Performance Debugging Techniques For HPC Applications

David Skinner
deskinner@lbl.gov
CS267 Feb 17 2015
Today’s Topics

- **Principles**
  - Topics in performance scalability
  - Examples of areas where tools can help
- **Practice**
  - Where to find tools
  - Specifics to NERSC’s Hopper/Edison...

Scope & Audience:
The budding simulation scientist, I want to compute.
The compiler/middleware dev, I want to code.
Overview of an HPC Facility

Serving all of DOE Office of Science
  domain breadth
  range of scales

Science driven
  sustained performance
  on real apps

Lots of users
  ~6K active
  ~500 logged in
  ~450 projects

Architecture aware
  system procurements driven by workload needs
Big Picture of Performance and Scalability
Performance, more than a single number

- Plan where to put effort
- Optimization in one area can de-optimize another
- Timings come from timers and also from your calendar, time spent coding
- Sometimes a slower algorithm is simpler to verify correctness

Formulate Research Problem

Queue Wait

Data?

UQ VV

Understand & Publish!

Perf Debug

jobs jobs jobs jobs

Debug

Algorithms
Performance is Relative

• To your goals
  – Time to solution, $T_q + T_{wall}$ …
  – Your research agenda
  – Efficient use of allocation

• To the
  – application code
  – input deck
  – machine type/state

Suggestion:
Focus on specific use cases as opposed to making everything perform well. Bottlenecks can shift.
Specific Facets of Performance

• Serial
  – Leverage ILP on the processor
  – Feed the pipelines
  – Reuse data in cache
  – Exploit data locality

• Parallel
  – Exposing task level concurrency
  – Minimizing latency effects
  – Maximizing work vs. communication
Performance is Hierarchical

- Registers
- Caches
- Local Memory
- Remote Memory
- Disk / Filesystem

instructions & operands

Think Globally, Compute Locally

lines

pages

messages

blocks, files
…on to specifics about HPC tools

Mostly at NERSC but fairly general
Tools are Hierarchical

- Registers
- Caches
- Local Memory
- Remote Memory
- Disk / Filesystem

- PAPI
- valgrind
- PMPI
- POSIX

Craypat
IPM
Tau
...

POSIX
PMPI
valgrind
PAPI
Craypat
IPM
Tau
...

Disk / Filesystem
Remote Memory
Local Memory
Caches
Registers
HPC Perf Tool Mechanisms (the how part)

- **Sampling**
  - Regularly interrupt the program and record where it is
  - Build up a statistical profile

- **Tracing / Instrumenting**
  - Insert hooks into program to record and time events

- **Use Hardware Event Counters**
  - Special registers count events on processor
  - E.g. floating point instructions
  - Many possible events
  - Only a few (~4 counters)
Things HPC tools may ask you to do

- Modify your code with macros, API calls, timers
- Re-compile your code
- Transform your binary for profiling/tracing
- Run the transformed binary
  - A data file is produced
- Interpret the results with another tool
Performance Tools @ NERSC

- **Vendor Tools:**
  - CrayPat

- **Community Tools:**
  - TAU (U. Oregon via ACTS)
  - PAPI (Performance Application Programming Interface)
  - gprof, many more,

- **Center tools:**
  - Integrated Performance Monitoring
What can HPC tools tell us?

- CPU and memory usage
  - FLOP rate
  - Memory high water mark
- OpenMP
  - OMP overhead
  - OMP scalability (finding right # threads)
- MPI
  - Detecting load imbalance
  - % wall time in communication
  - Analyzing message sizes
Using the right tool

Tools can add overhead to code execution
• What level can you tolerate?

Tools can add overhead to scientists
• What level can you tolerate?

Scenarios:
• Debugging a code that is “slow”
• Detailed performance debugging
• Performance monitoring in production
One quick tool example: IPM

- Integrated Performance Monitoring
- MPI profiling, hardware counter metrics, POSIX IO profiling
- IPM requires no code modification & no instrumented binary
  - Only a “module load ipm” before running your program on systems that support dynamic libraries
  - Else link with the IPM library
- IPM uses hooks already in the MPI library to intercept your MPI calls and wrap them with timers and counters
IPM: Let’s See

1) Do “module load ipm”, link with $IPM, then run normally
2) Upon completion you get

Maybe that’s enough. If so you’re done.
Have a nice day 😊

---

#IPM2v0.xx##################################################
#
# command   : ./fish -n 10000
# start     : Tue Feb 08 11:05:21 2011   host      : nid06027
# stop      : Tue Feb 08 11:08:19 2011   wallclock : 177.71
# mpi_tasks : 25 on 2 nodes              %comm     : 1.62
# mem [GB]  : 0.24                         gflop/sec : 5.06
...
**IPM : IPM_PROFILE=full**

<table>
<thead>
<tr>
<th>#</th>
<th>host</th>
<th>mpi_tasks</th>
<th># start</th>
<th>wallclock</th>
<th>%comm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>s05601/006035314C00_AIX</td>
<td>32 on 2 nodes</td>
<td>11/30/04/14:35:34</td>
<td>29.975184 sec</td>
<td>27.72</td>
</tr>
<tr>
<td></td>
<td>gbytes</td>
<td>6.65863e-01 total</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#</td>
<td>wallclock</td>
<td>[total]</td>
<td>[avg]</td>
<td>min</td>
<td>max</td>
</tr>
<tr>
<td></td>
<td>953.272</td>
<td>29.7897</td>
<td>29.6092</td>
<td>29.9752</td>
<td></td>
</tr>
<tr>
<td></td>
<td>user</td>
<td>837.25</td>
<td>26.1641</td>
<td>25.71</td>
<td>26.92</td>
</tr>
<tr>
<td></td>
<td>system</td>
<td>60.6</td>
<td>1.89375</td>
<td>1.52</td>
<td>2.59</td>
</tr>
<tr>
<td></td>
<td>mpi</td>
<td>264.267</td>
<td>8.25834</td>
<td>7.73025</td>
<td>8.70985</td>
</tr>
<tr>
<td></td>
<td>%comm</td>
<td>27.7234</td>
<td>25.8873</td>
<td>29.3705</td>
<td></td>
</tr>
<tr>
<td></td>
<td>gflop/sec</td>
<td>2.33478</td>
<td>0.0729619</td>
<td>0.072204</td>
<td>0.0745817</td>
</tr>
<tr>
<td></td>
<td>gbytes</td>
<td>0.665863</td>
<td>0.0208082</td>
<td>0.0195503</td>
<td>0.0237541</td>
</tr>
<tr>
<td></td>
<td>PM_FPU0_CMPL</td>
<td>2.28827e+10</td>
<td>7.15084e+08</td>
<td>7.07373e+08</td>
<td>7.30171e+08</td>
</tr>
<tr>
<td></td>
<td>PM_FPU1_CMPL</td>
<td>1.70657e+10</td>
<td>5.33304e+08</td>
<td>5.28487e+08</td>
<td>5.42882e+08</td>
</tr>
<tr>
<td></td>
<td>PM_FPU_FMA</td>
<td>3.00371e+10</td>
<td>9.3866e+08</td>
<td>9.27762e+08</td>
<td>9.62547e+08</td>
</tr>
<tr>
<td></td>
<td>PM_INST_CMPL</td>
<td>2.78819e+11</td>
<td>8.71309e+09</td>
<td>8.20981e+09</td>
<td>9.21761e+09</td>
</tr>
<tr>
<td></td>
<td>PM_LD_CMPL</td>
<td>1.25478e+11</td>
<td>3.92118e+09</td>
<td>3.74541e+09</td>
<td>4.11658e+09</td>
</tr>
<tr>
<td></td>
<td>PM_ST_CMPL</td>
<td>7.45961e+10</td>
<td>2.33113e+09</td>
<td>2.21664e+09</td>
<td>2.46327e+09</td>
</tr>
<tr>
<td></td>
<td>PM_TLB_MISS</td>
<td>2.45894e+08</td>
<td>7.68418e+06</td>
<td>6.98733e+06</td>
<td>2.05724e+07</td>
</tr>
<tr>
<td></td>
<td>PM_CYC</td>
<td>3.0575e+11</td>
<td>9.55467e+09</td>
<td>9.36585e+09</td>
<td>9.62227e+09</td>
</tr>
<tr>
<td>#</td>
<td>time</td>
<td>[calls]</td>
<td>&lt;%mpi&gt;</td>
<td>&lt;%wall&gt;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MPI_Send</td>
<td>188.386</td>
<td>639616</td>
<td>71.29</td>
<td>19.76</td>
</tr>
<tr>
<td></td>
<td>MPI_Wait</td>
<td>69.5032</td>
<td>639616</td>
<td>26.30</td>
<td>7.29</td>
</tr>
<tr>
<td></td>
<td>MPI_Recv</td>
<td>6.34936</td>
<td>639616</td>
<td>2.40</td>
<td>0.67</td>
</tr>
<tr>
<td></td>
<td>MPI_Barrier</td>
<td>0.0177442</td>
<td>32</td>
<td>0.01</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>MPI_Reduce</td>
<td>0.00540609</td>
<td>32</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>MPI_Comm_rank</td>
<td>0.00465156</td>
<td>32</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td></td>
<td>MPI_Comm_size</td>
<td>0.000145341</td>
<td>32</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>
Advice: Develop (some) portable approaches to performance

• There is a tradeoff between vendor-specific and vendor neutral tools
  – Each have their roles, vendor tools can often dive deeper

• Portable approaches allow apples-to-apples comparisons
  – Events, counters, metrics may be incomparable across vendors

• You can find printf most places
  – Put a few timers in your code?

printf? really?
Yes really.
Performance Principles in HPC Tools
Scaling studies involve changing the degree of parallelism.
  – Will we be changing the problem also?

**Strong scaling**
  – Fixed problem size

**Weak scaling**
  – Problem size grows with additional resources

**Speed up** = $\frac{T_s}{T_p(n)}$

**Efficiency** = $\frac{T_s}{(n \times T_p(n))}$

Be aware there are multiple definitions for these terms
Strong vs. Weak Scalability (applications)

- **Strong Scaling**
  - Overall problem size is fixed
  - Goal is to run same size problem faster
  - Perfect scaling means problem runs in $1/P$ time (compared to serial)

- **Weak Scaling**
  - Problem size *per processor* is fixed
  - Goal is to run larger problem in same amount of time
  - Perfect scaling means a problem $P_x$ larger runs in same time as single processor run

from Supercomputing 101, R. Nealy (good read)
MPI Scalability: Point to Point

Ferreira, Kurt B., et al, SC14
MPI Scalability: Disseminate

Ferreira, Kurt B., et al, SC14
MPI Scalability: Synchronization

Ferreira, Kurt B., et al, SC14
Let’s look at a parallel algorithm.

With a particular goal in mind, we systematically vary concurrency and/or problem size

Example:

How large a 3D \((n^3)\) FFT can I efficiently run on 1024 cpus?

Looks good?

Watch out for variability: cross-job contention, OS jitter, perf weather
Let’s look a little deeper....
Performance in a 3D box (Navier-Stokes)

Simple stencil, simple grid

Transpose/FFT is key to wallclock performance

What if the problem size or core count change?

One timestep, one node

61% time in FFT
The FFT(W) scalability landscape

3D complex-complex FFTW ($N=n*n*n$)

Don’t assume performance is smooth → scaling study

Whoa!

Why so bumpy?

- Algorithm complexity or switching
- Communication protocol switching
- Inter-job contention
- ~bugs in vendor software

Don’t assume performance is smooth → scaling study

U.S. DEPARTMENT OF
Office of
E N E R G Y
Science
Scaling is not always so tricky

Main loop in jacobi_omp.f90; ngrid=6144 and maxiter=20
Weak Scaling and Communication
Load Imbalance: Pitfall 101

Communication Time: 64 tasks show 200s, 960 tasks show 230s

MPI ranks sorted by total communication time
Load Balance: cartoon

Unbalanced:

Task 1
Task 2
Task 3
Task 4

Balanced:

Task 1
Task 2
Task 3
Task 4

Universal App

Sync
Flop
I/O

Time saved by load balance
Watch out for the little stuff.

Even “trivial” MPI (or any function call) can add up

Where does your code spend time?
Communication Topology
Where are bottlenecks in the code & machine?

Node Boundaries, P2P, Collective
Interconnect Networks – Tying it all Together

nD Hypercube
- Number of outgoing ports scales with the log of the machine size
- Difficult to scale out

Dragonfly
- Hierarchical design
- All-to-all connectivity between groups

nD Torus
- Nearest neighbor
- Torus == “wrap around”
- BlueGene/Q is a 5D torus

Fat Tree
- Increases available bandwidth higher levels in the switch tree
- Tries to neutralize effect of hop counts

Programmer Challenges:
- Job layout
- Topology mapping
- Contention/performance
- DOE pushes boundaries of scaling

Research Approaches:
- Network simulation
- Design to match application needs
- DesignForward

Future:
- Hierarchical combinations of these.
- Node Network Interface (NIC) moving onto Processor
Communication Topology
As maps of data movement
Cactus Communication *PDE Solvers on Block Structured Grids*
PARATEC Communication

PARATEC Point-to-Point Communication (bytes)

Processor

0 50 100 150 200 250

Processor

0 50 100 150 200 250

1.2e+09
1.0e+09
8.0e+08
6.0e+08
4.0e+08
2.0e+08
0.0e+00

3D FFT
not all performance is inside the app.

Time to solution?

Don’t forget the batch queue.
A few notes on queue optimization

Consider your schedule

- **Charge factor**
  regular vs. low
- **Scavenger queues**
  when you can tolerate interruption
- **Xfer queues**
  Downshift concurrency

Consider the queue constraints

- **Run limit**: How many running at once
- **Queue limit**: How many queued
- **Wall limit**
  Soft (can you checkpoint?)
  Hard (game over)

*BTW, jobs can submit other jobs*
Marshalling your own workflow

• Lots of choices in general
  – PBS, Hadoop, CondorG, MySGE
• On hopper it’s easy

```bash
#PBS -l mppwidth=4096
aprun -n 512 ./cmd &
aprun -n 512 ./cmd &
...
aprun -n 512 ./cmd &
wait
```

```bash
#PBS -l mppwidth=4096
while(work_left) {
  if(nodes_avail) {
    aprun -n X next_job &
  }
  wait
}
```
Scientific Workflows More Generally

- **Tigres**: Design templates for scientific workflows
  - Explicitly support Sequence, Parallel, Split, Merge
- **Fireworks**: High Throughput job scheduler
  - Runs on HPC systems

Ask about NERSC DAS

Mining Databases for Predicting New Materials

Performance is Time-to-knowledge