Parallel Circuits Simulation of Adders

Final Report

Wenjie Yao
Georgia Institute of Technology
Atlanta, Georgia
wyao48@gatech.edu

Jidong Li
Georgia Institute of Technology
Atlanta, Georgia
jli807@gatech.edu

Chengcheng Zhang
Georgia Institute of Technology
Atlanta, Georgia
czhang461@gatech.edu

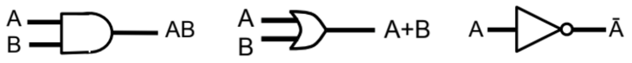
Yiou Zhu
Georgia Institute of Technology
Atlanta, Georgia
yzhu455@gatech.edu

1 PROBLEM DESCRIPTION

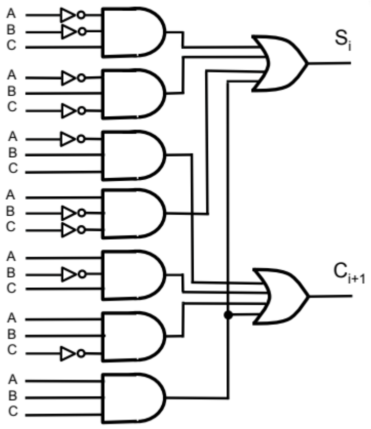
Addition of two numbers can be represented by binary addition of two n bit numbers, say the first number is $A_{n-1}A_{n-2}...A_1A_0$, and second number is $B_{n-1}B_{n-2}...B_1B_0$ and their sum is $S_{n-1}S_{n-2}...S_1S_0$, where A_i, B_i, S_i are 0 or 1 for i = 0, fi n. To conduct the calculation, a one-bit full adder can be used to perform the addition of the numbers bit by bit. It takes three one-bit numbers, A_i, B_i and C_i as input and output C_{i+1} and S_i, where A_i and B_i are the operands and C_i is the overflowed bit from previous stage. The logic of the full adder can be represented by the combination and execution of logical gates as follows:

$$\begin{aligned} S_i &= (A_i\ XOR\ B_i)\ XOR\ C_i \\ &= \overline{A_i}\overline{B_i}C_i + \overline{A_i}B_i\overline{C_i} + A_i\overline{B_i}\overline{C_i} + A_iB_iC_i \\ C_{i+1} &= A_i\ AND\ B_i\ OR\ C_i\ (A_i\ XOR\ B_i) \\ &= \overline{A_i}B_iC_i + A_i\overline{B_i}C_i + A_iB_i\overline{C_i} + A_iB_iC_i \end{aligned}$$

According to the definition of logical gates, we have following operators to represent AND, OR and NOT, respectively.



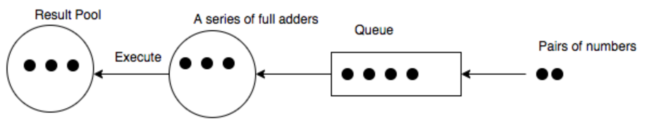
The calculation of Si and Ci+1 can be represented as following:



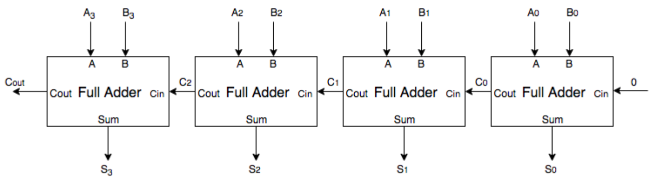
Therefore, it is a full adder by definition. The aim of our project is to simulate the computation of a series of full adder to calculate the addition of pairs of numbers. Concerning problems are synchronization and whether parallel computation can be applied to the adder. For the synchronization problem, message C_i needs to be sent and received by next full adder to guarantee the calculation sequence. For the efficiency, we plan to study whether we parallelize the computation of the 7 logical gates above can improve the overall efficiency of computation since parallel will involve the overhead and integration of results, which is also costly. In a word, our project is to study how parallel computation influences the efficiency of adders by simulating the circuits of adders using logical gates.

2 CONCEPTUAL MODEL

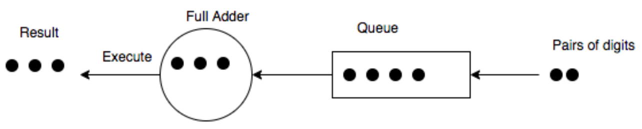
From the overview of the system, it takes several pairs of numbers as input and wait in the queue, FCFS according to the timestamps, and then computes the sum of each digits via a series of full adders, stores the result and outputs.



From a more specific view of the calculation, as figure shows, we take an example of addition of two 4-bit numbers to elaborate the model.



Firstly, it separates the binary number bit by bit and records each digit pairs according to their positions. Then it takes 0, A_0, B_0 as the input for the first full adder and calculate the S_0 and C_0. C_0 is then working as the message for the next full adder to work and S_0 is stored as part of result. The next digits follow the same procedure and finally the result will be $C_{out}S_3S_2S_1S_0$. After the computation in the first adder, if there are another pair of digits coming for adding the next pairs of number, the first adder can deal with the next pair according to their sequence. Thus, the model for each full adder is:



Entities
a) Resource: The whole simulator can be treated as the combination of multiple full adders, while each full adder can be a resource which consists of several AND/OR/NOR/XOR gates.

b) Consumer: Digits of the specific pairs of numbers Ai, Bi. And the output from the last adder Ci.

c) Queue: A queue of numbers waiting for processing. For each full adder, there are also three queues waiting to enter: queue of A_i, B_i and C_i.

Attributes
a) Data, including both numbers and their digits.

b) Timestamp of each number/digit.

c) Precision of the input numbers in binary form.

Events
a) Receive a message from the last Full Adder.

b) Process digits A_i, B_i, C_i using AND/OR/NOR/XOR gates, which has the precondition a) and end event c).

c) Send a message to the next Full Adder

Activity
Executing a pair of digits, which begins with receiving the message from the last Full Adder and ends with getting the summary and sending a message to the next Full Adder.

3 IMPLEMENTATION AND PSEUDOCODE

GNode
A class named GNode was built to represent the logic gates of the circuit in our simulation, which includes NOT gate, OR gate and AND gate.

In NOT gate, the output will be 0 if the input=1, otherwise will be 1.

```
if(this->type == "NOT"){
        ans = input[0].Load();
        if(ans == 1) output.exchange(0);
        else output.exchange(1);
}
```

In AND gate, the output will be 1 only if all the three inputs=1, otherwise 0.

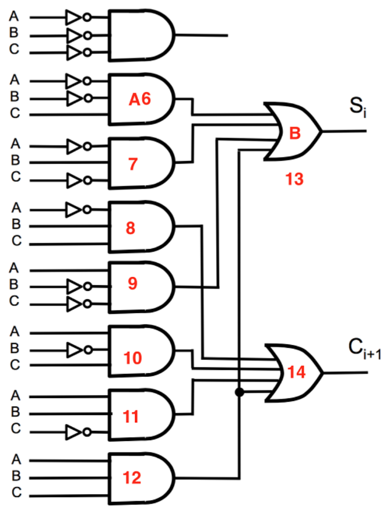
```
if(this->type == "AND"){
        ans = 0;
        for(auto &&itr : input){ans += itr.Load();}
        if(ans == 3) output.exchange(1);
        else output.exchange(0);
}
```

In OR gate, the output will be 1 if any of the input=1, otherwise 0.

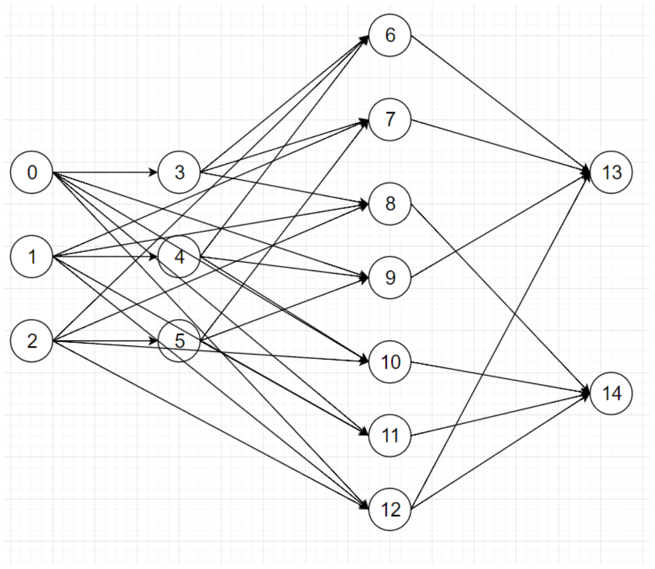
```
if(this->type == "OR"){
        ans = 0;
        for(auto &&itr : input){ans += itr.Load();}
        if(ans >= 1) output.exchange(1);
        else output.exchange(0);
}
```

In addition, every electric signal in the circuit is represented as an event in the simulation, and signal propagation is simulated as message passing between neighboring nodes in graph.

Graph
A graph consists of GNode was built to connect all the logic gates in the circuit. 15 nodes are required to build a full adder which could calculate 1 bit. Nodes 0 2 are the original input of the first addend a, second addend b and overflowed bit c. Nodes 3 5 represent the NOT gate of each input. Nodes 6 12 are the AND gates while the last two Nodes 13 and 14 represent OR gates. Here is the circuit of 1-bit calculator:

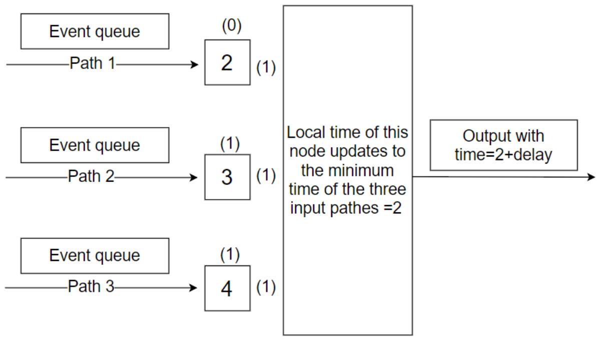


Then we transform the circuit into a graph which contains 15 nodes through connecting their inputs and outputs:

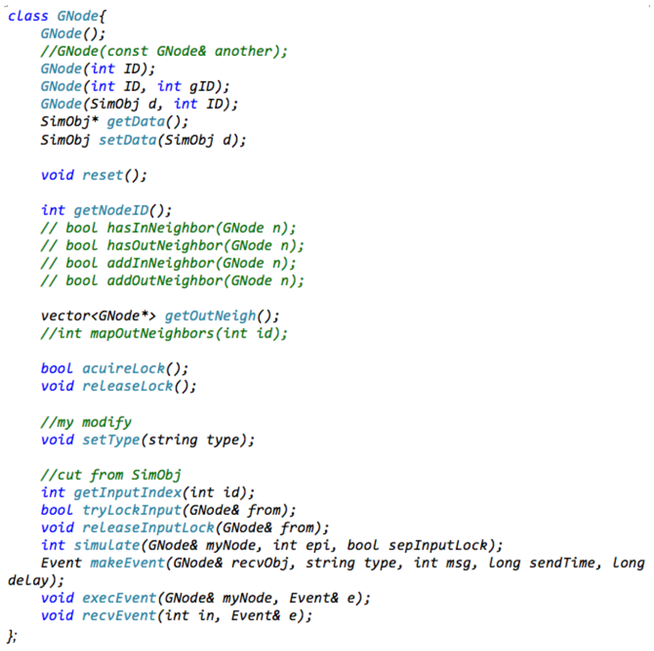


Distributed algorithm and time control

During our simulation, each initial event generated by the input nodes will cause new messages and additional events to be generated, processed, and propagated through the edges in the graph to the nodes. The simulation ends after all the events, including initial events and events generated during the simulation, are processed. Every event has a timestamp associated with it. The timestamp is the time when an event should be processed by the node which is holding that event. And the events must be processed in their timestamp order. A delay=1 is applied between the time the node processes an event and the time the nodefis neighbors receive the processing result.



In order to apply parallel computing later in our simulation, we use a distributed algorithm for discrete event simulation which removes the requirement of global event controlling and allows every individual node to process its local events independently and simultaneously. Each node maintains its own local clock and event queue. A nodefis local clock is defined as the minimum of the timestamps of the last event the node has received on all its input ports. All the events in the nodefis event queue with timestamp smaller than or equal to the nodefis local clock are called ready events. Ready events are safe to be removed from the queue and processed in their timestamp order to generate new events for the neighboring nodes in the fanout. Since ready events are processed in the timestamp order, new events will also be generated and stamped in the order of time, which means, for any input port of any node in the graph, events come in the order of time. Thus, for any node, any new event coming from any input port of the node will have a timestamp greater than or equal to the timestamps of all current ready events of the node.

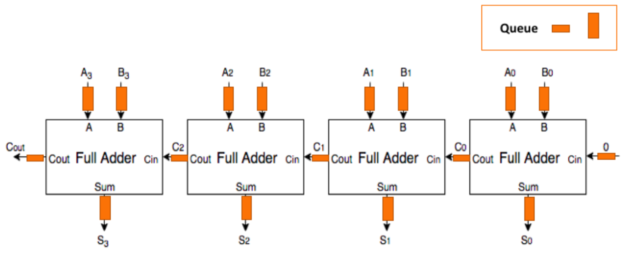


```
class GNode{
    GNode();
    //GNode(const GNode& another);
    GNode(int ID);
    GNode(int ID, int gID);
    GNode(SimObj d, int ID);
    SimObj* getData();
    SimObj setData(SimObj d);

    void reset();

    int getNodeID();
    // bool hasInNeighbor(GNode n);
    // bool hasOutNeighbor(GNode n);
    // bool addInNeighbor(GNode n);
    // bool addOutNeighbor(GNode n);

    vector<GNode*> getOutNeigh();
    //int mapOutNeighbors(int id);

    bool acuireLock();
    void releaseLock();

    //my modify
    void setType(string type);

    //cut from SimObj
    int getInputIndex(int id);
    bool tryLockInput(GNode& from);
    void releaseInputLock(GNode& from);
    int simulate(GNode& myNode, int epi, bool sepInputLock);
    Event makeEvent(GNode& recvObj, string type, int msg, long sendTime, long
delay);
    void execEvent(GNode& myNode, Event& e);
    void recvEvent(int in, Event& e);
};
```

After an input node sends out all its initial events, it sends a NULL message with timestamp infinity to inform all its neighboring nodes in the fanout that there will be no new events anymore. Similarly, after a normal node has received NULL messages from all its input ports, it sends out a NULL message via its outgoing edges. After all the output nodes receive a NULL message, the simulation terminates.

Architecture of Multi-bit Adder

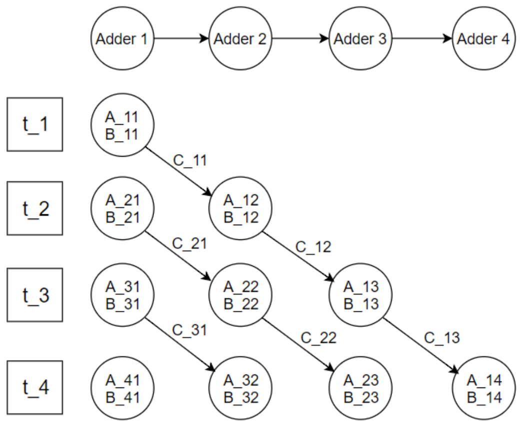
A 1-bit adder takes 3 inputs and gives out 2 outputs for a single run. In a multi-bit adder, we connect k 1-bit adder together to be k-bit adder. We claim 4k+1 queues to store inputs and outputs between 1-bit adders. So that, with synchronization algorithm applied, the program allows several threads to work at the same time without running into error. Each time a thread picks up a 1-bit adder to work with.



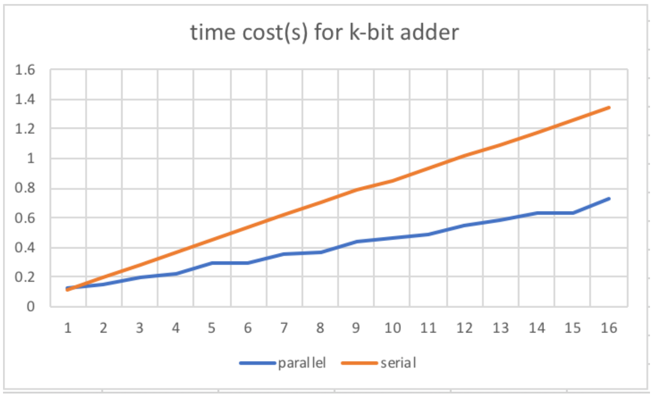
In code, Adder class is to build multi-bit adders. It defines the relationship between input/output queues and 1-bit adders. It also contains a read function that translate decimal number into binary and send them to input queues. And a final output function is defined to sum up binary to decimal result for each updated timestamp.

Parallel Computing

Since our program is thread-save, which means we can have different threads work on different 1-bit adder at the same time. Therefore, we choose openMP to run our program in parallel. The parallel part is mainly among running 1-bit adders. Dynamic schedule is applied. As a result, the running time reduced to about a half of serial version.



In fact, the program is run with 4 threads, and expect to reduce cost time to 25 percent. The reason why it only reduces to 50 percent maybe that when two threads work at two connected 1-bit adder, if the former 1-bit adder is writing to its output queue that saves its carry-over, the later 1-bit adder must wait to read it input. Because the output queue of the former is the input queue of the later. Better schedule strategies may be applied in future work.



This figure shows program running time for 500 time-steps with k-bit adder.

4 EXPERIMENTS AND ANALYSIS OF RESULTS

Validation and Verification

Experiments of results is important for our simulation to prove the correctness of it. In this part, two methods were used for the validation and verification which were testing of final results and of AND gates. Take an example of:

$$00111111(31) + 00100101(37) = 01000100(68)$$

In our simulation, the final result of each bit (each graph) should be node 13. Because of the time delay of each node, the result of each graph would present sequentially. Through observing our event list from each node, the time of finishing all the calculation should be 18 since the output from the node 13 in the last graph became 1 at time=18. Therefore, the output of node 13 and node 14 of each graph had been integrated:

Graph	Node[13]	Node[14]
0	0	1
1	0	1
2	1	1
3	0	1
4	0	1
5	0	1
6	1	0
7	0	0

The second column is the output of each bit while the third column is the overflowed number. Through checking the final result, the final output of our simulation is 01000100 which is exactly the correct answer.

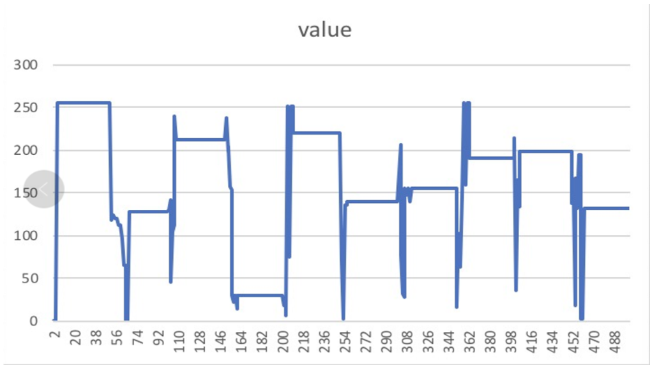
However, a precise result dose not mean a good simulation. Since we focus on the working process of the logic circuit not the result. Another things we have to check is that if the simulation run as we expected. To be sure, the No.6 No.12 gates in each graph are AND gates which is the most important of the logic circuit since they decide the results directly. Therefore, the output of AND gates in each graph should be tested to see if the distributed system work smoothly.

Graph	Input a	Input b	Input c	"And" gate output=1
0	1	1	0	11
1	1	0	1	10
2	1	1	1	12
3	1	0	1	10
4	1	0	1	10
5	0	1	1	8
6	0	0	1	6
7	0	0	0	\

This figure shows the AND gates whose output is 1 in each graph. Compared with our node network in conceptual model, all the graph meet the requirement. For example, the No.8 gate in graph 5 has the output 1 with the input a=0,b=1,c=1, which means the No.8 node would receive the message that value=1 from No.3, No.1 and No.2. No.3 gate is the NOT gate connected to No.0, which represents the original input of a, while No.1 and No.2 are the original input of b and c.

Results Analysis

Based on the results, a phenomenon found by us was that when we put a series of numbers as input, the results would become unstable.



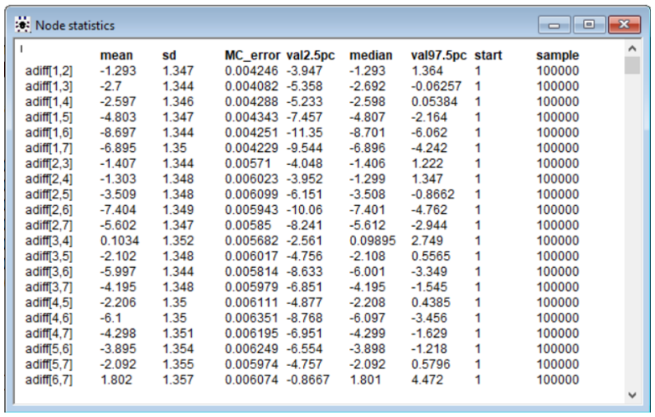
The figure shows that each time a new pair of number is added in the simulation, there will be a great jump on the result. But finally it will converge to the correct answer. In general, the convergence time might be correlated to the total number of graphs we set for our simulation. However, since our addends were generated randomly, a strict way is needed to figure out if there exist correlation between the convergence time and the number of bits.

Bits	Convergence time									
2	2	4	1	3	2	1	4	3	1	1
3	5	2	5	4	4	3	1	3	3	5
4	5	4	5	2	11	2	3	5	7	5
5	5	2	3	4	6	8	3	3	9	5
6	8	5	11	10	7	3	3	3	8	12
7	16	9	19	20	8	8	9	5	5	10
8	10	9	8	9	5	8	8	11	10	13

An ANOVA approach based on Bayesian statistics was used to test if the difference is significant. Assume that T(i)= the convergence time of i bits simulator:

$$T(i) \sim N(\mu_i, \tau)$$
$$\mu_i = \mu_0 + \alpha_i$$
$$\alpha_i \sim N(0, 100^2)$$
$$D_{ij} = \alpha_i - \alpha_j$$

Our goal is to test D_{ij}, which is the difference between the convergence time between each pair of group, is significant or not. A Bayes analysis software based on Markov chain Monte Carlo named WinBUGs was used to test it:



```
            mean     sd      MC_error val2.5pc  median   val97.5pc start  sample
 adiff[1,2] -1.293   1.347   0.004246 -3.947    -1.293   1.364     1      100000
 adiff[1,3] -2.7     1.344   0.004082 -5.358    -2.692   -0.06257  1      100000
 adiff[1,4] -2.597   1.346   0.004288 -5.233    -2.598   0.05384   1      100000
 adiff[1,5] -4.803   1.347   0.004343 -7.457    -4.807   -2.164    1      100000
 adiff[1,6] -8.697   1.344   0.004251 -11.35    -8.701   -6.062    1      100000
 adiff[1,7] -6.895   1.35    0.004229 -9.544    -6.896   -4.242    1      100000
 adiff[2,3] -1.407   1.344   0.00571  -4.048    -1.406   1.222     1      100000
 adiff[2,4] -1.303   1.348   0.006023 -3.952    -1.299   1.347     1      100000
 adiff[2,5] -3.509   1.348   0.006099 -6.151    -3.508   -0.8662   1      100000
 adiff[2,6] -7.404   1.349   0.005943 -10.06    -7.401   -4.762    1      100000
 adiff[2,7] -5.602   1.347   0.00585  -8.241    -5.612   -2.944    1      100000
 adiff[3,4] 0.1034   1.352   0.005682 -2.561    0.09895  2.749     1      100000
 adiff[3,5] -2.102   1.348   0.006017 -4.756    -2.108   0.5565    1      100000
 adiff[3,6] -5.997   1.344   0.005814 -8.633    -6.001   -3.349    1      100000
 adiff[3,7] -4.195   1.348   0.005979 -6.851    -4.195   -1.545    1      100000
 adiff[4,5] -2.206   1.35    0.006111 -4.877    -2.208   0.4385    1      100000
 adiff[4,6] -6.1     1.35    0.006351 -8.768    -6.097   -3.456    1      100000
 adiff[4,7] -4.298   1.351   0.006195 -6.951    -4.299   -1.629    1      100000
 adiff[5,6] -3.895   1.354   0.006249 -6.554    -3.898   -1.218    1      100000
 adiff[5,7] -2.092   1.355   0.005974 -4.757    -2.092   0.5796    1      100000
 adiff[6,7] 1.802    1.357   0.006074 -0.8667   1.801    4.472     1      100000
```

According to the result, the credible set of most of the difference do not include 0, which means they are significant. Others shows non-significant probably because of the size of our sample is not enough.

REFERENCES

[1] Wei-chen Xiao, Jisheng Zhao, Vivek Sarkar. *Parallelizing a Discrete Event Simulation Application Using the Habanero-Java Multicore Library.* Department of Computer Science, Rice University, 2010.