Dual Processor Nodes for HPC Clusters: Memory Contention Issues

Douglas Eadline, Ph.D.

24th March 2003

deadline@basement-supercomputing.com

Abstract

Using dual SMP CPU nodes for Beowulf/HPC clusters is common method for reducing the cost of the cluster. Results are provided that indicate the efficiency of dual nodes is very dependent on the application and compiler.

Background

Using dual SMP CPU nodes in Beowulf/HPC clusters is a good way to add extra processors at a low cost. Often times however, the amount of work that is delivered by the extra CPU is unknown. As almost all clusters utilize a batch system where both CPUs may be used independently, the issue of efficiency becomes important. The overall efficiency of a parallel program is a complex problem to solve and is made more difficult when it does not have exclusive use of a compute node. In this report, we will take a first look at memory contention issues.

To measure the effect of two programs running on a dual node we can measure the time it takes for a single copy of a program and then measure the time it takes for two copies of the same program to run concurrently. If the program scales perfectly then the times should be identical. If the program does not scale perfectly the time should be between one and two times the single program time. A speedup measure can be calculated as follows:

$$speedup = \frac{(CTIME1+CTIME2)*STIME}{(CTIME*CTIME2)}$$

Where *CTIME1* is the time for running program copy1 concurrently with copy2 of the program, *CTIME2* is the time for running program copy2 concurrently with copy1 of the program, and *STIME* is the time to run one copy of the program.

From this equation we see that if *CTIME1*=*CTIME2*=*STIME* (perfect speed up) we get a speedup of 2. If (*CTIME1*+*CTIME2*)=2**STIME* (no speedup) we get an answer of 1 which indicates no speed-up.

Test Method and Hardware

In order to test the efficiency of dual nodes we can employ tests from the Beowulf Performance Suite (BPS) (http://www.plogic.com/bps). For the test hardware we used a node with an Intel SE7500WV2 motherboard. The test system includes dual 2.2Ghz Xeons, 1GB RAM, and a 2.4.20 SMP kernel. Both the Intel V6 (icc and ifc) compilers and the GNU V2.96 (gcc and g77) were used to compile the programs. Information about the test programs and the compile arguments can be found in the NAS suite in the BPS package.

A simple script has been written to run the tests and calculate the speedup for the NAS test suite (see below). The NAS test suite was compiled for one node using test size A. The script was run three times for and the results were averaged. The compiler can be changed by editing the "run_suite" line in the script.

We also ran the stream-wall program from the BPS package.

Results

Using the methodology above, we can run the stream-wall test (part of the BCS package) from the command prompt. To determine *STIME* we run:

```
$ time /usr/bps/stream-wall
real 0m1.616s
user 0m1.490s
sys 0m0.120s
```

To determine *CTIME1* and *CTIME2* the following is run:

```
$ time /usr/bps/bin/stream-wall& time /usr/bps/bin/stream-wall
real 0m2.839s
user 0m2.660s
sys 0m0.170s
real 0m2.844s
user 0m5.360s
sys 0m0.330s
```

Therefore, *STIME* is 1.6s, *CTIME1* is 2.8s and *CTIME2* is 2.8s. Using the formula shown above, we get a speedup of 1.14! This result indicates that if we run two programs that have a similar memory access patterns to stream, we will see very little performance gain on a dual node when running two programs.

In order to test more "real world" programs the NAS parallel test suite was employed. The test script (see below) was run for both the GNU compilers and the Intel compilers. The speed-up for each test is listed below. An average speedup for the whole suite is also given.

Test	GNU2.96	Intel 6.0
CG	1.1	1.05
BT	1.45	1.34
EP	1.99	2
FT	1.55	1.55
LU	1.4	1.17
IS	1.78	1.7
SP	1.56	1.2
MG	1.34	1.04
Average	**1.52**	**1.38**

Speed-up for concurrent NAS programs on a dual SMP node

Discussion

Before we make any general conclusions, keep in mind that the performance of your application(s) will certainly vary from those tested above. Indeed, virtually all performance issues stem from the specific nature of the application run on the cluster. It is therefore important that you test your application before committing to specific hardware design.

The results are very interesting. For the GNU compiler suite we see a range of 1.1 to 1.99 with an average speedup of 1.52. The results for the Intel compiler follow these trends and range from 1.05 to 2.00 with and average speed-up of 1.38. Neglecting the difference in compilers we see that the best average speed-up is barely over 1.5 times. In the case of CG there is no benefit to running two copies of the program on a dual SMP node. In the case of EP, two copies run virtually as fast as one copy indicating perfect speed-up. Or first general conclusion is:

- memory contention on a dual SMP system is very application sensitive and in many cases will reduce the efficiency of the second processor.

In general, the Intel compiler produced lower speed-up for most tests. This result is attributed to better optimization of the CPU by the Intel compilers which then increases the contention for memory access. This result leads to our second general conclusion:

- Improving compiler efficiency may, in some cases, increase the memory contention and therefore lower the efficiency of the second processor.

While the addition of a second CPU seems minimal from a cost perspective, it may lead to false sense of efficiency for the cluster. Indeed, the tests indicate that a program that normally takes 20 minutes on a single CPU node, may take as long as 40 minutes on an active dual CPU node. This situation may be further compounded by the fact that the batch scheduler may place a program on the first or second processor of any node and thus provide the program with a very heterogeneous memory contention environment. (i.e. some nodes may have large memory contention and other may have none.)

In these tests, we have not considered communication issues or the mix of different programs on the same node. These issues will be addressed in upcoming reports. In addition, we have run the tests on single hardware platform with two compilers.

Test Script

```
#!/bin/bash
PROGS="cg.A.1 bt.A.1 ep.A.1 ft.A.1 lu.A.1 is.A.1 sp.A.1 mg.A.1"
echo "SMP Memory Test" |tee smp-mem.out
echo "`date`" |tee -a smp-mem.out
# generate single cpu codes change -c for different compiler
./run_suite -n 1 -t A -m dummy -c gnu -o
for TEST in $PROGS
do
bin/$TEST>& temp.mem0
bin/$TEST>& temp.mem1 &
bin/$TEST>& temp.mem2
wait
SINGLE=`grep Time temp.mem0 |gawk '{print $5}'`
DOUBLE1=`grep Time temp.mem1 |gawk '{print $5}'`
DOUBLE2=`grep Time temp.mem2 |gawk '{print $5}'`
SPEEDUP=`echo "2 k $DOUBLE1 $DOUBLE2 + $SINGLE * $DOUBLE1 $DOUBLE2 * / p" | dc`
echo "SMP Program Speed-up for $TEST is $SPEEDUP" |tee -a smp-mem.out
done
/bin/rm temp.mem*
echo "`date`" |tee -a smp-mem.out
```

Acknowledgments

I wish to acknowledge all the authors of the tests suites used in this package.
All Trademarks are property of their respective owners.

Copyright