RC 18558 (#81158) 12/2/92
Computer Science 9 pages

Research Report

Parallelization of Cellular Automata Code and its Performance on Two Types of RS/6000 Clusters

Joefon Jann

IBM Research Division
T. J. Watson Research Center
Yorktown Heights, NY 10598

LIMITED DISTRIBUTION NOTICE

This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. It has been issued as a Research Report for early dissemination of its contents and will be distributed outside of IBM up to one year after the date indicated at the top of this page. In view of the transfer of copyright to the outside publisher, its distribution outside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g., payment of royalties).

IBM Research Division
Almaden • T.J. Watson • Tokyo • Zurich

Parallelization of Cellular Automata Code and its Performance on Two Types of RS/6000 Clusters

Joefon Jann

IBM Research
T. J. Watson Research Center
Yorktown Heights, NY 10598

Abstract

We describe a method to effectively parallelize a Cellular Automata simulation for execution on a distributed-memory parallel architecture. We also present experimental timing measurements and analysis of this algorithm, and compare the performance of this parallel program implemented in Express running on an RS/6000 cluster connected via Ethernet, with its performance when implemented in EUI running on an RS/6000 cluster connected via the VULCAN switch.

Introduction

Since the IBM RISC System/6000 processors provided a cost-effective platform for scientific computing, one has seen the emergence of clusters of them used for loosely coupled parallel computing. In these clusters, the processors may be connected by Ethernet, Token Ring, specialized adaptors, or specialized interconnect switches. In a cluster with Ethernet or Token Ring connections, the inter-processor communication speed is typically much slower than a processor's computational speed. This implies that one has to devise algorithms carefully to effectively exploit parallelism in these clusters. In this study, we describe a method to effectively parallelize a two dimensional Cellular Automata simulation. In this algorithm, as will be demonstrated by detailed analysis later, the dominant communication cost is constant with respect to the number of processors, and involves the transfer of a relatively small amount of information amongst processors. The same technique can also be applied to an n-dimensional Cellular Automata simulation.

We also present experimental timing measurements and analysis of this algorithm, and compare the performance of this parallel program implemented in Express ([1,2]) running on an RS/6000 cluster connected via Ethernet, with its performance when implemented in EUI [3] running on an RS/6000 cluster connected via the VULCAN switch [4].

Serial Algorithm

Cellular Automata is the study of the influence of neighboring cells on a cell over a number of time steps. From a computational point of view, the algorithm for the serial code can be summarized as follows:

1. The computational domain consists of a 2-D square grid of cells with cyclic boundary condition i.e. the domain can be viewed as a torus.

2. The value at each grid point (cell) in each time step is computed as a function of the values of its neigbors in the previous time step.

3. Averages are computed periodically at every user-specified number of time steps (NINT).

Parallel Algorithm

The parallel algorithm developed is as follows:

1. The processors are logically ordered into a ring. Since the computational domain is a torus, and the ring is a natural divisor of the torus, this ring ordering enables an efficent domain decomposition, resulting in the need for only a small amount of changes to the serial code.

2. The grid of cells is divided into strips. An overlapping border of cells is added to the edges of adjacent strips. Each resulting strip is assigned to a processor as it computational domain.

3. The value at each grid point in each time step is computed as a function of the values of its neigbors in the previous time step.

4. Only the border values of the strips are interchanged amongst neighboring strips (CPUs) at the end of each time step. As a side benefit, the global boundary computations are automatically taken care of by the ring design.

5. Sums of heights and squared heights of each strip are computed at every NINT time steps by each processor, and then combined using the Express function KXCOMB or the EUI function mp_combine to compute the global average height, the global height variance, and other global measures.

6. In the simulation, each processor uses a linear-shift-register based random number generator [5]. To ensure independence of the random numbers generated, each generator is initialized with a different seed.

Experiment details:

We examined the elapsed time used for computing the Cellular Automata effects on a square grid of cells of various sizes using 1 to 8 processors.

The variables used in the experiments are:

```
NSIZE  = 20, 200, 400 (size of the square grid, e.g. 20 means a 20x20 matrix)
NCPU   = 1 to 8       (number of processors used in parallel exexcution)
NSTEPS = 1000         (total number of time steps)
NINT   = 100          (time step interval at which averages are printed out)
```

Definitions:
We define

$$\text{Speedup} \equiv S \equiv \frac{T_{serial}}{T_{parallel}}$$

where

T_{serial}

is the elapsed Time for the serial program
and

$T_{parallel}$

is the elapsed Time for the parallel program.

We also define

$$\text{Efficency} \equiv E \equiv \frac{S}{P}$$

where P is the number of Processors used.

Understanding that elapsed time is dependent on factors like:

- Load on the system by other users (interactive or batch)
- Load on the system by system services e.g. cron jobs

we have tried to run the programs at times when there is little or no other users and batch jobs on the system.

Experimental Results using Express with Ethernet

S/W: Express 3.2.4, xlf 2.2.1.3, AIX 3.2.1
H/W: 7 of RS/6000 model 560s (50 MHz clock) with 256MB of memory
1 of RS/6000 model 540 (30 MHz clock) with 256MB of memory
Comm: Ethernet (10 Megabits/sec)

Elapsed time in seconds

P -> / NSIZE v	1	2	3	4	5	6	7	8
400	2333.7	1179.6	799.8	612.9	492.9	426.7	367.1	556.6
200	584.3	324.4	211.7	158.4	135.3	119.7	107.8	156.9
20	6.3	11.6	12.1	10.4	12.5	15.2	14.3	11.6

Speedup S

P -> / NSIZE v	1	2	3	4	5	6	7	8
400	1.00	1.98	2.92	3.81	4.73	5.47	6.36	4.19
200	1.00	1.80	2.76	3.69	4.32	4.88	5.42	3.72
20	1.00	.54	.52	.61	.50	.41	.44	0.54

Efficiency E in %

P -> / NSIZE v	1	2	3	4	5	6	7	8
400	100.0%	99.0 %	97.3 %	95.3 %	94.6 %	91.2 %	90.9 %	52.4 %
200	100.0%	90.0 %	92.0 %	92.2 %	86.4 %	81.3 %	77.4 %	46.5 %
20	100.0%	27.0 %	17.3 %	15.1 %	10.0 %	6.9 %	6.3 %	6.8 %

Experimental Results using EUI with VULCAN Switch

S/W: EUI, xlf 2.2.1.3, AIX 3.2.1
H/W: 8 of RS/6000 model 550 (41.6 MHz) with 32MB of memory
Comm: VULCAN switch

Elapsed time in seconds

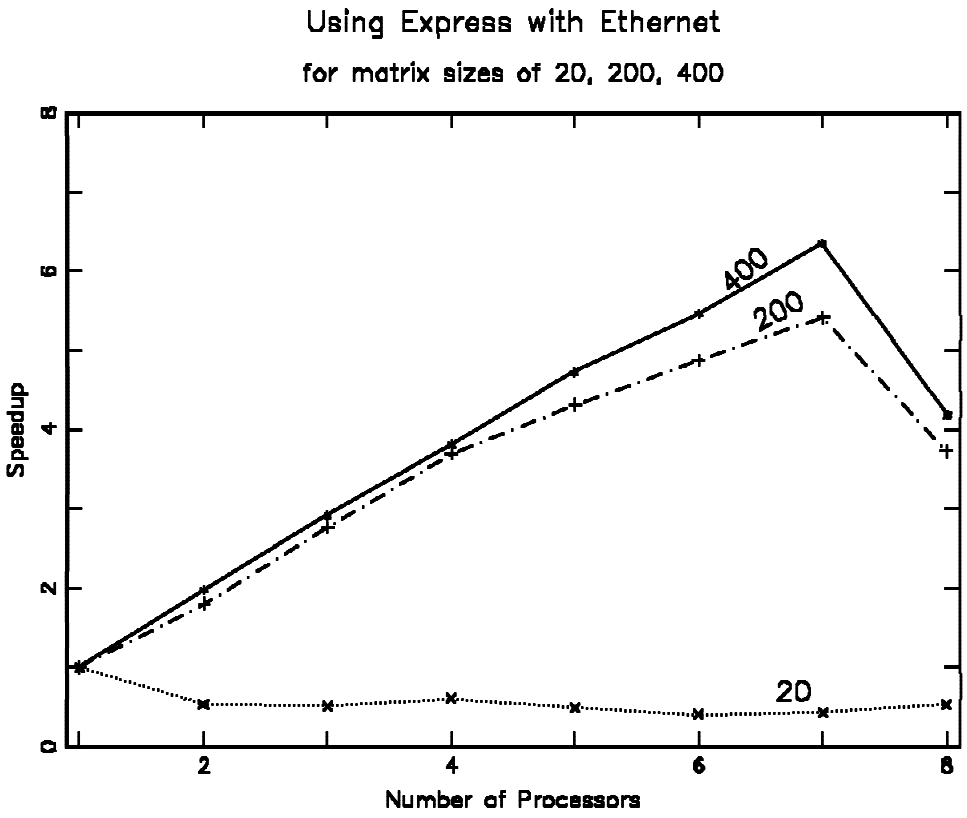
P -> / NSIZE v	1	2	3	4	5	6	7	8
400	1634.81	820.25	549.10	411.90	328.90	275.23	237.81	207.04
200	409.95	200.73	135.03	101.39	81.60	69.33	59.59	53.70
20	4.75	2.88	2.16	1.76	1.56	1.62	1.43	1.45

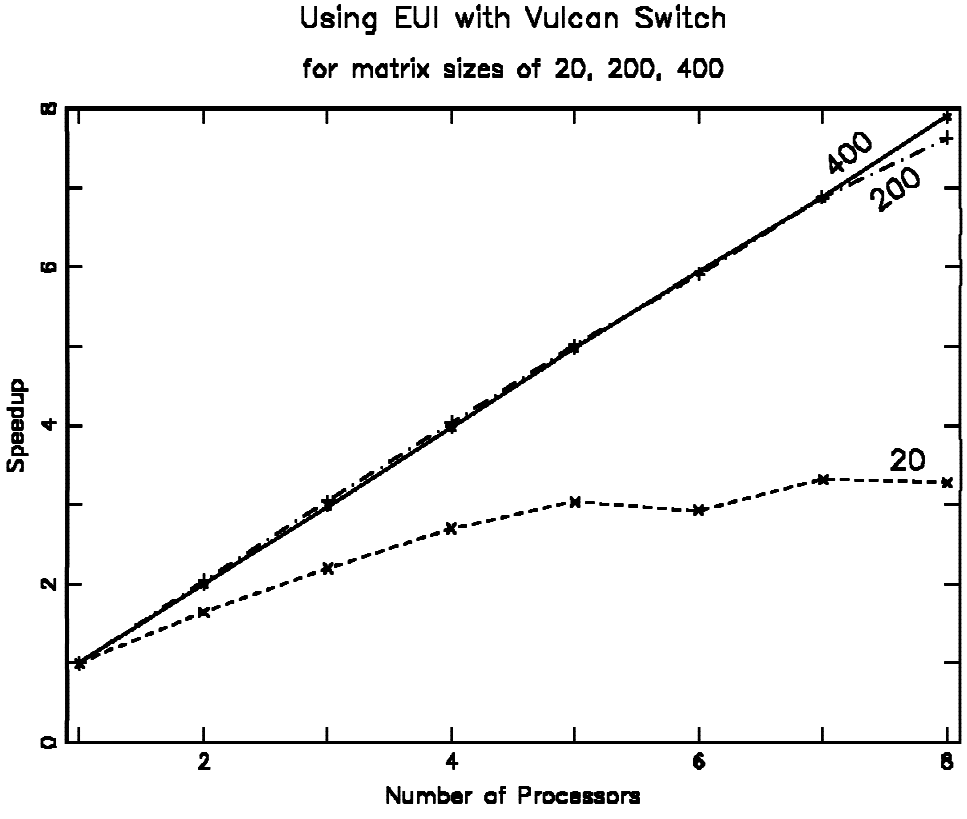
Speedup S

P -> / NSIZE v	1	2	3	4	5	6	7	8
400	1.00	1.99	2.97	3.97	4.97	5.94	6.87	7.90
200	1.00	2.04	3.04	4.04	5.02	5.91	6.88	7.63
20	1.00	1.65	2.20	2.70	3.04	2.93	3.32	3.28

Efficiency E in %

P -> / NSIZE v	1	2	3	4	5	6	7	8
400	100.0%	99.5%	99.0%	99.3%	99.4%	99.0%	98.1%	98.8%
200	100.0%	102.0%	101.3%	101.0%	100.4%	98.5%	98.3%	95.4%
20	100.0%	82.5%	73.3%	67.5%	60.8%	48.8%	47.4%	41.0%


Using Express with Ethernet
for matrix sizes of 20, 200, 400
Speedup
Number of Processors
400
200
20
0
2
4
6
8


Using EUI with Vulcan Switch
for matrix sizes of 20, 200, 400
Speedup
Number of Processors
400
200
20
0
2
4
6
8

Timing Analysis

$T_{parallel}$, the elapsed time for the parallel program, can be brok up into the following major components:

1. T_0 = Initialization time for reading input from file, for Express or EUI initialization, and for grid initialization.
2. Time to compute neighboring influence on each cell (T_{cpt1}) in a strip at every time step.
3. Communications time of interchanging border values of neighboring strips at every time step.
4. Time to compute average and variance (T_{cpt2}) of each strip at every NINT time steps.
5. Time to compute global average and variance using collective communication at every NINT time steps.
6. T_e = Time to compute global measures e. g. average, variance, etc., using collective communication routines at the end.

In other words,

$$
\begin{aligned}
T_{parallel} \cdot \; & T_0 \\
& + T_{cpt1} \times \frac{NSIZE^2}{P} \times NSTEPS \\
& + (T_L + \frac{8(NSIZE+2)}{B}) \times 2 \times NSTEPS \\
& + T_{cpt2} \times \frac{NSIZE^2}{P} \times \left\lfloor \frac{NSTEPS}{NINT} \right\rfloor \\
& + (T_L + \frac{16}{B}) \times 2 \times \lceil \log_2 P \rceil \times \left\lfloor \frac{NSTEPS}{NINT} \right\rfloor \\
& + T_e
\end{aligned}
$$

where

$T_L + \frac{k}{B}$ is the time to send k bytes of message from one processor to another,

T_L is Latency

B is Bandwidth

$2 \times \log_2 P$ is the number of steps for traversing up and down a binary tree of P nodes, used in the collective communication subroutines.

In practice, $\frac{NSTEPS}{NINT}$ is a small integer, and T_{cpt1} is much larger than T_{cpt2}. Hence, to a good approximation,

$$T_{parallel} \cdot \quad T_{cpt1} \times \frac{NSIZE^2}{P} \times NSTEPS$$

$$+ \ (T_L + \frac{8(NSIZE+2)}{B}) \times 2 \times NSTEPS$$

where the first term is computation time and the second term is communication time. Thus, as indicated in the introduction, the dominant communication cost is independent of the number of processors.

The ratio of computation time to communication time is thus proportional to

$$\frac{NSIZE^2}{2P(T_L + \frac{8(NSIZE+2)}{B})}$$

Hence for networks with large T_L and/or small B, one expects poor speedup for small NSIZE. This can be observed from the Ethernet experimental data.

When T_L is small and B is large, as in the case of the VULCAN switch, $T_{parallel}$ is of order

$O(\frac{NSIZE^2}{P})$ when P is small. This can also be observed from the VULCAN experimental data.

Conclusions

For a given set of hardware and supporting software, there exists, for each problem and problem size, an optimal number of processors for which efficiency is maximal. See the peaks in figure 1.

When designing domain decomposition of a problem for parallelization, it is convenient to choose a topology of the processors that is a divisor of the topology of the problem domain.

Collective communications is a very efficient way of computing global quantities, since the time is only proportional to $\log^2 P$.

The elapsed time of parallel programs in a loosely coupled cluster is greatly dependent on the speed of the inter-processor links. We have shown with this example that even with faster

processors, the elapsed time cannot be much reduced unless the communication speed between processors is greatly increased, for example by a fast switch.

References

[1] Express FORTRAN Version 3.0 User's Guide (ParaSoft Corporation)

[2] Express FORTRAN Version 3.0 Reference Guide (ParaSoft Corporation)

[3] EUI is an Applications Programming Interface for distributed memory computing developed by IBM.

[4] VULCAN switch is a high performance switch developed by the Computer Science department of IBM T. J. Watson Research Center.

[5] ESSL Engineering & Scientific Subroutine Library Version 2 Guide & Reference Volume 3 (SC23-0526-00) and the references therein.

[6] Highly Parallel Computing by George S. Almasi and Allan Gottlieb (Benjamin/Cummings Publishing Co. Inc. 1989, ISBN 0-8053-0177-1).

[7] Theory and Applications of Cellular Automata by Stephan Wolfram (World Scientific Publishing Co. Inc. 1986, ISBN 9971-50-124-4).

Express is a product of ParaSoft Corporation. It provides Fortran and C callable libraries and programming environments to facilitate the use of MIMD Risc clusters for parallel computing. Parasoft Corp. phone #: (818)792-9941.

Acknowledgments

The author would like to acknowledge Geoffrey Grinstein who developed the serial version of the code, and allowed me the opportunity to parallelize it. Much thanks are due to Peter Hochschild for providing the EUI software and the VULCAN switch, and to Bucky Pope for his encouragement.

Copies may be requested from:

IBM Thomas J. Watson Research Center
Distribution Services F-11 Stormytown
Post Office Box 218
Yorktown Heights, New York 10598