Timings for Associative Operations on the MASC Model

Mingxian Jin, Johnnie Baker, and Kenneth Batcher
Department of Mathematics and Computer Science
Kent State University, Kent, Ohio 44242
Phone: (330) 672-2430 Fax: (330) 672-7824
Email: {mjin, jbaker, batcher}@mcs.kent.edu

Abstract

The MASC (Multiple Associative Computing) model is a generalized associative-style computational model that naturally supports massive data-parallelism and also control-parallelism. A wide range of applications has been developed on this model. Recent research has compared its power to the power of other popular parallel models such as the PRAM and MMB models using simulations. However, the simulation of MMB has identified some important issues regarding the cost of certain basic MASC operations required for associative computing such as broadcasts, reductions, and associative searches. This paper investigates these issues and gives background information and an analysis of timings for these operations, based on implementation techniques and comparison fairness with respect to other models. It aims to provide justification and clarify arguments on the timings for these constant-time or nearly constant-time basic MASC operations.

1. Introduction

The MASC (Multiple Associative Computing) model is a generalized associative-style of computing that has been in use since the introduction of associative SIMD computers in the early 1970s [18,19]. It provides a practical, highly scalable model that naturally supports both data-parallelism and control-parallelism with a wide range of applications. The MASC model is an MSIMD type model that provides one or more instruction streams (ISs), each of which is sent to a unique set in a dynamic partition of processing elements (PEs). This allows a task currently in execution to be partitioned into multiple tasks using control parallelism. The associative feature of the model allows data in the local memories of processors to be located by content rather than by address.

A large amount of research work has been done for this model. For example, an associative language, called ASC, has been implemented for the MASC model with one instruction stream across many platforms [19]. A number of efficient algorithms have been developed for various problems [3,10,18,19]. Also, recent research has compared its power to the power of the CRCW PRAM [24] and MMB (Meshes with Multiple Broadcasting) models [4] using simulations.

However, the simulations between MASC and MMB has identified some important issues regarding the cost assigned to certain basic MASC operations such as the broadcast and reduction operations. Both the accuracy of the cost assigned to these operations and its fairness with respect to the costs assigned on other parallel models are extremely important, as they determine the accuracy of the comparison between MASC and other models. Likewise, the accuracy and the fairness are essential in determining the ability of MASC to efficiently support applications in different areas, both for programming and complexity analysis of algorithms. In making fair cost assignments, it is necessary to consider the theoretical asymptotic rate of increase in the cost that occurs as the data size increases. However, it is equally important to also consider how these operations can be implemented in hardware and the running time of the implementations of these operations. The theoretical asymptotic rate of increase, together with the running times for hardware implementations can be used to produce graphs that project feasible running time for an operation on data sets of varying size. When this graph is bounded above by a small constant even when the number of processors exceeds what is considered to be feasible in the foreseeable future, we explore the option of assigning a constant running time to these basic MASC operations.

This paper investigates these issues based on the implementation details of Goodyear's STARAN, which is the architectural ancestor of the MASC model. We will discuss and analyze comparative fairness for the timings of certain basic operations on the MASC model with respect to the timings on other parallel models, e.g., the PRAM and MMB models.

The paper is organized as follows. Section 2 gives a brief description of the MASC model and the motivation for MASC. Section 3 describes the hardware

0-7695-0990-8/01/$10.00 (C) 2001 IEEE

implementation from STARAN to support these basic operations. Section 4 discusses the individual operations based on the implementation facts. Section 5 compares the costs of these basic operations on the PRAM and MMB models and addresses the issue of what costs should be assigned to MASC when it is compared (using simulations) to related models. The last section is a summary.

2. The MASC model

The MASC model consists of an array of processing elements (PEs) and an array of instruction stream processors (ISs). Each PE, paired with its local memory, is called a cell. There are three real or virtual networks to connect cells and ISs, namely, a cell network used for communication among cells, an instruction stream broadcast/reduction network used for communication between ISs and their cells, and an IS network used for communication among ISs (Figure 1).

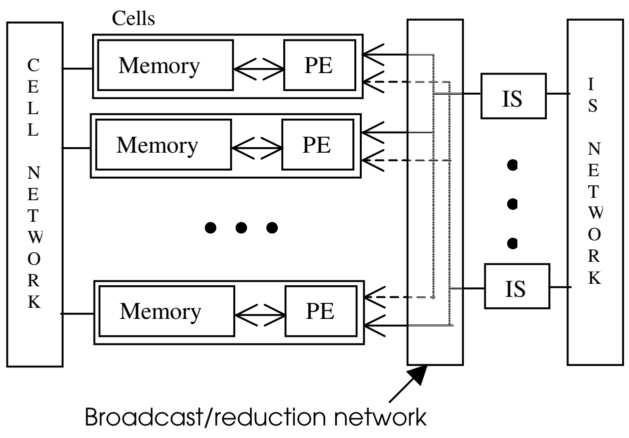

Figure 1. The MASC model

A complete description of MASC can be found in [18]. The following provides a brief summary. The MASC model is an MSIMD type model, where at least one or more ISs each send commands to a unique set in a dynamic partition of the PEs. The number of ISs is normally expected to be small, compared to the number of PEs. Each PE is capable of performing local operations as a sequential processor other than issuing instructions. Each PE can only access its own local memory. An IS is logically a processor which has a connection to each cell. Each IS has a copy of the program being executed and issues instructions to all its assigned cells. Each cell listens to only one IS and initially all cells listen to the same IS. The cells can switch to other ISs in response to commands from the current IS and the results of a data test. A cell can be active, inactive, or idle. An active cell executes the program instructions broadcast by the IS to which it is currently listening while an inactive cell listens to but does not execute these instructions. An IS can instruct an inactive cell to become active again. An idle cell does not contain any needed data and can be reassigned to a new task.

Assuming the word length is considered to be a constant, the MASC model supports the following important constant time operations for an IS:

- Broadcasting an instruction stream or a data item to the set of PEs listening to the IS
- Global reduction of a binary value stored in each active PE using logic OR or AND
- Global reduction of an integer (or real) value stored in each active PE using maximum or minimum
- Associative search to find the cells whose data values match the search pattern (called *responders*) or do not match the search pattern (called *non-responders*)
- The AnyResponder operation to determine if there is any existing responder after an associative search
- The PickOne operation to select (or "pick") an arbitrary responder from its set of active cells

These basic operations are essential to support the associative style of computing. They serve as core properties for the MASC model uniquely to achieve its effectiveness. In hardware, they are implemented through the broadcast/reduction network in a bit-serial fashion. The correctness of the timings assigned to each of these operations depends on both possible hardware implementations and comparative fairness with respect to other parallel model, as we will discuss in the following sections.

3. Broadcast/reduction network

In this section, we take a close view on a possible hardware implementation of the broadcast/reduction network based on the STARAN computer, which is the architectural motivation for the MASC model. The STARAN was an associative SIMD computer with 512 to 4096 PEs, depending upon the size of a particular installation. The STARAN was built in the early 1970's and the ASPRO, its architectural descendent, was built in 1980's by Goodyear Aerospace. Currently, the ASPRO is produced by Martin-Marietta and is used by the U.S. Navy. Their hardware implementation of associative operations through the broadcast/reduction network provides a possible implementation for these basic associative operations on the MASC model.

The broadcasting/reduction network on the STARAN is constructed using a group of resolver circuits [6]. A N-PE resolver consists of N PEs labeled PE_0 , PE_1 , …, PE_{N-1} and each PE_i has a responder bit R_i that is equal to 0 or 1. The resolver is designed to be able to tell each PE whether or not any earlier PE has a responder bit equal to 1. Thus for each i, it computes V_i

$= R_0 \vee R_1 \vee R_2 \vee \ldots \vee R_{i-1}$, where $\vee$ is the logic-OR operator and $0 < i \leq N$. The resolver also sends V_N to the control unit to tell it whether or not any responder bits are equal to 1. A 4-PE resolver is illustrated in Figure 2.

In practice, by using the parallel prefix idea, a resolver for N PEs can be built as a $\log_4(N)$-level tree of 4-PE resolvers. Each leaf represents a PE and the PEs are partitioned into groups of 4 PEs, which are fed into the first level of 4-PE resolvers. A reduction operation is executed by sending the signals down the tree to the PE leaves and then back up to the root of the tree to obtain the final result, while accumulating partial results in middle. Obviously, the delay from any input to any output is at most $(2 \log_4 N - 1)$ gates. A 16-PE resolver tree is shown in Figure 3.

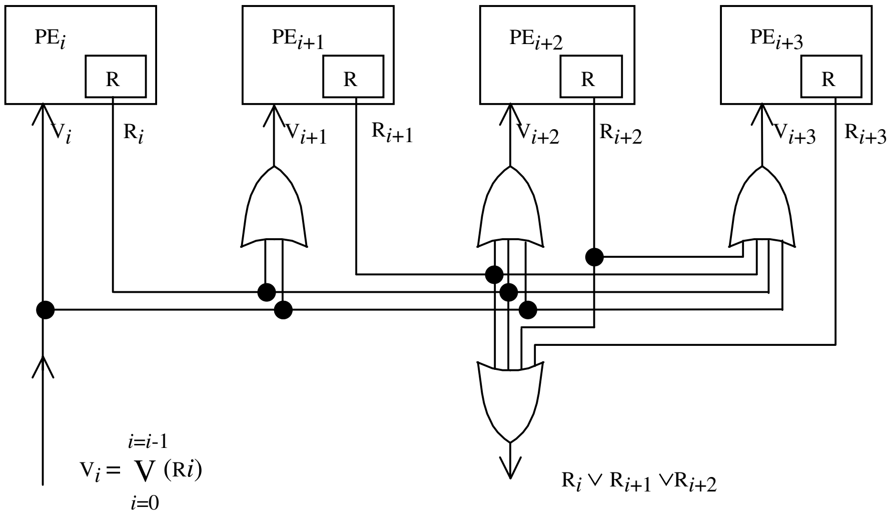

Figure 2. A 4-PE resolver with at most an 1-gate delay from any input to any output

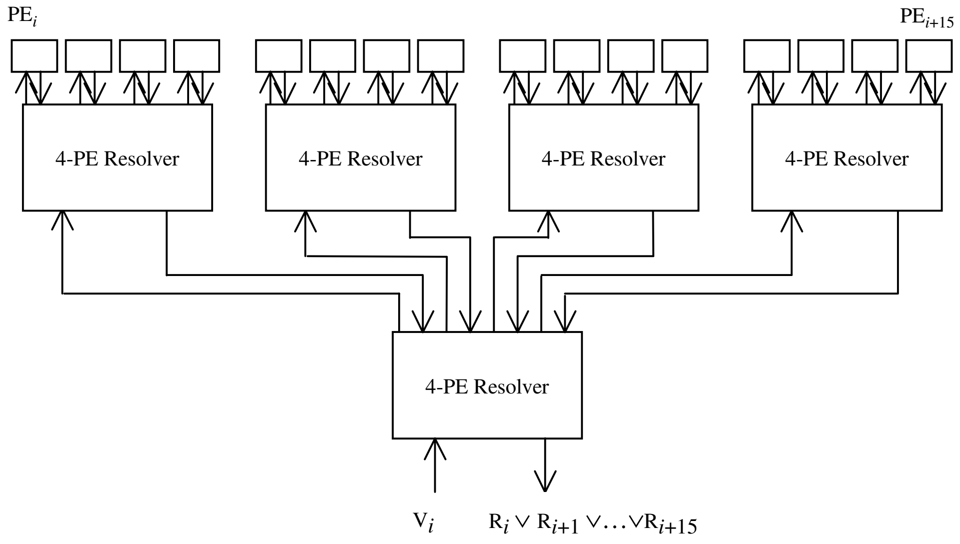

Figure 3. A 16-PE resolver with at most a 3-gate delay from any input to any output

4. Discussion on the basic operations

In this section, we discuss the timings on the individual basic MASC operations given in Section 2.

We consider broadcasting first. Broadcasting for bus-based architectures provides a fast way to transmit data between processors that are far apart. The assumption that an arbitrary subset of broadcasts on a bus-based architecture can occur in constant time, regardless of the number of processors or the length of buses, has been generally accepted in the literature [7,8,12,13,14,15,20,21,23]. This provides consistency among researchers in the field and is supported by experimental evidence even for very large architectures by today's standards [14,17]. For example, in today's architectures with at most tens of thousands of processors, the time to broadcast a data item over a bus is usually no more than that to perform a basic operation such as an addition by a sequential processor. Hence it can be bounded within one or a few machine cycles [7,17]. This constant time assumption for broadcasting greatly simplifies calculation of running times of algorithms and comparison of various models without loss of prediction accuracy. [1]

In a similar way, we assign the timing for a broadcast on the MASC model from an IS to its PEs to be constant. On the MASC model, broadcasting is used to issue instruction streams from an IS to its PEs and transmit data between them. Broadcasting a bit or a word on the MASC model is performed through the broadcast network. Practically, the broadcast network may be implemented as a separate network from the reduction network yet with the same structure (like in the STARAN), or they may share the same network. It is easy to observe that it takes $\log_4 N$ gate delay for a bit to travel from the root of the tree network to the N PE leaves at the bottom. Recall that broadcasting on a bus-based architecture technically requires linear order with respect to the bus length and the number of the processors. It is clear that the time for broadcasting increases asymptotically slower when using this tree-based network than when using a bus-based architecture.

The gate delay from any input to any output on the broadcast/reduction network for the MASC model is at most $(2\log_4 N - 1)$. This cost is given particular attention here. We argue that for practical purpose it could be bounded by a small constant. The reasoning is as follows. Typically, a gate delay takes about 1-5 nanoseconds. We may imagine even if we built an extremely large machine with size of $2^{2^{10}}$ processors, the gate delay would be at most $(2\log_4 2^{2^{10}} - 1) \times 5$ nanoseconds ≈ 5.1 microseconds. On the other hand, building a machine with $2^{2^{10}}$ processors appears to be impractical, since the number of atoms in the known universe is estimated to be less than 2^{2^9} [2]. It is likely that machines in the foreseeable future will have at most a few million processors. A machine with 100 million processors would have the gate delay less than 50 nanoseconds, which is comparable to the time for a memory access in today's systems. Since there are approximately 8 billion neurons in the human brain, it seems reasonable to conjecture that an efficient machine with that many processors would necessarily have to use a model quite different from those used today. Also, since a processor can handle much more complex computations than is considered possible for a neuron, it seems likely that if a machine were built to be able to handle the computations typical of the brain, then it would probably have far fewer than 8 billion processors.

We give brief diagrams for the function graph of $(2\log_4 N - 1)$ gate delay for this timing. Given the average 1-gate delay is 2 nanoseconds, the function graphs are shown in Figure 4 in regular scales and Figure 5 with the vertical axis in logarithmic scales, while N is between 1 to 2^{2^9}. (Notice that 2^{2^9} is approximately equal to 1.2e+154 shown in the figures. We did not give the function curves from $2^{2^9}+1$ to $2^{2^{10}}$ due to the limitations of our plotting device.) The two horizontal lines in Figure 5 are reference lines. It is clear that this function changes very slowly and is bounded by a small constant even if N is unreasonably large.

For a very large architecture, wire length is also an issue. According to the optimal VLSI layout, when the number N of processors is very large, the wire length can increase as large as $N^{1/3}$, based on the 3-D cube topology. However, this same problem exists in bus-based architectures. We may use the same solution (if any) to the problem. An option used in [7] is formulating a parameter, i.e., the *cycle* of a computation, as the maximal length of a single bus to capture this notion.

Let ω be the greater of the length of an instruction or a data item (i.e., word). Obviously, broadcasting an instruction or a data item one bit at a time from an IS to its PEs takes $O(\omega)$. With architectures that have already been built or those currently envisioned, the size of ω is a small constant. However, most parallel models assume that the processor's identification number can be stored in a word, thus requiring that for a machine with N processors, ω must be at least $\lceil \log N \rceil$ in size. It is traditional to assume in the bus-based architectures that the buses have bandwidth ω. In particular, for MASC we may build the separate broadcast network with the bus

[1] Since the number of gates that a bus is able to drive is limited, perhaps a re-evaluation of the time for broadcasting at the VLSI level is needed by researchers in the field before making this assumption for millions of processors. In considering VLSI designs to maximize the fan-out of a gate and minimize the layers of gates, a possible approach might be to increase the fan-out of the *n*-ary broadcast tree used here (e.g., to a 100-ary tree).

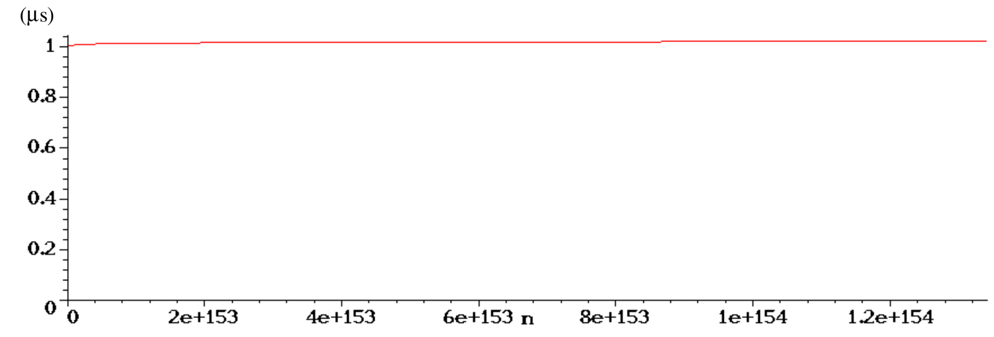

Figure 4. The gate delay of (2log $_4$ *N* -1) in regular scales.

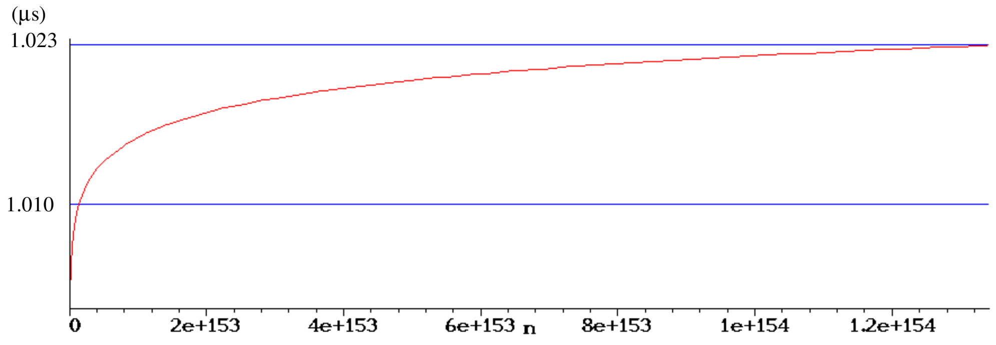

Figure 5. The gate delay of (2log $_4$ *N* -1) with the vertical axis in logarithmic scales

bandwidth ω so that it transmits ω bits simultaneously. Therefore, it is very reasonable to assume that both the word and instruction broadcasts require constant time.

Next, we discuss the logic OR global reduction operation. With each PE holding a binary value, a global OR is performed through the $\log_4 N$-level tree of 4-PE resolver network, as shown in Figure 2 and Figure 3. Thus, the same reasoning used for the 1-bit broadcasting can be used to justify assuming that a global OR takes constant time, although the tree traversal is in the opposite direction.

A global AND operation is implemented using a global OR operation simply by adding compliment gates to the inputs. So it can be assigned the same timing as a global OR operation.

An associative search is a unique feature of an associative model like MASC. On MASC, it is executed through the broadcast/reduction network by broadcasting a predefined search pattern to the PEs from an IS. Those PEs with matching values set their responder bit and remain active. Then an AnyResponder operation can be used to check if there exists any responder. Also, using the reduction network, the IS can select (or pick) an arbitrary (usually the first) active PE to do any special processing. We refer this to as a PickOne operation.

Clearly, an AnyResponder operation is essentially a global OR operation over all the responder bits. It involves one tree traversal of the reduction network and returns true if any responder bit is set among the active PEs and false otherwise. A PickOne operation is also performed by the reduction network by going up through the same tree traversal as a global OR. So both these two operations should have the same timing as a broadcast or a global OR, i.e., constant time. Notice that, after a PE is picked and processed, the PickOne operation clears the responder bit of the PE and the IS can proceed to pick another of the active processors. An associative search involves three steps, i.e., a broadcast of word-length pattern; a sequential comparison of two word data by each active PE; and a global AnyResponder operation. Hence, the associative search also requires constant time.

Let the word length be ω. In a bit-serial SIMD architecture such as the STARAN, operations on multiple bits are performed bit-serially. This means that, if a 1-bit local addition takes O(1), then a two-word local addition takes O(ω). If ω is assumed to be constant, then addition/subtraction is also a constant time operation, as one would expect. Some researchers argue that the word length in parallel computers with N PEs should always be (log$N$) or larger [2] so that it can store the identification number of each PE. However, since a parallel computer could have over 1.8×10^{19} PEs before log N is larger than 64, the assumption that the word length is also a constant seems to be reasonable, based on the size of current parallel computers and those in the foreseeable future.

Consider computing a global Maximum (or Minimum) of N integer (or real) values. As before, we continue to assume that the word length ω is considered a constant. Let each of the N PEs contains a value in a common word-size field. The global Maximum (or Minimum) operation can be implemented as a bit-serial operation with a global OR being computed for each bit of the field in the order of the left (most significant) bit to the right (least significant) bit. Each global OR bit operation keeps active those PEs whose bit value is 1 (or 0 for Minimum) until either only one responder is left or until all bits have been processed [19]. Additionally, if all active processors have a 0 value for one bit, then all remain active for the next round. If we assume that the time to compute a global OR is constant and ω is the word length, then the time for this operation is O(ω). If ω is also assumed to be a constant, as assumed for the addition of two words above, the time for a global Maximum (or Minimum) operation will also require only constant time.

As a related observation, we compare the implementation of the broadcast/reduction network of the MASC model that is in a bit-serial fashion and a sequential processor that is in a word-serial fashion. In fact, the hardware needed to support a sequential addition operation is not that different from the hardware needed to support the mentioned MASC basic operations on the broadcast/reduction network. It has been shown that the lower bound for a sequential processor to perform addition is $\Omega(\log k)$ where k is the number of bits of the two operands [11,16]. A sequential processor performing addition using carry-look-ahead adders as building blocks is much like a MASC performing a global reduction using resolvers as building blocks. The cost of addition for a sequential processor is considered to take constant time regardless of the number of its input bits (i.e., length of the operands). With the almost same hardware circuits, it is reasonable to apply the same measurement that ignores the number of input bits (i.e., number of PEs) on the MASC reduction operations.

The prefix sum operation is not supported in hardware in either the STARAN or the ASPRO. Also, it is not yet a basic operation on the MASC model. But based on the principle we discussed earlier, it is possible to implement this operation using the reduction network for a group of 1-bit binary values with each stored in a PE. As claimed in [16], "any design for a prefix sum operation can be converted to a carry computation network by simply replacing each adder with the carry operator." This could take the same time as a sequential addition operation, as well as a MASC reduction operation. However, for a ω-bit prefix sum, it may need more expensive hardware support to take the partial sums and partial carries to the next round of addition.

It is important to keep in mind that all of the above basic operations are implemented in hardware, in contrast to other parallel computational models that execute most of them using software algorithms. These features make the MASC model a unique parallel computation model that is powerful and feasible.

5. Comparison of timings with other models

In order to compare the MASC model with other computation models such as PRAM or MMB, it is critical that the timings charged for the basic operations of each model be fair. For example, if two models both support a broadcast with the same hardware, it would be unreasonable to charge one model O(log N) for a broadcast and the other O(1). Also, for two operations using the same hardware in one model, we should not charge one O(1) and the other O(log N).

It is useful to compare the timing cost for the basic MASC operations to those of similar operations for RAM and PRAM memory access. We give a short summary of the computational complexity of RAM and PRAM memory access presented in [2]. The memory access on these two classic models can be implemented by a combinatorial circuit that is called the Memory Access Unit (MAU). The MAU of RAM is implemented as a binary tree of switches. Each memory access from the processor has to use the path between the root of the tree and a memory location at a leaf of the tree. When the memory size is M, the depth of the MAU is clearly $\Theta(\log M)$. For PRAM, the lower bound for the depth of a MAU with M global memory locations and N (= O(M)) processors is also $\Omega(\log M)$. A theoretical optimal MAU for PRAM with depth $\Theta(\log M)$ is possible using a sorting circuit and a merging circuit. However, there is an efficient and more practical MAU for PRAM with depth $\Theta(\log M)$. Since the circuit in the optimal MAU has depth O(log M), each memory access is performed with a O(log M) switch delay. This is exactly the same situation as observed with the basic operations for the MASC model. Researchers and algorithm developers have ignored this small theoretical logarithmic factor for many years and have elected to treat memory access time as a constant, just like the time for other basic operations such as addition and comparison. This timing approach has dominated algorithm analysis on both the RAM and PRAM models in the literature and has significantly simplified and standardized complexity analysis of algorithms, both in terms of comparing algorithm performance and of comparing computational models.

We may further consider reduction operations for PRAM. Using prefix operation algorithms, all PRAM models can calculate logic OR/AND or maximum/ minimum in O(log$N$). However, using concurrent writes, the CRCW PRAM can compute every Boolean function with a small domain (e.g., binary values) in constant time [22]. But this does not hold for other variants of PRAM without concurrent writes, as they require at least $\Omega(\log N)$ algorithm steps for such a computation [9]. Thus, the charges for reductions and basic operations of logic OR/AND are reduced for the CRCW PRAM due to an assumed superior architectural implementation.

Next, we compare timings on MASC with timings on MMB, which is considered to be a more practical computational model than PRAM. A MMB is a basic mesh enhanced with several global buses, one bus for each row and one bus for each column. A processor can broadcast along its row or column bus to allow fast data transmission to all processors that are on the same row or column bus. In one time unit, only one processor is allowed to write a value on a bus. All other processors are assumed to simultaneously read the value being broadcast in constant time.

As with other bus-based architectures we discussed earlier in Section 3, the time to broadcast along a bus on MMB increases linearly as the number of processors increases, i.e., O(N). It has been generally accepted that this time can be considered to be constant. As for other basic operations on MMB, it has been shown in [21] that on a MMB with size of $N^{1/2} \times N^{1/2}$ and each PE holding a data item, the global reduction of finding a maximum/minimum or logic OR/AND can be computed in O($N^{1/6}$). Also, if all data items lie in the same row with only one item per processor, these reductions take O(log N). For the latter cases, it has been shown in [13] that $\Omega(\log N)$ is the lower bound for these reductions. Since MMB does not contain a circuit to perform these reductions, specific algorithms have to be designed for them. It is clear that substantially different methods are used to execute these operations on the MASC model.

Most recently, an augmented MMB model called the MHB (Meshes with Hybrid Buses) model was proposed in [12]. A MHB is a MMB enhanced with precharged 1-bit row and column buses. By draining the precharged high voltage to ground on the augmented bus, concurrent broadcast by several processors can be supported. This makes it possible for a global OR reduction operation to be done in constant time. This is called an ".OR" (dot OR) bus by hardware people and differs from the logic OR via gates that we discussed earlier for the MASC model. It also ignores a small but non-constant time for precharging or draining on the augmented bus (which is sometimes claimed to be logarithmic). Furthermore, it supports our simulation results in [4] that show that a MMB is less powerful than a MASC with a 2-D mesh unless it is augmented with extra hardware.

The above discussion and comparison of MASC with other models justify our constant time assumption concerning the timings of the broadcast and reduction operations on the MASC model. We argue that our assumption uses the same methodology that the other models have used, both for the purpose of theoretical research and for practical implementations.

6. Summary

We have given the details of our timing justification for the MASC basic operations, based on their hardware implementation on the STARAN, which is the architectural ancestor of the MASC model. We have also compared our timings with those of other models. A summary of the MASC operations is given in Table 1. In the second column, we list the timings when we assume that the broadcast bus has a 1-bit width. In the third column, we assume that the broadcast bus has a ω-bit width. These timings are based not only on actual implementation (e.g., STARAN and ASPRO), but on the comparable timings used by RAM, PRAM, and bus-based architectures. These models were included as part of our study in order to evaluate which MASC timing charges result in the fairest possible comparison of MASC to other models for parallel computation.

Table 1. Summary of the timings of the basic MASC operations

Operations	1-bit B. Bus	ω-bit B. Bus
Broadcast	O(ω)	O(1)
Addition/Subtraction	O(1)	
Logic OR	O(1)	
Logic AND	O(1)	
Associative Search	O(ω)	O(1)
AnyResponder	O(1)	
PickOne	O(1)	
Maximum/Minimum	O(ω)	O(1)

There are several issues to consider in determining the proper cost for basic operations for a model. These include implementation considerations, experimental data involving prototypes, theoretical considerations, and comparison with the costs assigned to other models with similar capabilities. Other considerations are that the costs assigned should be kept simple whenever possible, consistent with the added requirement that they must provide accurate comparison of performance of algorithms designed for the model and with the performance of this model when compared with other models. It would be useful to have more experimental data using modern technology to test the timings assigned to the basic MASC operations.

Acknowledgments

The authors wish to thank the following people for their help in writing this paper: Profs. Will Meilander, Robert Walker, Jerry Potter, and Arden Ruttan. We also wish to thank an anonymous referee for his very useful comments. This work was supported by a grant from Ohio Board of Regents CS Enhancement Initiative.

References

[1] N. Abu-Ghazaleh, P. Wilsey, J. Potter, R. Walker, J. Baker, Flexible Parallel Processing in Memory: Architecture + Programming Model, *Proc. of the 3rd Petaflow Workshop*, Feb. 1999. http://vlsi.mcs.kent.edu/~parallel/papers/tpf3.pdf

[2] S. G. Akl, *Parallel Computing: Models and Methods*. Prentice Hall, New York, 1997.

[3] M. Atwah and J. Baker, An Associative Dynamic Convex Hull Algorithm, *Proc. of the 10th Int'l Conf. on Parallel and Distributed Computing Systems*, 1998, pp. 250-254.

[4] J. Baker and M. Jin, Simulation of Enhanced Meshes with MASC, a MSIMD Model, *Proc. of the 11th Int'l Conf. on Parallel and Distributed Computing Systems*, Nov., 1999, pp. 511-516.

[5] K. Batcher, STARAN Parallel Processor System Hardware, *Proc. Of the 1974 National Computer Conf.* (1974), pp. 405-410.

[6] K. Batcher, The Resolver Network, seminar notes, Octobor, 1999.

[7] Y. Ben-Asher, D. Peleg, R. Ramaswami and A. Schuster, The Power of Reconfiguration, *J. of Parallel and Distributed Computing*, 13, 1991, pp. 139-153.

[8] S. Bokhari, Finding Maximum on an Array Processor with a Global Bus, *IEEE Trans. On Computers,* C-33, 2 (1984), pp.133-139.

[9] M. Dietzfelbinger, M. Mkutylowski, and R. Reischuk, Feasible Time-optimal Algorithms for Boolean Functions on Exclusive-write Parallel Random-access Machines, *SIAM J. Computing*, Vol. 25, No. 6, Dec. 1996, pp.1196-1230.

[10] M. Esenwein, and J. Baker, VLCD String Matching for Associative Computing and Multiple Broadcast Mesh, *Proc. of the 9th Int'l Conf. on Parallel and Distributed Computing Systems*, 1997, pp. 69-74.

[11] I. Koren, *Computer Arithmetic Algorithms*, Prentice Hall, New Jersey, 1993.

[12] R. Lin, S. Olariu, J. Schwing, The Mesh with Hybrid Buses: an Efficient Parallel Architecture for Digital Geometry, *IEEE Trans. on Parallel and Distributed Systems*, Vol. 10, No. 3, Mar. 1999, pp.226-279.

[13] R. Lin, S. Olariu, J. Schwing, and J. Zhang, Simulating Enhanced Meshes, with Applications, *Parallel Processing Letters*, vol.3, No.1, 1993, pp.59-70.

[14] M. Maresca, Polymorphic Processor Arrays, *IEEE Trans. on Parallel and Distributed Systems*, Vol. 4, No.5, May, 1993, pp. 490-506.

[15] R. Miller, V. Prasanna-Kumar, D. Reisis, and Q.Stout, Meshes with Reconfigurable Buses, *Proc. of the 5th MIT Conf. on Advanced Research in VLSI*, Boston, 1988.

[16] B. Parhami, *Computer Arithmetic Algorithms and Hardware Design*, Oxford University Press, New York, 2000.

[17] D. Parkinson, D. Hunt, and K. MacQueen, The AMT DAP 500, *Proc. of the 33rd IEEE Computer Society Int'l Conf.*, 1988, pp. 196-199.

[18] J. Potter, J. Baker, S. Scott, A. Bansal, C. Leangsuksun, and C. Asthagiri, ASC: An Associative-Computing Paradigm, *Computer*, 27(11), 1994, 19-25.

[19] J. Potter, *Associative Computing: A Programming Paradigm for Massively Parallel Computers,* Plenum Press, New York, 1992.

[20] V. Prasanna-Kumar and C. Raghavendra, Array Processor with Multiple Broadcasting, *J. of Parallel Distributed Computing,* 4:173-190, 1987.

[21] V. Prasanna-Kumar and D. Reisis, Image Computation on Meshes with Multiple Broadcast, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol.11, No.11, Nov. 1989, pp.1194-1201.

[22] R. Reischuk, Simultaneous Writes of PRAM Do Not Help to Compute Simple Arithmetic Functions, *J. of ACM*, vol. 34, No.1, Jan., 1987, pp.163-178.

[23] Q. Stout, Mesh-connected Computers with Broadcasting, *IEEE Trans. On Computers*, vol. C-32, No.9, Sept. 1983, pp. 826-830.

[24] D. Ulm, J. Baker, Simulating PRAM with a MSIMD Model (ASC), *Proc. of the Int'l Conf. on Parallel Processing*, 1998, pp.3-10.