International Journal of
Science and Engineering Investigations vol. 5, issue 59, December 2016
ISSN: 2251-8843

Applying Program Evaluation and Review Technique to Parallel Computing

Janusz Kowalik[1], Piotr Arłukowicz[2]
[1](ret) The Boeing Company, USA
[2]The University of Gdańsk
([1]j.kowalik@comcast.net, [2]piotao@inf.ug.edu.pl)

Abstract- Using Amdahl's Law it is possible to estimate the upper bound for parallel speedup. In deriving the Law it is assumed that the considered application can use limitless number of processors so that the processing time of the code parallel fraction can be reduced to 0. In real applications it is more useful to calculate the shortest parallel computing time and the required number of processors. This can be accomplished by using the ideas related to the methods of Program Evaluation and Review Technique and critical path in directed acyclic graphs representing parallel algorithms. We can calculate the number of processors needed to process parallel code in the shortest time. We call this number the critical number of processors.

Keywords- *PERT, Parallel processing,Ccritical path*

I. INTRODUCTION

The first researcher who considered the issue of parallel speedup limits was Gene Amdahl. He assumed that program has two fixed components: parallel component and sequential component. Processing sequentially parallel component takes fraction p of the entire code time and processing sequential component takes s=1-p fraction of time. The sequential computing time is p+s=1 and parallel computation takes s+p/N time where N is the number of processors used to compute the parallel fraction. Hence we get the following speedup limit:

$$T_1 = s + p = 1$$
$$S_p = \frac{1}{s + p / N} \qquad (1)$$
$$\lim S_{PN \to \infty} S_P = s^{-1}$$

The inverse of the sequential fraction is the upper bound of the achievable speedup. This upper bound is optimistic because we assume that the parallel fraction time can be reduced to zero. This result has been called the Amdahl's Law. Couple of decades later John Gustafson observed that for real life large problems solved by large computers parallel fraction increases in increasing problem sizes relative to the sequential fraction. This observation led him to very different results related to speedup.

$$T_1 = s + pN^{\alpha}$$
$$\alpha \geq 0$$
$$S_p = \frac{s + pN^{\alpha}}{s + pN^{\alpha} / N} = \frac{s + pN^{\alpha}}{s + pN^{\alpha-1}} \qquad (2)$$
$$\lim_{s \to 0} S_p \to N$$

Gustafson results have been confirmed experimentally and indicate that for large scale systems solved using supercomputers speedup converges to N and efficiency to 100 %. Both Amdahl's Law and Gustafson's result are limits. In practical application of parallel processing we are also interested in finding the minimum parallel processing time and the number of processors that are required for achieving the shortest parallel processing time.

For accomplishing these objectives we can use a classic technique used to manage large complex projects.

In the 1950s Remington Rand, Booz, Allen and Hamilton and Lockheed Missile Systems Division developed technique called PERT (Program Evaluation and Review Technique) [1-4] for the Special Projects Office of the Department of the US Navy. The motivation was helping the research and development of the US Navy InterContinental Ballistic Missile (ICBM) Polaris.

The main capability of the method was to serve as a tool for managing complex projects involving hundreds or thousands of interrelated tasks. In the mid of 1960s PERT was implemented on the British computer Elliott 803B and used in the civilian Gdansk Shipyard in Poland [5]. The applications of PERT in the Polaris project and at the commercial shipyard in Gdansk were successful. The project's completion times were reduced by many months. A unique application of PERT was for managing the preparation for the Winter Olympics in Grenoble in 1968. In this application scientific technique of PERT proved to be useful for large nonindustrial projects. In PERT project is partitioned into major tasks represented by directed acyclic graph (DAG). A small example of DAG is shown in Fig.1

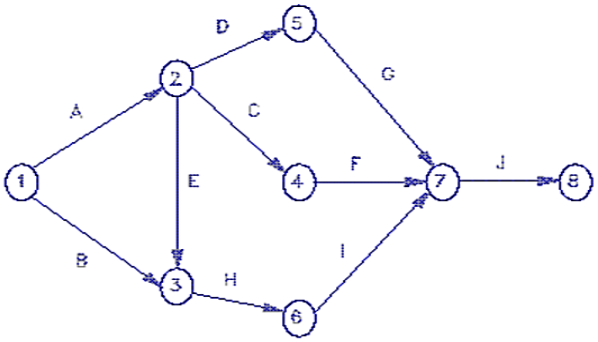

Figure 1. Directed acyclic graph example

In directed acyclic graph edges (arrows) represent tasks to be performed. The project graph specifies scheduling sequence and duration of tasks. The project is completed when all tasks are finished. Each task has duration time. In some PERT versions it is possible to specify three duration times for every task. The optimistic shortest time, the pessimistic longest time and the most likely time. One of the most useful concepts is the critical path of a given graph. The critical path is the longest path from the beginning (in our example node 1) to the end (node 8). For the sake of discussion let us assume that for our example this path is from node1 via nodes 2, 4, 7 to node 8.

In managing projects the critical path is important because any delay along this path will impact adversely the completion of the project. On the other hand if we desire to shorten the project we must reduce the critical path time. Other paths from node 1 to 8 have some time slack. The critical path has no slack. The total length of the critical path is the shortest time for completing the project represented by the graph. We can summarize advantages and disadvantages of PERT.

Among advantages are:

- better understanding of scheduling tasks and their relationships,
- identification of tasks that cannot be delayed,
- estimation of the project completion date

Disadvantages include:

- large graphs are hard to display or print
- It is difficult to provide time duration for uncertain tasks.

II. USING PERT CONCEPTS FOR PARALLEL COMPUTING

A significant application of the PERT techniques was hinted by Richard Brent from The Australian National University. Brent published in 1974 a paper [6] on parallel evaluation of arithmetic expression and pointed out that parallel algorithms could be represented by directed acyclic graphs. Parallel algorithms are projects and some techniques of PERT apply to them. Brent did not mention PERT but used a related concept of the critical path developed in the same period of time.

If a DAG represents parallel algorithm every directed edge is a computing task executed serially by single processor. Different task can be executed in parallel if they are independent but every task is executed by a single processor. Furthermore we assume every task is ready to execute as soon as all connected predecessor tasks have been done. We also assume that it is possible to estimate execution time for every task and that these times cannot be reduced. Given this information the sequential computing time is the sum of all tasks times in the graph. By definition the critical path is the longest path in the graph from its beginning to the end. Since each task of the critical path has to be executed serially the execution time of the critical path cannot be shortened.

Thus the critical path execution time is the shortest possible parallel execution of the entire DAG representing some parallel algorithm.

The sequential processing time is the longest time. Adding more processors the parallel computing time shrinks. Initially the total parallel time is larger than the critical path time because there are not enough processors. We are interested to find the minimum number of processors for processing graph in the shortest possible time that is the critical path time. This minimal number of processors will be called the critical number of processors.

To illuminate the meaning and the calculation of critical number of processors consider DAG in Fig.2.

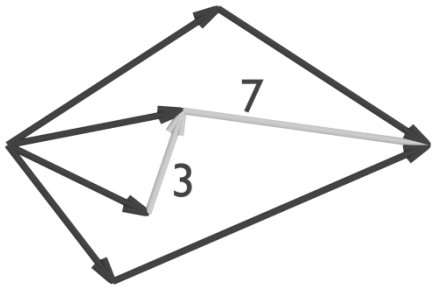

Figure 2. Simple DAG of parallel computation. All tasks execution times are 5 except two tasks as shown

For this graph the sequential time is 40. The critical path time is 15. For two processors the parallel time is 25. For three processors we get 15. This means that for three processors the parallel time is the shortest possible processing time. Further increase of processors would not reduce the computation since 15 is the critical path time.

By definition speedup is the ratio of the sequential processing time to the parallel processing time. The shortest time for parallel computing is the time required for computing tasks along the critical path. This can be accomplished if the number of processors working in parallel is unlimited or putting it differently if there is sufficient number of processors

for executing all tasks in the parallel processing DAG graph within the critical path time.

If there is no sufficient number of processors they will not be able to process all tasks within the critical path time. In this case the parallel time is longer and speedup worse than optimal. This can be summarized by following inequalities

$$S_p \leq \frac{T_{Seq}}{T_{CP}} for p< p_{CR}\ T_p > T_{CR} \quad (3)$$

The Fig 3 illustrates the relationship of processing time on the used number of processors.

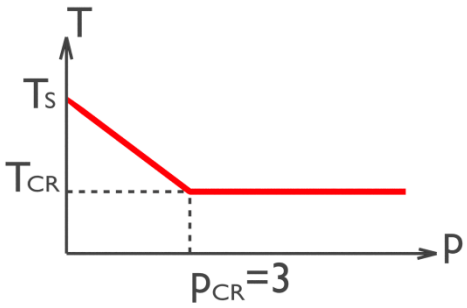

Figure 3. Parallel times. T – time, p – number of processors, critical number of processors p_{CR}=3, T_S – sequential time

What remains to be stated is the formula for evaluating the critical number of processors.

This formula is:

$$p_{CR} = \text{smallest\_integer} \geq \frac{T_{SEQ}}{T_{CP}} \quad (4)$$

In the considered case the smallest integer greater than 40/15 is 3.

III. Simple Linear Algebra Example

To illustrate specific parallel computation we consider computing z=pTAp where p is a vector and A is a matrix.

Firstly we calculate vector v=Ap and then the dot product of v and p. The calculation of v will be done in parallel. For large matrix A calculation of dot product pv is relatively very short and can be ignored. Fig. 4 shows the DAG representation.

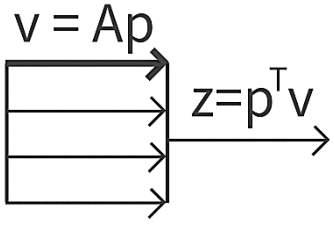

Figure 4. The DAG for $z=p^TAp$

Assumptions:

a) Matrix A is 1000x1000

b) A is sliced by rows 400, 200, 200, 200. One slice is larger than others to get the unique critical path.

c) Each slice is processed by a single processor.

d) The time to compute dot products of 200 rows and vector p is called T.

In this simple example the sequential time is 5T and the critical path time is 2T. The upper bound on parallel speedup is 5/2=2. 5

If there are only two processors (p=2) the parallel time is 2T+T=3T and the speedup is 5/3=1.66. For p=3 processors the parallel time is 2T, the same as the critical path time. Adding more processors could not shortened the completion time. Other examples of parallel speedup can be found in [7].

IV. Handling Loops

The above analysis of parallel computing applies to algorithms represented by DAGs. Very often algorithm tasks are loops. If DAG contains tasks that are loops the corresponding paths can be reduced by applying fork-join parallelism (the OpenMP style, Fig. 5).

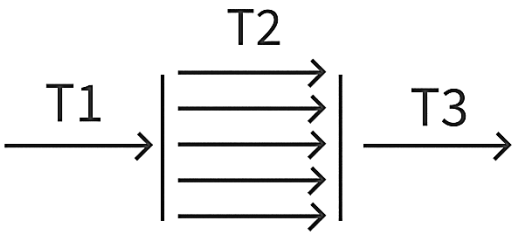

Figure 5. Fork-join parallelism

For example if a path contains three tasks and the second task is a loop the execution time can be reduced from TSEQ=T1+T2+T3 to Tp=T1+T2/m+T3, where m is the number of used threads. The speedup is

$$S = \frac{T_1+T_2+T_3}{T_1+\frac{T_2}{m}+T_3} \quad (5)$$

In reality the speedup will be lower than calculated from the formula for S due to the OpenMP overhead.

References

[1] B. Ralph Stauber, H. M. Douty, Willard Fazar, Richard H. Jordan, William Weinfeld and Allen D. Manvel. Federal Statistical Activities. The American Statistician 13(2): 9-12 (Apr., 1959), pp. 9-12

[2] Malcolm, D. G, J. H. Roseboom, C. E. Clark, W. Fazar Application of a Technique for Research and Development Program Evaluation

OPERATIONS RESEARCH Vol. 7, No. 5, September–October 1959, pp. 646–669

[3] Klastorin, Ted (2003). Project Management: Tools and Trade-offs (3rd ed.). Wiley. ISBN 978-0-471-41384-4

[4] Project Management Institute (2013). A Guide to the Project Management Body of Knowledge (5th ed.). Project Management Institute. ISBN 978-1-935589-67-9.

[5] O. Pawlowski, M. Brewka,W. Majewski and J. Kowalik, "Pert Cpa, Cpm Task Networks and their analysis" (in Polish), Wydawnictwo Morskie, 1967.

[6] R. P. Brent, "The Parallel Evaluation of General Arithmetic Expressions", Journal of the ACM, 1974.

[7] Arch D. Robinson, Michael McCool and James Reinders "Structured Parallel Programming: Patterns for Efficient Computation", Elsevier Morgan Kaufmann, 2012.