Journal of Universal Computer Science, vol. 3, no. 7 (1997), 803-812
submitted: 5/3/97, accepted: 23/5/97, appeared: 28/7/97 © Springer Pub. Co.

Bounds on the Performance of Work-greedy Assignment Schemes

Sathiamoorthy Manoharan
(Department of Computer Science,
University of Auckland,
New Zealand
mano@cs.auckland.ac.nz)

Abstract: The heuristics most of the current assignment schemes use is based on satisfying the following rule of thumb: keeping the processors busy leads to a 'good' assignment. Such schemes are said to be work-greedy. This paper presents new bounds on the performance of work-greedy schemes, taking into account the degree of parallelism visible between the tasks and the inter-task communication delays.

Key Words: Allocation, Dependency graphs, Instruction-level parallelism, Scheduling, Processor assignment.
Category: A.0

1 Introduction

A program with multiple tasks can be viewed as a dependency graph: the vertices represent the tasks and the edges represent the dependencies between the tasks.

Dependency graphs may have weights associated with their vertices and edges: the weight on a vertex indicates the amount of computation the corresponding task performs, and the weight on an edge indicates the amount of communication between the tasks the edge connects. [1]

Assignment of a dependency graph is a many-to-one mapping function $M : \mathbf{T} \mapsto \mathbf{P}$, which maps the set of tasks $\mathbf{T}$ onto the set of processors $\mathbf{P}$. M is defined for each task of $\mathbf{T}$. In essence, the assignment divides the task set $\mathbf{T}$ into m, some possibly empty, ordered subsets or partitions. Here m is the cardinality of the set $\mathbf{P}$. The total time the set of tasks $\mathbf{T}$ takes to execute on the set of processors $\mathbf{P}$ is called the *makespan*. The objective of the assignment is to minimize the makespan.

A naïve approach to solve the assignment problem is to enumerate all the possible assignments and choose the assignment that gives the minimum makespan. However, this approach will take exponential time. It is very unlikely that there would be any cleverer scheme to find the optimal assignment in polynomial

[1] Another representation for a program with multiple tasks is an interaction graph where the vertices represent the tasks and the edges represent interactions between the tasks. Dependencies are not explicit in an interaction graph. Some models of computation, for instance CSP [Hoare, 1978] or CCS [Milner, 1989], are well suited to the interaction graph forms whilst some other models of computation, for instance a dataflow computation model [Gaudiot et al., 1988], are well suited to the dependency graph forms. The ease of transformation of the program into a suitable graph form thus depends on the user's model of computation. Programs written using PVM, for instance, are easy to model as an interaction graph whereas programs written in SISAL can be easily modelled as a dependency graph.

time, since even the restricted cases of the assignment problem have been proved to be NP-complete [Ullman, 1976, Rayward-Smith, 1987]. Practical assignment schemes thus settle for some heuristics that would find sub-optimal assignments in polynomial time [El-Rewini and Lewis, 1990, Gerasoulis et al., 1990, Shirazi et al., 1995, Manoharan and Thanisch, 1991, Wu and Gajski, 1990].

Most of these heuristic schemes are work-greedy: they do not let a processor idle when there is a task the processor could execute. That is, an assignment is work-greedy if no processor remains idle when there is a task the processor could execute. Work-greedy assignments are time-driven: tasks and processors are selected at specific time instances, i.e. when a processor becomes free or when a task finishes its execution. For a review of some work-greedy schemes, see [El-Rewini et al., 1995] and for a comparison of some of the schemes, see [Manoharan and Topham, 1995].

Work-greedy assignment schemes, in addition to finding *where* to execute a task, attempt to find *when* to execute a task. That is, they always predict the start and finish times of the tasks. This permits computation of bounds on the makespans of work-greedy assignments.

A work-greedy assignment does not guarantee optimality. But, it is possible to show how close to optimal a work-greedy assignment is. This paper presents some new results bounding the makespans of work-greedy assignments.

The rest of this paper is organized as follows. Section 2 presents the bound on the makespan of a work-greedy assignment by taking into account the possible communication delays between the tasks. This bound is an improvement over the bounds presented by Hwang et al. [Hwang et al., 1989], Sarkar [Sarkar, 1989], and Lee et al. [Lee et al., 1988]. Section 3 discusses the implications of these bounds on makespans. The final section concludes with a summary.

1.1 Notations

Some notations that need to be used subsequently are defined here. Other notations may be defined in context.

n	number of tasks.
m	number of processors.
$\mathbf{T}$	set of tasks $\{ T_0, T_1, \ldots, T_{n-1} \}$.
$\mathbf{P}$	set of processors $\{ P_0, P_1, \ldots, P_{m-1} \}$.
τ_i	execution time of T_i assumed common on all P_j.
$v(T_i, T_j)$	volume of information transfer between task T_i and task T_j.
ω	total execution time of $\mathbf{T}$ on $\mathbf{P}$ (i.e. the makespan).

1.2 Assumptions

The primary architectural considerations are the set of processors and the topology in which the processors are connected. The processor topology is modelled as a graph with vertices representing the processors and weighted edges representing the interconnections between the processors. The weight on a processor graph edge represents the message transfer rate between the processors connected by this edge. All the processors are assumed to be capable of doing the functions required by the tasks.

Task graphs are assumed to be acyclic. A dataflow execution model is assumed for the execution of task graphs. That is, a task can begin its execution when all its inputs are available, and finishes only when it has produced all the required outputs. Communication delay may occur when a task sends its output to its successor tasks. This delay is dependent on the volume of information being transferred and the distance the information transfer rates. Tasks, once scheduled, cannot be preempted. Task replication is not considered, that is, no task can execute on more than one processor. Nothing is assumed about the granularity of the tasks: a task may be a procedure; or it may be an instruction.

See Figure 1 for example task and processor graphs. Figure 1(a) shows the task dependency graph corresponding to the evaluation of an expression $z = F(f(x), g(y))$. Figure 1(b) shows a three-processor system where all processors are connected to each other.

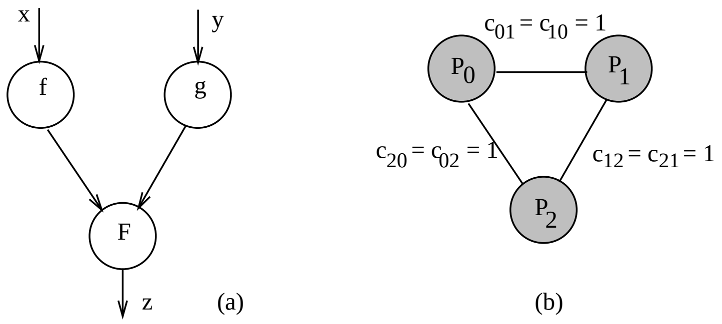

Figure 1: Example task and processor graphs.

2 Bound on the Makespan

In this section we establish a bound on the makespan of work-greedy assignments of dependency graphs, considering communication costs. Examples of work-greedy assignment schemes that consider communication costs include ETF [Hwang et al., 1989], ERT [Lee et al., 1988], MCP [Wu and Gajski, 1990] and MH [El-Rewini and Lewis, 1990].

Hwang et al. [Hwang et al., 1989] and Lee et al. [Lee et al., 1988] proved bounds on the makespans of ETF and ERT. They have proved that

$$\omega' \leq \left(2 - \frac{1}{m}\right) \omega^i + C_{comm}$$

where ω' is the makespan of the work-greedy assignment (either ETF or ERT), ω^i is the makespan of the optimal assignment *without* considering communication delays, and C_{comm} is the maximum communication delay along some chain in the task graph.

We note that the above bound can be improved in three ways:

1. When giving a guarantee for the makespan of a certain assignment, it is more useful to give a guarantee in terms of ω, the optimal makespan *not ignoring* the communication delay. That is, it is less useful to express ω' in terms of ω^i than to express it in terms of ω.
2. It is possible to generalize the bound so as to make it applicable to *all* work-greedy assignments.
3. Incorporating a degree of average software parallelism in the bound will highlight a symmetrical relationship between software parallelism and hardware parallelism.

We thus present in the following theorem a new generalized bound.

Let τ^* be the sum of execution times of tasks along the longest chain (ignoring communications) of the dependency graph and τ^+ be $\sum \tau_i$; and let $\pi = \tau^+/\tau^*$. Then we have

Theorem 1.

$$\frac{\omega'}{\omega} \leq 1 + \frac{(m-1)}{\pi} + m\,\frac{C_{comm}}{\tau^+} \qquad \textit{if } m < \pi$$

$$\frac{\omega'}{\omega} \leq 1 + \frac{(\pi-1)}{m} + \pi\,\frac{C_{comm}}{\tau^+} \qquad \textit{if } m \geq \pi$$

where ω is the length of the optimal makespan, that is not necessarily work-greedy; and ω' is the makespan of any arbitrary work-greedy assignment. C_{comm} is the maximum communication delay along some chain of tasks.

The proof of this theorem relies on a chain of tasks that we use to calculate C_{comm} and the sum of processor idle times. Before proceeding with the proof, therefore, we will look at an example illustrating how one would find such a chain. Refer to the Gantt chart shown in Figure 2.

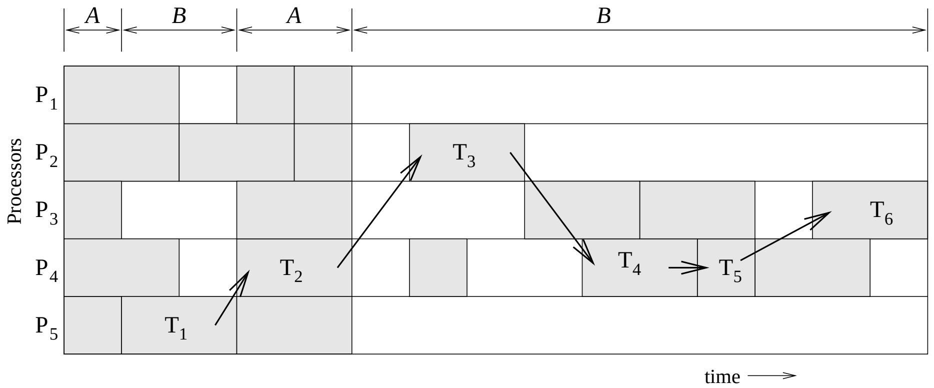

Figure 2: Gantt chart showing a task schedule.

The shaded areas in the figure denote executing tasks and the unshaded areas denote idling processors. In the regions marked A all processors are busy executing, and in the regions marked B, at least one processor is idle.

Task T_6 finishes execution at ω'. Task T_6 could not have started execution earlier because it is not ready until it receives input from task T_5. Task T_5 could not have started execution earlier because it is dependent on task T_4 and therefore cannot start before T_4 is finished and supplied input. Using similar arguments we find tasks T_3, T_2, and T_1. Task T_1 could not have started earlier because there is no free processor that could execute it earlier.

If $\prec$ be the partial order on $\mathbf{T}$, the tasks T_1 through to T_6 form a chain

$$T_1 \prec T_2 \prec \ldots \prec T_6$$

such that at every time instant $t \in B$ some task T_j in the chain is being executed or is waiting for input from T_{j-1} (that has finished executing) to start its execution.

C_{comm}, in this example, is the sum of communication times between T_1 and T_2, between T_2 and T_3, between T_3 and T_4, between T_4 and T_5, and between T_5 and T_6. The sum of all processor idle times, I, is given by

$$I \leq m\left(\sum_{i=1}^{6}\tau_i + C_{comm}\right) - \sum_{i=1}^{6}\tau_i$$

where m, the number of processors, is 5. The equality holds if no more than one processor is busy in the regions B.

Proof of Theorem 1.

For *any* (and thus, the optimal) assignment of makespan ω, the following inequality holds true:

$$\omega \geq \max\left[\frac{\tau^+}{m}, \tau^*\right] \tag{1}$$

Let $\prec$ be the partial order on $\mathbf{T}$. The rule of work-greedy assignments dictates that for any arbitrary work-greedy assignment of makespan ω', there exists a chain of tasks

$$T_{c,1} \prec T_{c,2} \prec \ldots \prec T_{c,y}$$

such that at every time instant $t \in B$ some $T_{c,j}$ is being executed or is waiting for input from $T_{c,j-1}$ (that has finished executing) to start its execution. Here B is the set of all points of time in $[0, \omega']$ for which at least one processor is idle.

Let $mtt(P_i, P_j)$ be the maximum time to transfer unit information from processor P_i to processor P_j (possibly via other processors). Recall that $M(T)$ is the processor to which task T is assigned. C_{comm} is calculated as follows:

$C_{comm} = \sum_{j=1}^{y-1} mtt(M(T_{c,j}),\ M(T_{c,j+1})) \times\ \ v(T_{c,j},\ T_{c,j+1})$

Let the sum of all the processor idle times in this assignment be I. Then,

$$I \leq m\left(\sum_{j=1}^{y}\tau_{c,j} + C_{comm}\right) - \sum_{j=1}^{y}\tau_{c,j} \tag{2}$$

But for any chain in an assignment, the following inequality holds true:

$$\sum_{j=1}^{y}\tau_{c,j} \leq \tau^* \tag{3}$$

Now since

$$\omega' = \frac{1}{m}\left[\tau^{+} + I\right],$$

from (2) and (3) we get,

$$\omega' \leq \frac{\tau^{+}}{m} + \frac{(m-1)\tau^{*}}{m} + C_{comm} \tag{4}$$

Note again that I includes all idle times of processors.

When $\tau^{+}/m \geq \tau^{*}$, from (1) and (4) we get the bound:

$$\frac{\omega'}{\omega} \leq 1 + \frac{m-1}{\pi} + m\,\frac{C_{comm}}{\tau^{+}} \tag{5}$$

When $\tau^{*} \geq \tau^{+}/m$, from (1) and (4) we get the bound:

$$\frac{\omega'}{\omega} \leq 1 + \frac{\pi-1}{m} + \pi\,\frac{C_{comm}}{\tau^{+}} \tag{6}$$

Both (5) and (6) always hold true. However, when $m \geq \pi$ the bound of (6) is tighter, otherwise the bound of (5) is tighter.

□

π is the degree of average software parallelism. It is a lower bound on the amount of parallelism within a task dependency graph.

Compare the bounds of Theorem 1 with that of Hwang et al.:

$$\frac{\omega'}{\omega^{i}} \leq \left(2 - \frac{1}{m}\right) + \frac{C_{comm}}{\omega^{i}} = \left(1 + \frac{m-1}{m}\right) + \frac{C_{comm}}{\omega^{i}} \leq \left(1 + \frac{m-1}{m}\right) + \frac{C_{comm}}{\max\left(\frac{\tau^{+}}{m}, \tau^{*}\right)}$$

The bounds of Theorem 1

- are applicable to all work-greedy assignments, not just ETF and ERT,
- express ω' in terms of the optimal makespan *not ignoring* the communication delay, and
- highlight the symmetrical relationship between m and π.

2.1 Construction of the chain

The set of all points in time in the interval $[0, \omega']$ is divided into two subsets A and B as follows. A is the set of points in time for which all processors are busy. B is the set of points in time for which at least one processor is idle.

Let ψ_i and ϕ_i denote respectively the start and finish times of T_i. The following algorithm constructs the chain. It is similar to the chain construction algorithm of Hwang et al. [Hwang et al., 1989]. The differences are the manner in which communication costs are computed and the notations used.

1. Let the chain C be an ordered set of tasks, set to null initially.
2. $T_a \leftarrow$ a task that finishes at time ω'.
3. If $\psi_a \in B$, then there exists a processor which for some $\epsilon > 0$ is idle during the time interval $[\psi_a - \epsilon, \psi_a]$. This occurs only when there is a task T_b, an immediate predecessor of T_a, such that $\phi_b + mtt(M(T_a),\ M(T_b))\ v(T_a,\ T_b)$ is equal to ψ_a. Insert T_a into C, $T_a \leftarrow T_b$ and go to 3.

4. Let $u =$ l.u.b.[2] $\{x|x < \psi_a$ and $x \in B\}$. If u is zero, output C and stop.
5. Find a task T_b such that $\psi_b = \max_i\{\psi_i|T_i$ a predecessor of T_a and $\psi_i < u\}$. There is a sequence of tasks, $T_c, T_{j1}, \ldots T_{jz}$, such that $T_b \prec T_c \prec T_{j1} \prec \ldots \prec T_{jz} \prec T_a$. Insert T_c into C, $T_a \leftarrow T_b$ and go to 3.

The maximum time to transfer information between processors depends as well on the underlying routing strategy and the network contention. These dependencies were ignored in the procedure above.

2.2 Bounds when communication cost can be ignored

By setting C_{comm} to zero in Theorem 1, we get the following bounds:

$$\frac{\omega'}{\omega} \leq 1 + \frac{(m-1)}{\pi} \qquad \text{if } m < \pi \tag{7}$$

$$\frac{\omega'}{\omega} \leq 1 + \frac{(\pi-1)}{m} \qquad \text{if } m \geq \pi \tag{8}$$

According to bounds (7) and (8), as $m \to \infty$, ω'/ω reaches unity (rather than 2 as Graham's bound suggests [Graham, 1976]). This highlights the fact that with unlimited processing resources, any work-greedy assignment is optimal. In practical terms, a work-greedy assignment is optimal if $m \geq n$.

Rearranging (1) and (4) and setting C_{comm} to zero lead to the following bound established by Sarkar [Sarkar, 1989], Theorem 4-4, page 60:

$$\max\left[\frac{\tau^+}{m}, \tau^*\right] \leq \omega \leq \frac{1}{m}\left[\tau^+ + (m-1)\tau^*\right].$$

2.3 A bound on the number of processors

The number of processors required to finish executing all the tasks in the minimum possible time is bounded below by the ratio of the total execution time requirement of the tasks and the minimum makespan [McNaughton, 1959]. The total execution time requirement is τ^+ and the minimum possible makespan is τ^*. A lower bound on the number of processors is thus given by

$$\left\lceil\frac{\tau^+}{\tau^*}\right\rceil = \lceil\pi\rceil$$

That is, *any* (not necessarily work-greedy) assignment will require at least $\lceil\pi\rceil$ processors, if it is to execute the task graph in the minimum possible time.

For tighter lower bounds on the number of processors, the reader is referred to [Al-Mouhamed, 1990].

2.4 A bound on speedup

The sequential execution time of a task graph is τ^+. The parallel execution time of the graph cannot be less than τ^*. The speedup from parallel execution, therefore, cannot be greater than $\frac{\tau^+}{\tau^*}$. The speedup cannot also be greater than m, the number of processors. Therefore, the maximum speedup one can obtain for a given task graph is $\min(m, \pi)$.

[2] Least upper bound

3 Implications of the Results

The hardware parallelism, m, and the degree of average software parallelism, π, have a symmetric relation in the bounds of Theorem 1. When $m > \pi$, the makespan may be limited by software 'sequentialism'; and when $\pi > m$ the makespan may be limited by hardware inadequacy.

Note that, since π is only a lower bound on software parallelism, we can find cases where $m \geq \pi$ and yet the makespan is limited by hardware inadequacy. For example, consider the task graph of Figure 3. Let $\tau_i = 1$ and $c_{0,i} = 0$ for all i. Thus $\tau^+ = 8$, $\tau^* = 2$, and $\pi = 4$. Let m, the number of processors, be 5. The makespan for the assignment of this task graph on these processors is 3. To reach the optimal makespan of 2, however, one needs to have 7 processors. We therefore see that the makespan here is limited by hardware inadequacy even though $m > \pi$.

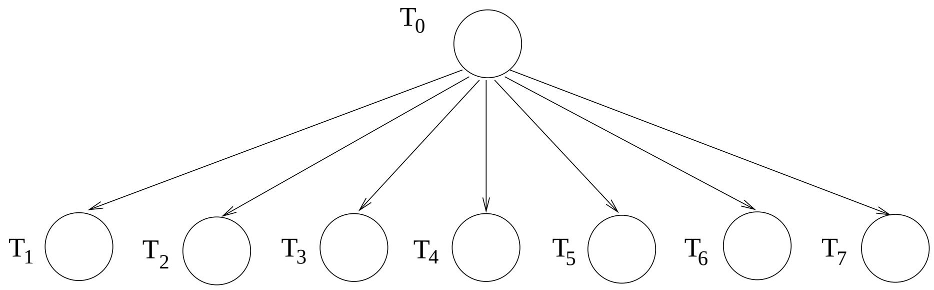

Figure 3: A task graph where makespan may be limited by hardware inadequacy.

It is known that, when communication costs can be ignored, *any* work-greedy assignment would be close to the optimal assignment by no more than a small constant factor. However, if communication costs are arbitrary, the performance can degrade considerably with bad assignment schemes. Consider the following loose bound derived from (5) and (6):

$$\frac{\omega'}{\omega} \leq 2 + \lambda$$

$$\text{where } \lambda = \min(m, \pi)\, \frac{C_{comm}}{\tau^+} = \frac{C_{comm}}{\tau^+ / \min(m, \pi)}.$$

λ signifies the communication to computation ratio along the critical path of the (arbitrary) assignment. Bad assignments will have large values of λ and thus they will have a poor performance compared to the optimal assignment. For instance, a work-greedy assignment scheme that ignores the communication costs when the dependency graph *does* have communication requirements may yield a large value of λ.

4 Summary and Conclusions

The heuristics most of the current assignment schemes use is based on satisfying the following rule of thumb: keeping the processors busy leads to a 'good' assignment. Such schemes are said to be work-greedy. Work-greedy assignments are important since most of them provide a solution with a guarantee: it is known that, when communication costs can be ignored, *any* work-greedy assignment would be close to the optimal assignment by no more than a small constant factor. It is proved that, should the communication costs be taken into account, this factor may no longer be small. That is, with communication costs, a work-greedy assignment can perform worse than the optimal assignment by a large factor that depends on the communication costs along some path in the task graph. Therefore, if an assignment problem dictates that it involves possible communication delays, then the heuristic assignment schemes must take these delays into account in order to produce good assignments.

References

[Al-Mouhamed, 1990] Al-Mouhamed, M. A. (December 1990). Lower bound on the number of processors and time for scheduling precedence graphs with communication costs. *IEEE Transactions on Software Engineering*, 16(12):1390–1401.

[El-Rewini et al., 1995] El-Rewini, H., Ali, H., and Lewis, T. (December 1995). Task scheduling in multiprocessing systems. *Computer*, 28(12):27–37.

[El-Rewini and Lewis, 1990] El-Rewini, H. and Lewis, T. (1990). Scheduling parallel program tasks onto arbitrary target machines. *Journal of Parallel and Distributed Computing*, 9:138–153.

[Gaudiot et al., 1988] Gaudiot, J. L., Pi, J. I., and Campbell, M. L. (1988). Program graph allocation in distributed multicomputers. *Parallel Computing*, 7:227–247.

[Gerasoulis et al., 1990] Gerasoulis, A., Venugopal, S., and Yang, T. (June 11-15, 1990). Clustering task graphs for message passing architectures. In *Proceedings of the International Conference on Supercomputing*, pages 447–456. ACM Press, Amsterdam, The Netherlands.

[Graham, 1976] Graham, R. L. (1976). Bounds on the performance of scheduling algorithms. In Coffman, E. G., editor, *Computer and Job Shop Scheduling Theory*, pages 165–227. John Wiley and Sons.

[Hoare, 1978] Hoare, C. A. R. (August 1978). Communicating sequential processes. *Communications of the ACM*, 21(8):666–677.

[Hwang et al., 1989] Hwang, J.-J., Chow, Y.-C., Anger, F., and Lee, C.-Y. (April 1989). Scheduling precedence graphs in systems with interprocessor communication times. *SIAM Journal of Computing*, 18(2):244–257.

[Lee et al., 1988] Lee, C.-Y., Hwang, J.-J., Chow, Y.-C., and Anger, F. (June 1988). Multiprocessor scheduling with interprocessor communication delays. *Operations Research Letters*, 7(3):141–145.

[Manoharan and Thanisch, 1991] Manoharan, S. and Thanisch, P. (April 1991). Assigning dependency graphs onto processor networks. *Parallel Computing*, 17(1):63–73.

[Manoharan and Topham, 1995] Manoharan, S. and Topham, N. (January 1995). An assessment of assignment schemes for dependency graphs. *Parallel Computing*, 21(1):85–107.

[McNaughton, 1959] McNaughton, R. (October 1959). Scheduling with deadlines and loss functions. *Management Science*, 6.

[Milner, 1989] Milner, R. (1989). *Communication and Concurrency*. Prentice Hall International.

[Rayward-Smith, 1987] Rayward-Smith, V. J. (1987). UET scheduling with unit interprocessor communication delays. *Discrete Applied Mathematics*, 18:55–71.

[Sarkar, 1989] Sarkar, V. (1989). *Partitioning and Scheduling Parallel Programs for Multiprocessors*. MIT Press, Cambridge MA.

[Shirazi et al., 1995] Shirazi, B., Chen, H.-B., and Marquis, J. (August 1995). Comparative study of task duplication static scheduling versus clustering and non-clustering techniques. *Concurrency: Practice and Experience*, 7(5):371–389.

[Ullman, 1976] Ullman, J. D. (1976). Complexity of sequencing problems. In Coffman, E. G., editor, *Computer and Job Shop Scheduling Theory*, pages 139–164. John Wiley and Sons.

[Wu and Gajski, 1990] Wu, M.-Y. and Gajski, D. (1990). Hypertool: A programming aid for meesage-passing systems. *IEEE Transactions on Parallel and Distributed Systems*, 1(3):330–343.