I plan on benchmarking several low cost quad-core processors in the coming weeks. I'm trying decide what to use in my Limulus Project upgrade. Currently, it uses Core 2 processors (three dual core and one quad-core) and while it works quite well, I want to see just how much compute power I can put in a desk-side case. I'll be testing the following:

- Intel Core2 Quad-core Q6600 running at 2.4GHz (current system)
- AMD Phenom II X4 quad-core 910e running at 2.6GHz
- Intel Core i5 Quad-core i5-2400S running at 2.5 GHz

Note that these are all 65W processors, which are necessary for the Limulus design.

When I test processors, I tend to use HPC tests. I do not use HPL (High Performance Linpack) because it can require a lot of tuning to get a good number. I prefer to use standard GNU compilers and the NAS Parallel Benchmark Suite. I also like to use Gromacs. This approach may not test the best performance for a given platform, but it allows me to compare "apples to apples" as close as possible. I can also run these tests on a single core, a multi-core CPU, or a cluster.

When multi-core processors first appeared, I wanted to know the answer to a simple question. If a program runs on a single core in $X$ number of seconds, then $Y$ copies should run in the same amount of time, provided $Y$ is less than or equal to the number of cores and there is perfect memory sharing (i.e. there is no memory contention). If it takes the collection of copies longer to run (than a single copy), then the number of "effective cores" is reduced. Surprisingly, I have not found anyone else that runs these types of tests.

The test is simple enough to run and can be easily scripted. An example script and a link to a complete set of scripts can be found below. To make the test useful, I use the NAS suite compiled for a single core. The NAS suite is a set of eight kernels that represent different aerodynamic application types. Each kernel is self checking and offers a different memory access pattern.
I call the script an "Effective Core" test. As I mentioned, it shows how many cores the application actually "sees." A few months back I was given a dual socket server with two Intel six-core Xeons to test (two Xeon X5670's running at 2.93GHz). There were a total of 12 cores available on a single node. The first thing I did was run my benchmark scripts to see how many effective cores I could see. I ran the tests using 2, 4, 8, 12 copies of the NAS kernels (test size A). The results are below.

<table>
<thead>
<tr>
<th>test</th>
<th>2 copies</th>
<th>4 copies</th>
<th>8 copies</th>
<th>12 copies</th>
</tr>
</thead>
<tbody>
<tr>
<td>cg</td>
<td>2.0</td>
<td>3.7</td>
<td>5.7</td>
<td></td>
</tr>
<tr>
<td>bt</td>
<td>2.0</td>
<td>3.4</td>
<td>4.6</td>
<td></td>
</tr>
<tr>
<td>ep</td>
<td>2.0</td>
<td>4.0</td>
<td>8.4</td>
<td></td>
</tr>
<tr>
<td>ft</td>
<td>2.0</td>
<td>3.8</td>
<td>7.0</td>
<td></td>
</tr>
<tr>
<td>lu</td>
<td>2.0</td>
<td>3.9</td>
<td>6.4</td>
<td></td>
</tr>
<tr>
<td>is</td>
<td>2.0</td>
<td>3.9</td>
<td>7.8</td>
<td></td>
</tr>
<tr>
<td>sp</td>
<td>2.0</td>
<td>3.5</td>
<td>5.0</td>
<td></td>
</tr>
<tr>
<td>mg</td>
<td>2.0</td>
<td>3.8</td>
<td>6.3</td>
<td></td>
</tr>
</tbody>
</table>

**Effective Cores for NAS Parallel Kernels**

Things seem to go well until eight copies are run at that same time. At this point we find two interesting things. First, not all tests "see" all the cores, one test, *bt*, sees only 4.6 cores. Second, another test, *ep*, sees more than eight cores! The reduction in performance can be attributed to memory contention and the increase is probably due to cache effects (test *ep* is CPU bound). Looking at the results for 12 copies, we see some of the stark reality for
Evaluating Multi-core for HPC

Written by
Saturday, 26 March 2011 11:32 - Last Updated Wednesday, 27 April 2011 12:35

multi-core.

Five tests see seven or less "effective cores," that is an efficiency of less than 60%. The worst, \( bt \), only achieves 4.9 effective cores, a 40% efficiency. The others that seem to scale well did so throughout all the tests. Again, the winner, \( ep \), is CPU bound so memory bandwidth will have little effect.

In all fairness, this is probably the worst case scenario for this multi-core system (i.e. 12 copies of the same program running in the same way). However, these are real program kernels that are not contrived tests. If I were to run 12 copies on 12 separate servers, then they would all scale to 12 effective cores. Keep this in mind when placing parallel codes on multi-core clusters.

You can download the tests scripts that will work on 2,4,8,12, and 16 cores. (Note: If I spent more time, I suppose I could make a single script that would use a command line argument, but I'm both lazy and short of time, plus I don't run these scripts all that often.) The following is an example of the script for the four-way test.

```bash
#!/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"
NPBPATH="../npb/"
echo "4 Way SMP Memory Test" |tee "smp-mem-test-4.out"
echo "\"date\"" |tee -a "smp-mem-test-4.out"
# if needed, generate single cpu codes change -c for different compiler
# just check for last program
if [ ! -e "$NPBPATH/bin/mg.A.1" ];
then
  pushd $NPBPATH
  ./run_suite -n 1 -t A -m dummy -c gnu4 -o
  popd
fi
for TEST in $PROGS
do
  $NPBPATH/bin/$TEST>& temp.mem0
  $NPBPATH/bin/$TEST>& temp.mem1 &
  $NPBPATH/bin/$TEST>& temp.mem2 &
  $NPBPATH/bin/$TEST>& temp.mem3 &
  $NPBPATH/bin/$TEST>& temp.mem4 &
  $NPBPATH/bin/$TEST>& temp.mem5 &
  $NPBPATH/bin/$TEST>& temp.mem6 &
  $NPBPATH/bin/$TEST>& temp.mem7 &
  $NPBPATH/bin/$TEST>& temp.mem8 &
  $NPBPATH/bin/$TEST>& temp.mem9 &
done
```

3 / 4
$NPBPATH/bin/$TEST>& temp.mem4
wait
S=`grep Time temp.mem0 |gawk '{print $5}'``
C1=`grep Time temp.mem1 |gawk '{print $5}'``
C2=`grep Time temp.mem2 |gawk '{print $5}'``
C3=`grep Time temp.mem3 |gawk '{print $5}'``
C4=`grep Time temp.mem4 |gawk '{print $5}'``
SPEEDUP=`echo "3 k $S $C1 / $S $C2 / $S $C3 / $S $C4 / + + + p" | dc`
echo "4 Way SMP Program Speed-up for $TEST is $SPEEDUP" |tee -a "smp-mem-test-4.out"
done
/bin/rm temp.mem*
echo "`date`" |tee -a "smp-mem-test-4.out"

The script can be easily modified for other programs. If you want to use the NAS suite, you may find it helpful to download the Beowulf Performance Suite which has the run_suite script that automates running the NAS suite in the script above.