Streaming PRAM

Darrell R. Ulm
Department of Computer Science
University of Akron
dulm@cs.uakron.edu

Michael Scherger
Department of Computer Science
Kent State University
mscherge@cs.kent.edu

Abstract

Parallel random access memory, or PRAM, is a now venerable model of parallel computation that that still retains its usefulness for the design and analysis of parallel algorithms. Parallel computational models proposed after PRAM address short comings of PRAM in terms of modeling realism of actual machines. In this work, we propose a multiple instruction stream partitioned PRAM, or "stream PRAM." This model embodies the reality of a small number of parallel processors, each with local memory (which could also be small), where a problem is generally evenly distributed among all processing elements. Actual hardware configurations limit the number of shared memories which can be efficiently implemented. By allowing each shared memory to also act as an independent instruction stream, more functionality is possible with a small extra cost. The additional instruction streams provide limited asynchronous abilities and offer the flexibility of a reconfigurable network as well as allowing the processing elements to perform independent actions. Because the proposed stream PRAM allows variable sizes for processors, memory, and problem sizes, it is valuable for present as well as future parallelism.

1. Introduction

We suggest the addition of multiple instruction streams and performance parameters to the limited resource PRAM model which can also be called partitionable PRAM (PPRAM) [2]. We call the new model *stream PRAM,* and this model intends to embody a superset of limited resource PRAM.

Limited resource PRAM (or PPRAM) models the bottleneck of shared memory. The PPRAM processors each have a local memory, which closely represents the reality of most of parallel computing done today [2]. PPRAM allows variable sizes for processors, memory, and problem sizes. It has been stated that by parameterization of each operation in PPRAM based on actual hardware measures, both scheduling of processors and run-time prediction are possible [2]. Models such as BSP and LogP contain this feature [9][30]; however PRAM contains a wealth of algorithmic research [3]. The authors believe that stream PRAM is a model suitable for fast algorithm design for both theory and practice and is compatible with many existing parallel systems and languages.

For a stream PRAM algorithm to execute on real hardware without shared memory, we can simulate the m shared memories with p processors and the network between the processors. There are many ways to do this efficiently, and even if it is done with embedded calls to popular message passing libraries such as MPI or PVM, communication, average time parameters are obtainable [5][17][18]. Furthermore, processor and I/O benchmarks are easily discovered for any given cluster. Given these values along with the stream PRAM worst or average case times for computational and communication complexity, a programmer or compiler can select the best number of processors and determine with some accuracy, the running time of most well behaved programs [2]. Experimental results are forthcoming.

1.1. Simple PRAM

There are three popular PRAM varieties to handle reading and writing conflicts: EREW, CREW, and CRCW [3]. 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 processing element (PE) 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 that permits both concurrent reading and concurrent writing. This paper deals mainly with priority CRCW although the EREW and CREW simulations are also derived. Priority CRCW allows the PE with the largest address to write its value when several PEs are writing to the same global memory. With combining

CRCW, all PEs, which write to the same location, are combined by some arithmetic or logical operation such as addition [3].

1.2. Multiple instruction stream shared memory partitioned PRAM (or stream PRAM)

This work proposes a PRAM model that has a limit on the shared memories. We call the model stream PRAM, which is a multiple instruction stream, partitioned PRAM. It is based on the PPRAM model by Agbaria, Ben-Asher, and Newman [2], and it is also derived from the MASC model defined by Potter and others [19][20]. The PPRAM model shows concisely how the memory bottleneck affects the speedup of algorithms, and the MASC model consistently defines multiple instruction streams in terms of modeling and hardware. The main contribution of the stream PRAM model is to allow multiple instruction streams to coordinate with PEs. While this may seem somewhat odd in terms of traditional PRAM, the extra hardware required is minimal, and what is gained is the flexibility of a reconfigurable network as well as MIMD-like capabilities [16][24][28][31][32]. The hardware to actualize the ISs of stream PRAM may vary. Architectures such as the forthcoming "Cell" (by IBM, Sony, and Toshiba) ought to execute a stream PRAM algorithm with the limitations such as local memory size set accordingly. While it is possible to allow each PE to act as an independent asynchronous processor (Asynchronous PRAM), the aim of stream PRAM is to specify the number of different instructions being issued on the order of the number of shared memories. It is arguable that a small number of instruction streams is feasible to implement by either real hardware or by simulation on clusters with message passing [5][17][18][21][22]. Figure 1 shows the hierarchical evolution of the new model.

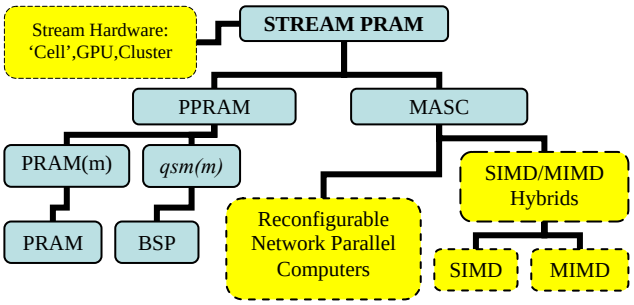

Figure 1: Family tree of stream PRAM (dashed boxes represent hardware)

We define a stream PRAM(p,m) machine as a collection of p sequential (RAM) machines and a set of m global shared memories that can also act as independent instruction streams. Each RAM of the PRAM has an instruction set, a local memory, and a specific address in the range *[0..p-1)*. During one cycle of execution, each processor executes one of the m instructions on local data. An instruction can be a local computation, a read from a global memory *[0..m-1],* or a write into global memory *[0..m-1].* Furthermore, it is assumed that $m \le p \le n$ where n is the size of the problem being solved. Although this inequality is not necessarily required, it is reasonable to state in terms of the limitations of today's hardware [2][7][12][26]. Most problems of size n solved on a stream PRAM will be split roughly into n/p sized data chunks, and each of the p RAM processors will work on their own data sequentially. Figure 2 shows a representation of the stream PRAM.

We shall assume there is some network connecting the PEs to the shared memories/instruction streams. In terms of algorithmic work, we may either assume that this network takes *O(1)* time, or that the network has a latency to perform the various operation. The stream PRAM can employ any of the exclusive or concurrent paradigms for reading and writing: EREW, ERCW, CREW, and varieties of CRCW. We also note that data I/O occurs through the shared memories. While in a real architecture, data may enter through a network perhaps from a large slower memory, DRAM for example. By routing the data through the shared memories, we can represent this bandwidth limitation. Furthermore, in the case that the local memories of PEs of real hardware implementations are small, that is, when all the data for a program it too large even after distribution across the PEs, then data is streamed through the shared memories to the PEs as the algorithm is executing. In fact for a wide bus architecture where there are few PEs, data could stream to the PEs directly, with even more ISs than the number of shared memories [24].

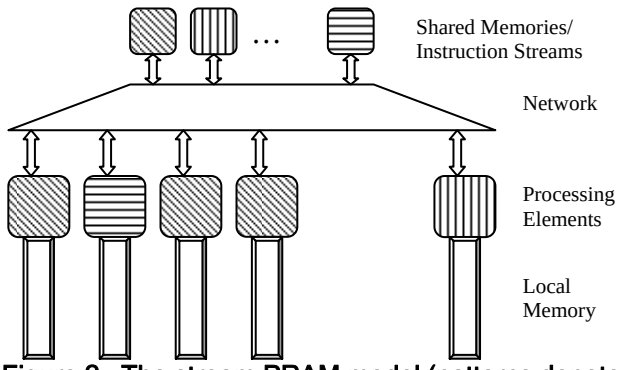

Figure 2. The stream PRAM model (patterns denote IS-PE partitions)

2. Related work

Here we summarize some of the related models. Notably, PPRAM and MASC figure prominently into the

design of stream PRAM. The PPRAM model defined by Agbaria, Ben-Asher, and Newman is derived from earlier variations of PRAM [2]. Vishkin and Wigderson investigated a PRAM, usually called PRAM(m), where $m < p$, strictly, and the input can be read concurrently by all processors from a global ROM. Later research showed a $\theta(\frac{n \log m}{m})$ sort on PRAM(m) [2]. The primary difference between PRAM(m) and PPRAM is the ability of PRAM(m) to access data in $O(1)$ whereas PPRAM and stream PRAM must load in the data into each PEs local memory [2]. Also, PPRAM is somewhat similar to a shared-memory variant of BSP called *qsm(m)*. Differences include the BSP-like ability of *qsm(m)* to compute in super-steps, as well as that fact the PPRAM allows all variants of PRAM, while the memory access of *qsm(m)* appears to be EREW only [12]. The focus of stream PRAM is to combine the scalability/speedup analysis features of the well-defined PRAM, along with the flexibility of using many instruction streams. In addition, the stream PRAM has the some abilities of the Asynchronous PRAM limited by the number instruction streams.

2.1. PPRAM Background

The PPRAM model is capable of computing several operations with a reasonable amount of speedup considering the limitations made by the model, and thus also the hardware. The work "Communication-Processor Tradeoffs in Limited Resource PRAM" shows that for many important problems, the time complexity scales exclusively with either the number of PEs or the amount of shared memory [2]. In the rest of this section we summarize some of the results from [2]. They showed that for summation, routing, k-selection, Boolean threshold, and list-reversal, that the time is $O(n/p + p/m)$, where the scaling is occurring either in terms of *m* or *p*, if the logarithmic factors are ignored [2]. However, some problems can be only be solved in approximately $O(n/m)$ eliminating factors of log. For the *O(n/m)* time complexity, the number of shared memories is the communication bottleneck when *m* is strictly smaller than *p [2]*. The work cited also notes that a PPRAM(*p,1*) simulates a PPRAM(*p,m*) in a simple fashion with a deterministic overhead of *O(m)* [2]. The cost to simulate a CRCW-PRAM(p,p^c), where $c \geq 1$ is some constant with a CRCW-PPRAM(p,m) is $O(\frac{p}{m} \log\log\log(p)(\log^* p)^2)$ with a high probability [2].

We shall use the notation of PPRAM(*n,p,m*) for solving a problem of size *n* with *p* processors and *m* shared memories. Given this, summing up *n* memories on PPRAM(*n,p,m*) takes $\theta(n/p + p/m + \log m)$ time with the upper bound being true for the CREW model [2]. A EREW-PPRAM(n,p,m) can solve integer sorting, and *k-k* routing in $O(\frac{n}{m} \min(\log p, 2^{\frac{\log m}{\log k}}))$ time where $k = n/p$ [2]. The list reversal problem requires $\theta(n/m + \log m)$ time, when $m < n/p$, for the priority EREW-PPRAM(*n,p,m*) model [2]. Lastly, *k*-selection is solvable with a PPRAM(*n,p,m*) in $O(n/p + (p/m + \log \frac{p}{m} \log m) \log \frac{n}{p})$ steps [2]. When implementing algorithms on clusters or other parallel machines, the authors of [2] indicate that for the given hardware, an effective number of shared memories can be found with measurement.

2.2. MASC Background

This section describes the associative model of computation presented in the IEEE Computer article "MASC: An Associative Computing Paradigm," which is based on work done at Kent State University [19]. Also, see [1][4][6][10][15][16][20][23][24][25][26][27][28][29][31][32][33]. The vast majority of existing platforms already efficiently support MASC, and there is an actual one IS language called ASC [14][20]. Currently work by D. R. Ulm is underway to build a C++ data-parallel class library, called *ZIPPAR* (downloadable from *www.sourceforge.net*), which is eventually to incorporate MPI and PVM for communication so that the stream PRAM paradigm, using data-parallel methods, will be easily portable to many platforms.

A frequent criticism of SIMD programming is that many processing elements (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 instruction streams (ISs) [19]. The associative model (MASC) has an array of PEs and one or more ISs that each broadcast their instructions to the mutually exclusive sets in a partition of the PEs. Here we define partition as the collection of disjoint sets.

Most applications require a small number of ISs in comparison to the number of PEs (with no firm restrictions). An MASC machine with *j* ISs and *n* PEs is written as MASC(*n, j*). Each PE (or cell) has a local memory, and MASC locates objects by content or location in the combined local memory of the PEs [20]. This is accomplished by searching a specified field of each PE for a given data item. Each PE is capable of performing local arithmetic and logical operations and the usual functions of a sequential processor other than issuing instructions. Figure 3 shows the MASC model.

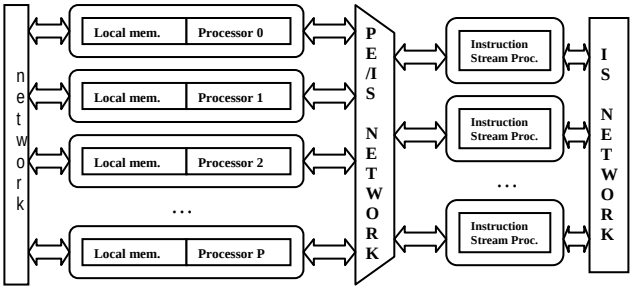

Figure 3. The MASC model

Cells may be active, inactive, or idle. Active cells execute the program which is broadcast from the IS that it is currently listening to. An inactive cell is considered in a group of IS cells, but does not execute instructions until the IS instructs inactive cells to become active again. Idle cells are currently inactive, not listening to any IS, and contain no essential program data but may be re-assigned as an active cell later [26].

ISs can be active or idle. An active IS issues instructions to a group of cells. An idle IS is not assigned to any PEs and is waiting until another IS forks, partitioning its PEs between itself and a new previously inactive IS. All PEs may be assigned to one of the ISs using local data and comparisons. If an IS broadcasts some value to a set of PEs, the PEs could set this value to their active IS in the next instruction cycle, or choose not to switch. That is, a PE can change the IS to which it listens dynamically [29].

MASC supports data parallel reduction operations: AND, OR, MIN and MAX; one or more instruction streams (ISs), each of which is sent to a distinct set in a dynamic partition of the processors; broadcasting from the ISs; and task assignment to ISs using control parallelism or data locality which allows PEs to switch ISs based on local data [26]. There are three networks, real or virtual, shown in Figure 3: the PE interconnection network, the IS interconnection network, and the network between the PEs and ISs.

To ensure running time predictability of the data parallel model on actual machines, parameters describe the running time for the basic operations of the MASC are as follows [24]:

I. T_{net} is the time to perform a routing of a word between all PEs, that is sending a word from each PE to any other PE location.
II. T_{comp} is the time to perform a sequential PE local operation indexed.
III. T_{comm} is the worst-case time for an IS to write or read data to or from one PE.
IV. T_{sync} is the maximum time for an IS communication or to synchronize ISs.
V. T_{broad} is the time to broadcast, and r is the time to do a data reductions and, or, min, max involving active PEs.
VI. T_{red} is the time to perform a reduction operation with data on a set of PEs.
VII. $T_{I/O}$ is the time to input or output data from or to an external source to PEs.
VIII. The variable *p* is the number of PEs.
IX. The variable *j* is the number of ISs.

The MASC model simulates PRAM with constant time overhead when the PRAM algorithm requires the same order of shared memories as MASC ISs [26]. *Table 1* shows relationships between MASC and PRAM. In short the ISs of MASC can act like shared memories where a MASC 'IS broadcast' operation simulates the concurrent read of a PRAM [26][29]. Also, a MASC 'IS-reduce' operation simulates the PRAM concurrent write [26][29]. Because MASC completes most of its work in the local memories of the PEs, the MASC machine more closely resembles PPRAM rather than PRAM. Specifically, MASC is almost equivalent to the priority-CRCW PPRAM with the exception that the PPRAM does not include multiple instruction streams. As is seen in *Table 1* MASC(*p,m*) simulates the priority-CRCW(*p,m)* PRAM with *O(1)* overhead. The next section show the results of merging PPRAM with the multiple instruction streams of MASC.

Table 1. MASC times to simulate PRAM and also for PRAM to simulate MASC [26]

Machine:	**Is Simulated By:**
Priority-CRCW(*p,m*)	MASC(*p,m*)
In Time: *O(1)*	
Priority-CRCW(*p,m*)	MASC(*p,j*), $j \le m$
In Time: Probabilistic *O(min(p/j, m/j)),* Deterministic *O(m/j)*	
Priority-CRCW(*p,m*)	MASC(*p,j*), $j \le m$, with a PE network
In Time: Probabilistic *O(min(p/j, m/j, net_route_(p, m)),* Deterministic *O(min(p/j, net_route(p, m))* (net_route(p,m)) is time to simulate PRAM with known network methods for a given network.	
MASC(*p, 1*)	Priority-CRCW(*p,1*)
In Time: *O(1)*	
MASC(*p, 1*)	CREW(*p,1*) or EREW(*p,1*)
In Time: *O(p),* due to reductions/broadcasts	
MASC(*p,m*)	Priority CRCW (*p, m*)
In Time: *O(m),* due to instruction streams, or *O(1),* if instruction set is finite, (no user defined)	

3. The stream PRAM model

The stream PRAM model adds multiple instruction streams roughly as defined by MASC to the PPRAM

model. The ISs are appended onto the existing *m* shared memories of the PPRAM. By only allowing one IS, the stream PRAM can simulate any PPRAM algorithm in constant time if the stream PRAM model has concurrent read capability.

The model is envisioned as executing in the following three cycles:

1. *Instruction Send Phase:* Each IS sends the next instruction to the set of PEs that are part of that stream's 'working group.' For stream PRAM this is accomplished by each PE concurrently reading from a particular memory/IS. Thus for the CREW and CRCW variants, this can be accomplished in *O(1)*. The EREW and ERCW variants must simulate the concurrent read, which requires $O(\log p)$ time with $O(m+p)$ extra memory. Since an algorithm that needs *O(t)* time with *m* shared memories can easily be simulated with $m' \le m$ shared memories in $O(\frac{m}{m'}t)$ time [3], then we need $O(\frac{m+p}{m}\log p)$ steps to simulate the extra memory, equating to the overhead of $O(\frac{p}{m}\log p)$ in the worst case (since $p>m$). An alternative method takes *m* steps, that is for *i=0 to m-1* each processor $0 \le j \le p-1$ reads from shared memory (*(j+i) mod m)*. Each processor now has every possible instruction, and each read is exclusive, so this second method completes in *O(m).* For a small number of shared-memories/instruction streams this is fairly practical [24]. Thus a hybrid algorithm for EREW or ERCW can complete *phase 1* in $O(\min(m, \frac{p}{m}\log p))$. One may be able to do better, since the PPRAM can sum up *p* memories in only $O(\frac{p}{m}+\log m)$ [2], but this question is left open for future efforts. Still, any stream PRAM that has concurrent read can complete this phase in *O(1)*. Interestingly, the architectural work done for MASC and PRAM (notably for PRAM in Akl's book[3]) has shown there is much evidence that concurrent reading is quite possible in hardware [3][16][28][31][32][33]. The cost for *phase 1* is the cost for a concurrent read, or T_{CR}.

2. *Synchronous Execution of Operations:* The PEs of any variant of stream PRAM now perform the operation loaded in *phase 1*. Each PE may execute one of the *m* potentially different instructions. It is assumed that these instructions will complete in *O(1)* time for simple operations or *O(f(n/p))* time for an operation, *f(),* which is a sequential function. *Phase 2* finishes when the PE with the slowest operation completes. Thus, there needs to be an implicit barrier synchronization at the end of this phase. The time needed is the average cost for each CPU instruction times the number of instructions plus the synchronization cost, or $T_{CPU} \times T_{SEQUENTIAL}(f(\frac{n}{p})) + T_{SYNC}$, where $T_{SEQUENTIAL}(f(\frac{n}{p}))$ may often be *O(1)* for simple operations.

3. *Next Instruction Determination Phase:* Each PE now decides which instruction stream it will execute next. Based on data and conditions local to each PE, a PE selects from which stream it will read in *Phase 1*. Note that the ISs do not directly dictate this. *Phase 3* has a cost of $T_{CPU} \times const$.

4. Goto *Phase1.*

One stream PRAM execution cycle is $T_{CR} + T_{CPU} \times T_{SEQUENTIAL}(f(\frac{n}{p})) + T_{SYNC} + T_{CPU} \times const$. For large *n* the sequential operations dominate. The special cases of EREW and ERCW stream PRAM(*n,p,m*) that always use only one (or constant) IS, but have *m* shared memories, can actually complete *Phase 1* in $O(\frac{p}{m}+\log m)$ with a modification of the summation algorithm from [2]. Instead of adding several values, one value is replicated with a 1-to-*p* broadcast. A concern of the reader at this point may be that the seemingly more powerful stream PRAM is only more powerful if concurrent read is allowed. This is actually true. The definition of the stream PRAM model states that each IS is grouped with a shared memory. The benefit is that the model still consists of only two layers of resources, the processors and the shared-memories/instruction-streams. Because exclusive read model stream PRAM PEs cannot concurrently read data from the shared memories, the PEs can also not concurrently read the next instruction. The authors and much previous literature believe that the exclusive read model is excessively restrictive [3][5][16][28][32][33]. If an architecture can afford multiple instruction streams, it probably ought to also have concurrent read. *Table 2* lists some of the nicely behaving simulations.

Table 2. Simulations of stream PRAM and MASC

Simulation	**In Extra Time**
Priority-CRCW PPRAM(*n,m*) **Simulated by:** MASC(*n,m*)	***O(1)***
MASC(*n,m*) **Simulated by:** Priority CRCW PPRAM(*n,m*)	***O(m), to simulate ISs Or O(1) for constant number of ops.***
PPRAM(*n,m*) **Simulated by:** Priority CRCW stream PRAM(*n,m*)	***O(1)***
MASC(*n,m*) **Simulated by:** Priority-CRCW stream PRAM(*n,m*)	***O(1)***
Priority-CRCW stream PRAM(*n,m*) **Simulated by:** MASC(*n,m*)	***O(1)***

From *Table 2* and what has been shown concerning stream PRAM properties, it is apparent that there is a strong relationship between MASC and stream PRAM. Specifically, priority-CRCW stream PRAM(*n,p,m*) is for all intensive purposes equivalent to MASC(*n,p,m*). Therefore any MASC algorithms already discovered will function in priority-CRCW stream PRAM in *O(1)* steps when the number of stream PRAM shared memories is on the same order as the number of instruction streams in MASC. Also, MASC can execute any existing stream PRAM or PPRAM algorithms with constant overhead [4][6][8][10][13][15][20][27][29].

4. Predictability, simulation on real systems, and future directions

Recent work to define the circuitry for the MASC model has shown that it is well within the realm of implementation [1][24][31][32][33]. By implication, the implementation of stream PRAM and PPRAM is also quite possible. The authors believe that clusters of workstations can support the stream PRAM paradigm with average time predictability [2][5][17][18][21][22]. Previous work in limited-resource PRAM show good results with PPRAM, BSP, and *qsm(m)* [2][30][12]. The extra overhead to run the *m* ISs ought to require only a slight amount of overhead when considering parallel slackness and many large problems where $n >> p$ [9]. *Figure 4* shows one naive method to execute the ISs.

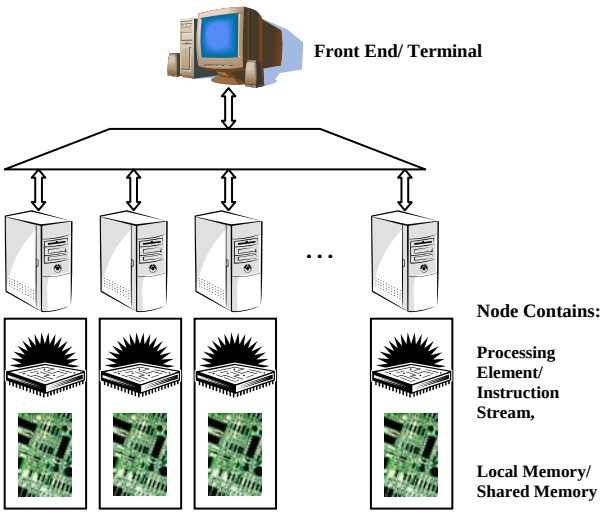

Figure 4: Simulation of stream PRAM on a cluster.

If we wish to use only a constant or log number of ISs, then each processor, PE that is, could keep track of all ISs. Given that local sequential operations that will operate on *n/p* sized data chunks, even for $O(\log \frac{n}{p})$ algorithms, $O(\log p)$ or so ISs would be trivial for each PE to run. Many local sequential operations, such as all-sums, have linear speedup, $O(n/p)$ [25].

In future work, we will experimentally measure methods to execute the instruction streams. Also, more work needs to be done concerning how the *m* instruction streams can solve some problems faster in stream PRAM. Maher Atwah's algorithm for $MASC(n,n,\log n)$ Convex Hull completes in $O(\log\log n)$ steps for randomized data with a high probability [4]. This algorithm can now function in priority CRCW stream PRAM*(n,n,log n)* in the same time. We need to look into more cases such as these, and we note that for the simple data parallel 'if-else' or 'case' statements, we can now use the ISs to compute all cases concurrently with the stream PRAM.

Lastly, we would like to determine relationships between stream PRAM, *qsm(m),* PPRAM, PRAM(*m*), and BSP. Obviously there exists overlap, and more effort is needed to determine time complexities for problems solved on stream PRAM. We desire to eventually determine simulations and costs for stream PRAM to these similar models. Even more may be gained by the use of the instruction streams beyond the regular non-IS algorithms.

5. References

[1] Nael B. Abu-Ghazaleh, Philip A. Wilsey, Jerry Potter, Robert Walker, and Johnnie Baker, "Flexible parallel processing in memory: architecture + programming Model," In *Proc. Third Petaflop Workshop*, February 1999.

[2] A. Agbaria, Y. Ben-Asher, and I. Newman, "Communication-processor tradeoffs in limited resources PRAM," In *Proc. 11th ACM Symposium on Parallel Algorithms and Architectures*, France, May 1999, pages 74-82.

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

[4] Maher M. Atwah and Johnnie W. Baker, "An associative dynamic convex hull algorithm," In *Proc. Tenth IASTED International Conference on Parallel and Distributed Computing and Systems,* October 1998, pages 250-254.

[5] Y. Aumann and A. Schuster, "Deterministic PRAM simulation with constant memory blowup and no timestamps," In *Proc. Third Symposium on the Frontiers of Massively Parallel Computation*, College Park, MD, IEEE Computer Society Press, October 8-10 1990, pages 22-29.

[6] Johnnie W. Baker and Mingxian Jin, "Simulation of enhanced meshes with MASC, a MSIMD model", In *Proc. 11th International Conference on Parallel and Distributed Computing Systems*, November 1999, pages 511-516.

[7] T. Blank and J. R. Nickolls, "A grimm collection of MIMD fairy tales," In *Proc. Fourth Symposium on the Frontiers of Massively Parallel Computation*, McLean,

VA, October 19-21, 1992, IEEE Computer Society Press, pages 448-457.

[8] G. E. Blelloch. “Programming Parallel Algorithms.” *Communications of the ACM,* 39(3), March 1996.

[9] David Culler, R. Karp, D. Patterson, A. Sahay, K.E. Schauser, E. Santos , R. Sub-ramonian and T. von Eicken. “LogP: towards a realistic model of parallel computation,” In *Proc. Symposium on Principles and Practice of Parallel Programming*, San Diego, CA., May 1993, pages 1-12.

[10] Mary Esenwein and Johnnie Baker, "VLCD string matching for associative computing and multiple broadcast mesh", In *Proc. IASTED International Conference on Parallel and Distributed Computing and Systems*, 1997, pages 69-74.

[11] A. Falkoff, “Algorithms for parallel search memories,” *Journal of Associative Computing*, March 1962, pages 488-511.

[12] P. B. Gibbons, Y. Mattias, and V. Ramachandran, "Can a shared-memory model serve as a bridging model for parallel computation?," In *9th Annual ACM Symposium on Parallel Algorithms and Architectures,* (Newport, Rhode Island), June 1997, pp. 72-83.

[13] S. Hillis, “Data parallel algorithms,” *Communications of the ACM*, 29(12), December 1986, 1170-1183.

[14] K. F. Hioe, “Asprol (Associative Programming Language),” Master's project, Kent State University, Math and Computer Science (MSB), August 1986.

[15] Mingxian Jin, “Evaluating the power of the parallel MASC model using simulations and real-time applications", Ph.D. Dissertation, Department of Computer Science, Kent State University, August, 2004.

[16] Mingxian Jin, Johnnie Baker and Kenneth Batcher, "Timings for associative operations on the MASC model", In *Proc. 15th International Parallel and Distributed Processing Symposium (Workshop in Massively Parallel Processing)*, April 2001, abstract on page 193, full text on CDROM.

[17] Anna R. Karlin and Eli Upfal. “Parallel hashing: An efficient implementation of shared memory,” *Journal of the ACM*, October 1988, 35(4):876-892.

[18] K. Li, Y. Pan, and S.-Q. Zheng. “Efficient Deterministic and Probabilistic Simulations of PRAMs on Linear Arrays with Reconfigurable Pipelined Bus Systems,” *The Journal of Supercomputing,* vol. 15, no. 2, February 2000, pp. 163-181.

[19] J. Potter, J. Baker, S. Scott, A. Bansal, C. Leangsuksun, C. Asthagiri, “MASC: An Associative Computing Paradigm,” *IEEE Computer*, November 1994, pages 19-25.

[20] J.L. Potter, Associative Computing - A Programming Paradigm for Massively Parallel Computers, Plenum Publishing, N.Y., 1992.

[21] Michael J. Quinn and Philip J. Hatcher, “Data-Parallel Programming on Multicomputers,” IEEE Software, J. Hatcher,” *IEEE Software,* 7(5), September 1990, 69-76.

[22] S.H. Noh, K. Dussa-Zieger, “Improving massively data parallel system performance with heterogeneity,” *Frontiers of Massively Parallel Computation,* 1992, Oct 19-21, 1992, McLean, VA, USA, pages 93-99.

[23] Michael Scherger, Johnnie Baker, and Jerry Potter, "An object oriented framework for and associative model of parallel computation", *In Proc. 16th International Parallel and Distributed Processing Symposium (Workshop in Advances in Parallel and Distributed Computational Models)*, April 2003.

[24] Michael Scherger, Johnnie Baker, and Jerry Potter, "Multiple instruction stream control for an associative model of parallel computation", *In Proc. 16th International Parallel and Distributed Processing Symposium (Workshop in Massively Parallel Processing),* April 2003.

[25] Michael Scherger, Jerry Potter, and Johnnie Baker, "On using UML to describe the MASC model of parallel computation", In *Proc. 2000 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2000)*, volume V, June 2000, pages 2639-2645.

[26] Darrell Ulm and Johnnie Baker, "Simulating PRAM with a MSIMD model (ASC)", In *Proc. International Conference on Parallel Processing*, August 1998, pages 3-10.

[27] Darrell Ulm and Johnnie Baker, "Solving a 2D knapsack problem on an associative computer augmented with a linear network", In *Proc. International Conference on Parallel and Distributed Processing Techniques and Applications*, 1996, pages 29-32.

[28] Mingxian Jin, Johnnie Baker, and Kenneth Batcher, "Timings for Associative Operations on the MASC Model", In *Proc. 18th Intertational Parallel and Distributed Processing Symposium (Workshop in Massively Parallel Processing)*, April 2004.

[29] D. Ulm and J. Baker, “Virtual parallelism by self-simulation of the multiple instruction stream associative model,” In *Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications*, III, Sunnyvale CA, August 1996, pages 1421-1430.

[30] L.G. Valiant. “A bridging model for parallel computation," *Comm. ACM,* 33:103-111, Aug. 1990.

[31] Robert A. Walker, Jerry Potter, Yanping Wang, and Meiduo Wu, "Implementing associative processing: rethinking earlier architectural decisions", In *Proc. of the 15th International Parallel and Distributed Processing Symposium (Workshop in Massively Parallel Processing),* April 2001, abstract on page 195, full text on CDROM.

[32] Hong Wang, and Robert A. Walker, "Implementing a scalable ASC processor", In *Proc. of the 17th International Parallel and Distributed Processing Symposium (Workshop in Massively Parallel Processing),* April 2003.

[33] Meiduo Wu, Robert A. Walker, and Jerry Potter, "Implementing associative search and responder resolution", In *Proc. of the 16th International Parallel and Distributed Processing Symposium (Workshop in Massively Parallel Processing),* April 2002, abstract on page 246, full text on CDROM.