Experiences with Tools at NERSC

Richard Gerber
NERSC User Services

Programming weather, climate, and earth-system models on heterogeneous multi-core platforms
September 7, 2011 at the National Center for Atmospheric Research in Boulder, Colorado
• Thanks for the invitation
• My professional goal is to enable scientists to use HPC easily and effectively
• Contribute to important discoveries about how our natural world works
• Make a difference
• So it is an honor & meaningful to me to participate in this conference
• One of my primary roles is as deputy on our next procurement team & we are extremely interested in learning about your experiences with hybrid systems and programming
Outline

• Recent experiences providing tools
• Observations about tool usage by NERSC users
• What I’d like to see in tools for the HPC community
• I’ll focus on performance tools
  – Who is the audience?
  – Scientists?
  – CS code engineers?
  – Scientist/engineers?

• Development tools
  – Eclipse: We considered installing this, but concluded users best served by installing themselves in their own directories
  – Compilers: PGI, Cray, HMPP directives-based (your experiences?)

• Debuggers
  – Allinea DDT & Rouge Wave Totalview
  – Both are here and will present today
  – Both are installed and supported at NERSC
  – Generally good experiences, but not extensively used by users
Disclaimers

• These are my observations and experiences
• Not a review or critique of any given tool
• Not comprehensive survey of tools
Hi Richard:

I started looking at the single-core performance on Hopper. The first problem that I encountered was that running the code with <tool name redacted> causes it to crash; so I gave up on that approach. What other profiling tools are you planning to support? I did look at gprof on the code but that also seems to have problems.

Chris
• Provide Chris with tools to scale and test his code’s (HiRAM?) performance on NERSC’s Cray XE6 Hopper (#5 in Top 500)

• At the same time test and install some tools to benefit all NERSC users

• Report on my successes and survey the tools we would have available at NERSC (including our test GPU system)
Chris & I went back and forth over a number of months, trying to install and use various tools.

All failed
- Couldn’t build a tool with some specific compiler
- Tool crashed (we did find some bugs)
- Tool appeared to run, but produced no output

Chris found a solution for his problem, but it was a lot of work
- Gettimeofday(), print, somebody’s private tool

My result: no success identifying and installing new tools for our entire user base
NERSC has about 4,000 users

- All levels of sophistication and experience
- We’re committed to supporting both the cutting edge & production HPC computing for the masses

Users often ask for advice on which tools to use and we give them suggestions

Our experience is that very few use programming/debugging/development tools

A few users use a few tools a lot, but many try a tool only once
• Extremely effective?
• More likely: Too confusing, difficult, didn’t work, don’t know how to use, don’t know which to use
• It’s not that we don’t have tools that address specific issues
  – GPU/CUDA tools & compilers
  – TAU, PAPI, HPC Toolkit
  – Craypat, IBM HPC tools, OpenSpeedShop
  – Valgrind
  – Vampirtrace
• But do most users have the resources to learn how to use these tools, esp. when they don’t know if there will be any benefit from any given one?
Vampir w/ CUDA

Timeline

Process 1
CUDA[0:0] 1:1
Thread 1:2
Visualisation::writeFile(Visualisation*, char*)
Process 2
CUDA[0:0] 2:1
Thread 2:2
Visualisation::writeFile(Visualisation*, char*)
Process 3
CUDA[1:0] 3:1
Thread 3:2
Visualisation::writeFile(Visualisation*, char*)
Process 4
CUDA[1:0] 4:1
Thread 4:2
Visualisation::writeFile(Visualisation*, char*)

Function Summary
CUDA[0:0] 1:1, Accumulated Exclusive Time per Function

<table>
<thead>
<tr>
<th>Function</th>
<th>400 μs</th>
<th>200 μs</th>
<th>0 μs</th>
</tr>
</thead>
<tbody>
<tr>
<td>deviceMove...tiles&lt;b1&gt;</td>
<td>598.997 μs</td>
<td>597.939 μs</td>
<td>556.552 μs</td>
</tr>
<tr>
<td>deviceMove...tiles&lt;b0&gt;</td>
<td>488.364 μs</td>
<td>451.28 μs</td>
<td>263.227 μs</td>
</tr>
<tr>
<td>kme</td>
<td>201.646 μs</td>
<td>132.91 μs</td>
<td>88.777 μs</td>
</tr>
</tbody>
</table>

All Processes, Accumulated Exclusive Time per Function

<table>
<thead>
<tr>
<th>Function</th>
<th>30.526 ms</th>
<th>16.907 ms</th>
<th>13.299 ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>CUDA_KERNEL</td>
<td>CUDA_SYNC</td>
<td>MPI</td>
</tr>
<tr>
<td>CUDA_API</td>
<td>8.297 ms</td>
<td>5.318 ms</td>
<td>&lt;1 ms</td>
</tr>
<tr>
<td>VT_CUDA</td>
<td>&lt;100 μs</td>
<td>VT_API</td>
<td></td>
</tr>
</tbody>
</table>

Aggregated Message Volume

- Process 1: 25 KiB
- Process 2: 20 KiB
- Process 3: 15 KiB
- Process 4: 10 KiB
- Process 5: 5 KiB
- Process 6: 0 KiB
Ease of Use

Use

Follow these 10 STEPS to perform the basic analysis of your program using a performance analysis tool, not a debugging tool, start with a fully debugged program capable of running to a planned completion or an intentional termination. This ensures that the correct links and libraries are available and that the program runs without issues.
• Users are asking for tools because HPC systems and programming models are changing

• More and more components to worry about
  – CPU (caches, FPUs, pipelining, …)
  – Data movement to main memory, GPU memory, levels of cache
  – I/O
  – Network (message passing)
  – CPU Threads (OpenMP)
  – GPU performance
Questions to You

• What tools do you use?
• What tools do you want?
• What would you like centers to support?
• Can you get to exascale without tools?
What I Want in a Tool

• Let the users help themselves
• Work for everyone all (most of?) the time
• Easy to use
• Useful
• Easy to interpret the results
• Affordable ($$ or manpower support costs)
• Simple, supplement existing complex tools
  – Point the way for a “deeper dive” in problem areas
• NERSC’s IPM: Integrated Performance Monitoring
• How it works (user perspective)
  – % module load IPM*
  – Run program as normal
  – Look at results on the web
• It’s that easy!
  – And extremely low overhead, so IPM is examining your production code

* (As long as your system supports dynamic load libs)
What IPM measures

- IPM “only” gives a high-level, entire-program-centric view
- Still, very valuable guidance
  - Shows whole-run info per MPI task, OpenMP thread, (CUDA under development)
  - Many pieces of data in one place
- Reveals what many users don’t know about their code
  - High-water memory usage (per task)
  - Load balance
  - Call imbalance
  - MPI time
  - I/O time
**IPM w/ CUDA: Accuracy**

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Kernel Invocations</th>
<th>GPU Kernel Execution Time (sec)</th>
<th>Difference (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlackScholes</td>
<td>512</td>
<td>CUDA Profiler: 2.540677</td>
<td>0.12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IPM: 2.543700</td>
<td></td>
</tr>
<tr>
<td>FDTD3d</td>
<td>5</td>
<td>CUDA Profiler: 0.101354</td>
<td>0.19</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IPM: 0.101550</td>
<td></td>
</tr>
<tr>
<td>MersenneTwister</td>
<td>202</td>
<td>CUDA Profiler: 1.126475</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IPM: 1.127000</td>
<td></td>
</tr>
<tr>
<td>MonteCarlo</td>
<td>2</td>
<td>CUDA Profiler: 0.001988</td>
<td>1.87</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IPM: 0.002025</td>
<td></td>
</tr>
<tr>
<td>concurrentKernels</td>
<td>9</td>
<td>CUDA Profiler: 0.613755</td>
<td>0.04</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IPM: 0.614000</td>
<td></td>
</tr>
<tr>
<td>eigenvalues</td>
<td>300</td>
<td>CUDA Profiler: 5.328266</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IPM: 5.331000</td>
<td></td>
</tr>
<tr>
<td>quasirandomGenerator</td>
<td>42</td>
<td>CUDA Profiler: 0.039536</td>
<td>0.51</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IPM: 0.039736</td>
<td></td>
</tr>
<tr>
<td>scan</td>
<td>3300</td>
<td>CUDA Profiler: 1.412912</td>
<td>1.22</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IPM: 1.430200</td>
<td></td>
</tr>
</tbody>
</table>

- Benchmarks from the CUDA SDK
  - Run on 1 node of Dirac @ NERSC
    - 2x Nehalem quad core, 1x NVIDIA Tesla C2050 (“Fermi”) GPU
  - Comparison of the results from the CUDA profiler with IPM
  - Very good agreement of the results
  - IPM timings always larger than CUDA profiler (bracketing)
Click on your job from a list on the NERSC web site.
Click on the metric you want.

### IPM Examples

<table>
<thead>
<tr>
<th>Metric</th>
<th>Sum over all tasks</th>
<th>Average (per task)</th>
<th>Task CV (%)</th>
<th>Task Minimum</th>
<th>Task Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregate Floating Point Operations (Flop x 10^3)</td>
<td>1.313e+01</td>
<td>2.565e-02</td>
<td>6.10</td>
<td>1.713e-02</td>
<td>2.758e-02</td>
</tr>
<tr>
<td>GFlop/sec</td>
<td>1.492e-01</td>
<td>2.039e-04</td>
<td>6.10</td>
<td>1.314e-04</td>
<td>5.114e-04</td>
</tr>
<tr>
<td>Maximum Memory Usage (Gbytes)</td>
<td>3.226e+01</td>
<td>6.301e-02</td>
<td>10.12</td>
<td>5.701e-02</td>
<td>1.947e-01</td>
</tr>
<tr>
<td>Time Spent in MPI Routines (sec)</td>
<td>4.044e+04</td>
<td>7.898e+01</td>
<td>4.05</td>
<td>9.801e+00</td>
<td>8.359e+01</td>
</tr>
<tr>
<td>Wallclock Time (sec)</td>
<td>4.537e+04</td>
<td>8.861e+01</td>
<td>0.10</td>
<td>8.848e+01</td>
<td>8.895e+01</td>
</tr>
</tbody>
</table>

Memory in units of gigabytes; time in seconds.

<table>
<thead>
<tr>
<th>Counter Name</th>
<th>Sum over all tasks</th>
<th>Average (per task)</th>
<th>Task CV (%)</th>
<th>Task Minimum</th>
<th>Task Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAPI FP OPE</td>
<td>1.151799e+12</td>
<td>2.269139e+05</td>
<td>6.09</td>
<td>1.51503e+09</td>
<td>2.439529e+09</td>
</tr>
</tbody>
</table>

### MPI Time Statistics - 512 tasks

<table>
<thead>
<tr>
<th>Call</th>
<th>Sum over all tasks</th>
<th>Average (per task)</th>
<th>Task CV (%)</th>
<th>Task Minimum</th>
<th>Task Maximum</th>
<th>% of MPI</th>
<th>% of wall</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Bcast</td>
<td>3.517e+01</td>
<td>6.869e+01</td>
<td>4.68</td>
<td>4.342e-01</td>
<td>7.269e+01</td>
<td>86.969</td>
<td>77.520</td>
</tr>
<tr>
<td>MPI_Scatennv</td>
<td>2.589e+03</td>
<td>5.057e+00</td>
<td>5.79</td>
<td>1.059e+00</td>
<td>5.540e+00</td>
<td>6.403</td>
<td>5.707</td>
</tr>
<tr>
<td>MPI_Wait</td>
<td>2.176e+03</td>
<td>4.249e+00</td>
<td>17.82</td>
<td>1.250e+00</td>
<td>4.988e+00</td>
<td>5.380</td>
<td>4.795</td>
</tr>
<tr>
<td>MPI_Gather</td>
<td>4.312e+02</td>
<td>8.422e-01</td>
<td>36.44</td>
<td>3.552e-03</td>
<td>2.271e+00</td>
<td>1.066</td>
<td>0.950</td>
</tr>
<tr>
<td>MPI_Init</td>
<td>5.250e+01</td>
<td>1.025e-02</td>
<td>11.96</td>
<td>7.182e-02</td>
<td>1.259e-01</td>
<td>0.130</td>
<td>0.116</td>
</tr>
<tr>
<td>MPI_Finalize</td>
<td>1.033e+01</td>
<td>2.017e-02</td>
<td>10.21</td>
<td>1.217e-02</td>
<td>2.613e-02</td>
<td>0.026</td>
<td>0.023</td>
</tr>
<tr>
<td>MPI_Comm_rank</td>
<td>4.563e-01</td>
<td>8.913e-04</td>
<td>4.74</td>
<td>7.799e-04</td>
<td>1.404e-03</td>
<td>0.001</td>
<td>0.001</td>
</tr>
<tr>
<td>MPI_Comm_size</td>
<td>9.629e-02</td>
<td>1.881e-04</td>
<td>10.65</td>
<td>1.462e-04</td>
<td>4.859e-04</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>MPI_Init</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>MPI_Finalize</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
<td>0.000e+00</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
</tbody>
</table>
Task distribution of Aggregate Floating Point Operations ($\text{Flop} \times 10^{\text{\text{*2}}}$) - as a percentage of maximum

The MPI rank is the sum of the column and row indices in the table.

Table Columns: \(\text{gflopsec, Gflop/sec, Time spent in MPI routines (sec), Wallclock time (sec)}\)

<table>
<thead>
<tr>
<th>Metric</th>
<th>Sum</th>
<th>Mean</th>
<th>Std. Dev</th>
<th>CV (%)</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aggregate Floating Point Operations (\text{Flop} \times 10^{\text{\text{*2}}}))</td>
<td>3.012e+02</td>
<td>1.470e-01</td>
<td>4.946e-03</td>
<td>3.36e+00</td>
<td>1.395e-01</td>
<td>2.161e-01</td>
</tr>
<tr>
<td>Gflop/sec</td>
<td>6.147e-01</td>
<td>3.002e-04</td>
<td>1.008e-05</td>
<td>3.36e+00</td>
<td>2.847e-04</td>
<td>4.411e-04</td>
</tr>
<tr>
<td>Maximum Memory Usage (GBytes)</td>
<td>4.101e+02</td>
<td>2.002e-01</td>
<td>9.606e-03</td>
<td>4.80e+00</td>
<td>1.781e-01</td>
<td>2.448e-01</td>
</tr>
<tr>
<td>Time spent in MPI routines (sec)</td>
<td>1.228e+06</td>
<td>5.998e+02</td>
<td>4.986e+01</td>
<td>6.31e+00</td>
<td>5.177e+02</td>
<td>6.903e+02</td>
</tr>
<tr>
<td>Wallclock time (sec)</td>
<td>1.003e+06</td>
<td>4.898e+02</td>
<td>6.028e+02</td>
<td>1.31e-02</td>
<td>4.898e+02</td>
<td>4.927e+02</td>
</tr>
</tbody>
</table>

CV = Coefficient of Variance = \(\text{Standard Deviation} / \text{Mean}\)
Time spent by each task in `MPI_Allreduce` as a percentage of the maximum value.

The MPI rank represented by each cell in the table is the sum of the cell’s column and row indices.

Table Columns: 48

| 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 |

Time vs. MPI Rank for `MPI_Allreduce`
IPM Examples

The MPI rank is the sum of the column and row indices in the table.
IPM Examples

Time spent by each task in *MPI_RECV* as a percentage of the maximum value.

The MPI rank represented by each cell in the table is the sum of the cell’s column and row indices.

Table Columns: 32 32

Time vs. MPI Rank for *MPI_RECV*
• HPC tools are, in general, difficult to use or require a large investment of time to learn to use and interpret the results

• There is a need for tools to help users get to exascale (or even petascale).

• Simple, easy to use tools would be extremely useful, even if they did not give low-level details