Dynamic Scheduling on Parallel Machines

Anja Feldmann* Jiří Sgall Shang-Hua Teng†

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

In this paper, we study the problem of on-line job-scheduling on various parallel architectures. In particular, we give an $O(\sqrt{\log \log n})$-competitive algorithm for on-line dynamic scheduling on an $n \times n$ mesh. We prove that this algorithm is optimal up to a constant factor – no on-line algorithm can achieve a competitive ratio better than $\frac{1}{42}\sqrt{\log \log n}$. Our algorithm is not greedy and the lower bound proof shows that no greedy-like algorithm can be very good. Our upper bound result can be generalized to any fixed dimensional meshes. We also present competitive scheduling algorithms for other architectures including hypercubes, trees, meshes of trees, and PRAMs.

1 Introduction

One function of a time-sharing operating system is to present each job with a virtual machine on which to run [11]. A particular job is oblivious to the fact that there are many other jobs running simultaneously on the same hardware. The problem we consider is how to do these kinds of simulations on parallel machines. In this case a given job has a degree of parallelism (assumed to be known at run-time) on the parallel architecture in question that it can make efficient use of. (In the case of a mesh, for example, a job might know that it will run efficiently on a 100 by 100 mesh, but no larger. A 100 by 100 mesh can be simulated on a 50 by 50 mesh with no loss in efficiency.)

In this paper we study the problem of on-line dynamic scheduling on various parallel architectures. In contrast to previous work [2, 4, 5, 10, 14], we study scheduling on various parallel architectures. The introduction of underlying architectures realistically captures the scheduling problem on practical parallel systems (such as iWarp and Connection machines).

The problem we consider is the following – given a set of jobs (each with unknown run time, but known resource requirements, i.e., the maximum number of processors on which it can efficiently run), find a way to schedule these jobs to run on a given parallel machine so that they all finish in the minimum amount of time. In our model, the scheduling is non-preemptive without restarts, i.e., once the job is started, it cannot be stopped and resumed or restarted later. Our measure of performance of a scheduling algorithm is the competitive ratio: the worst-case ratio of the algorithm's performance to that of an off-line algorithm that knows all the run times in advance [13].

We present algorithms with optimal competitive ratios (to within a constant factor) for this problem on a variety of different parallel architectures: meshes, hypercubes, trees, meshes of trees, and the PRAM model. All algoriothms use specific of the the topology of the parallel machine in question.

As far as we know, this paper is the first study of on-line dynamic scheduling on parallel machines with specific architectures. It is a first step in a new area of research, and opens up many interesting new problems.

The scheduling problem is made difficult by the underlying geometrical structure. This applies especially to the case of a mesh architecture. In the proof of the lower bound it is necessary to prove that the adversary can always insure that a small set of running jobs will prohibit any efficient scheduling of other jobs. This is based on a geometrical lemma 4.1 which is interesting on its own right.

We present an $O(\sqrt{\log \log n})$-competitive on-line

*This work was supported in part by fellowship of the Studienstiftung des deutschen Volkes.

†This work was supported in part by National Science Foundation grant DCR-8713489. Current Address: Xerox PARC, 3333 Coyote Hill Road, Palo Alto, CA 94304.

algorithm for scheduling on a $n \times n$ mesh. Moreover we also prove a matching lower bound on the competitive ratio of $\Omega(\sqrt{\log \log n})$ for any on-line scheduling algorithm for the $n \times n$ mesh.

It is worth noting that our optimal algorithm is not greedy; at each time it uses only a small portion of the mesh efficiently. The proof of the lower bound shows that this is not an arbitrary choice – no greedy-like algorithm can achieve a substantially better competitive ratio than $\Omega(\log \log n)$. This shows that the general heuristic of using greedy algorithms for scheduling can be misguided despite the fact that it works well in all the previous scheduling algorithms known to the authors [5, 4, 14].

Our algorithm and results for the $n \times n$ mesh can be generalized to meshes of any fixed dimension.

Efficient parallel algorithms have been developed for a wide variety of problems and for various parallel architectures [6]. Mesh are one of the most widely used architectures in numerical analysis, solving PDE's, image analysis, etc [8]. Until now no efficient scheduling algorithm for meshes was known. The result has been that meshes are generally used to schedule one process at a time. Our results demonstrate that mesh-based parallel machines can also be used to efficiently support multi-task operating systems.

In Section 2 we give the definitions and state the main results. In Section 3 we present the optimal scheduling algorithm for the 2-dimensional mesh. The lower bound is proved in Section 4. Section 5 is a survey of the results for other architectures.

2 Definitions and Results

When scheduling multiple jobs on a parallel architecture, a sequence of jobs, each requiring a specific amount of resources, arrive in a dynamic fashion and with dynamically determined running time. The problem is to design an on-line scheduling algorithm that minimizes the total time required to process all jobs. [1]

However, this general problem is so complex that we do not know any good solution. As a first approximation of the general problem, we assume that all jobs are given at the beginning of the scheduling as well as the amount of resources they need, but the running time of each job is not given, it can only be determined by actually running the job. By the technique of [14] we can weaken the requirement that all jobs are given at the beginning. It is sufficient if each job is released at some fixed time (independent of the schedule), and at that time also the needed resources are known. However, this still does not capture the most interesting case with dependencies between jobs.

We feel that the model of non-preemptive scheduling without restarts is more close to its practical counterpart. The model with restart, although theoretically has better scheduling algorithms, is an oversimplification of the practical scheduling problem. This is because in practice there are other non neglectable costs and software complications associated with restart, such as reloading, buffer and communication reallocation.

In the following, we define our notion of parallel architectures, jobs, and schedules.

A *parallel architecture* of n processors is defined as a graph with n nodes.

Let $\mathcal{G}$ be a set of graphs (e.g., a set of trees or 2-dimensional meshes), called a *job-type*. A $\mathcal{G}$*-job* J is characterized as a pair $J = (G,t)$ where $G \in \mathcal{G}$ and t is the running time of J on a parallel architecture with graph G.

In general, we assume that a computation on one parallel architecture can be simulated on another architecture within a constant time called *the simulation factor* [1, 7]. For example, the computation of an r-dimensional hypercube can be simulated on an $r' < r$ dimensional hypercube by increasing the running time by a factor of $2^{r-r'}$.

Each instance of the scheduling problem has three components $(\mathcal{A}, \mathcal{G}, \mathcal{R})$ where $\mathcal{A}$ is the architecture of the given parallel machine, $\mathcal{G}$ the type of jobs, and $\mathcal{R}$ a set of simulation assumptions stating simulation factors of each graph in $\mathcal{G}$ by each subgraph of $\mathcal{A}$.

Given a set $\mathcal{J}$ of $\mathcal{G}$-jobs, a *schedule* of $\mathcal{J}$ on a parallel architecture $\mathcal{A}$ is an assignment of a subgraph of $\mathcal{A}$ and a time interval to each job in $\mathcal{J}$ such that at any given time no two subgraphs assigned to different running jobs are overlapping. The *length* of a schedule S, denoted by $T_S(\mathcal{J})$, is the time needed for S to process all jobs in $\mathcal{J}$ (using the simulation assumptions $\mathcal{R}$).

A scheduling algorithm is *off-line* if it receives the running time of each job as input. It is *on-line* if it does not depend on knowledge of the running time of jobs.

Definition 2.1 (On-line Scheduling on Parallel Architectures) *Given a parallel architecture $\mathcal{A}$, a job-type $\mathcal{G}$, a set of simulation assumptions $\mathcal{R}$, and a set $\mathcal{J}$ of $\mathcal{G}$-jobs with unknown running time, compute an efficient schedule S for $\mathcal{J}$.*

Let $T_{opt}(\mathcal{J})$ be the length of an *optimal* (off-line) schedule, i.e., a schedule with a minimal possible length. Note that deciding whether a given schedule is optimal is NP-complete.

We say a schedule S is σ*-competitive* for $\mathcal{J}$ if

[1] A closed related problem is *dynamic bin packing* [3].

$T_S(\mathcal{J}) \leq \sigma T_{opt}(\mathcal{J})$. An on-line algorithm for scheduling $\mathcal{G}$-jobs on an architecture $\mathcal{A}$, is σ-competitive if for each set $\mathcal{J}$ of $\mathcal{G}$-jobs, it generates a σ-competitive schedule.

2.1 On-line Scheduling on the $n \times n$ Mesh

The mesh architecture is of particular interest not only because it is one of the most practical and widely used parallel architectures [8], but also the combinatorial problem behind it is very interesting and challenging. We now define the on-line scheduling problem on 2-dimensional meshes.

A 2-dimensional $l_1 \times l_2$ *mesh* is a collection of processors an $l_1 \times l_2$ grid, one processor per grid point. A processor is denoted by its position (x_1, x_2) in a grid, where $0 \leq x_i \leq l_i - 1$. A $u_1 \times u_2$ *submesh* at grid point (x_1, x_2) contains all grid points $\{(y_1, \ldots, y_d) | x_i \leq y_i < x_i + u_i\}$, where $u_i \leq l_i - x_i$.

The architecture $\mathcal{A}$ for the on-line 2-dimensional mesh scheduling problem is the $n \times n$ mesh. Each 2-dimensional job is characterized by a 3-tuple (a, b, t), where (a, b) means that J needs an $a \times b$-submesh and t is the running time.

Since the mesh $a \times b$ is entirely equivalent to mesh $b \times a$, we assume without loss of generality that $a \geq b$ for each job $J = (a, b, t)$. We use the following *mesh simulation assumption*: each job (a, b, t) can be processed on a $u \times v$ mesh in time $t \max(1, a/u) \max(1, b/v)$. In fact, our upper bound result does not use the mesh simulation assumption, but all our lower bound results hold even when the above assumption is applied.

Definition 2.2 (On-line Scheduling on the $n \times n$ Mesh) *Given a set $\mathcal{J}$ of 2-dimensional jobs with unknown running time, compute an efficient schedule S for $\mathcal{J}$.*

Let $\mathcal{J} = \{J_1, \ldots, J_m\}$ be a set of 2-dimensional jobs, where $J_i = (a_i, b_i, t_i)$. We define $T_{eff}(\mathcal{J}) = (\sum_i a_i b_i t_i)/n^2$, $t_{max}(\mathcal{J}) = \max_i t_i$ and $T_{min}(\mathcal{J}) = \max(T_{eff}(\mathcal{J}), t_{\max}(\mathcal{J}))$. It is easy to see that $T_{min}(\mathcal{J}) \leq T_{opt}(\mathcal{J})$ for all $\mathcal{J}$.

The problem on d-dimensional meshes can be defined in the same way.

An on-line dynamic scheduling problem can be viewed as a game between the scheduling algorithm and the adversary who dynamically determines the run-time of each jobs. The scheduling algorithm tries to map jobs to submeshes of the $n \times n$ mesh and the adversary decides when to remove the jobs from the mesh, namely, it decides when a job finishes. For technical simplicity, we assume that both the removal from and the mapping onto a submesh takes no time. The goal of the adversary is to maximize the competitive ratio of the on-line algorithm while the goal of the scheduling algorithm is to minimize it.

A processor in the $n \times n$ mesh is *busy* at time t if it belongs to a submesh on which a job is mapped. The *efficiency* of a schedule at time t is the number of busy processors divided by n^2. Similarly, we can define the efficiency with respect to a submesh and the efficiency of a currently running job J and of a set $\mathcal{C}$ of currently running jobs, denoted by eff(J) and eff($\mathcal{C}$), respectively. Finally, for each $\alpha \leq 1$, let $T_{<\alpha}(S, J)$ be the total time during which the efficiency of S is less than α. The following is a basic property of scheduling.

Lemma 2.3 *Let $\alpha \leq 1$, $\beta \geq 0$ and let S be a schedule for a set $\mathcal{J}$ of d-dimensional jobs such that $T_{<\alpha}(S, \mathcal{J}) \leq \beta T_{opt}(\mathcal{J})$. Then $T_S(\mathcal{J}) \leq (1/\alpha + \beta) T_{opt}(\mathcal{J})$.* □

2.2 Main Results

Our main results are upper and lower bounds for scheduling on a two dimensional mesh.

Theorem 2.4 (Two Dimensional Mesh - Upper Bound) *There is an on-line scheduling algorithm that is $O(\sqrt{\log \log n})$-competitive for the $n \times n$ mesh.*

Theorem 2.5 (Two Dimensional Mesh - Lower Bound) *There is a lower bound on the competitive ratio of $\Omega(\sqrt{\log \log n})$ on any on-line scheduling algorithm for the $n \times n$ mesh.*

We also prove upper bounds for a number of other architectures (see Section 5); the most interesting one is:

Theorem 2.6 (The d-Dimensional Mesh) *If d is a constant, then there is an on-line scheduling algorithm that is $\Theta(\sqrt{\log \log n})$-competitive for the d-dimensional $n \times \cdots \times n$ mesh.*

3 Two Dimensional Meshes

The basic idea for 2-dimensional mesh scheduling is to reduce the problem to a one-dimensional mesh scheduling problem. We start by giving some efficient algorithms for the 1-dimensional case. Then we go on and formulate some general methods for building 2-dimensional algorithms. We construct first an $O(\log \log n)$ algorithm and then an optimal one using these tools.

3.1 One Dimensional Problem

A natural method for scheduling is the *greedy method.* In terms of scheduling on meshes, it schedules a job on a submesh whenever the submesh is sufficiently large. In this subsection, we present two greedy methods and show they are 3-competitive and $2\sqrt{2}$-competitive. Interestingly, we observe that greedy methods do not work well for 2-dimensional on-line scheduling.

In both algorithms we suppose that the jobs are sorted by size, i.e. $a_1 \geq a_2 \geq \ldots \geq a_m$, m is the number of jobs. During the schedule, let $\mathcal{C}$ denote the set of currently running jobs and $F(\mathcal{C})$ denote the set of remaining intervals induced by $\mathcal{C}$. Let $size(I)$ denote the size of an interval I.

In the first greedy algorithm, jobs are scheduled in the order of decreasing size of the submesh they require. We call this algorithm *largest-job-first-greedy algorithm.*

ALGORITHM: *largest-job-first-greedy*

1. **let** $i = 1$;
2. **while** $i \leq m$ **do**
   if there is an interval I in $F(\mathcal{C})$, $I = [u..v]$, such that $size(I) \geq a_i$,
   then map job J_i to $[u..u + a_i - 1]$.

Lemma 3.1 *In the algorithm above, as long as $i \leq m$, the efficiency of S is at least $1/2$. Therefore $T_{<1/2}(S, \mathcal{J}) \leq t_{max}(\mathcal{J})$.*

Proof: Let J_i be the largest unscheduled job. Then for all intervals $I \in \mathcal{C}$, $size(I) \geq a_i$, while for all intervals $I' \in F(\mathcal{C})$, $size(I') \leq a_i$. Moreover, the number of intervals in $\mathcal{C}$ is equal to the number of intervals in $F(\mathcal{C})$. □

Consequently by Lemma 2.3,

Theorem 3.2 *The* largest-job-first-greedy-algorithm *is 3-competitive.* □

The following *balanced-greedy-algorithm* uses the mesh simulation assumption.

ALGORITHM: *balanced-greedy-algorithm*

1. **let** $i = 1$;
2. **while** $i \leq m$ **do**
   if there is an interval in $F(\mathcal{C})$, $I = [u..v]$, such that $size(I) \geq a_i$ **then**
   (a) **if** $a_i \geq \frac{1}{\sqrt{2}} size(I)$, **then map** job J_i to $[u..u + a_i - 1]$, and **let** $i = i + 1$;
   (b) **otherwise** find the smallest l such that $a_i + a_{i+1} + \cdots + a_{i+l} \geq size(I)$, **map** job J_{i+s} to $[\beta_{s-1} + 1..\beta_s]$, where $\beta_0 = u - 1$ and $\beta_s = \lfloor \frac{\sum_{j=1}^{s} a_j}{\sum_{i=1}^{l} a_j} size(I) \rfloor$; and **let** $i = i + l$.

Lemma 3.3 *In the algorithm above, as long as $i \leq m$, the efficiency of S is at least $1/\sqrt{2}$. Therefore $T_{<1/\sqrt{2}}(S, \mathcal{J}) \leq \sqrt{2} t_{\max}(\mathcal{J})$.* □

Consequently by Lemma 2.3,

Theorem 3.4 *The* balanced-greedy-algorithm *is $2\sqrt{2}$ -competitive.* □

In the next subsection, we will work with the following generalization of the 1-dimensional mesh scheduling problem, called *multi-mesh scheduling problem*, namely the problem of scheduling a set of 1-dimensional jobs $\mathcal{J} = \{J_1, \ldots, J_m\}$ on a set of p 1-dimensional meshes of size $n_1, \ldots, n_p$, respectively (we assume $a_i \leq n_h$, for all $1 \leq i \leq m$ and $1 \leq h \leq p$).

Corollary 3.5 *The* largest-job-first-greedy-algorithm *and the* balanced-greedy-algorithm *are 3-competitive and $2\sqrt{2}$-competitive, respectively, for the multi-mesh scheduling problem.* □

3.2 An $O(\log \log n)$-competitive Algorithm

In this section we construct an $O(\log \log n)$-competitive on-line scheduling algorithm for the $n \times n$ mesh. Later we will prove a better bound, but this algorithm will be used to schedule the larger jobs.

Because of the more complex geometrical structure of the 2-dimensional mesh, the simple greedy approach does not work too well. But by partitioning the jobs according to their size we can do reasonably well by scheduling jobs in a number of consecutive phases that use the greedy algorithms of the last section. We start with some useful definitions.

Definition 3.6

1. *Let $\mathcal{D}_1, \ldots, \mathcal{D}_h$ be a partition of the set of d-dimensional jobs $\mathcal{J}$ and let $S_1, \ldots, S_h$ be schedules for $\mathcal{D}_1, \ldots, \mathcal{D}_h$, respectively. $S = S_1 \odot S_2 \odot \cdots \odot S_h$ (*a serial composition*) denotes a schedule for $\mathcal{J}$ that first uses S_1 for $\mathcal{J}_1$, when it finishes, it uses S_2, and so on until S_h finishes.*
2. *Let $\mathcal{J}$ be a set of two-dimensional jobs with $a_i \leq n/h$. Let $\mathcal{D}_1, \ldots, \mathcal{D}_h$ be a partition of jobs in $\mathcal{J}$ and $U_1, \ldots, U_h$ be a partition of the $n \times n$ mesh into h of $\lfloor n/h \rfloor \times n$ submeshes (see Figure 1). Let S_j be a schedule that maps jobs $\mathcal{D}_j$ only on U_j. $S = S_1 \oplus S_2 \oplus \cdots \oplus S_h$ (*a parallel composition*) denotes a schedule that simultaneously applies $S_1, \ldots, S_h$.*
3. *Let $\mathcal{J}$ be a set of two-dimensional jobs. We define a partition $\mathcal{J} = \mathcal{J}^{(0)} \cup \ldots \cup \mathcal{J}^{(\lfloor \log n \rfloor)}$ by $\mathcal{J}^{(l)} = \{J_i | \frac{n}{2^{l+1}} < a_i \leq \frac{n}{2^l}\}$. Define the* order *of $\mathcal{J}$ to be $order(\mathcal{J}) = |\{\mathcal{J}^{(l)} | \mathcal{J}^{(l)} \neq \emptyset\}|$.*

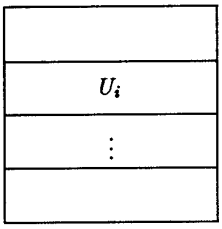

Figure 1: A Partition of the $n \times n$ Mesh

First we will look at the serial composition. We use it to schedule sets of jobs of small order.

Lemma 3.7 (Serial Composition) *For all* $\alpha \leq 1$, $T_{<\alpha}(S_1 \odot S_2 \odot \cdots \odot S_h, \mathcal{J}) = \sum_{i=1}^{h} T_{<\alpha}(S_i, \mathcal{J}_i)$. □

From the above Serial Composition Lemma an $O(\log n)$-competitive on-line scheduling algorithm follows immediately.

For each $0 \leq l \leq \lfloor \log n \rfloor$, define the *single-set-greedy algorithm on* $\mathcal{J}^{(l)}$ to be the on-line algorithm that partitions the $n \times n$ mesh into 2^l submeshes of size $\frac{n}{2^l} \times n$ and applies the one dimensional largest-job-first-greedy-scheduling algorithm to $\mathcal{J}^{(l)}$ by viewing each $\frac{n}{2^l} \times n$ submesh as a 1-dimensional mesh of size n, each processor as an 1-dimensional mesh of size $\frac{n}{2^l}$, and each job $J_i \in \mathcal{J}^{(l)}$ as a 1-dimensional job of size b_i.

Let S_l be the schedule generated by the single-set-greedy algorithm on $\mathcal{J}^{(l)}$. Now let $S = S_0 \odot S_1 \odot \cdots \odot S_{\lfloor \log n \rfloor}$. We call the algorithm that generates S the *one-set-a-time algorithm.* It follows from Lemma 3.1 and Corollary 3.5 that $T_{<1/4}(S_l, \mathcal{J}^{(l)}) \leq t_{max}(\mathcal{J}^{(l)})$. By Lemma 3.7, we have $T_{<1/4}(S, \mathcal{J}) = \sum_{l=0}^{\lfloor \log n \rfloor} T_{<1/4}(S_l, \mathcal{J}^{(l)})$, and thus $T_{<1/4}(S, \mathcal{J}) \leq order(\mathcal{J}) t_{\max}(\mathcal{J})$. By Lemma 2.3, we have

Lemma 3.8 *The* one-set-a-time algorithm *is* $(4 + order(\mathcal{J}))$*-competitive for each set* $\mathcal{J}$ *of two dimensional jobs.* □

Notice that the one-set-a-time algorithm uses a static partition of $\mathcal{J}$. In the following, we shall show how to *dynamically* partition $\mathcal{J}$ into $h = O(\log \log n)$ subsets $\mathcal{D}_1, \ldots, \mathcal{D}_h$ and generate a schedule S of form $S = S_1 \odot \cdots \odot S_h$, where S_l is a schedule on $\mathcal{D}_l$ such that $T_{<1/4}(S_l, \mathcal{D}_l) \leq t_{max}(\mathcal{J})$. We use the following lemma to perform the dynamic partition.

Lemma 3.9 (Parallel Composition) *Let* $S = S_1 \oplus S_2 \oplus \cdots \oplus S_h$, *let* $0 \leq \alpha, \beta \leq 1$. *If at time* t *the efficiency of at least* βh *of* S_j*'s (with respect to the submesh* U_j *of size* $\lfloor n/h \rfloor \times n$*) is at least* α, *then the efficiency of* S *at time* t *is at least* $\alpha\beta$.

Proof: Let E_j be the efficiency of S_j at time t (with respect to submesh U_j) and E be the efficiency of S. Clearly, $E = (\sum_j (E_j n^2/h))/n^2 \geq \alpha\beta$. □

Let $\mathcal{J}' = \bigcup_{l=0}^{\lceil \log \log n \rceil + 2} \mathcal{J}^{(l)}$. Note that if $J_i \in \mathcal{J} - \mathcal{J}'$, then $b_i \leq a_i \leq \lfloor n/\log n \rfloor$. We now show that the following on-line scheduling algorithm is $O(\log(order(\mathcal{J} - \mathcal{J}')))$-competitive for $\mathcal{J} - \mathcal{J}'$.

ALGORITHM *greedy-parallel-serial-partition*
Let $\mathcal{J}_1 = \mathcal{J} - \mathcal{J}'$. Then proceed in phases. In phase i schedule jobs according to the following rules.

1. let $h_i = order(\mathcal{J}_i)$; assign to each nonempty set $\mathcal{J}_i^{(l)}$ a $\lfloor n/h_i \rfloor \times n$ submesh denoted by $U_{i,l}$;
2. apply the single-set-greedy algorithm to all nonempty set $\mathcal{J}_i^{(l)}$ on $U_{i,l}$ until the first time when the efficiency of the current running jobs (in whole $n \times n$ mesh) is less than $1/k$ for a predefined $k > 1$ (we call this time a *checkpoint* in phase i); then stop the single-set-greedy algorithm but allow all current jobs to be processed.
3. let $\mathcal{J}_{i+1}$ be the jobs in $\mathcal{J}_i$ that have not been scheduled yet;
4. when all currently running jobs have been processed, start phase $i + 1$;

Claim 3.10 *In the algorithm above, for all* i, $h_{i+1} \leq \frac{4}{k} h_i$.

Proof: Following Lemma 3.1, if $\mathcal{J}_i^{(l)}$ is not empty after the phase i, then the efficiency of the schedule for the jobs in $\mathcal{J}_i^{(l)}$ (with respect $U_{i,l}$) is at least $1/4$ before the checkpoint in the phase i. Therefore, by Lemma 3.9, the efficiency at the checkpoint in phase i is at least $h_{i+1}/4h_i$. On the other hand, by the definition of checkpoint, this efficiency is less than $1/k$. Hence $h_{i+1} \leq \frac{4}{k} h_i$. □

Lemma 3.11 *The* greedy-parallel-serial-partition-algorithm *is* $O(\log(order(\mathcal{J} - \mathcal{J}')))$*-competitive for* $\mathcal{J} - \mathcal{J}'$.

Proof: Let $\mathcal{D}_i$ be the set of jobs scheduled in phase i, S_i be the schedule generated by the algorithm for $\mathcal{D}_i$ and p be the number of the phases of the algorithm. The schedule generated by the algorithm is of the form $S = S_1 \odot \cdots \odot S_p$. Each S_i is of the form $S_i = \bigoplus_{\mathcal{J}_i^{(l)} \neq \emptyset} S_{i,l}$, where $S_{i,l}$ is a schedule generated by applying the single-set-scheduling-algorithm to $\mathcal{J}_i^{(l)} - \mathcal{J}_{i+1}^{(l)}$ on submesh $U_{i,l}$. The time between the checkpoint of phase i and the beginning of phase $i + 1$ is bounded by t_{max}. By

Claim 3.10, the number of phases is bounded by $\log_{k/4}(order(\mathcal{J}-\mathcal{J}'))$. Thus, by Lemmas 3.7, 3.9, and 2.3, the greedy-parallel-serial-partition-algorithm is $(k+\log_{k/4}(order(\mathcal{J}-\mathcal{J}')))$-competitive. By choosing $k=8$, we show that the above algorithm is $O(\log(order(\mathcal{J}-\mathcal{J}')))$-competitive. In fact, by choosing $k=\frac{\log\log n}{(\log\log\log n)^2}$ and more careful analysis, we can show that the above algorithm is $O(\frac{\log\log n}{(\log\log\log n)^2})$-competitive. □

The next lemma is stronger than we need now, but it will be useful later.

Lemma 3.12 *Let $\mathcal{J}'=\cup_{l=0}^{\lceil\log\log n\rceil+2}\mathcal{J}^{(l)}$. There is an $O(\log\log\log n)$-competitive on-line scheduling algorithm for $\mathcal{J}'$.*

Proof: Let $\mathcal{J}''=\cup_{l=0}^{\log\log\log n}\mathcal{J}^{(l)}$. Applying the greedy-parallel-serial-partition-algorithm to $\mathcal{J}'-\mathcal{J}''$ generates a schedule S_1' which is $O(\log\log\log n)$-competitive for $\mathcal{J}'-\mathcal{J}''$. Applying the one-set-a-time-algorithm to $\mathcal{J}''$ constructs a $O(\log\log\log n)$-competitive schedule S_2' for $\mathcal{J}''$. Using Lemma 3.7, we see that $S_1'\odot S_2'$ is a $O(\log\log\log n)$-competitive schedule for $\mathcal{J}'$. □

Theorem 3.13 *There is an $O(\log\log n)$-competitive on-line scheduling algorithm for the $n\times n$ mesh.*

Proof: By Lemmas 3.11 and 3.12. □

3.3 An Optimal Algorithm

In this section we construct an $O(\sqrt{\log\log n})$-competitive on-line scheduling algorithm for the $n\times n$ mesh. This scheduling algorithm is *prudent* rather than greedy. The main ingredient of the algorithm is to make an optimal trade-off between the average efficiency of the schedule and the amount of time that the efficiency of the schedule is below the average. In the next section, we shall prove a matching lower bound.

Following Lemma 3.12, we assume for all $J_i\in\mathcal{J}$, $a_i\le\lfloor n/\log n\rfloor$. For a given integer k, let U_j, $1\le j\le k$, be a $n\times\lfloor n/k\rfloor$ submesh (See Figure 3.3). The algorithm is performed in *phases* and each phase consists of k *steps*.

ALGORITHM
prudent-parallel-serial-partition
Let $\mathcal{J}_{1,1}=\mathcal{J}$. In step j of phase i, the algorithm schedules jobs according to the following rules.

1. let $h_{i,j}=order(\mathcal{J}_{i,j})$; assign to each nonempty set $\mathcal{J}_{i,j}^{(l)}$ an $h_{i,j}\times\lfloor n/k\rfloor$ submesh of U_j, denoted by $U_{i,j,l}$ (see Figure 3.3);
2. apply the single-set-greedy algorithm to all nonempty sets $\mathcal{J}_{i,j}^{(l)}$ on $U_{i,j,l}$ until the first time when the efficiency of all current running jobs (with respect to U_j) is less than 1/8, (we call this time a *checkpoint* of step j in phase i and if $j=k$, we call it the *checkpoint* of phase i), then stop all the single-set-algorithms but allow the current jobs be processed;
3. if $j<k$, then let $\mathcal{J}_{i,j+1}$ be the jobs in $\mathcal{J}_{i,j}$ that have not yet been scheduled and start step $j+1$ in phase i immediately; otherwise let $\mathcal{J}_{i+1,1}$ be the jobs in $\mathcal{J}_{i,j}$ that have not yet been scheduled, and when all current running jobs terminate, start step 1 of phase $i+1$;

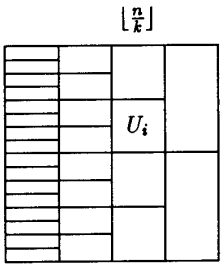

Figure 2: Mesh Partition and the Submeshes of the Prudent Algorithm

Claim 3.14 *Let p be the number phases of the algorithm above. For all $0\le i\le p-1$ and $1\le j<k$, $h_{i,j+1}\le h_{i,j}/2$ and $h_{i+1,1}\le h_{i,k}/2$.*

Proof: Following Lemma 3.1 we have that if $\mathcal{J}_{i,j}^{(l)}$ is not empty at the checkpoint of step j in phase i, then the efficiency of the schedule for jobs in $\mathcal{J}_{i,j}^{(l)}$ (with respect $U_{i,j,l}$) is at least 1/4 at this checkpoint. Therefore, by Lemma 3.9, the efficiency (with respect to U_j) at this checkpoint is at least $h_{i,j+1}/4h_{i,j}$. On the other hand, by the definition of the checkpoint, this efficiency is less than 1/8. Hence we proved the first part of the lemma. The second part of the lemma holds for the same reason. □

Consequently the number of phases p is bounded by $p\le\log\log(order(\mathcal{J}))/k+1$.

By choosing $k=\lceil\sqrt{\log\log order(\mathcal{J})}\rceil$, we prove that the prudent-parallel-serial-partition-algorithm is $O(\sqrt{\log\log order(\mathcal{J})})$-competitive for $\mathcal{J}$. This proves Theorem 2.4

4 Lower bound

We prove that no on-line dynamic scheduling on $n\times n$ mesh can achieve a competitive ratio better than

$\frac{1}{42}\sqrt{\log\log n}$. This proves that the algorithm in Section 3 is optimal up to a constant factor.

The adversary strategy tries to restrict the possibilities of the scheduler so that he has to act in a similar way as in the optimal algorithm we presented in Section 3. The key technical point is Lemma 4.1 which shows that the adversary can restrict the actions of the scheduler substantially. He will keep running a small portion of jobs which will effectively block a large space and thus he will prevent efficient scheduling.

4.1 Notation

For the proof of lower bound it is more convenient to represent mesh as a square with both coordinates running through the real interval $[0,n]$. A processor then corresponds to a unit square and a $x\times y$-submesh at (X,Y) corresponds to the $x\times y$ rectangle with the lower left corner at (X,Y). We will refer to these rectangles as to submeshes.

During the proofs we will also use rectangles with non-integer dimensions and coordinates. A *normal $x\times y$-rectangle* is a rectangle with width x and height y such that its left coordinate is an integer multiple of its width and its bottom coordinate is an integer multiple of its height. (Thus whole mesh can be partitioned into a set of non-intersecting normal $x\times y$-rectangles and a small leftover, if x and y are smaller than n). A *normal (x,y)-rectangle* is a normal $x\times y$- or $y\times x$-rectangle. We say that a rectangle intersects a set of rectangles $\mathcal{R}$, if $R\cap R'$ has a non-zero area for some $R'\in\mathcal{R}$.

Now we are ready to define the set of jobs used by the adversary.

Put $k=\lfloor\frac{1}{3}\sqrt{\log\log n}\rfloor$, $s=\lceil(\log\log n)^2\rceil$, $t=\lfloor\frac{1}{2}\log_s n\rfloor$.

We have $t+1$ different job classes, $\mathcal{J}=\mathcal{J}_0\cup\ldots\cup\mathcal{J}_t$. A job class $\mathcal{J}_j$ contains nk^2 jobs of size $n/s^j\times s^j$ (the times of jobs will be determined dynamically by the adversary depending on the strategy of the on-line scheduler). Note that for $i\le j\le t$, $n/s^i\ge n/s^j\ge s^j\ge s^i$.

Suppose we have some (on-line) scheduler running on this set of jobs. For $I\subseteq[0..t]$, let $\mathcal{C}(I)$ denotes all submeshes corresponding to the currently running jobs from $\mathcal{J}_j$, $j\in I$. Let $\mathcal{C}=\mathcal{C}([0..t])$, i.e. $\mathcal{C}$ denotes the set of submeshes corresponding to all currently running jobs.

4.2 Adversary strategy

The adversary strategy is based on the following lemma that will be proved in Section 4.3.

Lemma 4.1 *Let $I'=[a..b]$, $0\le a\le b\le t$, $\overline{I'}$ denotes $[0..t]-I'$. There exists a set $\mathcal{C}'\subseteq\mathcal{C}(\overline{I'})$ such that $\mathrm{eff}(\mathcal{C}')\le 1/8k$ and every normal $(n/4ks^b, s^a/4k)$-rectangle intersected by $\mathcal{C}(\overline{I'})$ is intersected also by $\mathcal{C}'$.*

Now we are ready to present the *adversary algorithm*. The adversary maintains the *active interval* denoted by I. I starts from $[0..t]$, it gradually decreases and never grows. Let T be some fixed time. The adversary reacts to the schedulers actions by the following steps.

SINGLE JOB: If the scheduler starts a job in a submesh of area less then n/k, then the adversary removes all other jobs (both running and waiting) and runs this single job for a sufficient amount of time.
DUMMY: If the scheduler starts a job that does not belong to the active interval (i.e. a job from $\mathcal{J}_j$, $j\notin I$), adversary removes it immediately.
CLEAN UP: If the time since the last CLEAN UP (or since the beginning) is equal to T and there was no SINGLE JOB step, adversary removes all running jobs.
DECREASE EFFICIENCY: If $\mathrm{eff}(\mathcal{C})$ exceeds $1/k$, the adversary does the following: Take an interval $I'\subseteq I$ such that $\mathrm{length}(I')=\lfloor\mathrm{length}(I)/2\rfloor$ and $\mathrm{eff}(\mathcal{C}(I'))\le\mathrm{eff}(\mathcal{C}(I))/2$ (such I' obviously exists - take either upper or lower half of I, whichever has lower eff). Take $\mathcal{C}'$ from Lemma 4.1. Remove all jobs except for $\mathcal{C}(I')\cup\mathcal{C}'$ and set the active interval to I'.

In the next two sections we prove that this strategy ensures that the competitive ratio is at most $\frac{1}{42}\sqrt{\log\log n}$ thus proving Theorem 2.5.

4.3 Evaluation of the Adversary Strategy

In this section we actually prove the lower bound comparing $T_S(\mathcal{J})$ for an on-line schedule S to $T_{min}(\mathcal{J})$. In Section 4.4 we prove that T_{opt} differs from T_{min} by a constant factor only.

We prove that the adversary strategy from section 4.2 achieves that $T_S(\mathcal{J})\ge kT_{min}(\mathcal{J})$ for the resulting schedule S. We use all notation from Section 4.

If the scheduler allows a SINGLE JOB step, then it is obvious. Starting a job that does not belong to the active interval and immediately removing it by a DUMMY step does not change the schedule essentially.

So we can assume that the scheduler always starts the jobs from the active interval only and adversary does only DECREASE EFFICIENCY and CLEAN UP steps. CLEAN UP steps divide the schedule into *phases*.

We prove that there can be only $6k$ of DE-

CREASE EFFICIENCY steps in one phase. This shows that every schedule will have at least k phases thus proving that $T_S(\mathcal{J}) \geq kT \geq kt_{max}(\mathcal{J})$. During whole schedule the efficiency is at most $1/k$, which proves that $T_S(\mathcal{J}) \geq kT_{eff}(\mathcal{J})$

First we prove Lemma 4.1 using the following Claim.

Claim 4.2 *Let $\mathcal{D}$ be a set of submeshes of height at most y/v for given y and v. There exists $\mathcal{D}' \subseteq \mathcal{D}$ such that $\text{eff}(\mathcal{D}') \leq 2/v$ and each normal $1 \times y$-rectangle intersected by $\mathcal{D}$ is intersected also by $\mathcal{D}'$.*

Proof: Let R be a normal $n \times y$-rectangle. We will define $\mathcal{D}_R \subseteq \mathcal{D}$, $\text{eff}(\mathcal{D}_R) \leq 2y/vn$ such that it intersects all columns of R intersected by $\mathcal{D}$. Then it is sufficient to take $\mathcal{D}' = \bigcup_R \mathcal{D}_R$, since every normal $1 \times y$-rectangle is a column of some normal $n \times y$-rectangle and there are only $\lfloor n/y \rfloor$ normal $n \times y$-rectangles.

Given R, define $\mathcal{D}_0, \ldots, \mathcal{D}_n$ as follows: $\mathcal{D}_0 = \emptyset$. Put

$$\mathcal{D}_i = \begin{cases} \mathcal{D}_{i-1} \cup \{D_i\} & \text{if the } i\text{th column of } R \text{ is intersected by } \mathcal{D} \text{ and not by } \mathcal{D}_{i-1}, \\ \mathcal{D}_{i-1} & \text{otherwise,} \end{cases}$$

where $D_i \in \mathcal{D}$ is a submesh intersecting the ith column with the maximal possible right coordinate. Put $\mathcal{D}_R = \mathcal{D}_n$. From the selection of D_i it follows that no column of R is intersected by more than two submeshes from $\mathcal{D}_R$. Since every submesh form $\mathcal{D}_R$ has height at most y/v and intersects some column of R, we have $\text{eff}(\mathcal{D}_R) \leq 2y/vn$. □

Proof of Lemma 4.1:

Divide $\mathcal{C}(\overline{I'})$ into three classes as follows:
$\mathcal{C}_1$ – submeshes with height at most s^{a-1},
$\mathcal{C}_2$ – submeshes with both height and width at most n/s^{b+1} and
$\mathcal{C}_3$ – submeshes with width at most s^{a-1}.

All submeshes from $\mathcal{C}([0..a-1])$ fall into $\mathcal{C}_1$ or $\mathcal{C}_3$ (depending on their orientation) and all submeshes from $\mathcal{C}([b+1..t])$ into $\mathcal{C}_2$, hence $\mathcal{C}(\overline{I'}) = \mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{C}_3$. Now we apply Claim 4.2 four times as follows:
Get $\mathcal{D}_1 \subseteq \mathcal{C}_1$ intersecting all normal
$1 \times s^a/4k$-rectangles intersected by $\mathcal{C}_1$;
get $\mathcal{D}_2 \subseteq \mathcal{C}_2 \cup \mathcal{C}_3$ intersecting all normal
$n/4ks^b \times 1$-rectangles intersected by $\mathcal{C}_2 \cup \mathcal{C}_3$;
get $\mathcal{D}_3 \subseteq \mathcal{C}_1 \cup \mathcal{C}_2$ intersecting all normal
$1 \times n/4ks^b$-rectangles intersected by $\mathcal{C}_1 \cup \mathcal{C}_2$;
get $\mathcal{D}_4 \subseteq \mathcal{C}_3$ intersecting all normal
$s^a/4k \times 1$-rectangles intersected by $\mathcal{C}_3$.
So $\mathcal{D}_1 \cup \mathcal{D}_2$ intersects all normal $n/4ks^b \times s^a/4k$-rectangles intersected by $\mathcal{C}(\overline{I'})$ and $\mathcal{D}_3 \cup \mathcal{D}_4$ intersects all normal $s^a/4k \times n/4ks^b$-rectangles intersected by $\mathcal{C}(\overline{I'})$.

Now take $\mathcal{D} = \mathcal{D}_1 \cup \mathcal{D}_2 \cup \mathcal{D}_3 \cup \mathcal{D}_4$. In all four applications of Claim we have $v = s/4k$, so $\text{eff}(\mathcal{D}) \leq 32k/s \leq 1/8k$ for sufficiently large n. □

The next claim is the key to the whole proof. It shows that the adversary strategy does not allow the scheduler to reuse the space efficiently.

Claim 4.3 *Let $j \in I$ and let R be a normal $(n/4ks^j, s^j/4k)$-rectangle that does not intersect $\mathcal{C}$ at some point. Then R never intersected $\mathcal{C}$ since the beginning of the current phase.*

Proof: It is sufficient to prove that no previous DECREASE EFFICIENCY step removed all jobs intersecting R, because by our assumption there are no other steps removing jobs during the phase. Let the active interval before some previous DECREASE EFFICIENCY step be $[a..b]$. We have $a \leq j \leq b$ (since the active interval never grows). So the dimensions of R are integer multiples of the dimension of the normal rectangles considered in Lemma 4.1. Thus R can be partitioned into a set $\mathcal{R}$ of these rectangles without any leftover. By the assumption, the rectangle R, and hence some rectangle from $\mathcal{R}$, is intersected by $\mathcal{C}$ before the DECREASE EFFICIENCY step. From the construction and Lemma 4.1 it follows that this rectangle, and hence also R, is intersected by $\mathcal{C}$ also after this step. □

Claim 4.4 *There can be at most $6k$ DECREASE EFFICIENCY steps in one phase.*

Proof: Suppose that the scheduler starts a job from $\mathcal{J}_j$, $j \in I$, in a submesh C. Because it does not cause a SINGLE JOB step, its dimensions are at least n/ks^j and s^j/k. Thus at least half of its area consists of normal $(n/4ks^j, s^j/4k)$-rectangles. By previous claim it follows that at least half of its area was not used during the current phase.

The efficiency $\text{eff}(\mathcal{C})$ immediately before the DECREASE EFFICIENCY step is at most $1/k + 1/n$ (since the threshold $1/k$ was reached by the last started job), so by construction and Lemma 4.1, $\text{eff}(\mathcal{C})$ after the step is at most $1/2k + 1/2n + 1/8k \leq 2/3k$ for sufficiently large n.

So in any phase the scheduler has to start jobs corresponding to the current active interval with eff at least $1/3k$, thus using at least $1/6k$ of the area that was never used before in the phase. This can be done at most $6k$ times. □

Theorem 4.5 *The adversary strategy forces that $T_S(\mathcal{J}) \geq \frac{1}{3}\sqrt{\log\log n}\, T_{min}(\mathcal{J})$.*

Proof: First we prove that after k phases the active interval I is nonempty. I is halved in each DECREASE

EFFICIENCY step, otherwise it is unchanged. During k phases there are at most $6k^2 \leq \frac{2}{3}\log\log n$ DECREASE EFFICIENCY steps. At the beginning the length of I is $t+1 \geq \frac{1}{2}\log_s n = \frac{\log n}{2\log\lceil\log\log n^2\rceil} > 2^{\lceil\frac{2}{3}\log\log n\rceil}$ for sufficiently large n. Thus it cannot decrease to 0.

Let j be in the active interval at the end of the kth phase. Then it was in the active interval during all k phases and adversary removed the jobs from $\mathcal{J}_j$ only in the CLEAN UP steps. During one clean up step he removed at most kn of these jobs (because of their area), so some of them were not finished before the end of the kth phase. Hence $T_S(\mathcal{J}) \geq kT \geq kt_{max}(\mathcal{J})$.

During whole schedule the efficiency was at most $1/k$, hence $T_S(\mathcal{J}) \geq kT_{eff}(\mathcal{J})$ and $T_S(\mathcal{J}) \geq k \times \max(T_{eff}(\mathcal{J}), t_{max}(\mathcal{J})) = kT_{min}(\mathcal{J})$. □

This together with the results of the next section proves Theorem 2.5.

4.4 Off-line scheduling

For the proof of the lower bound we still need to prove that an off-line scheduler on the 2-dimensional mesh can schedule jobs in a constant multiple of T_{min}. The problem is essentially a special case of three dimensional bin packing.

Let $\mathcal{J}$ be a set of jobs and $t = t_{max}(\mathcal{J})$. Divide $\mathcal{J}$ into the classes $\mathcal{J}_{l,j} = \{J_i \in \mathcal{J} | \frac{n}{2^{l+1}} < a_i \leq \frac{n}{2^l}, \frac{n}{2^{j+1}} < b_i \leq \frac{n}{2^j}\}$, $0 \leq l \leq j \leq \lfloor\log n\rfloor$. Now partition each $\mathcal{J}_{l,j}$ into $\mathcal{D}'_{l,j}$ and $\mathcal{D}_{l,j,u}$, $u = 1, \ldots, u_{l,j}$ so that $t/2 \leq \sum_{J_i \in \mathcal{D}_{l,j,u}} t_i \leq t$ and $\sum_{J_i \in \mathcal{D}'_{l,j}} t_i \leq t$. This can be done by the first fit bin packing algorithm. For each $\mathcal{D}_{l,j,u}$ ($\mathcal{D}'_{l,j}$) there exists a schedule $S_{l,j,u}$ ($S'_{l,j}$) for a $\frac{n}{2^l} \times \frac{n}{2^j}$-mesh in time t with efficiency at least $1/8$. Now we can think of each of these schedules as a single job. Because of their special sizes and times it is easy to schedule them.

First let us consider $S_{l,j,u}$'s for a fixed l. Sort them by increasing j and by first fit bin packing divide them into schedules $S_{l,v}$, $v = 1, \ldots, v_l$ and S'_l, all on $\frac{n}{2^l} \times n$-mesh in time t. Because of special sizes, all $S_{l,v}$ have efficiency at least $1/8$ (even if n is not a power of 2).

Similarly divide all $S_{l,v}$'s into schedules S_w on $n \times n$-mesh in time t with efficiency at least $1/8$ and a leftover S'.

The final schedule $S(\mathcal{J})$ will be as follows.

ALGORITHM: *off-line*
1. Run all schedules S_w one by one.
2. Run S'.
3. Run S'_0.
4. Run simultaneously all $S'_l, 1 \leq l \leq \lfloor\log n\rfloor$, place S'_l at $(n/2^l, 0)$.
5. Run $S'_{0,0}$.
6. Run simultaneously all $S'_{0,j}, 1 \leq j \leq \lfloor\log n\rfloor$ at $(0, n/2^j)$.
7. Run simultaneously all $S'_{l,j}, 1 \leq l \leq \lfloor\log n\rfloor$ at $(n/2^l, n/2^j)$.

Theorem 4.6 *For every set of jobs $\mathcal{J}$ and $S(\mathcal{J})$ as above, $T_S(\mathcal{J}) \leq 14T_{min}(\mathcal{J})$.*

Proof: The phase 1 runs with the average efficiency at least $1/8$, rest of the schedule takes at most $6t$. Hence $T_S(\mathcal{J}) \leq 8T_{eff}(\mathcal{J}) + 6t_{max}(\mathcal{J}) \leq 14T_{min}(\mathcal{J})$. □

This together with Theorem 4.5 gives the following theorem and hence also Theorem 2.5

Theorem 4.7 *No on-line dynamic scheduler on $n \times n$-mesh can achieve the competitive ratio better than $\frac{1}{42}\sqrt{\log\log n}$.* □

5 Other Architectures

In this section, we extend the results on $n \times n$ mesh to various parallel architectures including hypercube, d-dimensional mesh, trees, mesh of trees [9], and PRAM. Due to the space limitation, the proofs will appear in the full version of the paper only.

5.1 The d-dimensional Mesh

We have shown that our upper bound method and results for $n \times n$ meshes can be generalized to the d-dimensional $n \times \cdots \times n$ mesh.

Theorem 5.1 *There is an on-line scheduling algorithm that is $\Theta(\sqrt{\log\log n})$-competitive on d-dimensional $n \times \cdots \times n$ mesh. Moreover, when d is a constant, the algorithm is optimal up to a constant factor.*

5.2 Hypercube

A d-dimensional *hypercube* can be viewed as a combination of two $(d-1)$-dimensional hypercube and in general as 2^r of $(n-r)$-dimensional hypercubes. Each job J in an hypercube-based architecture is characterized by a pair of number (r, t), where r is the dimension of the hypercube it requires and t is the run time of J on a r-dimensional hypercube. We assume that each job $J = (r, t)$ can be processed on an r'-dimensional hypercube ($r' \leq r$) in $2^{r-r'}t$ time.

We can modify the *largest-job-first-greedy* algorithm presented in Section 3 for scheduling jobs on hypercube and prove

Theorem 5.2 *The* largest-job-first-greedy-algorithm *is 2-competitive for hypercube.*

5.3 The PRAM

PRAM is one of the most widely used models for parallel programming [12, 6]. Here, each job on PRAM is characterized by (p,t) where p is the number of processors required and t is the run time. We assume that a job $J = (p,t)$ can be processed on $p' \leq p$ processors in $(p/p')t$ time.

The PRAM job scheduling problem was studied before under the name of *Gang Scheduling* by Feitelson and Rudolph [4] and by Shmoys, Wein and Williamson [14] where they stated a 3-competitive on-line algorithm.

We can generalize the *balanced-greedy-algorithm* to schedule jobs on PRAM and prove

Theorem 5.3 *Under the PRAM simulation assumption, the* balanced-greedy-algorithm *is* $2\sqrt{2}$*-competitive.*

5.4 Other Architectures

For other architectures we got the following results.

Theorem 5.4 *There is an* $\Theta(\sqrt{\log \log n})$*-competitive scheduling algorithm for 2-dimensional mesh of trees.*

There is a 3-competitive on-line scheduling algorithm for trees.

6 Concluding Remark

In this paper we have made the assumption that just the running times are given dynamically. The principle question which remains open is, how much the results can be extended towards more general models, when there are dependencies between jobs. Another issue concerns with the model of dynamic scheduling, e.g., whether jobs can be scheduled preemptively or with zero cost restart [14]. We have observed that on-line scheduling algorithms with constant competitive ratio exist when zero cost restart is assumed. However, there are many practical and theoretical problems with such an assumption. We are still searching for better model which more faithfully captures the practical job scheduling problems. Also of interest is whether randomization can help to improve the performance of the scheduling algorithm.

Acknowledgements:

We would like to thank Steven Rudich, Danny Sleator, Ming-Yang Kao and David Peleg for helpful discussions and comments.

References

[1] S. N. Bhatt, F.R.K. Chung, J.-W.Hong, F.T. Leighton, and A.L. Rosenberg. Optimal simulations by butterfly networks. In *ACM STOC*, pages 192–204. 1988.

[2] E. Davis and J. M. Jaffe. Algorithm for scheduling tasks in unrelated processors. *JACM*, pages 712–736, 1981.

[3] Jr. E. G. Coffman, M. R. Garey, and D. S. Johnson. Dynamic bin packing. *SIAM J. Computing*, 12(2):227–260, 1983.

[4] F. G. Feitelson and L. Rudolph. Wasted resources in gang scheduling. In *Proc. of the 5th Jerusalem Conference on Information Technology*, 1990.

[5] R. L. Graham. Bounds for certain multiprocessor anomalies. *Bell System Technical Journal.*, pages 1563–1581, 1966.

[6] R. M. Karp and V. Ramachandran. A survey of parallel algorithms for shared memory machines. Technical Report UCB/CSD 88-408, UC. Berkeley, EECS, 1988.

[7] S. R. Kosaraju and M. J. Atallah. Optimal simulation between mesh-connected arrays of processors. In *ACM STOC*, pages 264–272. 1986.

[8] H. T. Kung. Computational Models for Parallel Computers. Technical report, CMU-CS-88-164, 1987.

[9] F. T. Leighton. *Complexity Issues in VLSI.* Foundations of Computing. MIT Press, Cambridge, MA, 1983.

[10] J. K. Lenstra, D. B. Shmoys, and E. Tardos. Approximation algorithm for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271, 1990.

[11] J. L. Peterson and A. Silberschatz. *Operating System Concepts.* Addison-Wesley Publishing Company, 1985.

[12] A. Ranade. How to emulate shared memory. In *IEEE FOCS*, pages 185–194, Los Angeles, Oct 1987.

[13] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *CACM*, 28(2):202–208, 1985.

[14] D. Shmoys, J. Wein and D. P. Williamson. On-line scheduling of parallel machines. This Proceeding (FOCS 1991).