Proceedings of the 11th International Conference
On Parallel and Distributed Computing Systems
November 3-6, 1999, Boston, MA, USA

Simulation of Enhanced Meshes with MASC, a MSIMD Model

Johnnie W. Baker and Mingxian Jin
Department of Math and Computer Science
Kent State University, Kent, OH 44242-0001
Voice: (330)-672-4004 Fax: (330)-672-7824
{jbaker, mjin}@mcs.kent.edu

Abstract: MASC (for Multiple Associative Computing) is a joint control parallel, data parallel model that provides a practical, highly scalable model that naturally supports small to massive parallelism and a wide range of applications. In this paper, we present efficient algorithms for a MASC model with a 2-D mesh to simulate enhanced meshes. Let MASC(n, j) denote a MASC model with n processing elements and j instruction streams. It is shown that a MASC(n, j) model with a 2-D mesh is strictly more powerful than a $\sqrt{n} \times \sqrt{n}$ MMB (Mesh with Multiple Broadcasting) when $j = \Omega(\sqrt{n})$. Simulation of a $\sqrt{n} \times \sqrt{n}$ MMB by MASC(n, j) with a 2-D mesh runs in O(1) time and requires no extra memory. Simulating a $\sqrt{n} \times \sqrt{n}$ BRM (Basic Reconfigurable Mesh) with MASC(n, j) with a 2-D mesh takes $O(\sqrt{n})$ extra time with O(n) extra memory when $j = \Omega(\sqrt{n})$. The reverse simulations of MMB or BRM with MASC with a 2-D mesh is also given. These simulations not only provide information about the power of the MASC model and also provide an automatic conversion of numerous algorithms designed for enhanced meshes to the MASC model.

Key Words: parallel models of computation, associative computing, simulation, mesh with multiple broadcasting, enhanced meshes, MSIMD

1. Introduction

The MASC (for *Multiple Associative Computing*) model for parallel computation is a generalized version of an associative style of computing that has been in use since the introduction of associative SIMD computers in the early 1970's [9]. It provides a practical, highly scalable model that naturally supports small and massive parallelism and a wide range of applications. MASC is a 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 the processing elements (PEs). This allows the task currently being executed to be partitioned into multiple tasks, using control parallelism. Most parallel applications are essentially data parallel in nature, but have several nontrivial regions where significant branching occurs. With MASC, control parallelism can be used to execute each of these different branches simultaneously, effectively utilizing the PEs. Briefly, this model supports data parallel execution of instructions, constant-time searching, constant-time maximum and minimum operations (assuming that words have constant length), one or more ISs, and synchronization of the ISs 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 possible implementation for the MASC model is discussed in [1].

The power of a computational model is indicated both by the efficiency of algorithms it can support and by the efficiency with which it can simulate other computational models. In this paper, we present efficient simulation algorithms between enhanced meshes and MASC. We also give some results based on these simulations. The simulations of MASC with the well-known enhanced mesh models provide a better understanding the power of the MASC model. In addition, they provide a method for converting algorithms designed for enhanced meshes to the MASC model.

This paper is organized into five sections. In particular, Section 2 gives a brief description of enhanced meshes and Section 3 provides an overview of the MASC model. Section 4 discusses the simulation algorithms in both directions. Section 5 gives the concluding remarks.

2. Enhanced Meshes

The mesh-connected computer (MCC) has been of considerable interest to researchers in parallel computation due to its regular structure and simple interconnection topology which is particularly well suited for VLSI implementation. The main drawback of MCC is its large diameter. For example, on a $\sqrt{n} \times \sqrt{n}$ 2-D MCC, routing a data may take $\Omega(\sqrt{n})$ time in the worst case.

To overcome the large diameter problem, an enhancement of the MCC has been proposed which gives the processors the ability to broadcast using buses. Such meshes are referred to as *enhanced meshes*. At any given time, only one processor is allowed to broadcast an item on a bus. The datum will be read by all of the remaining processors on that bus. The two types of buses used for enhanced meshes are fixed buses and reconfigurable buses[3]. A mesh can be enhanced with a fixed bus using the *single global bus* model in which all processors are connected to a single bus. The other fixed mesh model is the *mesh with multiple broadcasting* (MMB) in which the basic mesh architecture is enhanced with row and column buses. At each step, broadcasts can occur along one or more rows or else along one or more columns. The row and column buses can not be used in the same step. The *reconfigurable bus* models allow buses to be created

302-288

dynamically on the mesh while a problem is being solved. The number, shape, and length of these buses are not fixed and are defined by the algorithm, as needed. Assume each processor has four ports referred to as N, S, E, and W. The model that allows up to two disjoint pairs of ports to be connected is called the *general reconfigurable mesh*. In this paper, we will use only the *basic reconfigurable mesh* (BRM) in which every processor may set at most one connection involving one of the pairs {N,S} or {E,W}. Further details can be found in [7].

Simulations between enhanced meshes and CRCW PRAM has been given in [7]. They show that a $\sqrt{n} \times \sqrt{n}$ MMB can be simulated by a CREW PRAM(*n, m) in* O(1) time with O(*n*) extra memory. Similarly, if α denotes the inverse Ackermann function, they show that a $\sqrt{n} \times \sqrt{n}$ BRM can be simulated by a Common CRCW PRAM(*n, m*) with O(α*(n)*) extra time and O(*n*) extra memory.

3. The MASC Model

MASC is a hybrid control parallel, data parallel model of parallel computation with an array of PEs and an array of ISs. This is illustrated in Figure 1. This model also includes three real or virtual networks; namely, a PE network used for communications among PEs, an instruction stream broadcast/reduction network used for communication between an IS and a set of cells, and an IS network used for IS communications. The array of MASC machine with *n* PEs and *j* ISs is written as MASC(*n*, *j*). Normally, *j* is expected to be much smaller than *n*. Each PE is capable of performing local arithmetic, logical operations, and the other usual functions of a sequential processor other than issuing instructions.

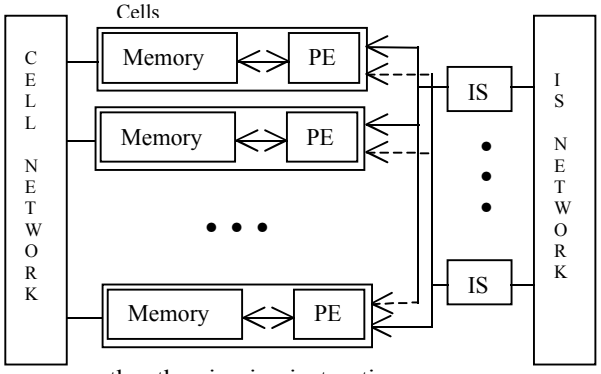

Figure 1. The MASC model

A *cell* is composed of a PE and its local memory. Each PE can only access its own local memory. An IS is logically a processor which has a bus connection to each cell. Each IS has a copy of the program being executed and issue an instruction to all cells in constant time. Each cell listens to only one IS and initially all cells listen to the same IS. The cells can switch to another IS in response to commands from the current IS. A cell is active, inactive, or idle. An active cell executes the program issued from the IS to which it is currently assigned while an inactive cell listens to but does not execute these instructions. An IS can instruct an inactive cell to become active again. Since each PE has a corresponding cell, we will use the concepts of a PE and a cell interchangeably when no confusion arises.

The MASC model supports the global reduction operations of OR and AND of binary values and of maximum and minimum of integer or real values for each IS and its active PEs in constant time (assuming the word length is considered to be a constant). Additionally, the MASC model supports a constant time associative search (assuming the word length is a constant) which allows data in the local memories of processors to be located by content rather than by address. The cells whose data value match the search pattern are called *responders* and the unsuccessful ones are called *non-responders*. The IS can activate either the responders or the non-responders. Each IS can select (or "pick one") arbitrary responder from the set of active cells in constant time. This IS can also instruct the selected cell to broadcast a data item on the bus and all other cells listening to this IS receive this value in constant time.

The assumption that an arbitrary subset of reductions and broadcasts on bus-based architectures can occur in constant time is supported by experimental evidence and is consistent with other researchers[5][7][11]. Also, the "pick one" operation can be implemented using a hardware circuit to select one of the active processors in what is reasonable to assume is constant time. While inexact, experimental tests with Goodyear's STARAN (with 1024 PEs) and Goodyear/Loral/Martin-Marietta's ASPRO (with approximately 1800 PEs) have shown that a resolver network can select one of the active PEs in about 5 clock cycles.

A standard associative language, called ASC, has been implemented for MASC(*n*,1) across many platforms including Goodyear/Loral/Martin-Marietta's ASPRO, the WaveTracer, and Thinking Machine's CM-2, and provides true portability for parallel algorithms[10]. In addition, an efficient ASC simulator has been implemented on both PCs and workstations running UNIX. It provides an efficient and easy way to design programs for algorithms that utilize massively parallelism. An object-oriented version of this language that supports multiple ISs is currently under development.

The MASC model supports algorithm development and analysis. A wide range of different type of algorithms and several large programs have been implemented using ASC language. For some examples of algorithms for this model, see [1][4][6][9][10].

It is convenient to assume that variables and constants that need to be globally available to multiple cells are stored in the memory of the ISs and may be broadcast to all active cells. The ISs have the ability to

read and store a value from a specific cell. The IS variables are called *scalar variables* while the cell variables are called *parallel variables*. To make a clear distinction between these two variable types, we add a "$" suffix to the parallel variable identifiers.

Previous MASC simulations include simulation of PRAM with MASC[12] and self-simulation[13]. Let PRAM(*n, m*) denote a PRAM with *n* processors and *m* shared memory. Without its network, MASC(*n, j*) can simulate priority CRCW algorithms that use only O(*j*) shared memory location in constant time with high probability. Also, MASC(*n,* 1) can simulate priority CRCW algorithms that use only a constant number *m* of global memory locations in constant time. Using both network algorithms and ISs to move data, MASC(*n*, *j*) can simulate priority CRCW PRAM(*n,m*) in O(min{*n/j, m/j*, route(*n*)}) with high probability, where route(*n*) is the average time for a general network routing. In the other direction, combining CRCW PRAM requires O(*j*) time to simulate MASC(*n*,*j*) since the ISs must be simulated sequentially. The self-simulation ability of MASC also allows the model to support a virtual machine with more PEs and ISs than are actually available, letting a programmer assume that enough PEs and ISs exist for their current program. MASC(*n, j*) with *n* PEs and *j* ISs can simulate MASC(*N,J*) with *N* PEs and *J* ISs in O(*N/n* + *J*) extra time and with O(*N/n* + *J*) extra memory for each PE, where $N \geq n$ and $J \geq j$. This establishes that the algorithms for this model are highly scalable.

4. Simulations of Enhanced Meshes

In this section, algorithms are presented for MASC to simulate enhanced meshes and to be simulated by enhanced meshes. We shall discuss simulations of enhanced meshes with MASC in Section 4.1 and 4.2; and then show the reverse simulation in Section 4.3.

Unless noted otherwise, we assume in our simulation algorithms that the mesh network for MASC(*n*, *j*) has size $\sqrt{n} \times \sqrt{n}$ and is in row–major order and that $j = \sqrt{n}$. The *i*th PE in the MASC array will be denoted PE_i ($1 \leq i \leq n$) and the *i*th IS will be denoted IS_i ($1 \leq i \leq \sqrt{n}$). Unless noted otherwise, the size of all enhanced meshes considered is $\sqrt{n} \times \sqrt{n}$ and is in row-major order. The processor in row *i* and column *j* ($1 \leq i, j \leq \sqrt{n}$) of an enhanced mesh is referred as to P(*i, j*).

For enhanced meshes, instructions used in algorithms consists of performing an arithmetic or boolean operation, communicating with a neighbor, broadcasting a value on a bus, or reading a value from a bus. In the MASC model, each PE is assumed to have exactly the same computation power as the PEs of an enhanced mesh. This assumption will simplify the simulations and require that the register and word length of both models be O(lg *n*), the word length assumed for enhanced meshes. Some researchers argue that the word length in parallel computers should always be lg *n* or larger[3] so that it can store the ID number of each PE. Since a parallel computer could have over 1.8×10^{19} PEs before lg *n* is larger than 64, this simplifying assumption seems to be reasonable, based on the size of current parallel computers. In order to compare the enhanced mesh models with the MASC models fairly, we assume the MASC buses have the same power as those of the enhanced meshes. In particular, the ID number of a PE or an IS can be broadcast on a bus and stored in constant time. Since both the MASC model used here and the enhanced mesh have a $\sqrt{n} \times \sqrt{n}$ mesh network, it is natural to map the processors in MASC to those in the same position in MMB. Since both models have identical mesh networks, the mesh operations can be simulated automatically. Therefore, the remaining work required is to simulate with MASC the operation of broadcasting over buses in an enhanced mesh.

Simulating an enhanced mesh with a global bus with MASC(*n,* 1) is trivial. All we need to do is to let all PEs listen to IS_1. Then any algorithm running on a $\sqrt{n} \times \sqrt{n}$ mesh with a global bus can run in MASC(*n,*1) with mesh connection in the same time. On the other hand, the reduction operation of maximum requires only O(1) time for MASC(*n,* 1) but non-constant time for the mesh with a global bus[2]. This gives the following obvious theorem.

Theorem 1. MASC (*n,* 1) with a 2-D mesh is more powerful than a $\sqrt{n} \times \sqrt{n}$ mesh with a global bus. Any algorithm on a $\sqrt{n} \times \sqrt{n}$ mesh with a global bus can be executed on MASC(*n*, 1) with a 2-D mesh with a running time that is at least as fast.

We next focus on the other two enhanced meshes models, namely, the meshes with multiple broadcasting (MMB) and the basic reconfigurable meshes (BRM).

4.1. Simulating MMB with MASC

Each PE_i ($1 \leq i \leq n$) in MASC with a 2-D mesh stores its position in the mesh network in two parallel variables, *row$* and *column$*. These positions are the same as that of the corresponding MMB processor. In particular, for all *i*, processor PE_i simulates the MMB-processor P(r_i , c_i), where $i = (r_i - 1)\sqrt{n} + c_i$, $r_i = \lceil i / \sqrt{n} \rceil$, and $c_i = (i - 1) \bmod \sqrt{n} + 1$. The desired simulation is a sequence of steps in which every processor in MASC is responsible for simulating the corresponding MMB processor.

As mentioned above, we only need to deal with the simulation of broadcasting in MMB. Since $\sqrt{n}$ instruction streams are assumed, we assign IS_i to both the *i*th row and the *i*th column. In one step, an IS can switch from instructing its row of PEs to instructing its column of PEs.

Initially, all PEs listen to IS_1 . In order to simulate a MMB row broadcast, the algorithm proceeds as follows. First all PEs switch to their assigned row IS. Each PE row should have at most one PE that needs to broadcast. The IS for each row checks to see if there is a PE that needs to broadcast a value and, if true, instructs this PE to place its broadcast value on the MASC bus. The broadcast operation is completed in constant time. The simulation of broadcast operations along column buses is done analogously. The only difference is that PEs switch to their assigned column IS.

Since each of the above steps take O(1) time, we have the total running time to be O(1). Each PE has two local variables to store its row and column indices. The resulting extra memory for all PEs is O(n), which is a minor cost. If these two variables are already resident in each PE, they can be omitted. Another additional cost is the $\sqrt{n}$ instruction stream processors that provide instructions to the PEs. However, since the total number of PEs and ISs is still O(n), this is also an insignificant cost in the simulation. Moreover, the role that the ISs play in the MASC model is similar to the role that the $2\sqrt{n}$ buses play in the MMB model. Due to the scalability of the MASC model, we can always use the self-simulation results in [13] to reduce the number of ISs required for this simulation to smaller number.

This simulation establishes that an algorithm running on a $\sqrt{n} \times \sqrt{n}$ MMB can be executed on MASC(n, j) with a 2-D mesh in asymptotically the same time when $j = \Omega(\sqrt{n})$. As a result, MASC (n, j) with a 2-D mesh is at least as powerful as a $\sqrt{n} \times \sqrt{n}$ MMB, when $j = \Omega(\sqrt{n})$. In particular, cost optimal algorithms for MMB are also cost optimal when executed through simulation on MASC. This raises the question as to whether this particular MASC model has the same power as the MMB model or is strictly more powerful.

We next show that there is a problem that can be solved faster using this MASC model than using the MMB model. Consider a $\sqrt{n} \times \sqrt{n}$ table and various partitions of these values into $\sqrt{n}$ *s*ets with $\sqrt{n}$ values, each of which contains exactly one value from each column and one value from each row of the table. An example of such a partition can be obtained using the wrap-around diagonals of this table. The problem is to efficiently calculate the maximum value for each set in any partition of this table. With MASC($n,\sqrt{n}$) , the n numbers of this table can be stored in n PEs with one value in each PE. Then all PEs with data for the ith set listen to IS_i ($1 \le i \le \sqrt{n}$) and locate the maximum value for that set in O(1) time. However, the MMB requires $\Omega(\sqrt{n} \log n)$ time due to the necessary data movement and the $\Omega(\log n)$ time needed to find the maximum of one set [7]. This gives the next theorem. The corollaries follow by making obvious adjustments to the preceding explanations.

Theorem 2. MASC (n, j) with a 2-D mesh is strictly more powerful than a $\sqrt{n} \times \sqrt{n}$ MMB when $j = \Omega(\sqrt{n})$. Any algorithm for a $\sqrt{n} \times \sqrt{n}$ MMB can be executed on MASC(n, j) with $j = \Omega(\sqrt{n})$ and a 2-D mesh with a running time at least as fast as the MMB time.

Corollary 3. For any constant c, MASC ($n,$ 1) with a $n \times c$ mesh connection is more powerful than a $n \times c$ MMB. Any algorithm for a $n \times c$ MMB can be executed on a MASC(n,1) with a $n \times c$ mesh connection with a running time that is at least as fast as on the MMB.

Corollary 4. MASC(mn, j) with a $m \times n$ mesh connection is more powerful than a $m \times n$ MMB, when j is $\Omega(\max\{m, n\})$. Any algorithm on a $m \times n$ MMB can be executed on MASC(mn, $\Omega(\max\{m, n\})$) with a $m \times n$ mesh connection with a running time at least as fast as on the MMB.

4.2. Simulating BRM with MASC

A reconfigurable mesh is a very powerful parallel computational model. A 2-D reconfigurable mesh was shown to be at least as powerful as the CRCW PRAM model in [14]. Later, it was proved to be strictly more powerful than the CRCW model in [8]. These indicate that simulation of the reconfigurable mesh with MASC has the potential to be a significant tool for evaluating the power of this model.

Consider a $\sqrt{n} \times \sqrt{n}$ BRM and a MASC(n, j) with mesh connection, where $j = \sqrt{n}$. Again, we view the desired simulation as a sequence of steps in which each PE in MASC is responsible for simulating one of the BRM processors. Since both models have a 2-D mesh connection, each processor in both models stores a row and column number that correspond to their position in the mesh. The MASC PE at location (r_i , c_i) in the mesh simulates the PE in the same position in the BRM.

Besides the above parallel variables *row$* and *column$,* several other MASC variables will be needed. The parallel variable, *connection$* stores the reconfiguration status of the corresponding BRM processor P(r_i , c_i). In particular, the variable *connection$* in PE_i stores 0 when the corresponding BRM processor P(r_i , c_i) has its {E,W} ports connected and stores 1 otherwise. The MASC parallel variable *leader$* stores the column (or row) number of the leftmost (or topmost) BRM processor which shares the same subbus with the BRM processor corresponding to PE_i , when connected in EW (or NS) direction. All the values in this parallel variable are initialized to 0 before each subbus broadcasts.

All PEs on the same subbus will have the same *leader$* value with one exception. This exception occurs with horizontal subbuses when two subbuses terminate at a common PE. In this case, the common PE will have the *leader$* value that identifies its left subbus but will be located in the column that identifies its right subbus. An analogous situation occurs for vertical subbuses. To simplify this discussion of the algorithm involving BRM, we assume that no two horizontal (or vertical) subbuses share a common PE. However, the algorithm techniques used are general and also work for this special case.

Now we need to deal with the simulation of broadcasting in BRM. Since there are $\sqrt{n}$ ISs assumed, let IS_i be assigned to the ith row and the ith column. Initially, all PEs listen to IS_1. They can switch to their row or column IS in one step. Also, in one step, an IS can switch from instructing its row PEs to instructing its column PEs. Recall that in one time unit, a BRM processor can connect one opposite pair of ports, namely {N,S} or {E,W}. Since the algorithms for broadcasting on the row and column subbuses are similar, we will cover only the algorithm for the row (or horizontal).

The first part of this algorithm handles the preprocessing needed to set up the row subbuses when the subbus configuration has been changed. Each PE_i initially switches and listens to its row IS. Then each PE_i sets its *connection$* variable to 0 if the BRM-processor $P(r_i, c_i)$ it simulates has its connection set to {E, W} and to 1 otherwise. Next, all PEs store 0 in their *leader$* variable. Then the local links of the mesh connection are used to check the connection status of their horizontal neighbors. All those PEs that do not receive a value of 0 are deactivated since they are not on a horizontal subbus. The algorithm next loops through the subbuses in each row sequentially from right to left, but the rows are processed in parallel. The left-most PE is identified in each subbus. This PE is called the *leader* of the subbus, and then its column number is broadcast to all PEs in the subbus and is stored in the *leader$* variable in these PEs.

The second part of this algorithm simulates a broadcast. First, the PEs that are on subbuses are activated and the rest remain inactive. These PEs are precisely the ones with a nonzero value in their *leader$* variable. Next, those PEs that wish to broadcast are instructed to set their broadcast flag. The broadcasts of the subbuses in the same row are processed sequentially from right to left, but the rows are processed in parallel.

Part 1 of the algorithm requires $O(\sqrt{n})$ for each IS_i to assign a leader for each of its subbuses in the worst case. The other operations of switching, activation, deactivation and finding the maximum take constant time. Local communication between neighbors in mesh connection takes constant time as well. In part 2, once the leader of a subbus has been found, a broadcast over the subbus takes constant time. However, if there are several subbuses in one row, and several broadcast requests simultaneously, the row IS must handle these requests sequentially. Since each row can have $O(\sqrt{n})$ broadcast requests, this operation requires $O(\sqrt{n})$ time in the worst case. The resulting worst case simulation time is $O(\sqrt{n})$. As for extra memory, each PE has three extra local variables to store the data needed in the simulation, so the total extra memory used is $3n$ or $O(n)$ for all PEs, which is insignificant. The other cost is that we need $\sqrt{n}$ extra processors as ISs in MASC which are not needed in BRM. However, we argue that $\sqrt{n}$ is asymptotically less than the total number n of PEs, so this cost can also be considered as an insignificant cost in the simulation. As mentioned before, in practice, the self-simulation results in [13] can be used reduce the $\sqrt{n}$ instruction streams to a smaller number, due to the scalability of the MASC model. Obviously, the preceding algorithm can be executed by a MASC(n, j) model when j is $\Omega(n)$. This yields the following theorem.

Theorem 5. MASC(n, j) with mesh connection where $j \in \Omega(\sqrt{n})$ can simulate a $\sqrt{n} \times \sqrt{n}$ BRM with $O(\sqrt{n})$ time and $O(n)$ extra memory.

4.3. Simulation of MASC with MMB and BRM

In this section, an algorithm is given to simulate MASC(n, j) with enhanced meshes. The MMB model is considered first. We make the same assumption for the size of the models and mapping for processors between the two models as before. Matching processors have the same location on the 2-D mesh. Hence, each $P(r_i, c_i)$ in the MMB corresponds to PE_i in the MASC, where $r_i = \lceil i / \sqrt{n} \rceil$ and $c_i = (i-1) \bmod \sqrt{n} + 1$. To simulate $\sqrt{n}$ ISs in the MASC, each of the $\sqrt{n}$ MMB-processors in the first column, i.e., $P(i,1)$ with $1 \le i \le \sqrt{n}$, is assigned to also simulate an IS and to issue an instruction stream. A copy of the program stored on each $P(i, 1)$ $(1 \le i \le \sqrt{n})$. The MMB simulates the execution of the instruction of the ISs sequentially. The MMB-processor $P(i, 1)$ broadcasts an IS_i instruction to all the processors in the first column. Next, this instruction is broadcast along each row to the remaining MMB-processors. Each MMB-processor allocates register (or memory) space for the two variables, *channel* and *active*. The ID of the IS that the simulated MASC-PE is listening to is stored in *channel* and whether or not this MASC-PE active is stored in *active*. Each MMB-processor checks these variables in order to decide whether to execute the current instruction issued. Initially, all PEs listen to IS_1, which is simulated by P(1, 1).

The MMB-processor executes a local computation or memory access in one step exactly as a MASC-

processor does. Also, a 2-D mesh data movement instruction by one IS is executed exactly as it is on the MMB. However, more work is required for the MASC reduction operations OR, AND, maximum, and minimum. When a MMB-processor receives an instruction operation for one of these reduction operations from an IS, e.g. IS_i, it does nothing at this step provided it currently is assigned to IS_i and is active. Otherwise, it determines the null value for this reduction operation and prepares to use this value in the reduction operation. The null values are as follows: 0 for OR, 1 for AND, MININT for maximum, etc. Next, the optimal algorithm provided in [5] is used to compute the reduction and get the value in P(1, 1). The final step is to send this result from P(1, 1) to P(i, 1), the processor simulating IS_i, using the first column bus.

The slowest part of this algorithm is the operation of the MASC reduction. According to [5], this reduction can be performed optimally on the MMB in $O(n^{1/6})$. Since this occurs inside a loop that is executed $\sqrt{n}$ times, the worst case time will be $O(\sqrt{n} \times n^{1/6})$ or $O(n^{2/3})$. There are two extra variables for each MMB-processor used to decide whether the processor executes the current instruction and what value should be provided to the instruction. Additionally, the MMB-processors in the first column require extra memory to store a copy of the program and to execute this program, which is constant length. The total extra memory used is $O(n)$.

Observe that an alternative to using the first column of MMB-processors to simulate ISs in MASC is to use $(\sqrt{n}+1) \times \sqrt{n}$ MMB in which the first column of processors are used only to simulate ISs of MASC. This will remove the load imbalance on the processors in the first column.

Since the BRM is more powerful than the MMB, it can also execute the above simulation of MASC with a 2-D mesh. As illustrated in the example in 4.1, problems can easily be chosen which do not allow effective uses of BRM subbuses. Since the primary use of BRM presently is to provide a model intermediate in power between the MMB and the reconfigurable mesh, a BRM reduction algorithm that is faster than an optimal MMB reduction algorithm is not known currently. We have established the following theorem.

Theorem 6. MASC(n, $\sqrt{n}$) with a 2-D mesh connection can be simulated by a $\sqrt{n} \times \sqrt{n}$ MMB or BRM with $O(n^{2/3})$ time and $O(n)$ extra memory.

5. Conclusions

Simulations between enhanced meshes and the MASC model have been presented in this paper. It is shown that MASC(n, j) with a 2-D mesh can simulate a $\sqrt{n} \times \sqrt{n}$ MMB in O(1) time with no extra memory when $j=\Omega(\sqrt{n})$. We show that MASC(n, j) with a 2-D mesh can simulate a $\sqrt{n} \times \sqrt{n}$ BRM in $O(\sqrt{n})$ time with $O(n)$ extra memory when $j=\Omega(\sqrt{n})$. Simulation of MASC(n, j) with a $\sqrt{n} \times \sqrt{n}$ MMB or BRM takes $O(n^{2/3})$ time with $O(n)$ extra memory. We also show that MASC(n, j) with a 2-D mesh is more powerful than a $\sqrt{n} \times \sqrt{n}$ MMB when $j=\Omega(\sqrt{n})$. These results provide an effective ways in understanding the power of the MASC model by comparing it with the well-known enhanced meshes. Also, the constant time simulations enable the algorithms designed for enhanced meshes to be transferred to MASC with the same running time.

Acknowledgments:

The authors wish to express their gratitude to Dr. Kenneth Batcher and Dr. C. Greg Plaxton for several helpful 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, *Proceedings of the 3rd Petaflow Workshop*, February 1999, 7pgs, http://www.capsl.udel.edu/conferences/TPF-3/agenda.shtml.

[2] A. Aggarwal, Optimal Bounds for Finding Maximum on Array of Processors with k Global Buses, *IEEE Trans. on Computers,* Vol. 35, 1986, pp.62-64.

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

[4] M. Atwah, J. Baker, An Associative Dynamic Convex Hull Algorithm, *Proceedings of the 10th IASTED International Conference on Parallel and Distributed Computing Systems,* 1998, pp. 250-254.

[5] D. Bhagavathi, S. Olariu, W. Shen, L. Wilson, A Unifying Look at Semigroup Computations on Meshes with Multiple Broadcasting, *Parallel Processing Letters,* Vol. 4, 1994, pp. 73-82.

[6] M. Esenwein, J. Baker, VLCD String Matching for Associative Computing and Multiple Broadcast Mesh, *Proceedings of the 9th IASTED International Conference on Parallel and Distributed Computing Systems,* 1997, pp. 69-74.

[7] R. Lin, S. Olariu, Simulating Enhanced Meshes with Applications, *Parallel Processing Letters,* 3(1), pp. 59-70, 1993.

[8] S. Olariu, J. Schwing, J. Zhang, On the Power of Two-dimensional Processor Arrays with Reconfigurable Bus Systems, *Parallel Processing Letters* vol. 1, No. 1 (1991), pp. 29-34.

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

[10] J. Potter, *Associative Computing: A Programming Paradigm for Massively Parallel Computers* (New York, 1992).

[11] D. Parkinson, D. Hunt, K. MacQueen, The AMT DAP 500, *Proceeding of the 33rd IEEE Computer Society International Conference,* 1988, pp. 196-199.

[12] D. Ulm, J. Baker, Simulating PRAM with a MSIMD Model (ASC), *Proceedings of the International Conference on Parallel Processing,* 1998, pp.3-10.

[13] D. Ulm, J. Baker, Virtual Parallelism by Self Simulation of the Multiple Instruction Stream Associative Model, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications,* 1996, pp. 1421-1430.

[14] B. Wang, G. Chen, Two-dimensional Processor Array with Reconfigurable Bus System Is At Least As Powerful As CRCW Model, *Information Processing Letters* 36 (1990), pp. 31-36.