Simulating PRAM with a MSIMD model (ASC)

Darrell R. Ulm and Johnnie W. Baker
Department of Mathematics and Computer Science
Kent State University
Kent, OH 44242 U.S.A.
Voice: (330)-672-4004 Fax: (330)-672-7824
dulm@mcs.kent.edu jbaker@mcs.kent.edu

Abstract: The ASC (MSIMD) model for parallel computation supports a generalized version of an associative style of computing that has been used since the introduction of associative SIMD computers in the early 1970's. In particular, this model supports data parallelism, constant time maximum and minimum operations, one or more instruction streams (ISs) which are sent to an equal number of partition sets of processors, assignment of tasks to the ISs using control parallelism. ASC also allows a network to interconnect the processing elements (PEs). This paper shows how ASC can be simulated with synchronous PRAM, and the converse. These results provide an important step in defining the power of associative model in terms of PRAM which is the most well studied parallel model. Also, these simulations will provide numerous algorithms for ASC by providing an automatic method of converting algorithms from PRAM to ASC.

Keywords : *computational models, PRAM, associative computing, ASC, data parallel, massively parallel, SIMD, parallel algorithms*

1 Introduction

This paper deals with the ASC model of computing developed by Dr. Johnnie W. Baker, Dr. Jerry L. Potter and others at Kent State University[1]. This model is a generalization of the SIMD paradigm where k *instruction streams (IS)* send commands to k partition sets in a collection of SIMD processors. The goal is to simulate ASC with the popular model of computing, PRAM, in order to give automatic upper bounds on algorithms run under PRAM simulation. PRAM is probably the widest known model of parallel computation, and more algorithms have been written for it than any other parallel model. By simulating PRAM, ASC can utilize all PRAM algorithms.

An algorithm to simulate priority CRCW PRAM with ASC is presented. The ASC simulation of PRAM is important because it gives a method to implement any of the numerous PRAM algorithms on the ASC model, and it also defines the speed with which a PRAM algorithm will run on the ASC model without an actual implementation. The ASC simulation of PRAM with only one instruction stream and some network connecting the processors lowers the best case simulation time to constant if there are only a constant number of PRAM shared memories to be simulated. In this case the ASC simulation of PRAM is optimal. However, a network connected set of processors without the addition of an instruction stream would be limited by the diameter of the network in terms of how fast it could simulate PRAM, even for PRAM with a single shared memory.

The main operations simulated are the concurrent reads and concurrent writes of PRAM. A great deal of previous work has been done concerning simulating PRAM with parallel computers. The work here expands what was known previously by combining existing network methods with the added power gained by having one or more instruction streams that can coordinate and communicate globally. Depending on the communication patterns of an algorithm executing on the simulated PRAM, the ASC simulation gives a better lower bound even with one instruction stream. A hybrid algorithm for simulation is discussed to move data with either the network interconnecting the ASC PEs or the network between the instruction streams and PEs.

2 The Multiple IS Associative Computing Model (ASC)

This section describes the associative model of computation presented in the IEEE Computer article "ASC: An Associative Computing Paradigm," which is based on work done at Kent State University[1]. Also, see [2][3][4][5][6]. ASC is a model for a currently buildable, massively parallel, multi-purpose parallel computer which is easy to program and can execute many types of programs efficiently on a wide variety of platforms. A parallel computing model with these characteristics is greatly needed if parallel computing is ever to be generally accepted by the computer industry. The ASC model supports a generalized associative style of computing that has been in use since the introduction of SIMD computers in the early 1970s[2][1]. The model reduces the arduous chore of task allocation and embodies the intuitive and well accepted method of data parallel programming. There are many data parallel languages available that ASC models well and also an actual language called ASC[1][7]. The ASC language has been implemented on various platforms such as the Connection Machine 2, Goodyear/Loral/Lockheed-Martin Aspro, the Wavetracer, personal computers, and UNIX workstations.

The data parallel style of programming is actually very sequential in nature, and therefore is more easily mastered by traditional SISD programmers than the task allocation of MIMD programming. A standard associative language (such as ASC) could be implemented across many distinct platforms providing true portability for parallel algorithms[7][1].

In the most basic terms, the *associative model (ASC)* has an large array of processing elements (PEs) and one or more instruction streams (ISs) that broadcast their commands to partitions in the set of PEs. The number of ISs is normally expected to be small in comparison to the number of PEs. The multiple ISs supported by the ASC model allows greater efficiency, flexibility, and reconfigurability than is possible with only one IS[8]. An ASC machine with j ISs and n PEs will be written as $ASC(n,j)$. Each PE has a local memory and ASC supports the associative processing concept, which is to locate objects in the local memory connected to PEs by content instead of by location. This is accomplished by searching a specified field of each PE for a given data item[9]. Each PE is capable of performing local arithmetic and logical operations and the other usual functions of a sequential processor[10]. A wide range of different types of algorithms and several very large programs have been implemented using the ASC language including a parallel optimizing compiler for ASC, two rule-based inference engines, and an associative PROLOG interpreter[5][1][11][3][12][13].

2.1 ASC; A Generalized Data Parallel Model

Hillis and Steele[14] introduced data parallel programming in the early 1980s, but this work did not provide a precise model. The ASC model extends this concept to be a complete programming paradigm. It is appropriate to have a model that emphasizes data parallel programming, even though other computational models may also support it. Other recently developed models such as BSP and LogP tend to focus on MIMD concepts and thus capture the MIMD style of computing best[15][16]. In fact, a 1991 survey reports that 90 percent of parallel applications are data parallel in nature [17][18].

The ASC model supports associative and data parallel programming with multiple instruction streams. No specific machine architecture is required to effectively use this model. It can be efficiently implemented on PCs, single workstations, SIMDs, MIMDs, and distributed systems. This model is intended to standardize the concept of associative computing and to provide a basis for complexity analysis for data parallel and associative algorithms[19].

The ASC model is a hybrid SIMD/MIMD model and is capable of both styles of programming. A frequent criticism of SIMD programming is that several PEs may be idle during if-else or case statements. The instruction streams provide a way to concurrently process conditional statements by partitioning the PEs among the ISs. In fact, a compiler can easily do this for the programmer. In ASC, PEs are assigned to an instruction stream based their local data, and ASC conceptually supports a very large number of PEs. Models such as LogP, BSP, and others envision future architectures maxing out in the hundreds or low thousands of PEs. Yet there remain large problems with huge amounts of data that require hundreds of thousands of PEs to avoid the Von-Neuman bottleneck, and the data parallel features of ASC are critical to these problems.

The ASC model for parallel computation specifically supports data parallelism, data reduction operations, broadcasting, one or more instruction streams (ISs) each of which is sent to a distinct partition set of processors out of the whole collection of processors, and task assignment to ISs using control parallelism. ASC also allows a network, real or virtual, to interconnect the processing elements (PEs) for generalized communication with a bandwidth determined by the particular network. The basic ASC model is shown in Figure 1.

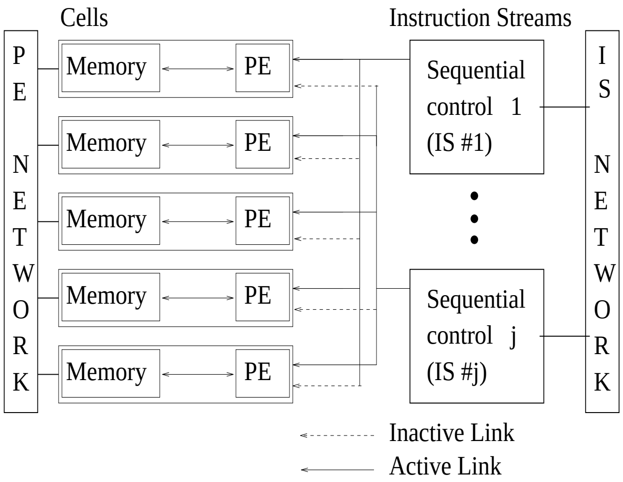

Figure 1: The Basic ASC Model

A PE is assumed to be reasonably basic so that the maximum number of PEs can be integrated on a chip when building an actual ASC machine. It is often argued that this trade-off of complexity for more processors gives a better cost for performance ratio over implemented MIMD system[18]. Furthermore, the network connecting the ISs to the PEs has the functionality to emulate an interconnection network between the PEs, thus providing a method to route PE data even if no PE network exists. There is no restriction on the network used with the ASC model. It could be the mesh, hypercube, shuffle-exchange, or many others. The programmer does not need to worry about the actual network present or the routing scheme, only that ASC is capable of generalized routing with some latency. Some of the most obvious choices for an actual ASC machine are the linear array or the mesh because of the ease to implement them in VLSI and their expandability.

There are three networks in the ASC diagram in Figure 2: the PE interconnection network, the IS interconnection network, and the network between the PEs and ISs. Any of these networks may be virtual or all three may be handled by only one or two networks. To ensure running time predictability of the data parallel model on actual machines, the basic operations of the model are abstracted into four parameters: u is the time to perform a routing of a word among all PEs (a word from each PE to any other PE location), t_i is the time to perform a sequential PE local operation where $1 \leq i \leq v$ where v is the number of different operations (ex. meta operations such as summing up an array of local integers are included), s is the maximum time for an IS communication or to synchronize ISs, and b is the time to perform a broadcast or a simple reduction (i.e. *and, or, min, max*) involving active PEs. The b parameter is also the time for an IS to read or write a single word. If the value of v is 1 then t will represent t_1. The speed of I/O can be measured as the amount of time it takes to read or write a word of information to a secondary storage unit and is represented by d. Not to be overlooked, the other important parameters are the number of PEs or n, and the number of ISs which is j.

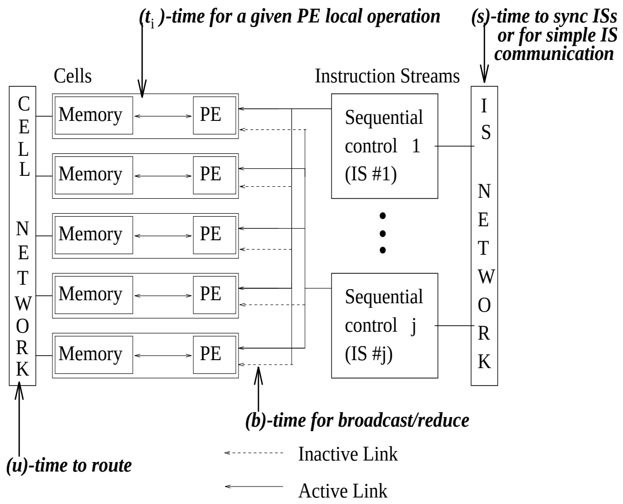

Figure 2: The ASC Model (with prediction parameters)

2.2 Definition of the ASC Model

The properties of the ASC model are presented in this section with the rules that govern each property of ASC. The components of an ASC machine are shown in Figure 2. The term *cell* is used as a definition for a processing element (PE) combined with a local memory for that element. The cell network is some network interconnecting the group of cells. The instruction stream processors store the programs for the cells and issue their commands to the PEs of the cells. There should also be some network for the ISs to communicate synchronization information. Cells can be either active, idle or inactive. Active cells are executing the program broadcast from an IS. An inactive cell is in an ISs group of cells, but do not execute instructions until the IS instructs inactive cells to become active again. Idle cells are not currently executing any part of the program but may be assigned to some IS at a later time. ISs can be active or idle. An active IS is currently issuing instructions to a group of cells. An idle IS is waiting until another IS forks, partitioning its PEs between itself and a new previously inactive IS.

1. THE PROPERTIES OF CELLS (informally referred to as PEs):

- Each cell has a local memory and a processing element (PE).
- All cells are connected by a network.

2. THE INSTRUCTION STREAM (IS) PROPERTIES:

- Each IS has a copy of the program being executed and may broadcast an instruction or a word to its cells.
- The IS processors communicate control information with each other only when groups of PEs are split or merged.
- Each cell is assigned to one IS. Some or all cells can switch to a different IS in response to instructions from the current IS.
- An active cell executes the instruction sent from its IS, while inactive cells wait until they are commanded to be active again.
- An IS may unconditionally activate all cells assigned to it.
- An IS may conditionally force certain cells assigned to it to become idle.

3. ASSOCIATIVE PROPERTIES:

- An IS can have all of its active cells execute an associative search. Active cells which perform a successful search are called responders and unsuccessful active cells are called non-responders. The IS may force either the set of responders or non-responders to be active. Previous sets of active cells can also be restored by some IS.
- An IS can select one arbitrary cell from the list of active cells.

4. GLOBAL OPERATIONS:

- An IS may compute the global AND or OR of a boolean value or the MAX or MIN of data in all active cells.
- An IS may broadcast a word to all its active cells.
- When an IS has only one active cell, it can either read a word from the memory of this cell or write a word to the memory of this cell.

5. CONTROL PARALLELISM:

- The collection of ISs operate as a MIMD computer and primarily employ control parallelism providing fork and join operations, transfer of control information, coordinate the network joining the cells, and manage the dynamic reallocation of idle cells.
- ISs are assumed to operate asynchronously.
- The ISs communicate control information and the transfer of active or inactive cells with each other only during a fork or join operation.
- The transfer of data between ISs is handled by reassignment of cells during forks and joins.

6. DATA ROUTING:

- A routing operation on a word held in each PE can be performed using a network that connects the PEs (ex. mesh, linear, hypercube). This routing allows combining of data sent to the same PE location involving an arithmetic or logical operation (ex. $+, max, min$). This involves a cost in time based on the PE interconnection network used.

3 Simulation of PRAM

This section presents algorithms for ASC to simulate PRAM and for PRAM to simulate ASC. Most notably the ASC to PRAM simulation has a lower bound when the PRAM algorithm uses the same order of shared memories as there are ASC ISs. A specific case is when there is only one IS on the ASC machine, an ASC simulation of a priority CRCW PRAM algorithm using only one shared memory can be completed in $\theta(1)$ time per cycle. Models that have networks connecting PEs with only a constant bound on the number of links per node *(bounded degree network)* can simulate priority CRCW PRAM cycle in $O(\log n)$ for only one shared memory or multiple shared memories with high probability. Thus by combining the one IS ASC simulation with existing methods that use networks, the resulting simulation has a lower bound of $\theta(1)$ when using a constant number of shared memories, and a probabilistic upper bound of $O(\log n)$ for certain logarithmic diameter bounded degree networks (like the hypercube)[20][21]. For any network this simulation has an upper-bound of $route(n)$, where $route(n)$ is the amount of time in terms of n it takes to perform a priority CRCW operation using the network. For example, a mesh network could perform this operation in $O(\sqrt{n})$[?][22][23][24][25][26][27][28][29][30][31][32].

When the *simulation* of a model is performed there are various operations to consider. For parallel models, the operations that need to be simulated are parallel execution of processors and the data movement between processors[33][34]. These operations are defined with a mapping of processor resources from one model to another and with the algorithms of operations that need to be simulated. When the time complexity of each operation performed in a cycle of simulation is divided by the complexity to perform the same operations on the machine being simulated, the maximum time to simulate any operation gives the slowdown of the simulation and also an indication of the relative powers of the two machines or how different they are [35][36][37][38][39][18]. The disparity in simulation times from one model to the other shows either how similar or dissimilar the 2 models are. Similar models should simulate each other in the same or near the same amount of time while dissimilar models require more time to simulate each other. The operations for simulation of the associative model (ASC) are specified in previously while a version of PRAM is defined below.

3.1 A Synchronous Definition of PRAM

A PRAM(n,m) machine is defined as a collection of n sequential (RAM) machines and a set of m global memories. Each RAM of the PRAM has a sufficient instruction set, a local memory and a specific address int the range $0...n-1$[40]. During one cycle of execution each processor executes the same instruction with different operands synchronously. The instruction can be a local computation, a read from a global memory, $0...m-1$, or a write into global memory, $0...m-1$[41]. Most all PRAM algorithms written are for synchronous PRAM.

It is assumed that one machine cycle takes a constant amount of time regardless of hardware requirements to build such a machine. There are 3 popular varieties to handle reading and writing conflicts: EREW, CREW, and CRCW. An EREW PRAM does not allow two or more processors to either read or write to the same global memory location concurrently. It is the most restrictive model. The CREW PRAM allows more than one processor to concurrently read from the same memory location, but only one PE may write to a global location at the same time. CRCW is the most powerful PRAM which permits both concurrent reading and concurrent writing. A diagram of PRAM is shown in Figure 3.

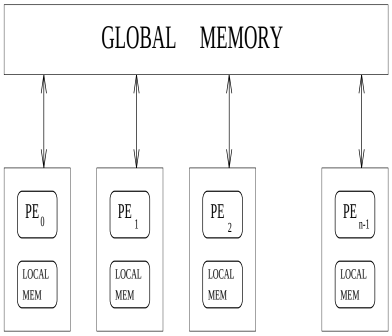

Figure 3: PRAM

There are several ways to handle concurrent writing, each possessing possibly a different computational power. COMMON CRCW requires that data written to the same location must all be of the same data value for the write to take place, otherwise an error occurs. RANDOM CRCW allows one of the PEs to write its value to global memory when several are writing to the same location. The PE that writes is chosen at random. PRIORITY CRCW allows the writing PE with the largest address to write its value when several PEs are writing to the same global memory. COMBINING CRCW is a very powerful writing model in that all PEs which write to the same location are combined by some arithmetic or logical operation such as addition. The hardware needed to implement any of these machines would at least be $O(\log n)$ in circuit depth and most likely, rather complex[2].

3.2 Why Simulate PRAM?

In terms of other models, one can think of ASC simulating PRAM where reads and writes are handled through the network at a specific cost based on the network used. Other operations such as broadcasting, searching, and reductions may have a lesser cost to perform. Thus, when writing an ASC program one should always be mindful that use of the network is slow and to use memory locality as much as possible to avoid extraneous network communications. Since common operations such as broadcasting can be efficiently implemented on many platforms, their use as an alternative to routing with the network is generally preferred and can be expressed very intuitively in an associative data parallel language[5].

Many PRAM algorithms map very well to ASC while some may need adjustment in terms of reducing communications at perhaps the cost of performing more computations, especially when considering *parallel slackness* which is defined as the ratio of data in a problem as compared to the number of processors. In other words there should be more data than processors in a sufficient amount to make a parallel algorithm run optimally in terms on the number of processors. This is usually due to processors communicating less and performing more local computations.

One algorithm that maps very well to ASC is the $O(\log n)$ expected time PRAM convex hull algorithm that assumes a random distribution of points[42]. It is shown in this section that an ASC machine with only one IS and no network can simulate priority CRCW PRAM with a constant number of shared memories in $O(1)$ time, and since the PRAM convex hull algorithm (assuming random points) uses only one shared memory for broadcasting, an ASC algorithm can easily be written to also run in $O(\log n)$ expected time with a high probability. This fact is verified by M. Atwah in his master's thesis[3] and elsewhere[13][12]. There are no doubt many other useful PRAM algorithms that also use only a constant number of shared memories, and any of these should have a corresponding ASC algorithm that executes in the same time as the best known priority CRCW PRAM algorithm[43]. Even without a PE network, ASC has intrinsic capabilities to perform some operations faster than PEs just connected by a bounded degree network (a network with a constant number of links per node). In other words, such a network could not even perform a broadcast operation in less time than the diameter of the network. For example a mesh needs $\theta(\sqrt{n})$ time to broadcast data, and a hypercube needs $\theta(\log n)$ time to broadcast, and neither can perform this operation any faster. A diagram of ASC is shown in Figure 2.

In the next subsections, algorithms for ASC to simulate synchronous PRAM will be investigated as well as a synchronous PRAM simulation of ASC. These two results will show the power of ASC relative to PRAM as well as provide a method to convert PRAM algorithms to ASC and ASC algorithms to PRAM. In terms of the simulation, the methods in this section place no restrictions on the number of processors or shared memory resources as long as the number of ASC processors is the same as the number of PRAM processors.

3.3 ASC Simulating PRAM

In order to understand the power of the ASC model, it should be compared to a well understood model such as PRAM[36][34][44]. Since more parallel algorithms have been written for PRAM than any other parallel model, it is important to establish an ASC(n,j) simulation of PRAM with minimal slowdown as this simulation allows numerous PRAM algorithms to be used on any machine that supports the ASC model. The principle focus of this section concerns simulating synchronous priority CRCW PRAM with n processors and m shared memories on a ASC(n,j) machine without a network. A hybrid algorithm that uses the presented ASC simulation method and known network simulations of PRAM provide a better solution. The ASC simulation of CREW and EREW PRAM are produced by noting that a priority CRCW PRAM can easily simulate CREW and EREW with constant time and memory costs[45].

Next it is shown how to simulate ASC(n,j) with a synchronous combining CRCW PRAM machine with n PEs and one shared memory. The combining PRAM is the most powerful of the synchronous PRAM models and has no trouble simulating one IS ASC in $O(1)$ time. An easy algorithm for combining CRCW to simulate ASC is given so that the times for EREW and CREW PRAMs to simulate ASC can be easily derived. Since EREW and CREW PRAM simulate the CRCW PRAM using the same number of PEs and shared memories with a known extra cost of $O(n/m + \log n)$, the extra time for these two models to simulate ASC are easily obtained after doing the combining CRCW PRAM simulation[45][46].

3.4 Operations Needed for Simulation

To simulate PRAM with any machine, three operations need to be handled: parallel execution of the PEs, reading from the shared memories, and writing to the shared memories. Here, priority CRCW PRAM with n PEs and m shared memory will be simulated by ASC(n,j). The ASC machine has the same number of PEs and it is assumed that an ASC PE has the same computational power as that of a PRAM PE. However, the PRAM reads and writes are highly parallel and complex in nature since all PEs may either simultaneously send or receive data using arbitrary memory locations. Thus all possible communication patterns need to be simulated. To solve this problem on ASC, it is first helpful to look at how to do permutation routing on ASC(n,j) for n data objects distributed among the n PEs. Since there are j possible data paths provided by the presence of the ISs, the time to route is dependent upon the number of ISs giving an $O(n/j)$ routing time using only ISs[47]. With a network connecting the PEs, the time to simulate a given PRAM with ASC is dependent upon the fastest possible routing scheme known for any given network. For example, if a mesh connects the PEs then it is well known that the PEs can simulate the reads and writes of an EREW PRAM in $O(\sqrt{n})$ for n PEs. This scheme becomes more complicated when concurrent reads and writes are allowed along with the number of shared memories being greater than the number of PEs.

3.5 Notation Used

For the remaining algorithms in this section, the following notation is used. The number of PEs is n, and the number of ASC ISs is j. There are m shared PRAM memories, and it is assumed that PRAM is synchronous. The term *priPRAM* will mean all PRAMs up to and including the power of *priority CRCW PRAM*, and the term *comCRCW* will denote all CRCW PRAMs at least as powerful as *combining CRCW PRAM*. The combining CRCW PRAM resolves write conflicts by combining data written to a shared memory with some operator (e.g. $+, -, /, *, MIN, MAX, AND, OR, ...$). The shared memories will be stored more or less evenly among the ISs since without a network this method is slightly more simple and is functionally equivalent to storing the memories in the PEs. The shared memories are stored in the PEs so either the PE network or the ISs can move the data.

3.6 ASC Simulation of Concurrent Read

The concurrent read operation of CRCW(n,m) can be simulated with ASC(n,j) in $O(min(n/j))$ time with high probability where the ASC machine does not have a network. Figure 4 shows the ASC algorithm. The PRAM shared memory is hashed in ASC's PEs, divided more or less equally between each PE. When PEs are concurrently reading, the reading process is simulated by having each IS collecting on average $O(n/j)$ read requests from the PEs. This step is accomplished in $O(n/j)$ with a high probability if assuming an optimal hashing function (uniform parallel hashing) is available. Then the ISs each process $O(n/j)$ read requests in $O(n/j)$ time reading the required data from the n PEs. This is possible because each IS connects to j strictly disjoint subsets of PEs for reading purposes, In the first step where the ISs read PE read requests, a PE is assigned to the IS that manages the memory locating in required PE partition.

assume ASC an PE has the power of a PRAM PE
MEM_t is stored in PE $Hash(t)$
***** Gather O(n) read requests to the j ISs ($O(n/j)$ avg. time)*****
1 : FOR all PEs and ISs do in parallel
2 : SET ALL PEs to ACTIVE
3 : IF (a PE is not READING) then SET the PE to IDLE
4 : ASSIGN each PE to IS which manages the MEMORY read REQUEST
5 : WHILE (any ($PEs\ are\ ACTIVE$))
6 : ALL ISs SELECT ARBITRARY active PE
7 : SELECT other PEs reading from same MEMORY
8 : INSERT READ request into an IS read-request LIST (PE,mem)
9 : SET these selected PEs to IDLE
***** Read O(n) data requests into j ISs ($O(n/j)$ avg. time)*****
10 : SET ALL PEs to ACTIVE
11 : ASSIGN each PE to IS which manages the MEMORY to be READ
12 : WHILE (any ($PEs\ are\ ACTIVE$))
13 : PROCESS NEXT entry in ISs read-request LIST (start at 1st)
14 : ISs read one Hashed(Memory) into read-request LIST
***** Broadcast read data into requesting PEs ($O(n/j)$ avg. time)*****
15 : IF (a PE is not READING) then SET the PE to IDLE
16 : ASSIGN each PE to IS which manages the MEMORY to be READ
17 : WHILE (any entries in IS lists are unprocessed)
18 : PROCESS NEXT entry in ISs read-request LIST (start at 1st)
19 : SELECT PEs reading from MEMORY location of this entry
20 : BROADCAST READ data into requesting PEs from IS LIST

Figure 4: $O(n/j)$ Algorithm to Simulate Concurrent Read of Synchronous CRCW(n,m) on ASC(n,j) with O(m/n) Memory per PE

3.7 Examining the MSIMD (ASC) Reading Algorithm

Figures 5, 6, 8 show an example of the major simulation steps of a concurrent read. The following steps referring to Figure 4 show the operations taken to simulate one concurrent read. First in line 1, all PEs and ISs perform operations in parallel. At line 2 all PEs are made active. Line 3 forces any PEs that are not reading to be temporarily inactive and no longer perform any operations. Line 4 makes each PE listen to the IS processor

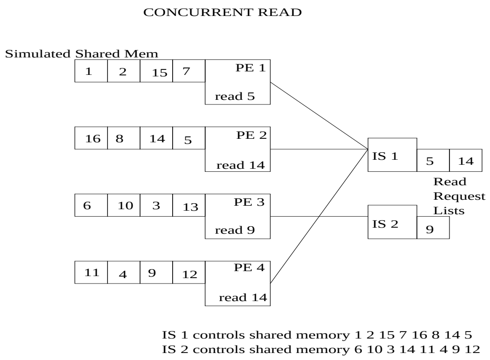

Figure 5: ASC Simulation of PRAM Concurrent Read: building of read request lists

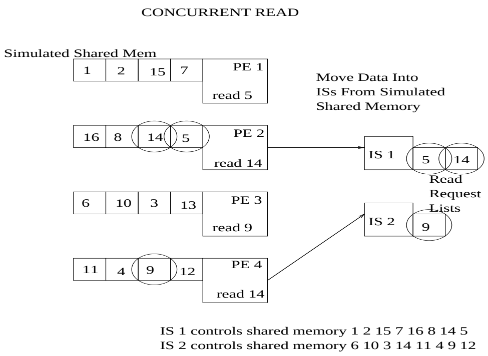

Figure 6: ASC Simulation of PRAM Concurrent Read: moving data to ISs

which manages the PE holding the memory location it wishes to read. In the $5th$ line, sequential iterations are performed until all PEs are inactive and thus have had their read requests saved in the ISs. Then at line 6 each IS selects an arbitrary PE out of all active PEs to save its read request in a list located at an IS. Each read request will be saved in only one list in one of the ISs where there is one list per IS. For each IS the selected PE's read address is compared with all other active listening PEs during line 7, and all PEs with the same read request are made active. Line 8 saves the read request (memory location, initial requesting PE) in a list in an IS. The selected PEs which are all reading from the same location, concurrently in fact, are made idle at step 9. This process continues until all PEs have their requests stored in some list in an IS. Since there are n PEs and j ISs, an optimal hashing scheme will yield $O(n/j)$ list entries per IS. If there there are only k, where $k \leq n$, the number of memories being read from, the time to complete this (and following steps) is $\theta(1)$. In an example shown in Figure 5, the read requests should be in the ISs distributed evenly.

Lines 10-14 bring data to be read from the PEs into the ISs, which afterwards lines 15-20 broadcast this data to the original requesting PEs which completes the concurrent reading algorithm. Continuing the line by line description, line 10 sets the PEs to be active again, line 11 assigns PEs to the ISs which manage their shared memories (each IS manages $O(n/j)$ PEs and therefore $O(m/j)$ memories), this assignment of PEs to ISs assumes an optimal parallel hashing function from PE addresses to IS addresses. Line 12 loops sequentially while any PEs are still active, and line 13 considers the next read request stored in the IS request list starting from the first entry in the list. At line 14 each IS reads data from the requested memory address in one of the PEs and

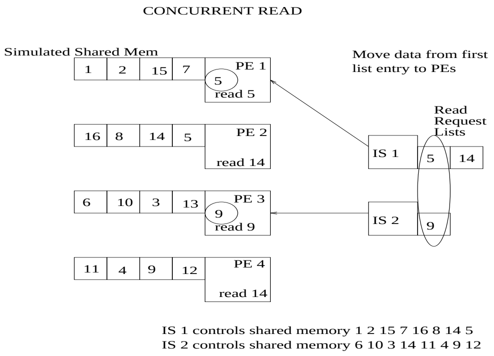

Figure 7: ASC Simulation of PRAM Concurrent Read: handle first list entries, and send data that was read to PEs

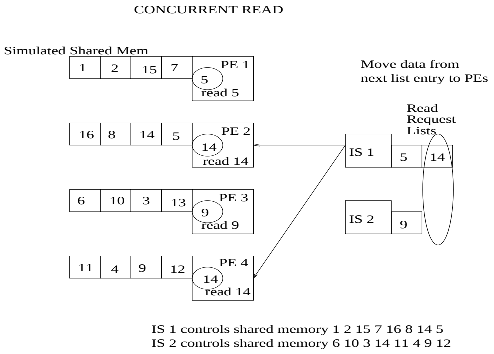

Figure 8: ASC Simulation of PRAM Concurrent Read: handle second list entries, and send data that was read to PEs

saves this data in the same location of the list (memory, location, initial requesting PE, data). See Figure 6 for an example of where the data is read.

The lines from 15-20 send this data back out to the PEs that are reading. Line 15 sets PEs idle if they are not reading. Line 16 assigns each active PE to the IS that manages the memory of that PEs original read request. A sequential loop is made in line 17 until all list entries are processed in the ISs. These lists could be empty if there were no read requests for memories that PE managed, or the lists could contain anywhere from a constant number of requests up to an average of $O(n/j)$ entries. Line 18 fetches the next entry in each IS list starting with the first. PEs that are reading at the memory location of this entry are selected in line 19, and then line 20 broadcasts the data to these PEs. All operations except the sequential loops are assumed to take a constant amount of time in ASC. Thus the limiting factor is the sizes of the read request lists created. Because there are n PEs, on average the lists should be $O(n/j)$ in size, and the entire read process takes $O(n/j)$ with high probability. If all PEs are reading from only k shared memories where $k \leq n$ then the size of the lists will be $O(k/j)$ and the time to complete a concurrent read is also $O(k/j)$. The following section shows that a priority concurrent write can be simulated in the same time as a concurrent read. Figures 8 and **??** show the data being moved out to the PEs.

3.8 ASC Simulation of Concurrent Write

In $O(n/j)$ time, ASC(n,j) can also simulate concurrent write of priority CRCW(n,m) with high probability, and the ASC algorithm is shown in Figure 9. Again, the shared memory of PRAM is stored in the PEs, divided roughly equally among each PE, and the memory addressing scheme is the same as when simulating concurrent reading, that is shared memories are hashed among the PEs where $O(n/j)$ PEs are managed by each IS. With concurrent writing the data needs to be moved from potentially all n PEs to the PEs containing the proper memory cells. If more than one PE is writing to the same cell, the priority rule states that the PE with the highest self address is the one allowed to write. This is handled by selecting the maximum PE that requests to write.

```
assume ASC an PE has the power of a PRAM PE
MEM_t is stored in PE Hash(t)
**** Gather O(n) write requests into IS lists (O(n/j) time) ****
**** Use REDUCE with MAX when > 1 PE wishes to write to same location ****
1 : FOR  all PEs and ISs do in parallel
2 :         SET ALL PEs to ACTIVE
3 :         IF (a PE is not WRITING)  then  SET the PE to IDLE
4 :         ASSIGN each PE to IS which manages the MEMORY request to be WRITTEN
5 :         WHILE  (any (PEs are ACTIVE))
6 :                   ALL ISs SELECT MAXIMUM active PE
7 :                   SELECT other PEs writing to same MEMORY
8 :                   For each IS REDUCE with MAX the selected WRITE requests
9 :                   INSERT reduced write data into an IS write-request LIST (PE,mem)
10 :                  SET these selected PEs to IDLE
**** WRITE O(n) data out to memories hashed in PEs (O(n/j) time) ****
11 :        SET ALL PEs to ACTIVE
12 :        ASSIGN each PE to IS which manages the MEMORY to be WRITTEN
13 :        WHILE  (any ENTRIES in write-request LIST remain)
14 :                  PROCESS NEXT entry in ISs write-request LIST (start at 1st)
15 :                  Each IS WRITEs one data into Hashed(Memory) location
```

Figure 9: $O(n/j)$ Algorithm to Simulate Concurrent Write of Synchronous Priority CRCW(n,m) on ASC(n,j) with O(m/n) Memory per PE

3.9 Examining the MSIMD (ASC) Writing Algorithm

Figures 10, 11, 12, 13 show an example of the major simulation steps of a concurrent write. The following steps show the operations for simulating a concurrent write as shown in Figure 9. In line 1, PEs and ISs start acting in parallel, and line 2 sets all PEs to active status. Line 3 temporarily deactivates PEs that are not performing a

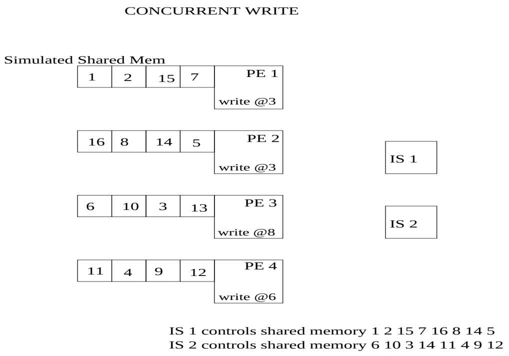

Figure 10: ASC Simulation of PRAM Concurrent Write: distribution of memories

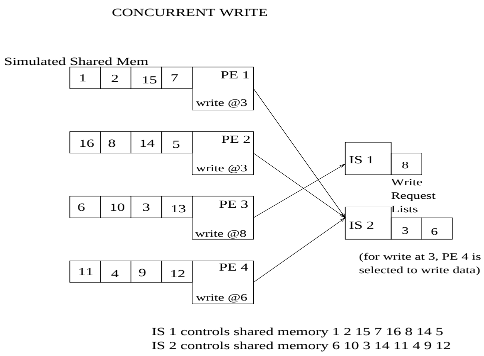

Figure 11: ASC Simulation of PRAM Concurrent Write: build write request lists

write operation. Line 4 assigns each PE to the IS that manages the memory location that a PE wishes to write. Line 5 loops until all write requests are considered. Inside the loop, line 6 selects an arbitrary active PE and then in line 7 all other active PEs writing to the same memory location are selected. The PE with the maximum self address is allowed to write its data in a priority fashion by perform a maximum reduction among the selected PEs in line 8. Line 9 inserts this data into a write request list in the ISs containing the PE address, the memory address, and the data. In line 10 the selected PEs, now having their priority write request considered, are made inactive. The list of write requests created is an average of $O(n/j)$ for each IS in size analogous to the read request lists formed in the concurrent read algorithm. Figure 11 shows the list of write requests built in an example.

Lines 11-15 write the data in the write request lists currently located in the ISs to the correct PEs. First, line 11 sets PEs back to active status. Line 12 assigns each PE to the IS which manages it. The algorithm then loops in line 13, each IS processing one list entry from their own lists until all are considered. Line 14 fetches the next entry in each ISs write request list starting with the first. Lastly, line 15 writes a word of data into the correct hashed memory location. Figures 12 and 13 show to where the data is moved from the ISs.

Much like the concurrent read algorithm, all statements except loops are assumed to take a constant time in the ASC model. The simulation of PRAM priority concurrent write takes the average time of the two loops each of which is based on the size of the lists created. If the memories are optimally hashed into the n PE, on average the list sizes created in each IS should be $O(n/j)$ and the algorithm should also run in $O(n/j)$ time. If

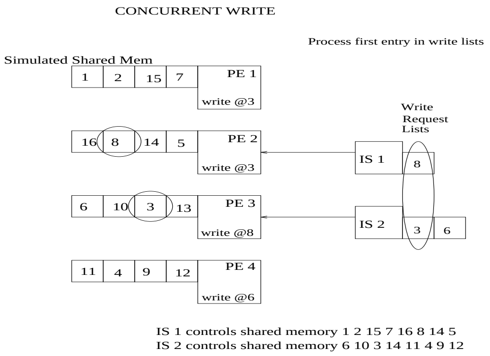

Figure 12: ASC Simulation of PRAM Concurrent Write: move data to write in 1st list entry

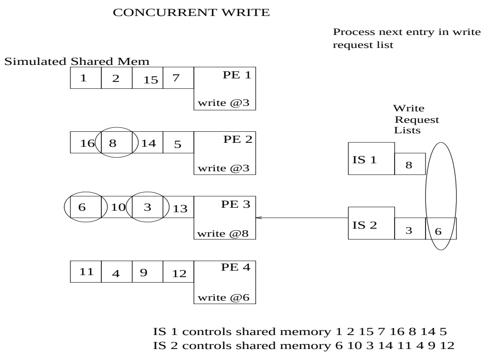

Figure 13: ASC Simulation of PRAM Concurrent Write: move data to write in 2nd list entry

$k \leq n$ PEs are being written into, then the list size will be $O(k/j)$ and the algorithm runs in $O(k/j)$.

4 Overview of ASC simulation of PRAM

The time to complete a priority concurrent write with ASC(n,j) simulating priority CRCW(n,m) is the same as that for completing a concurrent read. By the priority rule, multiple PEs writing to the same memory location are handled in one constant time step, using the ASC maximum reduction operation, so in this case the write operation will finish in $O(n/j)$ time. Again if there are reads or writes involving only k memories, where $0 \leq k \leq n$, then the time can be written as $O(k/j)$. This fact is useful for certain PRAM algorithms. In short, any n processor PRAM algorithm that requires only a constant number of shared memories can be simulated with an n processor, one IS ASC machine in constant time. An example of this is shown in a paper by M. Atwah where the PRAM algorithm to solve the convex hull problem for n randomly distributed points can be solved in $O(\log n)$ time with only a constant number of shared memories[3][12][13]. His paper presents a one IS ASC algorithm that also solves this problem in $O(\log n)$ time. There are no doubt many existing priority CRCW PRAM algorithms that are efficient using a constant number of shared memories. All of these algorithms can execute on a simple one IS ASC machine in the same time as PRAM.

By transitivity, ASC(n,j) can simulate CREW(n,m) and EREW(n,m) PRAM in the same time as it simulates priority CRCW(n,m)[46][42]. Since priority CRCW easily simulates CREW and EREW with no extra resources, ASC can also simulate them with $O(n/j)$ extra time per cycle using the CRCW result.

Also, if the number of PRAM shared memories equals the number of instruction steams, $m = j$, then thus ASC(n,j) can simulate priority CRCW(n,j) in constant time. Furthermore, it may be interesting to note that no matter how much memory or ISs are present, as long as all PEs write to a subset of memory cells that is constant in size, a step of simulation also takes constant time. So it is possible for the simulation to run in $\theta(1)$ extra time per machine cycle in the best case.

4.1 Considerations when ASC has a PE Network

If the ASC machine that is simulating PRAM has an interconnection network between the PEs, traditional methods can be used to simulate PRAM with the network alone, not using the ISs to route data. The simulation of PRAM with parallel machines that have PEs with local memories and a network has been well studied. The concurrent read and write problem is essentially a routing problem on the network. For instance simulation methods that use a mesh network exist where the time to simulate each cycle is $O(\sqrt{n})$. This is fact is the same time it takes to sort n data items on a mesh with n processors. Simulation methods for other networks exist with a bounded number of connection per node (bounded degree networks). The fastest simulates PRAM with a high probability in $O(\log n)$ time, while others simulate PRAM in average time $O(D)$ where D is the diameter on a wide variety of constant and non-constant degree networks. These networks include the cube-connected cycles, butterfly, shuffle-exchange, mesh, hypercube, the star, etc[36][34]. The shared memories are generally hashed amongst the local memory of the n PEs similarly to the ASC simulation method.

However, even if the PRAM being simulated has a constant number of shared memories (ex. 1 shared memory), the network simulation of PRAM time is still upper bounded by the diameter of the network, or the time it takes one PE to send a message to another PE. The priority CRCW ASC simulation of PRAM with a constant number of shared memories requires $\theta(1)$ time which is better than the network methods cited above.

An ASC simulation of PRAM for ASC with a network should be a hybrid algorithm consisting of a simulation algorithm considering only the network, and the ASC method as presented in this section. The algorithm proceeds by performing one network routing step, and then alternately one ASC routing step. Whenever one of the methods finishes a single cycle of simulation, the other algorithm is terminated, and the next cycle is then started. At worst if both methods finish at the same time, the simulation will take only some constant factor more time, and will be no worse than the time for the *fastest* method to complete. At best when considering $k \leq n$ shared memories, this hybrid algorithm can simulate a step of PRAM in $O(min(k/n, route(n))$ where $route(n)$ is a function of the number of PEs(n) it takes to simulate on cycle of PRAM using the network, and k/n is the time to simulate PRAM using the presented ASC method. The ASC method has the capability of improving the fastest simulation of PRAM time over a network, even with the one IS ASC machine. The lower bound of this hybrid algorithm is $\theta(1)$ for a constant number of shared memories and is no worse than $O(route(n))$ using a network simulation method. This is interesting, since reexamining the PRAM algorithm for convex hull on random data points considering the fastest known network method to simulate this algorithm on a logarithmic diameter network (e.x. a hypercube, cube-connected cycles, or butterfly), takes $O(\log n)$ simulation steps each of $O(\log n)$ time for a total execution time of $O(\log^2 n)$. Whereas the ASC method alone of the ASC-Network hybrid method simulating PRAM can execute this algorithm in the optimal $O(\log n)$ time, an improvement over existing network algorithms.

5 The Obvious PRAM Simulation of ASC

This section presents a way to simulate ASC(n,j) with combining CRCW(n,m) and is included for completeness even though the algorithm is simple in order to provide some way to compare the two models. Combining CRCW PRAM *(cCRCW)* is used since it simplifies implementing the ASC reduction operators of AND, OR, MAX, and MIN by having the power to make such combinations. The extra time that other PRAMs such as EREW and CREW take to simulate ASC are found by a transitive argument where a weaker PRAM simulates cCRCW and then the following cCRCW algorithm is used to simulate ASC. The product of extra time needed for both simulations gives the time for the weaker PRAM to simulate ASC.

The n PRAM PEs perform the same operations that the ASC PEs can perform, and the IS information for IS_k is stored in $PE_{k\ MOD\ n}$ where $0 \leq k \leq j$. The $PE_{k\ MODn}$ will simulate the operations of instruction stream k, and it is assumed that each PRAM PE also has the power of an IS.

However, since ASC ISs are asynchronous and synchronous PRAM PEs are not, the execution of all ISs is handled iteratively in j steps. The IS information is distributed in the set of PEs to evenly balance memory used for IS simulation. Furthermore, since j steps must be taken regardless, all control communication between ISs finished in $O(j)$ time by having each PE with an IS write and read synchronization information to a j sized array stored within a single shared memory. This can be accomplished in 2 main steps. In the first step the j PEs write synchronization information, and during the second step each PE simulating an IS may read information left by other ISs. This is all completed in $O(j)$ time.

The algorithm below in Figure 14 handles all other necessary aspects of the cCRCW simulation of ASC. All crucial operations are combined into one algorithm where a case statement is used to perform the command that an IS is issuing. The outermost loop cycles through the j instruction streams iteratively, while the statements on the inside take $O(1)$ time to complete on the cCRCW. Also, only one shared memory is needed to complete the simulation so one record of data in memory location 0 is used for all data movement.

5.1 Examining Obvious PRAM simulation of ASC

The following is a description of what each step in the algorithm does. The loop in line 1 is used to sequentially process the instructions of the j ISs. PE_0 can handle this outer loop. During line 2 the PE address of the next read or write location from IS_k is written to shared memory 0, and the iteration number which is the current IS address is also saved in shared memory 0. Line 3 states that all PEs listening to IS_k are made active, and an operation will be performed among all these listening PEs in the following CASE statement. Line 4 is performed if IS_k is writing a single value to a PE, where first the data must be sent from the PE emulating IS_k to shared memory, and then the data is sent to the listening PE whose self address matches the address the IS indicated. When IS_k wishes to read a single value from a PE, line 5 is carried out, and the data from the one PE that the IS is reading from is sent to shared memory. This value is then read by the PE simulating IS_k. Line 6 simulates ASC reduction, combining a data value from all active PEs listening to IS_k into shared memory using the power of cCRCW. This combined value is then read by the PE containing IS_k. Broadcasting is handled by line 7. To broadcast a value from IS_k to all listening PEs, the value is written to shared memory from the PE simulating IS_k, and then the data is read in parallel by all listening PEs by the power of concurrent read PRAM. Furthermore, during line 8, instructions from the IS are broadcast to all listening PEs in the same way

Assume each PRAM PE has as much local memory as an ASC PE.

1 : for $k = 0$ to $j-1$ do

2 : $MEMaddr_0 = PEstreamAddr_k$

$MEMstream_0 = k$

3 : for all PE_i do in parallel

if $(PElisten_i = MEMstream_0)$

case (Operation) of

4 : "ISwriteOne": $MEMdata_0 = PEstreamData_k$

if $(PEaddr_i = MEMaddr_0)$

$PEdata_i = MEMdata_0$

5 : "ISreadOne": if $(PEaddr_i = MEMaddr_0)$

$MEMdata_0 = PEdata_i$

$PEstreamData_k = MEMdata_0$

6 : "ISreduce": $MEMdata_0 = PEdata_i$ combine

with AND-OR-MIN-MAX

$PEstreamData_k = MEMdata_0$

7 : "ISbroadCast": $MEMdata_0 = PEstreamData_k$

$PEdata_i = MEMdata_0$

8 : "ISsetInstruction": $MEMinstruction_0 = PEstreamInstruction_k$

$PEinstruction_i = MEMinstruction_0$

9 : "LocalCommand": Perform PE_i localized execution

Figure 14: $O(j)$ Algorithm to Simulate ASC(n,j) with combining CRCW(n,m)

that line 7 broadcasts data values. It is included as a separate line for completeness. Lastly, line 9 executes local ASC commands that involve no data movement between PEs and ISs so all of these operations can be simulated directly with PRAM's PEs. Common local computations may involve arithmetic and logical operations, setting PEs active or inactive, or changing the IS channel address that the PE is currently listening.

Since each inside the main sequential loop takes constant time on the cCRCW(n,m), the simulation of ASC(n,j) takes $O(j)$ extra time per cycle. If transitive simulations are used where either EREW(n,m) or CREW(n,m) simulate cCRCW(n,m) in $O(n/m + \log n)$ extra time per cycle, then EREW(n,m) or CREW(n,m) can simulate ASC(n,j) in no worse than $O(j(n/m + \log n))$[46]. These times for EREW and CREW to simulate cCRCW are optimal[46]. The time for EREW and CREW to simulate ASC are also optimal by the argument that it takes either of these models $O(n/m + \log n)$ time to perform any of the data movement operations (MIN, MAX, AND, OR, BROADCAST) with n PEs and m shared memories [46].

6 Conclusions of ASC-PRAM simulations

It has been shown how the ASC model can simulate PRAM, and also how PRAM can simulate ASC. The results in this section allow the number of ISs, and shared memories to be unbounded to allow wide comparisons of the models. Table 1 summarizes all the current simulation results for the indicated models. If the current technology in terms of hardware implementation is taken into account, then these resources would definitely be restricted, both for ASC *and* PRAM. The ASC model is intended to encompass classes of machines that are buildable today using reasonable resources. Also, the simulations are left more open to show the power of the ASC model in theoretical terms. Table 2 shows the comparison of PRAM and ASC when the amount of shared memory is proportional to the number of ISs or $m = O(j)$. When this is done ASC(n,m) simulates nPRAM(n,m) in constant time with a constant amount of space, while cCRCW(n,m) simulates ASC(n,m) in $O(m)$ time with $O(m/n)$ extra space required per PE. The actual number of ASC ISs implementable is no doubt a slow growing function of n, yet even ASC with one IS has a great deal of computational power in practice[5].

SIMULATE	With		Extra Time	Extra Memory
ASC(n,j)	RAM		$O(n+j)$	$O(n+j)$
priPRAM(n,m)	ASC(n,j)		$O(n/j)$	$O(m/j)$per IS $O(m/n)$per PE
ASC(n,1)	comCRCW(n,m)		$O(1)$	constant
ASC(n,1)	CREW(n,m)		$O(n/m + \log n)$	constant
ASC(n,1)	EREW(n,m)		$O(n/m + \log n)$	constant
ASC(n,j)	comCRCW(n,m)		$O(j)$	$O(j/n)$per PE
ASC(n,j)	CREW(n,m)		$O(\frac{jn}{m} + j * \log n)$	$O(j/n)$per PE
ASC(n,j)	EREW(n,m)		$O(\frac{jn}{m} + j * \log n)$	$O(j/n)$per PE

Table 1: Overview of Simulation Times

SIMULATE	With		Extra Time	Extra Memory
priPRAM(n,m)	ASC(n,m)		$O(1)$	constant
ASC(n,m)	cCRCW(n,m)		$O(m)$	$O(m/n)$per PE
ASC(n,m)	CREW(n,m)		$O(n + m * \log n)$	$O(m/n)$per PE
ASC(n,m)	EREW(n,m)		$O(n + m * \log n)$	$O(m/n)$per PE
priPRAM(n,1)	ASC(n,1)		$O(1)$	constant
ASC(n,1)	cCRCW(n,1)		$O(1)$	$O(1)$per PE
ASC(n,1)	CREW(n,1)		$O(n)$	$O(1)$per PE
ASC(n,1)	EREW(n,1)		$O(n)$	$O(1)$per PE

Table 2: Comparative Simulation Times for PRAM and ASC when $m = cj$ (c is a constant), and for $j = m = 1$

The main goal and future direction of this work is to provide a theoretical foundation for a parallel programming system (ASC) that can be implemented on virtually any type of machine, sequential or parallel, tightly bound or loosely coupled, and to show that ASC has the power to execute well known PRAM algorithms, some of which optimally. It is hoped that this model perhaps implemented as a data parallel compiler with an underlying system that handles the data parallelism could be constructed such that parallel programs are portable to existing and future machines in such a way that the masses of programmers accept the model as easily as they accept classic sequential programming, thus finally creating a bridge to practical useful parallel programming.

References

[1] J. Potter J. Baker S. Scott A. Bansal C. Leangsuksun C. Asthagiri. Asc: An associative computing paradigm. *IEEE Computer*, pages 19–25, November 1994.

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

[3] Maher M. Atwah. Computing the convex hull on the associative model. Master's project, Kent State University, Math and Computer Science (MSB), August 1994.

[4] Khin Fat Hioe. Asprol (associative programming language). Master's project, Kent State University, Math and Computer Science (MSB), August 1986.

[5] J.L. Potter. *Associative Computing — A Programnming Paradigm for Massively Parallel Computers.* Plenum Publishing, N.Y., 1992.

[6] Chandra R. Asthagiri. *An Associative Parallel Compiler for an Associative Computing Language.* Phd thesis, Kent State University, Math and Computer Science (MSB), August 1991.

[7] WAVETRACER. *The MultiC Programming Language.* Preliminary Documentation, Jan 6th, 1991.

[8] Y. Ben-Asher D. Peleg R. Ramaswami A. Schuster. The power of reconfiguration. *Journal of Parallel and Distributed Computing*, 13:139–153, 1991.

[9] B. Svensson C. Fernstrom, I. Kruzela. *LUCAS Associative Array Processor.* Springer-Verlag, Berlin-Heidelberg, 1986.

[10] Bruce K. Hiller and David Elliot Shaw. Execution of ops5 production systems on a massively parallel machine. *Journal of Parallel and Distributed Computing*, (3):236–268, 1986.

[11] Mary C. Esenwein. Parallel string matching algorithms using associativve computing and mesh with multiple broadcast. Master's project, Kent State University, Math and Computer Science (MSB), 1995.

[12] Selim Akl Maher Atwah, Johnniw Baker. An associative implementation of classical convex hull algorithms. *Proceedings of the Eighth IASTED International Conference on Parallel and Distributed Computing and Systems*, pages 435–438, 1996.

[13] Selim Akl Maher Atwah, Johnnie Baker. An associative implementation of graham's convex hull algorithm. *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems*, pages 273–276, 1995.

[14] G. Steele Jr. W. D. Hillis. Data parallel algorithms. *Communications of the ACM*, 29(12):1170–1183, December 1986.

[15] D. Culler R. Karp D. Patterson A. Sahay K.E. Schauser E. Santos R. Subramonia n T. von Eicken. Logp: Towards a realistic model of parallel computation. *Proceedings of the ASC SIGPLAN Symposium on Principles and Practice of Parall el Programming*, pages 235–261, 1993.

[16] Eunice Santos Richard Karp, Abhijit Sahay. Optimal broadcast and summation in the logp model. Technical report, University of California, Berkeley, 1993.

[17] G.C. Fox. What have we learnt from using real parallel machines to solve real problems? Technical report, Caltechreport C3P-522, Dec 1989.

[18] Tom Blank and John R. Nickolls. A grimm collection of mimd fairy tales. In *Proceedings of the Fourth Symposium on the Frontiers of Massively Parallel Computation*, pages 448–457, McLean, VA, October 19-21, 1992. IEEE Computer Society Press.

[19] A. Falkoff. Algorithms for parallel search memories. *J. Associative Computing*, pages 488–511, March 1962.

[20] V. Prasanna Kumar C. Raghavendra. Permutations on illiac iv-type networks. *IEEE Trans. on Comp.*, C-35(7):662–669, July 1986.

[21] S. Orcutt. Implementation of permutation functions in illiac iv-type computers. *IEEE Trans. on Comp.*, C-25(9):929–936, September 1976.

[22] S. Sahni D. Nassimi. An optimal routing algorithm for mesh-connected parallel computers. *Journal of the ACM*, 27(1):6–29, Jan 1980.

[23] F. Leighton. Average case analysis of greedy routing algorithms on arrays. *Proc. 2nd ACM Symp. on Parallel Algorithms and Architectures*, pages 1–10, 1990.

[24] H. Lang M. Schimmler H. Schmech H. Schroeder. Systolic sorting on a mesh-connected network. *IEEE Trans. on Comp.*, C-34(7):652–658, July 1985.

[25] M. Kunde. Lower bounds for sorting on mesh-connected architectures. *Acta Informatica*, 24(2):121–130, April 1987.

[26] C. Seitz W. Dally. Deadlock free routing in multiprocessor interconnection networks. *IEEE Trans. on Comp.*, C-36(5):547–553, May 1987.

[27] Prabhakar Raghavan. Robust algorithms for packet routing in a mesh. In *Proceedings of the 1989 Symposium on Parallel Algorithms and Architectures*, pages 344–350, Santa Fe, New Mexico, 1989. ACM.

[28] T. Tensi M. Kunde. Multi-packet-routing on mesh connected arrays. In *Proceedings of the 1989 Symposium on Parallel Algorithms and Architectures*, pages 336–343, Santa Fe, New Mexico, 1989. ACM.

[29] I. Tollis T. Leighton, F. Makedon. A 2n-2 step algorithm for routing in an n by n array with constant size queues. In *Proceedings of the 1989 Symposium on Parallel Algorithms and Architectures*, pages 328–335, Santa Fe, New Mexico, 1989. ACM.

[30] Qian Ping Gu and Jun Gu. Routing algorithms on a mesh-connected computer. In *Proceedings of the Third Symposium on the Frontiers of Massively Parallel Computation*, pages 524–527, College Park, MD, October 19-21 1992. IEEE Computer Society Press.

[31] Torsten Suel. Permutation routing and sorting on meshes with row and column buses. *Parallel Processing Letters*, 5(1):63–80, 1995.

[32] C. Weems A. Krikelis. Associative processing and processors. *IEEE Computer*, pages 12–17, November 1994.

[33] R.A. Heaton J.M. Jennings, E.W. Davis. Comparative performance evaluation of a new simd machine. In *Proceedings of the Third Symposium on the Frontiers of Massively Parallel Computation*, pages 255–258, College Park, MD, October 8-10, 1990. IEEE Computer Society Press.

[34] Yonatan Aumann and Assaf Schuster. Deterministic pram simulation with constant memory blow-up and no time-stamps. In *Proceedings of the Third Symposium on the Frontiers of Massively Parallel Computation*, pages 22–29, College Park, MD, October 8-10, 1990. IEEE Computer Society Press.

[35] J. Zhang R. Lin, S. Olariu. Simulating enhanced meshes with applications. *Parallel Processing Letters*, 3(1):59–70, 1993.

[36] S. Rajasekaran M. Palis. Packet routing and pram emulation on star graphs and leveled networks. *Journal of Parallel and Distributed Computing*, (20):145–157, 1994.

[37] J. Trahan R. Vaidyanathan. Optimal simulation of multidimensional reconfigurable meshes by two-dimensional reconfigurable meshes. *Information Processing Letters*, (47):267–273, October 1993.

[38] R. Staraman A. Rosenberg, V. Scarano. The reconfigurable ring of processors: Efficient algorithms via hypercube simulation. *Parallel Processing Letters*, 5(1):37–48, 1995.

[39] F. Annexstein M. Baumslag M. Herbordt B. Obrenic A. Rosenberg C. Weems. Achieving multigauge behavior in bit-serial simd architectures via emulation. In *Proceedings of the Third Symposium on the Frontiers of Massively Parallel Computation*, pages 186–195, College Park, MD, October 8-10, 1990. IEEE Computer Society Press.

[40] S. Fortune and J. Wyllie. Parallelism in random access machines. *Proceeding of the ASC Symposium on the Theory of Computing*, pages 114–118, 1978.

[41] M.J. Quinn. *Parallel Computing;: Theory and Practice*. McGraw Hill, 1994.

[42] Joseph JaJa. *An Introduction to Parallel Algorithms*. Addison-Wesley, Reading, Massachusetts, 1992.

[43] B.F. Wang Gen-Huey Chen. Deriving algorithms on reconfigurable networks based on function decomposition. *Theoretical Computer Science*, 120:215–227, 1993.

[44] Paraskevi Fragopoulou. On the comparative powers of the 2d-parbus and the crcw-pram models. *Parallel Processing Letters*, 3(3):301–304, 1993.

[45] Joseph JaJa. *An Introduction to Parallel Algorithms.* Addison-Wesley, Reading, Massachusetts, 1992.

[46] John Rife, editor. *Synthesis of Parallel Algorithms.* Morgan Kofmann Publishers, San Mateo, CA, 1993.

[47] D. R. Ulm. The power of asc: Comparisons and algorithms (in preparation). Technical report, Kent State University, 1996.