

BlueGene Test Report

James Frye

November 7, 2004

1 Summary

This report expands on my preliminary report of October 25th, which summarized work done at IBM Watson Lab, October 18-23 2004. The data collected has been further analyzed, compared with equivalent runs on the Cortex cluster, and differences noted.

There were three goals for this visit:

- To port the NCS program to the BlueGene environment.
- To obtain performance measurements for NCS on BlueGene relative to the UNR Cortex cluster.
- If time allowed, to compile and test the NEURON software.

1.1 Positives

The first two goals were accomplished. The NCS program was ported to BlueGene with minimal effort. It appears that the same should be true of most programs designed for non-interactive use.

As to performance, when running NCS on equal numbers of processors, the BlueGene delivers approximately the same performance as the Cortex Beowulf cluster, which uses 2.2 GHz P4 processors. Test runs up to the number of processors available (512/1024) demonstrate that NCS shows roughly linear speedup with increased number of processors. Thus the BlueGene system should deliver performance roughly proportional to the number of processors when running NCS, or parallel programs of similar design.

1.2 Negatives

Two negatives were noted. First, the BlueGene system intentionally does not have much support for user interaction, or for system services such as multi-threading. Software which has an interactive or GUI component, such as NEURON, or which uses a threaded architecture, will take considerably more effort to port. Second, the current processor sharing scheme will require coordination and cooperation between users.

In my opinion the positives far outweigh the negatives.

2 BlueGene Hardware & Software

The basic CPU unit of the BlueGene is a variant of the Power PC architecture, running at 700 MHz. Each processor card contains two of these CPUs. 16 cards are placed on a board. 16 of these boards make up a midplane, and two midplanes make a rack. Each compute node is connected to its 6 nearest neighbors via a high-speed link, to form a 3D torus network. Boards also contain additional processors which are dedicated to inter-node communication & I/O. Each of these can communicate with the outside world via a Gigabit Ethernet link.

The BlueGene processors run a simple single-tasking OS. This eliminates the overhead associated with multi-processing systems, so that processor cycles are devoted exclusively to program execution. The dedicated I/O nodes provide system services such as file system access and socket connections to the outside world. This is handled transparently: the program makes standard Unix system calls, and receives expected results. However, some system services such as virtual memory & multithreading, either aren't available, or exist only in limited form.

All development and related work - compiling & linking programs, starting jobs, reserving blocks of processors - is done on one or more frontend systems, which run a standard Linux OS.

The machine at Watson has only 512 cards (1024 processors). It is actively used for development and testing of both software and hardware, and often was unavailable due to such work. When available, it is shared between a number of users. This made it impossible to run a full set of tests, especially on large numbers of processors.

2.1 Control Program

Users interact with the BlueGene through a control program, which reserves machine resources to individual users. With the current BlueGene control program (MMCS), CPU sharing is somewhat limited. There's no overall batch queue or job scheduling. (Though such is planned at some future date.) Instead, each user requests a block of CPUs (with a group of 32 cards being the smallest allocatable unit), and has exclusive control of that block until it is released. The user may then use as many of the CPUs in the block as desired to run a program. However, any unused processors remain idle, and if all are unused, the block is still unusable to others until released by the first user.

The block may be used in one of two modes: virtual or coprocessor. In coprocessor mode, one CPU per card is devoted to computation, the other handles all the communication tasks. In virtual mode, both do computation and handle their own communication.

There are benefits as well as disadvantages to this blocking scheme. It requires coordination between users, especially when jobs need to use all of the machine. On the other hand, it allows the machine to effectively be split into many smaller clusters which can be used independently, while the systems and support costs are only those of a single cluster.

2.2 Software Development

Software development tasks are done as usual: the user simply changes compiler names and options to those of the BlueGene cross-compiler, and compiles on a front-end which shares a filesystem with the BlueGene. A script similar to `mpirun` is used to start

programs, which typically take less than a minute to load on the whole system. Files and standard output are used just as on a standard cluster.

3 Software Conversion

NCS was easily converted to run on BlueGene: under three hours to an initial working version, and about two days to one with all tracing and measurement enabled.

Code changes were minor. The routines in `Port.cpp` were modified to return NULL values, since some socket communication functions aren't fully supported under BlueGene. (This doesn't affect primary functionality of NCS, and will be easy to re-code if & when needed.) The `NodeInfo` queries for the bogomips number (a measure of processor capability used in distribution) were replaced by fixed values, since all BlueGene CPUs are identical. Finally, the memory-measurement code in code `Memory.cpp` was replaced with a BlueGene version.

The NEURON program proved far more difficult to convert: indeed, I did not manage to get a working version in the time available. The problems seemed to be not so much with the NEURON code itself, but with the GNU `configure` script that is used to configure NEURON to different operating systems. BlueGene code must be cross-compiled, but the script is set up not to allow this. Likewise, the script assumes that certain parts of an interactive NEURON program (`curses`, `termcap`, etc.) must always be compiled in, but they do not exist on the BlueGene system. I don't believe these problems are insurmountable: one of the NEURON developers could probably fix them in short order.

The differences between porting NCS and NEURON appear to stem from the fact that NCS was designed run non-interactively, while NEURON has an interactive mode and GUI. I believe that this differences will hold true for most other software: if a program was designed to be used interactively, it will be difficult to port to the BlueGene. If not, porting should be almost trivial.

4 Testing

4.1 Model Used

In developing models for this testing, the primary consideration was the fact that spike processing consumes by far the largest fraction of compute time. Since the goal is both to obtain an idea of performance on realistic models, and to compare that performance across a wide range of processor counts, the model needed a spike rate (measured in spikes per cell per simulated second) that was both in a biologic range and consistent across model size and processor count. Thus it was necessary to sacrifice a certain degree of anatomical realism in order to make meaningful comparisons possible.

The models tested use as their basic element a simple column, which was taken from one of the models under study in the Reno lab. This column is composed of a single layer, made up of four cell types. The cells contain Km, Ka, and Kahp channels, and are connected semi-randomly to produce about 250K internal synapses per column.

Each model has a single column that receives external stimulus input. Other columns are arranged in an NxN grid. The input column connects to the $i = 0$ columns. Each column in the grid connects to its neighbors in the three boxes around it, with the connection probability attenuating by 0.8 for each box.

This gives a model that, after about 0.25 sec startup, settles into a fairly stable periodic oscillation in average spike rates, with about 60 spikes per cell per second. It seems to be quite scalable, with models of all sizes tested showing the same average behavior.

Models with $N = 5, 10, 20, 30, 40,$ and 50 were tested. Cell and synapse counts are shown in Table 1.

N	Columns	Cells (Thousands)	Synapses (Millions)
5	26	26	7.0
10	101	101	27.3
20	401	401	108.4
30	901	901	243.6
40	1601	1601	432.9
50	2501	2501	676.3

Table 1: Model Sizes.

4.2 Performance

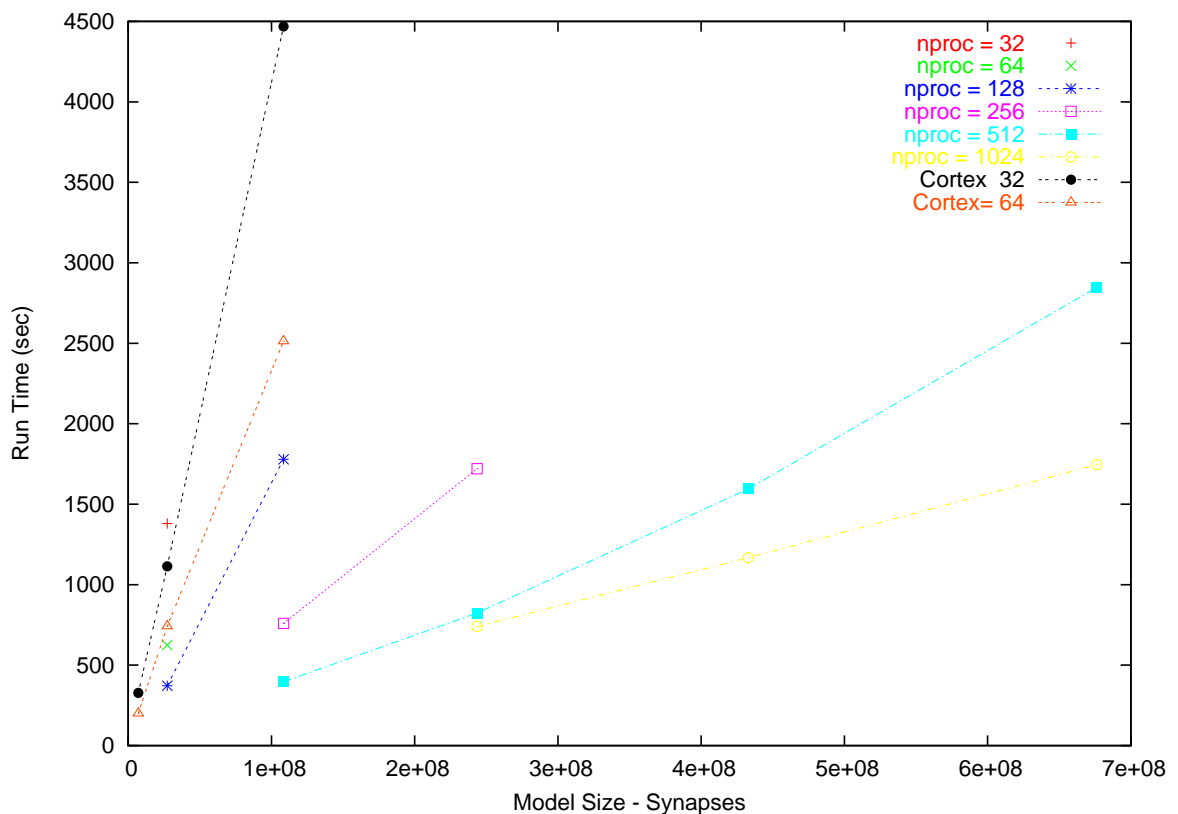


Figure 1: Run Times vs Model Size

Figure 1 shows total run times for various combinations of processor count and model size. Performance shows a near-linear relationship to changes in both model size

(synapse count) and number of CPUs used. (See notes below re the 1024-processor runs, however.)

Since spike processing consumes by far the largest part (typically 70-90%) of processing time, computing the number of spikes processed per second per CPU gives a better relative performance measure. (Only simulation time is used, not the initialization time.) This spike rate measure is shown in Table 2. It is fairly consistent across all models, excepting those done in 1024-processor virtual mode, where it drops significantly, probably due to the extra communication workload.

The average rate does fall off somewhat as the number of processors increases. This is probably a side-effect of the poor communication distribution. Since each CPU is directly connected only to its nearest neighbors, a larger processor count makes it more likely that a message will need to be forwarded multiple times. The fall-off is not large, so it appears that even current NCS code will show speed increases up to at least several thousand processors.

CPUs	N	Spikes *10 ⁹	Rate		Average (CPU)
			BlueGene	Cortex	
8	5	0.18	20294.3	29105.6	20294.3
16	5	0.16	21995.9	30949.9	21995.9
32	5	0.19	22458.9	18021.5	
32	10	0.88	20295.7	25750.7	
32	20	3.66	16166.7	27002.6	19640.4
64	5	0.18	19941.9	14606.8	
64	10	0.88	22545.1	20490.7	21243.5
128	10	0.88	19011.8	17450.5	
128	20	3.66	16382.7	20079.7	17697.2
256	20	3.66	19407.3		
256	30	8.23	19157.5		19282.4
512	20	3.66	19162.4		
512	30	8.28	20831.9		
512	40	14.75	19047.4		
512	50	23.39	16865.3		18976.7
1024	30	8.33	11893.9		
1024	40	14.55	13264.9		
1024	50	23.38	14396.5		13185.1

Table 2: Per-Processor Spike Rates

4.3 Memory Use

Both computation and memory use are strongly dependent on the synapse count and firing rate of a model. The models tested were tuned to have firing rates in a biologically-realistic range. The N=50 model, which has $6e8$ synapses, uses only about 170 MBytes of the 512 MBytes possible when run on 512 processors, and about 110 MBytes per processor on 1024 processors. This will allow for model sizes of $5.0 * 10^9$ synapses or more to be run on 512 processors, and correspondingly larger models on larger numbers of CPUs. For comparison, the largest model run to date on Cortex is approximately $1.1 * 10^9$ synapses.

4.4 Memory and Maximum Model Size

NCS was developed to run on the Cortex cluster, which has 4 GBytes of memory per each dual-CPU node. Large models (more than 10^9 synapses) approach the limits of that memory. The BlueGene has only 512 MBytes per each 2-CPU card, although it will have many more processors.

Therefore there is a question of how large a model can be run on any given set of processors. There is no exact answer, since memory use depends substantially on the spike rate of the model. If that rate can be approximated (say by limiting it to biologically realistic ranges), then an approximate answer is possible.

Most of the memory is used in two areas: Synapses, by which in NCS we mean an entire connection between a pre- and post-synaptic cell; and spike processing, including messages & PSG template. There's other memory used, of course, but either the amounts are insignificant, or it can be freed if things like e.g. dynamic report creation aren't needed.

For synapses, there are two components, a `SendTo` item on the pre-synaptic side, and a `Synapse` item on the post-synaptic. In general, the number of pre- and post- will be roughly equal on a node (and of course will be exactly equal for the whole model). Thus

$$M_{synapse} = N_{synapse} * size$$

where

- $M_{synapse}$ is the memory used by synapses;
- $N_{synapse}$ is the number of synapses on the node;
- $size$ is `sizeof (SendTo) + sizeof (Synapse)`, currently 56 bytes.

Test models typically have 0.5 to 1.5 million synapses/node, for about 25-75 MBytes.

The spike processing is the most problematic, since the spike rate isn't known in advance. If the model used is fairly stable, however, a reasonable approximation to the rate can be made. An incoming spike remains in an active state for the length of its PSC template, generally 200 timesteps, and requires some memory to maintain that state. So the memory used is approximately

$$M_{spike} = N_{synapse} * Rate * (0.0001 * L_{PSG}) * size$$

where

- M_{spike} is the memory used by spikes;
- $N_{synapse}$ is the number of synapses on the node;
- $Rate$ is the approximate firing rate, in spikes per cell per second;
- L_{PSG} is the length of the PSC template, in timesteps;
- $size$ is `sizeof (ActiveSyn)`, currently 20 bytes.

For an average firing rate of 100 spikes per cell per second, this gives about 40 Mbytes per million synapses. Our rule of thumb is that about half the available memory can be used for synapses. This means models with up to about 2.5 million synapses per node can be run in virtual CPU mode, or about 5.0 million in coprocessor mode.

4.5 Communications

NCS was developed to run on the Cortex Beowulf cluster. This has a switched Myrinet communication network, which means that the communication path between any two nodes has essentially the same cost. The BlueGene, by contrast, has many more processors, which are connected in a 3D torus. Therefore the cost of communication between two nodes is proportional to the distance between them, and it is necessary to take this cost into account when assessing performance.

The current distribution algorithm attempts to balance computation (or memory use) between nodes, but takes no account of communication cost. It sorts cell clusters by compute weight, and attempts to assign an equal weight to each processor. The test model has four clusters per column, of unequal compute weights. There is high connectivity within each column, less between columns.

The distribution algorithm sorts clusters by compute weight, so all of type A are assigned to processors, then all of type B, etc. This means that inter-CPU communication is actually about as high as it could be for the model type. That is, the model is (quite unintentionally) testing near worst-case communication performance

Table 3 shows time spent in communication. Typically half to two thirds of the time is being spent in communication. Of course in coprocessor mode this is spent on a separate CPU, but the effect can be seen in the higher percentages (and thus lower spike rates in Table 2) of the smaller 1024 processor runs.

CPUs	N	Run Time	Communication Time			Percentage
			Min	Med	Med	
8	5	1125.84	492.75	514.30	756.42	45.68
16	5	474.10	237.33	257.99	332.20	54.42
32	5	279.91	111.57	164.55	215.67	58.79
32	10	1379.77				
32	20	7143.51	4070.41	4116.85	5235.93	57.63
64	5	145.38	54.63	96.96	145.88	66.70
64	10	624.33				
128	10	372.13	197.60	251.78	305.56	67.66
128	20	1778.89	1112.87	1139.83	1399.00	64.08
256	20	759.59	481.31	516.43	582.94	67.99
256	30	1720.82	976.47	1006.24	1294.05	58.47
512	20	396.63	230.45	281.76	329.95	71.04
512	30	824.20	507.67	532.75	645.85	64.64
512	40	1594.64	948.58	982.88	1220.33	61.64
512	50	2847.74	1731.13	1797.99	2183.93	63.14
1024	30	740.38	551.38	597.53	636.07	80.71
1024	40	1167.08	811.15	851.01	947.79	72.92
1024	50	1745.64	1078.44	1136.21	1388.18	65.09

Table 3: Communication Time (Seconds)

It thus appears that BlueGene communications should be more than sufficient for normal NCS runs with the current code, while a distribution algorithm tuned to BlueGene should give a significant speedup. Although finding a best-case distribution is difficult, a routine which gives a moderately good one should be fairly simple to implement.

4.6 Direct Comparison with Cortex

On the combinations of model size and CPU count where a direct comparison is possible, the BlueGene's 700 MHz processors generally deliver roughly the same throughput as the 2.2 GHz processors used by Cortex. Table 2 shows the spike rates for Cortex as well as BlueGene. Up to 32 processors (1 CPU on each dual-CPU node) Cortex is substantially faster, although the ratio is less than half the 3:1 CPU speed ratio. At 64 processors and above ¹ (both CPUs on a P4 node, and the 1.0 GHz P3 nodes), rates are nearly equal.

5 Miscellaneous Notes

Several points should be noted:

- 1) The distribution algorithm used by NCS is designed to distribute equal amounts of computation across a Beowulf cluster which has relatively few processors with different speeds, but essentially the same communication cost between any two. The BlueGene, by contrast, has a larger number of processors, and communication cost varies between different ones. The distribution algorithm thus assigns a rather unbalanced load on the BlueGene. An algorithm designed to factor in communication load would be likely increase performance significantly.
- 2) All BlueGene runs were done in coprocessor mode, with the exception of the ones using 1024 processors (since the BlueGene machine at Watson has only 512 dual-CPU cards). As is seen in the figure, using virtual mode on smaller, less balanced models gives much less than a twofold speedup.
- 3) Single-CPU performance on the BlueGene is roughly the same as that of the 2.2 GHz P4 CPUs used by the faster half of Cortex, even though the clock speed is less than a third as fast. This corroborates something we have suspected for some time, that NCS is limited not by CPU speed, but by memory bandwidth. Some low-level profiling was done on the BlueGene using IBM tools, which suggests that there is still room for significant performance improvement (at least an order of magnitude) on the software side.
- 4) IBM provides a set of optimized math functions (exp, pow, etc). Using these instead of the standard math library gave about a 9% decrease in run time on initial tests. They were therefore used on all the test runs documented here.

¹Due to hardware problems, the 128-CPU cortex runs are actually using only 114 CPUs