J.King Saud Univ., Vol.9, *Computer & Information Sci.*, pp. 1-20 (A.H 1417/1997)

A Simulation Approach to Evaluate the Time Cost of Parallel Computations

Bin Qin *, Reda A. Ammar and Howard Sholl **

** IBM Canada Limited, 3T/640/1150/TOR, 1150 Eglinton Avenue East, North York, Ontario M3C 1H7, Canada*

***U-155,Computer Science & Engineering Dept., The University of Connecticut Storrs, CT 06269-3155, USA*

(Received, 05 June 1993; accepted for publication, 03 July 1995)

Abstract. In this paper, the execution time cost of a parallel computation in a shared memory environment is defined as a function of the input, the algorithm, the data structure, the processor speed, the number of processors, the processing power allocation and the communication between processors. The computation structure model is modified to describe the impact of these seven factors on the computation's execution time cost. A simulation technique is developed to analyze and evaluate the computation's execution time. A software tool, TCSS (Time Cost Simulation System) that implements the simulation technique is also described.

Key words: Performance modeling; simulation; Parallel computations; Time cost analysis.

1. Introduction

Execution time cost is one of the important metrics to measure the quality of software designs. Research work in the area of time cost analysis has resulted in many models and tools [2-9, 11-14, 16,17]. Most of the work done in this area has been concentrated on time cost analysis of sequential computations. However, as parallel computing is

widely used in applications where time critical requirements exists, there is a necessity to develop models, techniques and tools to investigate the execution time cost of parallel computations [10,12].

The execution time cost of parallel computations depends on many factors [12] such as:

1. The size of the input to the computation. If the computation is used to sort a table of n elements, the execution time cost of the computation increases as the size of the table increases.

2. The algorithm used in the computation. To sort a table, we may use a bubble sort algorithm or a quick sort algorithm. Using different algorithms will yield different execution time costs.

3. The data structure used in the computation. A string can be represented as an array or a link list. Different representations need different processing and therefore, they impact the execution time of the computation.

4. The processor speed. In general, as the speed of the processor increases, the execution time cost of the computation decreases.

5. The number of processors available. In general, as the number of processors increases, we may achieve lower execution time cost.

6. The processing power allocation policy [12]. Each processor has a processing power equal 1. If the number of processors is 5, we say that the processing power is 5. The processing power allocation policy determines how much processing power is allocated to each computation. If a computation is allocated more processing power, it may have lower execution time cost. If a computation is allocated a processing power smaller than 1, it means that another computation will be executed by the same processor.

7. The communications between computations. If two computations need to communicate with each other in order to fulfill a task, the way the communication is carried out and the execution time spent on the communication will influence the execution time costs of both computations.

Often, the execution time cost of a computation is only defined as a function of the first four factors. To better analyze the execution time of parallel computations, we need to model the impact of other factors on the execution time cost. In this paper, we develop a modeling technique and a simulation tool TCSS (Time Cost Simulation System) to study the execution time cost of parallel computations as a function of the above seven factors. We assume that the underlying execution environment is CREW MIMD [10] with a limited number of processors such as the Sequent machine or the Sparc 2000. All the processors in the environment have unity processing power and they communicate with each other through a shared memory that allows concurrent read but does not allow concurrent write (CREW MIMD model [10]).

This paper is organized as follows. The execution time cost modeling in the TCSS system is first discussed in Section 2. The general structure of the TCSS system and how it works are presented in Section 3. The simulation approach is given in Section 4. An example is used in Section 5 to illustrate the use of the TCSS system. Concluding remarks are finally given in Section 6.

2. Modeling of the Time Cost

Our work is based on the computation structure model [12, 13] that describes the detailed execution time of computations. This model has been successfully used as the underlying models in several time cost analysis systems [2-7, 17]. In this section, we briefly describe this model. More details about the model can be found in [12, 13].

A computation in the computation structure model is represented as two directed graphs, a control graph and a data graph. The control graph shows the order in which operations are performed, while the data graph shows the relations between operations and data.

The control graph contains a start node, an end node, operation nodes, decision nodes, or nodes, fork nodes and join nodes. The start and end nodes indicate the beginning and end of the computation; an operation node specifies an operation to be performed; a decision node checks conditions at a branch point; an or node serves as a junction point; a fork node creates parallel execution paths; and a join node merges parallel execution paths into a single path. The execution of a computation is triggered

when an activation signal enters the control graph of the computation. When a signal enters an operation node, the operation specified by the node is performed and the signal then leaves the node. When a signal arrives at a decision node, the decision node checks some conditions and the signal leaves the node from one of its outgoing edges depending on the result of checking. When a signal arrives at an or node from one of its incoming edges, it immediately leaves the node from its outgoing edge. When a signal reaches a fork node, the fork node creates parallel execution paths and an activation signal is put on each parallel path. When a join node receives one signal from each of its incoming edges, an activation signal is created and it leaves the join node from its outgoing edge. When an activation signal finally arrives at the end node, the execution terminates.

The data graph contains operation nodes and data nodes. The former is represented as a circle and the latter as a box. A data item is the input to an operation if there is an edge from the data node to the operation node. Similarly, a data item is the output of an operation if there is an edge from the operation node to the data node.

Figure 1 illustrates the computation structure of a sequential computation. The control graph is given in Fig. 1.a and the data graph is given in Fig. 1.b. The weight of each edge in the control graph indicates the number of times the edge is traversed by activation signals. This computation copies a string t with a length of n to string s. Operations in the computation are defined as follows:

$$op_1 : i \leftarrow 1$$
$$op_2 : s\ [i\ ] \leftarrow t\ [i\ ]$$
$$op_3 : i \leftarrow i + 1$$
$$D : i \leq n$$

The execution time of a computation is defined as follows. Each node in the control graph is associated with execution time which is equal to the time needed to perform the operations specified by the node. When a node with execution time cost C receives all the required activation signals, the execution of the specified operations are started and after C amount of time, activation signals leave the node. The execution time cost of a computation is then defined as the time for an activation signal to travel from the start node to the end node.

The computation structure model, described above, only defines the execution time cost as a function of the input, the algorithm, the data structure and the processor speed. To model the impact of the processing power, allocation policy and the

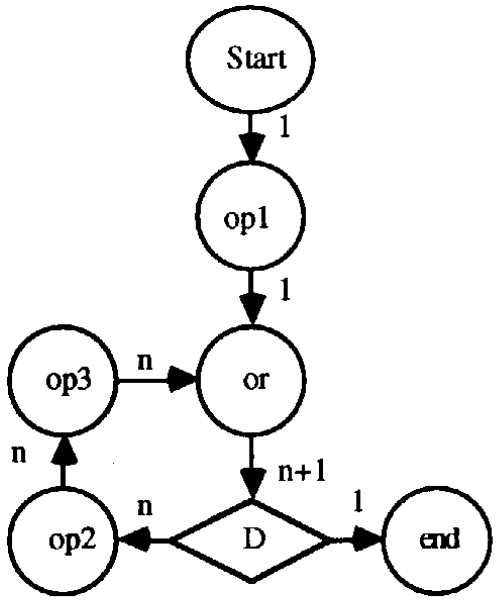

Fig. 1.a

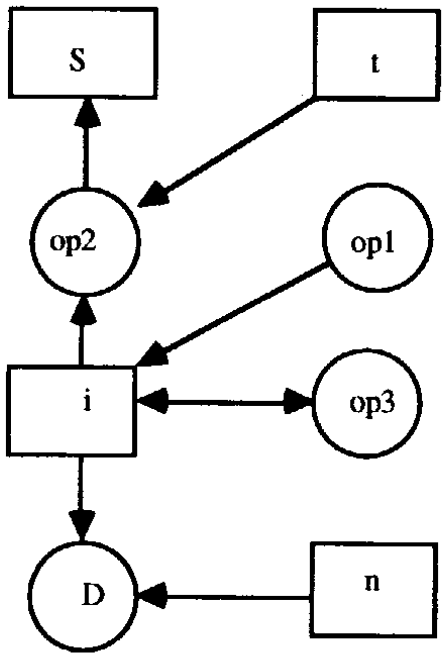

Fig. 1.b

Fig. 1. Example of a sequential computation.

communication on the execution time, we modify the computation structure model. In the modified model, it is assumed that the underlying execution environment has a limited number of processors, all the processors in the environment have unity processing power and they communicate with each other through a shared memory (CREW MIMD model [10]).

From the above description, it is clear that all factors except the processing power and the allocation policy are implicitly represented by the computation structure model. To model the impact of processors and processing power allocation on the execution time, we assume activation signals have the abilities to carry processing power and allocation policy as they travel in the control graph of a computation. Given a computation C , we define its execution time cost as $\text{Cost}_C(P, A)$ if P processing power is provided to C and P is allocated using the processing power allocation policy A . The only restriction put on P is $P > 0$. $P \geq 1$ means the computation C uses more than unity processing power. $P < 1$ means the processor shares one processor with other computations and C only gets $P * 100\%$ of the processing time. It is assumed that the processing power allocation and release are carried out at the beginning and at the end of a computation's execution respectively. That is, a computation will hold the same amount of processing power during the execution. The amount of processing power carried to the start of a computation by the activation signal is what is allocated to the computation. On the other hand, the allocation policy carried to the start node is the one passed to the computation. When an activation signal reaches a node V, node V receives the processing power and the allocation policy carried by the signal, and it uses them to perform the specified operations. If the amount of processing power and the allocation policy received are P and A, respectively, the execution time cost of node V is then equal to $\text{Cost}_V(P, A)$. As in the original model, the execution time cost of the computation is defined as the time for an activation signal to travel from the start node to the end node of the computation. However, the amount of time an activation signal stays in a node depends on how much processing power the node receives which on turn, depends on the allocation policy.

Suppose the amount of processing power and the allocation policy carried by an activation signal are P and A, respectively. If an activation signal enters a node which is not a fork nor a join node, an activation signal will leave the node with the same amount of processing power and the same allocation policy after the specified operation is performed.

Consider the parallel structure illustrated in Fig. 2 (the weights of an edge in

Fig. 2 represent the amount of the processing power and the allocation policy carried by the activation signal which traverse the edge), the fork node uses the received allocation policy to allocate the received processing power among its n successor nodes. N activation signals are created by the fork node. All the activation signals generated by the fork node carry the same allocation policy A . Node op_i carries P_i processing power. How the allocation is carried out is totally determined by the allocation policy A. The only restrictions put on the allocation are as follows:

$$\sum_{i=1}^{n} P_i = P, \quad 0 < P_i \leq P$$

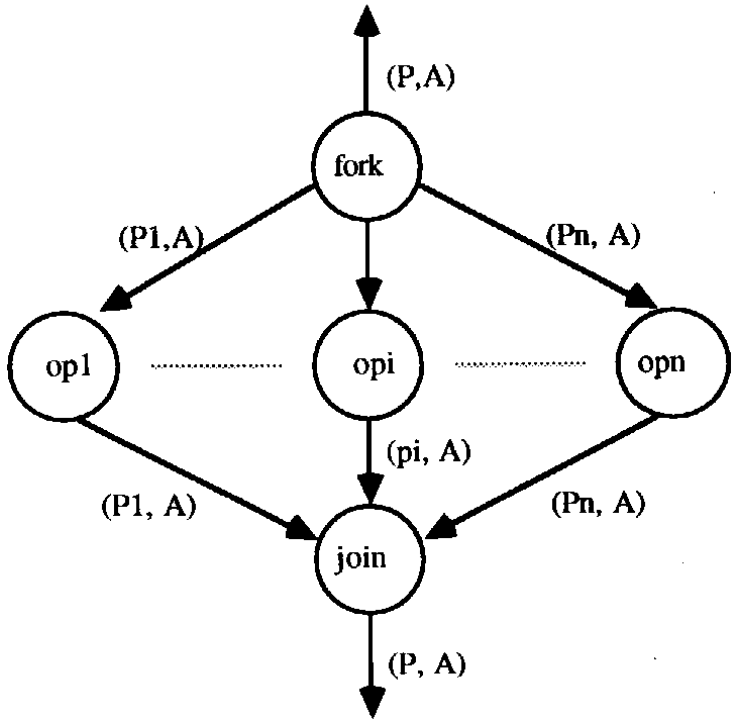

Fig. 2. An example of parallel structure.

For the join node in Fig. 2, when it receives an activation signal from each of its incoming edges, it creates an activation signal. The signal created by the join node carries allocation policy A, and if the activation signal coming from node op_i carries P_i processing power, the activation signal generated by the join node then carries $P = \sum_{i=1}^{n} P_i$ processing power.

The execution time cost of a computation is still defined as the time for an activation signal to travel from the start node to the end node. It depends on the seven factors described above. The first five factors are represented by the structure of the model, whereas the last two (the processing power and the allocation policy) are represented as parameters, the amount of processing power P is not necessary to be an integer number. The only restriction put on P is $P > 0$. $P \geq 1$ means computation C uses more than unity processing power. $P < 1$ means computation C shares one processor with other computations and C can only get P * 100% of the processor time.

For the parallel structure as illustrated in Fig. 2, if the amount of processing power and the allocation policy received by the fork node are P and A, respectively, we define its execution time cost as:

$$Cost_{fork}(P, A) + max\{Cost_{op1}(P_1, A), \ldots, Cost_{opn}(P_n, A)\} + Cost_{join}(P, A)$$

A sequential computation (or a node N which contains no parallel paths) needs a special treatment. It is assumed that the execution of a sequential computation can be continued without waiting if no les than unity processing power is provided during its execution. For sequential computations, we define its base time cost as its execution time cost given that unity processing power is provided. Since such a node doesn't contain any parallel paths, its execution time cost is independent of the processing power allocation. Therefore, the base time cost of a sequential computation C is equal to $Cost_C(1, A')$ where A' is any allocation policy. Given a sequential computation C, suppose P processing time is provided. We define the execution time cost of C as follows:

$$Cost_C(P, A) = \frac{Cost_C(1, A')}{min(1, P)} = \frac{C\text{'s base time cost}}{min(1, P)}$$

where A and A' are any allocation policies.

The above indicates that the base time cost of a sequential computation is the best execution time cost one can get. Providing more than unity processing power to a sequential computation will not reduce its execution time cost, and providing less than unity processing power to such a node will force the computation to wait during its execution, therefore, resulting in an increase of its execution time cost.

To model the communication, we use a locking technique [15] to manipulate the accesses to shared data. It is assumed that each shared data item is associated with two

locks, a read lock and a write lock. To read a shared data item, one needs to obtain a read lock on that shared data time. Similarly, a write lock on a shared data item is required if one wants to perform a write operation. Several operations performed in parallel can read a shared data item at the same time. However, no other read or write lock on a shared data item is granted if a write lock on that shared data item has already been held.

To include the data access control into the computation structure model, two new types of nodes, a lock node and an unlock node, are added to the computation structure model. The lock node is used to obtain locks on shared data and the unlock node is used to release locks. Each lock or unlock node has an incoming edge and an outgoing edge associated with it in the control graph. A lock node needs to obtain a read lock on data item X if X appears as an input to the lock node in the data graph. Similarly, a lock node needs to obtain a write lock on X if X appears as an output from the lock node in the data graph. If X is connected with an unlokc node in the data graph, the unlock node releases a read or write lock on X depending on whether X appears as an input or output of the unlock node. Figure 3 illustrates an example of lock and unlock nodes. In Figure 3, the lock node needs to obtain read locks on X and Z, and a write lock on Y; the unlock node releases write locks on A and C, and a read lock on B.

It is assumed that a lock node obtains all its required locks at the same time. If the lock node can not get all the required locks, it is put into a lock waiting queue. In this way, a lock node will never hold some locks and wait for some other locks. When some locks are released by an unlock node, lock nodes in the lock waiting queue are checked to see if their reqests can be satisfied. The checking is done in an FIFO order. Once all the requests of a lock node are satisfied, the lock node is removed from the queue.

The execution time cost of a computation here is again defined as the time for an activation signal to travel from the start node to the end node. When an activation signal enters an unlock node, the unlock node starts to release locks. After required locks are released, the activation signal leaves the unlock node. The execution time cost of an unlock node is then equal to the time to release the required locks.When an activation signal enters a lock node, the lock node starts to wait for the required locks. When all the locks on required data are available, the lock node manipulates these locks. After all the required locks are obtained, the activation signal leaves the lock node. The execution time cost of a lock node is then equal to the sum of the time to wait for the required locks and the time to manipulate these locks.

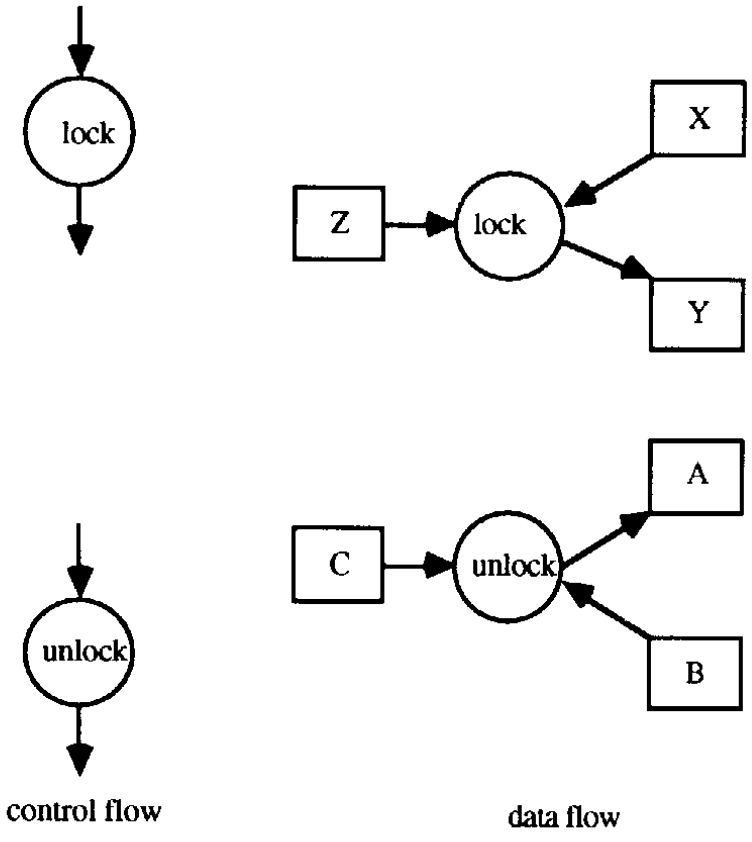

Fig. 3. An example of lock and unlock nodes.

Given the model defined in this section, some uncertainties may occur. This happens when more than one lock node in parallel requires conflict locks on a shared data item (read and write locks, or write and write locks). Consider the parallel structure in Fig. 4. A conflict occurs in the parallel structure since lock1 and lock2 both require write locks on X at the same time.When this occurs, it is assumed that the computer system will arbitrarily choose one lock node to obtain its required locks first (all lock nodes are equally likely). The probability of selecting a lock node to get its required locks first is assumed to be uniformly distributed. The problem of ordering different lock nodes is studied in depth in other works[15]. If any uncertainty occurs, the average execution time cost is used as the execution time cost.

3. The Time Cost Simulation System (TCSS)

To automate the exection time cost analysis of parallel computations, a software tool (TCSS) was designed and implemented . TCSS is an interactive system. It is

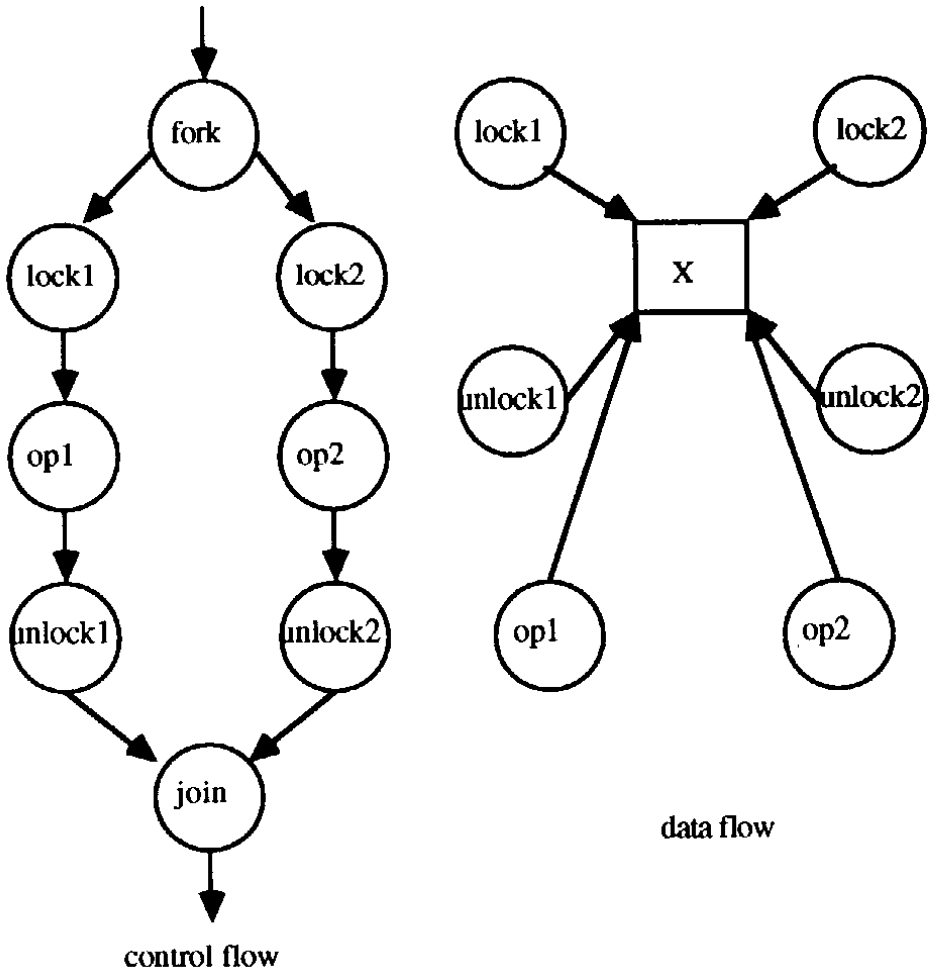

Fig. 4. An example of conflict.

implemented in C and runs under the Unix system. A set of commands are provided in TCSS for users to carry out the execution time cost analysis and evaluation. Generally speaking, what the TCSS system does is to take the user's computation as input and determine its execution time by using the simulation approach. The modified computation structure model discussed in section 2 is used as the underlying model in TCSS. Three parameterss, the number of processors, the processor speed and the allocation policy, are used in TCSS to characterize the property of the environment that supports the execution of the computation to be analyzed. The user can use some TCSS commands to change the characteristics of the execution envionment. By changing the characteristics of the environment, the execution time of the user's computation changes.

Figure 5 illustrates the general structure of the TCSS system. The user interface provides interaction between the TCSS system and users. It interprets user commands and calls corresponding system modules to fulfill the user's task. The user's computation to be analyzed is written in a language supported by TCSS. One of these language is a graphical language that allows a user to build the computation structure model directly [3]. However, a model deriver can be built for each language supported by TCSS. Currently, TCSS supports C and a Pascal-like language. The derived computation structure model contains all the information necessary to carry out the execution time cost analysis. Typically, each node in the model has a base time cost; each edge has a flow count that indicates the number of times the edge is traversed; and if a node is a lock or an unlock node, what and how many locks are to be obtained or released in that node are also specified.

Once the computation structure is derived, a flow analysis technique [2] is used in the flow balance analyzer to determine the number of times each operation in the computation is performed. The computation with flows analyzed is then sent to the execution time cost simulator which uses the simulation technique to determine the execution time cost of the computation. The execution time cost obtained from simulation is then used by the execution time cost evaluator for further time cost analysis.

The execution time cost evaluator contains a number of packages to calculate different execution time statistics of the user's computation. It can determine the minimum time cost, the maximum time cost, the average time cost and the time cost variance. It can also graphically represent the time cost against different performance paremeters and the time cost distribution.

4. Time Cost Analysis Through Simulation

The execution time cost simulator uses the simulation technique to determine the execution time cost of the computation to be investigated. What the execution time cost simulator does is to simulate the traveling of the activation signal in the computation to be investigated using the modified computation structure model described in Section 2. During the simulation, an activation signal carries three attributes, execution time cost attribute, processing power attribute and allocation policy attribute. Locks of all the variables in the computation are initialized before simulation. The simulation process starts when an activation signal enters the start node of the computation. The execution

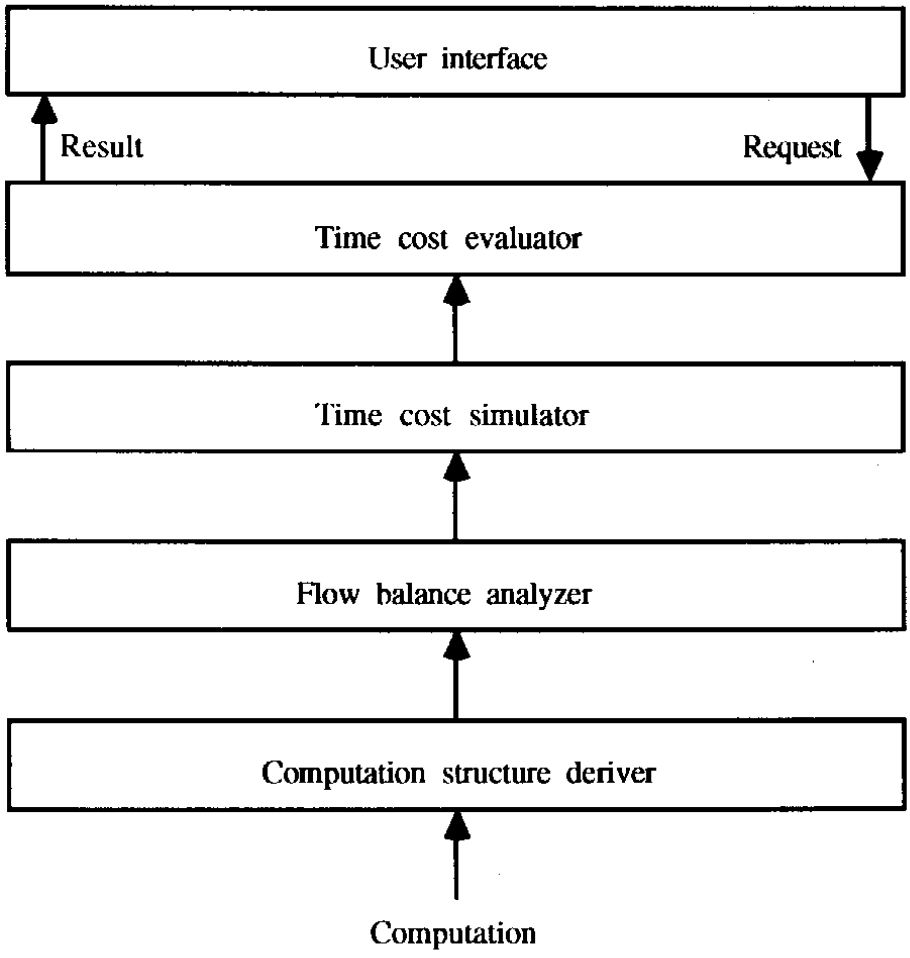

Fig. 5. General structure of TCAS system.

time cost attribute of this activation signal is initialized to zero. The processing power and allocation policy attributes of the signal are initialized according to the property of the execution environment in which the user's computation resides.

When an activation S carrying processing power P and allocation policy A arrives at a node N that has a base time cost T, $\frac{T}{min(1,P)}$ is added to the execution time cost attribute of signal S.

If node N is a decision node, a random number generator is used to determine which outgoing edge signal S should take.

If node N is a fork node, the execution time cost simulator creates one activation

signal for each parallel path.The execution time cost and allocation policy attributes of signal S. The processing power attribute of each signal is assigned a proper value according to allocation policy A.

If node N is a join node, signal S is buffered in the join node. When the join node receives all the required signals, an activation signal is created and then it leaves the join node. For this activation signal, its execution time cost attribute is set to the maximum value of the execution time cost attributes of the signals buffered in the join node; its processing power attribute is set to the sum of the processing power attributes of these signals; and its allocation policy attribute is set to the allocation policy attributes of these signals.

If node N is a lock node, the execution time cost simulator checks if all the required locks can be obtained. If so, all the required locks are set. Otherwise, signal S is put into the lock waiting queue.

If node N is an unlock node, certain number of variables are unlocked according to the description of the unlock node. Once some locks are released, signals in the lock waiting queue are checked. if all the locks for a signal can be obtained, such a signal is then removed from the queue. The amount of time the signal stays in the queue is added to the execution time cost attribute of the signal.

Due to parallelism in the computation, there may be more than one signal that is active at a time. If two or more signals require conflict locks at the same time, a random number generator determines which signal can get locks first. All the other signals are then put into the lock waiting queue.

When an activation signal finally arrives at end node of the computation, the simulation process terminates. The execution time cost attribute of the signal is then the execution time cost of the computation.

5. Example

To illustrate the use of the TCSS system, consider the classical producer - consumer problem. Assume there are one producer and one consumer. The producer produces n products and puts the products into a buffer, while the consumer gets these products from the buffer and consumes them. However, the producer and the consumer can not access the buffer at the same time, the buffer can only hold ten products, the producer can not put a product into the buffer when the buffer is full and the consumer can not get a product from the buffer when the buffer is empty. The following is the corresponding computation which is written in a Pascal-link language.

```
computation (n: integer)
    { @ perf(n, full, empty) }
    var count, i, next, j, first, produce, consume: integer;
        buffer: array [1..10] of integer;
    begin
        count := 0;
        fork
            begin  { .....  producer }
                next := 1;
                for { @ n } i := 1 to n do      begin
                    produce := i; { .....  produces product }
                    lock (buffer, count: write);
                    while { full } count = 10 do
                        begin
                            unlock (buffer, count: write);
                            lock (buffer, count: write)
                        end;
                    count := count + 1;
                    buffer [next] := produce; .....  puts product into buffer
                    unlock (buffer, count: write);
                next := next - (next / 10) * 10 + 1
            end
        end;
    begin  ..... consumer
        first := 1;
        for { n } j := 1 to n do
            begin
                lock (buffer: read, count: write);
                while { @ empty } count = 0 do
                    begin
                        unlock (buffer: read, count: write);
                        lock (buffer: read, count: write);
                    end;
                count := count - 1;
                consume := buffer [first]; {.....  gets product from buffer}
                unlock (buffer: read, count: write);
                consume := consume + j; {.....  consumes product }
                first := first - (first / 10) * 10 + 1
            end
        end
    join
end
```

In the above computation, the fork - join statement is used to create two parallel paths that correspond to the producer and the consumer. The lock and unlock statements are used to obtain and release locks, respectively. Special count segments that start with "signs are used to provide information needed for time cost analysis. The first special comment segment indicates the computation has three performance parameters, n , full and empty, with n indicating the number of products to be produced or consumed, full indicating the number of times the producer finds the buffer full when it needs to put products into the buffer, and empty indicating the number of times the consumer finds the buffer empty when it needs to consume products. For each " for " loop statement or while loop statement, a special comment segment is used to indicate the number of time the body of the loop is executed. Statement produce = i in the producer is used to simulate the product producing process, while statement consume = consume + j in the consumer simulates the product consuming process.

Assume the underlying computer system has one Vax processor and processing power is equally allocated. Assume the number of products to be produced is 8 and the buffer is never full or empty. Thus, n = 8, full = 0 and empty = 0. In the following, the chg command is used to set the execution environment and the cost command determines the execution time cost (units are microseconds):

```
= > chg p 1
= > chg ma Vax
= > chg a equal
= > cost produce_consume (8, 0, 0)
produce_consume: execution time cost: 1059.00 μ sec
```

Assume the underlying computer system has two PDP - 11 processors and processing power is equally allocated. Assume the number of products to be produced is 50 and the consumer is always slower than the producer. Thus, n = 50 and empty = 0. Further assume the number of times the producer finds the buffer full varies from 0 to 100 and it is binomially distributed with p = 0.6. The following *TCAS* commands set the assumptions and determine the average execution time cost (command mean determines the average execution time cost):

```
= > chg p 2
= > chg ma PDP11
= > mean binomial produce_consume (50, 0:100, 0)
produce_consume: computation structure derived
```

value for parameter p: 0.6
average execution time cost of produce_consume: 16950.69 μ *sec*

Assume the underlying computer system is the same as before. Assume the number of products to be produced is 20, the consumer is always faster than the producer, and the number of times the consumer finds the buffer empty is 45. Thus, n = 20, full = 0 and empty = 45. The corresponding execution time cost is:

= > cost produce_consume (20, 0, 45)
produce_consume: execution time cost: 9137.13 μ *sec*

Assume the underlying computer system has the same properties as before. Assume the number of products to be produced is 30, the number of times the producer finds the buffer full is 40, and the number of times the consumer finds the buffer empty is 50. The corresponding execution time cost is:

= > cost produce_consume (30, 40, 50)
produce_consume: execution time cost: 13037.31 μ *sec*

We also used the tool to evaluate the execution time for the matrix multiplication when it runs on the Sequent Symmetry machine [10]. Figure 6 shows the results produced by the tool and the results produced by real measurement of the execution time. The dashed line represents results produced by the simulation tool whereas the solid line represents measurement. It is clear that performance evaluation derived by the tool matches real measurement.

6. Conclusions

This paper describes a software tool that has been designed and implemented to aid users in simulating the execution time of parallel computations. The tool is useful in that it provides considerable insight into the factors that are important in determining the execution time of parallel computations. This insight can be used to decide what changes are necessary to improve the execution time. With this tool, it becomes easy for users to compare different designs of a parallel computation and choose the one with the minimum execution time as the best design [14]. The evaluation tool is being used

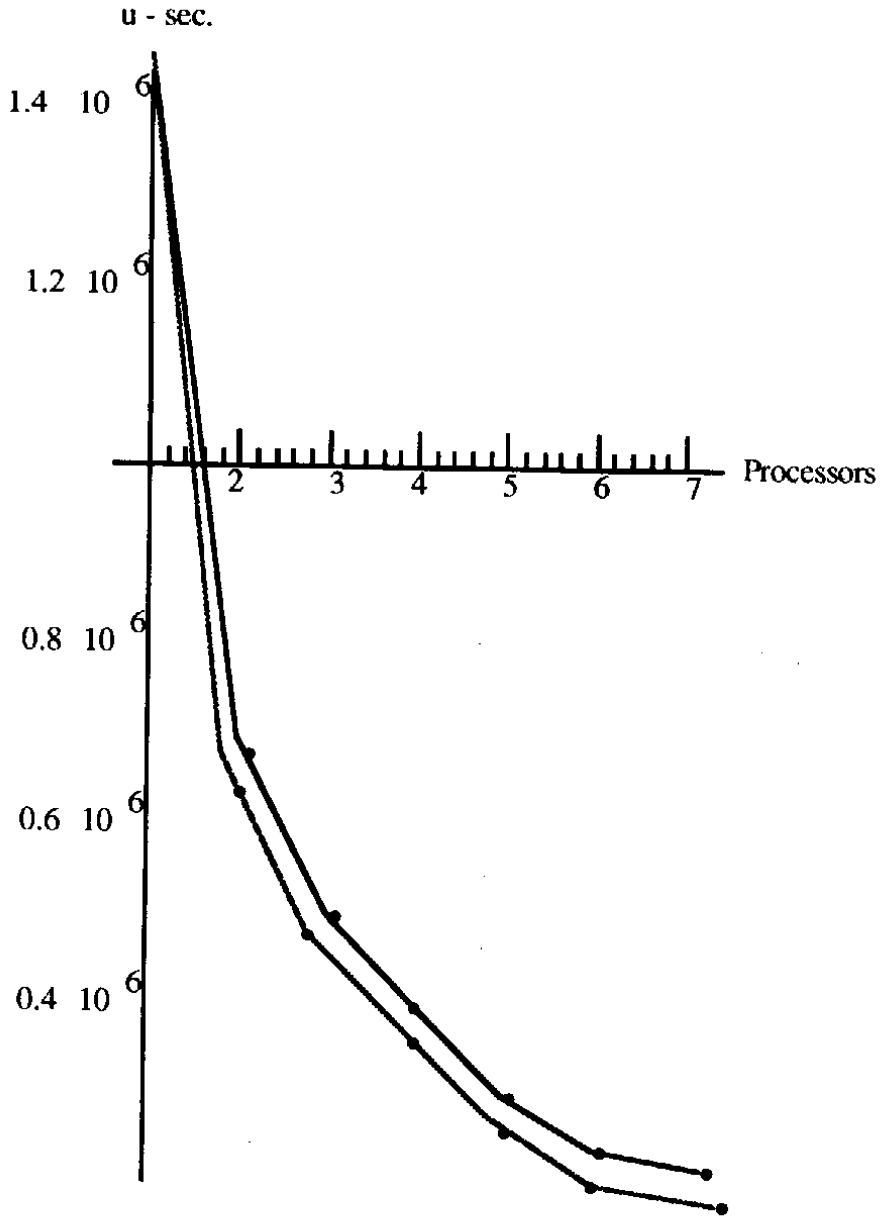

Fig. 6. Execution time of matrix multiplication.

to develop high performance computations and to study the effect of parallelism on real-life applications [1].

References

[1] Alherbish, J. *High Performance Arabic Character Recognition*, Ph.D. dissertation, University of Connecticut, in preparation (1996).

[2] Reda, A. A. and Qin, B. "A Flow Analysis Technique for Parallel Computations." *Proceeding of the IEEE PCCC*, Litchfield Park, AZ (March 1988).

[3] Reda, A. A.; Wang, J., and Sholl, H. "Software Performance Analysis Using a Graphic Modeling Technique." *The Journal of Information and Software Technology*, 33, No.2 (1991), 151-156.

[4] Reda, A. A. "A Computer Aided Design System to Develop High Performance Software." *Journal of Systems and Software*, Published by Elsevier Science Publishers B. V. (North-Holland), 5, No. 2 (May 1991), 139-147.

[5] Reda, A. A. "An Experimental-Analytic Approach to Derive Software Performance." *The Journal of Information and Software Technology*, 43, No. 4 (April 1992), 229-239.

[6] Booth, T.; Reda, A., and Lenk, R. "An Instrumentation System to Measure User Performance in Interactive." *Journal of Systems and Software*, 1, No. 2 (Dec.1981).

[7] Booth, T.; Zhu, D.; Kim, M.; Qin, B., and Albertoli, C. "PASS: A Performance Analysis Software System to Aid the Design of High Performance Software." The First *Beijing International Conference on Computers and Applications*, Beijing: (June 1984).

[8] Estrin, G. *et. al.* "SARA (System Architect's Apprentice): Modeling, Analysis, and Simulation Support for Design of Concurrent System." *IEEE Trans. Software Engineering*, SE-12, No.2 (1986), 293-311.

[9] Gabrielian, A.; McNamee, L., and Trawick, D. " The Qualified Function Approach to Analysis of Program Behavior and Performance." *IEEE Transactions on Software Engineering*, SE-11, No. 8 (Aug. 1985).

[10] Hwang, Kai. *Advanced Computer Architecture.* New York: Mc Graw-Hill, 1993.

[11] Molloy, M. K. *Fundamentals of Performance Modeling.* New York: McGraw-Hill, 1990.

[12] Qin, B.; Sholl, H., and Ammar, R. A. "Micro Time Cost Analysis of Parallel Computations." *IEEE Transactions on Computers*, 40, No.5 (May 1991), 613-628.

[13] Sholl, H. and Booth, T. "Software Performance Modeling Using Computation Structures." *IEEE Transactions on Software Engineeirng*, SE-1, No. 4 (Dec.1975).

[14] Smith, C. and Williams, L. "Software Performance Engineering: A Case Study Including Performance Comparison with Design Alternatives." *IEEE Trans. on Software Engineering*, 19, No.7 (July 1993).

[15] Tanenbaum, A. *Operating Systems: Design and Implementation.* Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1987.

[16] Wegbreit, B. "Verifying Program Performance." *Journal of ACM*, 23, No. 4 (Oct. 1976).

[17] Wetmore IV, T. *The Performance Compiler - a Tool for Software Designs*. MSc Thesis, Computer Science & Engineering Dept., University of Connecticut, 1980.

محاكاة الأداء العام لتكلفة زمن التنفيذ للوحدات الحاسوبية المتوازية

*** بن كين، رضا عمار و** هوارد شول**

** شركة IBM، أونتاريو، كندا، ** جامعة كنتيكت، بالولايات المتحدة الأمريكية*

ملخص البحث . في هذه الورقة تعرف تكلفة التنفيذ لوحدة حاسوبية متوازية كدالة في المدخلات، الخوارزمية، تراكيب البيانات، سرعة وحدة المعالجة، عدد وحدات المعالجة، تقسيم طاقة المعالجة، ونوع الاتصال بين المعالجات. وقد تم تعديل نموذج هيكل الحوسبة ليوضح أثر هذه العوامل السبعة.

كما تم تطوير نظام محاكاة لتحليل وتقييم زمن التنفيذ، هذا النظام تم تنزيله فعلياً في برنامج TCSS.