Performance Debugging Techniques For HPC Applications

David Skinner
deskinner@lbl.gov
CS267 Feb 19 2013
Today’s Topics

• **Principles**
  – Topics in performance scalability
  – Examples of areas where tools can help

• **Practice**
  – Where to find tools
  – Specifics to NERSC and Hopper/Franklin

• **Scope & Audience**
  – Budding simulation scientist app dev
  – Compiler/middleware dev, YMMV
• Serving all of DOE Office of Science domain breadth range of scales
• Science driven sustained performance on real apps

• Lots of users
  ~4.5K active
  ~500 logged in
  ~300 projects

• Architecture aware 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?

jobs jobs jobs

UQ VV

Understand & Publish!

Coding

Debug

Perf Debug
Performance is Relative

• To your goals
  – Time to solution, \( T_q + T_{\text{wall}} \) …
  – 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**
  – Expose task level concurrency
  – Minimizing latency effects
  – Maximizing work vs. communication
Performance is Hierarchical

- Registers: instructions & operands
- Caches
- Local Memory: lines, pages
- Remote Memory: messages
- Disk / Filesystem: blocks, files

Think Globally, Compute Locally
...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
- SAR

- Craypat
- IPM
- Tau
HPC Perf Tool Mechanisms

- **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

- (Sometimes) Modify your code with macros, API calls, timers
- Re-compile your code
- Transform your binary for profiling/tracing with a tool
- 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

• IPM: 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**
  - % wall time in communication
  - Detecting load imbalance
  - 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
Introduction to CrayPat

• Suite of tools to provide a wide range of performance-related information

• Can be used for both sampling and tracing user codes
  – with or without hardware or network performance counters
  – Built on PAPI

• Supports Fortran, C, C++, UPC, MPI, Coarray Fortran, OpenMP, Pthreads, SHMEM

• Man pages
  – intro_craypat(1), intro_app2(1), intro_papi(1)
Using CrayPat @ Hopper

1. Access the tools
   - module load perftools

2. Build your application; keep .o files
   - make clean
   - make

3. Instrument application
   - pat_build ... a.out
   - Result is a new file, a.out+pat

4. Run instrumented application to get top time consuming routines
   - aprun ... a.out+pat
   - Result is a new file XXXXX.xf (or a directory containing .xf files)

5. Run pat_report on that new file; view results
   - pat_report XXXXX.xf > my_profile
   - vi my_profile
   - Result is also a new file: XXXXX.ap2
### Guidelines for Optimization

<table>
<thead>
<tr>
<th>Derived metric</th>
<th>Optimization needed when*</th>
<th>PAT_RT_HWP_C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computational intensity</td>
<td>&lt; 0.5 ops/ref</td>
<td>0, 1</td>
</tr>
<tr>
<td>L1 cache hit ratio</td>
<td>&lt; 90%</td>
<td>0, 1, 2</td>
</tr>
<tr>
<td>L1 cache utilization (misses)</td>
<td>&lt; 1 avg hit</td>
<td>0, 1, 2</td>
</tr>
<tr>
<td>L1+L2 cache hit ratio</td>
<td>&lt; 92%</td>
<td>2</td>
</tr>
<tr>
<td>L1+L2 cache utilization (misses)</td>
<td>&lt; 1 avg hit</td>
<td>2</td>
</tr>
<tr>
<td>TLB utilization</td>
<td>&lt; 0.9 avg use</td>
<td>1</td>
</tr>
<tr>
<td>(FP Multiply / FP Ops) or (FP Add / FP Ops)</td>
<td>&lt; 25%</td>
<td>5</td>
</tr>
<tr>
<td>Vectorization</td>
<td>&lt; 1.5 for dp; 3 for sp</td>
<td>12 (13, 14)</td>
</tr>
</tbody>
</table>

* Suggested by Cray
Perf Debug and Production Tools

- 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

```bash
##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
...
```

Maybe that’s enough. If so you’re done.
Have a nice day 😊
IPM : IPMPROFILE=full

<table>
<thead>
<tr>
<th># host</th>
<th>mpi_tasks : 32 on 2 nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td># start</td>
<td>wallclock : 29.97184 sec</td>
</tr>
<tr>
<td># stop</td>
<td>%comm : 27.72</td>
</tr>
<tr>
<td># gbytes</td>
<td>gflop/sec : 2.33478e+00 total</td>
</tr>
<tr>
<td># wallclock</td>
<td>[total]</td>
</tr>
<tr>
<td># user</td>
<td>[avg]</td>
</tr>
<tr>
<td># system</td>
<td>min</td>
</tr>
<tr>
<td># mpi</td>
<td>max</td>
</tr>
<tr>
<td># %comm</td>
<td>&lt;mpi&gt;</td>
</tr>
<tr>
<td># gflop/sec</td>
<td>&lt;%mpi&gt;</td>
</tr>
<tr>
<td># gbytes</td>
<td>&lt;%wall&gt;</td>
</tr>
<tr>
<td># PM_FPU0_CMPL</td>
<td>2.28827e+10</td>
</tr>
<tr>
<td># PM_FPU1_CMPL</td>
<td>1.70657e+10</td>
</tr>
<tr>
<td># PM_FPU_FMA</td>
<td>3.00371e+10</td>
</tr>
<tr>
<td># PM_INST_CMPL</td>
<td>2.78819e+11</td>
</tr>
<tr>
<td># PM_LD_CMPL</td>
<td>1.25478e+11</td>
</tr>
<tr>
<td># PM_ST_CMPL</td>
<td>7.45961e+10</td>
</tr>
<tr>
<td># PM_TLB_MISS</td>
<td>2.45894e+08</td>
</tr>
<tr>
<td># PM_CYC</td>
<td>3.0575e+11</td>
</tr>
<tr>
<td># MPI_Send</td>
<td>188.386</td>
</tr>
<tr>
<td># MPI_Wait</td>
<td>69.5032</td>
</tr>
<tr>
<td># MPI_Irecv</td>
<td>6.34936</td>
</tr>
<tr>
<td># MPI_Barrier</td>
<td>0.0177442</td>
</tr>
<tr>
<td># MPI_Reduce</td>
<td>0.00540609</td>
</tr>
<tr>
<td># MPI_Comm_rank</td>
<td>0.00465156</td>
</tr>
<tr>
<td># MPI_Comm_size</td>
<td>0.000145341</td>
</tr>
</tbody>
</table>

U.S. DEPARTMENT OF

Energy

Office of

Science
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.
Examples of HPC tool usage
Scaling: definitions

- Scaling studies involve changing the degree of parallelism. Will we be change 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** $= \frac{T_s}{(n \cdot T_p(n))}$

Be aware there are multiple definitions for these terms
Conducting a scaling study

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?
Let’s look a little deeper....
The scalability landscape

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

<table>
<thead>
<tr>
<th>MPI Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
</tr>
<tr>
<td>32</td>
</tr>
<tr>
<td>64</td>
</tr>
<tr>
<td>128</td>
</tr>
<tr>
<td>256</td>
</tr>
<tr>
<td>512</td>
</tr>
<tr>
<td>1024</td>
</tr>
</tbody>
</table>

Why so bumpy?

- Algorithm complexity or switching
- Communication protocol switching
- Inter-job contention
- ~bugs in vendor software
Not always so tricky

Main loop in jacobi_omp.f90; ngrid=6144 and maxiter=20
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
Too much communication
Simple Stuff: What’s wrong here?

Communication Event Statistics (100.00% detail)

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Buffer Size</th>
<th>Ncalls</th>
<th>Total Time</th>
<th>Min Time</th>
<th>Max Time</th>
<th>%MPI</th>
<th>%Wall</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Allreduce</td>
<td>8</td>
<td>3278848</td>
<td>124132.547</td>
<td>0.000</td>
<td>114.920</td>
<td>59.35</td>
<td>16.88</td>
</tr>
<tr>
<td>MPI_Comm_rank</td>
<td>0</td>
<td>35173439489</td>
<td>43439.102</td>
<td>0.000</td>
<td>41.961</td>
<td>20.77</td>
<td>5.91</td>
</tr>
<tr>
<td>MPI_Wait</td>
<td>98304</td>
<td>13221888</td>
<td>15710.953</td>
<td>0.000</td>
<td>3.586</td>
<td>7.51</td>
<td>2.14</td>
</tr>
<tr>
<td>MPI_Wait</td>
<td>196608</td>
<td>13221888</td>
<td>5331.236</td>
<td>0.000</td>
<td>5.716</td>
<td>2.55</td>
<td>0.72</td>
</tr>
<tr>
<td>MPI_Wait</td>
<td>589824</td>
<td>206848</td>
<td>5166.272</td>
<td>0.000</td>
<td>7.265</td>
<td>2.47</td>
<td>0.70</td>
</tr>
</tbody>
</table>
Not so simple: Comm. topology

MILC

MAESTRO

GTC

PARATEC

IMPACT-T

CAM
Performance in Batch Queue Space
A few notes on queue optimization

Consider your schedule

- Charge factor
  - regular vs. low
- Scavenger queues
- Xfer queues
  - Downshift concurrency

Consider the queue constraints

- Run limit
- Queue limit
- Wall limit
  - Soft (can you checkpoint?)

Jobs can submit other jobs
Marshalling your own workflow

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

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

```
#PBS -l mppwidth=4096
while(work_left) {
  if(nodes_avail) {
    aprun –n X next_job &
  }
  wait
}```
Contacts:  
help@nersc.gov  
deskinner@lbl.gov  

Thanks!