A Multiple Associative Model to Support Branches in Data Parallel Applications using the Manager-Worker Paradigm

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

Abstract

ASC (Associative Computing Model) and MASC (Multiple Associative Computing Model) have long been studied in the Department of Computer Science at Kent State University While the previous studies provide the background and the basic definition of the model, the description of the interactions between the instruction streams (ISs) is very brief, high level, and incomplete. One change here is that we specify the interaction between ISs and consider that all of the ISs operate on the same clock in order to support predictable worst case computation times, while earlier the ISs were assumed to interact in a MIMD type fashion. This paper provides a detailed explanation as to how these interactions can be supported in the case where only a few ISs are supported.

1. Introduction

The MASC parallel computational model is an extension of the associative or ASC model, whose original motivation was the STARAN associative computer in the early 1970's. The MASC model supports massively parallelism using an associative, multiple SIMD style of computation. The associative properties of MASC support constant time searching for data in the memory by content rather than by location. (However, MASC does not assume an associative memory.) Additionally, MASC is able to perform MIN/MAX and logical reductions in constant time, [Potter, 4], and [Jin, 16].

We will consider two variations of the MASC model. The first one is the model with a potentially unlimited number of instruction streams, which increases as a function of the number of PEs (e.g., logarithmic with respect to the number of PEs). The second variation of the MASC model has a small, fixed number of instruction streams.

The first variation has been proved to be an extremely powerful model [Atwah, 2], [Baker, 13], and [Ulm, 14]. While the second variation is less powerful, it is still extremely useful, and allows for the simultaneous executions of most or all branches of typical data-parallel applications. The current focus is on the second variation because it is simpler and more practical to build. The support for the IS-interactions proposed here for the second version uses one manager-IS and a small number of worker-ISs and utilizes the work pool and manager-worker paradigm.

2. Properties of the model

The basic components of the model are a set of instruction streams, an array of cells, an instruction stream network, a cell network, and a broadcast/reduction network capability for each instruction stream (see Figure 1).

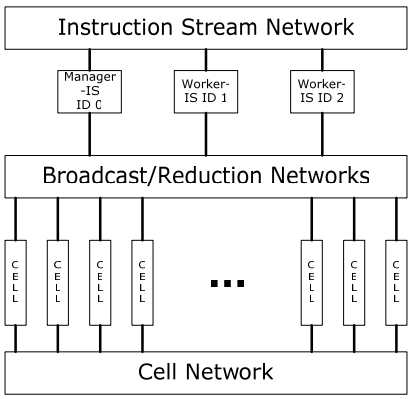

Figure 1. MASC with one manager-IS and two worker-ISs

Instruction streams consist of a manager instruction stream (manager-IS) and a small number

of identical worker instruction streams (worker-IS). The manager-IS and worker-ISs are processors and are able to fetch, decode, and broadcast instructions to PEs using their broadcast networks.

The instruction stream with *ID 0* is dedicated to be the manager-IS and the rest of the instruction streams are the worker-ISs (*ID* 1 to *ID m*, if the total number of instruction streams is m+1).

The main functions of the manager-IS are managing the work pool of tasks, coordinating and assigning subtasks (using a FORK operation), and combining finished tasks (using a JOIN operation) from worker-ISs. The manager-IS uses parts of its memory for *the work pool queue* (WP-Q) and *the idle worker-IS queue* (idle-WIS-Q) and it can access the first items in each of the queues using a queue operation in unit time. Newly created subtasks will be inserted at the end of the *work pool queue*. The first subtask in the front of the *work pool queue* will be assigned to the first worker-IS in the front of the *idle worker-IS queue*. When a worker-IS finishes executing a task, it notifies the manager-IS and sets itself to ready. Then, the manager inserts the ID of that worker-IS at the end of *the idle worker-IS queue*.

The main functions of a worker-IS is to execute tasks in an associative (e.g., data parallel, SIMD) fashion using the PEs currently assigned to it. A worker-IS can be either ready or busy. A ready worker-IS always waits for the manager-IS to assign a task. A busy worker-IS is currently executing a task with its set of PEs (see Figure 2). Each instruction stream has the same copy of the program but two instruction streams may execute different subtasks simultaneously.

A part of each PE's local memory will be dedicated for the *Task-History Stack* (THS) whose default content is empty. The top of the stack always shows the current task the PE is currently executing, the task that has just been finished when the manager-IS has not determined which next task the PE belongs to, or new task that has not yet assigned to a worker-IS.

Each cell consists of a processing element (PE) and its local memory. PEs are identical and are very simple, i.e., basically ALUs. Each PE contains the *IS-Selector Register* (default is 0) which holds the instruction stream ID, to which that PE is currently listening. The *IS-Selector Register* can be set or reset by an instruction stream, i.e., the instruction stream that the PE is listening to instruct the PE to reset its *IS-Selector Register* to 0 (the manager-IS ID) or set to anything from 1 to *m*, where *m* is the number of worker-ISs and each worker-IS has an IDs from 1 to *m*.

At any point in time, each PE listens to exactly one IS, creating a dynamic partition of the PEs, with each set of PEs in the partition listening to the same IS. At first, all PEs listen to the manager-IS, since the default content of the *IS Selector Register* is 0, which is the ID of the manager-IS. Some or perhaps all of PEs will be activated at the beginning of the execution of the program.

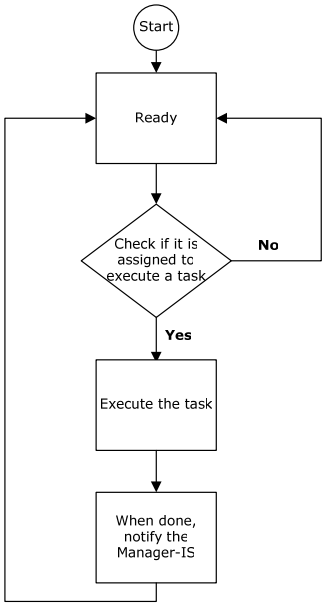

Figure 2. A worker-IS flowchart

Instruction streams use the instruction stream network to communicate to each other. This network may be a simple point-to-point bus or a more complicated network such as a broadcast/reduction network. Normally, a communication between instruction streams is the one between the manager-IS and worker-ISs, i.e., assigning of a new task or receiving a notification for a finished task.

The cell network can be a simple bus, linear array, or any other network suitable for intended applications. Many ASC (i.e., MASC model restricted to one IS) applications such as convex hull algorithm or minimum spanning tree algorithm do not required the use of the cell network, but some such as FFT algorithm requires an extensive amount of network usage, [Atwah, 2], [Potter, 4], [Ulm, 14, 15], and [Esenwein, 17].

Each instruction stream is logically connected to all PEs by its broadcast/reduction network (which may be two separate networks, one for the broadcast

and another for the reduction). An instruction stream can detect responder cells (if the data in the cells satisfy the condition) and can select an arbitrary responder cell in constant time using this broadcast/reduction network.

Intended applications for the MASC are data parallel in nature and may involve branches that can be executed in parallel using multiple instruction streams. In general, the number of branches existing in most real applications at any point in the execution time is small [Fox, 11, 18]. Having only a small number of instruction streams in this model allows most or all of the branches to be executed simultaneously.

Most SIMD computers allow masking of PEs while determining whether or not that PE should participate in the operation in a parallel **IF-THEN-ELSE** statement (or a parallel **CASE** statement, which is a generalized "parallel IF-THEN-ELSE" statement and can be handled similarly). Suppose we have a parallel **IF-THEN-ELSE** statement as shown in the Figure 3. If the system has at least two available instruction streams, one instruction stream will execute the **THEN** part using the PEs that satisfy the **IF** condition and another will execute the **ELSE** part using the PEs that do not satisfy the condition; moreover, these two data-parallel computations are executed concurrently. Thus, both sets of PEs complete their part of the computation prior to the time they are joined back together at the end of the **IF-THEN-ELSE** statement.

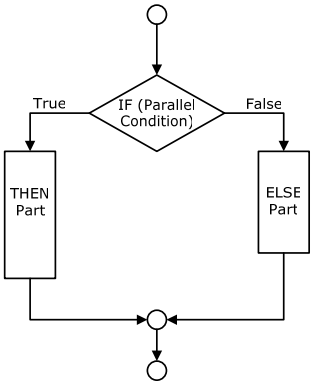

Figure 3. A parallel IF-THEN-ELSE statement

A simple blind task assignment to an instruction stream is assumed as the default assignment scheme. This assignment scheme will select the first task in the *work pool queue* and assign it to the first idle worker-IS in the *idle-worker-IS queue*. A task may consist of multiple subtasks. Each of these subtasks may also contain multiple smaller subtasks. A predetermined assignment or the programmer's choice scheme is also allowed in the model. There is no preemption of tasks. Once a worker-IS is assigned a task to be executed, the worker-IS must be allowed to complete the execution before it is re-assigned. There is no interaction between instruction streams either between worker-ISs during the executions of separate branches of the program or between the manager-IS and a worker-IS during the execution of the given task by the worker-IS.

3. Supported operations: FORK and JOIN

A *subtask identification process* can be done offline for the current ASC compiler, i.e., as a backend of the compiler. The process is dependent on the structure of the program. It identifies different through branches (e.g., a parallel IF-THEN-ELSE statement) that can be executed concurrently by disjoint sets of PEs which are controlled by different worker-ISs. This identification process will produce *a task graph*, which provides the workflow of the algorithm and shows subtasks along with their order of execution.

The identification can produce what will be called either *fine grain tasks* or *coarse grain tasks*. A fine grain subtask will contain no branch (e.g., no parallel IF-THEN-ELSE statement) within that subtask. Each subtask may contains simple assignment statements, parallel FOR statements, parallel WHILE statements, and LOOP statements, [Potter, 6]. Each of these subtasks will be assigned to worker-ISs. On the other hand, a coarse grain subtask may contain a number of branch statements within that subtask. One might think of a coarse grain subtask as a group of related fine grain subtasks bundled together and different parts of its execution may be assigned to different worker-ISs.

When the program that will be executed in the system has a branch such as a parallel IF-THEN-ELSE statement, then assuming that sufficient idle worker-ISs are available, each path of the branch can be assigned to different instruction streams and be executed concurrently. In this research, a FORK operation is an operation that generates one or more subtasks from a branch; later, those subtasks will be assigned to worker-ISs to be executed concurrently, provided sufficient worker-ISs are available. A JOIN operation will later recombines those subtasks into the original parent task (i.e., the one existing prior to the fork) after they have been successfully executed.

During a FORK operation, the manager-IS assigns one path of the branch to one worker-IS and other paths to other idle worker-ISs. PEs will be partitioned

into groups based on the subtasks they are executing. The different PEs in each set within in the partition depend upon on the content of the PEs and the previous subtask those PEs have been in. The assignment of subtasks also involves deletion of subtask from the WP-Q and the worker-IS ID from the idle-WIS-Q maintained by the manager-IS as described in Figure 4.

When a worker-IS notifies and returns its PEs back to the manager-IS, the manager-IS tries to combine (join) the subtask with other subtasks forked from the same branch. The JOIN operation is success if all of the required subtasks have been returned to the manager-IS. Otherwise, the JOIN operation is postponed.

4. Quickhull algorithm using manager-worker paradigm

In this section, the associative ASC-quickhull [Atwah, 2] is modified to fit the MASC model with a limited number of instruction streams and using work pool and manager-worker paradigm. In [2], O(lg n) instruction streams were used.

The convex hull of a set of points S is the smallest convex set containing S. In particular, each point of set S is either on the boundary of or in the interior of the convex hull.

To make it easier to describe the algorithm, we assume that no two points in S have the same (x,y) coordinates and no three points in S lie on the same line segment. As in [2], in order to keep this algorithm simple, it only identifies the convex hull points and does not produce an ordered (e.g., clockwise) list of the convex hull points. However, as observed in [2], it is easy to modify this algorithm so that an ordered list of convex hull points is produced.

The parallel variables used in this algorithm are x\$, y\$, p\$, q\$, e\$, *area*\$, and *in_hull*\$. (The "\$" suffix is used to identify parallel variables from IS variables.) Here. x\$ is x-coordinate and y\$ is y-coordinate of the point stored in each of the PEs. Also, p\$ and q\$ are x and y coordinates of the left point and the right point, respectively, and w\$ and e\$ are x and y coordinates of the leftmost point and the rightmost point, respectively. The area of triangle *pra is stored in area*\$. If *in_hull*\$ is true, then the point stored in that PE is in the convex hull.

At the beginning of the execution, all PEs have x[\$] as the x-coordinate and y[\$] as the y-coordinate for the point stored in the PE, the variables p[\$], q[\$], w[\$], and e[\$] are empty; and *in_hull*[\$] is false.

Algorithm MASC-Quickhull (for the upper hull)

Input: A set of points S given as (x,y) coordinates, each PE holds one point in S
Output: vertices of the upper convex hull

A) The manager assigns the initialization task (i.e., $task_0$) to a worker IS, who performs steps 1-3 using all PEs.

1. All PEs compute x-max and x-min
2. Restrict to one PE where x\$ equals x-max, using minimum y\$ as the tie-breaker
   a Set *in_hull*\$ = *true*
   b Broadcast x\$ and y\$ of this PE to every PEs, and each stores x\$ and y\$ in e\$
3. Restrict to one PE where x\$ equals x-min; if more than one such PE exists, use minimum y\$ as the tie-breaker
   a Set *in_hull*\$ = *true*
   b Broadcast x\$ and y\$ of this PE to every PEs, and each store x\$ and y\$ in w\$

B) The manager creates $task_{we}$ and places it in the work pool queue. The PEs associated with this task are the ones whose point lies above segment *we,* as specified in Figure 5.

C) The manager assigns each $task_{pq}$ in the work pool to a worker IS, who performs the following steps using the PEs assigned to this task.

4. For each PE, compute *area*\$ = the area of triangle *prq*, where r is the point (x\$, y\$) stored in that PE
5. Restrict to one PE whose *area*\$ is maximum; if more than one such PE exists, use maximum y\$ as the tie-breaker
   Set *in_hull*\$ = *true*

D) The manager places $task_{pr}$ and $task_{rq}$ in the work pool queue. The PEs associated with each task are the ones whose point lies above corresponding line segment (See Figure 5).

E) The manager continues to execute steps C and D until there are no active tasks and no tasks remain in the work pool, and then terminates the algorithm. See Figure 4 for more details.

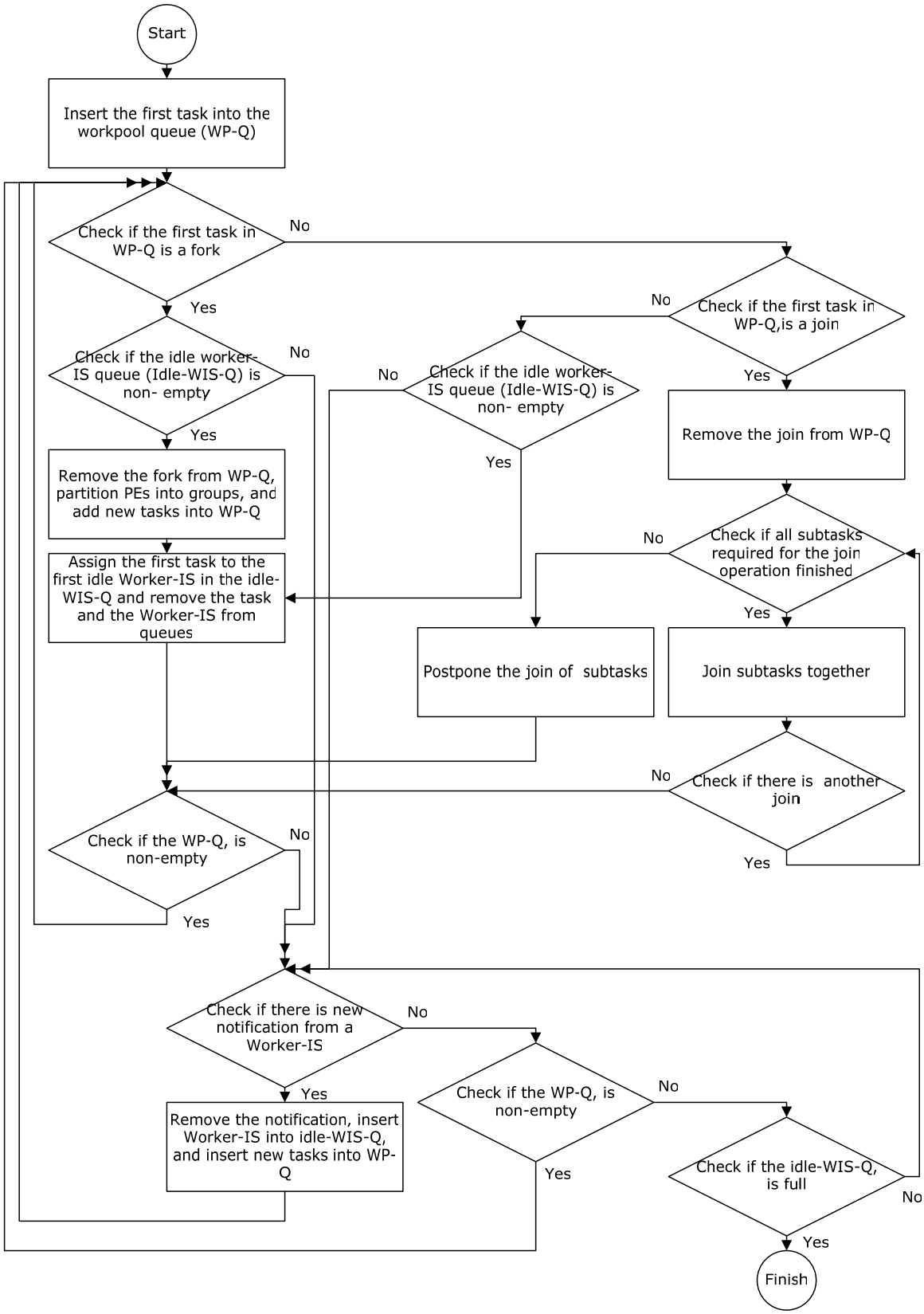

Figure 4. The manager-IS flowchart

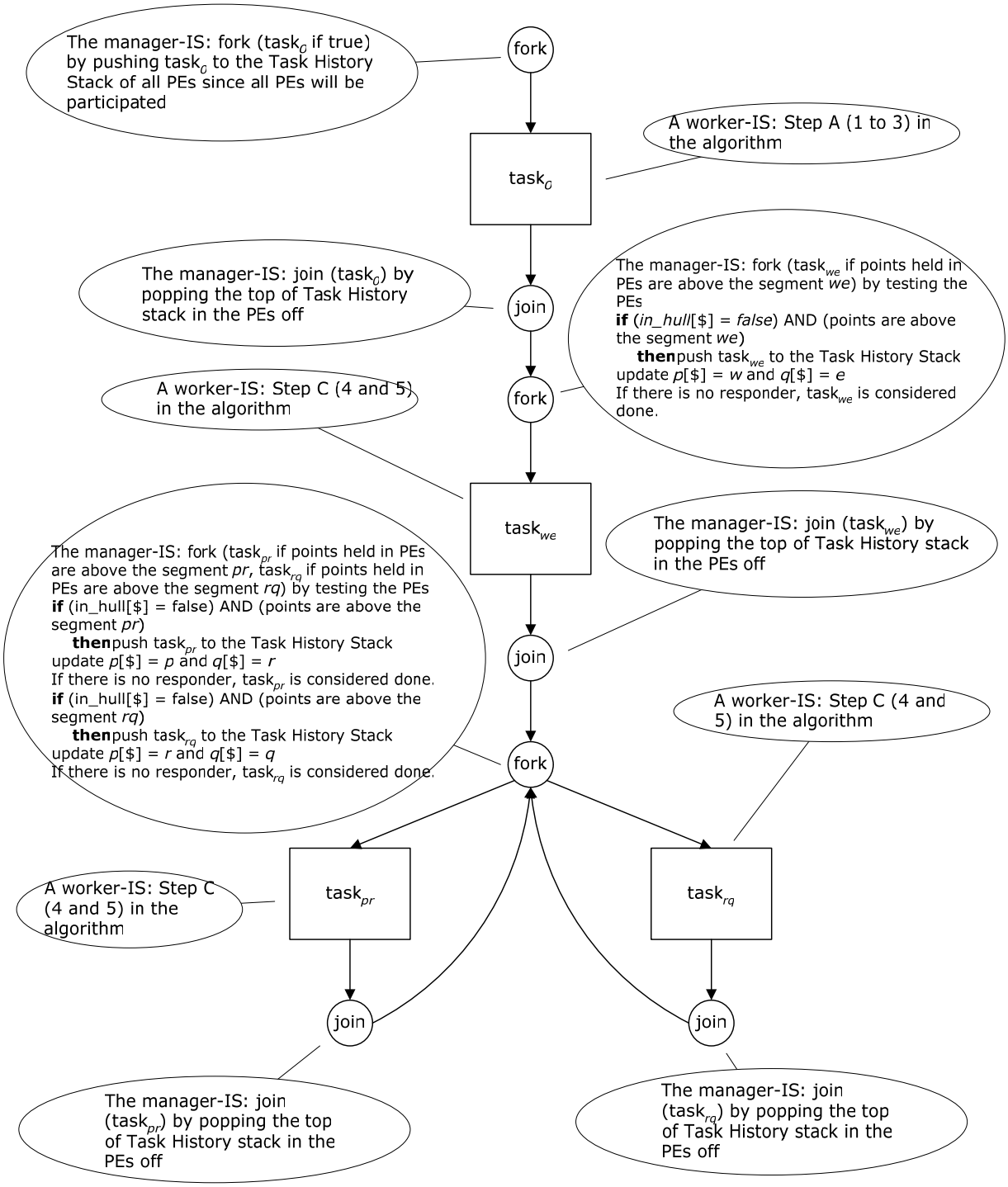

Figure 5: The MASC quickhull algorithm workflow

Step A (1 to 3) is executed by one worker-IS. Step C (4 and 5) can be executed by more than one worker-IS concurrently with the disjoint sets of PEs. The task graph of the MASC quickhull algorithm in shown Figure 5 provides further details for this algorithm.

The running time for the above algorithm is $O(n/m)$ in the worst case where n is the number of points in S and m is the number of instruction streams. If we assume that on the average $O(lg\ n)$ is the number of convex hull points, there will be $O(lg\ lg\ n)$ levels of recursions and $O(2^0+2^1+2^2+\ldots+2^{lg\ lg\ n})$ convex hull points will be identified[2]. If m ISs are available, the average case running time is $O((lg\ lg\ n)(lg\ n)/m)$ and the average cost is $O((n + m)(lg\ lg\ n)(lg\ n)/m)$. Since m is considered to be a small positive constant, one may reduce the average cost to $O((n)(lg\ lg\ n)(lg\ n))$. However, this algorithm still produces a constant speedup of approximately m over the 1-IS version, according to preceding observations. A speedup of m is the maximum speedup that that can be expected since the algorithm is using only m ISs.

5. Conclusion

The MASC model was introduced in [Potter, 4], but the interactions between the IS were described at a very high level. This paper provides a detailed description of how the interactions between the instruction streams for the MASC model could be supported in the restricted case where only a few instruction streams are allowed. These interactions are supported using a manager instruction stream to coordinate the activities of the worker instruction streams. This information is expected to be useful in providing an implementation of the MASC model with a small number of ISs.

The explanations of components of the MASC model along with their properties are stated for what the model is allowed or not allowed to do. Two instruction stream operations, FORK and JOIN, are also defined in this paper. A version of MASC quickhull algorithm for this restricted model with a small number of instruction streams is also presented to show the usefulness, power, and ease-of-programming this model. A number of papers have been published about MASC, including simulations of other models using MASC [Baker, 13], [Ulm, 14], algorithms for MASC [Maher, 2], [Ulm, 14], and further information about the model [Jin, 16], and [Scherger, 10].

6. References

[1] L. G. Valiant, "A Bridging Model for Parallel Computation," *Communication of the ACM*, vol. 33, no. 8, August 1990.

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

[3] K. Batcher, "Architecture of a Massively Parallel Processor," in *25 Years of the International Symposium on Computer Architecture: Selected Papers*, Gurindar Sohi (ed.), ACM Press, 1998. pp. 147-149.

[4] J. Potter, J. W. Baker, S. Scott, A. Bansal, C. Leangsuksun, and C. Asthagiri, "ASC: An Associative-Computing Paradigm," IEEE Computer, November 1994, pp. 19-25.

[5] Krikelis, A., and C. C. Weems, *Associative Processing and Processors*, IEEE Computer Society Press, 1994.

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

[7] Garey, M. R., and D. S. Johnson, *Computer and Intractability: A Guide to the Theory of NP Completeness*, W. H. Freeman, New York, 1979.

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

[9] W. Meilander, J. Baker, and M. Jin, "Importance of SIMD Computation Reconsidered," in *Proc. of the 17th International Parallel and Distributed Processing Symposium (Workshop on Massively Parallel Processing)*, to appear, April 2003.

[10] 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)*, to appear, April 2003.

[11] Fox, G. C., R. D. Williams, and P. C. Messina, *Parallel Computing Works*, Morgan Kaufmann Publishers, CA, 1994.

[12] M. Jin, J. W. Baker, and W. Meilander, "The Power of SIMDs vs. MIMDs in Real-Time Scheduling," in *Proc. of the 16th International Parallel and Distributed Processing Symposium (Workshop in Massively Parallel Processing)*, April 2002.

[13] J. Baker and M. Jin, "Simulation of Enhanced Meshes with MASC, a MSIMD Model," in *Proc. of the 11th International Conference on Parallel and and Distributed Computing Systems*, November. 1999, pp. 511-516.

[14] D. Ulm and J. Baker, "Simulating PRAM with a MSIMD Model (ASC)," in *Proc. of the International Conference on Parallel Processing*, August 1998, pp. 3-10.

[15] 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.

[16] 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.

[17] M. C. 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.

[18] G. C. Fox, “What Have We Learnt from Using Real Parallel Machines to Solve Real Problems?,” in G. C. Fox, editor, *The Third Conference on Hypercube Concurrent Computers and Applications, Volume 2,* Caltech Report C3P-522, ACM Press, New York, January 1988, pp. 897-955.