MODELLING AND SCHEDULING OF COMPUTER PROGRAMS FOR PARALLEL PROCESSING SYSTEMS +

J. L. Baer* and E. C. Russell*

Computer programs which involve large amounts of computation time are targets for analysis to determine two factors; (1) the portions of the program which are most time-consuming, and (2) the effect upon solution time of multiprocessor systems [1].

In our automatic modelling system, analysis begins with a working source program written in a language such as FORTRAN. From this source program a graphical representation is prepared. The initial graph represents the program just as it might be processed on a single processor. In this model the flow of control, as explicitly stated by the programmer, is separated from the flow implicit in the language. Data-dependency is also recorded.

From this initial model, a 'parallel' graph is constructed which preserves the computational determinancy [5] of the program. This graph is prepared by examining the sequentiality imposed by the language and by the nature of cycles (loops). Whenever possible such sequentiality is eliminated thus yielding a highly parallel statement of the program [4].

Figure 1 illustrates the graph-model of programs prepared by the process. In this particular example individual vertices represent FORTRAN statements. The arcs which connect the vertices represent control-flow, either, explicit or data-dependent. For multiple out-branching or in-branching, logic is specified.

Each vertex has either conjunctive ("AND") or disjunctive ("Exclusive OR") logic associated with its input and output.

*University of California at Los Angeles Department of Engineering

+This work was sponsored by the Office of Naval Research, the U.S. Atomic Energy Commission, and the Advanced Research Project Project Agency. Reproduction in whole or in part is permitted for any purpose of the U.S. Government.

In order to prepare for an evaluation of the execution of such a program, several measurements are obtained. Since we assume a working sequential program as our source, we can make measurements on the execution of that program on a single processor [2] to aid in evaluating the effect of multiprocessors. Measurements of particular value are the probabilities of executing each portion of the program and the number of iterations in each cycle. These can be measured as a single parameter, the <u>frequency</u> of execution of each portion of the program. Another measurement of significance is the expected execution time of each portion of the program. This is obtained by tallying the instructions actually generated by a FORTRAN compiler and then introducing the hardware timing considerations at the machine instruction level.

After these preliminary measurements are obtained, two paths of investigation may be pursued. First, the execution of the program on a single processor is evaluated by determining which portions of the program constitute the major time delays. Second, the effect of multiprocessing may be evaluated by a "one-shot" assignment and sequencing process. The overall process is depicted in Figure 2.

The one-shot scheduling algorithm works on an acyclic graph obtained by removing all feedback arcs (e.g. the arc (w_9,w_5) in Figure 1) from the graph model. In order to have a temporally equivalent structure, the time t_i' associated with a vertex w_i is multiplied by its activity a_i, that is the expected number of times that w_i will be executed after it has been reached for the first time, giving a time attribute t_i. Now a weight $p_i t_i$ (where p_i is the probability of ever reaching w_i) is associated with each vertex w_i (note that $p_i a_i$ is the measurable frequency noted above). We say that:

- A vertex is <u>candidate</u> for scheduling if all of its immediate predecessors have been scheduled.

- A machine P_i is available if its associated clock C_i has value 0.
- All candidates are in a queue, ranked by their urgency numbers.
- When a machine is not available, it means that there is at least one vertex running on it. The set of all vertices currently running on a machine P_i forms the set of temporarily assigned vertices on P_i.

The following scheduling rules are used when P_i becomes available and the queue is not empty:

Scheduling rule # 1: Select the first candidate in the queue and temporarily assign it to P_i.

Scheduling rule # 2: Find the first candidate in queue, if any, which is mutually exclusive with all temporarily assigned vertices and repeat the process until there is no success.

The step by step algorithm is then as follows:

1) Initialization: all machines P_i are available ($C_i = 0$). TIME = 0.
2) Initial vertex w_1 temporarily assigned to P_1 and set $C_1 = P_1 t_1$.
3) If any $C_i \neq 0$ then let $\tau = \min_i(C_i)$ for $C_i \neq 0$, else $\tau = 0$. All P_i such that $C_i = 0$ or $C_i = \tau$ become available. TIME = TIME + τ. $C_i = \max(C_i - \tau, 0)$.
4) Assign and sequence all temporarily assigned vertices on the available machines. Let W1 be the set of those scheduled vertices at this step.
5) If all vertices have been scheduled go to step 10.
6) Determine the set of new candidates from the immediate successors of the elements of W1. Let W2 be their set.
7) If W2 = $\emptyset$ go to step 3 else merge the elements of W2 in the queue.
8) If there is no machine available or the queue is empty, go to step 3.
9) Let the first machine available be P_i. Apply scheduling rules #1 and #2 until the latter fails; $C_i = \Sigma\, p_j t_j$ where w_j is a vertex temporarily assigned to P_i. Go to step 3.
10) End of the scheduling process. TIME gives an (optimistic) estimate of the mean path length of the process.

Urgency: when a vertex joins the queue, it is ranked according to its urgency number. The following policies were tested:

1) The vertices are placed in the queue in the order they arrive: 'FIFO".
2) Largest weight first : "L.W.F.'.
3) Smallest weight first : "S.W.F.'
4) Largest number of immediate successors first : 'IANTE'.
5) Largest number of successors first : 'NANTE'.
6) The vertex such that $u(i) = p_i t_i + \Sigma\, p_j t_j$ is maximum is placed first (where w_j is an immediate successor of w_i) : 'GLOBAL1'
7) The vertex such that $u(i) = (\max(u_j) + p_i t_i)$ is maximum is placed first (where the u(i) are computed in starting by the terminal vertex) : 'GLOBAL2'.

Some variations such as vertices preassigned to particular processors are also embedded in the algorithm. Extensions to non-homogeneous parallel processors could be made by introducing a vector time attribute instead of a scalar one. Results of experiments on example graphs drawn from [3] are shown in Figures 3, 4, 5. The path lengths are normalized to $\frac{PL_{uni}}{NPROC}$, where PL_{uni} is the mean path length on one processor (namely $PL_{uni} = \Sigma\, p_i t_i$ for all i) [3] and NPROC is the number of machines for which the scheduling is performed. The ordinate called DEGRADATIONS is equal to

$$\frac{PL - \frac{PL_{uni}}{NPROC}}{\frac{PLuni}{NPROC}} \times 100.$$

The DYNAM curves correspond to a dynamic scheduling on sampled paths. As can be seen from the examples:

- The GLOBAL1 (and GLOBAL2) urgencies always behave best in 'a priori' scheduling.
- In general a pure dynamic scheduling is not as good as an 'a priori' scheduling with appropri-

ate urgencies. (The exception of Fig. 5 is due to a very symmetric graph).

REFERENCES

1. Martin,D.F. and G.Estrin. "Experiments on Models of computations and systems". Transactions of IEEE vol. EC-16 no. 1, pp. 59-69, Feb. 1967.

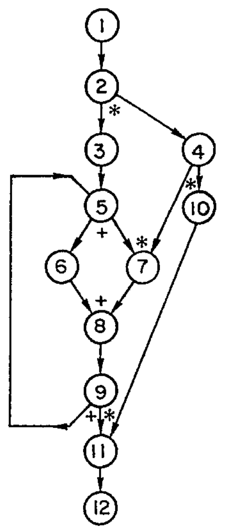
2. Estrin,G. et al "SNUPER Computer. A Computer Instrumentation Automaton". Proceedings of the SJCC, pp. 647-656, 1967.

3. Martin,D.F. "The automatic assignment and sequencing of computations on parallel processor systems." Ph.D. in Engineering, UCLA, Jan. 1966.

4. Bernstein,A.J. "Analysis of programs for parallel processing " IEEE Transactions in Electronic Computers, vol. EC-15, no. 5, pp. 757-767, Oct. 1966.

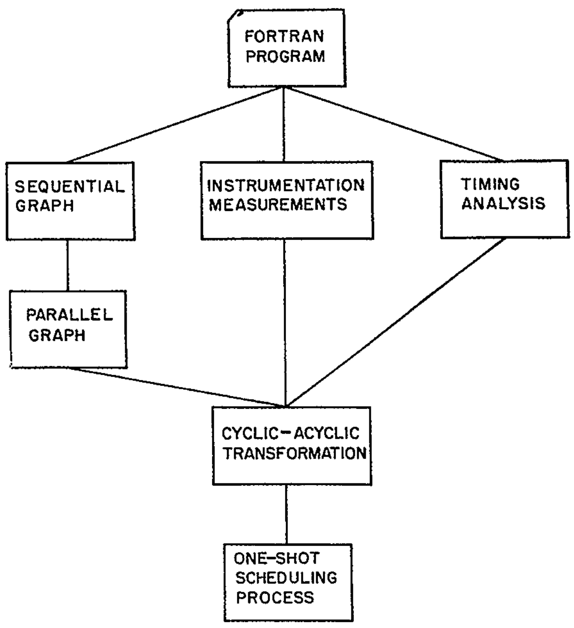
5. Karp,R.M. and R.E.Miller "Properties of a model for parallel computations: determinancy, termination queueing." SIAM Journal on Applied Mathematics, Vol. 14, no. 6, pp. 1390-1411, Nov. 1966.

```
1    SUBROUTINE CALC (A, H)
2    B = A + 1.
3    C = B + 1.
4    D = B + 2.
10   G = D + 3.
5    IF ( C ) 6, 6, 7
6    E = C + 2.
     GO TO 8
7    E + D + 1.
8    F = E + 1.
9    IF ( F ) 5, 5, 11
11   H = G - F + 7.
12   RETURN  .
     END
```



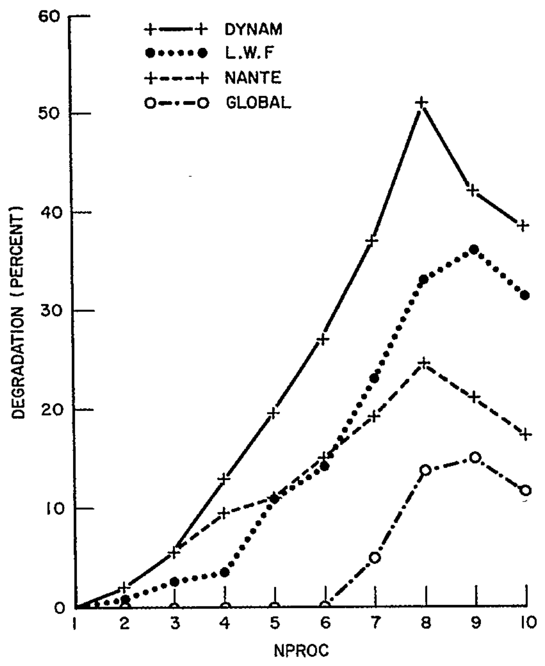
* AND logic
+ OR Logic

FORTRAN PROGRAM AND "PARALLEL" GRAPH REPRESENTATION
FIGURE 1


FORTRAN PROGRAM
SEQUENTIAL GRAPH
INSTRUMENTATION MEASUREMENTS
TIMING ANALYSIS
PARALLEL GRAPH
CYCLIC-ACYCLIC TRANSFORMATION
ONE-SHOT SCHEDULING PROCESS

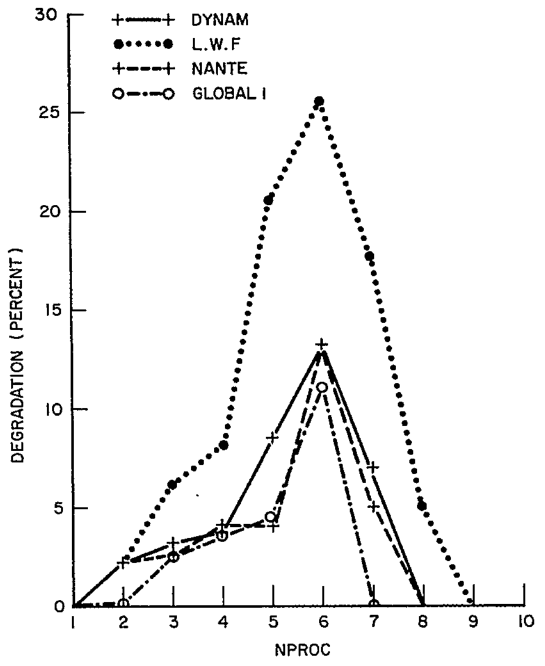
OVERALL MODELING AND SCHEDULING PROCESS

FIGURE 2



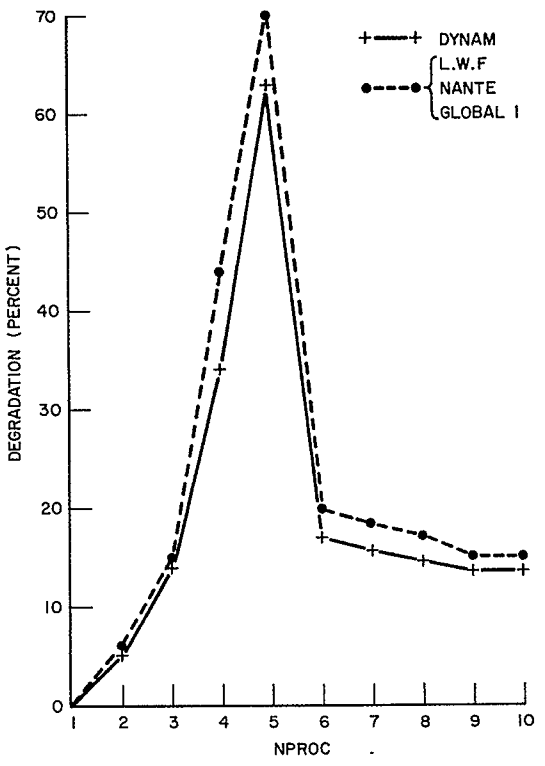
EXAMPLE OF GRAPH NO. 2 (147 VERTICES)
SUBROUTINE OF A NUMERICAL WEATHER PREDICTION PROGRAM

FIGURE 4



EXAMPLE GRAPH NO. 1 (32 VERTICES)
SUBROUTINE OF A NUMERICAL WEATHER PREDICTION PROGRAM

FIGURE 3



EXAMPLE GRAPH NO. 3 (223 VERTICES)
X-RAY CRYSTOLLOGRAPHY PROGRAM

FIGURE 5