An Extension of the ASC Language Compiler to Support Multiple Instruction Streams in the MASC Model using the Manager-Worker Paradigm

Wittaya Chantamas, Johnnie Baker, and Michael Scherger
Department of Computer Science
Kent State University, Kent, OHIO 44242 USA
Telephone: (330) 672-9055
Fax: (330) 672-7824
{wchantam, jbaker, mscherge}@cs.kent.edu

Abstract

In this paper, we describe and implement compiler extension for a parallel computer language called Associative Computing (ASC) language to support multiple instruction streams in a Multiple Associative Computing (MASC) model using manager-worker paradigm. A user directed MASC directive is used to enable concurrent executions of the THEN part and the ELSE part in a parallel IF-THEN-ELSE statement by using two different worker-instruction streams. For most applications, this technique should substantially improves the performance of the system over its performance using only one instruction stream; moreover it is more effective than using multiple instruction streams to execute every parallel IF-THEN-ELSE statements found in a program. When the overhead outweighs the benefit gained from using multiple instruction streams, a user can choose to use only one instruction stream to execute the IF-THEN-ELSE statement. While not explicitly covered here, parallel CASE statements can be handled similarly.

Keywords: parallel associative computing, multiple SIMD, multiple data parallel execution, compilation technique, runtime environment

1. Introduction

The associative computing model or the ASC model had been first introduced in the IEEE Computer in 1994. The ASC model is a SIMD model with an associative property and some additional hardware features. The associative property is the ability to locate items in the memory of cells by content rather than by the location. The additional hardware features enable the ASC model to broadcast data to cells, do a global reduction such as AND/OR of Boolean values or MAX/MIN of integer values in constant time, and do an associative search in constant time [5]. A language has also been designed for the ASC model and is called the ASC language [7].

The multiple associative computing model or MASC is an extension of the ASC model. That is, the MASC model is the ASC model with two or more instruction streams. While [6] identified the MASC computational model‘s properties, these were high-level and did not specify the interactions between the instruction streams (ISs) needed to support a MASC architecture. One possible approach to supporting multiple ISs in the MASC model is to use the manager and worker paradigm to support multiple executions of branches in a data parallel program. One instruction stream is designated to be the manager instruction stream and the rest of the instruction streams are worker instruction streams. Additional details of the model can be found in the earlier work presented in [2, 6]. One of the advantages of using worker and manager paradigm in the MASC model is the simplicity of the model. It is preferable to build a prototype or a run-time system using a paradigm such as the manager and worker paradigm that does not require the user to be aware of the use of this paradigm in order to run a program on it. Since we assumed there are only a small number of instruction streams in the model, we can avoid overworking the manager instruction stream and creating a bottleneck in the system.

In this paper, we discuss a design and implementation of the ASC language compiler extension for the MASC computational model with a MASC directive used. Although a previous work on compiler extension for the ASC language has been reported [8], this work is different from the previous work in the following aspects.

- *A user has a control on the use of multiple instruction streams to execute a branch.* The user has a tight control on which parallel IF-THEN-ELSE

statement the user wishes to use multiple instruction streams to execute the branch because a fork of a branch in a parallel IF-THEN-ELSE statement is only done when the user places a MASC directive before the statement. There is always an overhead when using a multiple instruction streams to execute a branch, i.e., an extra cost to fork and join tasks. In [8], all branches are forked automatically regardless of the number of instructions within those branches. If the overhead from using multiple instruction streams seems to outweigh the benefit gained from using only one instruction stream, in this work, a user has a choice of using only one instruction stream to execute the entire IF-THEN-ELSE statement. The analysis of the actual overhead of the fork and join operations is not included in this paper and will be discussed in future work.

- *The number of tasks generated is independent of the number of instruction streams.* The number of tasks generated using the technique proposed here depends only on the number of data parallel branches in the program and which ones of these are selected by the user to be executed in parallel. Since the task execution procedure in this work uses pool of tasks, the number of tasks can be more (or less) than the number of instruction streams available. Tasks in the work pool will be assigned to an available idle worker instruction stream one at a time based on a scheduling policy selected until no tasks are available in the work pool or no available worker instruction stream are left. In [8], the number of tasks is always equal or less than the number of instruction stream and a static scheduling procedure is used for the task assignment.
- *A worker instruction stream is free to be assigned a new task as soon as it finished the previous assigned task.* In [8], all instruction streams executing tasks generated from the same fork operation have to wait for the last instruction streams to finish its task before all of them can proceed to execute new tasks.
- *The testing of the conditional expression of a parallel IF-THEN-ELSE statement is done by a manager instruction stream.* This approach simplifies this technique and is easier to implement. In [8], the testing of a conditional expression is done by the instruction steam currently executing the task.

The rest of the paper is organized as the following. The basic components of the MASC computational model are described in section 2. The ASC language compiler extension for the MASC model is described in section 3. A description of a parallel IF-THEN-ELSE statement in the ASC language, its format, and how an instruction stream in the ASC model executes the statement are discussed in this section. What the MASC directive is and the execution of the IF-THEN-ELSE in the MASC model has been changed from [8] are also discussed in this section. Section 4 includes a summary and a discussion of expected future work.

2. MASC Computational Model

The basic component of the MASC model using the manager and worker paradigm are a set of instruction streams, an array of cells, an instruction stream network, a cell network, and broadcast and reduction networks (one for each instruction stream).

The instruction stream network is a network such as a simple bus or a broadcast and reduction network that instruction streams use to communicate to each other.

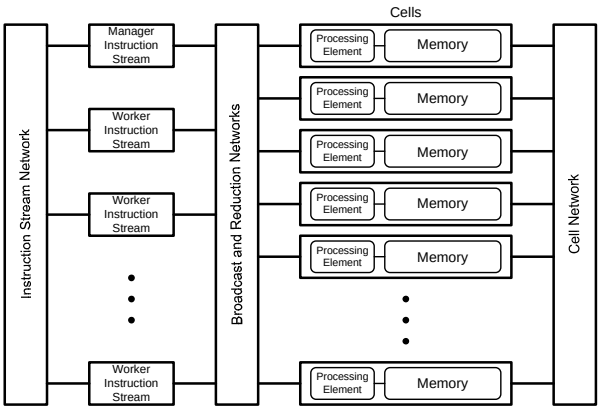

Figure 1: MASC computational model with manager and worker instruction streams.

A set of instruction streams (ISs), also known as control units, will consists of two types of instruction streams based on their functionalities. One of them is the manager instruction stream (manager-IS) and the rest of the ISs are worker instruction streams (worker-ISs). The functions of the manager-IS are managing a pool of tasks (work pool) and a pool of idle worker-ISs, coordinating and assigning tasks based on the policy set by a scheduler to idle worker-ISs, and combining finished tasks from worker-ISs. The only function of a worker-IS is to execute a task given to it by the manager-IS.

The next component of the MASC model is a set of broadcast and reduction networks. While it is assumed that, logically, each instruction stream has its own broadcast and reduction network, alternate methods might be used to support this capability. A broadcast and reduction network may be implemented as two separate networks (one for the broadcast and another for the reduction) and is used to connect a set of cells to an instruction stream. An instruction steam can broadcast instructions or data to

cells, perform a reduction operation of data across its cells, detect responder cells, and select an arbitrary responder cell in constant time using these networks.

A cell consists of a processing element (PE) and its local memory. The processing element in each cell is associated with a row of memory. A cell can listen to any instruction stream but it can only listen to one of the instruction stream at a time. When a cell satisfies a data test from an instruction stream it listens to, it becomes a responder. Otherwise, this cell is not a responder. Each cell had a mask register, a task history stack, and a IS selector register. The mask register indicates whether that cell is a responder or not. If the task history stack is not empty, the top of stack indicates the current task that the cell is executing and the rest of the stack indicate the tasks that this cell had participated, which have not yet been joined back to the parent tasks. The IS selector register holds the ID of the current instruction stream that this cell is listening to. Records for related data items are stored in tabular fashion in the parallel memory, with one item stored in the row corresponding to an individual cell. When the number of records is greater than the number of cells, a second table is used to store a second record in the cells, etc. However, using k tables will, in general, degrade the running time for operations accessing these records by a factor of k. In order to simplify the discussion, we assumed here that each cell contains at most one record.

The cell network can be varied based on the requirement of the applications The MASC model does not require a specific cell network and the best choice depends on the requirements of the application. Many of the algorithms (e.g., the convex hull algorithm [1] or the minimum spanning tree algorithm [6]) do not require the use of the cell network. The VLDC string matching algorithm [3] requires a 1-D cell network and the 2-D Knapsack [9] problem requires a 2-D cell network. Fortunately, most standard networks can efficiently simulate other standard networks.

Further information about the MASC model can be found in [6, 5, 4]

3. An ASC Language Compiler Extension for the MASC Model

The Associative Computing (ASC) language was developed during the 1980's to provide a language for associative SIMD computers. The SITDAC (Single Instruction Tabular Data Associative Computing) model in [7] provides an architectural model for an associative SIMD computer. The word "tabular" reflects the fact that data is stored in a two dimensional tabular data structure. An associative searching is efficiently supported using hardware features rather than an associative memory [5]. After the MASC model was introduced, its restriction to one instruction stream (and eventually called ASC) provided a computational model for the associative SIMD computer [6]. The ASC language was initially designed as a parallel language, rather than an extension to an existing sequential language. The ASC language has been implemented on two associative SIMD computers (STARAN and ASPRO) and two SIMD computers (Connection Machine CM2 [7] and the WaveTracer).

When compiling an ASC program using the ASC compiler, the compiler will generate an intermediate code. This intermediate code is intended to run on an associative SIMD, e.g., the ASPRO, a traditional SIMD, or an ASC emulator which emulates a parallel associative computer. Due to the introduction of the multiple instruction streams for the ASC model in the MASC model, a new compiler is needed to compile an ASC program and produce an intermediate code that can be executed on a MASC computer.

```
IF parallel condition expression
   THEN  block A
   ELSE  block B
ENDIF;
```

Figure 2: The format of a parallel IF-THEN-ELSE statement in the ASC language.

In the ASC language, the branches in a parallel IF-THEN-ELSE statement are independent and can be executed simultaneously (in data parallel fashion) provided two instruction streams are available. An IF-THEN-ELSE statement is a parallel IF-THEN-ELSE statement if the result after the evaluation of the condition expression produces a parallel logical result. If the result is a scalar logical result, the statement is a scalar IF-THEN-ELSE statement. Only the THEN part or the ELSE part of a scalar IF-THEN-ELSE statement will be executed, but not both. Unlike a scalar IF-THEN-ELSE statement, both block A in the THEN part and block B in the ELSE part of the parallel IF-THEN-ELSE statement in figure 2 are executed.

Although a parallel CASE statement is not currently supported in the ASC language, the branches in the parallel CASE statement can also be executed in parallel provided at least two instruction streams are available. This process is similar to that of the branches in a parallel IF-THEN-ELSE statement, and is not addressed here.

Normally, in order to execute a parallel IF-THEN-ELSE statement in the ASC model, the instruction

stream searches for responders that satisfy the condition expression. The responders are active and execute block A. Then, the instruction stream does another search for non-responders to the previous search which now become active responders, and execute block B. This second search is normally done by complementing the responder bits (for responders that did conditional test). This converts responders into non-responders and conversely. However, any processor that is inactive at the time the conditional statement is evaluated remains inactive during the execution of the IF-THEN-ELSE statement and does not execute either block A or B. Note that, the phases of testing the parallel condition expression, executing the THEN part, and executing the ELSE part are done sequentially, see figure 3.

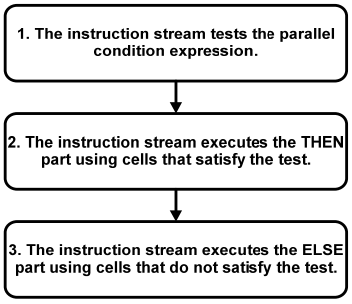

Figure 3: Execution of a parallel IF-THEN-ELSE in the ASC model or a typical SIMD computer.

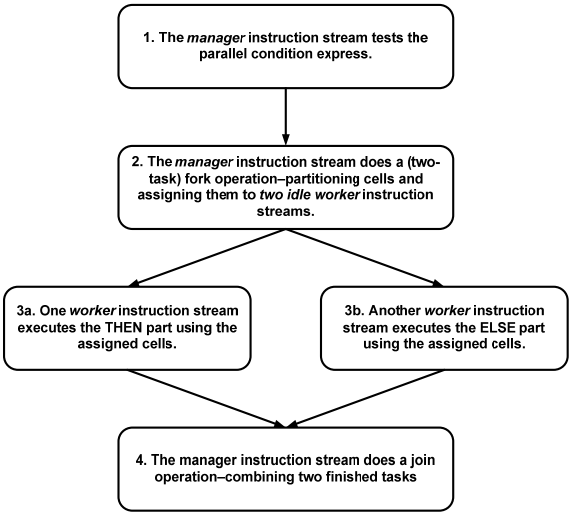

Figure 4: Execution of a parallel IF-THEN-ELSE in MASC model.

If we have available instruction streams, both block A and block B can be executed simultaneously using one instruction streams and a partition of cells (responders) for the THEN part and another instruction stream and another partition of cells (non-responders from the testing of the conditional expression) for the ELSE part, see figure 4.

When creating an ASC program for the MASC model, a user needs to place a MASC directive just before that parallel IF-THEN-ELSE statement to enable simultaneously executions of both the THEN part and the ELSE part of a parallel IF-THEN-ELSE statement in the ASC language, see figure 5. The format of the directive is */* .MASC fork */*. The directive starts with /*, then .MASC fork, and ends with */. At least one white space must be used to separate between ".MASC" and "fork". The directive used in this MASC backend compiler is not case-sensitive and it is considered as a comment by the ASC compiler.

```
/* .MASC fork */
IF parallel condition expression
   THEN  block A
   ELSE   block B
ENDIF;
```

Figure 5: The format of a parallel IF-THEN-ELSE statement with the MASC directive used.

A MASC directive is used to support user-directed concurrent executions of branches. The user must explicitly specify the directive for a parallel IF-THEN-ELSE statement in the program so the runtime system will execute the branches concurrently. A parallel IF-THEN-ELSE statement is expected to immediately follow this directive. Otherwise, during the compile time, an error message will be generated if the statement immediately following the directive is not a parallel associative IF-THEN-ELSE statement.

When using this directive with a parallel associative IF-THEN-ELSE statement, the MASC backend compiler looks for the directive and generates exactly two tasks to be executed concurrently, one from the THEN part and another from the ELSE part. If there is another parallel IF-THEN-ELSE statement embedded within either the THEN part or the ELSE part, that parallel IF-THEN-ELSE statement will not be effected by this /* .MASC fork */ directive. If the user wants the MASC backend compiler to generate two more tasks from that embedded IF-THEN-ELSE statement, another instance of /* .MASC fork */ directive is required.

An ASC program with MASC directives will first be compiled by an ASC compiler and then by the ASC language compiler extension or the MASC backend compiler. The intermediate code from the MASC backend compiler will be executed on a MASC prototype, or a MASC emulator. An

intermediate code from the ASC compiler will be analyzed by the MASC backend compiler. Two new instructions, .MI_BEGIN and .MI_END, will be inserted into the intermediate code. These two instructions are the same one used in [8].

In this context, .MI_BEGIN marks the beginning of a block of intermediate code that will be executed by an instruction stream. .MI_END marks the ending of the intermediate code block. A block of intermediate code no longer has to be a basic block mentioned in [8]. It can now contain one or more embedded parallel IF-THEN-ELSE statements.

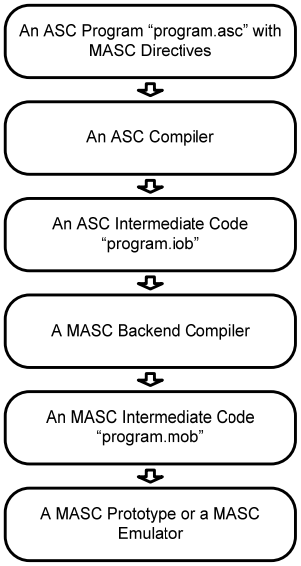

Figure 6: The process of compiling and executing an ASC program with MASC directives.

Every .MI_BEGIN and .MI_END instruction sets will be followed by a task id. A task id has two parts in it. The first part can be identified by a character M or W. The second part is a sequence of characters in the set {X, 0, 1, 2}. A structure code is used for the sequence number in the second part. Note that the depth of the task graph indicates the length of this task id.

In the first part of the id, M indicates that this task is for the manager-IS. W indicates that this task is for any worker-IS. In the second part of the id, a sequence of all 0s indicates a null task id. X indicates that this task has two parents--resulting from a join operation of two tasks. A sequence should be examined from left to right. If the last non-zero character is 1, this task is the left child or the only child. If the last non-zero character is 2, this task is the right child of its parent. Moreover, the first task id always is M followed by 1 and 0s.

As an example, if the task id is W1120000, the information one can get from this task id is (1) it is a worker-IS task because of the W character and (2) it is the right child task or the ELSE part of the branch of task {M or W}1100000. For another example, if the task id is W111X100, the information one can get from this task id are (1) it is a worker-IS task because of the W character and (2) it is the left child task or the only child of tasks {M or W}1111000 and {M or W}1112000.

The benefit of using a structure code for the task id is it gives a concept of associations. That is, a task id contains structural information on how this task relates logically to other tasks in the system. The manager-IS can use this information to search for the next task to be added into the work pool based on the previously finished task or what tasks that have to be joined back together while maintaining a constant time associative searching.

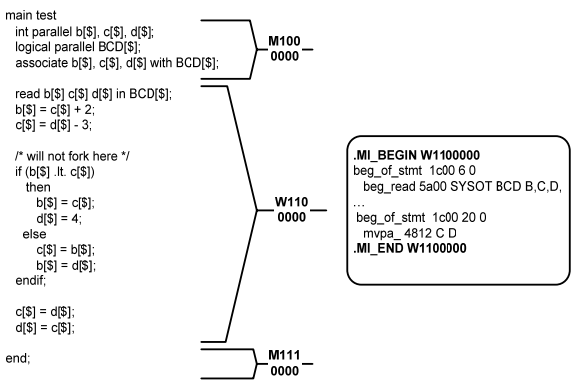

Figure 7: An ASC program with no MASC directive used. The intermediate code of task W1100000 is shown with new MASC instructions (in bold). All tasks have to be executed in order, i.e., first M1000000, second W1100000, and then M1110000.

Suppose we have an ASC program as shown in figure 7. In the ASC model, the instruction stream will take one instruction at a time from the beginning of the intermediate code and broadcast it to cells. When the instruction is the last line of the intermediate code (it is ***endmain 7400 test*** for this example program), the execution of the program is finished.

If we use a MASC model to execute the ASC program in figure 7, first, the manager-IS will execute the intermediate code of task M1000000 (the block of code starting with .MI_BEGIN M1000000 and ending with .MI_END M1000000). When done, the manager-IS needs to do a one-task fork operation for the next task, W110000.

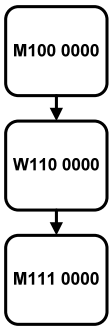

Figure 8: A task graph from the ASC program in figure 7.

The basic steps of a one-task fork operation are the following.

- Partitioning cells by perform an associative search on cells that satisfy and on those that do not satisfy the condition--in this case, the condition is TRUE. All active cells satisfy this condition.
- Performing another search for an idle worker-IS and, if found, proceed to the next step.
- Switch cells to listen to the worker-IS found in previous step.
- Send a signal to the worker-IS to start the execution of the assigned task.

When task W1100000 has been finished, cells are switched back to the manager-IS. The worker-IS enters the idle state. Then, the manager-IS does the next task, M1110010, which is exiting the program.

```
main test
   int parallel b[$], c[$], d[$];
   logical parallel BCD[$];
   associate b[$], c[$], d[$] with BCD[$];

   read b[$] c[$] d[$] in BCD[$];
   b[$] = c[$] + 2;
   c[$] = d[$] - 3;

   /* .MASC fork */
   if (b[$] .lt. c[$])
     then
        b[$] = c[$];
        d[$] = 4;
     else
        c[$] = b[$];
        b[$] = d[$];
   endif;

   c[$] = d[$];
   d[$] = c[$];

end;
```

M100 0000
W110 0000
M111 0000
W111 1000
W111 2000
W111 X100
M111 X110

```
.MI_BEGIN W1112000
beg_of_stmt  1c00 16 0
  mvpa_ 4812 B C
beg_of_stmt  1c00 17 0
  mvpa_ 4812 D B
.MI_END W1112000
```

Figure 9: The same ASC program as in figure. 7 but with a MASC directive used. The intermediate code of task W1112000 is shown. Tasks W1111000 and W1112000 can be executed simultaneously by two different worker instruction streams.

Suppose we have the ASC program shown in figure 9. The program has a MASC directive before the parallel IF-THEN-ELSE statement. The ASC model will execute this program the same way it does with the program in figure 7 because the ASC model does not see any difference between the programs in figures 7 and 9. On the other hand, the MASC model will execute this program differently than the program in figure 7.

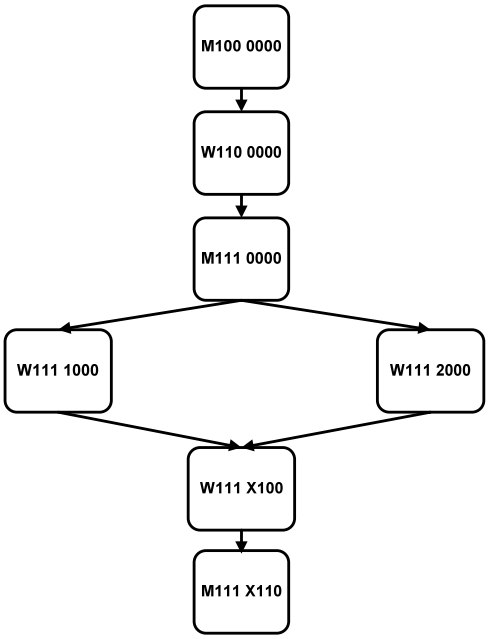

Figure 10: A task graph from the MASC program in figure 9.

In the MASC model, the manager-IS will start off with the execution of the first task M1000000. Then, it does a one-task fork operation, which is explained earlier, for task W1100000. When task W1100000 is finished, the manager-IS does a two-task fork operation. This two-task fork operation is similar to the previously explained one-task fork operation. The only difference is the testing of the parallel condition expression *b[$] .lt. c[$]* (testing if the components of the parallel variable *b* are less than the components of the parallel variable *c* instead of the parallel condition expression TRUE). All active cells that satisfy the parallel condition expression will be assigned to task W1111000. Any active cell that does not satisfy this condition will be assigned to task W1112000. If there are at least two available worker-ISs, both tasks can be executed simultaneously. When these two tasks are finished, the manager-IS does a one-task fork operation for task W111X110. After task W111X100 is finished, it executes task M111X110 which is exiting the program.

In order to compare the MASC performance of the two programs in figures 7 and 9, let N_{then} be the number of instructions executed in the THEN part and N_{else} be the number of the instructions executed in the ELSE part. Then m = maximum $\{N_{then}, N_{else}\}$ is the maximum number of instructions in the THEN and ELSE parts. Let h = the over head incurred from doing a fork and a join operations. The benefit gained (in term of the number of instructions executed) from using multiple instruction streams to execute a parallel IF-THEN-ELSE statement is $k = (N_{then} + N_{else}) - (m + h)$. If k is positive, it is beneficial to use the MASC program in figure 9 that uses multiple instruction streams to execute this parallel IF-THEN-ELSE statement.

4. Summary and Future Work

This work on the ASC language complier extension or a MASC backend compiler using a MASC directive is one-step closer to enable researchers to write and test their MASC algorithms and programs on a real MASC computer or a MASC prototype. Using multiple instruction streams to execute the THEN part and the ELSE part of a parallel IF-THEN-ELSE simultaneously should improve the performance of the system over the execution with one instruction stream as long as the time to compute both the body of either the THEN part and the body of the ELSE part is greater then the overhead incurred from doing a fork and a join operation.

Our next project is to build a run time system of the MASC model using manager-worker paradigm. Work on developing a MASC prototype using FPGA and creating a MASC emulator that emulates exact MASC hardware's behavior in order to test and evaluate the runtime of both ASC programs intended for an ASC model and ASC programs with a MASC directive used for the MASC model is in progress.

A MASC emulator is projected to be a modified version of the existing ASC emulator that uses manager/worker paradigm for the instruction streams and the idea of a pool of tasks (a work pool). At that stage, many scheduling policies for the pool of tasks will be tested on the emulator to determined which one is the best policy under which circumstance, i.e., under resource constraint. The overhead incurred from a fork or join operation will be analyzed at this stage.

A research group in the Computer Science Department at KSU directed by Dr. Robert Walker is building a MASC prototype using FPGA. When that project is done, this MASC backend compiler may be adapted to be used for the prototype as well.

5. References

[1] M. Atwah, J. Baker, and S. Akl, "An Associative Implementation of Classical Convex Hull Algorithms," in *Proc. of the Eighth IASTED International Conference on Parallel and Distributed Computing Systems*, October. 1996. pp. 435-438.

[2] W. Chantamas and J. Baker, "A Multiple Associative Model to Support Branches in Data Parallel Applications using the Manager-Worker Paradigm," in *Proc IPDPS 2005, p. 266b, 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 14*, 2005.

[3] M. Esenwein and J. Baker, "VLDC String Matching for Associative Computing and Multiple Broadcast Mesh," in Proc. of *the IASTED International Conference on Parallel and Distributed Computing and Systems*, 1997, pp. 69-74.

[4] M. Jin, "*Evaluating the Power of the Parallel MASC Model using Simulations and Real-time Applications*," a dissertation submitted to Kent State University, August 2004.

[5] M. Jin, J. Baker, and K. Batcher, "Timing for Associative Operations on the MASC Model," in *Proc. of the 15th International Parallel and Distributed Processing Symposium (Workshop in Massively Parallel Processing)*, April 2001.

[6] J. Potter et al., "ASC: An Associative-Computing Paradigm," *IEEE Computer*, November 1994, pp. 19-25.

[7] J. Potter, "*Associative Computing: a Programming Paradigm for Massively Parallel Computer*," Plenum Press, New York, 1992.

[8] M. Scherger, J. Baker, and J. Potter, "Multiple Instruction Stream Control for an Associative Model of Parallel Computation," in *Proc. of the 16th International Parallel and Distributed Processing Symposium (Workshop in Massively Parallel Processing)*, April 2003.

[9] D. Ulm and J. Baker, "Solving a 2D Knapsack Problem on an Associative Computer Augmented with a Linear Network," in *Proc. of the International Conference on Parallel and Distributed Processing Techniques and Applications*, 1996, pp. 29-32.