# TASK SCHEDULING ON MULTIPROCESSORS

by

R. Mehrotra & S.M. Talukdar

December, 1932

DRC-18-55-82

# TASK SCHEDULING ON MULTIPROCESSORS

Ravi Mehrotra Sarosh N. Taiukdar

Department of Electrical Engineering
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

## ABSTRACT

The paper describes a technique for estimating the minimum execution time of an algorithm or a mix of algorithms on a multiprocessor. Bottlenecks that would have to be removed to further reduce the execution time are identified.

The main applications are for designing special purpose. dedicated multiprocessors. Today, a bewildering array of computer components * processors and devices with which to interconnect them - are available. The future will bring even more of these components. To intelligently choose mixes of them one needs systematic procedures.

In the procedure of this paper the multiprocessors are modelled by P. a set of processors and R, a set of resources that the processors can use. The algorithms are modelled by T. an ordered set of tasks. The problem of optimally assigning tne processors to the tasks while meeting the resource constraints is NP-complete. A heuristic using maximum weighted matchings on graphs has been devised that is extremely fast and produces solutions that are reasonably close to the optimal solutions. The heuristic has been coded in Fortran and illustrations of its use included.

## 1. INTRODUCTION

### 1.1 The General Form of a Distributed Multiprocessor

Most existing codes have been written for Von-Neumann, general purpose computers with targe virtual memories. However, it is now possible to assemble non-Von-Neumann architectures, often using just off-the-shelf components. The general form of one such class of architectures is shown in Fig. 1.1. The processors may have diverse processing, input and output characteristics and may be physically close or geographically separated. The communication network may use a transmission means (such as wires.satellites and optical fibers) and a variety of configurations (such as stars.trees and loops). We will elaborate on these alternatives in 1.4 and 1.5.

### 1.2 A Very Brief Review of Previous Work [i]-[5]

Research into the use of special computers for power system appreciations has concentrated on three limited possibilities:

1. The exclusive use of large vector machines like the CRAY-1

2. The combination of a host machine with an array processor like the AP120-B.

3. Homogeneous multiprocessors (large numbers of identical processors symmetrically interconnected)

These research efforts have met with some, but not spectacular, success. The reason is that power system algorithms contain a wide variety of tasks. Some work well on vector machines and array processors, others do not. Some work well on homogeneous multiprocessors. others do not. Therefore, the exclusive use any one limited hardware arrangement will inevitably lead to severe bottlenecks.

### 1.3 The Design Problem

To alleviate bottlenecks we need to ask the question: How can we assemble a computer system with the diverse skills needed to efficiently process all the tasks in a given power system algorithm or mix of algorithms? One •••a*' to go about answering this question is to take the folio.-.'ng four steps:

1. Identify the computer components that are no*/ available or will soon be available.
2. Categorize the tasks or, alternately, identify a set of primitives from which the tasks can be synthesized.

3. Determine how effective each component is for each primitive.

4. Devise a scheme for assembling mixes of components to best handle given mixes of primitives.

The result of taking these steps will be a computer of the type shown in Fig. 1.1 and dedicated to a mix of algorithms.

The emphasis of this paper is on step 4. We will discuss the other steps but to considerably lesser degrees.

### 1.4 Network Alternatives

Computer communication networks can be divided into three categories.

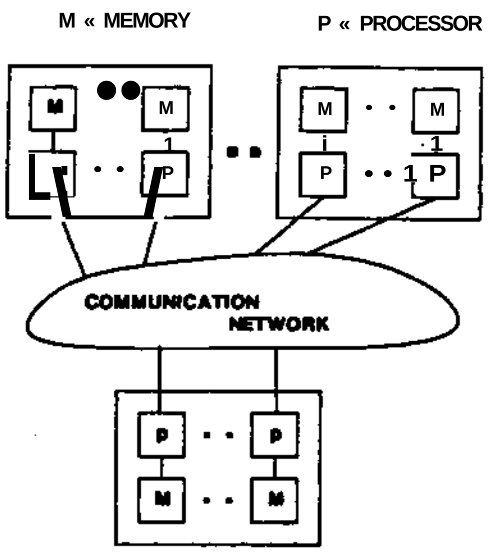


Fig. 1.1 A General Fdrm of a Multiprocessor.

1. Large scale computer networks such as ARPANET which is designed to interconnect dispersed and dissimilar computers, allowing users and programs at one computer center to access and interactively use facilities at other, geographically remote, centers. Other examples are TYMNET and GE networks which facilitate commercial time sharing [63.

2. Loca! Area Networks [7]-[11]. When the processors connected to the communication network *are* within 2-3 kilometers of one another, the network is called a local area network. Local area networks for minicomputers such as Xerox . Corporation's ETHERNET and Zilog Corporation's ZNET have now been in existence for sometime. The formats, protocols, operational sequences and logical structures for functions needed to achieve meaningful communication among processor units connected to such networks are readily available. Infact the processor interfaces will soon be available on VLSI chips and their costs are expected to drop to the range of hundreds of dollars.

2. Bus Structures. Processors can be interconnected '". :.--f. * 'T-'JCT c-f s?".!Ctures is posriMe [12]-[16]. An example of a s^ple bus structure is the UNIBUS of a PDP-11 which can be used to connect a host computer to a number of peripheral processors. Examples of more complicated bus structures are those used in multiprocessors such as C.mmp, Cm* and BBN PLURIBUS.

1.5 Processor Alternatives

The number of processor alternatives is extensive and growing. The alternatives can be categorized on the basis of a number of factors such as speed, availability and cost. Whatever classification scheme is used, it is difficult to keep the categories from overlapping. One set of categories is:

- Special purpose, dedicated VLSI processors [171-[24] that will soon be appearing in large numbers to do tasks like matrix multiplication and LU factorizations very fast.
- Relatively inexpensive programmable processors including array processors (such as the AP 12OB. FPS 100 and FPS 164) and microprocessors (such as the Z80 and Intel's 8086).
- Bit slice processor elements (such as Texas Instruments S481/LS480 that can be assembled into systems for specific applications.
- Minicomputers like DEC'S VAX/780 and PDP-11/70.
- Large, general purpose machines like IBM's 3030, Burroughs B5000, CDC's 7600. as well as large vector machines like the CRAY-1 and STAR-100.

1.6 Goals and Organization

In the past, computer alternatives were evaluated bv tedious benchmarking or almost as tedious simulations, in view of the variety of alternatives now available, neither of these approaches is practical until the very last stages of the search for a computer system. For the early parts of the search one needs evaluation tools that are qi»c^ and easy to use. The rest of this paper will be devoted to the development of one such toot and to a simple example of its use. Specifically, the tool is an efficient, interactive program that estimates the minimum execution time and identifies the bottlenecks for a given mix of tasks on a given mix of computer components. The results enable the user to determine whether the mix of components is promising and how to incrementally change it to improve performance.

Section 2 gives a formal aescnpnon of tne p'OLte-r. we shall consider. Section 3 indicates a solution metnoa after a brief review of methods that have been used to solve similar problems. The procedure described in section 3 has been coded as a user friendly interactive program in FORTRAN on DEC-20 system and is used to study a proposed multiprocessor architecture in section 4. Section 5 summarizes the results obtained so far and lists further work to be done.

## 2. MATHEMATICAL FORMULATION OF THE PROBLEM

This section establishes the basic vocabulary for the remainder of the paper and gives a precise mathematical formulation of the problem to be considered.

### 2.1 Algorithms

An algorithm A is described by A*{T.*| where T and *c* are sets defined as follows:

T is a set of tasks {$T_r$ T$_\wedge$ . . . $T_N$) and the set [illegible] denotes the partial ordering relation on T such that $T_p$ *a* T implies that the execution of tasks M called the successor of T) cannot begin until the execution of T (called the predecessor of $T_\$$) has been completed. We will represent an algorithm A by a directed graph called the task graph $G_A$(V.E) of A so that there is one node in V for each task in T and and one arc in E for each relation in partial order *c.* When *a* is empty the tasks are called independent. It is assumed that the tasks to describe A are chosen from a finite set of tasks $T^p$ « {T^. $T_2^P$. . . $T_n^p$) of n primitive tasks. Each task T < T corresponds to some primitive task $T_y^p$ * $T^p$.

### 2.2 Multiprocessors

We may think of the multiprocessor architecture at a high level as having been assembled from processors that execute tasks in $T^p$ with the use of certain resources such as disk drives, tapes, memory and the interconnecting devices including the buses and data links that form the communication network of the system. The multiprocessor system MP with M processors and L resources will be denoted by MP{P,R> where P«{$P_1$. $P_r$ . . $P_M$) is a set of M processors and R MR,. $R_2$. . . $R_L$) is a set *of* L resources.

## 2.3 Algorithm - Multiprocessor Interaction

Each task T < T may be executed on anv processor in P.We define a'function ^(TWt,.. t^. . . t$_{M>}$) so that the value rf t represents the exnecterl time it tales to evrcute task T on processor P. Furthermore we define a function r(T)*(r '. r.. . . r{) to represent the resource requirements of task T such that r is equal to the amount of discrete resource R needed while executing T and y?(RJ is the total units of $R_x^x$ in the system.

## 2.4 Execution time of A on MP(P.R).

Let *r(l)* represent the starting time of the execution of task T( • T.'

Define X(k)«1( if task T is executed on processor P( at time k and zero otherwise! It is assumed that time is measured in terms of equal and indivisible units. Using the notation ' introduced in this section we define a feasible schedule to be a mapping ¥: T->I such that the following 3 conditions are satisfied. (1 is a one dimensional space of integers representing time).

C1: $\sum_{i=1..M} X_{ij}(k) = 1$ for j*l..H, all $k \in I$

C2: If $T_i \ll T_j$ then

$r(T_j) \geq r(T_i) + \sum_{p=1..M} t_{pi} X_{pi}(k)$ for i,j=l..K, all $k \in I$.

C2: $\beta(R_i) \geq \sum_{j=1..N} \sum_{p=1..}$ $r_{iy}$ $X_{pj}$U) fcr all $k \in I$.

Cl is needed to avoid the assignment cf a task to more than one processor. C2 is a statement of the procedure constraints of A. C3 is needed to ensure that the resources required by a job will be available while the job executes.

Corresponding to each feasible schedule M': T->I we define the execution time of the algorithm A on MP as :

$L_A^{MP}$ « jnin {3r(k)«0 for i«Lm, j«UN}

Thus the problem of finding the optimal assignment of tasks in the algorithm A on a multiprocessor MP so as to minimize the overall execution time may be stated as GSP

£$^G$SP] minimize $L_A$
subject to ¥:T->I

We can find a lower bound $L_A^{oo}$ on $L_A^{MP}$ as follows:

For every node n( t G(V.E) define the weight of n,, W(n). as

W<n>min (t,,. t$_{2i}$. . . . tJ

Define the length of a directed path from node n to node $n_t$ to be equal to the sum of weights of all the nodes in the directed path from $n_s$ to $n_t$. The longest directed path from a node with no predecessor to a node with no successor represents a lower bound on *ij**. This lower bound $L_A^{oo}$ is obtained.by assuming that all the tasks in •G<V,E) are assigned to the processor on which they take minimum execution time and there is a sufficient number of processors and resources in the system.

If the solution to GSP gives a value of $L_A^{MP} \gg L_A^{oo}$ then the elements of the set P and R may be modified to reduce the difference between $L_A$** and $L_A^{oo}$. This allows us to reconfigure a multiprocessor system that is most suited for executing the specific algorithm. The solution procedure ( see section 3 ) used to solve *(2.4)* enables us to identify, what limits performance, thus suggesting a natural modification of the set P and/or R.

## 2.5 A Cost Constraint

Let CP(P) represent the cost of processor P(. i«1...M.

A cost constraint may be added to obtain GSPC as follows:

[GSPC] minimize $L_A$
subject to ¥:T->I
$\sum_{i=1..M}$ CP($P_i$) ≤ C

All processors in the set P are not necessarily used. The solution procedure would select the particular mix of processors that minimizes the overall execution time with the total cost of processors in the mix being no more than C.

## 3. SOLUTION PROCEDURE

GSP is a notoriously hard combinatorial analysis problem known as the General Scheduling Problem. This problem is MP-corr.plete. Instead of seeking po'yr.crr.io! time optimal algorithms for NP-compiete problems, cne uses heuristic \Ot opproAin:uie'/ algorithms wine;, t.opfc^u... ,.ci^. "good" solutions in polynomial time. Most work done m the area of scheduling has been devoted to the case when all processors in the system are identical[25]-[25]. Some enumerative and iterative techniques such as 'local search' and 'branch and bound' have been applied to subproblems of GSP [36]-[39]. But there are no heuristics for solving GSP itself and branch and bound techniques are computationally impractical for solving it.

We will proceed to develop a heuristic technique for solving GSP. The technique is based on finding maximum weighted matchings on graphs. It yields reasonably good solutions to GSP and GSPC.

The essential steps are:

1. Input Edge List Matrix of $G_A$(V.E). Execution Time Matrix [$t_{ij}$] and Resource Requirement Matrix [r$_{>J}$] (see Example 3.1).

2. Assign levels to the nodes. $T_x$, of the task graph GJV.E). Intuitively, levels assigned to nodes are distances from a node with no successor and represent the precedence structure of $G_A$<V.E).

3. Making use of the levels of the nodes, assign corresponding tasks on the processors, disregarding the resource constraints. This step is carried out by finding maximum weighted matchings*

4. Schedule the tasks on the processors they have been assigned to. taking resource constraints into account. Make a list of resource shortages if any.

5. Repeat steps 3 and 4 until all tasks have been scheduled.

6. Output the schedule and the list of resource shortages (see Example XI).

The details of steps 2-5 are given in the appendix.

The heuristic to solve GSPC is similar to the above heuristic. The cost constraint is factored into the solution process by including it in the objective function of the maximum matching problem [40]-[42].

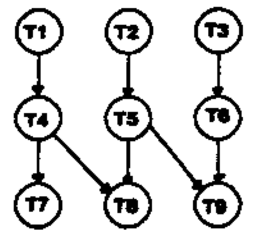


Figure 3.1. A task graph, G (V,E)
A

Table ai(a). Edge List Matrix.

$$D\begin{bmatrix} 2 & 3 & 4 & 4 & 5 & 6 & 6 \\ 5 & 6 & 7 & 8 & 8 & 9 & 9 \end{bmatrix}$$

Table ZJ2. Resource Shortage Table.

| Task | R1 | R2 |
|---|---|---|
| T4 | 3 | 2 |

Table 3.1(b). Execution Tims Matrix.

| | T1 | T2 | T3 | T4 | T5 | T6 | T7 | TB | T9 |
|---|---|---|---|---|---|---|---|---|---|
| PI | 2 | 2 | 1 | 1 | 1 | 1 | 3 | 1 | 1 |
| P2 | 3 | 1 | 1 | 3 | 1 | 2 | 1 | 2 | 1 |
| P3 | 1 | 1 | 2 | 4 | 3 | 1 | 4 | 1 | 3 |

Table &1(C). Resource Raquirement Matrix.

| R | A | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | T9 |
|---|---|---|---|---|---|---|---|---|---|---|
| R1 | 3 | 1 | 1 | 1 | 3 | 1 | 0 | 2 | 0 | 3 |
| R2 | 2 | 0 | 1 | 1 | 2 | 2 | 0 | 1 | 0 | 0 |

A * Amount of Resource.

| | | | | | |
|---|---|---|---|---|---|
| PI | T3 | | T4 | T9 | |
| P2 | T2 | T5 | | | T7 |
| P3 | T1 | T6 | | TB | |
| TwneUnit | 1 | 2 | 3 | 4 | 5 |

Fig&2. Pictorial Representation of the Schedule.

Example 3.1. Consider a multiprocessor system MPt'P.R) vwiih $P^*\{P_,.P_2.$ p,} end R«{R.,$R_2$).

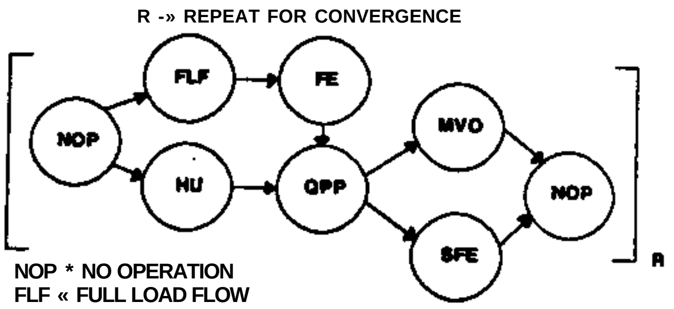
The inputs required, for task graph of Fig 3.1 are given in Table 3.1(aHc).

Table 3.1(a) is a matrix representation of $G_A$<V.E). The values of [t ] and [r (] are specified in Table 3.Kb) and 3.1(c) respectively. (R,)^ units and($R_2$>«2 units. L^-3.

Fig 3.2 is a pictorial representation of the output of the procedure. Note that $L_A^{MP}$*5. Table 3.2 is the output which indicates the bottlenecks which must be removed to reduce the overall execution time. Task T4 could not be started in parallel with T5 and T6 because of resource shortage. It needed 3 units of $R_1$ and 2 units of $R_2$ and none were available. If we let $\beta(R_1)$=6 and /ft'$R_2$)*4 then L would be equal to $L_A$°°.

## 4. BJJJSTRATION OF THE DESIGN PROCESS

The solution procedure outlined *in* the previous section has been translated into a user friendly, interactive, FORTRAN program called SNONUET. It allows the user to modify the input parameters until either satisfactory execution time is obtained, or no further improvement is possible. SNONUET has been tested on a number of randomly generated examples and it produced near optimal schedules in most cases. The purpose of this section is to illustrate some of the uses of SNONUET. To do this we will use a simple example, chosen for explanatory purpose rather than realism.



NOP * NO OPERATION
FLF « FULL LOAD FLOW
HU » HESSIAN UPDATE ( BFGS FORMULA )
OPP * QUADRATIC PROGRAMMING PROBLEM SOLUTION
FE .FUNCTIONEVALUATION
MVO « SOME MATRIX AND VECTOR OPERATIONS
SFE * FUNCTION EVALUATION AND SEARCH

Fig 4.1 (a) Task Graph of *mn* Optimum Power Flow using high level primitives.

### 4.1 Algorithmic Primitives

The first step in preparing the input data for SNONUET is to identify a set of primitive tasks, $T^p$, in terms of which to describe the algorithm^) in question.

The primitives can be at various levels. Very high level primitives result in simple task graphs with a few nodes. For instance, an optimum power flow [43] can be described in terms of high level primitives by the task graph in Fig 4.1(a). Or. using the primitives given in Tatle *A.",,* v:c could expand each node of the task graph. The task graph corresponding to FLF is shown in Fig 4.Kb).*

A reasonable way to proceed is to use high level primitives for the initial design and then refine the design with lower level primitives.

### 4.2 Processor arid Communication Network Alternatives

The ?frnr.1 *r.ter.* in p»ep?"»nc :he input data for SNONUET is to choose the processor and communication network alternatives to be considered.

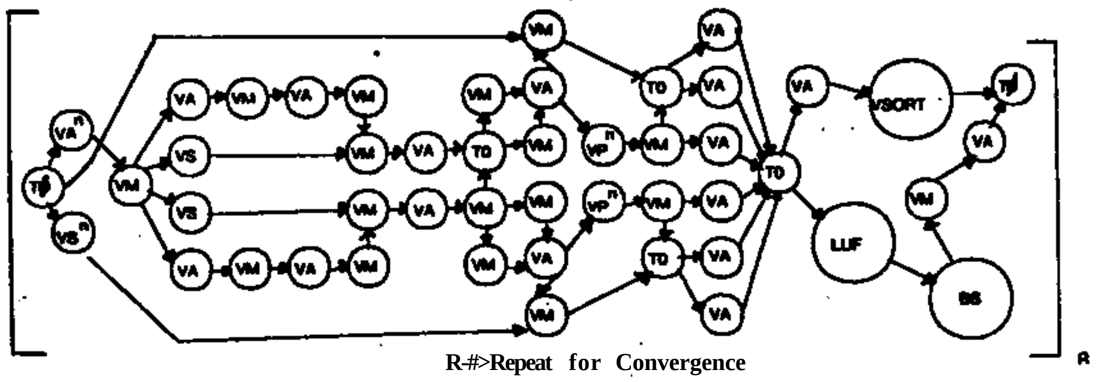
We start with a unibus multiprocessor system shown in Fig 4>2.,**which is a special case of the general

TABLE 4.1

| | Set of Primitive Tasks | Mnemonic |
|---|---|---|
| T∅ | No operation (used for synch- ronlzatlon) | T0 |
| T1 | Vector Add ($C^* \cdot a_\pm + b_{if} i$«l..n) | VA |
| T2 | Vector Inner Product ($C - \sum_{i=1}^{n} tt^\wedge b_\pm$) | VP |
| T3 | Vector Scale ($b_i$«$Ca_{i}$.t 1-1..a) | VS |
| T4 | Vector Divide ($b_i$.-a^C, 1*1..n) | VD |
| T5 | Vector Multiply ($C_\pm$ - $a_\pm b_{it}$ 1-1..a) | VM |
| T6 | Vector Sort (arrange elements in an increasing order) | V Sort |
| T7 | LU Pactorzation | LOT |
| T8 | Back Substitution (used to solve a set of linear equations). | BS |

* Page 5
** Page 6

Fig 4.1.(b). The task graph corresponding to node FLF (Full loud flow) of Fig. *l.l(a). $(T)^n$ denotes that n of (T) may be done in parallel.



R-#>Repeat for Convergence

multiprocessor system of Fig 1.1 - the communication network is now the data channel of the host computer. The motivation for using the common data bus is the simplicity of the interconnection, also if the communication over the bus does not limit performance there would be no need to consider more sophisticated interconnection schemes. Each special purpose device is a processor/multiprocessor realized in VLSI, with its own private memory. The unibus of the system is considered to be a resource of the system. If the total time needed for all data transfers over the bus of all tasks in G <V.E) at any level is found to be more than 10H of L °° then the bus is considered to be a bottleneck. The details of the bus modelling procedure are described in [42]. SNONUET finds the latest finishing time of all tasks and identifies bottlenecks. If the unibus of the system is not a bottleneck, the number of special processors of a given type may be increased to check if further reduction in the overall execution time is possible. On the other hand if the unibus turns out to be a bottleneck, we introduce another bus amongst the processors sharing the congested bus. to relieve the congestion and improve speedup. The above steps are repeated until no further reduction in execution time results.

## 4.3 Cost and Time Data

The third and the major step in preparing the input data for SNONUET is to estimate the cost of the processors and the task running times.

We consider the execution of the GJtV.E) of FLF shown in Fig 4.Kb) on the multiprocessor of Fig 4.2. Three different types of processors are considered, an array processor AP (such as AP 1208) and two special purpose VLSI peripheral processors SPl and SP2. The estimates of the execution time of the host and the three types of processors considered are listed in Table 4.3.1. The per unit cost of the three types of processors is shown in Table 4.3.2.

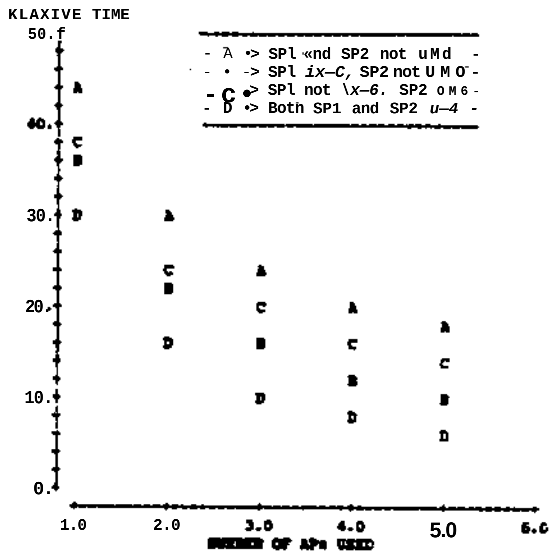
## 4.4 Results

The output of SNONUET when no cost constraint was placed indicated that overall execution time could be reduced by increasing the number of APs to 5. The overall execution time obtained by SNONUET has been plotted vs the number of APs in Fie 4.3.1, after sca«inq it so that the stand alone host could sequentially execute FLF in 100 units of time.

The overall running time vs cost is plotted in Fig 4.3.2. It is important to note that SNONUET with a cost constraint generates only the points on the broken line. The other points correspond to mixes of processors that should not considered because they provide lesser speedups for the same cost.

TABLE 4.3.1.
Istlatt* of Execution Tlae

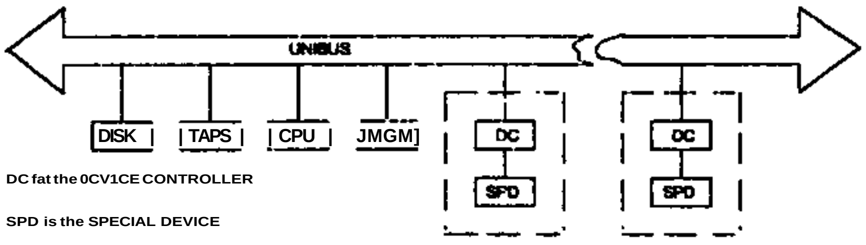
| Prlaltlv* Task | BOST | AP | SPl | SP2 |
|---|---|---|---|---|
| T# | 0 | 0 | 0 | 0 |
| TA | 100 | 35 | oa | ei |
| *P | 1100 | 410 | at | m |
| VS | 1000 | 340 | 00 | 0f |
| TO | 1000 | 340 | 00 | 00 |
| *H | 1000 | 340 | 00 | 00 |
| • sort | 4§50 | 5000 | 700 | m |
| UJF1 »*J | 4000 | 3000 | • | 400 |



rig. 4.3.1. Execution times obtained by *UOKVET.*

TABLE 4.3.2

| Processor | Cost |
|---|---|
| AP | 50 |
| SPl | 10 |
| SP2 | 20 |
| HOST | 0 |



Rg4.2. A Unfeus multiprocessor system.

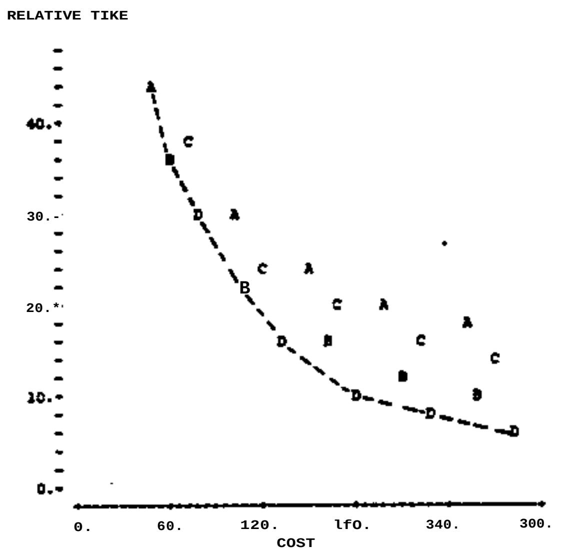


Fig. 4.3.2. Plot of relàtive tlae vs cost. Points on the Broken line are oBtauned By SBOVUET.

## 5. CONCLUSIONS

This paper has described a systematic procedure that is useful in the selection and design of dedicated multiprocessors. The procedure has been coded into an interactive FORTRAN program called SNONUET.

Before SNONUET can be used one must break the algorithm^) into ordered tasks, select a set of processors for consideration and select a set of resources for the processors to use. One must also estimate the time and resource requirements for each primitive task. SNONUET will then schedule the tasks on the processors and identify bottlenecks that are must be removed if further decreases in the overall execution time are to be obtained. One may include the cost of the processors and an upper limit on what the system is to cost. SNONUET will select a subset of processors and schedule them so as to minimize the execution time of the algorithm(s) and satisfy the cost constraint. This procedure enables us to plot the optimal speedup vs cost for a fixed communication network. It would be very useful when selecting a set of processors from those commercially available, when a local area network is used as the communication netork of the system.

The example in Section 4 was chosen to illustrate the use of SNONUET, and not as a realistic design exercise. It could however be used for designing multiprocessors if they were to be dedicated to solving load flow problems.

Some further work needs to be done to obtain a better way to model the communication network when its architecture is specified and queueing delays are involved as a result of packet switching. In the present model, for each task requiring the communication network, expected queuing delays are added to the message transit times. The sum is treated as a deterministic time for which the resource corresponding to the communication network may not be used by another task. This is a major drawback of the procedure and we are working towards a remedy.

## 6. ACKNOWLEDGEMENT

This work was supported in part by the Electric Po.ver Research Institute under project RP1764-3.

## References

1. Chen, F.M., "Evaluation of an Array Processor for Power System Applications," *1979 Power Industry Computer Applications Conference Procee*OMigx.. CE£. 197S. pp. 345 350.

2. *Podmo*. tLjnmnon,* ULwangM. Britten. .Land \finnant S.. "Appacafcons of an Array Processor for Power System Network Cawnpmafcom." *1979* flMWr *taOustry Computf Applications Conference Proceedings*., EEE.T979. pp. 32*330.

3. Happ. K K. PMte. C and Wbgan. K. A.. ~An. . . . . . . .nt of Cowpmor Tactmotogy tor Large Scale Power System Simulation.," *1979 Power Industry Computer Applications Conference Proceedings*., IEEE, 1979, pp. 316-324.

4. •an*. F. M-, van Nam J. E-. Kane and Seng-Caul. -Tha uat of a Multiprocessor Network for the Transient Stability Problem.," *1979 Power Industry Computer Applications Conference*., IEEE, 1979, pp. 337-344.

5. PMdiard, ft and Petãa. C "High-Spaad Fewer Fieae Lawg Attached Scientific (ARRAY) Processors.," 1981 Pfrawr *to+tmy Cam*v*r ***cmam* Canfimwic*.. CEE. fcfay 54 1981. pp. 149-155.

e. Schwartz, Mischa, *Computer Communication Network Design and Analysis*, Prentice-Hall, Inc., Englewood Cliffs, N.J. 07632, 1977.

7. Unger, Brian, Birtwistle, Dan-. *Lomam.* G«e.. Billion. PML. Ha***. CaroL and Jain. Navindra., "An OASIS Simulation of ZNET Microcomputer Network*.." «£ *kUC/tO.* Attgust 1982, pp. 70-83.

a Metcalfe, R. M. and Boggs, D. R., "Ethernet: Distributed Packet Switching for Local Computer Networks.," *COMM. ACM.* Vol. 19, No. 7, July 1976, pp. 395-404.

ã. Thurbar. *Kmmm* J.. Fmoawt and Harvey A.. "JtachaactMral Conaidaraaont tor Lpcat Computer Networfca." «nx.*Hm*M7*Com.* *Distributed Computing Systems*, Oct 1979. pp. 131-142.

10. Afat Map Land Von, OBMJ.. The *Chmgfmg* Economics for Computer and Terminal Interconnection-An Overview.," Technical R[illegible]

11. OaM. Vegan IC. IJao of aMioto IniliiBitii in *e Xonx Moftaorii Syatam." *Computm,* OcU*tri9K.pp.«91.

12. Wolf, W. A., "Minicomputer Complexes: Progress and Proapactr," *Compvm* Seiane* *m** Scientific Computing., Cnoga. J.M.. a*.. Puc 01 *• 3rd CASE Conf. on Saanaae Computing., April 1978, pp. 293-305.

13. Stone, H. S., "Computer Architecture in the 1980's," *Computer Science and Scientific Computing*., Ortega, J. M., ed., April 1976, pp. 285-281.

14. Lanahan, J. J^ "Performance of Cooperative Loosely Coupled Microprocessor Architectures toonmmraea^aOaiaBase Task.," *IEEE Trans.* Vol. C-29, No. 2, Feb 1980, pp. 161-178.

15. Fanao. Cdw^d P.. Ghaw, W. and Ji•!••••", PH*p C, "A Concurrent Computer Architecture and a Ring Boatd liiUmimaiion.," Teen, report. Cawpming Laboratory, University of New Castle upon Tyne, England., 197-.

16. Hane, J.Archer, and Smith, David R., "Simulation Experiments of a Tree Organized Multicomputer.," *The 6th Annual Symposium on Computer Architecture*., April 1979, pp. 83-89.

17. Kung, S.Y., "Matrix Data Flow Language for Matrix Operations on Dedicated Array Processors.," *Proc. ECCTD* 1981, *The Hague, The Netherlands*., Aug 1981, pp. 393-398.

18. Kung, S.Y. and Rao, D.V. Bhaskar., "Highly Parallel Architectures for Solving Linear Equations.," *Proc. ICASSP* 1981, Atlanta, 1981, pp. 39-42.

19. Kung, S.Y., Arun, K. S., Rao, D. V. Bhaskar, and Hu, Y. H., "A Matrix Data Flow Language / Architecture for Parallel Matrix Operations Based on Computational Wavefront Concept.," *Carnegie-Mellon University Conference on VLSI Systems and Computations*., Oct 1981, pp. 235-244.

JO. •uric Mnha Hand Maad. Carwar A.. -MSartal amar Product Processors in VLSI.," *Proceedings of the Second Caltech Conference on VLSI*., January 1981, .

21. Kung, S.Y., "VLSI Array Processor for Signal Processing.," *Conference on Advanced Research in Integrated Circuits, MIT, Cambridge, MA.*, January 28-30 1980, .

22. Speiser, J. M. and Whitehouse, H. J., "Architectures for Real-Time Matrix Operations.," *Proceedings, GOMAC*, November 1980, .

23. Kung, H. T., "Let's Design Algorithms for VLSI Systems.," *Proceedings CALTECH* Conf. on VLSI, January 1977, pp. 65-90.

24. Kung, H. T., "The Structure of Parallel Algorithms.," Advances in Computers, VW.1t.1M0. pp. 65-111.

25. Coffman, E. G., editor, Computer and Job-Shop Scheduling Theory., Wiley, 1976.

26. Lam, S. and Sethi, R., "Worst Case Analysis of Two Scheduling Algorithms.," SIAM Journal of Computing, Vol. 6, 1977, pp. 518-535.

27. Goyal, D. K., Scheduling Equal Execution Time Tasks Under Unit Resource Restriction., PhD dissertation, Washington State University, 1976, Computer Science Department

28. Leung, J. Y. T., "Bounds on List Scheduling of UET Tasks With Restricted Resource Constraints.," Information Processing Letters, Vol. 9, 1979, pp. 167-170.

29. Jaffe, J., Parallel Computation: Synchronization, Scheduling and Schemes., PhD dissertation, Massachusetts Institute of Technology, 1979, Department of Electrical Engineering and Computer Science

ao. Ullman, J. D., "NP-Complete Scheduling Problems.," Computer and Systems, 1975, pp. 384-393.

31. Ullman, J. D., "Polynomial Computer Scheduling Problems.," Operating Systems Review, 1973, pp. 96-101.

32. Coffman, E. J., editor, Complexity of Sequencing Problems., Wiley, Computer and Job Shop Scheduling, 1976, pages 130-164

33. •runo. J-. Cofciw. E. J. an* Safe It. "Scheduling Independent Tasks to Reduce Mean finaNag Taw.." CAOML1t?4. pp. 3tt»7.

34. Lenstra. J.R. aad rwweoi, M n i H Q , -CowpHiiHj «f Scheduling Under Precedence Constraints.," Operations Research, 1978, pp. 22-35.

35. Uoytf. E.L-. Se»i#M>m T M * S I I W I •#* l i i i mi i.. PhD dissertation k M. I. T$_M$ May 1W0. Department« «f *Btttnat* Eia*Mmg and Computer Science

36. Held, M and Karp, R., "A Dynamic Programming Approach to Sequencing Problems.," SIAM JL Appl «•» ta 1.1W2.P*. itS-210.

X, 37. Horowitz, E. and StfMi. S.K..-CMd wd Approximate Algorithms for Scheduling Nonidentical P»0fMWI.,~ JMCtf2X2. *tVr*.* pp.317-327.

38. Coffman, E. G., editor, Enumerative and Iterative Computational Approaches., John Wiley & Sons, Inc, Computer Job/Shop Scheduling Theory, 1976.

39. Kogan, M. J., Heuristic Programming Applied to Scheduling Problems., PhD dissertation, Princeton University, 1970.

40. Edmonds, J. "Matching: A well Solved Class of Integer Linear Programs.,~ *Froc al ** C*9*f* iwt. Co*. on *Como* . jStructures and Their Appl., Gordon and Breach, 1970, pp. 89-92.

41. Mehrotra, R., "Maximum Weighted Matchings on Bipartite Graphs.", Technical Report, Carnegie Mellon University, December 1982.

42. Mehrotra, R., "An Algorithm Based on Finding Weighted Matchings on Graphs to Schedule Tasks on Multiprocessors.", Technical Report, Carnegie Mellon University, December 1982

43. Talukdar, S. N. and Giras, T. C., "A Fast and Robust Variable Metric Method for Optimum Power Flows.," IEEE Transactions on PAS, Vol. PAS-101, No. 2, Feb 1982, pp. 415-425.

## APPENDIX

The nodes, $T_{x>}$ of the task graph $G_A(V,E)$ of an algorithm A are assigned two levels $L^B(T_x)$ and *L*(TJ* by the algorithm below

Algorithm Assign Levels:

1. If T has no successors, then $L^B(T)$ «1; otherwise, $L^B(T)$ «1 • max{L^Ivy

2. Let $L^B(T_{majr})$ represent the smallest integer such that $L^B(T_{max}) \geq$ L $^8$ ^ for all tasks $T_K$.

a. If $T_K$ has no predecessor, then $L^F(T)$ « $L^B(T_{max})$; otherwise, $L^F(T_K)$ » min{ L^ I $T_y$ *a* T-MM-

We present an intuitively obvious elementary theorem. Theorem. AM tasks $T_E$ with L^^or L^] may be started in parallel as independent tasks if all tasks T with L^T) - L V j + 1 [or $L^F$(Ty) « L^fT^ + 1] have completed execution, without violating the precedence constraints imposed by $G_A(V,E)$.

The tasks $T_x$ e T are first assigned to processors P eP without regard to the resource constraints of R^ e R and then scheduled on them taking into account the resource constraints. If resource constraints are violated, the starting time of the task is delayed untM sufficient amount of resources are released by tasks which have already been scheduled. The tasks in T are scheduled in the decreasing order of thair levels $L^B$(T).

In order to understand how the scheduling procedure works it is convenient to assume that all tasks T with L^T J > I have already been assigned to processors and scheduled on them: Consider the set • of tasks T¥ such that $L^B(T_x)$ « I. Let • - *{J%* , *Tn* , . . . $T_x$. .and define the set J - {1,2,...|+|} so that the elements & J are to I one to one correspondence with tasks in •. Let PI -{1,2... M} represent the set of processor indices to which tasks are to be assigned without regard to the resource requirements. This assignment problem may be formulated as an NP-complete integer linear program (ILP)

[ILP] •iniaize COMP TIME

subject to £j t,j z^COMP TIME, all icPI

$\sum_{i \in PI}$ $z_{ij}(k)$ - 1. all jeJ

z^ « 0 or 1. til icPI and all jeJ

Solution of ILP gives the optimal processor assignment that minimizes the latest finish time COMP TIME of the independent task set 4*. z.»1 if $T_K$ is assigned to processor *9t* and *z^** 0. otherwise. ILP can be soWed*by a general ILP algorithm such as cutting- plana method or branch and bound but such solution procedures arc NP-complete. We solve ILP by transforming it to another problem ILP'.

[ILP'] max $\sum_{i \in PI, j \in J} c_{ij}\, y_{ij}$

subject to $\sum_{j \in J}$ $y_{ij} \leq b_i$ for all iePI

$\sum_{i \in PI}$ $y_{ij}(k)$ = 1, all jeJ

$y_{ij}$ * 0 or 1, all icPI & all jcJ

$ILP^1$ is known as the Maximum Weighted Matching problem, y^'s have the same interpretation as $z_{ij}$'s. c Js and $b_i$'s are defined by me algorithm Assign Tasks. Solution to (ILP') yields an upper bound, UB, for the solution to (ILP). The inequality constraints and the objective function of »$LP^f$ are modified to improve UB and bring it closer to the solution of ILP.

Algorithm Airekin Tasks:

```
1. Initialize : b^M. for all iePI.  c_ij=(
   £M  tu)/ tt|_   for all icPI and jcJ.

2. Solve (ILP').

3. tp, • Jj  t1jyir for til iePI
   i* • {x| tpK ^ tpt for all iePI}
   if (MTC is TRUE) Go to step 6.
   if «ax  (tp, >_ UB) Go to step 5.

4. b1 = |J|   for all iePI
   b1•= (Σj∈J  yi•j) - 1
   zij=yij    for all iePI, j e J
   UB « tp/ 4  Go to step 2.

6. b1 = |J|    iePI
   b1•= (Σj∈J  yi•j) - 1
   MTC • TRUE
   cij={(Σi∈PI  tij)}/tij/Σj∈J tij zij. all iePI and jeJ.
   Go to step 2.
```

6- ˉ'ˉ (tp,* ≥ US) go to step 7.

    $z_{ij}$ = $Y_{ij}$ for a11 iePI. jeJ

    UB$_f$ « tp$_f$*

    Go to step 5.

7- *Pi • *fa* *U $^z$U for all iePI

    $b_i$ • Σ $z_{iJ}$ for all 1ePI.

    COMP TINE -f tPi•

JUIIA/Y aceinnaH task* •#* *%****• ^^^*^*m* we form the following 2 sets

^.*Hz*^ (output of algorithm assign task) is 1, task $T_{\&}$ has been assigned to processor $P_i$. For each jePI, form a set $J_i$ = {j z$_{ij}$ = 1} and a set ,- f T J, z - . for i = 1..,¥J -1} Set *$_i$ is a l * of taskTkhat have been assigned to processor *P*$_i$ in increasing order of their total resource requirements.

*2.* For each set $J_{jt}$ iePI, tp$_i$ (output cf algorithm assign task) represents the total time taken on processor $P_i$ assuming ail tasks assigned to it could be executed on it in succession without violation of resource constraints. Form a set TP « {tPj # 0. for all icPlJtp$_i$ ≤ tp$_u$,}. Note that the set TP may have fewer than n elements.

The tasks on a processor are scheduled in the increasing order of their total resource requirements. Let *V^$_r$* .. r*$_L$] represent the total resource requirement of all tasks $T_y$ such that $L^B$(Ty)-l +1 or *$L^F$(TJ* -l and task $T_y$ has been scheduled . The remaining tasks are scheduled by the following algorithm:

<u>Algorithm Schedule Tasks</u>

```
1. While TP is not empty perform the step
    a. Lets » 1* max {τ(T_x) • t_{p_x} JL^B(T_x) = 1*1}
    b. For each v, 1 ≤ v ≤ L, let r/ * £ r^ where x is such
       that L^B(T_x)»U1 or L^8^-! and T_x has been
       scheduled.
    c. Let tp( be the first element in TP. While ¥ is not empty
       perform the step
        i. let T_x be the first elementin*_i. *
        it. if for each v. 1<v<L, r_y' • r^<fiirj then let
            r(T_x) * s, for each v, let r/ J r/ • r^ and
            remive T_x from •._i Task T_ is scheduled on
            processor P_i and X_ta^j « 1 for k«
            s,s + 1,...,s + t_{x_j}.
    d. Remove tp_i from TP.
2. Form a set <* of all tasks T_y which have not been scheduled
   and have L^ )_y = L
a Repeat this step until + is empty. If for each v, 1≤v≤L, r_y*
   + r_vy ≤)5(r^ and if for some i.t^^idte time of processor p„
   then schedule T_y on P_j and remove T_y from <*. Else remove
   T_y from^.
```

Each time a task which has been assigned cannot be scheduled because of insufficient resource, an entry is mad** in a Resource Shortage Table indicating the particular task which could not be scheduled together with units of the particular resource/resources which were needed but were not available. Once all tasks $T_x$ with $L^B(T_X)$ * I have been scheduled, tasks $T_y$ wlth $L^B$Oy » 1-1 are assigned to processors and then scheduled. The steps are repeated until ail tasks $T_z$ with L $^8$ ^ » 1 have been scheduled. The specific details and the rules for breaking ties while forming the different sets are described in [43].