproblem defined in Theorem 2 must not be larger than a constant factor $\epsilon$ of the expected load of a single PE. This means

$$\mathbf{E}M = \left(\sigma+\frac{1}{2}\right)^k \leq \epsilon 2^{-n}$$

or

$$k \geq \frac{n-\log\epsilon}{\log\frac{1}{\sigma+1/2}}.$$

Using asymptotic notation this result can be grasped in the following theorem:

**Theorem 4** *For good efficiency on the average*

$$2^k \in \Omega\left(2^{n/\log\frac{1}{\sigma+1/2}}\right)$$

*is required.*

The remainder of this section is concerned with proving the following counterpart to Theorem 4 which gives an upper bound on $k$.

**Theorem 5** *Let $Z_i$, ($i \in \{1,\ldots,2^n\}$) be the load of PE $i$. If $k = \frac{2(n-\log\epsilon)}{\log\frac{1}{2\sigma^2+1/2}}$ then*

$$\mathbf{E}\max_{i=1}^{n} Z_i \leq (1+2\epsilon)2^{-n}.$$

*For good efficiency on the average*

$$2^k \in O\left(2^{2n/\log\frac{1}{2\sigma^2+1/2}}\right)$$

*is therefore sufficient.*

For this value of $k$ we can conclude from Theorem 3:

**Lemma 1** $\mathbf{Var} Z_i \leq \epsilon^2 8^{-n}$.

**Proof:**

$$\begin{aligned}
\mathbf{Var} Z_i &\leq 2^{k-n}\left(\left(\sigma^2+\frac{1}{4}\right)^k - 4^{-k}\right) \leq 2^{-n}\left(2\sigma^2+\frac{1}{2}\right)^k \\
&= 2^{-n+2(n-\log\epsilon)\frac{\log(2\sigma^2+1/2)}{\log\frac{1}{2\sigma^2+1/2}}} = \epsilon^2 8^{-n}
\end{aligned}$$

■

In order to exploit this knowledge, we need the following two results from the literature:

**Theorem 6 (Chebyshev Inequality [3])** *For a random variable $X$ with existing second moment:*

$$\mathbf{P}\left[|X-\mathbf{E}X|\geq t\right] \leq \frac{\mathbf{Var} X}{t^2}$$

**Lemma 2 ([10])** *Let $X_1, \ldots, X_l$ be identically distributed random variables, and $a := \inf\{x | \mathbf{P}[X > x] \leq l^{-1}\}$. Then, no matter how the $X_i$ are dependent*

$$\mathbf{E} \max_{i=1}^{l} X_i \leq a + l \int_a^\infty \mathbf{P}[X > x]\, dx.$$

The Chebyshev Inequality makes it possible to estimate the $a$ from Lemma 2:

**Lemma 3** $\mathbf{P}[Z_i \geq (1+\epsilon)2^{-n}] \leq 2^{-n}$

**Proof:**

$$\mathbf{P}\left[Z_i \geq (1+\epsilon)2^{-n}\right] \leq \mathbf{P}\left[|Z_i - 2^{-n}| \geq \epsilon 2^{-n}\right] \leq \frac{\mathbf{Var} Z_i}{(\epsilon 2^{-n})^2} \leq \frac{\epsilon^2 8^{-n}}{\epsilon^2 4^{-n}} \leq 2^{-n}$$

■

Now we can prove Theorem 5:
Let $a := \inf\{z | \mathbf{P}[Z_i > z] \leq 2^{-n}\}$. From Lemma 3 we know that $a \leq (1+\epsilon)2^{-n}$ and Lemma 2 gives

$$\begin{aligned}
\mathbf{E} \max_{i=1}^{2^n} Z_i &\leq a + 2^n \int_a^\infty \mathbf{P}[Z > z]\, dz \\
&= a + 2^n \int_a^{(1+\epsilon)2^{-n}} \mathbf{P}[Z > z]\, dz + 2^n \int_{(1+\epsilon)2^{-n}}^1 \mathbf{P}[Z > z]\, dz.
\end{aligned}$$

By definition of $a$:

$$2^n \int_a^{(1+\epsilon)2^{-n}} \mathbf{P}[Z > z]\, dz \leq (1+\epsilon)2^{-n} - a.$$

$\mathbf{P}[Z > z]$ can be estimated by again using the Chebyshev Inequality:

$$\mathbf{P}[Z > z] \leq \mathbf{P}\left[|Z - 2^{-n}| \geq z - 2^{-n}\right] \leq \frac{\epsilon^2 8^{-n}}{(z-2^{-n})^2}$$

and therefore,

$$
\begin{aligned}
2^n \int_{(1+\epsilon)2^{-n}}^{1} \mathbf{P}\left[Z > z\right] dz &\leq \epsilon^2 4^{-n} \int_{(1+\epsilon)2^{-n}}^{1} \frac{1}{(z-2^{-n})^2} \\
&= \epsilon^2 4^{-n} \left( \frac{1}{2^{-n}-1} + \frac{2^n}{\epsilon} \right) \leq \epsilon 2^{-n}.
\end{aligned}
$$

Putting everything together yields

$$
\mathbf{E} \max_{i=1}^{2^n} Z_i \leq a + (1+\epsilon)2^{-n} - a + \epsilon 2^{-n} = (1+2\epsilon)2^{-n}
$$

■

## 5.3 Scalability analysis

The bounds on $k$ derived in the preceding section can be used to assess the performance of Algorithm 1. Let $w$ be the total load, $wZ_i$ the load of PE $i$. The root problem can be distributed in time $\Theta(d(N) + r(w))$; producing $2^{k-n}$ subproblems on each PE takes time $k2^{k-n}s(w)$ and processing problems takes time $w \max_{i=1}^{N} Z_i$. We further assume that collecting the results is no more expensive than distributing the root problem and computing $\pi$ is no more expensive than doing $k$ splits. Then,

$$
T_{\text{par}} \in \Theta(d(N) + r(w) + k2^{k-n}s(w) + w \max_{i=1}^{N} Z_i)
$$

If $k$ is chosen in such a way that $\mathbf{E} \max_{i=1}^{N} Z_i \in O(2^{-n})$) then Theorems 4 and 5 can be used to infer bounds on the expected parallel execution time:

$$
\mathbf{E}T_{\text{par}} \in \Omega \left( d(N) + r(w) + \log N s(w) N^{\frac{1}{\log \frac{1}{\sigma+1/2}} - 1} + \frac{w}{N} \right)
$$

$$
\mathbf{E}T_{\text{par}} \in O \left( d(N) + r(w) + \log N s(w) N^{\frac{2}{\log \frac{2}{2\sigma^2+1/2}} - 1} + \frac{w}{N} \right)
$$

Using these equations we can decide how large $w$ must be in order to achieve good efficiency. We derive the isoefficiency functions for the case that $s(w) \in \Theta(log^a w)$ $(a \geq 1)$ and $r(w) \in O(d(n) + \log N s(w) N^{\epsilon})$ (for some $\epsilon > 0$) which is typical for search problems with exponential search space. As long as $w$ remains polynomial in $N$ we can conclude that $s(w) \in \Theta(log^a N)$. And for $\sigma > 0$ the terms for network diameter and work splitting dominate $r(w)$. It is now easy to solve the equation $E = \frac{w}{NT_{\text{par}}}$ for $w$:

$$
w \in \Omega(Nd(N) + N^{\frac{1}{\log \frac{1}{\sigma+1/2}}} \log^{a+1} N) \tag{1}
$$

and

$$
w \in O(Nd(N) + N^{\frac{2}{\log \frac{2}{2\sigma^2+1/2}}} \log^{a+1} N). \tag{2}
$$

It is interesting to compare this result with *random polling* [9, 20], one of the best available dynamic load balancing algorithms. Using the above notations and additionally assuming $r(w) \in O(s(w))$:

$$
w \in O(Nd(N) \log^{a+1} N)
$$

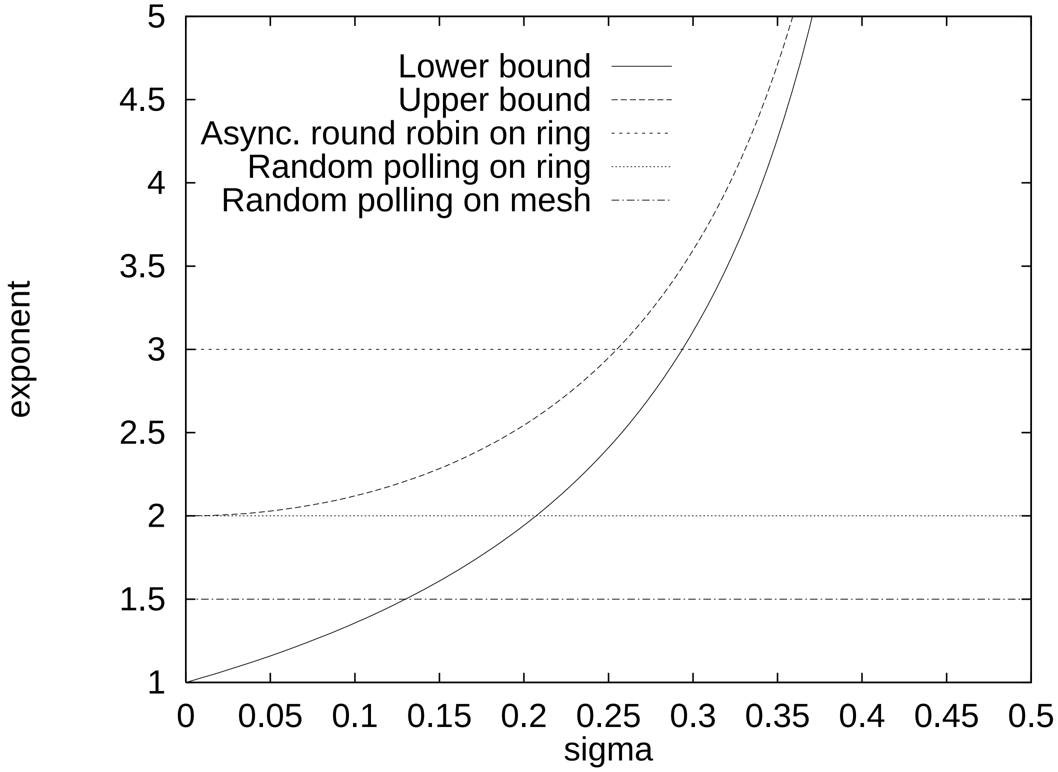


Figure 2: Exponent of polynomial part of isoefficiency function depending on the quality of the splitting function.

is sufficient for good efficiency on the average[6]. Since $d(N) \in O(N)$ for any interconnection network, random polling is strictly better than the upper bound from Equation 2. However, for small $\sigma$ this lead is much smaller than the lead to other popular load balancing algorithms like *asynchronous round robin* [9] or even simple nearest neighbor schemes [17]. If the true performance turns out to be closer to the lower bound from Equation 1, Algorithm 1 is better than random polling for large network diameters and small $\sigma$. Figure 2 shows the exponent of the bounds on the isoefficiency of Algorithm 1 due to load imbalance compared to the exponent for some dynamic load balancing algorithms. It demonstrates that the performance is heavily dependent on the quality of the splitting function while dynamic load balancing is (at least asymptotically) unaffected by it. The figure also shows that there is a significant gap in the analysis for small $\sigma$.

# 6 Conclusions and future work

We have analyzed a randomized static load balancing algorithm for tree shaped computations which is very space efficient and requires no communication except for a broadcast of the initial problem and for collecting results. The key to this algorithm is a pseudo-random permutation which can be efficiently generated in parallel. This algorithm is scalable in the sense that a problem size polynomial

[6] This result is derived for a different load model but can be adapted to the case of a probabilistic splitting function.

in the number of processors is sufficient for good efficiency. However, the degree of this polynomial is heavily dependent on the quality of the splitting function. Therefore, the algorithm can only compete with state of the art dynamic load balancing algorithms if a high quality splitting function is available and if communication is slow.

The quantitative results about the behavior of the splitting process go beyond this particular algorithm because they might also help to understand other algorithms. For example, in [18] a combined static/dynamic load balancing algorithm is described which starts by allocating a constant number of subproblems to each processor. Later, idle processors get work from nearby busy processors. In this phase, subproblems are never split in order to avoid the negative effects of splitting in the context of neighborhood communication described in [17]. However, the authors report problems due to uneven sizes of the subproblems. If we assume that our probabilistic splitting model is applicable to the applications described in [18] (e.g. iterative deepening search for solutions of the 15-puzzle) then Theorem 4 can explain this problem: A number of subproblems *polynomial* in $N$ has to be allocated to each PE, or else we must expect to have one subproblem which is so large that it dominates the execution time of the algorithm. This also changes the analytical results for the isoefficiency function of this algorithms from $O(Nd(N))$ to $O(d(N)N^{1/\log\frac{1}{\sigma+1/2}})$.

Nevertheless, the general idea to combine static and dynamic load balancing algorithms is promising for the future. For example, consider the following algorithm: Each PE is randomly assigned exactly 1 subproblem. Then the PEs are disconnected into $\log^\alpha N$ clusters (for some appropriate constant $\alpha$) which locally perform random polling dynamic load balancing. The results derived here can be used to show that this algorithm has an isoefficiency function $w \in O(N^{1+1/u})$ on a $u$-dimensional mesh. This is asymptotically optimal.

Other interesting questions for the future are:

- Can the gap between upper and lower bounds be made smaller, perhaps by making additional (reasonable) assumptions about the work splitting process?
- How does the algorithm work in practice?
- Can the work splitting model be made more realistic? What, if the splitting factor depends on the remaining problem size? What if subsequent split operations are dependent?

# References


[1] O. I. El-Dessouki and W. H. Huen. Distributed enumeration on between computers. *IEEE Transactions on Computers*, C-29(9):818–825, September 1980.

[2] W. Ertel. *Parallele Suche mit randomisiertem Wettbewerb in Inferenzsystemen.* PhD thesis, TU München, 1992.

[3] W. Feller. *An Introduction to Probability Theory and its Applications.* Wiley, 3rd edition, 1968.

[4] P. Heidelberger. Discrete event simulations and parallel processing: Statistical properties. *SIAM Journal of Statistical Computation*, 9(6):1114–1132, 1988.

[5] D. Henrich. Initialization of parallel branch-and-bound algorithms. In *Proceedings of PPAI-93*, 1993.

[6] D. Henrich. *Lastverteilung für Branch-and-bound auf eng-gekoppelten Parallelrechnern*. PhD thesis, Universität Karlsruhe, 1994 (in preparation).

[7] D. E. Knuth. *The Art of Computer Programming — Seminumerical Algorithms*, volume 2. Addison Wesley, 2nd edition, 1969.

[8] C. P. Kruskal and A. Weiss. Allocating independent subtasks on parallel processors. *IEEE Transactions on Computers*, 11(10):1001–1016, 1985.

[9] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing. Design and Analysis of Algorithms*. Benjamin/Cummings, 1994.

[10] T. Lai and H. Robbins. A class of dependent random variables and their maxima. *Zeitschrift für Wahrscheinlichkeitstheorie*, 42:89–111, 1978.

[11] R. Lüling and B. Monien. Load balancing for distributed branch & bound algorithms. In *Int. Parallel Processing Symposium (IPPS)*, 1992.

[12] R. P. Ma, F. S. Tsung, and M. H. Ma. A dynamic load balancer for a parallel branch and bound algorithm. In *3rd Conference on Hypercubes, Concurrent Computers, and Applications*, pages 1505–1530, Pasadena, 1988. ACM.

[13] T. Minkwitz. Personal communication. Department of Informatics, Universität Karlsruhe, 1994.

[14] D. M. Nicol and J. H. Saltz. An analysis of scatter decomposition. *IEEE Transactions on Computers*, 39:1337–1345, 1990.

[15] W. H. Press, S. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C*. Cambridge University Press, 2nd edition, 1992.

[16] V. N. Rao and V. Kumar. Parallel depth first search. Part I. *International Journal of Parallel Programming*, 16(6):470–499, 1987.

[17] V. N. Rao and V. Kumar. Parallel depth first search. Part II. *International Journal of Parallel Programming*, 16(6):501–519, 1987.

[18] A. Reinefeld and V. Schnecke. Work-load balancing in highly parallel depth-first search. In *Scalable High Performance Computing Conference*, pages 773–780, Knoxville, 1994.

[19] P. Sanders. Suchalgorithmen auf SIMD-Rechnern – Weitere Ergebnisse zu Polyautomaten. Master's thesis, Universität Karlsruhe, August 1993.

[20] P. Sanders. Analysis of random polling dynamic load balancing. Technical Report IB 12/94, Universität Karlsruhe, Fakultät für Informatik, April 1994.