Computer Science
Technical Report

Colorado State University

Preemptive Job Scheduling in Star Graph Networks*

Shahram Latifi
Department of Electrical Engineering
University of Nevada
Las Vegas, NV 89154

Pradip K. Srimani
Department of Computer Science
Colorado State University
Ft. Collins, CO 80523

Technical Report CS-96-119

Computer Science Department
Colorado State University
Fort Collins, CO 80523-1873

Phone: (970) 491-5792 Fax: (970) 491-2466
WWW: http://www.cs.colostate.edu

*To appear in the ***Proceedings IEEE International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP-96)***

Preemptive Job Scheduling in Star Graph Networks*

Shahram Latifi
Department of Electrical Engineering
University of Nevada
Las Vegas, NV 89154

Pradip K. Srimani
Department of Computer Science
Colorado State University
Ft. Collins, CO 80523

Abstract

In this paper we develop a feasibility algorithm for preemptively scheduling a given set of jobs with dimension and time requirements on a star graph network of given size with a given deadline. We show that the algorithm runs in $\mathcal{O}(n \log n)$ time where n is the number of jobs.

1 Introduction

One of the attractive topologies for constructing the symmetric interconnection networks is the star graph [AK89, AK87]. The star graph, being a member of the class of Cayley graphs, has been shown to possess appealing features including low degree of the node, small diameter, partitionability, symmetry, and high degree of fault-tolerance. For this reason, recently much research has been directed toward studying properties of these star graphs [DT94, QMA92], its fault-tolerance aspects [Lat93], or implementing various algorithms on it [QAM94, MS90, FA91, MS92].

When parallel algorithms are mapped and implemented on a massively parallel architecture, the dimension of the network plays an important role as a parameter of the algorithm. This is especially true for highly regular and hierarchical networks such as the hypercube and star graph. Depending on the size of the incoming task, one portion of the network (which preserves the topological properties of the

*To appear in the ***Proceedings IEEE International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP-96)***

original network) is allocated to it; subsequent tasks are then assigned to disjoint subnetworks and if no subnetwork of the required size is available, the task(s) are queued until some tasks run to completion and make subnetworks with the required size available. There are many important issues of concern in this area. One of the fundamental problem is of preemptive scheduling of independent jobs (each with a processing time and a size requirement); while the ultimate goal is to compute the minimum finish time given the network size, the more immediate (and easier) problem is to determine the feasibility of scheduling given a network and a specific deadline. This problem has been recently solved for hypercube networks [AZ90, ZA93]. Our purpose in the present paper is to propose a feasibility algorithm that decides if a given set of jobs, where each job is associated with a dimension d and a processing time t (meaning that the job needs be processed on a d-dimensional star graph for t units of time), can be preemptively scheduled on an m-dimensional star graph system within a given deadline T. We also show that our algorithm has a $\mathcal{O}(n \log n)$ run time complexity where n is the number of jobs in the given set of jobs.

2 Basic Preliminaries

In this section we briefly introduce the basic terminology about star graphs and scheduling of jobs in star graph networks. Graph theoretic terms not defined here can be found in [Har72] and a detailed treatment of star graphs can be found in [AK89, AK87].

A star graph S_n, of order n, is defined to be a symmetric graph $G = (V, E)$ where V is the set of $n!$ vertices, each representing a distinct permutation of n elements and E is the set of symmetric edges such that two permutations (nodes) are connected by an edge iff one can be reached from the other by interchanging its first symbol with any other symbol. For example, in S_3, the node representing permutation 123 has edges to two other permutations (nodes) 213 and 321. Throughout our discussion we denote the nodes by permutations of English numerals. For example, the identity permutation is denoted by $I = (1234...)$.

Remarks:

- These star graphs are members of the family of Cayley group graphs. For a star graph S_n of di-

mension n, there are $n-1$ generators, $g_2, g_3, \cdots, g_n$, where g_i swaps the first symbol with the i-th symbol of any permutation. Each generator is its own inverse, i.e., the star graph is symmetric. Also, the star graph S_n is a $(n-1)$-regular graph with $n!$ nodes and $n!(n-1)/2$ edges.

- It is easy to see that any permutation of n elements can also be specified in terms of its cycle structure with respect to the identity permutation I. For example, $345216 = (135)(24)(6)$. The maximum number of cycles in a permutation of n elements is n and the minimum number is 1. When a cycle has only one symbol, that symbol is in its correct position in the permutation with respect to the identity permutation. The singleton cycles may be omitted in the cycle representation of a permutation if the number of symbols in the permutation is understood from the context.

The problem of job scheduling on star graphs can be formulated as follows. We are given a set of n independent jobs $J = \{J_i : 1 \leq i \leq n\}$ and a star graph S_m of dimension m. Each job $J_i = (d_i, t_i), 1 \leq i \leq n$ requires a star graph of dimension d_i (i.e., a d_i-substar) for t_i units of time where $0 \leq d_i \leq m$ and t_i is a rational number, $t_i > 0$. The problem is to compute a schedule such that the finish time (the time when all jobs are finished) is minimized (we call this an *optimal schedule*). A schedule is called *preemptive* if a job may be preempted before completion and can resume at a later time, possibly on a different substar. We also assume, for the sake of simplicity (without any loss of rigor) that the jobs are ordered, i.e., $\forall i, 1 \leq i \leq n, d_i \geq d_{i+1}$.

Each S_m contains m disjoint S_{m-1}'s. Let Λ be the symbol set $\{1, 2, \ldots m-1, m, x\}$, where x denotes a *don't care* symbol. Every substar of S_m can be uniquely labeled by a string of symbols in Λ such that the only repeated symbol be x. Notably, the number of x symbols in the label determines the dimension of the substar. For instance, the substar $x3x2x$ is 3-dimensional and contains the set of nodes $\{13425, 13524, 43125, 43521, 53124, 53421\}$. The first position (i.e. the leftmost position) of the label of any substar is always equal to x due to connectivity conditions of the star graph, unless the substar is a single node. The S_m is labeled as x^m, where the superscript is the repetition factor. It is also well known [Knu72] that all the $m!$ permutations of m distinct symbols can be uniquely numbered from 0 through $m! - 1$. We use this scheme to number the vertices of any star graph S_m; for details of the numbering scheme, see [Knu72].

Definition 1 *For any $a, b \in V$ with $a < b$, let $[a, b]$ denote the set of processors $\{p \in V : a \leq p \leq b\}$. We call $[a, b]$ a* processor interval *or a* p-interval.

Remark 1 *For any given $\ell, \ell > 0$, an m-star S_m can be divided into ℓ consecutive p-intervals $[a_1, b_1], \cdots [a_\ell, b_\ell]$ where $a_1 = 0$, $b_\ell = m! - 1$ and $(\forall i : 1 \leq i < \ell : a_{i+1} = b_i + 1)$.*

Remark 2 *Not all p-intervals of size $x!$ (x is a positive integer) are x-substar; in this paper we are interested only in those p-intervals which are valid substars and hence we use the terms p-interval and substar interchangeably.*

Definition 2 *The* profile *[AZ90, ZA93] of a schedule is defined to be a function F that maps a processor $p \in V$ to a time $f = F(p)$ such that the processor p has been busy until time f and f denotes the time when the processor p is available for more work.*

So, if T denotes the given deadline for the job set, $r = T - f$ denotes the *Remaining Processing Time* or the RPT of the processor p. If we attempt to find the schedule one job at a time, we need to know the finish time of all the processors for the existing schedule and this information is stored in the profile. We use $S(i)$ to denote the schedule after job J_i is scheduled and use $P(i)$ to denote the corresponding profile.

When we schedule the jobs on a star (each job needs a substar of some dimension), the profile function maps a p-interval (all the processors in the interval) to a time. Thus, the profile of the complete schedule on the star is a sequence of ordered pairs of p-intervals and finish times

$$P = ([a_1, b_1], f_1), ([a_2, b_2], f_2), \cdots, ([a_y, b_y], f_y)$$

for some integer y where the y intervals are consecutive and divide the given m-star. Again, we logically extend the concept of RPT to the intervals; RPT of an interval is the RPT of its processors; more specifically, for a give deadline of the jobs, $r_j = T - f_j$ will denote the RPT of the p-interval $[a_j, b_j]$.

Remark 3 *If a p-interval has zero RPT in a schedule, it cannot be used for scheduling further jobs and will be deleted from the profile.*

Definition 3 *A profile P is called* stair-like *[ZA93] if $\forall i : f_{i+1} < f_i$.*

3 Feasibility Algorithm

Given a set of jobs $J = \{J_1, J_2, \cdots, J_n\}$, where $J_i = (d_i, t_i)$ as explained earlier and an m-star, the feasibility algorithm computes if the given jobs can be scheduled on the m-star to meet a given deadline T. Obviously, if the given deadline T is feasible, we must have $\forall i : 1 \leq i \leq n : T \geq t_i$ and $T \geq \frac{1}{m!} \sum_{i=1}^{n} t_i d_i!$. We can safely assume that the given T satisfies both of these requirements or we can declare the deadline to be infeasible.

We assume that the job set J is sorted in descending order of dimensions of the substars needed, as explained earlier. We attempt to schedule the jobs in this order one at a time. Let $S(i)$ and $P(i)$ denote respectively the schedule and the profile after the job J_i is scheduled. $S(0)$ is the initial schedule (null) and $P(0)$ is the initial profile (before any job is scheduled). So, $P(0) = ([0, m! - 1], 0)$. We use k to denote the number of p-intervals with nonzero RPT in the profile $P(i-1)$. If $k = 0$, job J_i cannot be scheduled; otherwise $P(i-1)$ will look like

$$P(i-1) = ([a_1, b_1], f_1), ([a_2, b_2], f_2), \cdots, ([a_k, b_k], f_k)$$

[Note: if this profile is stair-like, the p-intervals in $P(i-1)$ are ordered in increasing order of their RPTs.]

The Algorithm to schedule $J_i = (d_i, t_i)$

Step 1: If $t_i > r_k$, then return "infeasible" (Job J_i cannot be scheduled).

Step 2: If $t_i < r_1$, then schedule job J_i *entirely* on the substar (p-interval) $[a_1, a_1 + d_i! - 1]$ from time f_1 to time $f_1 + t_i$.

Step 3: If there exists an integer j such that $t_i = r_j$, then schedule the job J_i entirely on the substar $[a_j, a_j + d_i! - 1]$ to use up all its RPT.

Step 4: Compute an integer j such that $t_i > r_j \wedge t_i \leq r_{j+1}$; schedule the job J_i on the substar $[a_j, a_j + d_i! - 1]$ to use up all its RPT r_j and schedule the remaining time $t_i - r_j$ of job J_i on the substar $[a_{j+1}, a_{j+1} + d_i! - 1]$ from time f_{j+1} to time $f_{j+1} + (t_i - r_j)$.

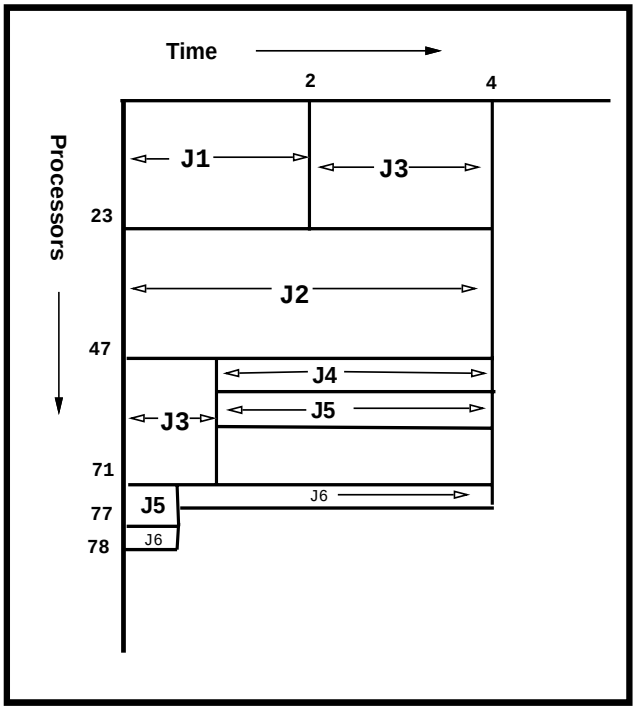

Figure 1: Schedule for the Example Job Set

Remark 4 *For any job J_i if Step 1 does not apply, our algorithm is able to schedule the job by either one of the 3 steps 2, 3 or 4.*

Remark 5 *Note that application of the steps of the algorithm involves appropriate update of the profile; scheduling of a job J_i may split a particular p-interval into two or may necessitate deletion of a p-interval (due to its RPT being completely used up). This updating of the profile P will depend on the data structure used and is not relevant to the correctness of the scheduling algorithm.*

Example: Consider a 5-star, a deadline $T = 4$, and a Job set $J = (J_1, J_2, J_3, J_4, J_5, J_6)$, where $J_1 = (4, 2)$, $J_2 = (4, 4)$, $J_3 = (4, 3)$, $J_4 = (3, 3)$, $J_5 = (3, 3.5)$ and $J_6 = (1, 4)$. Note that the jobs are arranged in nonincreasing order of dimension. The initial profile is $([0, 119], 0)$ and Figure 1 shows the final schedule obtained by the algorithm. We show below the profiles generated after scheduling each job in the set:

$$([0,119],0) \xrightarrow[\text{Step 2}]{J_1 \text{ scheduled}} ([0,23],2),([24,119],4) \xrightarrow[\text{Step 3, } j=2]{J_2 \text{ scheduled}} ([0,23],2),([48,119],0)$$

$$\xrightarrow[\text{Step 4,} j=1]{J_3 \text{ scheduled}} ([48,71],1),([72,119],0) \xrightarrow[\text{Step 3, } j=1]{J_4 \text{ scheduled}} ([54,71],1),([72,119],0) \xrightarrow[\text{Step 4, } j=1]{J_5 \text{ scheduled}}$$

$$([60,71],1),([72,77],0.5),([78,119],0) \xrightarrow[\text{Step 4, } j=2]{J_6 \text{ scheduled}} ([60,71],1),([73,78],0.5),([79,119],0)$$

Lemma 1 *The profiles $P(i), 0 \leq i \leq n$ are stair-like.*

Proof : The profile $P(0)$ is trivially stair-like. Assume that $P(i-1)$ is stair-like; we need to show that $P(i)$ is stair-like after the job J_i is scheduled by the algorithm.

- Assume Step 2 is executed to schedule J_i. There are two cases: if $|[a_1, b_1]| = d_i!$, the profile $P(i)$ is obtained by replacing the first entry $([a_1, b_1], f_1)$ in $P(i-1)$ by an entry $([a_1, b_1], f_1 + t_i)$; else if $|[a_1, b_1]| > d_i!$, then the profile $P(i)$ is obtained by replacing the first entry $([a_1, b_1], f_1)$ in $P(i-1)$ by two elements $([a_1, a_1 + d_i! - 1], f_1 + t_i)$ and $([a_1 + d_i!, b_1], f_1)$. In either case, the resulting profile $P(i)$ maintains the stair-like property.

- Assume Step 3 is executed to schedule J_i. Profile $P(i)$ is obtained by replacing the entry $([a_j, b_j], f_j)$ by a new entry $([a_j + d_i!, b_j], f_j)$; the stair-like property is maintained. Note that if $b_j = a_j + d_i!$, then the original entry is simply deleted.

- Assume Step 4 is executed to schedule J_i. Profile $P(i)$ is obtained by deleting the entry $([a_j, b_j], f_j)$ and replacing the entry $([a_{j+1}, b_{j+1}], f_{j+1})$ by two entries $([a_{j+1}, a_{j+1} + d_i! - 1], f_{j+1} + (t_i - r_j))$ and $([a_{j+1} + d_i!, b_{j+1}], f_{j+1})$. Since $f_{j+1} + (t_i - r_j) \leq f_j$, the stair-like property is maintained in the profile $P(i)$.

□

Lemma 2 *The algorithm generates a feasible schedule iff one exists.*

Proof : We only need to prove that the algorithm generates a schedule if a feasible schedule exists. We use contradiction. Let S' be a feasible schedule of the job set J and the deadline T. Assume that the jobs $J_0, J_1, \cdots, J_{i-1}$ are scheduled in S' in the same way as in $S(i-1)$ and job J_i is scheduled differently

in S' than it would be in $S(i)$. We show that S' can be modified so that J_i is scheduled in S' as in $S(i)$. Thus, the schedule S' can be transformed to $S(n)$, the schedule generated by the proposed algorithm. Let $P(i-1) = ([a_1, b_1], f_1), \cdots, ([a_k, b_k], f_k)$. Since the job J_i is scheduled in $S' - S(i-1)$, $f_k + t_i \leq T$ and hence our algorithm is able to schedule J_i and can generate $S(i)$. Assume our algorithm schedules J_i in $S(i)$ on substar $A = [a_j, a_j + d_i! - 1]$ from time f_j to time $f_j + t_i$ ($= \tau$, say) (Step 2 or 3 of our algorithm); or on substar A from time f_j to T and on substar $B = [a_{j+1}, a_{j+1} + d_i! - 1]$ from time f_{j+1} to time $f_{j+1} + (t_i - r_j)$ ($= \tau'$, say) (Step 4 of our algorithm). If the job J_i is scheduled in S' in the same way, we are done; if not, we rearrange jobs $J_i, J_{i+1}, \cdots, J_n$ in $S' - S(i-1)$ using the following procedure such that J_i is scheduled in S' just like in $S(i)$.

- Divide the entire time interval $[0, T]$ into equal length intervals of size δ (call those intervals δ-intervals) such that each job in S' is preempted or finished at the end of some δ-interval; this can always be done by choosing δ sufficiently small. For an arbitrary δ-interval α, let $JS(\alpha)$ denote the set of jobs (from among $J_i, J_{i+1}, \cdots, J_n$) that are scheduled in S' in the δ-interval α, i.e., $JS(\alpha) = \{J_k : i \leq k \leq n,$ and J_k is scheduled in S' over $\alpha\}$.

- Divide the m-star into $m(m-1)\cdots(m-d_i+1)$ d_i-substars across the entire interval $[0, T]$; line up jobs in $JS(\alpha)$ over each interval α such that no job is scheduled on two d_i-substars – this is possible because $\forall J_k \in JS(\alpha) : d_k \leq d_i$.

- Let $T' = T - t_i$. Divide the schedule S' into two parts: left and right of T'. Let $I_1 = \{\alpha : \alpha$ is a δ-interval on left of T' and $J_i \notin JS(\alpha)\}$ and let $I_2 = \{\alpha' : \alpha'$ is a δ-interval on right of T' and $J_i \in JS(\alpha')\}$. Obviously, number of intervals in I_1 and I_2 are equal. Now we can think of a one-to-one function from I_1 to I_2. Consider an interval α in I_1 and the corresponding α' in I_2. Since the profile $P(i-1)$ is stair-like, number of $d_i!$-substars over α in $S' - S(i-1)$ is at least as many as over α'. Thus, since J_i is over α' and not over α, there is at least a $d_i!$-substar over α which is either an empty interval or occupied by a job in $JS(\alpha) - JS(\alpha')$ – thus we can interchange.

- Now the job J_i is in the interval $[T', T]$; we now move it to the desired subcubes and time intervals as is done in $S(i)$. We use the following rules:

(1) If Step 2 is used to schedule J_i on $S(i-1)$ to produce $S(i)$, J_i is scheduled on substar $A = [a_1, a_1 + d_i! - 1]$ from f_1 to $f_1 + t_i = \tau$. In this case, $T' > f_1$ and $\tau > f_1$. For each α in $[T', T]$ in S', we interchange J_i in its d_i-substar with jobs in A; we then swap J_i in A over $[T', T]$ with that in A over $[f_1, X]$. Because A extends from f_1 to T in $S' - S(i-1)$, the swapping can always be done.

(2) If Step 3 is used to schedule J_i on $S(i-1)$ to produce $S(i)$, J_i is scheduled entirely on substar $A = [a_j, a_j + d_i! - 1]$ from time f_j to time $f_j + t_i = \tau$. In this case, $T' = f_j$ and $T = \tau$. For each α in $[T', T]$ in S', we just interchange J_i in its d_i-substar with jobs in A.

(3) If Step 4 is used to schedule J_i on $S(i-1)$ to produce $S(i)$, J_i is scheduled on substar A from time f_j to T and on substar $B = [a_{j+1}, a_{j+1}+d_i!-1]$ from time f_{j+1} to time $f_{j+1}+(t_i-r_j) = \tau'$. In this case, $f_j > T' > f_{j+1}$ and $f_j > \tau' > f_{j+1}$. For each α in $[f_j, T]$ in S', we interchange J_i in its d_i-substar with jobs in A; we interchange J_i in substar B over $[T', f_j]$ with that in B over $[f_{j+1}, \tau']$. Because B extends from f_{j+1} to T in $S'-S(i-1)$, the swapping can always be done.

□

Theorem 1 *The number of preemptions in a feasible schedule produced by the algorithm is upper bounded by* $n - 1$.

Proof : The first job J_1 is scheduled without any preemption and for each subsequent job we need at most one preemption; thus the result follows. □

Theorem 2 *The feasibility algorithm has a run time complexity of* $\mathcal{O}(n \log n)$.

Proof : The feasibility algorithm involves updating the profile by scheduling one job at a time from the job set starting from a profile of a single entry and assuming that the job set is orderd in non increasing order of dimension requirement. The jobs can be ordered in $\mathcal{O}(n \log n)$ time using a sorting algorithm like heapsort. The profile can be maintained by using some kind of a balanced tree structure like AVL trees. The initial tree contains only one node. Update of the tree for scheduling one job involves, in

the worst case, one deletion and one insertion (i.e., one entry of the profile may need be deleted and one additional entry may need be inserted). Insertion and/or deletion in an AVL tree can be done in $\mathcal{O}(\log n)$ time where n is the number of elements in the tree. Thus, the entire operation of the profile updating can be done in $\mathcal{O}(n \log n)$ time. Lastly, to schedule each job, we need to decide on the particular step of the algorithm. There are only 4 steps in the algorithm; to decide if a particular step is applicable, we need to do a search on the tree which can take at most $\mathcal{O}(\log n)$ time and hence the decision process for all the n jobs will take $\mathcal{O}(n \log n)$ time. □

4 Conclusion

We have developoed a feasibility algorithm to preemptively schedule a set of job with time and dimension requirements on a given star graph with a given deadline. We have shown that the algorithm runs in $\mathcal{O}(n \log n)$ time where n is the number of jobs. It'd be interesting to design strategies to compute the minimal finish time of a job set for a given star graph network as well as to design nonpreemptive scheduling strategies for tasks on a star graph network. We plan to report our results soon.

References

[AK87] S. B. Akers and B. Krishnamurthy. The star graph: an attractive alternative to n-cube. In *Proceedings of International Conference on Parallel Processing (ICPP-87)*, pages 393–400, St. Charles, Illinois, August 1987.

[AK89] S. B. Akers and B. Krishnamurthy. A group-theoretic model for symmetric interconnection networks. *IEEE Transactions on Computers*, 38(4):555–566, April 1989.

[AZ90] M. Ahuja and Y. Zhu. An o(n log n) feasibility algorithm for preemptive scheduling of n independent jobs on a hypercube. *Information Processing Letters*, 35:7–11, June 1990.

[DT94] K. Day and A. Tripathi. A comparative study of topological properties of hypercubes and star graphs. *IEEE Transactions on Parallel and Distributed Systems*, 5(1):31–38, January 1994.

[FA91] P. Fragopoulou and S. G. Akl. Parallel algorithm for computing Fourier transforms on the star graph. In *Proceedings of the International Conference on Parallel Processing*, volume III, pages 100–106, St. Charles, Illinois, 1991.

[Har72] F. Harary. *Graph Theory*. Addison-Wesley, Reading, MA, 1972.

[Knu72] D. E. Knuth. *The Art of Computer Programming, Volume II*. Addison-Wesley, 1972.

[Lat93] S. Latifi. On fault diameter of star graphs. *Information Processing Letters*, 46:143–150, June 1993.

[MS90] A. Menn and A. K. Somani. An efficient sorting algorithm for the star graph interconnection network. In *Proceedings of the International Conference on Parallel Processing*, volume III, pages 1–8, St. Charles, Illinois, 1990.

[MS92] V. E. Mendia and D. Sarkar. Optimal broadcasting on the star graph. *IEEE Transactions on Parallel and Distributed Systems*, 3(4):389–396, July 1992.

[QAM94] K. Qiu, S. G. Akl, and H. Meijer. On some properties and algorithms for the star and pancake interconnection networks. *Journal of Parallel and Distributed Computing*, 22(1), July 1994.

[QMA92] K. Qiu, H. Meijer, and S. G. Akl. On the cycle structure of star graphs. Technical Report 92-341, Department of Computer Science, Queen's University, Ontario, Canada, November 1992.

[ZA93] Y. Zhu and M. Ahuja. On job scheduling on a hypercube. *IEEE Transactions on Parallel and Distributed Systems*, 4(1):62–69, January 1993.