A Detailed Analysis of Random Polling Dynamic Load Balancing

Peter Sanders
LS Informatik für Ingenieure und Naturwissenschaftler
Universität Karlsruhe, Germany

Abstract

Dynamic load balancing is crucial for the performance of many parallel algorithms. Random Polling, a simple randomized load balancing algorithm, has proved to be very efficient in practice for applications like parallel depth first search. This paper presents a detailed analysis of the algorithm taking into account many aspects of the underlying machine and the application to be load balanced. It derives tight scalability bounds which are for the first time able to explain the superior performance of Random Polling analytically. In some cases, the algorithm even turns out to be optimal. Some of the proof-techniques employed might also be useful for the analysis of other parallel algorithms.

1 Introduction

Load Balancing is one of the central issues in parallel computing. Since for many applications it is almost impossible to predict how much computation a given subproblem involves, a dynamic load balancing (DLB) strategy is necessary which is able to keep the processors busy without incurring an undue overhead.

DLB comes in many guises. This paper is concerned with a very simple yet important model of the problem domain. Initially there is only one large root problem. Subproblems are generated by splitting existing problems into two parts, but nothing is known about the relative size of the two parts or their interactions. The only thing the load balancer knows about a subproblem is whether it is exhausted or not.

One application domain for which this is a useful model is parallel depth first tree search. Search trees are often very irregular and the size of a subtree is hard to predict, but it is easy to split the search stack into two parts. Also, interactions between the subtrees often follow the tree structure (e. g. reporting results) or they are hard to exploit by a load balancer anyway (e. g. broadcasting of the current best solution or accessing distributed hash tables). Note that depth first tree traversal is a central aspect of many AI and OR applications and of parallel functional and logical programming languages.

This paper focuses on *Random Polling* (RP), a simple yet effective randomized DLB scheme. Every processing element (PE) works on at most one subproblem at a time. A PE whose subproblem is exhausted polls randomly determined PEs until it finds one which is busy. The busy PE splits its subproblem and transmits one part to the idle PE.

Our approach towards analyzing RP is somewhat atypical. We do not present implementation results or comparisons between a variety of new and old algorithms as it is usual in many papers with practical orientation. This has been done elsewhere

[3, 5, 2, 6, 15, 14]. Nor do we delve deep into a particular detail that is most beautiful, difficult, interesting or tractable from a theoretical point of view. Instead, we try to throw light on as many aspects of the algorithm as possible; hoping that this approach helps to understand and implement the algorithm in a variety of settings.

The remainder of this introduction gives basic definitions and an overview of related work. Section 1.3 gives some basic results about asymptotic behavior in a probabilistic setting. Section 2 contains the analysis of Random Polling which is complemented by a modification for the SIMD programming model in Section 3 where polling is handled in a different and interesting way. Due to space constraints some of the more technical derivations had to be omitted. They can be found in [13].

1.1 Definitions

The analysis considers a MIMD computer consisting of n identical PEs numbered 0 through $n-1$ which interact by exchanging messages through a network of diameter $d(n)$. The size w of the root problem is measured in units of sequential execution time. Subproblems are derived from the root problem by subsequent splitting operations and can be represented by a message of length at most $l(w)$. When a subproblem has been split $h(w)$ times it is assumed to have been reduced to some atomic size $g(w)$.

The performance of a DLB scheme can be assessed by analyzing a problem without interactions between subproblems where DLB is the only source of parallelization overhead. Natural performance measures are the parallel execution time $T_{\mathrm{par}}(n, w)$ or the efficiency

$$E = \frac{w}{nT_{\mathrm{par}}(n, w)} \tag{1}$$

But the complexity of discussing bivariate functions can be avoided by fixing E and solving Equation (1) for w yielding the *isoefficiency function* $w(n)$. This function is a convenient measure for the degree of scalability of an algorithm. For a more detailed discussion see [6].

Since RP is a randomized algorithm, a meaningful analysis has to be a probabilistic one. We use the following notion of behavior *with high probability.*

Definition 1 ([10]) *A random variable $X(n)$ is in $O(f(n))$* with high probability *or $X(n) \in \tilde{O}\,(f(n))$ for short iff*

$$\exists c > 0, n_0 > 0 : \forall \beta \geq 1, n \geq n_0 : \mathbf{P}\left[X(n) > c\beta f(n)\right] \leq n^{-\beta}$$

(In [13] it is shown that behavior with high probability is a stronger notion than the more customary notion of average case behavior for our purposes.)

1.2 Related work

In [4] it is proved that for $d(n) \in O(1)$ and $\{l(w), g(w)\} \subseteq O(1)$ RP has an isoefficiency function in $O(n^2 \log n)$ with high probability. Much tighter is the result in [5]: If $\{l(w), g(w)\} \subseteq O(1)$ and $h(w) \in O(\log w)$, the isoefficiency function of RP is in $O(nd(n)(\log n)^2)$ on the average. This already indicates a quite good scalability. But it falls short of explaining why RP is in practice more efficient than a deterministic algorithm introduced in the same paper which has an isoefficiency in $\Theta(nd(n) \log n)$.

Another randomized DLB algorithm is based on dynamic tree-embeddings into butterfly networks or hypercubes [8, 11]. For $\{l(w), g(w)\} \subseteq O(1)$ it has an isoefficiency function in $O(nh(n))$ with high probability (for butterflies and hypercubes) which is asymptotically optimal. However, the algorithm has no notion of granularity control resulting in high memory requirements and a possibly quite small upper bound on the achievable efficiency due to communication overhead. Another interesting result from [8] is that no deterministic tree embedding with the same performance can exist.

1.3 Some basic results

A useful property of behavior with high probability is that it allows very simple conclusions on the maximum of random variables whose behavior with high probability is known:

Theorem 1 ([13]) *Let $X_1(n) \in \tilde{O}(f_1(n)), \ldots, X_m(n) \in \tilde{O}(f_m(n))$ be random variables where m is at most polynomial in n. Then*

$$\max_{i=1}^{m} X_i(n) \in \tilde{O}\left(\max_{i=1}^{m} f_i(n)\right)$$

This result has a special case of particular importance for parallel algorithms: The run time of a parallel algorithm is determined by the slowest PE. If the runtime of individual PEs is known with high probability Theorem 1 makes it easy to estimate the overall run time.

Theorem 2 cites a frequently used result about coin flipping experiments.

Theorem 2 (Chernoff bounds) *Let the random variable X represent the number of heads after n independent flips of a loaded coin where the probability for a head is p. Then [10, 7]:*

$$\mathbf{P}\left[X \leq (1-\epsilon)np\right] \leq e^{-\epsilon^2 np/3} \text{ for } 0 < \epsilon < 1 \tag{2}$$

$$\mathbf{P}\left[X \geq \alpha np\right] \leq e^{\left(1-\frac{1}{\alpha}-\ln\alpha\right)\alpha np} \text{ for } \alpha > 1 \tag{3}$$

2 Analysis

Figure 1 shows pseudocode for a generic RP dynamic load balancer. All PEs execute the same program with the exception that PE 0 initially gets all the work.[1] Idle PEs poll randomly selected PEs for work and reject requests they receive. Busy PEs cycle between doing work and servicing at most one request.

Note that a busy PE will not block if no requests are imminent nor can it be swamped by requests without being able to do "useful" work. In addition, some protocol for termination detection is necessary which is not considered here since it is not a bottleneck if implemented properly.

Based on the above algorithm, Section 2.1 isolates a subproblem which captures the probabilistic character of RP in a mathematically simpler question which is then answered in Section 2.2. This information is used to derive upper bounds for the isoefficiency of RP which are complemented by lower bounds in Section 2.4. Finally Sections 2.5 and 2.6 discuss the validity of assumptions frequently made about the quality of the splitting function and the influence of network contention.

[1] It is advisable to improve the initial PE utilization using an appropriate initialization scheme [13].

```
initialize PE 0 with the root problem
WHILE NOT finished DO for all PEs in parallel (asynchronously)
    IF subproblem is empty THEN
        REPEAT
            send a request R to a randomly determined PE
            wait for a reply and reject any incoming requests
        UNTIL R is not rejected
        reinitialize subproblem from incoming message
    WHILE subproblem is not empty DO
        IF there is an incoming request THEN
            split subproblem
            asynchronously send one part to the initiator of the request
        do some work on subproblem
```

Figure 1: Pseudocode for RP

2.1 Framework of the analysis

The starting point is the definition of efficiency, $E = \frac{w}{nT_{\mathrm{par}}}$. For any $\gamma \in [0, 1/2)$ we can set $T_{\mathrm{par}} = T_{<\gamma} + T_{\geq\gamma}$ where $T_{<\gamma}$ is the length of all time intervals during which less than γn PEs are idle. If we neglect the time to test for a request[2] and assume that γn active PEs are delayed servicing requests of the idle PEs we get

$$T_{<\gamma} < \frac{w}{n(1-2\gamma)}$$

since in this time the active PEs can process the entire problem.

During $T_{\geq\gamma}$ there will be at least $\frac{\gamma n}{T_{\mathrm{req}}}$ work requests per time unit if T_{req} is the time needed for a work request. Let the random variable $K(n, h(w))$ denote the number of work requests necessary such that every subproblem has been split at least $h(w)$ times. Then

$$T_{\geq\gamma} \leq \frac{K(n, h(w))T_{\mathrm{req}}}{\gamma n} + g(w)$$

because after time $\frac{K(n,h(w))T_{\mathrm{req}}}{\gamma n}$ every subproblem is reduced to an atomic size $g(w)$. Now the efficiency can be estimated.

$$E \geq \frac{w}{n\left(\frac{w}{n(1-2\gamma)} + \frac{K(n,h(w))T_{\mathrm{req}}}{\gamma n} + g(w)\right)} = \frac{w}{\frac{w}{(1-2\gamma)} + \frac{K(n,h(w))T_{\mathrm{req}}}{\gamma} + ng(w)} \tag{4}$$

2.2 The order of $K(n, h(w))$

Under the reasonable assumption that there is at least a constant number of atomic work units for each PE (i. e. $w \in g(w)\Omega(n)$) the asymptotic behavior of $K(n, h(w))$ can be derived.

Lemma 1 *Let the random variable $K_t(n, h(w))$ denote the number of requests necessary to hit a particular subproblem "t" $h(w)$ times; Then $K_t(n, h(w)) \in \tilde{O}\left(nh(w)\right)$ if $w \in g(w)\Omega(n)$.*

[2] If incorporated into the analysis, it would turn out that the test implies an upper bound on the efficiency. However, the test can often be implemented very efficiently and intervals between tests can be made arbitrarily large without affecting the asymptotic scalability. The maximal efficiency can therefore be made as close to 1 as desired.

Proof: We need to find a c such that for all $\beta \geq 1$ and sufficiently large n

$$P := \mathbf{P}\left[K_t(n, h(w)) > c\beta n h(w)\right] \leq n^{-\beta}$$

or

$$\mathbf{P}\left[\text{after } c\beta n h(w) \text{ requests: } (\#\text{ of requests for } t) < h(w)\right] \leq n^{-\beta}.$$

Since the requests are independent and subproblem t is hit with the uniform probability $\frac{1}{n}$, Theorem 2 (Equation (2)) is applicable. By writing $h(w)$ as $\left(1 - \left(1 - \frac{1}{c\beta}\right)\right)(c\beta n h(w))\frac{1}{n}$ we get

$$P \leq \exp\left[-\left(1 - \frac{1}{c\beta}\right)^2 \frac{c\beta h(w)}{3}\right]$$

Since $w \in g(w)\Omega(n)$, $h(w)$ is in $\Omega(\log n)$ because even a perfect splitting function would always leave a subproblem not in $O(g(w))$ after less than logarithmically many splits. So, there is a constant $d > 0$ such that $h(w) \geq d \ln n$ for sufficiently large n. Using $\beta \geq 1$ we can further estimate:

$$P \leq \exp\left[-\left(1 - \frac{1}{c}\right)^2 \frac{c\beta d \ln n}{3}\right] = n^{-\beta\left(1-\frac{1}{c}\right)^2 \frac{cd}{3}} \leq n^{-\beta}$$

if $c \geq 1 + \frac{3+\sqrt{12d+9}}{2d}$.

■

There are only $O(n)$ subproblems, and therefore Theorem 1 allows us to conclude that the asymptotic behavior of $K(n, h(w))$ with high probability is the same as the behavior of $K_t(n, h(w))$:

Corollary 1 *$K(n, h(w)) \in O(nh(w))$ with high probability if $w \in g(w)\Omega(n)$*

2.3 The isoefficiency of RP

If we neglect network contention (see Section 2.6 for a discussion) then the request delay T_{req} is in $O(d(n)l(w))$, and together with Relation (4) and Corollary 1 we can conclude that there is a constant c such that for sufficiently large n and w:

$$E \geq \frac{w}{\frac{w}{(1-2\gamma)} + \frac{cnh(w)d(n)l(w)}{\gamma} + ng(w)} \tag{5}$$

with high probability. An immediate observation is that for $g(w) > ch(w)l(w)d(n)$ the scalability is dominated by the atomic grain size. In this case it may not be necessary to bother about routing delays, quality of splitting function or message lengths. If $h(w)l(w) \in \Omega(w)$ or $g(w) \in \Omega(w)$ we have $\lim_{n\to\infty} E = 0$ according to our estimate. So, for large $h(w)l(w)$ or $g(w)$ RP may not be scalable at all. Indeed, for $h(w) \in \Omega(w)$ or $g(w) \in \Omega(w)$ the problem contains a sequential component of size $\Omega(w)$ and no scalable parallel algorithm is possible.

For $\max\{h(w)l(w), g(w)\} \notin \Omega(w)$ we can choose $\gamma < \frac{1-E}{2}$ and get $\lim_{w\to\infty} E = 1$ i. e. for sufficiently large w any desired efficiency can be achieved. The degree of scalability for RP can be assessed by fixing E and solving for the isoefficiency function $w(n)$. For three characteristic cases we get the following results: (For details refer to [13].)

- $h(w)l(w) \in O((\log w)^a)$, $a \geq 1$, $g(w) \in O((\log w)^b)$, $b \geq 0$:

$$w(n) \in \tilde{O}\left(n \max\{d(n)(\log n)^a, (\log n)^b\}\right) \tag{6}$$

- $h(w)l(w) \in O(w^\alpha)$, $0 < \alpha < 1$, $g(w) \in O(w^\beta)$, $0 \leq \beta < 1$:

$$w(n) \in \tilde{O}\left(\max\{(nd(n))^{\frac{1}{1-\alpha}}, n^{\frac{1}{1-\beta}}\}\right) \tag{7}$$

- $h(w) \in O(\log w)$, $l(w) \in O(w^\alpha)$ with $\alpha \geq 0$:

$$w(n) \in \tilde{O}\left(\max\{(nd(n)\log n)^{\frac{1}{1-\alpha}}, n^{\frac{1}{1-\beta}}\}\right) \tag{8}$$

2.4 Lower bounds

In order to judge the quality of RP and its analysis it is interesting to compare the results with lower bounds on the scalability of dynamic load balancing in general and RP in particular. Based on the fact that some piece of work has to travel through the entire network a lower bound of $\Omega(nd(n))$ for the isoefficiency of dynamic load balancing is derived in [5].[3] Using our more detailed model of the application, two additional limitations on performance can be identified: First, some processor has to process a subproblem of size $\Omega(g(w))$. This information can be used to derive a lower bound for the isoefficiency (for details refer to [13]) which shows that the granularity is the limiting factor if $g(w) \in \Omega(h(w)l(w)d(n))$. For coarse grained problems RP is therefore asymptotically optimal. Second, in some way the root problem has to be split $\Omega(\log n)$ times. If we assume that splitting is at least as expensive as copying $\Omega(l(w))$ bytes, we get a lower bound of $\Omega(l(w)\log n)$ on the execution time. Similar to the argument above we can conclude that for $d(n) \in O(1)$ and $h(w) \in O(\log w)$ RP is asymptotically optimal.

In RP splits are coupled to global communications so that we get a lower bound of $\Omega(d(n)\log n)$ on the execution time and of $\Omega(nd(n)\log n)$ on the isoefficiency. Therefore, for bounded message lengths ($l(w) \in O(1)$) and good splitting functions ($h(w) \in O(\log w)$) our analysis turns out to be tight.

2.5 Estimating $h(w)$

Splitting a subproblem of size v produces two subproblems with sizes αv and $(1-\alpha)v$ where $0 < \alpha \leq 1/2$. If it is guaranteed that α is bounded from below by a positive constant then it is fairly straightforward to show that $h(w) \in O(\log w)$ [5], making the analysis of RP somewhat simpler.

However, this assumption is not always warranted. In depth first tree search for example, a very popular splitting function splits the search tree by distributing the successors of the root-node between the two subproblem. If the degree of tree nodes is bounded by a constant, $h(w)$ is proportional to the height of the tree. If the tree is sufficiently "dense" the height is indeed logarithmic in w. But, there are search algorithms where both the height of the tree and w are polynomial in some input measure [9] and therefore figures like $h(w) \sim \sqrt{w}$ are quite conceivable.

Although there are more sophisticated splitting functions for search trees [12], it is an open question in which cases these functions can guarantee that $h(w) \in O(\log w)$ (perhaps in some probabilistic sense). But even if this works, the analysis for general $h(w)$ may help to decide whether the additional expense for a more sophisticated splitting function is worth the effort.

Another example for trees of very irregular shape are computation trees induced by functional programs. According to [1], it is quite difficult to come up with a useful splitting function for those trees. The same problem is to be expected for logical programming languages.

[3] Actually this is not true for pathological networks like one where $n/2$ PEs are fully connected and another $n/2$ PEs are arranged as a linear array ("clique plus tail").

2.6 The request delay T_{req}

It is often assumed that on networks like crossbars, grids, hypercubes or multistage networks a message of length $l(w)$ can be routed in time $O(d(n)l(w))$. But for Random Polling this is certainly not true in the strict worst case sense. For example, when all idle PEs choose to poll the same PE there will be considerable network contention around the target PE. Here we want to outline why the assumption is justified for Random Polling in a probabilistic sense.

Since there cannot be more messages than idle PEs, there are at most n messages at a time. The requests have independent randomly determined destinations. The contention due to requests headed for the same PE can be estimated using Theorem 2 (Equation 3):

$$\begin{aligned}\mathbf{P}\left[\#\text{requests for PE } i \ \geq \beta \ln n\right] &\leq e^{\left(1-\frac{1}{\beta \ln n}-\ln(\beta \ln n)\right)(\beta \ln n)n\frac{1}{n}} \\ &\leq e^{-\beta \ln n} = n^{-\beta}\end{aligned}$$

for $\beta \geq 1$ and $\ln \ln n \geq 2$. Since answers to requests never have conflicting destinations we can conclude that the number of messages contending for PE i is in $\tilde{O}\left(\log n\right)$. And Theorem 1 says that the same is true for the PE with the maximum number of contending messages.

So, we have a routing problem with at most n messages of length at most $l(w)$ where $\tilde{O}\left(\log n\right)$ messages contend for one destination. In [7] there are a number of results which indicate that under these circumstances

$$T_{\text{req}} \in \tilde{O}\left(d(n)l(w)\right) \tag{9}$$

for grids, hypercubes and various multistage networks with logarithmic diameter. Note that this may be a conservative estimate since it may be possible to pipeline messages so that routing could be completed in time $\tilde{O}\left(d(n) + l(w)\right)$. This depends on the bisection bandwidth of the network and the fraction of messages with constant size (requests and rejects).

The situation is a little bit more complicated for networks with constant (or sublogarithmic) diameter since here the number of contending messages appears to become a limiting factor. But a closer look reveals that this is not true. Due to the asynchronous nature of RP an asymptotically significant delay can only occur when some PE gets more than $O(h(w))$ of the requests over the *entire* runtime of the program. The $\tilde{O}\left(nh(w)\right)$ total requests are again independent and uniformly distributed and we can apply Theorem 2. We also know that $h(w) \in \Omega(\log n)$. Using an argument as before, it can be derived that the maximum contention is in $\tilde{O}\left(h(w)\right)$.

For some networks, contention *is* a problem due to their limited bisection bandwidth. For example, on buses or trees it can only be said that $T_{\text{req}} \in O(nl(w))$. The subsequent analysis works with $T_{\text{req}} \in O(d(n)l(w))$. The result for other cases is easy to obtain by substituting appropriate values for d.

3 Random Polling on SIMD machines

The Random Polling algorithm as described above is not directly applicable to a SIMD setting since a PE cannot independently issue a request. This problem can be solved by introducing separate load balancing phases. A load balancing phase is triggered when the fraction of idle PEs rises above some constant γ. This condition is tested periodically by counting the number of idle PEs; an operation that is quite efficient on many SIMD-machines. (Still, the interval between tests needs to be in $\Omega(d(n))$ if it shall not become a bottleneck).

Now we can use a very similar argument as in Section 2.2. Due to space constraints we only present the key ideas. If any subproblem is hit with a probability of at least γ during each phase, and if the phases are independent, then Theorem 2 can be used to show that there is a constant c, such that for sufficiently large n, $\frac{c\beta}{\gamma}h(w)$ load balancing phases are sufficient to hit every subproblem at least $h(w)$ times with probability $1-n^{-\beta}$ for any $\beta \geq 1$; i.e. the number of required load balancing phases is in $\tilde{O}(h(w))$.

An important point in this argument is that only the phases need to be independent, while the requests of one phase can be dependent. This can be exploited by using a very special kind of globally determined random permutation that is easy to route on the given hardware. The only requirement is that from the point of view of a particular subproblem all PEs are equally likely to be a communication partner. For example, on a $d_1 \times \cdots \times d_k$ meshes or torus we can determine a random vector $(r_1, \ldots, r_k)$ and let the PE with coordinate $(i_1, \ldots, i_k)$ communicate with PE $(i_1+r_1 \bmod d_1, \ldots, i_k+r_k \bmod d_k)$ which yields a "shift"-permutation that can be routed in $d(n)$ steps without collisions. For the special case $k = \log n$ and $d_1 = \cdots = d_{\log n} = 2$ we get an efficient scheme for the hypercube which involves a single **XOR** for address calculation.

So, for many interconnection networks, a load balancing phase takes time in $O(d(n)l(w))$. This also holds if we count the time which passes until it is detected that the percentage of idle PEs has risen above γ, as part of the load balancing phase. Using this construction it can be shown that $T_{\geq\gamma}$ is in $\tilde{O}(d(n)l(w)h(w))$ and the results from Section 2.3 also hold for the SIMD case.

4 Conclusions

The simple randomized dynamic load balancing algorithm Random Polling has often proved useful in practice. This paper explains its good performance under a variety of circumstances using the notion of isoefficiency function. The influence of low quality splitting functions and nonconstant message lengths is investigated. Also, the paper helps to justify the simplifying assumption that network contention is no problem as long as the bisection bandwidth is high enough. New tight scalability bounds are derived for bounded message lengths and good splitting functions. Random Polling is asymptotically optimal if the time needed for the delivery of a message depends only on the problem size or if the communication overhead is dominated by the atomic grain size of the problem. Finally, a new SIMD variant of Random Polling is described for which it can be proved that an "almost deterministic" communication pattern is as good as a truly random permutation.

References

[1] G. Aharoni, A. Barak, and Y. Farber. An adaptive granularity control algorithm for the parallel execution of functional programs. *Future Generation Computing Systems*, 9:163–174, 1993.

[2] R. Feldmann, P. Mysliwietz, and B. Monien. Distributed game tree search on a massively parallel system. In *Data structures and efficient algorithms: Final report on the DFG special joint initiative*, volume LNCS 594, pages 270–288, 1991.

[3] R. Finkel and U. Manber. DIB— A distributed implementation of backtracking. *ACM Trans. Prog. Lang. and Syst.*, 9(2):235–256, Apr. 1987.

[4] R. M. Karp and Y. Zhang. Parallel algorithms for backtrack search and branch-and-bound. *Journal of the ACM*, 40(3):765–789, 1993.

[5] V. Kumar and G. Y. Ananth. Scalable load balancing techniques for parallel computers. Technical Report TR 91-55, University of Minnesota, 1991.

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

[7] T. Leighton. *Introduction to Parallel Algorithms and Architectures.* Morgan Kaufmann, 1992.

[8] T. Leighton, M. Newman, A. G. Ranade, and E. Schwabe. Dynamic tree embeddings in butterflies and hypercubes. In *ACM Parallel Processing Symposium*, pages 224–234, 1989.

[9] J. Pearl. *Heuristics.* Addison Wesley, 1984.

[10] S. Rajasekaran. Randomized algorithms for packet routing on the mesh. In L. Kronsjö and D. Shumsheruddin, editors, *Advances in Parallel Algorithms*, pages 277–301. Blackwell, 1992.

[11] A. Ranade. Optimal speedup for backtrack search on a butterfly network. *Mathematical Systems Theory*, pages 85–101, 1994.

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

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

[14] P. Sanders. Massively parallel search for transition-tables of polyautomata. In *Parcella 94, VI. International Workshop on Parallel Proccessing by Cellular Automata and Arrays*, Potsdam, 1994.

[15] P. Sanders. Portable parallele Baumsuchverfahren: Entwurf einer effizienten Bibliothek. In *Transputer Anwender Treffen (to appear)*, Aachen, 1994.