Selection and Assignment of Machines: a Parallel Approach

José Francisco Ferreira Ribeiro
Department of Computer Science, University of São Paulo at São Carlos, Brazil

In this paper, a two-phase method is presented for selection of machines to be kept on the shop floor and assignment of parts to be manufactured on these machines. In the first phase, dynamic programming or a heuristic procedure identifies a set of feasible solutions to a knapsack problem. In the second phase, implicit enumeration technique or a greedy algorithm solves an assignment problem. The proposed method is written in language C and runs on a parallel virtual machine called PVM-W95. The results obtained from the parallel implementation on several examples which are found in the literature, as well as examples generated at random, were used to establish a comparison with the sequential algorithm and to perform a speedup analysis.

Keywords: job shop production, assignment problem, optimization techniques, parallel virtual machine.

1. Introduction

This article proposes a parallel implementation of a two-phase method including optimization algorithms for solving (1) the problem of the selection of machines to process jobs on parts in a job shop production system and (2) the problem of the assignment of parts to be manufactured in the shop to the machines chosen in phase (1). These two problems belong to the class of NP-complete combinatorial problems [1]. Polynomial algorithms are available for reduced size instances only; however, these reduced size instances can be optimally solved by complete search. This approach cannot be used for solving large-scale instances and the use of heuristic algorithms [2] for solving large-scale instances becomes a widely employed rule.

The method implemented here desires to use available resources in the best manner. For this purpose, it attempts to find the best matching between parts and a subset of available machines in the shop. The machines are selected among those available and parts are assigned to these machines, leading to the construction of an incidence matrix $[\textit{parts} \times \textit{machines}]$. This problem is solved in two steps: searching feasible solutions to the knapsack optimization problem for the choice of the set of machines, followed by an implicit enumeration procedure or a greedy algorithm for the assignment of parts to the machines chosen, leading to a solution which is either optimal or feasible [2]. Parameters such as duration of operations and machine capacity are taken into account.

Parallel implementation of the proposed method is performed in this work with the objective to solve simultaneously all optimization problems on the types of machines available in the shop. In fact, for the types of machines available in the shop, such as lathe, miller, drill, etc., a problem $X[i]$ completely independent of the problem $X[j]$, where i and j are two different types of machines, must be solved. In terms of computational time, the gain is considerable: for solving m problems related to m types of machines, the computational time is approximate to the time spent to solve the biggest of these problems, if m nodes of processing are available.

The proposed two-phase method was written in language C and was executed on the Parallel Virtual Machine for Windows95, called PVM-95 [3]. The PVM-W95 is a software system that permits a PC-compatible computer network to be used as a single large parallel computer.

2. Problem Statement

The problem consists in finding an assignment of the parts to the machines available in the shop. Consider a set of r parts and a set of m machine types:

(1) For each part i ($i = 1$ to r):

- $nb[i]$ is the number of units to be manufactured.
- $g[i]$ is the number of operations of the production sequence.
- $type[i, k], k = 1$ to $g[i]$ is the type of machine used for the k^{th} operation on part i.
- Note that any type of machine may appear many times in the production sequence of part i.
- $duration[i, k], k = 1$ to $s[i]$, is the duration of the k^{th} operation on part i, given with respect to the reference machine of the corresponding type.

(2) For each type of machine j ($j = 1$ to m):

- $t[j]$ is the number of machines available.
- $sr[j, s]$, $s = 1$ to $t[j]$, is the speed ratio of the s^{th} machine of type j, given with respect to the reference machine.
- Speed ratio of the reference machine equals 1.
- $cap[j, s] = sr[j, s] \times period$ is the capacity of the s^{th} machine of type j, $s = 1$ to m.
- Reference machine capacity is the lowest in the type.
- Period is the total duration of work in the shop.
- $cost[j, s]$, $s = 1$ to $t[j]$, is the utilization cost of the s^{th} machine of type j.
- $p[j, s]$, $s = 1$ to $t[j]$, is the penalty weight for the s^{th} machine use.

Tables I and II show an example where $nb[i] = 1$ ($i = 1$ to r). This example will be used throughout this paper to illustrate the proposed method.

Part	Production Sequence						Duration of the operation					
1	2	4	5	6			1,0	2,0	3,0	3,0		
2	2	3	5	6	7	5	1,0	4,0	1,0	2,0	3,0	2,0
3	3	6	7	6			1,0	1,0	3,0	1,0		
4	1	2	3	1	4	5	1,0	3,0	2,0	1,0	1,0	2,0
5	1	3	4	5	4		3,0	1,0	2,0	1,0	2,0	
6	2	3	4				4,0	1,0	2,0			
7	1	2	4	5	7	4	1,0	2,0	1,0	3,0	1,0	1,0
8	1	2	3	5	7		2,0	2,0	3,0	1,0	3,0	
9	1	2	3	5	7	3	2,0	2,0	3,0	2,0	1,0	2,0

Table I. Production sequences and duration.

Type of Machine	Number of Machines	Speed Ratio			
		N1	N2	N3	N4
1	2	1,0	1,2		
2	3	1,0	1,3	1,2	
3	4	1,0	1,2	1,2	1,1
4	3	1,0	1,1	1,0	
5	2	1,0	1,1		
6	2	1,2	1,0		
7	3	1,0	1,1	1,1	

Table II. Machines and speed ratios.

3. Load Matrix [Parts × Types of Machines]

Given the production sequences of the parts to be manufactured, the duration of operations and the number of units to be manufactured for each part, the elements of the load matrix $[parts \times machines]$ are given by:

$$load[i,j] = nb[i] \times \sum_{k \mid type[i,k]=j} duration[i,k] \quad (1)$$

In the example proposed the load matrix is given in Table III.

	T 1	T 2	T 3	T 4	T 5	T 6	T 7
Part 1	0,0	1,0	0,0	2,0	3,0	3,0	0,0
Part 2	0,0	1,0	4,0	0,0	3,0	2,0	3,0
Part 3	0,0	0,0	1,0	0,0	0,0	2,0	3,0
Part 4	2,0	3,0	2,0	1,0	2,0	0,0	0,0
Part 5	3,0	0,0	1,0	4,0	1,0	0,0	0,0
Part 6	0,0	4,0	1,0	2,0	0,0	0,0	0,0
Part 7	1,0	2,0	0,0	2,0	3,0	0,0	1,0
Part 8	2,0	2,0	3,0	0,0	1,0	0,0	3,0
Part 9	2,0	2,0	5,0	0,0	2,0	0,0	1,0

Table III. Load matrix $[parts \times types\ of\ machines]$.

4. Selection of Machines

First, the machines which will remain effective in the workshop are chosen, irrespective of the process of assignment of parts to machines, but only in accordance with the global load accepted by each type of machine (see Table III).

The choice of machines is done separately for each type of machine j. This choice is made for feasible solutions to the knapsack problem (KNS) detailed below.

- KNS

$$\text{Minimize} \quad \sum_{s=1}^{t[j]} c[j,s] \times m_s \quad (2)$$

subject to

$$\begin{cases} \displaystyle\sum_{s=1}^{t[j]} cap[j,s] \times m_s \geq \sum_{i=1}^{r} load[i,j] & (3) \\ m_s = \begin{cases} 1 & \text{if machine s is chosen} \\ 0 & \text{otherwise} \end{cases} & (4) \end{cases}$$

where $c[j,s]$ is an optimization criterion, for example:

- number of machines in the workshop

$$c[j,s] = 1 \quad (j = 1 \text{ to } t[j]) \quad (5)$$

- total capacity of machines in the workshop

$$c[j,s] = cap[j,s](j = 1 \text{ to } t[j]) \quad (6)$$

- total utilization cost of machines in the workshop

$$c[j,s] = cost[j,s](j = 1 \text{ to } t[j]) \quad (7)$$

Objective-function of KNS minimizes the selected optimization criterion. Constraint (3) insures that the workshop machine capacity is sufficient to carry out the global part workload and the constraint (4) yields the possible values of m_s.

The search for a solution is performed by subjecting possible combinations of machines to the principal constraint using dynamic programming or heuristic enumeration and by listing feasible combinations according to their decreasing objective-function value. Some of the combinations found feasible for this problem are set aside *a posteriori* verification of the assignment of parts.

Dynamical programming generates all the possible combinations of machines and is used for small size problems (less than 100 variables). This technique is based on the Belman's optimality principle: an optimal trajectory is composed of optimal parts [2]. Then all the optimal parts are generated and the best trajectories are recovered from the list with best solutions. A heuristic procedure is used when the dimension of the problem requires very large calculating time. It uses the branch and bound technique [2], and stops the execution after a limited time

(5 minutes) or after reaching a determined number of feasible solutions (10 solutions).

Once the machines are chosen, the assignment issue is of a size less than, or equal to that of the assignment problem to be solved when all the initially available machines are taken into account. The existence of a feasible assignment of parts to the machines chosen must subsequently be verified.

5. Assignment of Parts to Machines

The assignment of parts to machines from the combination proposed by KNS is obtained through the solution of the problem PM01^j stated bellow.

- PM01^j

$$\text{Minimize} \quad \sum_{i=1}^{r} \sum_{s=1}^{t[j]} p[j,s] \times x_{is} \qquad (8)$$

subject to

$$\begin{cases} \sum_{i=1}^{r} load[i,j] \times x_{is} \leq cap[j,s] \; (s = 1 \text{ to } t[j]) & (9) \\ \sum_{s=1}^{t[j]} x_{is} = 1 \; (i = 1 \text{ to } r) & (10) \\ x_{is} = \begin{cases} 1 & \text{if part } i \text{ is assigned to machine } s \\ 0 & \text{otherwise} \end{cases} & (11) \end{cases}$$

where $p[j,s]$ is a penalty weight for the s^{th} machine use. Such weights can be introduced in the objective function of PM01 to direct assignment of parts to some machines (e.g. the fastest or least expensive ones) at the operator's choice. Constraint (9) ensures that each machine is able to produce the parts it has been attributed with; constraint (10) establishes indivisible nature of the operations that have to be performed on the parts; constraint (11) yields possible values of x_{is}.

Solution of PM01 is obtained by implicit enumeration [2] if the number of variables is less than approximately fifty. Otherwise, a greedy algorithm [2] is used to obtain the assignment. In the case where the assignment is not valid for the first KNS solution, the second is tried, and so on, until a successful assignment is achieved.

Implicit enumeration was proposed by Balas [2] for the resolution of integer problems with variables 0/1 in a faster manner. Greedy algorithm [2] allows to solve an integer programming problem very quickly, because it does not re-evaluate the decisions taken at each step. The greedy procedure consists of taking the best decision at each step and making this decision final, obtaining, then, solutions very quickly, without guaranteeing optimality.

Incidence matrix [*part* $\times$ *machines*] representing the assignment obtained is then presented to the operator (see Table IV, where all penalty weights equal 1). The latter may modify the assignment and introduce new machines, etc.

	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12
	T1	T2	T2	T3	T3	T4	T4	T5	T5	T6	T7	T7
	N1	N1	N3	N1	N4	N1	N3	N1	N2	N2	N1	N2
P1	0,0	1,0	0,0	0,0	0,0	2,0	0,0	3,0	0,0	3,0	0,0	0,0
P2	0,0	1,0	0,0	4,0	0,0	0,0	0,0	0,0	2,8	2,0	3,0	0,0
P3	0,0	0,0	0,0	0,0	0,9	0,0	0,0	0,0	0,0	2,0	3,0	0,0
P4	2,0	3,0	0,0	2,0	0,0	1,0	0,0	2,0	0,0	0,0	0,0	0,0
P5	3,0	0,0	0,0	1,0	0,0	4,0	0,0	1,0	0,0	0,0	0,0	0,0
P6	0,0	0,0	3,3	0,0	0,9	0,0	0,2	0,0	0,0	0,0	0,0	0,0
P7	1,0	2,0	0,0	0,0	0,0	2,0	0,0	3,0	0,0	0,0	1,0	0,0
P8	2,0	2,0	0,0	3,0	0,0	0,0	0,0	1,0	0,0	0,0	3,0	0,0
P9	2,0	0,0	1,7	0,0	4,6	0,0	0,0	0,0	1,8	0,0	0,0	0,9

Table IV. Incidence matrix [*parts* $\times$ *machines*].

6. Parallel Approach

The use of parallel computing for solving selection and the assignment problems described above is evident: for each machine j in the workshop ($j = 1$ to r), i.e., for each column of the incidence matrix $[parts \times machines]$ an optimization problem must be solved. These problems are completely independent and its treatment can be made simultaneously if a parallel computer is available.

6.1. Parallel Virtual Machine

Distributed parallel computing offers a higher performance to the applications that do not need a large parallel machine, but demand a higher power than these offered by a sequential machine [4, 5, 6]. The PVM – Parallel Virtual Machine and the MPI – Message Passing Interface are two examples of environment for message passing that allow us to build parallel virtual machines in workstations, generally machines RISC with Unix operational system [6, 7].

Although personal computers (PCs) and Windows operational system are nowadays being widely used, research works exploiting parallel computing in this environment have been reported only recently. Three works have been reported so far: WPVM [8] from the University of Coimbra, Portugal; the WPVM from the PVM-team [9], USA; and the PVM-W95 [3] from the University of São Paulo, Brazil. These works present alternative solutions for possessing parallel computing on a network of PCs running the Windows of 32 bits.

6.2. PVM-W95

The PVM-W95 is a message passage environment that offers required resources for developing concurrent programs using the C or C++ programming language, on a network of PCs running the Windows95/NT operating systems. This environment is entirely compatible with the original PVM platform for UNIX systems [3]. The PVM-95 was developed entirely based on the original source code. The significant difference between the PVM-95 and the original PVM resides in the use of object-oriented concepts: abstraction, encapsulation, polymorphism and inheritance [10]. The PVM-95 is composed of two main modules: a daemon and a library of routines with the PVM interface [3].

7. Parallel Implementation

The proposed method was written in language C and was executed in the PVM-W95. Computer network has NS processing nodes: one is responsible for the master process and the others for slave processes. These NS-1 nodes execute the method for solving the assignment problem for each type of machine.

In order to establish the minimum number of processors necessary for solving the problem, one of the processors realizes a test on the set of feasible solutions obtained by the solution of KNS for all types of machines and verifies if the optimal solution is evident without necessity to solve $PM01^j$. Otherwise, this processor builds the corresponding mathematical problems to solve $PM01^j$ for each type of machine and calculates their number of variables and constraints: based on this data, the program distributes the problems to solve to the available processors. If the number of types of machines is greater than NS-1, distribution of these supplementary problems is obligatory, allocating them first to the processors responsible for solving the problems of smaller size, and to the others afterwards.

The objective-function chosen for KNS is the number of machines in the shop and the penalty established for the machines in $PM01^j$ is equal to 1 for all machines. The mathematical model KNS is solved by dynamic programming and $PM01^j$ by implicit enumeration when the optimal solution for the assignment is not evident.

Communication between the master process and slave processes makes use of the routines which are specific for each kind of data. Such routines transform certain data (e.g., an integer value) into a chain of characters in ASCII. A header is coupled to that chain of characters, also in ASCII, which specifies the kind of variable that is being sent. Next, these data are processed using an inverse routine, which takes the data in ASCII and gets the same integer value that was sent.

The pseudo code s&a-par, below, summarizes the main steps of the parallel implementation.

```
pseudo code s&a-par
• construction of KNS mathematical models
• KNS solving (by dynamical programming or heuristic procedure)
• allocation of machine types to the processors
• for each type of machine $j$
    1. construction of the PM01$^j$ mathematical models
    2. model solving (by implicit enumeration or greedy algorithm)
• end for
end pseudo code s&a-par
```

8. Computational Results

Parallel implementation was tested in 10 job shop production systems found in the literature and in 2 examples generated at random $(R1, R2)$. Table V below shows the result of computational tests realized. In the 1st column, reference of the problem chosen is given (there can be more than one tested example from the same reference: 2 examples were extracted from the references [12, 15, 17]); in the 2nd the triplet $(m \times t[j] \times r)$, where m is the number of the type of machines available in the shop, $t[j]$ is the number of machines available for planning and r is the number of parts; in the 3rd the total number of jobs to process is given. In the other columns, T_{dp} is the time to run the sequential program, spent for the choice of machines by dynamic programming and T_{he} is the time by heuristic enumeration; T_{ie} is the time spent for the assignment of parts to machines by implicit enumeration and T_{gr} is the time of the greedy algorithm. In this Table, the character $(-)$ means computational time less than 0,01. Computational times are given in seconds (all PCs utilized in the PVM-W95 are Pentium, 200 MHz, 64 Mbytes).

Table VI compares the time, in seconds, to run the program using one processor and the number of processors initially established.

Table VII describes the speedup of the algorithm (Sp = Sequential Time / Parallel Time). Table VIII gives the Efficiency (Sp / Number of Processors).

Example	$(m, t[j], r)$	Jobs	T_{dp}	T_{he}	T_{ie}	T_{ga}
[11]	(5, 12, 7)	29	$(-)$	$(-)$	2, 36	$(-)$
[12] – 1st	(7, 13, 9)	45	$(-)$	$(-)$	0, 28	$(-)$
[12] – 2nd	(7, 21, 9)	45	$(-)$	$(-)$	2, 42	$(-)$
[13]	(15, 30, 10)	46	$(-)$	$(-)$	5, 16	$(-)$
[14]	(10, 25, 20)	49	$(-)$	$(-)$	2, 75	$(-)$
[15] – 1st	(20, 26, 20)	79	$(-)$	$(-)$	4, 62	$(-)$
[15] – 2nd	(12, 24, 30)	131	$(-)$	$(-)$	18, 07	$(-)$
[16]	(30, 50, 41)	128	$(-)$	$(-)$	53, 94	$(-)$
[17] – 1st	(16, 19, 43)	127	$(-)$	$(-)$	10, 27	$(-)$
[17] – 2nd	(16, 20, 43)	127	$(-)$	$(-)$	10, 27	$(-)$
$R1$	(20, 35, 30)	240	$(-)$	$(-)$	18, 23	$(-)$
$R2$	(10, 30, 30)	270	$(-)$	$(-)$	26, 15	$(-)$

Table V. Examples tested and computational time.

Example	[11]	[12]	[12]	[13]	[14]	[15]	[15]	[16]	[17]	[17]	R1	R2
Sequential Time	2,40	0,31	2,49	5,20	2,82	4,65	18,15	53,98	10,34	10,34	18,32	26,30
Parallel Time	1,76	0,11	2,25	4,61	1,93	4,28	5,55	38,72	5,49	5,49	5,55	5,82
Number of Processors	2	3	2	2	2	2	4	2	2	2	5	5

Table VI. Comparison between sequential and parallel computational times.

Example	[11]	[12]	[12]	[13]	[14]	[15]	[15]	[16]	[17]	[17]	R1	R2
Sp	1,36	2,82	1,11	1,13	1,46	1,09	3,27	1,39	1,88	1,88	3,30	4,52

Table VII. Speedup.

Example	[11]	[12]	[12]	[13]	[14]	[15]	[15]	[16]	[17]	[17]	R1	R2
Efficiency	68,0	94,0	55,5	56,5	73,0	54,5	81,8	69,5	94,0	94,0	66,0	90,4

Table VIII. Efficiency (%).

9. Conclusions

With parallel implementation of the two-phase method proposed for solving the assignment problem of parts to machines, some important conclusions were reached:

- The method presented here allows to obtain good results while maintaining algorithmic simplicity and making effective use of parallel computing facilities.
- The method uses a combination of exact and approximate algorithms for solving the problem and obtaining an optimal or feasible solution within a very reasonable computational time.
- Implementation is efficient, because in the tests accomplished, efficiency in most of the examples was above 66%.
- Implementation has better performance when the size of the problems increases.

Acknowledgment

This work has been supported by the Ministry of Science and Technology of Brazil through CNPq – Brazilian Center for Research and Development of Science and Technology.

References

[1] M.R. GAREY, D.S. JOHNSON, *Computers and intractability: a guide to the theory of NP-completeness*, Freeman, 1979.

[2] C.H. PAPADIMITRIOU, K. STEIGLITZ, *Combinatorial optimization: algorithms and complexity*, Prentice Hall, 1982.

[3] M.J. SANTANA, P.S.L. SOUZA, R.H.C. SANTANA, S.R.S. SOUZA, Parallel Virtual Machine for Windows95, *Proceedings 3rd EuroPVM'96*, 1996.

[4] G.S. ALMASI, A. GOTTLIEB, *Highly parallel computing*, Benjamin Cummings, 1994.

[5] A. GEIST, A. BEGUELIN, J. DONGARRA, W. JIANG, R. MANCHEK, V. SUNDERAM, *PVM: Parallel Virtual Machine – a user's guide and tutorial for networked parallel computing*, MIT Press, 1994.

[6] A. GEIST, A. BEGUELIN, J. DONGARRA, W. JIANG, R. MANCHEK, V. SUNDERAM, *PVM3 User's guide and reference manual*, Technical Report Oak National Laboratory, 1994.

[7] O.A. MCBRYAN, An overview of message passing environments, *Parallel Computing*, 20 (1994), pp. 417–444.

[8] A. ALVES, L. SILVA, J. CARREIRA, J.G. SILVA, WPVM: parallel computing for the people, *Proceedings HPCN'95*, 1 (1995), pp. 582–587.

[9] M. FISHER AND J. DONGARRA, Another architecture: PVM on Windows95/NT, *Proceedings 3rd EuroPVM'96*, 1996.

[10] K. IRVINE, *C++ and Object-Oriented Programming*, Prentice Hall, 1997.

[11] J.F.F. RIBEIRO, B. PRADIN, A methodology for cellular manufacturing design, *International Journal of Production Research*, 31(1993), pp. 235–250.

[12] A.B. NORONHA, J.F.F. RIBEIRO, C.M. RIBEIRO, Job Shop Scheduling with Disjunctive Constraints, *Brazilian Journal of Management and Production* (in Portuguese), 3 (1996), pp. 204–219.

[13] H.M. CHAN AND D.A. MILNER, Direct clustering algorithm for group formation in cellular manufacture, *Journal of Manufacturing Systems*, (1) 1981, pp. 399–416.

[14] G. SRINIVASAN, T.T. NARENDAN, B. MAHAVEDAN, An assignment model for the part-families problem in group technology, *International Journal of Production Research*, 28(1):145–152 (1990).

[15] G. HARHALAKIS, R. NAGI AND J.M. PROTH, An efficient algorithm in manufacturing cell formation for group technology applications, *International Journal of Production Research*, 28 (1990), pp. 185–198.

[16] K.R. KUMAR AND A. VANELLI, Strategic subcontracting for efficient disaggregated manufacturing, *International Journal of Production Research*, 25 (1987), pp. 1715–1728.

[17] J.L. BURBIDGE, *The introduction of group technology*, John Wiley, 1975.

Received: January, 2003
Accepted: December, 2003

Contact address:

José Francisco Ferreira Ribeiro
USP – University of São Paulo
Department of Computer Science
Av. Trabalhador SaoCarlense, 400
13560-970 Sao Carlos
Brazil
e-mail: jffr@icmc.usp.br

JOSÉ FRANCISCO FERREIRA RIBEIRO is a professor at the University of São Paulo at São Carlos, SP, Brazil. His research field is parallel computer and mathematical programming.