

# Parallel Load Balancing for Problems with Good Bisectors

Stefan Bischof    Ralf Ebner    Thomas Erlebach  
Institut für Informatik  
Technische Universität München  
D-80290 München, Germany  
{bischof | ebner | erlebach}@in.tum.de

## Abstract

*Parallel load balancing is studied for problems with certain bisection properties. A class of problems has  $\alpha$ -bisectors if every problem  $p$  of weight  $w(p)$  in the class can be subdivided into two subproblems whose weight (load) is at least an  $\alpha$ -fraction of the original problem. A problem  $p$  is to be split into  $N$  subproblems such that the maximum weight among them is as close to  $w(p)/N$  as possible. It was previously known that good load balancing can be achieved for such classes of problems using Algorithm HF, a sequential algorithm that repeatedly bisects the subproblem with maximum weight. Several parallel variants of Algorithm HF are introduced and analyzed with respect to worst-case load imbalance, running-time, and communication overhead. For fixed  $\alpha$ , all variants have running-time  $O(\log N)$  and provide constant upper bounds on the worst-case load imbalance. Results of simulation experiments regarding the load balance achieved in the average case are presented.*

## 1. Introduction

Dynamic load balancing for irregular problems is a major research issue in the field of parallel computing. It is often essential to achieve a balanced distribution of load in order to reduce the execution time of an application or to maximize system throughput. Load balancing problems have been studied for a huge variety of models, and many different solutions regarding strategies and implementation mechanisms have been proposed. A good overview of recent work can be obtained from [14], for example. Ongoing research on dynamic load balancing is reviewed in [13], emphasizing the presentation of models and strategies within the framework of classification schemes.

We consider a general scenario where an irregular problem is generated at run-time and must be split into subproblems that can be processed on different processors. If  $N$

processors are available, the problem is to be split into (at most)  $N$  subproblems, and the goal is to balance the weight of the subproblems assigned to the individual processors. The weight of a problem represents the load (e.g., CPU load) caused by processing that problem. It is assumed that the weight of a problem can be calculated (or approximated) easily once it is generated.

Instead of splitting the original problem into  $N$  subproblems in a single step, we propose to use repeated *bisections* in order to generate the  $N$  subproblems, because in practice it is often easier to find efficient bisection methods than to find a method for splitting a problem into an arbitrary number of subproblems in one step. A bisection subdivides a problem into two subproblems.

Our research is motivated by an application from distributed numerical simulations, namely a parallel solver for systems of linear equations resulting from the discretization of partial differential equations. This solver uses a variant of the finite element method (FEM) with adaptive, recursive substructuring [6, 7]. The recursive substructuring phase yields an unbalanced binary tree (called FE-tree). In order to parallelize the main part of the computation, the FE-tree must be split into subtrees that can be distributed among the available processors. Useful bisection methods for FE-trees and the speed-up achieved by incorporating dynamic load balancing using bisections are reported in [1].

Besides distributed hierarchical finite element simulations, load balancing using bisections can be applied to computational fluid dynamics, domain decomposition in the process of chip layout [12], or multi-dimensional adaptive numerical quadrature [4]. In this paper, we will not be concerned with details regarding any particular application. Instead, we study parallel load balancing from a more abstract point of view and for a very general class of problems.

## 2. Load balancing model

As in [1, 2], we study the following simplified model for dynamic load balancing. The parallel system consists

of  $N$  processors, numbered from 1 to  $N$ . The number of a processor is referred to as its *id*. The  $i$ -th processor is denoted  $P_i$ . Initially, a problem  $p$  resides on  $P_1$ .  $P_1$  is called *busy*, while the other processors are idle and called *free*. A free processor becomes busy when it receives a subproblem from a busy processor. The goal of the load balancing is to split  $p$  into  $N$  subproblems  $p_1, \dots, p_N$  such that subproblem  $p_i$  can subsequently be processed by  $P_i$ ,  $1 \leq i \leq N$ . (A load balancing algorithm is allowed to split  $p$  into fewer than  $N$  subproblems. In this case, some processors remain idle.) Assuming a weight function  $w$  that measures the resource demand (for example, CPU load or main memory requirement, depending on the application at hand) of subproblems, the goal is to minimize the maximum weight among the resulting subproblems, i.e., to minimize  $\max_{1 \leq i \leq N} w(p_i)$ .

Repeated bisections must be used to split a problem into  $N$  subproblems. Bisections will not always split a problem  $p$  with weight  $w(p)$  into two subproblems with weight  $w(p)/2$  each, but for many classes of problems there is a bisection method that guarantees that the weights of the two obtained subproblems do not differ too much.

**Definition 1** [1] Let  $0 < \alpha \leq \frac{1}{2}$ . A class  $\mathcal{P}$  of problems with weight function  $w : \mathcal{P} \rightarrow \mathbb{R}^+$  has  $\alpha$ -bisectors if every problem  $p \in \mathcal{P}$  can be efficiently divided into two problems  $p_1 \in \mathcal{P}$  and  $p_2 \in \mathcal{P}$  with  $w(p_1) + w(p_2) = w(p)$  and  $w(p_1), w(p_2) \in [\alpha w(p), (1 - \alpha)w(p)]$ .

Note that our definition characterizes classes of problems that have  $\alpha$ -bisectors in a very abstract way. In a particular application, problems might correspond to subdomains of a numerical computation, to parts of a simulated system, to parts of the search space for an optimization problem (cf. [9]), or to program execution dags. A definition very similar to ours ( $\alpha$ -splitting) is used in [10, pp. 315–318] under the assumption that the weight of a problem is unknown to the load balancing algorithm.

```

algorithm HF( $p, N$ ):
 $P := \{p\}$ ;
while  $|P| < N$  do
     $q :=$  a problem in  $P$  with maximum weight;
    bisect  $q$  into  $q_1$  and  $q_2$ ;
     $P := (P \cup \{q_1, q_2\}) \setminus \{q\}$ ;
od;
return  $P$ ;

```

**Figure 1. Algorithm HF**

Figure 1 shows Algorithm HF (Heaviest Problem First), which receives a problem  $p$  and a number  $N$  of processors as input and divides  $p$  into  $N$  subproblems by repeated application of  $\alpha$ -bisectors to the heaviest remaining subproblem.

A perfectly balanced load distribution on  $N$  processors would be achieved if a problem  $p$  of weight  $w(p)$  was divided into  $N$  subproblems of weight exactly  $w(p)/N$  each. We say that an algorithm has *performance guarantee*  $\rho$  if the maximum weight produced by the algorithm is at most a factor of  $\rho$  larger than this ideal weight  $w(p)/N$  even in the worst case. Note that we ignore other factors that might affect the quality of a load distribution in a particular application (e.g., we assume that there are no precedence constraints among the subproblems, so that all subproblems can be processed in parallel once the load has been distributed, and we do not consider communication costs between subproblems). The following theorem shows that the performance guarantee of Algorithm HF can be bounded by an expression in  $\alpha$  that does not depend on  $N$ .

**Theorem 2** [1] Let  $\mathcal{P}$  be a class of problems with weight function  $w : \mathcal{P} \rightarrow \mathbb{R}^+$  that has  $\alpha$ -bisectors. Given a problem  $p \in \mathcal{P}$  and a positive integer  $N$ , Algorithm HF uses  $N - 1$  bisections to partition  $p$  into  $N$  subproblems  $p_1, \dots, p_N$  such that  $\max_{1 \leq i \leq N} w(p_i) \leq \frac{w(p)}{N} \cdot r_\alpha$ , where  $r_\alpha = \lfloor \frac{1}{\alpha} \rfloor \cdot (1 - \alpha)^{\lfloor \frac{1}{\alpha} \rfloor - 2}$ .

Note that  $r_\alpha$  is equal to 2 for  $\alpha \geq 1/3$ , smaller than 3 for  $\alpha > 1 - 1/\sqrt[3]{2} \approx 0.159$ , and smaller than 10 for  $\alpha \geq 0.04$ . Hence, Algorithm HF achieves provably good load balancing for classes of problems with  $\alpha$ -bisectors for a surprisingly large range of  $\alpha$ .

It is common to represent the process of recursively subdividing a problem into smaller problems by a tree. In [1] a *bisection tree* was used to represent the run of a bisection-based load balancing algorithm on input  $p$  and  $N$ . The root of the bisection tree  $T_p$  is the problem  $p$ . If the algorithm bisects a problem  $q$  into  $q_1$  and  $q_2$ , nodes  $q_1$  and  $q_2$  are added to  $T_p$  as children of node  $q$ . In the end,  $T_p$  has  $N$  leaves, which correspond to the  $N$  subproblems computed by the algorithm, and all problems that were bisected by the algorithm appear as internal nodes with exactly two children.

### 3. Parallel load balancing

Algorithm HF is a sequential algorithm that bisects only one problem at a time. Hence, the time for load balancing grows (at least) linearly with the number of processors. We assume that each bisection step is performed sequentially on one processor. Therefore, the time spent in load balancing can be reduced substantially if several bisection steps are executed concurrently on different processors.

We assume that a processor that bisects a problem can quickly (in constant time) acquire the number of a free processor and, after bisecting the problem into two subproblems, send one of the two subproblems to that free processor. In Section 3.4 we will discuss ways to implement such

a management of free processors in a parallel system without increasing the asymptotic running-time.

Furthermore, we assume that the bisection of a problem into two subproblems requires one unit of time and that the transmission of a subproblem to a free processor requires one unit of time. Our results can easily be adapted to different assumptions about the time for bisections and for interprocessor communication. Finally, we assume that standard operations like computing the maximum weight of all subproblems generated so far or sorting a subset of these subproblems according to their weights can be done in time  $O(\log N)$ . This assumption is satisfied by the idealized PRAM model [8], which can be simulated on many realistic architectures with at most logarithmic slowdown.

### 3.1. Parallelizing Algorithm HF

In this section, we describe a parallelization of Algorithm HF, referred to as Algorithm PHF (Parallel HF). Algorithm PHF is shown in Figure 2.

```

algorithm PHF( $p, N$ )           co code for  $P_i$ 
if  $i = 1$  then  $q := p$ ;
else wait until a problem  $q$  is received; fi;
while  $w(q) > \frac{w(p)}{N} \cdot r_\alpha$  do
  bisect  $q$  into  $q_1$  and  $q_2$ ;
(a)  send  $q_2$  to a free processor;
      $q := q_1$ ;
od;
(b)  barrier;
(c)   $f :=$  number of free processors;
     while  $f > 0$  do
(d)   $m :=$  maximum weight of remaining subproblems;
(e)   $h :=$  number of proc. with subproblem  $\geq m(1 - \alpha)$ ;
     if  $h \leq f$  then
       if  $w(q) \geq m(1 - \alpha)$  then
         bisect  $q$  into  $q_1$  and  $q_2$ ;
(f)  send  $q_2$  to a free processor;
          $q := q_1$ ; fi;
       else determine the  $f$  heaviest subproblems;
         if  $q$  is among them then
           bisect  $q$  into  $q_1$  and  $q_2$ ;
(g)  send  $q_2$  to a free processor;
            $q := q_1$ ; fi;
     fi;
      $f := f - \min\{h, f\}$ ;
(h)  if  $f > 0$  then barrier; fi;
     od;

```

**Figure 2. Algorithm PHF**

Steps that require a form of global communication (communication involving more than two processors at a time) are shaded in the figure; on many parallel machines these steps will not take constant time, but time  $\Theta(\log N)$ .

In order to parallelize Algorithm HF, we must, on the one hand, perform several bisections simultaneously, but, on the other hand, ensure that no subproblem is bisected unless it would also have been bisected by the sequential Algorithm HF. First, we observe that subproblems with weight greater than  $\frac{w(p)}{N} \cdot r_\alpha$  are certainly bisected by Algorithm HF and can always be bisected by Algorithm PHF as well. Next, note that subproblems with weight at most  $w(p)/N$  are certainly not bisected any further by Algorithm HF; such subproblems can be assigned to a processor immediately by Algorithm PHF.

Algorithm PHF can carry out a first phase of load balancing as follows. Before the first bisection is made, the values  $w(p)$ ,  $N$ , and  $\alpha$  are broadcast.  $P_1$  bisects the original problem  $p$  into  $p_1$  and  $p_2$ , and then sends  $p_2$  to  $P_2$ . Whenever a processor gets a subproblem  $q$  in the following, it checks whether  $w(q) > \frac{w(p)}{N} \cdot r_\alpha$  and, if so, bisects it into two subproblems and sends one of the two subproblems to a free processor. This phase ends when all subproblems have weight at most  $\frac{w(p)}{N} \cdot r_\alpha$ . A barrier synchronization is performed in step (b) to ensure that all processors finish the first phase of the algorithm at the same time. Free processors that have not received a subproblem during the first phase go to step (b) of the algorithm directly as soon as they are informed about the termination of phase one.

Consider the bisection tree representing the bisections performed in the first phase. Let  $D$  denote the maximum depth of a leaf in this bisection tree. Under our assumptions, the running-time for the first phase is clearly bounded by  $O(D)$ . Now observe that a node at depth  $d$  in the bisection tree has weight at most  $w(p)(1 - \alpha)^d$ . Therefore,  $D$  can be at most  $\log_{\frac{1}{1-\alpha}} N$ . The running-time for the first phase can thus be bounded by  $O(\log_{\frac{1}{1-\alpha}} N)$ .

If one was only interested in obtaining a parallel algorithm with the same performance guarantee as Algorithm HF, one could stop the load balancing after this first phase and leave the remaining free processors idle. However, as will be seen from the simulation results presented in Section 4, the maximum load obtained from Algorithm HF is much smaller than the worst-case bound for many problem instances, especially when  $\alpha$  is small. Thus we aim at a parallel solution that produces the same partitioning as the sequential Algorithm HF.

In order to achieve this, Algorithm PHF continues as follows. After the barrier synchronization in step (b), the number  $f$  of free processors is calculated in step (c) and made known to all processors. At the same time, the free processors can be numbered from 1 to  $f$ , and the id of the  $i$ -th free processor can be stored at  $P_i$ . In this way, any processor can later on determine the id of the  $i$ -th free processor by questioning  $P_i$ . Then, the second phase of the algorithm consists of iterations of the following steps: (1) Determine the max-

imum weight  $m$  among the subproblems generated so far ( $m$  is broadcast to all processors). (2) Determine the number  $h$  of processors that have a subproblem of weight at least  $m(1 - \alpha)$ , and number them from 1 to  $h$  ( $h$  is broadcast to all processors). (3a) If  $h \leq f$ , all  $h$  processors that have subproblems of weight at least  $m(1 - \alpha)$  bisect their subproblem; the  $i$ -th such processor sends one of its two resulting subproblems to the  $i$ -th free processor that has not received a subproblem in a previous iteration. (3b) If  $h > f$ , the  $f$  heaviest subproblems are determined and numbered from 1 to  $f$  (employing either selection or sorting as a subroutine); for  $1 \leq i \leq f$ , the processor with the  $i$ -th among the  $f$  heaviest subproblems bisects its subproblem and sends one of the two resulting subproblems to the  $i$ -th free processor that has not received a subproblem in a previous iteration.

In each iteration, the value of  $f$  represents the number of remaining free processors. Every processor can update its copy of  $f$  locally by subtracting  $\min\{h, f\}$ . The load balancing terminates when there are no free processors left.

Observe that the  $\min\{h, f\}$  subproblems chosen for bisection in each iteration would also have been bisected by the sequential Algorithm HF, because none of the bisections in the current iteration can generate a new subproblem with weight greater than  $m(1 - \alpha)$ . Hence, Algorithm PHF produces the same partitioning of  $p$  into  $N$  subproblems as Algorithm HF.

Note that global communication is required in every iteration of phase two. The values of  $m$  and  $h$  can be determined and broadcast to all processors in steps (d) and (e) by simple prefix computations (see [8]) in time  $O(\log N)$ . The barrier at the end of every iteration takes time at most  $O(\log N)$  as well. In all iterations except the last one, no further global communication is required: a processor that bisects a problem in that iteration can determine the id of a free processor by a single request to another processor whose id it can determine locally. Only in the last iteration it can be necessary to determine the  $f$  heaviest subproblems and number them from 1 to  $f$ . This can be done in time  $O(\log N)$  by employing a parallel sorting or selection algorithm (see [8]).

Taking all this into account, the reader may verify easily that the running-time for one iteration of phase two is  $O(\log N)$  under our assumptions. In addition, the maximum weight among all subproblems is reduced by a factor of  $(1 - \alpha)$  in each iteration. As the maximum weight can never become smaller than  $w(p)/N$ , the algorithm will terminate no later than after  $I$  iterations if  $I$  satisfies  $\frac{w(p)}{N} \cdot r_\alpha \cdot (1 - \alpha)^I \leq \frac{w(p)}{N}$ . This inequality is satisfied if  $(1 - \alpha)^{I + \lceil \frac{1}{\alpha} \rceil - 2} \leq \alpha$ . As  $(1 - \alpha)^{\frac{1}{\alpha} \ln \frac{1}{\alpha}} \leq \alpha$  (note that  $(1 - \alpha)^{\frac{1}{\alpha}} \leq 1/e$  and  $(1/e)^{\ln \frac{1}{\alpha}} = \alpha$ ), the former inequality is surely satisfied if  $I \geq \frac{1}{\alpha} \ln \frac{1}{\alpha}$ . Hence, the number of iterations performed by Algorithm PHF in phase two is bounded from above by  $\frac{1}{\alpha} \ln \frac{1}{\alpha}$ , and each iteration takes

time  $O(\log N)$ . Altogether, phase two has running-time  $O(\frac{1}{\alpha} \ln \frac{1}{\alpha} \log N)$ , which is  $O(\log N)$  for fixed  $\alpha$ .

**Theorem 3** *Given a problem  $p$  from a class  $\mathcal{P}$  of problems with  $\alpha$ -bisectors for a fixed constant  $\alpha$ , Algorithm PHF subdivides  $p$  into  $N$  subproblems in time  $O(\log N)$  such that the resulting subproblems are the same as for the sequential Algorithm HF.*

While the described parallel implementation of Algorithm HF has optimal running-time for constant  $\alpha$  under our assumptions, there are also drawbacks of this approach. First, we have completely ignored the management of free processors in phase one so far. Although we will show in Section 3.4 that this can be implemented without increasing the asymptotic running time, it must be expected that substantial communication overhead will occur. Further, the second phase of Algorithm PHF requires global communication in each iteration.

### 3.2. Algorithm BA

To overcome the difficulties encountered with Algorithm PHF, we now propose an alternative algorithm for the load balancing problem that is inherently parallel. The maximum load generated by this algorithm in the worst case is larger than the worst-case bound for Algorithm HF only by a small constant factor. In addition, the management of free processors can be realized efficiently for this algorithm, and no global communication is required at all.

```

algorithm BA( $p, N$ )
if  $N = 1$  then return  $\{p\}$ ;
else bisect  $p$  into  $p_1$  and  $p_2$ ;
    co assume w.l.o.g.  $w(p_1) \leq w(p_2)$ 
     $\hat{\alpha} := w(p_1)/w(p)$ ;
    if  $\hat{\alpha}N - \lfloor \hat{\alpha}N \rfloor \leq \hat{\alpha}$  then  $N_1 := \lfloor \hat{\alpha}N \rfloor$ ;
    else  $N_1 := \lceil \hat{\alpha}N \rceil$ ; fi;
     $N_2 := N - N_1$ ;
    return BA( $p_1, N_1$ )  $\cup$  BA( $p_2, N_2$ );
fi;

```

**Figure 3. Algorithm BA**

Algorithm BA (Best Approximation of ideal weight) is shown in Figure 3. It receives a problem  $p$  from a class of problems that has  $\alpha$ -bisectors and a number  $N$  of processors as input. Its output is a set containing the  $N$  subproblems generated from  $p$ . If  $N > 1$ , the problem  $p$  is bisected into subproblems  $p_1$  and  $p_2$ , and the  $N$  processors are partitioned between the two subproblems according to their relative weight. Subproblem  $p_i$  receives  $N_i \geq 1$  processors,  $i = 1, 2$ . Then, Algorithm BA is invoked recursively with input  $(p_i, N_i)$ ,  $i = 1, 2$ . These recursive calls can be executed in parallel on different processors. Note

that Algorithm BA does not require knowledge of the bisection parameter  $\alpha$  of the given class of problems.

Algorithm BA chooses  $N_1$  and  $N_2$  such that  $N_1 + N_2 = N$  and the maximum of  $w(p_i)/N_i$ ,  $i = 1, 2$ , is minimized. We assume w.l.o.g. that  $w(p_1) \leq w(p_2)$  and let  $\hat{\alpha} := w(p_1)/w(p)$ . Furthermore, let  $\lceil \hat{\alpha}N \rceil = \hat{\alpha}N + u$  and  $\lfloor \hat{\alpha}N \rfloor = \hat{\alpha}N - d$ , where  $0 \leq d, u < 1$ . It is not difficult to show that the maximum of  $w(p_i)/N_i$ ,  $i = 1, 2$ , is minimized if  $N_1$  is set to  $\lfloor \hat{\alpha}N \rfloor$  if  $d \leq \hat{\alpha}$ , and to  $\lceil \hat{\alpha}N \rceil$  if  $d > \hat{\alpha}$ .

Next, we give three lemmata (proofs are omitted due to space restrictions) that allow us to obtain a performance guarantee for Algorithm BA. In the following, let  $\mathcal{P}$  be a class of problems with weight function  $w : \mathcal{P} \rightarrow \mathbb{R}^+$  that has  $\alpha$ -bisectors.

**Lemma 4** *Given a problem  $p \in \mathcal{P}$  and an integer  $N \geq 2$ , it holds for each bisection step made by Algorithm BA that  $\max \left\{ \frac{w(p_1)}{N_1}, \frac{w(p_2)}{N_2} \right\} \leq \frac{w(p)}{N} \cdot \frac{N}{N-1}$ , where  $N_i \geq 1$  is the number of processors assigned to subproblem  $p_i$ ,  $i = 1, 2$ , by Algorithm BA.*

**Lemma 5** *Given a problem  $p \in \mathcal{P}$  and a positive integer  $N \leq 1/\alpha$ , Algorithm BA uses  $N - 1$  bisections to partition  $p$  into  $N$  subproblems  $p_1, \dots, p_N$  such that  $\max_{1 \leq i \leq N} w(p_i) \leq w(p)(1 - \alpha)^{\lfloor \frac{N}{2} \rfloor}$ .*

**Lemma 6** *Let  $p \in \mathcal{P}$ ,  $\rho \geq \alpha$ , and  $N \in \mathbb{N}$ . Then, for any subproblem  $\hat{p}$  generated by Algorithm BA on input  $(p, N)$  that has been assigned  $\hat{N} \geq \rho/\alpha + 1$  processors it holds that  $\frac{w(\hat{p})}{\hat{N}} \leq \frac{w(p)}{N} \cdot e^{(1-\alpha)/\rho}$ .*

Now we can prove a performance guarantee for Algorithm BA.

**Theorem 7** *Let  $\mathcal{P}$  be a class of problems with weight function  $w : \mathcal{P} \rightarrow \mathbb{R}^+$  that has  $\alpha$ -bisectors. Given a problem  $p \in \mathcal{P}$  and a positive integer  $N$ , Algorithm BA partitions  $p$  into  $N$  subproblems  $p_1, \dots, p_N$  such that*

$$\max_{1 \leq i \leq N} w(p_i) \leq \frac{w(p)}{N} \cdot e \cdot \left\lfloor \frac{1}{\alpha} \right\rfloor \cdot (1 - \alpha)^{\lfloor \frac{1}{2\alpha} \rfloor - 1}.$$

*If  $\alpha$  is a fixed constant, the running-time of Algorithm BA is  $O(\log N)$ .*

**Proof:** If  $N \leq 1/\alpha$ , the theorem follows from Lemma 5. Therefore, we assume  $N > 1/\alpha$  for the remainder of this proof. Consider a path  $Q = (q_0, q_1, \dots, q_k)$  from the root  $p =: q_0$  of the bisection tree  $T_p$  to some leaf  $p_i =: q_k$ ,  $i \in \{1, \dots, N\}$ . For  $0 \leq j \leq k$ , let  $N_j$  denote the number of processors assigned to  $q_j$  by Algorithm BA. Let  $q_{l+1}$  be the first node on this path (i.e., the node with minimum index) such that  $N_{l+1} \leq \frac{1}{\alpha}$ . Using Lemma 5 we conclude:

$$w(p_i) \leq \frac{w(q_{l+1})}{N_{l+1}} \cdot \left\lfloor \frac{1}{\alpha} \right\rfloor \cdot (1 - \alpha)^{\lfloor \frac{1}{2\alpha} \rfloor}. \quad (1)$$

Since  $N_l > 1/\alpha$  we can apply Lemma 6 for  $q_l$  with  $\rho = 1 - \alpha$ . This yields:

$$\frac{w(q_l)}{N_l} \leq \frac{w(p)}{N} \cdot e. \quad (2)$$

It remains to consider the bisection step at subproblem  $q_l$ . By Lemma 4 and from  $N_l > 1/\alpha$  we derive:

$$\frac{w(q_{l+1})}{N_{l+1}} \leq \frac{w(q_l)}{N_l} \cdot \frac{N_l}{N_l - 1} \leq \frac{w(q_l)}{N_l} \cdot \frac{1}{1 - \alpha}. \quad (3)$$

Combining (1), (3), and (2) completes the proof regarding the performance guarantee of Algorithm BA because the bounds are valid for any root-to-leaf path.

The management of free processors for Algorithm BA is very simple and does not introduce any communication overhead (see Section 3.4). Therefore, the running-time for Algorithm BA in our model is obviously bounded by the maximum depth of a node in the bisection tree representing the run of Algorithm BA, because in every step of the algorithm the internal nodes at one level of the bisection tree are bisected in parallel. It is not difficult to see that the number of processors is reduced by at least a factor of  $(1 - \alpha/2)$  in each bisection step, and thus the depth of a leaf in the bisection tree can be at most  $\log_{\frac{1-\alpha/2}{1-\alpha}} N$ .  $\square$

### 3.3. Combining BA and HF: Algorithm BA-HF

Our bound on the performance guarantee of Algorithm BA (Theorem 7) is not as good as the one for Algorithm HF. To obtain a highly parallel algorithm for our load balancing problem with a performance guarantee closer to the one for Algorithm HF we integrate Algorithm BA and Algorithm HF by dividing the bisection process into two phases. While the number of processors available for a particular subproblem is large enough, Algorithm BA-HF (see Figure 4) acts like Algorithm BA.

```

algorithm BA-HF( $p, N$ )
if  $N \geq \sigma/\alpha + 1$  then
  bisect  $p$  into  $p_1$  and  $p_2$ ;
  co assume w.l.o.g.  $w(p_1) \leq w(p_2)$ 
   $\hat{\alpha} := w(p_1)/N$ ;
  if  $\hat{\alpha}N - \lfloor \hat{\alpha}N \rfloor \leq \hat{\alpha}$  then  $N_1 := \lfloor \hat{\alpha}N \rfloor$ ;
  else  $N_1 := \lceil \hat{\alpha}N \rceil$ ; fi;
   $N_2 := N - N_1$ ;
  return BA-HF( $p_1, N_1$ )  $\cup$  BA-HF( $p_2, N_2$ );
else return [P]HF( $p, N$ ); fi;

```

**Figure 4. Algorithm BA-HF**

If the number of processors assigned to a subproblem is below a certain threshold, Algorithm HF is used to partition this subproblem further. To define this threshold precisely,

we assume that Algorithm BA-HF has knowledge about the bisection parameter  $\alpha$  of the given problem class. Algorithm HF is invoked by Algorithm BA-HF if  $N < \sigma/\alpha + 1$ , where  $\sigma \in \mathbb{R}^+$  is a parameter predefined by the application to reach the desired performance guarantee. Depending on the value of  $\sigma/\alpha$ , it may be advantageous to choose either the sequential Algorithm HF or Algorithm PHF for the implementation of the second phase of Algorithm BA-HF.

Regarding the running-time, it is clear that the first phase of Algorithm BA-HF is at most as long as the running-time for Algorithm BA and can thus be bounded by  $O(\log N)$  for fixed  $\alpha$ . If  $\sigma$  and, therefore,  $\sigma/\alpha$  are considered constants as well, the second phase of Algorithm BA-HF requires only constant additional work per processor, no matter whether the sequential Algorithm HF or Algorithm PHF is used. In this case, the overall running-time for Algorithm BA-HF is  $O(\log N)$ . If  $\sigma$  is allowed to be arbitrarily large, it is necessary to use Algorithm PHF in the second phase of Algorithm BA-HF in order to achieve running-time  $O(\log N)$ . A bound on the performance guarantee for Algorithm BA-HF can be proved in a similar way as for Algorithm BA (Theorem 7).

**Theorem 8** *Let  $\mathcal{P}$  be a class of problems with weight function  $w : \mathcal{P} \rightarrow \mathbb{R}^+$  that has  $\alpha$ -bisectors. Given a problem  $p \in \mathcal{P}$ , a positive integer  $N$ , and  $\sigma \geq \alpha$ , Algorithm BA-HF partitions  $p$  into  $N$  subproblems  $p_1, \dots, p_N$  such that*

$$\max_{1 \leq i \leq N} w(p_i) \leq \frac{w(p)}{N} \cdot e^{(1-\alpha)/\sigma} \cdot \left(1 + \frac{\alpha}{\sigma}\right) \cdot r_\alpha.$$

*If  $\alpha$  is a fixed constant, the running-time of Algorithm BA-HF is  $O(\log N)$ .*

According to this theorem we can bring the worst-case bound of Algorithm BA-HF on the ratio between  $\max_{1 \leq i \leq N} w(p_i)$  and  $w(p)/N$  arbitrarily close to the corresponding bound for Algorithm HF. For any  $\varepsilon > 0$ , if we let  $\sigma \geq 1/\ln(1 + \varepsilon)$ , the performance guarantee of Algorithm BA-HF is increased by a factor of at most  $1 + \varepsilon$  in comparison to Algorithm HF.

### 3.4. Managing the free processors

For Algorithm PHF, the problem of managing the free processors is the most challenging. In the first phase, it can be the case that a large number of processors bisect problems in parallel simultaneously and need to get access to a free processor in order to send a newly generated subproblem to it. Each sender must get the id of a different free processor. Depending on the machine model, various solutions employing distributed data structures for managing the free processors may be applicable: (randomized) work stealing [3], dynamic embeddings [5, 11], etc. In the following, we outline a solution that employs a modified version of Algorithm BA as a subroutine.

Let Algorithm  $\overline{\text{BA}}$  be an algorithm that is identical to Algorithm BA except that it does not bisect any subproblems with weight at most  $\frac{w(p)}{N} \cdot r_\alpha$ . The first part of phase one of Algorithm PHF consists of an execution of Algorithm  $\overline{\text{BA}}$  on inputs  $p$  and  $N$ . Theorem 7 implies that this execution takes time  $O(\log N)$ . At the end of this execution, only subproblems of weight greater than  $\frac{w(p)}{N} \cdot r_\alpha$  have been bisected. Furthermore, no remaining subproblem can be heavier than  $\frac{w(p)}{N} \cdot \lfloor \frac{1}{\alpha} \rfloor \cdot (1-\alpha)^{\lfloor \frac{1}{2\alpha} \rfloor - 1}$ . This follows from Theorem 7, because the only subproblems of weight greater than  $\frac{w(p)}{N} \cdot r_\alpha$  that are not bisected by Algorithm  $\overline{\text{BA}}$  are those that have been assigned to a single processor, and these subproblems would not have been bisected by Algorithm BA either.

The second (and last) part of phase one of Algorithm PHF is very similar to phase two and consists of a constant number of iterations (for fixed  $\alpha$ ) in each of which all remaining subproblems with weight greater than  $\frac{w(p)}{N} \cdot r_\alpha$  are bisected. As each iteration reduces the maximum weight among the remaining subproblems by a factor of  $(1 - \alpha)$ , it is not hard to see that  $\lfloor \frac{1}{2\alpha} \rfloor + 1$  iterations suffice to reach a situation where all remaining subproblems have weight at most  $\frac{w(p)}{N} \cdot r_\alpha$ .

After the end of the first phase of Algorithm PHF, the remaining free processors are determined and assigned numbers in time  $O(\log N)$ . As explained in Section 3.1, this knowledge can be exploited during the second phase of Algorithm PHF as follows: a busy processor can locally determine the number  $i$  of the free processor to which a newly generated subproblem must be sent, and a single request to processor  $P_i$  suffices to acquire the id of that free processor.

For Algorithm BA, the management of the free processors can be done in a very simple and efficient way. With each subproblem  $q$ , we simply store the range  $[i, j]$  of processors available for subproblems resulting from  $q$ . Initially, the original problem  $p$  has the full range  $[1, N]$  of processors available. When  $p$  is divided into  $p_1$  and  $p_2$  such that  $p_1$  gets  $N_1$  processors and  $p_2$  gets  $N_2$  processors,  $p_1$  stays on  $P_1$  and gets the range  $[1, N_1]$ , while  $p_2$  is sent to  $P_{N_1+1}$  and gets the range  $[N_1 + 1, N]$ . Similarly, a problem  $q$  with range  $[i, j]$  is always bisected at  $P_i$ , and the resulting subproblems  $q_1$  and  $q_2$  with corresponding numbers of processors  $N_1$  and  $N_2 = j + 1 - i - N_1$  are associated with ranges  $[i, i + N_1 - 1]$  and  $[i + N_1, j]$ , and problem  $q_2$  is sent to processor  $i + N_1$ . In this way, each processor can locally determine to which free processor it should send a newly generated subproblem, and no overhead is incurred for the management of free processors at all. This is one of the main advantages of Algorithm BA.

For Algorithm BA-HF, the simple management of free processors that is applicable for Algorithm BA can be used while the number of processors for a subproblem is large. For subproblems that are to be partitioned among a small

**Table 1. Worst-case upper bounds (ub) and observed minimum, average, and maximum ratios for  $\hat{\alpha} \sim U[0.01, 0.5]$ ,  $\sigma = 1.0$**

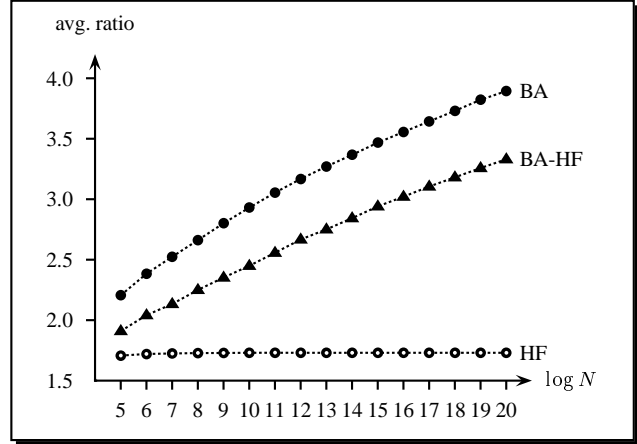
BA				
$\log N$	ub	min	avg	max
5	27.25	1.70	2.73	4.98
10	166.12	3.14	4.01	5.80
15	166.12	4.31	5.04	6.47
20	166.12	5.27	6.03	8.16
BA-HF				
$\log N$	ub	min	avg	max
10	101.51	1.99	2.27	3.40
15	101.51	2.40	2.92	4.72
20	101.51	3.16	3.88	5.89
HF				
$\log N$	ub	min	avg	max
5	23.43	1.55	1.94	2.63
10	37.35	1.87	1.96	2.08
15	37.35	1.94	1.96	1.98
20	37.35	1.96	1.96	1.97

number of processors ( $N < \sigma/\alpha + 1$ ), the management of free processors is trivial if the sequential Algorithm HF is used. If Algorithm PHF is used in the second phase of Algorithm BA-HF, however, the more complicated management of free processors described above can be employed.

#### 4. Simulation results

To gain further insight about the balancing quality achieved by the proposed algorithms, we carried out a series of simulation experiments. The goal of this simulation study was to obtain information about the average-case behavior of Algorithms BA and BA-HF in comparison to Algorithm HF. Since Algorithm PHF produces the same partitioning as Algorithm HF, no separate experiments were conducted for Algorithm PHF. The following stochastic model for an average-case scenario that may arise from practical applications seems reasonable: Assume that the actual bisection parameter  $\hat{\alpha}$  is drawn uniformly at random from the interval  $[\alpha, \beta]$ ,  $0 < \alpha \leq \beta \leq 1/2$ , and that all  $N - 1$  bisection steps are independent and identically distributed. (Such an assumption is valid, for example, if the problems are represented by lists of elements taken from an ordered set, and if a list is bisected by choosing a random pivot element and partitioning the list into those elements that are smaller than the pivot and those that are larger.) We write  $\hat{\alpha} \sim U[\alpha, \beta]$  if  $\hat{\alpha}$  has uniform distribution on  $[\alpha, \beta]$ .

We repeated each simulation experiment 1000 times to obtain data that is statistically meaningful. In each experiment the *ratio* between the maximum load generated by our



**Figure 5. Comparison of the average ratio for  $\hat{\alpha} \sim U[0.1, 0.5]$ ,  $\sigma = 1.0$**

algorithms and the uniform load distribution was recorded. The main focus of interest was the sample mean of the observed ratios for all three algorithms, but also the maximum and minimum ratio, as well as the sample variance was computed.

Simulations of this stochastic model for several choices of the interval  $[\alpha, \beta]$  and  $N = 2^k$ ,  $k \in \{5, 6, \dots, 20\}$ , were performed. As an example for the outcome of the experiments, the observed ratios for the case  $\hat{\alpha} \sim U[0.1, 0.5]$  are plotted in Figure 5. We chose the number of processors as consecutive powers of 2 to explore the asymptotic behavior of our load balancing algorithms (experiments with values of  $N$  that were not powers of 2 gave very similar results). The influence of the threshold parameter  $\sigma$  on the performance of Algorithm BA-HF in our stochastic model was also studied.

It is remarkable that the sample variance was very small in all cases except if an interval  $[\alpha, 2\alpha]$  with very small  $\alpha$  was chosen. Even more astonishingly, the outcome of each individual simulation was fairly close to the sample mean of all 1000 experiments. Especially for Algorithm HF the observed ratios were sharply concentrated around the sample mean for larger values of  $N$  (see Table 1).

In all experiments, Algorithm HF performed best and Algorithm BA-HF outperformed Algorithm BA. Usually, the observed ratios differed by no more than a factor of 3 for fixed  $N$ . Table 1 shows that the average and even the maximum ratios in our simulations were substantially smaller than our worst-case bounds (which were calculated according to Lemma 5, Theorems 7, 8, 2, and Theorem 6 from [1]). This indicates that the performance of the proposed load balancing algorithms will most likely be significantly better than the worst-case guarantees for practical applications.

As exemplified by Figure 5, the average ratio obtained from Algorithm HF was observed to be almost constant for the whole range of  $N = 32$  to  $N = 2^{20} = 1,048,576$

processors. Its exact value depended only on the particular choice of the interval  $[\alpha, \beta]$ . Only when the range for the bisection parameter was very small ( $\beta - \alpha$  smaller than 0.1), the observed ratios varied with the number of processors.

Finally, we studied the influence of the threshold parameter  $\sigma$  on the average-case performance of Algorithm BA-HF for the case  $\hat{\alpha} \sim U[0.1, 0.5]$ . We observed that the improvement of the average ratio was approximately 10% when  $\sigma$  increased from 1.0 to 2.0 and another 5% when  $\sigma = 3.0$ . So we can expect a sufficient balancing quality from Algorithm BA-HF using relatively small values of  $\sigma$ .

## 5. Conclusion

Based on previous results for the sequential Algorithm HF, we have derived three promising parallel algorithms for load balancing of problems with good bisectioners. While the sequential Algorithm HF has running-time  $O(N)$  for distributing a problem onto  $N$  processors, all three parallel algorithms require only running-time  $O(\log N)$  on  $N$  processors under reasonably general assumptions about the parallel machine architecture.

Algorithm PHF is a parallelization of Algorithm HF that produces the same load balancing as the sequential algorithm. It inherits the good performance guarantee of Algorithm HF, but the management of free processors is costly and global communication is required. Algorithm BA is inherently parallel and partitions a problem into  $N$  subproblems without requiring any global communication. Furthermore, the management of free processors is trivial for Algorithm BA. Our bound on the worst-case performance guarantee of Algorithm BA is not as good as for Algorithm PHF, but still constant for fixed  $\alpha$ . Algorithm BA-HF is a combination of Algorithm BA and Algorithm HF or PHF. By adjusting the threshold parameter  $\sigma$ , the worst-case performance guarantee of Algorithm BA-HF can be brought arbitrarily close to that of Algorithm PHF at the expense of inheriting drawbacks of Algorithm PHF.

Finally, we conducted extensive simulation experiments to determine the relative quality of the load balancing achieved by the individual algorithms in the average case. The performance in the average case was substantially better than the worst-case upper bounds for all three algorithms, and the balancing quality was the best for Algorithm HF and the worst for Algorithm BA in all experiments. In order to choose one of the proposed load balancing algorithms for application in practice, one must take into account the characteristics of the parallel machine architecture as well as the relative importance of fast running-time of the load balancing algorithm and of the quality of the achieved load balance. Our worst-case bounds and simulation results provide helpful guidance for this decision.

## References

- [1] S. Bischof, R. Ebner, and T. Erlebach. Load Balancing for Problems with Good Bisectioners, and Applications in Finite Element Simulations: Worst-case Analysis and Practical Results. SFB-Bericht 342/05/98A, SFB342, Institut für Informatik, TU München, 1998. <http://www.mayr.in.tum.de/berichte/1998/TUM-I9811.ps.gz>
- [2] S. Bischof, R. Ebner, and T. Erlebach. Load Balancing for Problems with Good Bisectioners, and Applications in Finite Element Simulations. In *Proceedings of the Fourth International EURO-PAR Conference on Parallel Processing EURO-PAR'98*, volume 1470 of LNCS, pages 383–389, Berlin, 1998. Springer-Verlag.
- [3] R. D. Blumofe and C. E. Leiserson. Scheduling Multithreaded Computations by Work Stealing. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science FOCS'94*, pages 356–368, Los Alamitos, 1994. IEEE Computer Society Press.
- [4] T. Bonk. A New Algorithm for Multi-Dimensional Adaptive Numerical Quadrature. In W. Hackbusch, editor, *Adaptive Methods – Algorithms, Theory and Applications: Proceedings of the 9th GAMM Seminar, Kiel, January 22–24, 1993*, pages 54–68. Vieweg Verlag, Braunschweig, 1993.
- [5] V. Heun. *Efficient Embeddings of Treelike Graphs into Hypercubes*. Berichte aus der Informatik. Shaker Verlag, Aachen, 1996.
- [6] R. Hüttl. *Ein iteratives Lösungsverfahren bei der Finite-Element-Methode unter Verwendung von rekursiver Substrukturierung und hierarchischen Basen*. PhD thesis, Institut für Informatik, Technische Universität München, 1996.
- [7] R. Hüttl and M. Schneider. Parallel Adaptive Numerical Simulation. SFB-Bericht 342/01/94 A, Technische Universität München, 1994.
- [8] J. JáJá. *An Introduction to Parallel Algorithms*. Addison-Wesley Publishing Company, Reading, MA, 1992.
- [9] R. M. Karp and Y. Zhang. Randomized parallel algorithms for backtrack search and branch-and-bound computation. *J. ACM*, 40(3):765–789, July 1993.
- [10] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms*. The Benjamin/Cummings Publishing Company, Redwood City, CA, 1994.
- [11] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays • Trees • Hypercubes*. Morgan Kaufmann Publishers, San Mateo, CA, 1992.
- [12] H. Regler and U. Rude. Layout optimization with Algebraic Multigrid Methods (AMG). In *Proceedings of the Sixth Copper Mountain Conference on Multigrid Methods, Copper Mountain, April 4–9, 1993*, Conference Publication, pages 497–512. NASA, 1993. Also available as technical report SFB 342/11/93 A, TU München.
- [13] T. Schnekenburger and G. Stellner, editors. *Dynamic Load Distribution for Parallel Applications*. TEUBNER-TEXTE zur Informatik. Teubner Verlag, Stuttgart, 1997.
- [14] B. A. Shirazi, A. R. Hurson, and K. M. Kavi, editors. *Scheduling and Load Balancing in Parallel and Distributed Systems*. IEEE Computer Society Press, Los Alamitos, CA, 1995.