On the Complexity of Scheduling Problems for Parallel/Pipelined Machines

DAVID BERNSTEIN, MICHAEL RODEH, AND IZIDOR GERTNER, MEMBER, IEEE

Abstract—**We consider a machine model with *two* functional units, each can be either sequential or pipelined. The complexity of optimal scheduling for a set of expressions on such machines is investigated. We review some previous NP-completeness results, and present several new ones. For one restricted case, a polynomial time algorithm is described and analyzed.**

Index Terms—**NP-completeness, machines with several functional units, pipelined machines, resource constraints, scheduling algorithms.**

I. Introduction

CERTAIN recent computer architectures such as Cray-1 [17], VP-200, Cyber-205, FPS-164 [10], and others have dedicated functional units (e.g., adders, multipliers, etc.). To take advantage of such machines, programs written for them must be scheduled properly. These scheduling problems are somewhat similar to job-shop scheduling [6], [9] but in the latter, usually specific assumptions are made. Some approximation solutions for this sort of scheduling problem can be found in [12] and [7].

In this paper, we consider a machine model which has two functional units P_1 and P_2, each of which specializes in a disjoint subset of the set of instructions. Li [14] considered the problem of the optimal scheduling of expression trees for such machines. He proposed a *pseudopolynomial reduction* [8] of the knapsack problem to his scheduling problem. However, since the knapsack problem is NP-complete in the weak sense [18], this does not prove the NP-completeness of the scheduling problem. We give a correct proof which shows that the scheduling problem is NP-complete in the strong sense [8], and therefore NP-complete in the regular sense.

A special case of interest is one in which the precedence constraints imposed by expression forests on a program are such that the operations of P_1 (say, memory access unit) can precede the operations of P_2 (say, an arithmetic/logic unit), but not vice versa. The scheduling problem for expression forests with such ordered precedence constraints (OPC) has been considered in [2], where an efficient optimal algorithm is given. However, the problem of optimal scheduling for expressions represented by directed acyclic graphs with OPC was proved to be NP-complete in [16].

When scheduling expression forests on real machines, we are confronted with such issues as a bounded number of machine registers, shared access to the system bus, limited storage, etc. To study the effect of such limitations, resource constraints are introduced into the machine model. The presence of resource constraints usually causes scheduling problems to appear intractable [15], [4]. Indeed, optimal scheduling of expression forests with OPC is shown to be NP-complete even when there exists just one resource with one unit of supply. Another type of resource constraint was considered in [3], where the OPC problem was analyzed for a machine which has a bounded number of registers. However, in this work, the complexity of the problem has not been resolved, and only an approximation algorithm was developed.

Another natural extension to our model is to consider pipelined machines (e.g., [13], [10]). Pipelining poses a new sort of constraint on program execution. A pipelined processor can start a new instruction every machine cycle, although it takes more than one unit of time to complete some of the machine instructions. An alternative view of a pipelined processor is as a machine which executes every instruction J in one unit of time, but can use the result of J only after some time delay d.

The problem of scheduling expressions which have the form of a directed acyclic graph on a *single* pipelined processor was considered in [11], where it was shown that, if the maximal delay d due to pipelining is unbounded, the problem of finding an optimal schedule is NP-complete. For $d = 1$, an optimal efficient algorithm exists [1], [5]. For OPC, the simple case in which P_2 is pipelined with $d = 1$ while P_1 is nonpipelined does not help, and as we show in Section III, the problem remains NP-complete. On the other hand, if only P_1 is pipelined, then the algorithm described in [2] may be extended to provide an optimal solution.

II. The Scheduling Model

Our job system consists of

1) Two *processors* P_1 and P_2. P_i can execute only jobs of *type i*.

2) A set of *jobs* $J = \{J_1, \cdots, J_n\}$ where every job J_i has the following job properties:

a) Type $K_i \in \{1, 2\}$

b) Unitary execution times.

c) *Delay* $D_i \in \{0, d\}$ for some fixed integer d. (The

Manuscript received February 9, 1987; revised July 28, 1988.

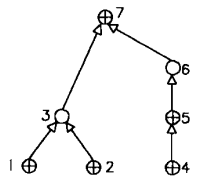
D. Bernstein is with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598.

M. Rodeh is with the IBM Israel Scientific Center, Technion City, Haifa 32000, Israel.

I. Gertner is with the Department of Electrical Engineering, Technion-Israel Institute of Technology, Haifa 32000, Israel.

IEEE Log Number 8929134.

0018-9340/89/0900-1308$01.00 © 1989 IEEE



job	D	R
1	0	0
2	1	1
3	0	0
4	1	0
5	0	0
6	1	1
7	-	0

Fig. 1. A job system for the scheduling problem.

delays model the pipelining structure of P_1 and P_2. For example, a memory operation may have one cycle access time, but another memory operation might have to be delayed to finish off the chips' cycle time.)

d) *Resource requirement* $R_i \in \{0, 1\}$.

3) A directed graph $G = (J, E)$ of *precedence constraints*.

A *legal* schedule is a mapping S from J into the set N of positive integers (interpreted as *time slots*) such that for all $i \neq j$:

1) If $(J_i, J_j) \in E$ then $S(J_j) - S(J_i) > D_i$.

2) If $S(J_i) = S(J_j)$ then $K_i \neq K_j$.

3) If $S(J_i) = S(J_j)$ then $R_i + R_j \leq 1$.

The first requirement ensures that the precedence constraints and the delay times are honored while taking into account the unitary execution times; the second requirement ensures that at most one processor of each type is active in every time slot; the third requirement ensures that no more than one unit of the resource is consumed per one unit of time.

An example of a job system is shown in Fig. 1. In this figure and throughout the paper, jobs of types 1 and 2 are designated by crossed and clear circles, respectively. The circles are connected by edges which describe the precedence constraints. In many cases, a *rooted tree* is obtained—a tree with one vertex designated as being the root. We also allow the precedence graph to be a *forest*, i.e., a collection of rooted trees. Three legal schedules for the job system of Fig. 1 appear in Fig. 2. For each schedule, a number k in row i and column j means that job J_k is computed at time slot j by P_i. In S_1, J_7 cannot be scheduled at time slot 7 as $D_6 = 1$. Also notice that in S_2, J_6 cannot be scheduled at time slot 4 due to the resource constraints ($R_2 + R_6 = 2$).

The *completion (maximum finishing) time* $c(S)$ of a schedule S is defined by $\max_i S(J_i)$. For example, in Fig. 2, $c(S_1) = 8$, $c(S_2) = 7$, and $c(S_3) = 6$. We will be interested in minimizing the completion time. A formulation which is easier to analyze is the corresponding *decision problem*, namely, given a number w, does there exist a schedule whose completion time is less than or equal to w. We shall refer to this problem by SW. In Section III, certain negative results are derived, while in Section IV, a polynomial algorithm for a special case of SW is presented.

III. NP-Completeness Results

To derive NP-completeness results, we reduce the 3-PARTITION problem to the (decision problem which corresponds to the) scheduling problem at hand by means of pseudopolynomial reductions. Since 3-PARTITION is NP-complete (in the *strong sense* [8]), so are the scheduling

	N	1	2	3	4	5	6	7	8
S_1	P_1	1	2	4		5			7
	P_2				3		6		
S_2	P_1	4	1	5	2			7	
	P_2					6	3		
S_3	P_1	4	2	5	1		7		
	P_2				6	3			

Fig. 2. Three legal schedules for the job system of Fig. 1.

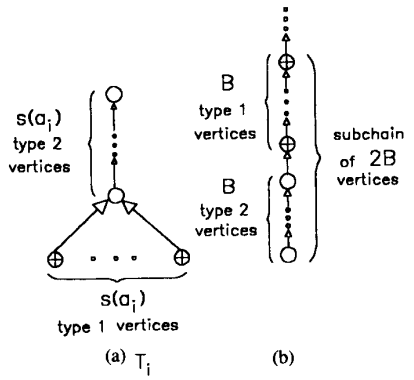


Fig. 3. The job system for the PC problem.

problems. The 3-PARTITION problem is formulated as follows.

Instance: Set A of $3m$ elements, a bound $B \in N$, and a size $s(a) \in N$ for each $a \in A$ such that $B/4 < s(a) < B/2$ and $\Sigma_{a \in A} s(a) = mB$.

Question: Can A be partitioned into m disjoint sets $A_1, A_2, \cdots, A_m$ such that, for $1 \leq i \leq m$, $\Sigma_{a \in A_i} s(a) = B$?

We consider several special cases of SW and analyze their complexity.

A. The Precedence Constraints Problem

The *precedence constraints problem* (PC) is a special case of SW in which all the delays D_i and all the resource requirements R_i are zero.

Instance: Given two dedicated processors P_1 and P_2, a job system, a forest of precedence constraints, and an integer w.

Question: Does there exist a schedule of completion time w or less?

To prove that PC is NP-complete in the strong sense (and therefore NP-complete in the regular sense), we assume an arbitrary instance of 3-PARTITION, and map it to an instance of PC.

1) For each element $a_i \in A$, $1 \leq i \leq 3m$, we build a tree T_i which consists of $s(a_i)$ independent type 1 jobs connected to a chain of $s(a_i)$ type 2 jobs as in Fig. 3(a).

2) We build one chain which consists of m subchains as in Fig. 3(b); each subchain has $2B$ jobs—the first B jobs are of type 2 and the other B jobs are of type 1.

The question we ask is: Does there exist a schedule for this instance of completion time $2mB$?

The following lemma will be used in all our NP-completeness proofs.

Lemma 1: Consider the trees T_i of Fig. 3(a). Overall, there

are $2mB = 2\Sigma_{a \in A} s(a)$ jobs that have to be scheduled. Let us ignore the jobs of Fig. 3(b) and concentrate on the initial part of the schedule which starts at time slot 1 and ends when $2B$ of the jobs of Fig. 3(a) have been scheduled. Moreover, assume that these first $2B$ jobs consist of B type 1 and B type 2 jobs. Then, these $2B$ jobs belong to three trees T_i, T_j, and T_k such that $s(a_i) + s(a_j) + s(a_k) = B$.

Proof: Let c be the number of trees whose entire set of leaves has been completed by some of the given B type 1 jobs. For all i, $B/4 < s(a_i) < B/2$. Since there are exactly B type 1 jobs, c cannot exceed 3. We claim that c must be equal to 3, since otherwise, there were less than B internal vertices available for P_2. This shows that the given $2B$ jobs finish executing exactly three trees, proving the lemma. ■

Theorem 2: PC is NP-complete in the strong sense.

Proof: It is obvious that PC is in NP. To show NP-completeness in the strong sense, we prove that there exists a solution to the given instance of 3-PARTITION if and only if there is a solution to the instance of PC generated by the above transformation.

Assume that there is a solution to the PC problem. Such a solution must keep both of P_1 and P_2 busy all the time, since over all there are $4mB$ jobs and they must be completed in $2mB$ time slots. Now, consider the first $2B$ time slots. During the first B time slots, P_2 is busy with the first B type 2 jobs of the long chain [Fig. 3(b)], while during the second B time slots, P_1 executes exactly B type 1 jobs from that chain. Therefore, during the first $2B$ time slots, exactly B type 1 jobs and B type 2 jobs of the trees of Fig. 3(a) are completed. By Lemma 1, there are three trees T_i, T_j, and T_k such that $s(a_i) + s(a_j) + s(a_k) = B$. Define $A_1 = \{a_i, a_j, a_k\}$. We can repeat this argument to build $A_2, \cdots, A_m$ which solves the 3-PARTITION problem.

The other direction, namely, given a solution to 3-PARTITION, show that the corresponding PC problem is solvable is even easier. Assume that we are given a solution to the 3-PARTITION problem which specifies a partition of the given $3m$ numbers into m triples such that the sum of the numbers in each triple is exactly B. Then to build a solution to the PC problem, in the first $2B$ time slots, we schedule the trees of Fig. 3(a) that correspond to the first triple, in the next $2B$ time slots the trees of Fig. 3(a) which correspond to the second triple are scheduled, and so on. ■

B. The Ordered Precedence Constraints Problem

Consider an SW problem in which jobs of type 2 cannot precede jobs of type 1, and refer to it as the *ordered precedence constraints problem* (OPC). The case in which the delay times D_i and the resource requirements R_i are all zero was discussed in [2], and an efficient optimal algorithm given. We are interested in the impact of delay times and resource constraints on the complexity of OPC.

1) OPC with Nonzero Resource Requirements: Let OPCR be the OPC problem in which nonzero resource requirements have been introduced. We claim that OPCR is NP-complete, and prove this result by reducing 3-PARTITION to OPCR. Given an instance of 3-PARTITION, we build an instance of OPCR as follows.

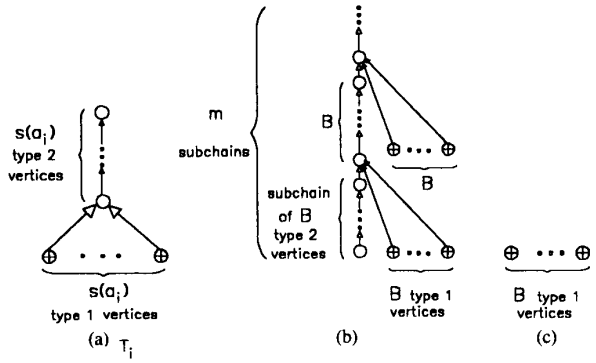


Fig. 4. The job system for the OPCR problem.

1) For each element $a_i \in A$, $1 \le i \le 3m$, we build a tree T_i as we did for the PC problem [see Fig. 4(a)]. The resource requirements of the jobs in these trees are all zero.

2) We build one long chain consisting of mB jobs of type 2. For every $1 \le i \le m - 1$, B independent type 1 jobs are connected to the long chain in such a way that all of them must be computed before the iB + 1st job of type 2 from the long chain [see Fig. 4(b)]. The resource requirements of these jobs are all one.

3) B isolated type 1 jobs are added [see Fig. 4(c)]. The resource requirements of these jobs are all one.

The question we ask is: Does there exist a schedule of completion time $2mB$?

Theorem 3: OPCR is NP-complete in the strong sense.

Proof: As in the PC problem, the only nontrivial part of the proof is to show that there exists a solution to the given instance of 3-PARTITION if and only if there is a solution to the instance of OPCR generated by the above transformation.

Assume that there is a solution to the OPCR problem. Consider the first $2B$ time slots and assume that P_i spends t_i units of time working on some of the trees [Fig. 4(a)]. Since there are $2mB$ jobs of each type, and since the bound on the completion time of the schedule is also $2mB$, each of the two processors must be busy all the time. Therefore, during the first $2B$ time slots, P_i spends $2B - t_i$ time units on the jobs of Fig. 4(b) or (c). Moreover, the jobs of Fig. 4(b) and (c) have resource requirements of one unit each. Therefore, at no time can both processors work on jobs in this group. Since the number of such jobs is $2mB$, at every time slot one of them must be executed. We see that $2B - t_1 + 2B - t_2 = 2B$, and therefore, $t_1 + t_2 = 2B$. By the structure of the trees T_i, $t_2 \le t_1$, because type 1 jobs are executed first. We claim that $t_2 \ge B$ which will prove that $t_1 = t_2 = B$. If $t_2 < B$ then $2B - t_2 > B$. P_2 can execute more than B jobs of Fig. 4(b) only if P_1 executes at least B type 1 jobs there. Therefore, if this happens then $2B - t_1 \ge B$ or $t_1 \le B$. Combined with $t_2 < B$ this will lead to $t_1 + t_2 < 2B$ contradicting $t_1 + t_2 = 2B$. Therefore, during the first $2B$ time slots, exactly B type 1 jobs and B type 2 jobs from the trees of Fig. 4(a) are completed. Now we can apply Lemma 1 to complete the proof as we did in Theorem 1.

The other direction, namely given a solution of 3-PARTITION, show that the corresponding PC problem is solvable is proved as in Theorem 2. ■

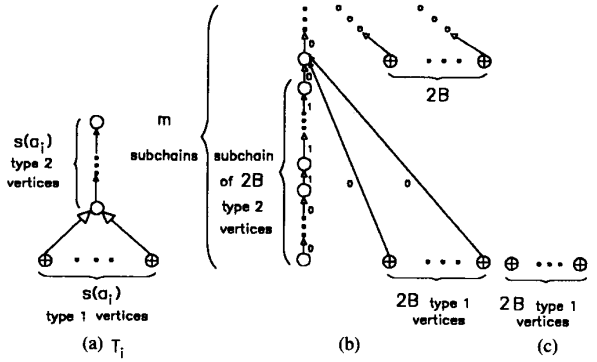


Fig. 5. The job system for the OPCD problem.

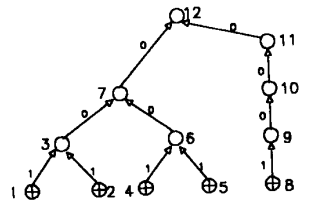


Fig. 6. A job system for the OPCD1 problem.

2) OPC with Delays: Let OPCD be the OPC problem in which delay times are allowed (but all resource requirements are zero). We claim that even for $d = 1$, OPCD is NP-complete. Again we use 3-PARTITION. Given an instance of 3-PARTITION, we build an instance of OPCD as follows.

1) For each element $a_i \in A$, $1 \le i \le 3m$, we build a tree T_i as we did for the PC problem [see Fig. 5(a)]. The delay times of all the jobs in these trees are set to zero.

2) We build one long chain consisting of $2mB$ jobs of type 2. For every $1 \le i \le m - 1$, $2B$ independent type 1 jobs are connected to the long chain in such a way that all of them must be computed before the $2iB + 1$st type 2 job from the long chain. For every $1 \le i \le m$, the delays of jobs $2iB - B, \cdots, 2iB - 1$ are one time unit; the delays of the other jobs are all zero [see Fig. 5(b)].

3) $2B$ isolated type 1 jobs are added [see Fig. 5(c)].

The question we ask is: Does there exist a schedule of completion time $3mB$?

Theorem 4: OPCD is NP-complete in the strong sense.

Proof: As in the previous proofs, the only nontrivial part is to show that there exists a solution to the given instance of 3-PARTITION if and only if there is a solution to the instance of OPCD generated by the above transformation.

Assume that there is a solution to the OPCD problem. Since the length of the long chain from Fig. 5(b) is $3mB$ (remember that there are mB one-unit delays), none of the long chain jobs may be delayed. The earliest time the $2B + 1$st job of the long chain can be computed is $3B + 1$. Consider the first $3B$ time slots. During this period, at least $2B$ type 1 jobs and exactly $2B$ type 2 jobs from Fig. 5(b) or (c) must be completed. Therefore, in the first $3B$ time slots, exactly B type 2 jobs of Fig. 5(a) must be computed. The number of computed type 1 jobs of Fig. 5(a) cannot be smaller than the number of computed type 2 jobs of Fig. 5(a). Therefore, during the first $3B$ time slots the number of computed type 1 jobs of Fig. 5(a) is also exactly B. Now we can apply Lemma 1 to complete the proof.

The other direction, namely, given a solution to 3-PARTITION, show that the corresponding PC problem is solvable is proved in a way similar to Theorem 2. ■

IV. The Polynomial Time Algorithm

In this section, we present a polynomial algorithm for a special case of OPCD called OPCD1, in which

1) all type 1 jobs are the leaves of the precedence tree, and their delays are equal to d;

2) all type 2 jobs have zero delays.

An example of an OPCD1 problem in which $d = 1$ is given in Fig. 6. Three legal schedules for this problem are shown in Fig. 7.

An algorithm to solve OPC problems when delay times and resource constraints are all zero has been presented in [2]. This algorithm can be modified to optimally solve OPCD1 problems.

A schedule S is *compressed* if the following conditions are satisfied.

1) All jobs of type 1 are consecutive starting at time slot 1.

2) All jobs of type 2 are consecutive at the end of the schedule.

3) The schedule S' obtained by setting $S(J_i) = S(J_i) - 1$, for all type 2 jobs J_i is illegal.

For example, in Fig. 7, S_2 and S_3 are compressed schedules while S_1 is not since type 2 jobs in S_1 are not consecutive. S_2 is in fact a compressed version of S_1 obtained by a procedure outlined in the following lemma.

Lemma 5: For every legal schedule S there exists a compressed schedule S' such that $c(S') \le c(S)$.

Proof: Given S, we compress it in three steps.

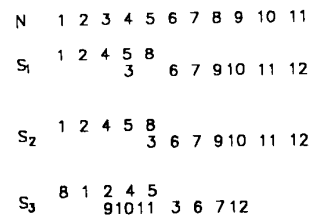


Fig. 7. Three legal schedules for the job system of Fig. 6.

1) First we shift all type 1 jobs to the left until they become consecutive starting at time slot 1. This is always possible, since there are no delay constraints among type 1 jobs, being the leaves of the precedence tree. The completion time of the schedule does not increase as result of this operation.

2) Type 2 jobs are shifted to the right, without changing the position of the latest type 2 job. This transformation is always possible since there are no delay times among type 2 jobs. Again, the completion time of the schedule does not increase.

3) Finally, we shift all type 2 jobs to the left as much as we can, without violating the precedence and delay constraints. This step may only decrease the cost of the schedule. ■

In the sequel, we assume that all the schedules of OPC and OPCD1 are compressed. Let I be an instance of OPC. We say that an instance I' of OPCD1 *corresponds* to I if they differ only by the appearance of delay times in I'. We would like to show that an optimal schedule of an instance of OPC can be modified to be an optimal schedule of the corresponding OPCD1 problem. First, let us assume that all the leaves of the precedence tree are of type 1, as in Fig. 6.

Theorem 6: Let I be an instance of OPC, and let S be an optimal schedule of I. Let S' be a schedule which is obtained from S by shifting all type 2 jobs to the right by d time units. Then S' is an optimal schedule of the OPCD1 problem I' which corresponds to I.

Proof: First, notice that since S is a legal schedule of I, S' is also a legal schedule of I' and $c(S') = c(S) + d$. Assume, by contradiction, that there exists an optimal schedule $\bar{S}'$ of I' such that $c(\bar{S}') < c(S')$. Let $\bar{S}$ be a schedule of I which is obtained from $\bar{S}'$ by shifting all type 2 jobs to the left by d time units. Since $\bar{S}'$ is legal, for every pair of jobs J_i and J_j such that J_i is of type 1, J_j is of type 2, and $(J_i, J_j) \in E$, $S(J_j) - S(J_i) \geq d + 1$. Also, since all the leaves of the precedence tree are type 1 jobs, the leftmost job of type 2 can be computed in $\bar{S}'$ no earlier than at time slot $d + 2$. Therefore, $\bar{S}$ is a legal schedule of I, and $c(\bar{S}) = c(\bar{S}') - d$. Since $c(\bar{S}') < c(S')$, we get that $c(\bar{S}) < c(S)$, which contradicts the assumption that S is optimal. ■

Now consider the case in which some of the leaves of the precedence tree T are of type 2. By iteratively deleting all type 2 leaves (and the internal vertices that become leaves as a result of deleting their children), we get a new precedence tree T' with no type 2 leaves. Then, we apply the above algorithm to T' deriving an optimal schedule S for it. Now we schedule the deleted type 2 jobs to the left of the type 2 jobs of T' (taking into account the precedence constraints among them). We get a schedule S' such that either $c(S') = c(S)$ or $c(S')$ is equal to the number of type 2 jobs in T. In both cases, S' is optimal.

The optimal algorithm for OPC described in [2] can be extended to a case in which internal vertices may be of type 1 provided that the ordered precedence constraints of Section III are preserved. This case may be extended to an OPCD1 problem by allowing nonzero delay times only when the parent of a type 1 job is a type 2 job.

V. Conclusions

In this paper, we considered the problem of optimal scheduling of a job system for two dedicated processors. It was proved that this problem (PC) is NP-complete even when the precedence graph is a forest. An interesting special case of the PC problem (OPC) can be solved by an efficient optimal algorithm [2]. In this special case, it is assumed that the machine architecture allows one memory access and one arithmetic operation to take place concurrently. When the number of registers is sufficiently large, we can model this problem as a scheduling problem in which type 1 jobs (memory access) are done at the leaves while type 2 jobs (arithmetic operations) are done at the internal vertices of a computational tree. Unfortunately, by introducing resource constraints (OPCR) or delay times (OPCD) into OPC, NP-complete problems are obtained. One special case (OPCD1) of OPCD is polynomially solvable.

Acknowledgment

The idea leading to the result of Theorem 2 was pointed out by D. Johnson. Also, we want to thank referees for helpful suggestions which improved the presentation of the paper.

References

[1] D. Bernstein and I. Gertner, "Computing expressions on a pipelined processor with a small number of stages," EE PUB No. 594, Dep. Elec. Eng., Technion, Haifa, Israel, June 1986.

[2] D. Bernstein, J. M. Jaffe, R. Y. Pinter, and M. Rodeh, "Optimal scheduling of arithmetic operations in parallel with memory access," TR 88.136, IBM-Israel Scientif. Cent., May 1985. A preliminary version appeared in *Proc. Twelfth ACM Symp. Principles Programming Languages,* Jan. 1985, pp. 325-333.

[3] D. Bernstein, J. M. Jaffe, and M. Rodeh, "Scheduling arithmetic and load operations in parallel with no spilling," in *Proc. Fourteenth ACM Symp. Principles Programming Languages,* Jan. 1987, pp. 263-273.

[4] J. Blazewicz, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Scheduling subject to resource constraints: Classification and complexity," *Discrete Appl. Math.*, vol. 5, pp. 11-24, 1983.

[5] J. Bruno, J. W. Jones, and K. So, "Deterministic scheduling with pipelined processors," *IEEE Trans. Comput.*, vol. C-29, no. 4, pp. 308-316, Apr. 1980.

[6] E. G. Coffman, *Computer and Job-Shop Scheduling Theory.* New York: Wiley, 1976.

[7] D. K. Goyal, "Scheduling processor bound systems," in *Proc. Sixth Texas Conf. Comput. Syst.*, 1977.

[8] M. R. Garey and D. S. Johnson, *Computers and Intractability, A*

Guide to the Theory of NP-Completeness. San Francisco, CA: Freeman, 1979.

[9] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnoy Kan, "Optimization and approximation in deterministic sequencing and scheduling: A survey," *Discrete Optimization,* 1977.

[10] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing.* New York: McGraw-Hill, 1984.

[11] J. L. Hennessy and T. R. Gross, "Postpass code optimization of pipeline constraints," *ACM Trans. Programming Languages Syst.,* vol. 5, no. 3, pp. 442–448, July 1983.

[12] J. M. Jaffe, "Bounds on the scheduling of typed task systems," *SIAM J. Comput.,* vol. 9, no. 3, pp. 541–551, Aug. 1980.

[13] P. M. Kogge, *The Architecture of Pipelined Computers.* New York: McGraw-Hill, July 1983.

[14] H. F. Li, "Scheduling trees parallel/pipelined processing environments," *IEEE Trans. Comput.,* vol. C-26, no. 11 pp. 1101–1112, Nov. 1977.

[15] E. L. Lloyd, "Scheduling task systems with resources," Ph.D dissertation, MIT-LCS-TR-236, May 1980.

[16] C. L. Monma, "Sequencing to minimize the maximum job cost," *Oper. Res.,* vol. 28, no. 4, pp. 942–951, July 1980.

[17] R. M. Russell, "The Cray-1 computer system," *Commun. ACM,* vol. 21, no. 1, pp. 63–72, Jan. 1978.

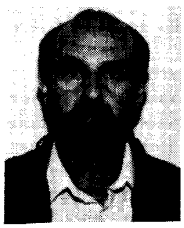
David Bernstein received the B.Sc. and Ph.D. degrees in electrical engineering from the Technion–Israel Institute of Technology, Haifa, in 1979 and 1987, respectively.

Currently he is a Postdoctoral Research Fellow at the IBM Thomas J. Watson Research Center. His research interests include programming language design, code generation algorithms, scheduling algorithms, and parallel processing.

Dr. Bernstein is a member of the Association for Computing Machinery.

Michael Rodeh was born in Israel in 1949. He received the B.Sc. degree in mathematics and physics (with distinction) from the Hebrew University, Jerusalem, Israel, in 1969, the M.Sc. degree in computer science from the Weizmann Institute of Science, Rehovot, Israel, in 1971, and the D.Sc. degree in computer science from the Technion–Israel Institute of Technology, Haifa, in 1976.

He joined IBM Israel Scientific Center in 1976. Currently, he is managing the Software Systems and Algorithms Department at the Haifa Research Group, IBM Israel. He also holds an Adjunct Associate Professor position at the Technion.



Izidor Gertner (S'76–M'78) was born in Kaunas, U.S.S.R., on June 4, 1946. He received the Radio Engineer diploma (M.S.E.E.) from Kaunas Politechnical Institute, U.S.S.R., in 1968 and the Ph.D. degree in electrical engineering from the Technion–Israel Institute of Technology, Haifa, in 1976.

He was a Junior Research Fellow at the Institute of Physical-Technical Problems of Energetics, Academy of Sciences, Lithuania, U.S.S.R., from 1968 to 1971. While at the Technion, he was a Teaching and Research Assistant with the Department of Electrical Engineering from 1972 to 1976 and a Lecturer with the faculty from 1976 to 1978. During 1978–1983 he was with RAFAEL as a Senior Research Engineer. Since 1983 he has been with the Electrical Engineering Faculty at the Technion as a Senior Lecturer. His main research interests are fast algorithms for processing multidimensional signals, as well as VLSI architectures for their implementation.