Learning Parallel Computing Concepts via a Turing Machine Simulator

Mônica Xavier Py, Laira Vieira Toscani, Luís C. Lamb, Tiarajú Asmuz Diverio
[1] Universidade Federal do Rio Grande do Sul
Instituto de Informática
PO Box 15064 — 91501-970 Porto Alegre
{mpy, laira, lamb, diverio}@inf.ufrgs.br

Abstract—

It is well-known that technology developments in Computing Science has led to new developments in Computing Theory and vice-versa. This work is a contribution towards the formalisation of Parallel Processing concepts. We exploit variations of the Turing Machine model, referred as natural extensions, which may be composed by several control units, tapes and heads in such a way that one can define a variety of Parallel Turing Machine models. Several notions and definitions of parallelism are identified including computability, complexity and performance of computations. A prototype of the Parallel Turing Machine model is presented. This prototype was used as a test bed for the assessment of the usefulness of these machine models as a parallel processing teaching tool, as well as a tool to facilitate the creation of a "culture" among students in high performance computing. Finally, we analyse the notions of performance, speedup, efficiency, load balancing, synchronisation and communication in parallel processing.

Keywords— **Parallel Turing Machine, parallel computing, parallel computing theory.**

I. Introduction

There is an increasing availability of high performance tools in the market. These tools are available in supercomputers, computer networks, computer clusters and even in multiprocessed personal computers whose computational power betters that of supercomputers of some years ago. This resource availability allows one to solve more complex problems faster by the use of more powerful, but also less expensive computers.

In spite of these resources availability, human users are not prepared to make use of them. We believe that most Universities have not been educating potential users for this new reality. Therefore, there is a demand for qualified professionals in parallel processing, which most probably contributes to enlarge the existing gap between software applications and technology developments.

Both researchers and educators have been motivated towards identifying new parallel programming teaching methodologies and techniques in order to answer the following: Have we not used parallel programming because our teaching environment lacks a parallel processing "culture" or is it a result of human beings being "sequential" creatures? How does one think in parallel? What kind of skills are necessary to enable the use of parallel processing? How can we teach in a parallel environment?

On the other hand, in order to teach parallel processing one has to identify skills and concepts which are essential in the understanding of the subjects being studied. In that way we aim at identifying the reasons why parallel programming has not been as successful as expected, and we conjecture that this may be due to the two reasons above (the sequential nature of human beings and the probable "cultural deficiency").

Currently, we are working on the extension of sequential machine models in order to represent some parallel computing paradigms. For instance, we are studying the application of Turing Machines with multiple heads in the representation of shared memory environments, where each distinct head represents a processor. The prototype we have been implementing allows the analysis and identification of a number of parallel notions such as performance, efficiency, speedup, load balancing, synchronisation, communication [MAR01]. One can easily understand how this model of parallelism works as it is among the easiest to understand and it is an adequate model for the visualisation of how parallel concepts actually work.

One of the aims of this work is to study Turing Machines with multiple heads in order to represent concurrency, developing a prototype model to analyse parallel processing definitions. In the development of our study, we adopted two methodologies. A theoretical approach, in which we analysed a variation of the Parallel Turing Machine (PTM) model, and a practical approach in which we used a prototype of the PTM, designed and implemented by the Computational Mathematics and High Performance Group at UFRGS (GMC-PAD), which allowed the study of a number of issues including computability, efficiency and performance. This prototype facilitated the observation of the practical-theoretical integration. The parallelisation of a sequential algorithm was run in the prototype which made it possible to analyse the characteristics and constraints of the shared memory parallel computing model. It is important to notice that the computation in this model significantly reduces the number of execution steps [MAR01].

The PTM model also facilitated the characterisation of

performance and efficiency notions. In the *computability* approach, we propose variations which allow parallel problem solving, both in a shared memory environment and in a distributed memory environment with message passing communication. The distributed memory model is represented by the use of multitape Turing Machines. We also show that these Turing Machine extensions can be simulated by the standard Turing Machines, and as consequence, they have the same computing power. These results are shown in [MIN67].

In the *efficiency* approach we tackle complexity notions. Complexity is measured by the number of elementary operations executed. In sequential computing one takes into account operations executed by a Turing Machine control unit over one single tape, whereas in parallel Turing Machines the existence of multiple heads requires that the number of operations performed over the tapes by each individual head to be added. In the case of message passing communication, one has to consider the time spent for message distribution. In addition, one should also take into account the minimal number of heads required in order to solve a problem, since this number may affect the complexity of the problem solution.

In the *performance* approach we aim at solving a problem in a minimum amount of time. This time interval is measured from the beginning to the end of the execution of the solution by every head of the Turing Machine.

II. The Theoretical Approach: Parallel Turing Machines

It is well-known that Turing Machines, proposed by Alan M. Turing, in 1936 are a formalisation of the notion of algorithm. Church's thesis [LEW99, ODI89, ROG67] states that Turing Machines are universal models of computing in the sense that it computers every computable function. Modifications in the structure of Turing Machines, such as the addition of tapes, control units, non-determinism, among others proved not to increase the computing power of the original model. However, technology advances and new generations of computers were presented and new types of parallel processing models were proposed. Computing became a concurrent process in which several processes can be executed simultaneously. The definition of parallel processing more widely accepted is the one formulated by Hwang. He says that parallel processing is an efficient way of processing information which emphasises the use of concurrent events in the computation process [HWA84].

A. Turing Machines

The computational model presented in this paper uses the formal definition presented by Diverio and Menezes [DIV00].

A Turing Machine is an 8-tuple $M = (\Sigma, Q, \Pi, q_0, F, V, ß, *)$, where:

- Σ the set of input symbols (or the input alphabet)
- Q a finite set of states
- Π the program or transition function
  $\Pi : Q \times (\Sigma \cup V \cup \{ß, *\}) \rightarrow Q \times (\Sigma \cup V \cup \{ß, *\}) \times \{L, R\}$
  (which is a partial function)
- q_0 the initial state of the machine, such that $q_0 \in Q$
- F a set of final states, such that $F \subset Q$
- V the auxiliary alphabet
- ß a special symbol, the blank symbol
- $*$ a special symbol to mark the beginning of the tape

The special symbol is used to indicate the leftmost position of the tape, and helps to control the head's motion. The transition function (program) has a pair composed by the current state and the symbol currently being read in the tape as its arguments in order to determine a new state, a symbol to be written over the tape, and the direction of the head's moves (either left (L) or right (R)). We denote the transition function by $\Pi(p, a_u) = (q, a_v, m)$.

It is supposed that $p, q \in Q, a_u, a_v \in (\Sigma \cup V \cup \{ß, *\})$ and $m \in \{L, R\})$.

B. On Parallel Machines Taxonomy

There exists a number of parallel machine classification criteria [HWA84]. Since parallel machines can be seen as a set of processors working cooperatively in the solution of a computational problem or algorithm, a relationship between these criteria (which are known as Flynn's taxonomy) and the ones used to classify Turing Machines can be established.

One can improve the performance of a computing system by applying parallel techniques into two of the system's components: its memory and its processing. As for the processor, one can change the number of processors used in the system, as well as change the connections between them. Usually, the type of connection between the processors and the system's memory defines the type of parallelism being executed/used.

In shared memory parallel computer systems, the memory is accessible to each individual processor, and the communication among processors is carried out through reading and writing in the system's memory. In distributed memory parallel systems, only the memory individually connected to a particular processor is accessible to that processor. Communication among processors is made by message passing from a processor to another.

In order to simulate shared memory parallel machines, we have used a Turing Machine with one tape to represent the shared memory and a control unit which manage the heads (each of which represents a processor). It is possible to increase the number of tapes and the number of control units, where each control unit can have one or more heads. Since

TABLE I

CLASSIFICATION TABLE.

Tape	CU	Heads	Description
S	S	S	original Turing Machine
S	M	S	shared-memory TM
S	S	M	single CU, multiple heads
S	M	M	multiple CUs, multiple heads
M	S	S	multiple tapes
M	M	S	message passing
M	S	M	multiple processors
M	M	M	cluster

one has the possibility of using a Turing Machine with multiple heads working over the tape (memory), it is clearly possible to simulate parallel computers with distributed memory. The study presented in this paper will focus on the shared-memory variant.

In [WOR97] the definition of multiple tapes, multiple heads, multiple control units Turing machines, denoted by MMM, is seen as a cluster. Notice that the processors of a parallel Turing Machine are controlled by the same program, but each individual processor (head) decides its own operation over the tape. Communication between processors is done by message passing mechanisms, which suggests a notion of a distributed algorithm. Therefore, this machine model can be seen as a model of distributed computing.

Table I presents a classification of different Turing Machine models.

C. On Turing Machines with Multiple Heads

Parallel Turing Machines can also be thought of as a combination of standard Turing Machines working over a single tape [WIE95]. Each individual processor of a parallel machine can then be represented by a standard Turing Machine. It is important to observe that some authors e.g. [WIE95, VAN90] define parallel Turing Machines as a non-deterministic Turing Machine with a single tape. However, this definition leads to what is known as false concurrency, which is unsuitable as a model for a large class of problems. The formal definition of a PTM is as follows.

Definition 1 *A Parallel Turing Machine (PTM) is a 9-tuple* $M = (Q, T, \Sigma, UC, F, q_0, \varepsilon, \delta, \beta)$ *where:*

Q	*finite state set*
T	*tape alphabet*
Σ	*input alphabet,* $\Sigma \subset T$
CU	*control unit*
F	*set of final states*
q_0	*initial state*
ε	*empty symbol*
δ	*transition function*
β	*blank symbol*

The program function $\Pi = Q \times \Sigma_j\, Q \times T_I \times D \times A$ where:

$$\begin{aligned} T_I &= \{\Sigma \cup \varepsilon\} \\ D &= \{R, L, S\} \\ A &= \{0, 1\} \text{ ('0' does not, '1' activates heads)} \end{aligned}$$

Each control unit, specified by CU, reads symbols from cells i being pointed by heads j. Formally:

$$UC : Q \times N \rightarrow N$$
$$UC = \{(q, \{p_{1,1}, \ldots, p_{i,j}\})/q \in Q, p_{i,j} \in /N\}$$

In the initial state, the PTM has only one active head, it is in state q_0, and pointing to the leftmost tape position. The input string is written on the tape, beginning in the leftmost cell. Cells on the right of the input string are filled with blank symbols. The PTM has at least one active head at each instant of time. Each individual head contains a copy of the symbols which control the transition to the next state, and every head executes the same transition function. There is a distinction between active heads which are currently working over non-terminating states, and passive heads which are the ones currently at final states. There is also a clear distinction between the empty symbol and the blank symbol. The computation works as follows: at each moment the PTM has one or more active processors. Each processor has a read/write head and a copy of the information which controls a state. In that way every processor executes the same program function. At any moment, each processor is exactly at one state in the set Q. As in the standard model, the tape is infinite to the right.

The PTM complexity measure definition is analogous to the standard Turing Machine complexity definition. The PTM's parallel execution time $PT(n)$ is defined as the (largest) number of execution steps taken by M for each input string of length n. The space complexity $PS(n)$ of the PTM is defined as the distance from the leftmost cell of the tape taken for a head to scan a symbol of length n.

A computation step of a PTM is defined as the parallel execution the following tasks. The transition function defines the values of the transition relation based on its current state and on the symbol being read on the tape. If the value of the head's activator is equal to one, the processor makes a copy of itself. Each copy of the processor is added to machine,

and from that point on, the new head shall act independently. According to the symbol scanned by the head, the processor executes one of the following:

```
a transition to a new state;
if new state = final state
    then processor passive
else
    a rewriting of the current symbol;
if ε read
    then no-write operation
if two heads attempting write on same cell
    then priority to head first activated;
```

Passive processors shall not execute any other action. One should also notice that the definition of the PTM not only allows the creation of processors, but also their elimination. PTM processors are controlled by a single program, but each processor independently decides about its own next move.

III. The Practical Approach: Implementing a Parallel Turing Machine Prototype

In order to simulate shared-memory parallel machines, we implemented PTM model in the prototype. This PTM has only one infinite tape, one control unit and a variable number of heads which execute the instructions defined by the program. The control unit controls the execution, creation and destruction of heads, and manages conflicts and dependencies.

In order to simulate concurrent behaviour, one has to think of the following considerations. Concurrency occurs whenever two or more processes interact with each other in order to solve a problem. This may require parallelisation of algorithms and the use of parallel and distributed architectures. If two concurrent processes need to share common resources, such as the memory space, their interaction can lead to what is known as race condition [TAN95], where the process execution ordering in time determines the result. The race condition may also lead to a behaviour that, in general, cannot be reproduced and is known as non-deterministic or non-functional behaviour. We define process synchronisation as the forced serialisation of events run by asynchronous concurrent processes [SHA96]. For instance, suppose that two heads are concurrently and asynchronously processing two distinct events. Since these heads are concurrent and asynchronous the events can happen at any instant, and in any order, even simultaneously. Two heads can be unified after a certain point (known as synchronisation point). The heads' synchronisation defines two kinds of synchronisation, synchronous and asynchronous [AND91].

We have implemented an example parallel program which computes the square function. This program allows the study of parallel computing notions, such as computability and performance issues.

The concept of computability is established by the fact

TABLE II

EXECUTIONS OF 10 SQUARED WITH SEVERAL HEADS.

No. heads	No. iterations	Percentage	No. operations
1	10201	100,0%	10201
2	5161	51,6%	10165
3	3481	34,1%	10131
4	2635	25,8%	10086
5	2135	20,9%	10067
10	1150	11,3%	10142
30	504	4,9%	10701
50	399	3,9%	10503

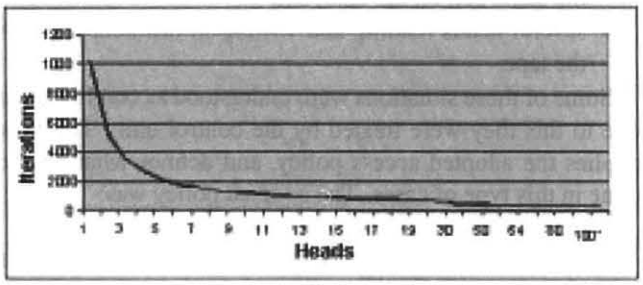

Fig. 1. Relation between the number of iterations and heads used.

that Parallel Turing Machine can be simulated by Standard Turing Machines (or equivalent). Thus, the computational power, represented by the class of problems solved by the standard machine, and just a few problems can be solved more easily and/or with more velocity. The facility is associated with the concept of the added computational resources. The velocity is associated with performance, that is, the time that is taken to solve the whole problem. The processing time does not mean number of operations carried out in each head or number of interventions of the control unit to solve the conflicts, is the interval of time that the program took to be executed running. In this case, the time was measured in iterations.

The Concurrency is studied with the purpose of solving the problem in a shorter period of time, through the utilisation of more than one head (processor). For instance, we can calculate, ten squared using one, two, three, ten or thirty heads. Table II relates the number of iterations with the number of heads used, and gives the percentage of the processing time in relation to the processing time with a single head (performance) and the number of conflicts generated by the use of the heads. Figure 1 presents the same information in a graphical form.

From these experiences, students notice the performance increase in processing with the use of more than one head. On the other hand, it is clear that to excessively increase the number of heads does not reduce the processing time. There

is an ideal number of heads to be used, which compares the processing time reduction with the increase of conflicts (efficiency).

Several situations were identified and analysed in order to maintain coherence computation. They are the following:

- more than one head trying to read the same cell on the tape;
- more than one head trying to write a different symbol in the same cell on the tape;
- more than one head trying to write different symbols in the same cell on the tape;
- one head trying to read while another tries to write in the same cell on the tape; and
- several heads reading and writing in different cells on the tape.

Some of these situations were understood as conflicts, and due to this they were treated by the control unit. This unit applies the adopted access policy, and defines what has be done in this type of cases. The adopted policy was:

- reading can always be done concurrently;
- in case of a conflict between reading and writing, must give priority to reading;
- in case of a writing conflict, we must always do the exclusive writing, i.e., only one head can perform a task at a given time (in a specific program, where the result is given by the number of cells used), in this way we have situations of postponement of tasks for conflict resolution.

A. An Algorithm for Conflict Resolution

The algorithm is described as follows:

1. The computation begins with one head, which scans the tape counting up to three; then the head returns to square one in the tape
2. After reaching square 3, a new head is created
3. A new head always returns to square one. Then it starts the computation, substituting the first symbol read
4. If necessary, A new head is created over the second tape cell. This new head then moves to the right until it reaches a blank symbol, writes an X over it, and restarts the computation from square one on the tape
5. If there is no symbol to substitute, the computation halts.

In the algorithm, there is no fixed set of states associated to each head, as every head works over every symbol. As a result, we reduce the size of the transition table and we have a greater degree of independence among heads (this is due to the fact that a head does not need to wait for a given symbol to begin the computation).

We can also study additional models of parallel computing in the prototype. For instance, we have studied some modifications over the Concurrent Reading Concurrent Writing (CRCW) model, which allows concurrent access for reading and writing operations can be considered, such as [JáJá92, GOU95]:

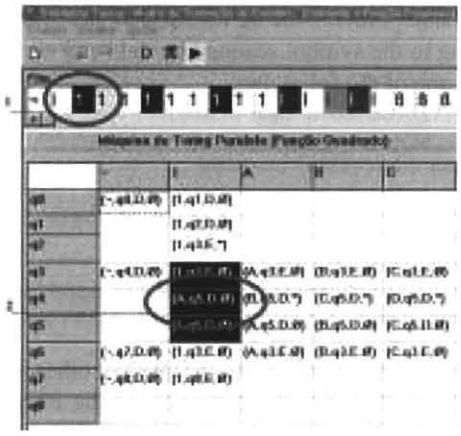

Fig. 2. Conflict situation in the running program x2.

- COMMON: this model allows concurrent reading/writing only when processors are trying to write the same symbol
- ARBITRARY: it allows an arbitrary processor to write a symbol
- PRIORITY: it assumes that processors are linearly ordered and gives priority according to their corresponding indexes.

This simulator use the PRIORITY model, we constructed a PTM to compute the square function. Whenever a conflict situation occurs, the head indexed by the smallest number is given priority to write over the cell, whereas the other conflicting head writes over the next cell containing a blank symbol.

Conflict situations are detected by the prototype as follows. Figure 2 shows the exact position after 33 iterations out of 1244. At the top of the Figure, the cells in black indicate that there is more than one head trying to write over them. In our example, the two heads positioned over cell 3 are trying to write the symbol "A" over the number "1". The master program allows only one head to execute this instruction and then moves to cell 4. The other head stays put, and does nothing.

Figure 3 indicates that there is another head, previously at cell 4, but we can notice from the picture that only one of the heads (which is in state q_4) is trying to write over a cell. The head currently at q_3 is trying to read a tape symbol and then move to the left. As this is not a conflict situation, the two heads execute their instructions and then move to the position indicated by their programs (the head which is writing a "B" moves to the right (R) and the head which is only reading a symbol moves to the left (L)).

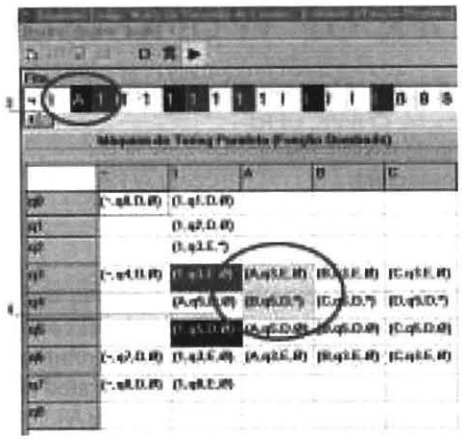

Fig. 3. Conflict solution.

With this study, we can identify the importance of the communication mechanism since the shared memory model required communication between the heads and the control unit. This was obtained through the addition of a fourth component in the transition function that describes the program. This unit manages conflicts, the creation, and exclusion of heads.

Finally, it is important to emphasise the study for this implementation. Two basic algorithms were developed. The first algorithm doubles the value to be squared in the tape (it multiplies the number by itself). Then, different heads were created in order to carry out the sums, as multiplication can be represented by successive sums. This model allowed the heads to execute, different tasks, which would make difficult the definition and formalisation of a corresponding Turing Machine. However, in the second algorithm, which was effectively implemented, it was used only one program for all heads. Executing a cooperative work; that is, they cooperated in all the tasks.

IV. Conclusions

This work is based on parallel machines theory, and establishes a relationship between practical aspects of parallel computing and theoretical concepts from computing theory. Our aim was to show that by using the notion of Parallel Turing Machines one can understand and learn parallel computing through examples implemented on a prototype, since one can visualise in a clear and concrete way a number of parallel computing notions such as performance evaluation, load balancing, synchronisation, and process communication. Our model shows to be an adequate tool to work on these notions, since we can see in the examples we have implemented that the increase in the number of heads (processors) renders a smaller number of instructions. Extensions of this study to a larger class of parallel computing models via Turing Machines are currently being investigated.

References

[AND91] ANDREWS, Gregory R. **Concurrent programming**: principles and practice. Redwood City: Benjamin/Cummings, 1991. 637p.

[DIV00] DIVERIO, T.; MENEZES, P. **Teoria da computação**: máquinas universais e computabilidade. Second.ed. Porto Alegre: Sagra-Luzzatto, 2000. (Livros Didáticos, v.5).

[GOU95] GOULART, Peter C. et al. **Paralelismo**: algoritmos e complexidade. [S.l.]: PPGC da UFRGS, 1995. (RP 306).

[HWA84] HWANG, K.; BRIGGS, F. A. **Computer architecture and parallel processing**. New York: McGraw-Hill, 1984. 846p.

[JáJá92] JáJá, Joseph. **An introduction to parallel algorithms**. Reading: Addison-Wesley, 1992.

[LEW99] LEWIS, H.; PAPADIMITRIOU, C. **Elements of the theory of computation**. [S.l.]: Prentice Hall, 1999.

[MAR01] MARQUEZAN, C. C. et al. Learning concurrency using parallel Turing Machine. In: PROC. OF THE 7TH WORLD CONFERENCE ON COMPUTER EDUCATION, 2001, Copenhagen. **Anais...** [S.l.: s.n.], 2001.

[MIN67] MINSKY, M. L. **Computation**: finite and infinite machines. Englewood Cliffs: Prentice Hall, 1967.

[ODI89] ODIFREDDI, Piergiorgio. **Classical recursion theory**: the theory of functions and sets of natural numbers. Amsterdam: North-Holland, 1989. Studies in Logic and the Foundations of Mathematics.

[ROG67] ROGERS JR, H. **Theory of recursive functions and effective computability**. [S.l.]: McGraw-Hill, 1967.

[SHA96] SHAY, William A. **Sistemas operacionais**. São Paulo: Makron Books do Brasil, 1996. 758p.

[TAN95] TANENBAUM, Andrew S. **Sistemas operacionais modernos**. Rio de Janeiro: Prentice Hall do Brasil, 1995. 493p.

[VAN90] VAN EMDE BOAS, P. Machine models and simulations. In: LEEUWEN, J. van (Ed.). **Handbook of theoretical computer science**. Amsterdam: Elsevier Science, 1990. v.A, p.1–66.

[WIE95] WIEDERMANN, J. Quo vadetis, parallel machines models. In: LEEUWEN, J. van (Ed.). **Computer science today**. Berlin: Springer, 1995. p.101–114. (Lecture Notes in Computer Science, v.1000).

[WOR97] WORSCH, T. On parallel Turing Machines with multi-head control units. **Parallel Computing**, v.23, p.1683–1697, 1997.