Multiple Instruction Stream Control for an Associative Model of Parallel Computation

M. Scherger, J. Baker, and J. Potter
Department of Computer Science
Kent State University
Kent, Ohio 44242
{mscherge, jbaker, potter}@cs.kent.edu

Abstract

This paper describes a system software design for multiple instruction stream control in a massively parallel associative computing environment. The purpose of providing multiple instruction stream control is to increase throughput and reduce the amount of parallel slackness inherent in single instruction stream parallel programming constructs. The Multiple ASsociative Computing (MASC) model will be used to describe this technique and a brief introduction to the MASC model of parallel computation is presented. A simple parallel computing example is used to illustrate the techniques for multiple instruction stream control in a massively parallel runtime environment.

Keywords: Multiple instruction streams, associative computing, system software, parallel processing.

1 Introduction

Traditional massively parallel processing computers such as single instruction stream, multiple data stream (SIMD) computers rely on a host or instruction stream to control the array of parallel processors or parallel array unit. By definition of Flynn's taxonomy of parallel computer architectures, the SIMD model has only one control unit.

But what if more than one control unit is introduced to a massive array of processors? Can multiple control units or instruction streams work seamlessly to improve application performance and throughput? Would multiple instruction streams reduce parallel slackness inherent in parallel programming constructs such as the parallel if-then-else and if so, how much overhead would a parallel runtime environment incur? How should these multiple instruction streams be controlled from a programming and system software perspective?

To answer some of these questions and examine some of these research topics, the Parallel and Associative Computing Research Group at Kent State University is conducting research on just such a model. The Multiple ASsociative Computing (MASC) model of parallel computation uses an M-SIMD (multiple SIMD) architecture that allows for multiple instruction streams to control a unique partition of a global set or array of processing elements (cells).

This research explores a method of controlling multiple instruction streams in a massively parallel processing environment such as defined by the MASC model of parallel computing. Furthermore, this research explores this control from a system software perspective (compilers and parallel runtime environments). This was achieved by examining the current system software requirements for single instruction stream computing and designing the compiler and system software modifications for a multiple instruction stream associative computing runtime environment.

Section 2 will present an overview of the MASC model of parallel computation. This model supports a generalized data and associative parallel programming paradigm with restricted support for control parallelism. Section 3 will discuss the system software support for the MASC Model. This includes the current MASC compiler and emulator as well as a runtime environment of the MASC model using clusters of workstations. Section 4 will present the theory and design for control of multiple instruction streams. New parallel byte code instructions for this control are introduced and a discussion of the overhead required for their implementation is presented. Section 5 will present a parallel associative computing example demonstrating multiple instruction stream control.

2 The MASC Model of Parallel Computation

The following is a conceptual description of the Multiple Associative Computing (MASC) model of parallel computation. As shown in figure 1, the MASC model consists of an array of processor-memory pairs called *cells* and an array of instruction streams.

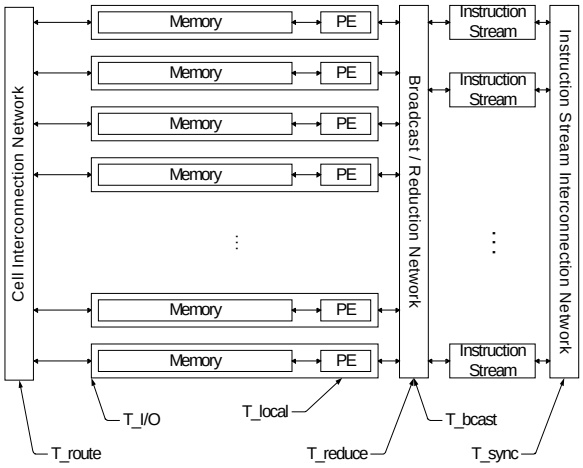

Figure 1: Conceptual view of MASC.

A MASC machine with *n* cells and *j* instruction streams is denoted as *MASC(n, j)*. It is expected that the number of instruction stream processors be much less than the number of cells. The model also includes three virtual networks:

1. A cell network used for cell-to-cell communication. This network is used for the parallel movement of data between cells. This network could be a linear array, mesh, hypercube, or a dynamic interconnection network.

2. A broadcast/reduction network used for communication between an instruction stream and a set of cells. This network is also capable of performing common reduction operations.

3. An instruction stream network used for inter-instruction stream communication.

Cells can receive their next set of instructions to execute from the instruction stream broadcast network. Cells can be instructed from their current instruction stream to send and receive messages to other cells in the same partition using some communication pattern via the cell network. Each instruction stream processor is also connected to two interconnection networks. An instruction stream processor broadcasts instructions to the cells using the instruction stream broadcast network. The instruction streams also may need to communicate and may do so using the instruction stream network. Any of these networks may be virtual and be simulated by whatever network is present.

MASC provides one or more instruction streams. Each active instruction stream is assigned to a unique dynamic partition of cells. This allows a task that is being executed in a data parallel fashion to be partitioned into two or more data parallel tasks using control parallelism. The multiple IS's supported by the MASC model allows for greater efficiency, flexibility, and reconfigurability than is possible with only one instruction stream. While SIMD architectures can execute data parallel programs very efficiently and normally can obtain near linear speedup, data parallel programs in many applications are not completely data parallel and contain several non-trivial regions where significant branching occurs [1]. In these parallel programming regions, only a subset of traditional SIMD processors can be active at the same time. With MASC, control parallelism can be used to execute these different branches simultaneously. Other MASC properties include:

- The cells of the MASC model consist of a processing element (PE) and local memory. The accumulated memory of the MASC model consists of an array of cells. There is no shared memory between cells.

- Each instruction stream is a processor with a bus or broadcast/reduction network to all cells. Each cell listens to only one instruction stream and initially, all cells listen to the same instruction stream. The cells can switch to another instruction stream in response to commands from the current instruction stream.

- An active cell executes the commands it receives from its instruction stream, while an inactive cell listens to but does not execute the command from its instruction stream. Each instruction stream has the ability to unconditionally activate all cells listening to it.

- Cells without further work are called idle cells and are assigned to a specified instruction stream, which among other tasks manages the idle cells.

- The average time for a cell to send a message through the cell network to another cell is characterized by

the parameter t_{route}. Each cell also can read or write a word to an I/O channel. The maximum time for a cell to execute a command is given by the parameter t_{local}. The time to perform a broadcast of either data or instructions is given by the predictability parameter t_{bcast}. The time to perform a reduction operation is given by the predictability parameter t_{reduce}. The time for a cell to perform this I/O transfer is characterized by the parameter $t_{i/o}$. The time to perform instruction stream synchronization is characterized by the parameter t_{synch}.

- An instruction stream can instruct its active cells to perform an associative search in time $t_{bcast} + t_{local} + t_{reduce}$. Successful cells are called *responders*, while unsuccessful cells are called *non-responders*.

- The instruction stream can activate either the set of responders or the set of non-responders. It can also restore the previous set of active cells in $t_{bcast} + t_{local}$ time.

- Each instruction stream has the ability to select an arbitrary responder from the set of active cells in $t_{bcast} + t_{local}$ time.

- An active instruction stream can compute the *OR*, *AND*, *Greatest Lower Bound*, or *Least Upper Bound* of a set of values in all active cells in t_{reduce} time.

- An idle cell can be dynamically allocated to an instruction stream in $t_{synch} + t_{bcast}$ time.

These predictability parameters were identified using an object oriented description of the MASC model in [7]. They were developed to identify the performance costs using different architecture classes of parallel computing equipment. When the MASC model is implemented using a traditional SIMD computer such as the STARAN or Wavetracer DTC or Zephyr, the MASC model is highly deterministic and the predictability costs can often be calculated and are often "best possible" [3]. Many of the predictability parameters for MASC operations become fixed or operate in one step [3].

3 MASC System Software

The current system software support for the MASC model consists of a compiler, an emulator for Windows or Linux based machines, and a parallel runtime environment using a cluster of Linux machines. The ASC programming language is a data parallel associative programming language. This language is further defined in [5][6]. The ASC compiler translates the ASC syntax into MASC byte code which can be executed, by a number of parallel runtime environments. Presently, single instruction stream runtime environments for the ASC compiler and emulator are available for Windows and Linux machines, the Connection Machine, the WaveTracer DTC and Zephyr, and the ASPRO.

A multiple instruction stream parallel runtime environment is presently in development using a cluster of workstations as a feasibility prototype. This runtime environment interprets and executes the MASC byte code generated by the current single instruction stream ASC compiler. Since the cluster implementation of MASC support multiple instruction streams and since the ASC compiler only generates single instruction stream byte code, a compiler option or optimization phase is being developed. This optimization procedure is called the ISGEN (Instruction Stream GENerate) and would be performed after linking. The phases of compilation for a MASC program are depicted in the following figure.

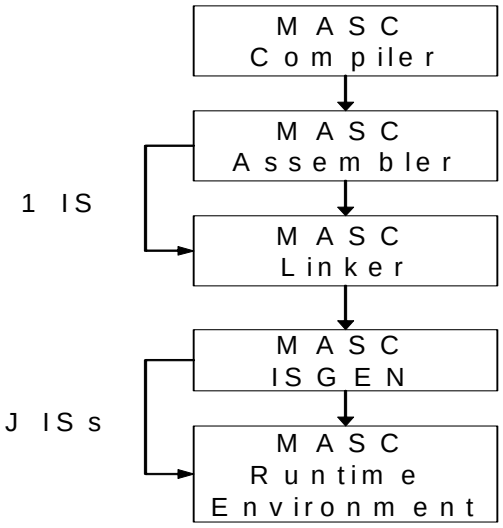

Figure 2: MASC phases of compilation including the ISGEN phase.

The ISGEN procedure is used to identify candidate regions that can be assigned a separate instruction streams. Within each multiple instruction stream region, several code regions or basic blocks are be identified and assigned to a virtual instruction stream. Each virtual instruction stream region is then assigned or mapped to a

physical instruction stream within the MASC parallel runtime environment during program execution.

4 Multiple Instruction Stream Control Design

This section will present the parallel programming constructs supporting multiple instruction streams and discuss the MASC byte code operations to support multiple instruction streams in the MASC parallel runtime environment.
Consider the parallel associative if-then-else programming construct.

```
if (parallel condition) then
      <body_1>
else
      <body_2>
endif;
```

In the ASC programming language the result of a parallel conditional statement is stored in a parallel bit vector. For a single instruction stream implementation, the instruction stream will instruct all cells to evaluate the parallel condition and then set their parallel responder bit accordingly. For those cell responders set to TRUE, programming region body_1 is executed. At the completion of body_1, the non-responders become active and programming region body_2 is executed. Thus, for a single instruction stream, body_1 and body_2 are both executed in sequence. The non-responders are inactive during the time body_1 is executing and vice versa.

The ISGEN phase during compilation would identify begin and end of an ASC parallel if-then-else construct and insert four new byte code instructions. It can transform a single instruction stream ASC program into a multiple instruction stream MASC program.

- MI_REGION_BEGIN – This command will be used to identify a candidate region for multiple instruction streams. This command will use the current assigned instruction stream to perform the evaluation of the parallel conditional expression.

- MI_REGION_END – This command will be used to identify the end of a candidate region of multiple instruction streams. This instruction will be used to collapse and combine multiple instruction streams into a single instruction (thread of execution). This can be achieved by performing a synchronization to wait for all instruction streams to complete.
- MI_BEGIN – This instruction will be used to mark the beginning of a basic block identified for a multiple instruction stream.

- MI_END – This instruction will be used to mark the end of a basic block identified for a multiple instruction stream.

In theory, several MI_BEGIN and MI_END statements may exist within a single MI_REGION_BEGIN and MI_REGION_END construct. However for a simple parallel if-then-else statement, the MASC byte code would have the following format:

```
MI_REGION_BEGIN A
(parallel conditional expression)
MI_BEGIN A0
<body_1>
MI_END A0
MI_BEGIN A1
<body_2>
MI_END A1
MI_REGION_END A
```

The labels used after each command are *structure codes* [5] used to keep track of which child instruction stream belongs to a parent region.

As the MASC runtime environment interprets this byte code, the runtime environment must now broadcast instructions from each of the MI_BEGIN and MI_END regions simultaneously. This can be done by runtime environment by reordering the byte code instructions from each of the instruction streams into a VLIW (very long instruction word). The VLIW instruction word is an array of MASC instructions which is indexed by an instruction stream ID number. The following illustration (figure 3) further illustrates the VLIW array or vector of MASC instructions.

There are sources of overhead identified and associated with implementing multiple instruction streams in the parallel runtime environment for MASC.

1. The MASC runtime environment must now simultaneously broadcast instructions on behalf of the multiple instruction streams simultaneously. This requires the runtime environment to buffer the instructions and then reorder them into the VLIW instruction word individually.

2. A synchronization is required to collect and collapse instruction streams. While the synchronization cost in a traditional SIMD is fixed at one step, the cost in a cluster implementation defined by the cost of such synchronization in the cluster communication library (for example MPI or PVM).

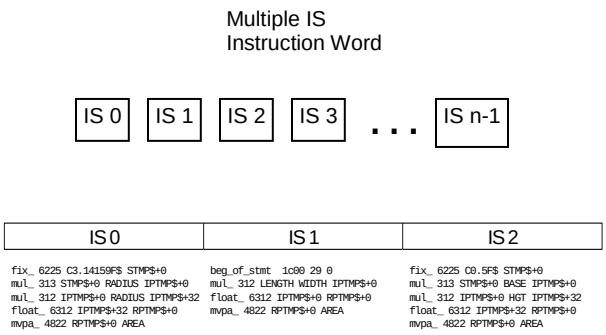

Figure 3: Example Multiple IS instruction words implemented as a VLIW array or vector of independent MASC instructions.

The overhead of the synchronization is limited by the efficiency of reaching a global consistent state among a set of processes using a parallel communication library. Using MPI, for example, using standard TCP/IP, the latency can be quite high (approximately 1-10 ms). This latency can be reduced using a different physical medium for data communication such as Myrinet. There are many such studies to demonstrate communication latency and process synchronization; however they are out of the scope of this research.

5 An Example of Multiple Instruction Streams

To illustrate the use of the four new MASC intermediate instruction codes, this section will present a simple parallel associative programming example. Consider the data parallel application of finding the area of various shapes: circles, rectangles, triangles, etc. A MASC program would first read into the array memory a set of shape data, one shape per cell in the associative memory. Next, the program would compute the area of the various shapes. Since the area of many shapes has to be computed, the program will test a "shape type" parallel variable to determine which area computation to perform for a given partition of cells. The following is a partial code listing for this example application (figure 4).

Note that parallel variables use a special programming syntax of having "[$]" as an identifier suffix to denote a parallel variable.

5.1 Single Instruction Stream Execution

For the single instruction stream case, as in the existing MASC emulator, the single instruction stream executes instructions for both branches of the parallel if-then-else constructs. The interpreter processes all the responder cells, or those cells that masked the parallel condition as TRUE, and then process all the non-responder cells; those cells that masked the parallel condition as FALSE.

While the cells that have responded as "circles" are computing their respective areas, all cells that have identified themselves as "not circles" are inactive and waiting to perform their area computations. Likewise, while "rectangles" are computing their respective areas, "triangles" are inactive and waiting to perform there area computations. Since a single instruction stream is used to execute both branches of a parallel if-then-else construct, parallel slackness is being introduced causing a reduction in throughput.

5.2 Multiple Instruction Stream Execution

To reduce the amount of parallel slackness and increase throughput, the use of multiple instruction streams is introduced at the MASC byte code level. Consider the following modification to the MASC byte code as shown in figure 5.

Since in this example the area of three different types of shapes is computed, three separate multiple instruction stream regions have been identified. The start of a multiple instruction stream region begins as the MASC byte code for a parallel comparison is identified. The region is initialized and assigned a label used by the parallel runtime environment for instruction stream synchronization. The current instruction stream is used to make the parallel comparison and set the responder vector for those cells that have identified themselves as "circles". The first MI_BEGIN-MI_END block (i.e. A0 in Figure 5) is next and identifies the section of code for computing the area of a circle. The second MI_BEGIN-MI_END block is used processing the outermost parallel else

```
if shapetype[$] .eq. CIRCLE then
  area[$]=3.14159*radius[$]*radius[$];
else
  if shapetype[$] .eq. RECTANGLE then
    area[$] = length[$] * width[$];
  else
    if shapetype[$] .eq. TRIANGLE then
      area[$] = 0.5*base[$]*height[$];
    endif;
  endif;
endif;
```

```
beg_of_stmt  1c00 29 0
begif 2000 IF$S1
eq_ 714 SHAPETYPE CIRCLE LPTMP$+0
mvpa_ 4832 LPTMP$+0 THEM
if_ 3002 LPTMP$+0 ELS$1
beg_of_stmt  1c00 30 0
decl_ 6125 C3.14159F$ 1 3.14159 0 32
entry_ 6f00 C3.14159F$
fix_ 6225 C3.14159F$ STMP$+0
mul_ 313 STMP$+0 RADIUS IPTMP$+0
mul_ 312 IPTMP$+0 RADIUS IPTMP$+32
float_ 6312 IPTMP$+32 RPTMP$+0
mvpa_ 4822 RPTMP$+0 AREA
beg_of_stmt  1c00 31 0
label 6800 ELS$1
else_ 2c02 IF$E1
beg_of_stmt  1c00 32 0
begif 2000 IF$S2
eq_ 714 SHAPETYPE RECTANGLE LPTMP$+0
mvpa_ 4832 LPTMP$+0 THEM
if_ 3002 LPTMP$+0 ELS$2
beg_of_stmt  1c00 33 0
mul_ 312 LENGTH WIDTH IPTMP$+0
float_ 6312 IPTMP$+0 RPTMP$+0
mvpa_ 4822 RPTMP$+0 AREA
beg_of_stmt  1c00 34 0
label 6800 ELS$2
else_ 2c02 IF$E2
beg_of_stmt  1c00 35 0
begif 2000 IF$S3
eq_ 714 SHAPETYPE TRIANGLE LPTMP$+0
mvpa_ 4832 LPTMP$+0 THEM
if_ 3002 LPTMP$+0 ELS$3
beg_of_stmt  1c00 36 0
decl_ 6125 C0.5F$ 1 0.5 0 32
entry_ 6f00 C0.5F$
fix_ 6225 C0.5F$ STMP$+0
mul_ 313 STMP$+0 BASE IPTMP$+0
mul_ 312 IPTMP$+0 HEIGHT IPTMP$+32
float_ 6312 IPTMP$+32 RPTMP$+0
mvpa_ 4822 RPTMP$+0 AREA
beg_of_stmt  1c00 37 0
label 6800 ELS$3
endif_ 2702 IF$E3
beg_of_stmt  1c00 38 0
endif_ 2702 IF$E2
beg_of_stmt  1c00 39 0
endif  2702 IF$E1
```

Figure 4: Partial MASC program and resultant byte code for computing area of various shapes.

```
.MI_REGION_BEGIN A
beg_of_stmt  1c00 29 0
begif 2000 IF$S1
eq_ 714 SHAPETYPE CIRCLE LPTMP$+0
mvpa_ 4832 LPTMP$+0 THEM
if_ 3002 LPTMP$+0 ELS$1
.MI_BEGIN A0
beg_of_stmt  1c00 30 0
decl_ 6125 C3.14159F$ 1 3.14159 0 32
entry_ 6f00 C3.14159F$
fix_ 6225 C3.14159F$ STMP$+0
mul_ 313 STMP$+0 RADIUS IPTMP$+0
mul_ 312 IPTMP$+0 RADIUS IPTMP$+32
float_ 6312 IPTMP$+32 RPTMP$+0
mvpa_ 4822 RPTMP$+0 AREA
.MI_END A0
.MI_BEGIN A1
beg_of_stmt  1c00 31 0
label 6800 ELS$1
else_ 2c02 IF$E1
.MI_REGION_BEGIN B
beg_of_stmt  1c00 32 0
begif 2000 IF$S2
eq_ 714 SHAPETYPE RECTANGLE LPTMP$+0
mvpa_ 4832 LPTMP$+0 THEM
if_ 3002 LPTMP$+0 ELS$2
.MI_BEGIN A1-B0
beg_of_stmt  1c00 33 0
mul_ 312 LENGTH WIDTH IPTMP$+0
float_ 6312 IPTMP$+0 RPTMP$+0
mvpa_ 4822 RPTMP$+0 AREA
.MI_END A1-B0
.MI_BEGIN A1-B1
beg_of_stmt  1c00 34 0
label 6800 ELS$2
else_ 2c02 IF$E2
.MI_REGION_BEGIN C
beg_of_stmt  1c00 35 0
begif 2000 IF$S3
eq_ 714 SHAPETYPE TRIANGLE LPTMP$+0
mvpa_ 4832 LPTMP$+0 THEM
if_ 3002 LPTMP$+0 ELS$3
.MI_BEGIN A1-B1-C0
beg_of_stmt  1c00 36 0
decl_ 6125 C0.5F$ 1 0.5 0 32
entry_ 6f00 C0.5F$
fix_ 6225 C0.5F$ STMP$+0
mul_ 313 STMP$+0 BASE IPTMP$+0
mul_ 312 IPTMP$+0 HEIGHT IPTMP$+32
float_ 6312 IPTMP$+32 RPTMP$+0
mvpa_ 4822 RPTMP$+0 AREA
.MI_END A1-B1-C0
beg_of_stmt  1c00 37 0
label 6800 ELS$3
endif_ 2702 IF$E3
.MI_REGION_END C
.MI_END A1-B1
beg_of_stmt  1c00 38 0
endif_ 2702 IF$E2
.MI_REGION_END B
beg_of_stmt  1c00 39 0
endif_ 2702 IF$E1
.MI_END A1
.MI_REGION_END A
```

Figure 5: Partial MASC byte code illustrating multiple instruction streams begin and end regions.

statement. When the next byte code for a parallel if statement is encountered (see region B in Figure 5), a new multiple instruction stream region is established. The two instruction streams for the second region are used to compute the area of rectangle cells and triangle cells (which are in a third multiple instruction stream region. After all of the byte code statements for a multiple instruction stream block are complete, the cells change to an inactive state and wait for a synchronization to complete for the corresponding multiple instruction stream region. For this non-optimized example three synchronizations are required to collapse the three instruction streams to one.

One such improvement is the multiple instruction stream region optimizations. Since each multiple instruction stream region concludes with a synchronization step, reducing the number of regions are reduce the overhead incurred by instruction stream synchronizations. This is illustrated in the MASC byte code shown in figure 6.

```
.MI_REGION_BEGIN A
beg_of_stmt  1c00 29 0
XLAT  nnnn SHAPETYPE
.MI_BEGIN A0
beg_of_stmt  1c00 30 0
decl_ 6125 C3.14159F$ 1 3.14159 0 32
entry_ 6f00 C3.14159F$
fix_ 6225 C3.14159F$ STMP$+0
mul_ 313 STMP$+0 RADIUS IPTMP$+0
mul_ 312 IPTMP$+0 RADIUS IPTMP$+32
float_ 6312 IPTMP$+32 RPTMP$+0
mvpa_ 4822 RPTMP$+0 AREA
.MI_END A0
.MI_BEGIN A1
beg_of_stmt  1c00 33 0
mul_ 312 LENGTH WIDTH IPTMP$+0
float_ 6312 IPTMP$+0 RPTMP$+0
mvpa_ 4822 RPTMP$+0 AREA
.MI_END A1
.MI_BEGIN A2
beg_of_stmt  1c00 36 0
decl_ 6125 C0.5F$ 1 0.5 0 32
entry_ 6f00 C0.5F$
fix_ 6225 C0.5F$ STMP$+0
mul_ 313 STMP$+0 BASE IPTMP$+0
mul_ 312 IPTMP$+0 HEIGHT IPTMP$+32
float_ 6312 IPTMP$+32 RPTMP$+0
mvpa_ 4822 RPTMP$+0 AREA
.MI_END A2
.MI_REGION_END A
```

Figure 6: Partial example MASC byte code illustrating multiple instruction streams begin and end regions as created optimized to reduce the number of multiple instruction stream regions.

This optimized version of the shape example using multiple instruction streams requires only one region, and therefore, only one synchronization.

6 Conclusions

This paper has discussed a method of controlling multiple instruction streams in a massively parallel associative computing environment. The MASC model is a model of parallel computation that supports an associative style of parallel computation that allows memory to be addressed by content rather than by address. This model was used to demonstrate how multiple instruction stream control is feasible by introducing four new MASC byte code operations that are interpreted by the parallel runtime environment. These operations are used to identify program regions where multiple instruction streams exist and define their entry and exit points. An example was developed to demonstrate the new MASC operations. This example discussed the execution using single and multiple instruction streams.

7 References

[1] Fox, Geoffrey, “What Have We Learnt from Using Real Parallel Machines to Solve Real Problems”, Proceedings of the Third Conference on Hypercube Concurrent Computers and Applications, Vol. 2, ACM Press, pp. 897-955, 1988.

[2] Herbordt, Martin, and Charles Weems, “Associative, Multiassociative, and Hybrid Processing”, Associative Processing and Processors, Anargyros Krikelis and Charles Weems, eds., IEEE Comupter Society Press, pp. 26-49, 1997.

[3] Jin, Minxian, Johnnie Baker, Kenneth Batcher, “Timings for Associative Operations on the MASC Model”, Proceedings of the 15th International Parallel and Distributed Symposium (Workshop on Massively Parallel Processing), April 2001.

[4] Todd Heywood and Claudia Leopold, "Models of Parallelism", Abstract Machine Models for Highly Parallel Computers, John R. Davy and Peter M. Dew, Eds., Oxford Science Publications, Oxford, England, 1995, pp. 1-16.

[5] Potter, Jerry L., Associative Computing: A Programming Paradigm for Massively Parallel Computers, Plennum Press, New York, NY, 1992.

[6] Potter, Jerry, Johnnie Baker, Stephen Scott, Arvind Bansal, Chokchai Leangsuksun, and Chandra Asthagiri, “ASC: An Associative Computing Paradigm”, IEEE Computer, Nov., 1994, pp. 19-25.

[7] Scherger, Michael, “On Using the UML to Describe the MASC Model of Parallel Computation”, Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000), June 2000, pp. 2639-2645.

[8] Scherger, Michael, “Using the UML to Describe the BSP Model of Parallel Computation”, Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2002), June 2002, pp. 578-583.