TECHNISCHE UNIVERSITÄT MÜNCHEN

INSTITUT FÜR INFORMATIK

Sonderforschungsbereich 342:
Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

On-Line Scheduling of Parallel Jobs with Runtime Restrictions

Stefan Bischof, Ernst W. Mayr

TUM-I9810
SFB-Bericht Nr. 342/04/98 A
April 98

TUM–INFO–04-I9810-130/1.–FI

Anforderungen an: Prof. Dr. A. Bode
Sprecher SFB 342
Institut für Informatik
Technische Universität München
D-80290 München, Germany

Druck: Fakultät für Informatik der
Technischen Universität München

On-Line Scheduling of Parallel Jobs with Runtime Restrictions

Stefan Bischof Ernst W. Mayr
Institut für Informatik
Technische Universität München
D-80290 München, Germany

{bischof|mayr}@informatik.tu-muenchen.de

http://wwwmayr.informatik.tu-muenchen.de/

April 3, 1998

Abstract

Consider the execution of a parallel application that dynamically generates parallel jobs with specified resource requirements during its execution. Generally, there is not sufficient knowledge about the running times and the number of jobs generated in order to precompute a schedule for such applications. Rather, the scheduling decisions have to be made on-line during runtime based on incomplete information. We present several on-line scheduling algorithms for a variety of interconnection topologies that use some a priori information about the job running times or guarantee a good competitive ratio that depends on the runtime ratio of all generated jobs. All algorithms presented have optimal competitive ratio up to small additive constants, and are easy to implement.

1 Introduction

The efficient operation of parallel computing systems requires the best possible use of the resources that a system provides. In order to achieve an effective utilization of a parallel machine a smart coordination of the resource demands of all currently operating applications is necessary. Consequently, the task of a scheduler is a clever assignment of the resources, most prominently the processors, to the jobs being processed. For the case of sequential jobs, i.e., jobs that require exactly one processor for execution, the involved scheduling problems have been studied intensively for decades [BEP+96]. But in many situations the problem arises to find a schedule for a set of parallel jobs [FR95, FR96, BEP+96]. Such a set could, for example, be a parallel query execution plan generated by the query optimizer of a parallel database management system [Rah96, GI97].

The model studied in this paper assumes that each parallel job demands a fixed number of processors or a specified sub-system of a certain size and topology (depending on the underlying structure of the parallel machine considered) for its execution. It is not possible to run a parallel job on fewer processors than requested, and additional processors will not decrease the running time. This reflects the common practice that the decision on the number of processors is made before a job is passed to the scheduler based on other resource requirements like memory, disk-space, or communication intensity. The processors must be allocated exclusively to a job throughout its execution, and a job cannot be preempted or restarted later. This is a reasonable assumption because of the large overhead for these activities in parallel machines. Furthermore, there may be precedence constraints between the jobs. A job can only be executed if all of its predecessors have already completed execution. Most frequently, precedence constraints arise from data dependencies such that a job needs the complete input produced by other jobs before it can start computation.

We are concerned with on-line scheduling throughout this paper to capture the fact that complete a priori information about a job system is rarely available. However, it has been shown [FKST93, Sga94] that the worst-case performance of any deterministic or randomized on-line algorithm for scheduling parallel job systems with precedence constraints and arbitrary running times of the jobs is rather dismal, even if the precedence constraints between the jobs are known in advance. Therefore, we study the case that there is some a priori knowledge about the execution times of the individual jobs but the dependencies are unknown to the scheduler.

We study three different gradations for this additional knowledge. The first model of runtime restrictions requires that all job running times are equal and that this fact is known to the on-line scheduler. We give a level-oriented on-line algorithm for this problem that repeatedly schedules a set of available jobs using bin packing and collects all jobs that arrive during a phase for execution in the next phase. We show that this algorithm is 2.7-competitive if the First Fit heuristic is used. Due to a lower bound of 2.691 for every deterministic on-line scheduler, our algorithm is almost optimal. Our algorithm can be used for parallel systems that support arbitrary allocation of processors to jobs and 1-dimensional arrays. For hypercube connected machines, we present a very similar, optimal on-line scheduling algorithm with competitive ratio 2.

We then explore the entire bandwidth between unit and arbitrary execution times and capture the variation of the individual job running times by a characteristic parameter that we call **runtime ratio** (the quotient of the longest and shortest running time). Our second model postulates that the runtime ratio of a job system is reasonably small and that the on-line scheduler knows the shortest execution time (but not the runtime ratio itself). We give a family of job systems with runtime ratio $T_R \geq 2$ that bounds the competitive ratio of any deterministic on-line scheduler by $(T_R + 1)/2$ from below. We note that the structure of the dependency graph is an out-forest in all of our lower bound proofs. Our bounds remain valid even if the scheduler knows the actual runtime ratio in advance. An on-line scheduler designated RRR (Restricted Runtime Ratio) for parallel systems supporting arbitrary allocations is described, and we demonstrate a competitive ratio of $T_R/2 + 4$ for this algorithm for any job system with runtime ratio $\leq T_R$. Therefore, the RRR algorithm is nearly optimal up to a small additive constant. The assumption that the shortest execution time is known to the on-line scheduler can be dropped without much loss of competitive performance. We present a modified algorithm called RRR_ADAPTIVE for this third model, and show it to be $T_R/2 + 5.5$ competitive.

The remainder of this paper is organized as follows. In Section 2 we introduce our scheduling model, some notation and definitions, as well as basic techniques for analyzing on-line scheduling algorithms. We then discuss previous and related work on on-line scheduling of parallel jobs in Section 3. Section 4 presents nearly optimal on-line schedulers for jobs with unit execution time, whereas in Section 5 we study job systems where the ratio of the running times of the longest and shortest job is bounded. Again, we describe and analyze on-line scheduling algorithms that are optimal up to small additive constants. We conclude giving some directions for future research in Section 6.

2 Preliminaries

Let N denote the number of processors of the parallel computer-system at hand. A *(parallel) job system* is a non-empty set of jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_m\}$ where each job specifies the type and size of the sub-system that is necessary for its execution together with precedence-constraints among the jobs in $\mathcal{J}$ given as a partial order $\prec$ on $\mathcal{J}$. If $J_a \prec J_b$ then J_b cannot be scheduled for execution before J_a is completed. A *task* is a job that requires one processor for execution, and a job system that only contains tasks is a *sequential* job system.

A *schedule* for a job system $(\mathcal{J}, \prec)$ is an assignment of the jobs to processors and start-times such that:

- each job is executed on a sub-system of appropriate type and size,
- all precedence-constraints are obeyed,
- each processor executes at most one job at any time,
- jobs are executed non-preemptively and without restarts.

The interconnection topology of the parallel computer-system may impose serious restrictions on the *job types* that can be executed efficiently on a particular machine. On a hypercube, for example, it is reasonable to execute jobs only on subcubes of a certain dimension rather than on an arbitrary subset of the processors. On the other hand, a number of interconnection networks do not restrict the allocation of processors to parallel jobs. For example, the CLOS-network of the very popular *IBM RS/6000 SP* system, which uses an oblivious buffered wormhole routing strategy, justifies the assumption that the running time of a job only weakly depends on a specific processor allocation-pattern (see [AG94, p. 512] for a short description of this system and [SSA+94] for in-depth information on its interconnection network). Therefore, we treat the various types of interconnection networks separately.

The *complete model* assumes that a job J_a requests n_a processors ($1 \leq n_a \leq N$) for execution and any subset of processors of size n_a may be allocated. The terminology has been chosen in analogy to a complete graph on N nodes. The *r-dimensional hypercube* (see Figure 1) consists of $N = 2^r$ processors, labeled from 0 to $N-1$, and has $r2^{r-1}$ point-to-point communication links. Two processors are connected iff the binary representations of their labels (an r-bit string) differ in exactly one bit. As a consequence, each processor is directly connected to $r = \log_2 N$ other processors (see [Lei92] for properties of hypercubes). A job J_a can only request a d_a-dimensional subcube ($0 \leq d_a \leq r$) for its execution.

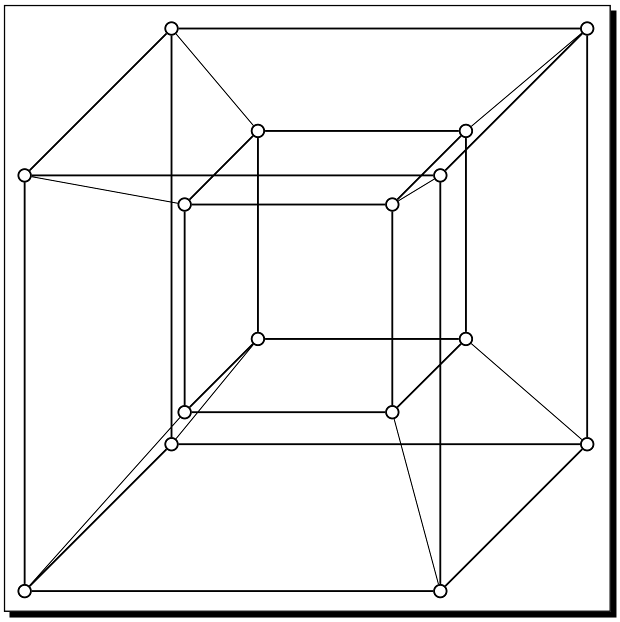

Figure 1: 4-dimensional hypercube

Another topology frequently used for parallel computing is the *r-dimensional array* with side-lengths $(N_1, N_2, \ldots, N_r)$, $N_i \geq 2$ for $i = 1, 2, \ldots, r$ (also called r-dimensional grid or mesh). The label of a processor is an r-dimensional vector $x = (x_1, x_2, \ldots, x_r)$ with $0 \leq x_i < N_i$ for $i = 1, 2, \ldots, r$. Two processors x and y are connected iff $\|x - y\| = 1$. Note that hypercubes form the subclass of arrays with side-length 2 in every dimension. Eligible job types are sub-arrays with side-lengths $(N_1', N_2', \ldots, N_r')$, $1 \leq N_i' \leq N_i$. The dimension of a job can be less than r if one or more of the N_i' are equal to 1.

It is always possible to transform a job system $(\mathcal{J}, \prec)$ into a directed acyclic graph $D = (\mathcal{J}, E)$ with $(J_a, J_b) \in E \Leftrightarrow J_a \prec J_b$. Removing all transitive edges from D we obtain the *dependency graph* induced by $(\mathcal{J}, \prec)$ (see Figure 3 on page 13 for an example). We call two jobs J_a and J_b *dependent* if $J_a \prec J_b$ or $J_b \prec J_a$, and *independent* otherwise. We shall use the terms *dependency* and *precedence-constraint* interchangeably in this paper. The *length of a path* in the dependency graph induced by $(\mathcal{J}, \prec)$ is defined as the sum of the running times of the jobs along this path. A path is called *critical* if its length is maximum among all paths in the dependency graph induced by $(\mathcal{J}, \prec)$. A job is *available* if all predecessors of this job have completed execution. An on-line scheduling algorithm is only aware of available jobs and has no knowledge about their successors. We assume that the on-line scheduler receives knowledge about a job as soon as the job becomes available. This event, however, may depend on earlier scheduling decisions.

Table 1: Frequently used notations

T_{opt}	Length of an optimal off-line schedule for $(\mathcal{J}, \prec)$
T_{ALG}	Length of a schedule for $(\mathcal{J}, \prec)$ generated by Algorithm ALG
$T_{\max}$	Maximal length of any path in the dependency graph induced by $(\mathcal{J}, \prec)$
$t_{\min}$	Minimal running time of any job in $\mathcal{J}$
$t_{\max}$	Maximal running time of any job in $\mathcal{J}$
$\lvert S \rvert$	Length of a schedule S
$T_{<\alpha}$	Total time of a schedule for $(\mathcal{J}, \prec)$ when the efficiency is less then α, $0 \leq \alpha \leq 1$

The *work* of a job is defined as the number of requested processors, multiplied by its running time. A schedule preserves the work of a job if the processor-time product for this job is equal to its work. The *efficiency* of a schedule at any time t is the number of busy processors at time t divided by N. In general, the running time of a job is also unknown to the on-line scheduler and can only be determined by executing a job and measuring the time until its completion. In Section 4, though, we study the case of unit execution times and therefore restrict the on-line model there to the case of unknown precedence-constraints.

Throughout the paper we use the notations in Table 1 (cf. [Sga94, FKST93]) for a given job system $(\mathcal{J}, \prec)$. To simplify our presentation, we do not attach the job system or schedule as arguments to the notations in Table 1. The relationships should always be clear from the context. Further notation is introduced when needed.

Our goal is to generate schedules with *minimum makespan*, i.e. to minimize the completion time of the job finishing last. We evaluate the performance of our on-line scheduling algorithms by means of competitive analysis [ST85]. A deterministic on-line algorithm ALG is called *c-competitive* if $T_{\text{ALG}} \leq cT_{\text{opt}}$ for **all** job systems and **arbitrary** N. The infimum of the values $c \in [1, \infty]$ for which this inequality holds is called the *competitive ratio* of ALG. The competitive ratio clearly is a worst-case measure. It is intended to compare the performance of different on-line algorithms that solve the same problem, since it is in general impossible to compute an optimal solution without complete knowledge of the problem instance. An *optimal* on-line algorithm is one with a best possible competitive ratio.

The following two lemmata provide useful tools for the competitive analysis of our scheduling algorithms.

LEMMA 2.1 Let S be a schedule for a job system $(\mathcal{J}, \prec)$ such that the work of each job is preserved. Let $0 \leq \alpha_1 \leq \alpha_2 \leq 1$ and $\beta \geq 0$. Suppose that the efficiency of S is at least α_1 at all times and $T_{<\alpha_2} \leq \beta T_{\mathrm{opt}}$. Then

$$|S| \leq \left(\beta + \frac{1-\alpha_1\beta}{\alpha_2}\right) T_{\mathrm{opt}}.$$

See [Sga94] for a proof of this lemma.

LEMMA 2.2 Consider a schedule for a job system $(\mathcal{J}, \prec)$. Then there exists a path of jobs in the dependency graph induced by $(\mathcal{J}, \prec)$ such that whenever there is no job available to be scheduled, some job of that path is running.

This lemma is due to GRAHAM [Gra66, Gra69]. The proof given there still holds for parallel jobs since it uses only the structure of the dependency graph.

3 Previous and Related Work

Extensive work on non-preemptive on-line scheduling of parallel jobs with or without precedence-constraints was done by FELDMANN, KAO, SGALL and TENG [FKST93, Sga94, FST94]. However, these results for general parallel job systems are bad news for users of parallel computers since they show that no deterministic on-line scheduler for N processors can have competitive ratio better than N. That is, the competitive ratio is asymptotically unbounded, and even randomization cannot improve this unsatisfactory situation substantially.

One possibility to improve the performance is to restrict the maximum job size to λN processors, $0 < \lambda < 1$. Given this restriction it has been shown that the GREEDY algorithm is optimal with competitive ratio $1 + \frac{1}{1-\lambda}$. Setting $\lambda = 1/2$, for example, yields a 3-competitive algorithm. This result holds for any type of parallel machine. Another alternative is the use of *virtualization*. This means that a parallel job J_a which requests n_a processors is executed on a smaller number of processors n'_a by the use of simulation techniques with a predetermined increase in running time. Under the assumption of proportional slowdown (the running time of a job is enlarged by the factor n_a/n'_a) it can be shown that there is an optimal on-line scheduler for the complete model with competitive ratio $2+\Phi$, where $\Phi = (\sqrt{5}-1)/2$ is the golden ratio. This improves a previous off-line result of WANG and CHENG [WC92] with asymptotic performance guarantee 3. For the hypercube, an algorithm with competitive ratio $O(\log N/\log\log N)$ has been given, and similar results hold for arrays. The two approaches just described can be combined to yield an optimal on-line scheduler with competitive ratio $2 + \frac{\sqrt{4\lambda^2+1}-1}{2\lambda}$ for the complete model.

Both approaches, though, have a severe drawback that arises due to the memory requirements of parallel jobs. Restricting the maximum size of a job to λN processors can thus severely restrict the problem size that can be solved on a particular machine. This is often unacceptable in practice because solving large problems is the main reason for the use of parallel computers besides solving problems fast. Virtualization may be impossible or prohibitively expensive if such memory limitations exist.

The job systems used in the lower bound proofs in [FKST93, Sga94] for the general case reveal an unbounded ratio of the running times of the longest and shortest job. Therefore, we think it necessary to study the influence of the individual running times on the competitive ratio of on-line schedulers for our scheduling problem. To gain insight into this relationship it is only natural to start with unit execution times as is done in Section 4. It turns out that the problem becomes manageable with small constant competitive ratio even if nothing is known about the precedence constraints.

To fill the gap between these two extremes — totally unrelated running times versus unit execution times — we identify the *runtime ratio* (the ratio of the running time of the longest and shortest job) as the distinctive parameter of a job system for the achievable competitive ratio. Our results for the proposed on-line schedulers in Section 5 demonstrate a smooth, linear transition of the competitive ratio from the case of unit execution times to unrelated execution times that is governed by the runtime ratio. The importance of this parameter has also been demonstrated recently in [CM96] for **off-line** scheduling of jobs with multiple resource demands, both malleable (allow for virtualization with proportional slowdown) and non-malleable.

Although we are interested in on-line scheduling, it might be appropriate to briefly mention some complexity results for the corresponding off-line problems. Not surprisingly, almost any variant of these scheduling problems is $\mathcal{NP}$-hard. Błażewicz, Drabowski, and Węglarz [BDW86] have proved that it is strongly $\mathcal{NP}$-hard to compute optimal schedules for parallel job systems with unit execution times and no dependencies if N is part of the problem instance. For any fixed N they showed that the problem can be solved in polynomial time. Furthermore, it is known [GJTY83] that the problem is $\mathcal{NP}$-hard for sequential job systems with precedence constraints that are the disjoint union of an in-forest and and an out-forest. The scheduling problem for parallel job systems with arbitrary job running times and without dependencies is strongly $\mathcal{NP}$-hard for every fixed $N \geq 5$ [DL89]. If precedence constraints consisting of a set of chains are involved, the problem of computing an optimal 2-processor schedule for a parallel job system is also strongly $\mathcal{NP}$-hard [DL89].

4 Jobs with Unit Execution Time

In this section, we restrict our model to the case where all jobs have the same execution time. When the dependency graph is known to the scheduler this problem has been intensively studied by Garey, Graham, Johnson and Yao [GGJY76]. We show that similar results hold in an on-line environment, where a job is available only if all its predecessors have completed execution.

4.1 Complete Model

The Level algorithm collects all jobs that are available from the beginning. Since available jobs are independent we can easily transform the problem of scheduling these jobs to the Bin Packing problem: the size of a job divided by N is just the size of an item to be packed, and the time-steps of the schedule correspond to the bins (see [CGJ96] for a survey on Bin Packing). Let Pack be an arbitrary Bin Packing heuristic. We parameterize the Level algorithm with Pack to express the fact that a schedule for a set of independent jobs is generated according to Pack. Thereafter, the available jobs are executed as given by this schedule. Any jobs that become available during this execution phase are collected by the algorithm. After the termination of all jobs of the first level a new schedule for all available jobs is computed and executed. This process repeats until there are no more jobs to be scheduled.

Algorithm Level(Pack) :
while *not all jobs are finished*
    do
    $\mathcal{A} := \{J \in \mathcal{J} | J \textit{ is available}\}$;
    schedule all jobs in $\mathcal{A}$ *according to* Pack;
    wait until all scheduled jobs are finished;
od

First, we use the Next-Fit (NF) bin-packing heuristic for scheduling on each level. NF packs the items in given order into a so-called active bin. If an item does not fit into the active bin, the active bin is closed and never used again. A previously empty bin is opened and becomes the next active bin.

Theorem 4.1 Level(NF) is 3-competitive.

Proof: The number of iterations of the while-loop is exactly the length of a critical path in the dependency graph. There are two possibilities for each level:

1. The partial schedule for this level has length 1. Let T_1 denote the number of levels of this type.

2. The partial schedule for this level has length ≥ 2. By the packing rule of NF it is clear that the average efficiency of 2 consecutive time-steps in such a partial schedule is $> 1/2$. From this we conclude that the average efficiency of all time-steps but maybe the last one is $> 1/2$. Let T_2 denote the number of final time-steps with efficiency $< 1/2$ in partial schedules for levels of this type.

Since $T_1 + T_2 \leq T_{\max} \leq T_{\mathrm{opt}}$ we can apply Lemma 2.1 with $\alpha_1 = 1/N$, $\alpha_2 = 1/2$, $\beta = 1$, yielding:

$$T_{\mathrm{LEVEL(NF)}} \leq \left(3 - \frac{2}{N}\right) T_{\mathrm{opt}}. \qquad \blacksquare$$

Since NF can be implemented to run in linear time (in the number of items to be packed) the scheduling overhead is very low when NF is used to compute partial schedules. Now we use the First-Fit (FF) bin-packing heuristic instead of NF to achieve a better competitive ratio with only a modest increase of the scheduling overhead. FF in contrast to NF considers all partially filled bins as possible destinations for the item to be packed. An item is placed into the first (lowest indexed) bin into which it will fit. If no such bin exists, a previously empty bin is opened and the item is placed into this bin. It has been shown [Joh74] that FF has time-complexity $\Theta(n \log n)$ for a list of n items.

THEOREM 4.2 LEVEL(FF) is 2.7-competitive.

The proof of this theorem uses the weighting function from [GGJY76]. Let $W : [0,1] \rightarrow [0, 8/5]$ be defined as follows:

$$W(\alpha) = \begin{cases} \frac{6}{5}\alpha & \text{for } 0 \leq \alpha \leq \frac{1}{6}, \\ \frac{9}{5}\alpha - \frac{1}{10} & \text{for } \frac{1}{6} < \alpha \leq \frac{1}{3}, \\ \frac{6}{5}\alpha + \frac{1}{10} & \text{for } \frac{1}{3} < \alpha \leq \frac{1}{2}, \\ \frac{6}{5}\alpha + \frac{4}{10} & \text{for } \frac{1}{2} < \alpha \leq 1. \end{cases}$$

Figure 2 depicts the graph of W.

We need the following results from [GGJY76]:

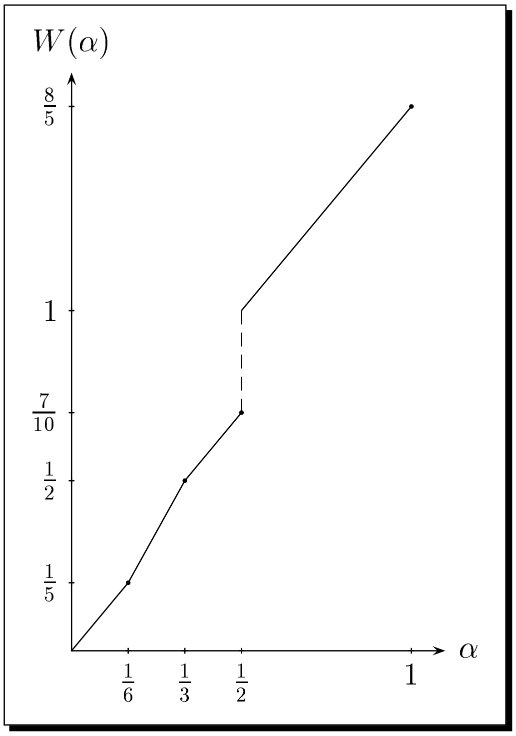

Figure 2: Weighting function for the analysis of Bin Packing

Lemma 4.3 Let B denote a set of items with total size ≤ 1. Then

$$\sum_{b \in B} W(\text{size}(b)) \leq \frac{17}{10}.$$

If all sizes are $\leq 1/2$, then

$$\sum_{b \in B} W(\text{size}(b)) \leq \frac{3}{2}.$$

Theorem 4.4 If L is a list of items with sizes ≤ 1, then

$$FF(L) < \sum_{x \in L} W(\text{size}(x)) + 1.$$

Together with the above lemma this theorem provides the best known upper bound for the number of bins used by first-fit. If L^* is the number of bins used in an optimal packing of L, then first-fit uses at most $\lceil (17/10)L^* \rceil$ bins. Now we are ready to prove Theorem 4.2:

Proof: Let $\mathcal{J}$ be a job system with unit execution time and arbitrary precedence-constraints. We define

$$\overline{W}(\mathcal{J}) = \sum_{j \in \mathcal{J}} W(\text{size}(j)).$$

Thus $\overline{W}(\mathcal{J})$ is the total weight of all job sizes. Let l be the number of levels of the job system. For $1 \le i \le l$ let U_i be the set of jobs of each level. By Theorem 4.4 we can upper bound the length of the partial schedule for each level i, $1 \le i \le l$, generated by LEVEL(FF):

$$T_{\text{LEVEL}(FF)}(U_i) < \overline{W}(U_i) + 1.$$

We can think of an optimal packing of $\mathcal{J}$ with the dependencies removed as a partition of $\mathcal{J}$ into $\mathcal{J}^*$ sets each of which has total size ≤ 1. Applying Lemma 4.3 yields $\overline{W}(\mathcal{J}) \le \frac{17}{10}\mathcal{J}^*$. Together with the fact that the length of the optimal schedule for $\mathcal{J}$ without dependencies cannot be longer than the length of the optimal schedule for $\mathcal{J}$ we conclude:

$$T_{\text{LEVEL(FF)}} = \sum_{i=1}^{l} T_{\text{LEVEL(FF)}}(U_i) < \sum_{i=1}^{l} (\overline{W}(U_i) + 1) = \overline{W}(\mathcal{J}) + l \le 1.7\, T_{\text{opt}} + l.$$

Since $l = T_{\max} \le T_{\text{opt}}$, the result follows. ■

The competitive ratio 2.7 of LEVEL(FF) is nearly optimal. To show this, we give an asymptotic lower bound of 2.691 for the competitive ratio of each deterministic on-line scheduling algorithm. For the sake of clarity, we first prove a slightly weaker lower bound of $2.69\, T_{\text{opt}} - 4$ for the length of a schedule generated by a deterministic on-line scheduler. Using Salzer numbers we refine this construction to derive the asymptotic lower bound.

Fix $N \in \mathbb{N}$, $N \ge 7 \cdot 1806$, and let

$$A := \left\lfloor \frac{N}{2} \right\rfloor + 1, \qquad B := \left\lfloor \frac{N}{3} \right\rfloor + 1,$$
$$C := \left\lfloor \frac{N}{7} \right\rfloor + 1, \qquad D := N - A - B - C - 1.$$

The job system (see Figure 3) consists of $l \ge 4$ levels with one chain of $l - 4$ tasks and l jobs of size A, $l - 1$ jobs of size B, $l - 2$ jobs of size C, $l - 3$ jobs of size D.

Additional dependencies are assigned dynamically by an adversary depending on which parallel job of each level is scheduled last by the on-line algorithm. This is possible because the on-line scheduler cannot distinguish between the parallel jobs on the same level. The optimal schedule has length l and is shown in Figure 4. Here, the parallel job with successors is scheduled first on each level. Contrary to the optimal solution, the on-line scheduler is forced to schedule and execute all jobs on one level to make the jobs on the next level available. The schedule generated by LEVEL(FF) is thus the best possible on-line schedule (see Figure 5) and has length

$$l + \left\lceil \frac{l-1}{2} \right\rceil + \left\lceil \frac{l-2}{6} \right\rceil + \left\lceil \frac{l-3}{42} \right\rceil + (l - 4) > 2.69\, l - 4,$$

if $2 \nmid (l-1)$, $6 \nmid (l-2)$, and $42 \nmid (l-3)$. It is easy to see that any $l \in \mathbb{N}$ with $42 \mid l$ fulfills the above conditions.

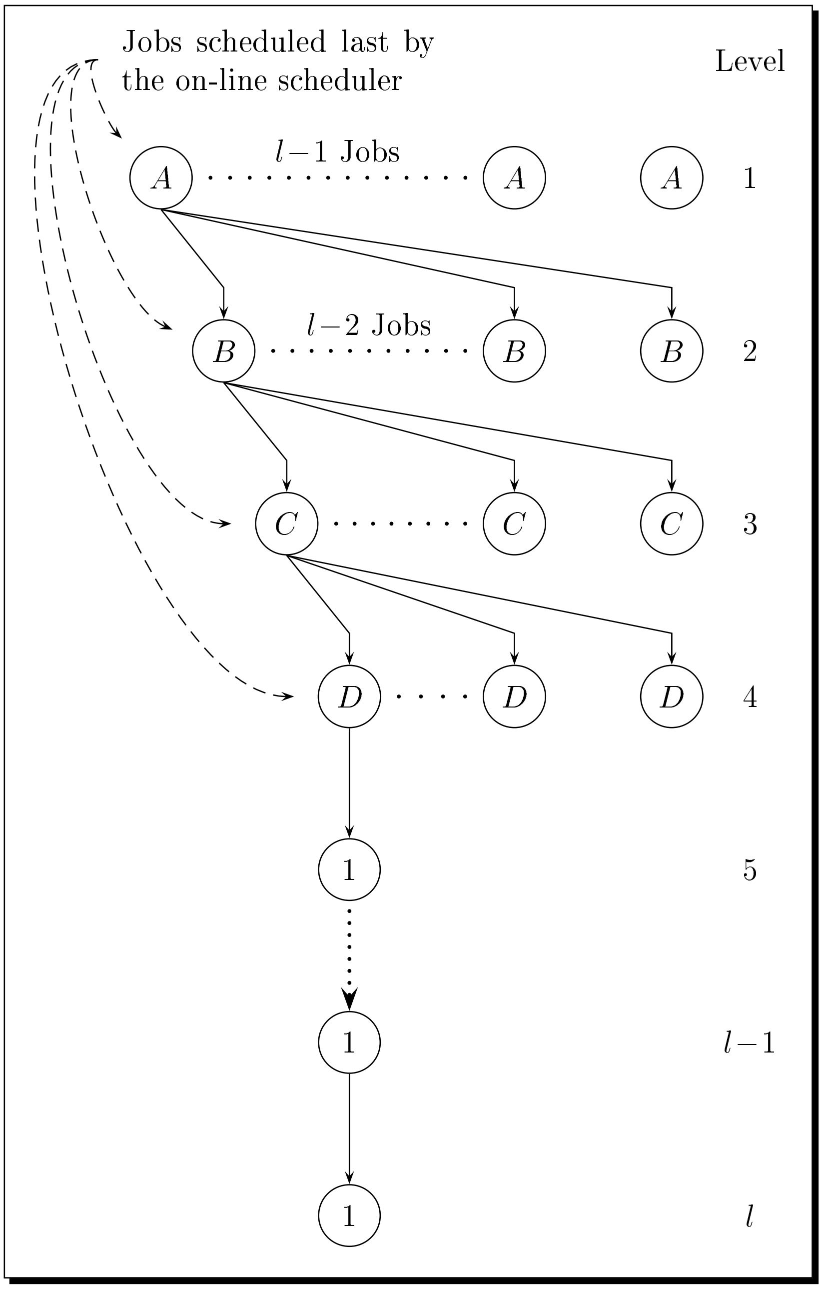

Figure 3: Job system used in lower bound proof

The following sequence $(t_i)_{i\in\mathbb{N}}$ was investigated by SALZER [Sal47]:

$$t_1 = 2,$$
$$t_{i+1} = t_i(t_i - 1) + 1, \quad \text{for} \quad i \geq 1.$$

The first five numbers of this sequence are $2, 3, 7, 43, 1807$. Closely related is the following series:

$$h_\infty = \sum_{i=1}^{\infty} \frac{1}{t_i - 1} > 1.69103. \tag{1}$$

There are two basic relations for the Salzer numbers that can be derived inductively from their definition:

$$\sum_{i=1}^{k} \frac{1}{t_i} + \frac{1}{t_{k+1} - 1} = 1,$$
$$\prod_{i=1}^{k} t_i = t_{k+1} - 1.$$

Let $A_i = \lfloor N/t_i \rfloor + 1$, $1 \leq i \leq k$, be the sizes of the parallel jobs on the first k levels. Setting $A_{k+1} = N - \sum_{i=1}^{k} A_i - 1$, we can conclude that

$$A_{k+1} < \frac{N}{t_{k+1} - 1} - 1,$$
$$A_{k+1} \geq \frac{N}{t_{k+1} - 1} - (k+1).$$

It is easy to see that $t_{k+1} - 1$ jobs of size A_{k+1} can be scheduled in one time-step on $N - 1$ processors. To ensure that no more than $t_{k+1} - 1$ jobs of size A_{k+1} can be co-scheduled on N processors we choose $N > (k+1)(t_{k+2} - 1)$. The job system again consists of $l \geq k+1$ levels with one chain of $l - (k+1)$ tasks and $l - (i-1)$ jobs of size A_i, $1 \leq i \leq k+1$. Dependencies are assigned dynamically as above. The length of the optimal schedule is l, whereas every schedule generated by a deterministic on-line scheduler has length at least

$$\sum_{i=1}^{k+1} \left\lceil \frac{l - (i-1)}{t_i - 1} \right\rceil + l - (k+1).$$

From this and (1) we see that the competitive ratio can be brought arbitrarily close to $1 + h_\infty$ for $k \to \infty$, $l = \omega(k)$.

The competitive ratio of Level(FF) can be improved if the maximum size of a job is restricted to $\lfloor N/2 \rfloor$.

Theorem 4.5 Level(FF) is 2.5-competitive, if no job requests more than half of the total number of processors.

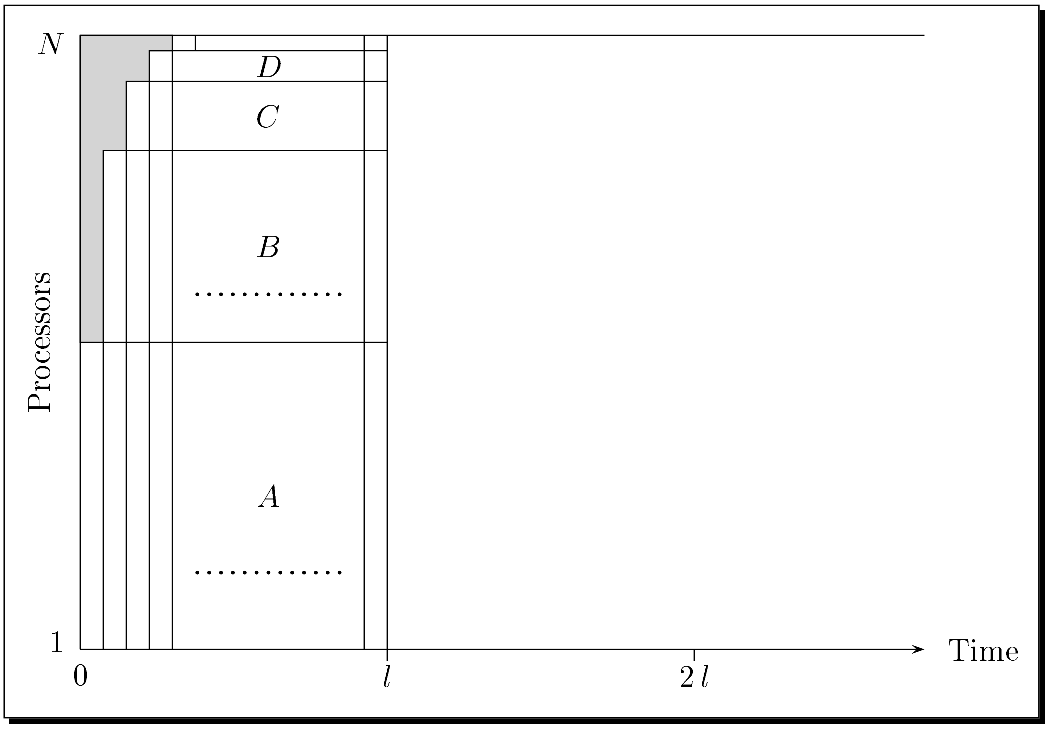

Figure 4: Optimal schedule

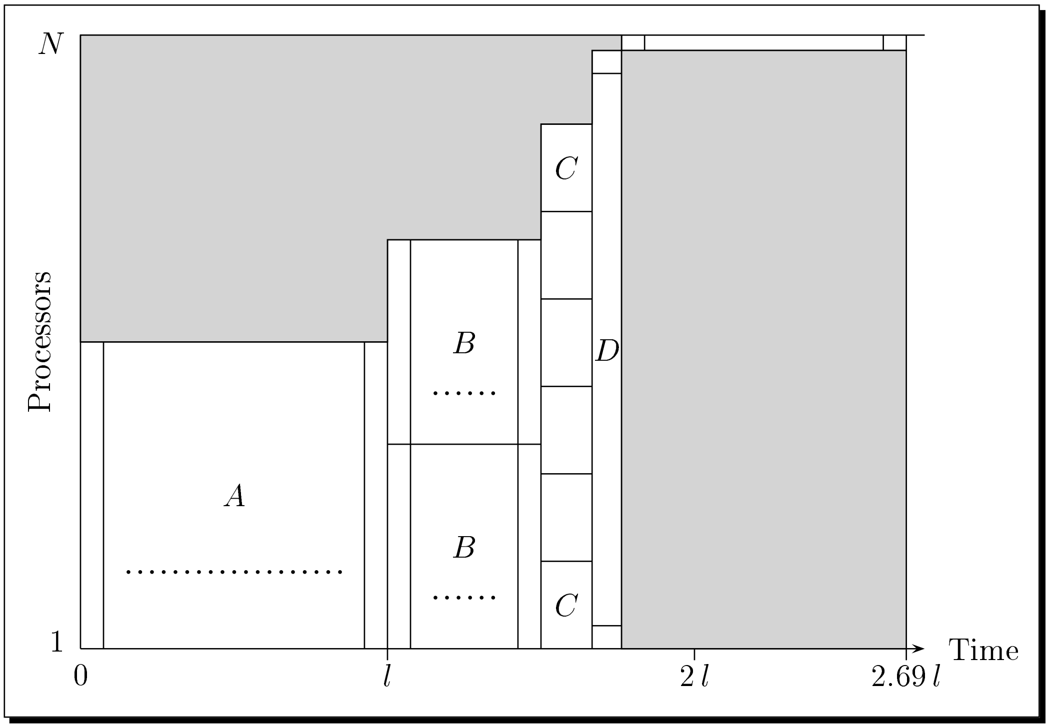

Figure 5: On-line schedule generated by Level(FF)

Proof: Analogous to the proof of Theorem 4.2 using the second inequality of Lemma 4.3. ■

Similarly to the unrestricted case, an asymptotic lower bound > 2.4 for the competitive ratio of any deterministic on-line scheduler for this problem can be derived. Further restrictions of the maximum job size might yield somewhat better competitive ratios for the LEVEL algorithm, but this situation is already handled well by the GENERIC algorithm in [FKST93, Sga94] which achieves competitive ratio $1 + 1/(1 - \lambda)$, if no job requests more than λN, $0 < \lambda < 1$, processors. For example, $\lambda = 1/2$ yields competitive ratio 3 for the GENERIC algorithm that is valid for job systems with arbitrary execution times.

We also remark that the results of this subsection remain valid if we assume a 1-dimensional array of length N as interconnection topology instead of using the complete model, since the BIN PACKING algorithms assign consecutive processors to the jobs and the assignments in different time-steps are independent from each other.

4.2 Hypercube

In this subsection we study the problem of on-line scheduling parallel job systems with arbitrary precedence-constraints and unit execution times for hypercube connected parallel computers.

It is not difficult to schedule a set of independent parallel jobs each of which requests a subcube of a certain dimension. First, we sort the jobs by size in non-increasing order. To avoid fragmentation, we use only normal subcubes for job execution:

DEFINITION 4.6 A k-dimensional subcube is called *normal*, if the labels of all its processors differ only in the last k positions.

For each time-step of our schedule we allocate jobs from the head of the sorted list to normal subcubes while there are unscheduled jobs left and the hypercube is not completely filled. If the time step is full we have to add a new time-step to our schedule (if there are any unscheduled jobs left).

It is easy to see that the efficiency of this schedule for independent jobs is 1 in all time-steps except possibly the last. We refer to this strategy as PACK_HC. The algorithm for job systems with arbitrary dependencies is just the LEVEL algorithm using PACK_HC instead of a BIN PACKING heuristic.

THEOREM 4.7 LEVEL_HC is an optimal deterministic on-line scheduler with competitive ratio 2.

Proof: The number of iterations of the while-loop is exactly the length of a critical path in the dependency graph. Thus $T_{<1} \leq T_{max} \leq T_{\mathrm{opt}}$. Since the efficiency of the schedule is at least $1/N$ all the time, we have for fixed N by Lemma 2.1:

$$T_{\text{LEVEL\_HC}} \leq \left(1 + \frac{1 - 1/N}{1}\right) T_{\mathrm{opt}} = \left(2 - \frac{1}{N}\right) T_{\mathrm{opt}}.$$

It remains to show that no deterministic on-line scheduler can achieve a better competitive ratio. To this behalf, we use a job system similar to the preceding subsection. It uses $N-1$ levels with $N+1$ tasks on each level. Again, the dependencies are assigned dynamically by an adversary according to the decisions of the deterministic on-line scheduler. The job from level i, $1 \leq i \leq N-2$, scheduled last by the on-line scheduler is designated to be predecessor of all jobs on level $i+1$. Therefore any on-line scheduler ALG needs at least 2 time-steps to schedule all jobs of one level. In an optimal schedule, the job with dependencies is scheduled first together with $N-1$ other jobs from the same level. The $N-1$ remaining jobs are scheduled in time-step N. This gives the desired lower bound for the competitive ratio:

$$\frac{T_{\text{ALG}}}{T_{\mathrm{opt}}} \geq \frac{2(N-1)}{N} = 2 - \frac{1}{N}. \qquad \blacksquare$$

Algorithm LEVEL_HC :
while *not all jobs are finished*
    do
    $\mathcal{A} := \{J \in \mathcal{J} | J \textit{ is available}\}$;
    schedule all jobs in $\mathcal{A}$ *according to* PACK_HC;
    wait until all scheduled jobs are finished;
od

The job system in the proof of Theorem 4.7 contains no parallel jobs and the hypercube structure isn't used at all. Therefore the derived lower bound is valid for any interconnection topology and sequential job systems as well as parallel job systems. Note that the structure of the dependency graph is an out-forest.

COROLLARY 4.8 No deterministic on-line algorithm for scheduling job systems with unit execution times and dependencies can have a competitive ratio better than 2.

Interestingly, this lower bound is identical to the lower bound proved by SHMOYS, WEIN, and WILLIAMSON [SWW95] for sequential job systems with arbitrary running times but without precedence-constraints.

5 Parallel Job Systems with Restricted Runtime Ratio

We have shown in the preceding section that on-line scheduling of parallel jobs with unit execution time and precedence-constraints is possible with small constant competitive ratio. On the other hand, if execution times are arbitrary, there exists no on-line scheduler with acceptable worst-case performance. It is only natural to explore the case that job runtimes are restricted by some criterion other than unit execution time in order to achieve a respectable competitive ratio.

For a set of jobs $\mathcal{J}$ we therefore define the *runtime ratio* $\mathrm{RR} = \mathrm{RR}(\mathcal{J}) := t_{\max}/t_{\min}$. In this section we study the problem of on-line scheduling parallel job systems with dependencies where the runtime ratio is restricted by a parameter $T_R \geq 1$ which is not known to the on-line scheduler. This problem often arises in practice when upper and lower bounds for the running time of a job are known in advance but the actual running time is unknown. This situation also makes clear that the parameter T_R cannot be used as additional information for scheduling decisions by the on-line scheduler and is therefore not part of the problem instance. Indeed, our results show that this knowledge is not necessary for the on-line scheduler to achieve a near optimal competitive ratio that depends only on T_R.

In this paper, we study this problem for the complete model. First, we give a lower bound of $\max\left\{(T_R+1)/2, h_\infty + 1\right\}$ for the asymptotic competitive ratio of any deterministic on-line scheduler for this problem. For simplicity we normalize the time of the shortest job to 1. The job system used in this lower bound argument is very simple (see Figure 6) and consists of N layers with two tasks and one parallel job of size N on each layer. The parallel job depends on one of the tasks on the same layer and is predecessor of both tasks of the following layer. The task scheduled first by the on-line scheduler is assigned running time T_R and the remaining task runs for 1 unit of time and is predecessor of the parallel job. Clearly, the makespan of any schedule generated by an on-line scheduler is at least $N(T_R+1)$. If T_R is sufficiently large (e.g., $T_R \geq 2$), the optimal solution first schedules the critical path which has length $2N$ followed by the tasks of length T_R in parallel. The competitive ratio of any deterministic on-line scheduler is thus lower bounded by

$$\frac{N(T_R+1)}{2N+T_R} \quad \xrightarrow[N\to\infty]{} \quad \frac{T_R+1}{2}.$$

For small T_R, this bound is quite weak. But in this case, we can use the job system from the lower bound construction in subsection 4.1. Since the

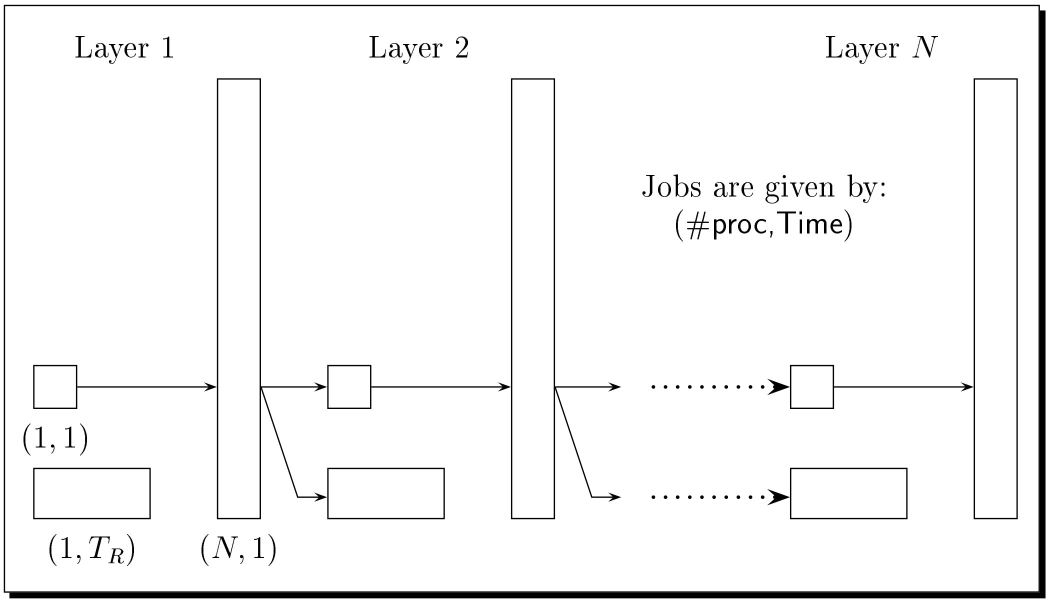

Figure 6: Difficult job system for RRR-scheduling

runtime ratio of this job system is 1 this yields an asymptotic lower bound of $h_\infty + 1$ for the competitve ratio.

We now describe an algorithm designated RRR that achieves competitive ratio $T_R/2+4$. A key feature of this algorithm is the distinction between *big* jobs that request more than half of the total number of processors and *small* jobs with size $\leq \lfloor N/2 \rfloor$. Let $\alpha := \alpha(t)$ denote the efficiency at time t. The RRR algorithm tries to keep the efficiency at least $1/2$ whenever possible. There are two reasons that hinder the RRR algorithm from achieving this goal. First, there might be no job available and second, there might be not enough processors available to schedule a big job. The second case is much more severe than the first one which can be handled by the Graham argument (cf. Lemma 2.2) without much loss of performance. Therefore, the RRR algorithm must prevent big jobs from being delayed too long in order to bound the fraction of the total schedule length with low efficiency. This is done by occasionally stopping to schedule small jobs, if all big jobs request more processors than currently available and the efficiency is below $1/2$.

We present two versions of the RRR algorithm. The first one assumes that $t_{\min}$ is a known quantity. Again, we normalize the running time of the shortest job to 1 and a *unit of time* refers to this normalized time quantum. In the second version we remove this assumption and employ an adaptive waiting-strategy to maintain a comparable competitive ratio. The RRR algorithm maintains two sets, L_1 and L_2, containing the available big respectively small jobs. We assume that any job that becomes available is immediately inserted into the appropriate set, and we will not state this activity explicitly in the pseudo-code description of our algorithms.

```
Algorithm RRR
while L1 not empty
        do
        schedule a big job exclusively;
od
while not all jobs are finished
        do
        while L2 not empty
                do
                schedule small jobs greedily;
        od
        if L1 not empty
           then
                if a big job can be scheduled
                   then
                        do it;
                   else
                        if α ≥ 1/2
                           then
                                wait for a scheduled job to finish;
                           else                          co start of a delay phase oc
                                collect small jobs that become available
                                during the next 2 units of time;
                                schedule those jobs greedily and
                                then wait for all scheduled jobs to finish;
                                while L1 not empty
                                        do
                                        schedule a big job exclusively;
                                od
                        fi
                fi
           else
                wait for next available job;
        fi
od
```

THEOREM 5.1 The RRR algorithm is $(T_R/2+4)$-competitive for any job system $(\mathcal{J}, \prec)$ and $\mathrm{RR}(\mathcal{J}) \le T_R$.

Proof: We partition the schedule generated by the RRR algorithm into 3 different kinds of phases:

1. Efficiency is at least $1/2$.
2. Efficiency is below $1/2$ and there is no job available.
3. Efficiency is below $1/2$ and the algorithm waits for the termination of all jobs.

We refer to the third type as a *delay phase* and denote the total time of each kind by $T_{\geq 1/2}$, T_{nojob}, and T_{delay} respectively. The total time of the RRR schedule that is spent in phases of type 1 and 2 can easily be bounded by $3\,T_{\text{opt}}$, because we have $T_{\geq 1/2} \leq 2\,T_{\text{opt}}$ by a straightforward area-argument and $T_{\text{nojob}} \leq T_{\max} \leq T_{\text{opt}}$ by Lemma 2.2.

It remains to show that $T_{\text{delay}} \leq (T_R/2+1)T_{\text{opt}}$. We define a *delayed job* as a big job that was available at the beginning of a delay phase. Let t_i denote the start time of delay phase i. First, we bound the length of a delay phase by T_R+2. If no small jobs become available during the first two units of time after the beginning of a delay phase, no more jobs are scheduled until all currently running jobs terminate. Since the running time of any job is no more than T_R, such a delay phase lasts at most time T_R. On the other hand, if small jobs become available during the first two units of time, then these are collected and scheduled greedily at time $t_i^s = t_i + 2$ (resp. $t_i^s < t_i + 2$ if all jobs running at time t_i terminate before two units of time have elapsed) in addition to those jobs still running at time t_i^s. If the total size of these small jobs is no more than the number of idle processors at time t_i^s, they can be scheduled immediately. Clearly, the length of a delay phase is bounded by $T_R + 2$ in this case. Should the total size of the small jobs exceed the number of idle processors at time t_i^s we can schedule enough small jobs to raise the efficiency above $1/2$ as long as small jobs that were collected during the interval $[t_i, t_i^s]$ are available. The time-span while the efficiency is at least $1/2$ is, of course, a phase of type 1 and not part of the delay phase. Clearly, the length of the second part of a delay phase is bounded by T_R and therefore the length of a delay phase is always bounded by $T_R + 2$.

Let d denote the number of delay phases in a schedule generated by the RRR algorithm. We distinguish two cases:

1. $d = 1$: We have to show that the optimal solution needs at least time 2. This follows immediately from the fact that each delayed job must have a predecessor in the job system because otherwise it would have been scheduled earlier.

2. $d > 1$: This case will be proven by constructing a chain of jobs in the dependency graph with total execution time at least $2d$. From that we have $T_{\text{opt}} \geq 2d$ and together with $T_{\text{delay}} \leq d(T_R + 2)$ the claim follows.

The construction of this chain proceeds as follows: Starting with an arbitrary delayed job that is scheduled after delay phase d we observe that there must be a small job that is ancestor of this delayed job and is scheduled immediately after the delayed jobs of delay phase $d-1$ (i.e. without having a small job as direct predecessor that is itself scheduled after the delayed jobs of delay phase $d-1$) because otherwise this delayed job would have been scheduled earlier. We add such a small job at the front of the chain.

To augment the chain, we state the possibilities for the direct predecessor of a small job that is scheduled by the RRR algorithm immediately after the delayed jobs of delay phase i:

Type 1: Delayed job of delay phase i or big job that is successor of a delayed job of delay phase i,

Type 2: Small job collected during delay phase i,

Type 3: Small job running from the beginning of delay phase i.

This is due to the fact that the RRR algorithm schedules all small jobs that are available by time t_i^s before the delayed jobs of delay phase i.

We continue the construction inductively according to these three possibilities. If there is a direct predecessor of Type 1 of the small job that is currently head of the list, we can repeat the initial construction step of the chain and add a delayed job and its small ancestor at the front of the chain. When there is no direct predecessor of Type 1 but a direct predecessor of Type 2, we add 2 more jobs at the front of the chain: the Type 2 job and a direct predecessor of this job that was running at the beginning of the delay phase during which this Type 2 job was collected. Finally, if there is only a direct predecessor of Type 3, we add this job at the front of the chain. The inductive construction stops as soon as the head of the chain is a small job that is scheduled before the delayed jobs of the first delay phase.

To complete the proof, we show that the total execution time of the jobs along this chain is at least $2d$. The construction of the chain starts with 2 jobs, a delayed job and its small ancestor. Since the minimum running time of any job is 1, these 2 jobs need at least 2 units of time for execution in any schedule. If the construction proceeds by adding a Type 1 job, the same argument applies. Continuing with a Type 2 job means that again 2 more jobs with were added to the chain. If a Type 3 job is encountered, we know that this job must have execution time at least 2 because it is direct predecessor of a small job that is scheduled immediately after the delayed jobs of the delay phase the Type 3 job belongs to. Thus, for each delay phase in the schedule generated by the RRR algorithm, the above construction adds jobs with total execution time at least 2 to the chain. ■

```
Algorithm RRR_Adaptive
i := 0;                                   co i counts the number of delay phases oc
while L_1 not empty
    do
    schedule a big job exclusively;
od
while not all jobs are finished
    do
    while L_2 not empty
        do
        schedule small jobs greedily;
    od
    if L_1 not empty
      then
          if a big job can be scheduled
            then
                do it;
            else
                if α ≥ 1/2
                  then
                      wait for a scheduled job to finish;
                  else                        co start of a delay phase oc
                      if i > 0
                        then
                            i := i + 1;
                            t^i_min := current minimum execution time;
                            collect small jobs that become available
                            for time ≤ 2 t^i_min;
                            schedule those jobs greedily and
                            then wait for all scheduled jobs to finish;
                        else
                            i := i + 1;
                            wait for all scheduled jobs to finish;
                      fi
                      while L_1 not empty
                          do
                          schedule a big job exclusively;
                      od
                fi
          fi
      else
          wait for next available job;
    fi
od
```

The assumption that $t_{\min}$ is known to the RRR algorithm can be dropped by employing an adaptive waiting strategy without much loss in competitive performance. We describe this adaptive version separately in order to keep our presentation modular. The modifications of the RRR algorithm are as follows: Since $t_{\min}$ is now unknown the RRR_ADAPTIVE algorithm doesn't collect small jobs during the first delay phase. In all following delay phases (if any), the algorithm calculates $t^i_{\min}$, the minimum execution time of any finished job up to the start of delay phase i. The duration during which small jobs are collected is now limited by $2\,t^i_{\min}$ (and, of course, by $t_{\max}$).

THEOREM 5.2 The RRR_ADAPTIVE algorithm is $(T_R/2 + 5.5)$-competitive for any job system $(\mathcal{J}, \prec)$ and $\mathrm{RR}(\mathcal{J}) \leq T_R$.

Proof: With the notation of the proof of Theorem 5.1 we conclude analogously that the above theorem holds for $d = 1$. If $d > 1$, we have

$$T_{\text{delay}} \leq dt_{\max} + 2\sum_{i=2}^{d} t^i_{\min}.$$

First, we show that $2\sum_{i=2}^{d} t^i_{\min} \leq 2\,T_{\text{opt}} - 2\,t_{\min}$. To see this, we observe that after delay phase i, $1 \leq i \leq d$, at least one delayed job has to be scheduled. Let $t^{d+1}_{\min} := t_{\min}$. The running time of such a delayed job is at least $t^{i+1}_{\min}$, since this job is executed before the start of delay phase $i+1$ (if $i < d$). Even in an optimal schedule all delayed jobs must be scheduled sequentially because they require more than half of the available processors for execution. Therefore:

$$2\,T_{\text{opt}} \geq 2\sum_{i=2}^{d+1} t^i_{\min} = 2\sum_{i=2}^{d} t^i_{\min} + 2\,t_{\min}. \tag{2}$$

As in the proof of Theorem 5.1 we can construct a chain of jobs in the dependency graph with total execution time at least $(2d-1)t_{\min}$. The only difference in the construction is that there is no collection of small jobs during the first delay phase and therefore a Type 3 job might only run for time $t_{\min}$ in this delay phase. This yields another lower bound for the optimum schedule length:

$$T_{\text{opt}} \geq (2d-1)t_{\min}. \tag{3}$$

From (2) and (3) we conclude:

$$\begin{aligned}
T_{\text{delay}} &\leq dt_{\max} + 2\sum_{i=2}^{d} t^i_{\min} \\
&\leq dt_{\max} + 2\,T_{\text{opt}} - 2\,t_{\min} \\
&\leq \frac{(d-1/2)T_R}{2d-1} T_{\text{opt}} + (T_R/2 - 2)t_{\min} + 2\,T_{\text{opt}} \\
&\leq \left(\frac{T_R}{2} + \frac{5}{2} - \frac{2}{T_R}\right) T_{\text{opt}}.
\end{aligned}$$

If the number of delay phases of a schedule generated by the RRR_ADAPTIVE algorithm is less than $(T_R + 1)/2$, we can derive a better upper bound:

$$T_{\text{delay}} \le \left(d + 2 - \frac{2}{T_R}\right) T_{\text{opt}}.$$

However, this bound is useful for a posteriori analysis only, since the number of delay phases can be arbitrarily large. Since the total schedule time that is spent in phases of type 1 and 2 (cf. proof of Theorem 5.1) is bounded by $3\,T_{\text{opt}}$, the proof is complete. ■

Clearly, both algorithms can easily compute the runtime ratio $RR(\mathcal{J})$ for any scheduled job system $\mathcal{J}$. From this, we can bound the actual performance for the generated schedules:

$$T_{\text{RRR}} \le (RR(\mathcal{J})/2 + 4)\, T_{\text{opt}},$$
$$T_{\text{RRR\_ADAPTIVE}} \le (RR(\mathcal{J})/2 + 5.5)\, T_{\text{opt}}.$$

For practical purposes it is desirable to have tools that allow to control the performance of a scheduler in addition to worst-case guarantees such as the competitive ratio. Let T_{big} be the sum of the execution times of all big jobs in $\mathcal{J}$, and let W_{total} denote the total work of all jobs. Then we have the following lower bound for the length of an optimal schedule:

$$T_{\text{opt}} \ge \max\left\{W_{\text{total}}/N, T_{\max}, T_{\text{big}}\right\}.$$

Again, our on-line algorithms can compute W_{total} and T_{big} during the scheduling process. Assuming that the on-line scheduler has knowledge of the predecessor/successor relationships (which usually will be the case after all jobs have been scheduled), $T_{\max}$ can be computed by searching a longest path in the dependency graph. The quotient of the length of the on-line schedule and the above lower bound is then an upper bound for the performance of our on-line schedulers.

6 Conclusion and Open Problems

We have presented and analyzed several on-line scheduling algorithms for parallel job systems. It has become evident that *runtime restrictions* improve the competitive performance achievable by on-line schedulers. Therefore, if enough a priori knowledge on job running times is available to bound the runtime ratio of a job system our schedulers can guarantee a reasonable utilization of the parallel system. But even without any such knowledge the RRR_ADAPTIVE algorithm produces schedules that are almost best possible from a worst-case point of view. All on-line algorithms considered in this paper are computationally simple, and thus the scheduling overhead involved can safely be neglected, provided that the system has suitable means to deliver the necessary load information.

It still remains to study the described scheduling problems for a number of other popular interconnection topologies. In the unit execution time model, we have preliminary results for 2- and 3-dimensional arrays with competitive ratio ≤ 10 but in general it appears that the competitive ratio might grow exponentially with the dimension of the array.

References

[AG94] George S. Almasi and Allan Gottlieb. *Highly Parallel Computing.* The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, second revised edition, 1994.

[BDW86] J. Błazewicz, M. Drabowski, and J. Węglarz. Scheduling Multiprocessor Tasks to Minimize Schedule Length. *IEEE Transactions on Computers*, C-35(5):389–393, 1986.

[BEP+96] J. Błażewicz, K.H. Ecker, E. Pesch, G. Schmidt, and J. Węglarz. *Scheduling Computer and Manufacturing Processes.* Springer-Verlag, Berlin, 1996.

[CGJ96] E.G. Coffman, Jr., M.R. Garey, and D.S. Johnson. Approximation Algorithms for Bin Packing: A Survey. In Dorit S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, chapter 2, pages 46–93. PWS Publishing Company, Boston, 1996.

[CM96] Soumen Chakrabarti and S. Muthukrishnan. Resource Scheduling for parallel database and scientific applications. In *Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA '96 (Padua, Italy, June 24–26, 1996)*, pages 329–335, New York, 1996. ACM SIGACT, ACM SIGARCH, ACM Press.

[DL89] Jianzhong Du and Joseph Y.-T. Leung. Complexity of Scheduling Parallel Task Systems. *SIAM J. Disc. Math.*, 2:473–487, 1989.

[FKST93] Anja Feldmann, Ming-Yang Kao, Jiří Sgall, and Shang-Hua Teng. Optimal Online Scheduling of Parallel Jobs with Dependencies. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing (San Diego, California, May 16–18, 1993)*, pages 642–651, New York, 1993. ACM SIGACT, ACM Press.

[FR95] Dror G. Feitelson and Larry Rudolph. Parallel Job Scheduling: Issues and Approaches. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing (IPPS' 95 Workshop, Santa Barbara, CA)*, LNCS 949, pages 1–18, Berlin, 1995. Springer-Verlag.

[FR96] Dror G. Feitelson and Larry Rudolph. Toward Convergence in Job Schedulers for Parallel Supercomputers. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing (IPPS' 96 Workshop, Honolulu, HI)*, LNCS 1162, pages 1–26, Berlin, 1996. Springer.

[FST94] Anja Feldmann, Jiří Sgall, and Shang-Hua Teng. Dynamic scheduling on parallel machines. *Theoretical Computer Science, Special Issue on Dynamic and On-line Algorithms*, 130(1):49–72, 1994.

[GGJY76] M.R. Garey, R.L. Graham, D.S. Johnson, and A.C.-C. Yao. Resource Constrained Scheduling as Generalized Bin Packing. *J. Comb. Theory Series A*, 21:257–298, 1976.

[GI97] Minos N. Garofalakis and Yannis E. Ioannidis. Parallel Query Scheduling and Optimization with Time- and Space-Shared Resources. In Matthias Jarke, Michael J. Carey, Klaus R. Dittrich, Frederick H. Lochovsky, Pericles Loucopoulos, and Manfred A. Jeusfeld, editors, *Proceedings of the 23rd International Conference on Very Large Data Bases VLDB '97 (Athens, Greece, August 25–29)*, pages 296–305, San Francisco, CA, 1997. Morgan Kaufmann Publishers, Inc.

[GJTY83] M.R. Garey, D.S. Johnson, R.E. Tarjan, and M. Yannakakis. Scheduling Opposing Forests. *SIAM J. Algebraic Discrete Methods*, 4(1):72–93, March 1983.

[Gra66] R.L. Graham. Bounds for Certain Multiprocessing Anomalies. *The Bell System Technical Journal*, pages 1563–1581, 1966.

[Gra69] R.L. Graham. Bounds on Multiprocessing Timing Anomalies. *SIAM J. Appl. Math.*, 17(2):416–429, March 1969.

[Joh74] David S. Johnson. Fast Algorithms for Bin Packing. *J. Comput. Syst. Sci.*, 8:272–314, 1974.

[Lei92] F. Thomson Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays • Trees • Hypercubes.* Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1992.

[Rah96] Erhard Rahm. Dynamic Load Balancing in Parallel Database Systems. In Luc Bougé, Pierre Fraigniaud, Anne Mignotte, and Yves Robert, editors, *Proceedings of the Second International EURO-PAR Conference on Parallel Processing, EURO-PAR'96 (Lyon, France, August 26–29),* Volume 1, LNCS 1123, pages 37–52, Berlin, 1996. Springer-Verlag.

[Sal47] H.E. Salzer. The Approximation of Numbers as Sums of Reciprocals. *American Mathematical Monthly*, 54:135–142, 1947.

[Sga94] Jiří Sgall. *On-Line Scheduling on Parallel Machines.* PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, 1994.

[SSA+94] Craig B. Stunkel, Dennis G. Shea, Bülent Abali, Mark Atkins, Carl A. Bender, Don G. Grice, Peter H. Hochschild, Douglas J. Joseph, Ben J. Nathanson, Richard A. Swetz, Robert F. Stucke, Michael Tsao, and Philip R. Varker. The SP2 Communication Subsystem. Research Report RC 19914, IBM Research Division, T.J. Watson Research, 1994.

[ST85] Daniel D. Sleator and Robert E. Tarjan. Amortized Efficiency of List Update and Paging Rules. *Communications of the ACM*, 28(2):202–208, 1985.

[SWW95] David B. Shmoys, Joel Wein, and David P. Williamson. Scheduling Parallel Machines On-Line. *SIAM J. Comput.*, 24(6):1313–1331, 1995.

[WC92] Qingzhou Wang and Kam Hoi Cheng. A Heuristic of Scheduling Parallel Tasks and its Analysis. *SIAM J. Comput.*, 21(2):281–294, April 1992.

SFB 342: Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

bisher erschienen :

Reihe A

Liste aller erschienenen Berichte von 1990-1994
auf besondere Anforderung

342/01/95 A Hans-Joachim Bungartz: Higher Order Finite Elements on Sparse Grids
342/02/95 A Tao Zhang, Seonglim Kang, Lester R. Lipsky: The Performance of Parallel Computers: Order Statistics and Amdahl's Law
342/03/95 A Lester R. Lipsky, Appie van de Liefvoort: Transformation of the Kronecker Product of Identical Servers to a Reduced Product Space
342/04/95 A Pierre Fiorini, Lester R. Lipsky, Wen-Jung Hsin, Appie van de Liefvoort: Auto-Correlation of Lag-k For Customers Departing From Semi-Markov Processes
342/05/95 A Sascha Hilgenfeldt, Robert Balder, Christoph Zenger: Sparse Grids: Applications to Multi-dimensional Schrödinger Problems
342/06/95 A Maximilian Fuchs: Formal Design of a Model-N Counter
342/07/95 A Hans-Joachim Bungartz, Stefan Schulte: Coupled Problems in Microsystem Technology
342/08/95 A Alexander Pfaffinger: Parallel Communication on Workstation Networks with Complex Topologies
342/09/95 A Ketil Stølen: Assumption/Commitment Rules for Data-flow Networks - with an Emphasis on Completeness
342/10/95 A Ketil Stølen, Max Fuchs: A Formal Method for Hardware/Software Co-Design
342/11/95 A Thomas Schnekenburger: The ALDY Load Distribution System
342/12/95 A Javier Esparza, Stefan Römer, Walter Vogler: An Improvement of McMillan's Unfolding Algorithm
342/13/95 A Stephan Melzer, Javier Esparza: Checking System Properties via Integer Programming
342/14/95 A Radu Grosu, Ketil Stølen: A Denotational Model for Mobile Point-to-Point Dataflow Networks
342/15/95 A Andrei Kovalyov, Javier Esparza: A Polynomial Algorithm to Compute the Concurrency Relation of Free-Choice Signal Transition Graphs
342/16/95 A Bernhard Schätz, Katharina Spies: Formale Syntax zur logischen Kernsprache der Focus-Entwicklungsmethodik

Reihe A

342/17/95 A	Georg Stellner: Using CoCheck on a Network of Workstations
342/18/95 A	Arndt Bode, Thomas Ludwig, Vaidy Sunderam, Roland Wismüller: Workshop on PVM, MPI, Tools and Applications
342/19/95 A	Thomas Schnekenburger: Integration of Load Distribution into ParMod-C
342/20/95 A	Ketil Stølen: Refinement Principles Supporting the Transition from Asynchronous to Synchronous Communication
342/21/95 A	Andreas Listl, Giannis Bozas: Performance Gains Using Subpages for Cache Coherency Control
342/22/95 A	Volker Heun, Ernst W. Mayr: Embedding Graphs with Bounded Treewidth into Optimal Hypercubes
342/23/95 A	Petr Jančar, Javier Esparza: Deciding Finiteness of Petri Nets up to Bisimulation
342/24/95 A	M. Jung, U. Rüde: Implicit Extrapolation Methods for Variable Coefficient Problems
342/01/96 A	Michael Griebel, Tilman Neunhoeffer, Hans Regler: Algebraic Multigrid Methods for the Solution of the Navier-Stokes Equations in Complicated Geometries
342/02/96 A	Thomas Grauschopf, Michael Griebel, Hans Regler: Additive Multilevel-Preconditioners based on Bilinear Interpolation, Matrix Dependent Geometric Coarsening and Algebraic-Multigrid Coarsening for Second Order Elliptic PDEs
342/03/96 A	Volker Heun, Ernst W. Mayr: Optimal Dynamic Edge-Disjoint Embeddings of Complete Binary Trees into Hypercubes
342/04/96 A	Thomas Huckle: Efficient Computation of Sparse Approximate Inverses
342/05/96 A	Thomas Ludwig, Roland Wismüller, Vaidy Sunderam, Arndt Bode: OMIS — On-line Monitoring Interface Specification
342/06/96 A	Ekkart Kindler: A Compositional Partial Order Semantics for Petri Net Components
342/07/96 A	Richard Mayr: Some Results on Basic Parallel Processes
342/08/96 A	Ralph Radermacher, Frank Weimer: INSEL Syntax-Bericht
342/09/96 A	P.P. Spies, C. Eckert, M. Lange, D. Marek, R. Radermacher, F. Weimer, H.-M. Windisch: Sprachkonzepte zur Konstruktion verteilter Systeme
342/10/96 A	Stefan Lamberts, Thomas Ludwig, Christian Röder, Arndt Bode: PFSLib – A File System for Parallel Programming Environments
342/11/96 A	Manfred Broy, Gheorghe Ştefănescu: The Algebra of Stream Processing Functions
342/12/96 A	Javier Esparza: Reachability in Live and Safe Free-Choice Petri Nets is NP-complete
342/13/96 A	Radu Grosu, Ketil Stølen: A Denotational Model for Mobile Many-to-Many Data-flow Networks

Reihe A

342/14/96 A	Giannis Bozas, Michael Jaedicke, Andreas Listl, Bernhard Mitschang, Angelika Reiser, Stephan Zimmermann: On Transforming a Sequential SQL-DBMS into a Parallel One: First Results and Experiences of the MIDAS Project
342/15/96 A	Richard Mayr: A Tableau System for Model Checking Petri Nets with a Fragment of the Linear Time μ -Calculus
342/16/96 A	Ursula Hinkel, Katharina Spies: Anleitung zur Spezifikation von mobilen, dynamischen Focus-Netzen
342/17/96 A	Richard Mayr: Model Checking PA-Processes
342/18/96 A	Michaela Huhn, Peter Niebert, Frank Wallner: Put your Model Checker on Diet: Verification on Local States
342/01/97 A	Tobias Müller, Stefan Lamberts, Ursula Maier, Georg Stellner: Evaluierung der Leistungsf"ahigkeit eines ATM-Netzes mit parallelen Programmierbibliotheken
342/02/97 A	Hans-Joachim Bungartz and Thomas Dornseifer: Sparse Grids: Recent Developments for Elliptic Partial Differential Equations
342/03/97 A	Bernhard Mitschang: Technologie f"ur Parallele Datenbanken - Bericht zum Workshop
342/04/97 A	nicht erschienen
342/05/97 A	Hans-Joachim Bungartz, Ralf Ebner, Stefan Schulte: Hierarchische Basen zur effizienten Kopplung substrukturierter Probleme der Strukturmechanik
342/06/97 A	Hans-Joachim Bungartz, Anton Frank, Florian Meier, Tilman Neunhoeffer, Stefan Schulte: Fluid Structure Interaction: 3D Numerical Simulation and Visualization of a Micropump
342/07/97 A	Javier Esparza, Stephan Melzer: Model Checking LTL using Constraint Programming
342/08/97 A	Niels Reimer: Untersuchung von Strategien für verteiltes Last- und Ressourcenmanagement
342/09/97 A	Markus Pizka: Design and Implementation of the GNU INSEL-Compiler gic
342/10/97 A	Manfred Broy, Franz Regensburger, Bernhard Schätz, Katharina Spies: The Steamboiler Specification - A Case Study in Focus
342/11/97 A	Christine Röckl: How to Make Substitution Preserve Strong Bisimilarity
342/12/97 A	Christian B. Czech: Architektur und Konzept des Dycos-Kerns
342/13/97 A	Jan Philipps, Alexander Schmidt: Traffic Flow by Data Flow
342/14/97 A	Norbert Fröhlich, Rolf Schlagenhaft, Josef Fleischmann: Partitioning VLSI-Circuits for Parallel Simulation on Transistor Level
342/15/97 A	Frank Weimer: DaViT: Ein System zur interaktiven Ausführung und zur Visualisierung von INSEL-Programmen
342/16/97 A	Niels Reimer, Jürgen Rudolph, Katharina Spies: Von FOCUS nach INSEL - Eine Aufzugssteuerung

Reihe A

342/17/97 A Radu Grosu, Ketil Stølen, Manfred Broy: A Denotational Model for Mobile Point-to-Point Data-flow Networks with Channel Sharing

342/18/97 A Christian Röder, Georg Stellner: Design of Load Management for Parallel Applications in Networks of Heterogenous Workstations

342/19/97 A Frank Wallner: Model Checking LTL Using Net Unfoldings

342/20/97 A Andreas Wolf, Andreas Kmoch: Einsatz eines automatischen Theorembeweisers in einer taktikgesteuerten Beweisumgebung zur Lösung eines Beispiels aus der Hardware-Verifikation – Fallstudie –

342/21/97 A Andreas Wolf, Marc Fuchs: Cooperative Parallel Automated Theorem Proving

342/22/97 A T. Ludwig, R. Wismüller, V. Sunderam, A. Bode: OMIS - On-line Monitoring Interface Specification (Version 2.0)

342/23/97 A Stephan Merkel: Verification of Fault Tolerant Algorithms Using PEP

342/24/97 A Manfred Broy, Max Breitling, Bernhard Schätz, Katharina Spies: Summary of Case Studies in Focus - Part II

342/25/97 A Michael Jaedicke, Bernhard Mitschang: A Framework for Parallel Processing of Aggregat and Scalar Functions in Object-Relational DBMS

342/26/97 A Marc Fuchs: Similarity-Based Lemma Generation with Lemma-Delaying Tableau Enumeration

342/27/97 A Max Breitling: Formalizing and Verifying TimeWarp with FOCUS

342/28/97 A Peter Jakobi, Andreas Wolf: DBFW: A Simple DataBase FrameWork for the Evaluation and Maintenance of Automated Theorem Prover Data (incl. Documentation)

342/29/97 A Radu Grosu, Ketil Stølen: Compositional Specification of Mobile Systems

342/01/98 A A. Bode, A. Ganz, C. Gold, S. Petri, N. Reimer, B. Schiemann, T. Schnekenburger (Herausgeber): "'Anwendungsbezogene Lastverteilung"', ALV'98

342/02/98 A Ursula Hinkel: Home Shopping - Die Spezifikation einer Kommunikationsanwendung in FOCUS

342/03/98 A Katharina Spies: Eine Methode zur formalen Modellierung von Betriebssystemkonzepten

342/04/98 A Stefan Bischof, Ernst W. Mayr: On-Line Scheduling of Parallel Jobs with Runtime Restrictions

SFB 342 : Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

Reihe B

342/1/90 B	Wolfgang Reisig: Petri Nets and Algebraic Specifications
342/2/90 B	Jörg Desel: On Abstraction of Nets
342/3/90 B	Jörg Desel: Reduction and Design of Well-behaved Free-choice Systems
342/4/90 B	Franz Abstreiter, Michael Friedrich, Hans-Jürgen Plewan: Das Werkzeug runtime zur Beobachtung verteilter und paralleler Programme
342/1/91 B	Barbara Paech1: Concurrency as a Modality
342/2/91 B	Birgit Kandler, Markus Pawlowski: SAM: Eine Sortier- Toolbox -Anwenderbeschreibung
342/3/91 B	Erwin Loibl, Hans Obermaier, Markus Pawlowski: 2. Workshop über Parallelisierung von Datenbanksystemen
342/4/91 B	Werner Pohlmann: A Limitation of Distributed Simulation Methods
342/5/91 B	Dominik Gomm, Ekkart Kindler: A Weakly Coherent Virtually Shared Memory Scheme: Formal Specification and Analysis
342/6/91 B	Dominik Gomm, Ekkart Kindler: Causality Based Specification and Correctness Proof of a Virtually Shared Memory Scheme
342/7/91 B	W. Reisig: Concurrent Temporal Logic
342/1/92 B	Malte Grosse, Christian B. Suttner: A Parallel Algorithm for Set-of-Support Christian B. Suttner: Parallel Computation of Multiple Sets-of-Support
342/2/92 B	Arndt Bode, Hartmut Wedekind: Parallelrechner: Theorie, Hardware, Software, Anwendungen
342/1/93 B	Max Fuchs: Funktionale Spezifikation einer Geschwindigkeitsregelung
342/2/93 B	Ekkart Kindler: Sicherheits- und Lebendigkeitseigenschaften: Ein Literaturüberblick
342/1/94 B	Andreas Listl; Thomas Schnekenburger; Michael Friedrich: Zum Entwurf eines Prototypen für MIDAS