Topics

• Cray performance tools overview
• Steps to using the tools
• Performance measurement on the Cray XE system
• Using HW performance counters
• Profiling applications
• Visualization of performance data through pat_report
• Visualization of performance data through Cray Apprentice2
• MPICH Rank Order
Overview
Design Goals
Assist the user with application performance analysis and optimization

• Help user identify important and meaningful information from potentially massive data sets
• Help user identify problem areas instead of just reporting data
• Bring optimization knowledge to a wider set of users
Design Goals
Focus on ease of use and intuitive user interfaces
  • Automatic program instrumentation
  • Automatic analysis
Target scalability issues in all areas of tool development
  • Data management
    • Storage, movement, presentation
**Strengths**

*solution from instrumentation to measurement to analysis to visualization of data*

- Performance measurement and analysis on large systems
  - Automatic Profiling Analysis
  - Load Imbalance
  - HW counter derived metrics
  - Predefined trace groups provide performance statistics for libraries called by program (blas, lapack, pgas runtime, netcdf, hdf5, etc.)
- Observations of inefficient performance
- Data collection and presentation filtering
- Data correlates to user source (line number info, etc.)
- Support MPI, SHMEM, OpenMP, UPC, CAF, OpenACC
- Access to network counters
- Minimal program perturbation
The Cray Performance Analysis Framework

Supports traditional post-mortem performance analysis

- Automatic identification of performance problems
  - Indication of causes of problems
  - Suggestions of modifications for performance improvement
- **pat_build**: provides automatic instrumentation
- CrayPat run-time library collects measurements (transparent to the user)
- **pat_report** performs analysis and generates text reports
- **pat_help**: online help utility
- **Cray Apprentice2**: graphical visualization tool

- To access software:
  - module load perftools
The Cray Performance Analysis Framework

CrayPat
- Instrumentation of optimized code
- No source code modification required
- Data collection transparent to the user
- Text-based performance reports
- Derived metrics
- Performance analysis

Cray Apprentice2
- Performance data visualization tool
- Call tree view
- Source code mappings
Steps To Using Tools
Application Instrumentation with pat_build

- pat_build is a stand-alone utility that instruments the application for performance collection
- Requires no source code or makefile modification
- Automatic instrumentation at group (function) level
  - Groups: mpi, io, heap, math SW, …
- Performs link-time instrumentation
- Requires object files
- Instruments optimized code
- Generates stand-alone instrumented program
- Preserves original binary
Application Instrumentation with pat_build (2)

- Supports two categories of experiments
  - Asynchronous experiments (sampling) which capture values from the call stack or the program counter at specified intervals or when a specified counter overflows.
  - Event-based experiments (tracing) which count some events such as the number of times a specific system call is executed.

- While tracing provides most useful information, it can be very heavy if the application runs on a large number of cores for a long period of time.
- Sampling can be useful as a starting point, to provide a first overview of the work distribution.
<table>
<thead>
<tr>
<th>Samp%</th>
<th>Samp</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Group</th>
<th>Function</th>
<th>Source</th>
<th>Line</th>
<th>PE=HIDE</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0%</td>
<td>8376.9</td>
<td>--</td>
<td>--</td>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>93.2%</td>
<td>7804.0</td>
<td>--</td>
<td>--</td>
<td>USER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>51.7%</td>
<td>4328.7</td>
<td>--</td>
<td>--</td>
<td>calc3_</td>
<td></td>
<td>heidi/DARPA/cache_util/calc3.do300-ijswap.F</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15.7%</td>
<td>1314.4</td>
<td>93.6</td>
<td>6.8%</td>
<td>line.78</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13.9%</td>
<td>1167.7</td>
<td>98.3</td>
<td>7.9%</td>
<td>line.79</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14.5%</td>
<td>1211.6</td>
<td>97.4</td>
<td>7.6%</td>
<td>line.80</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.2%</td>
<td>103.1</td>
<td>26.9</td>
<td>21.2%</td>
<td>line.93</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.1%</td>
<td>88.4</td>
<td>22.6</td>
<td>20.8%</td>
<td>line.94</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0%</td>
<td>84.5</td>
<td>17.5</td>
<td>17.6%</td>
<td>line.95</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.0%</td>
<td>86.8</td>
<td>33.2</td>
<td>28.2%</td>
<td>line.96</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.3%</td>
<td>105.0</td>
<td>23.0</td>
<td>18.4%</td>
<td>line.97</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.4%</td>
<td>116.5</td>
<td>24.5</td>
<td>17.7%</td>
<td>line.98</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

144.1 38%
Where to Run Instrumented Application

- By default, data files are written to the execution directory.
- Default behavior requires file system that supports record locking, such as Lustre ( /mnt/snx3/… , /lus/…, /scratch/…, etc.)
  - Can use PAT_RT_EXPFILE_DIR to point to existing directory that resides on a high-performance file system if not execution directory.
- Number of files used to store raw data
  - 1 file created for program with 1 – 256 processes
  - $\sqrt{n}$ files created for program with 257 – n processes
  - Ability to customize with PAT_RT_EXPFILE_MAX
- See intro_craypat(1) man page
CrayPat Runtime Options

- Runtime controlled through PAT_RT_XXX environment variables

- Examples of control
  - Enable full trace
  - Change number of data files created
  - Enable collection of HW counters
  - Enable collection of network counters
  - Enable tracing filters to control trace file size (max threads, max call stack depth, etc.)
Example Runtime Environment Variables

• Optional timeline view of program available
  • export `PAT_RT_SUMMARY=0`
  • View trace file with Cray Apprentice

• Request hardware performance counter information:
  • export `PAT_RT_HWPC=<HWPC Group>`
  • Can specify events or predefined groups
Predefined Trace Wrappers (-g tracegroup)

- blas   Basic Linear Algebra subprograms
- caf    Co-Array Fortran (Cray CCE compiler only)
- hdf5   manages extremely large data collection
- heap   dynamic heap
- io     includes stdio and sysio groups
- lapack Linear Algebra Package
- math   ANSI math
- mpi    MPI
- omp    OpenMP API
- pthreads POSIX threads
- shmем  SHMEM
- sysio  I/O system calls
- system system calls
- upc    Unified Parallel C (Cray CCE compiler only)

For a full list, please see `pat_build(1)` man page
Example Experiments

• > pat_build –O apa
  • Gets you top time consuming routines
  • Lightest-weight sampling

• > pat_build –u –g mpi ./my_program
  • Collects information about user functions and MPI

• > pat_build –w ./my_program
  • Collections information for MAIN
  • Lightest-weight tracing

• > pat_build –gnetcdf,mpi ./my_program
  • Collects information about netcdf routines and MPI
pat_report

- Combines information from binary with raw performance data
- Performs analysis on data
- Generates text report of performance results
- Generates customized instrumentation template for automatic profiling analysis
- Formats data for input into Cray Apprentice$^2$
Automatic Profiling Analysis
Why Should I generate a “.ap2” file?

• The “.ap2” file is a self contained compressed performance file
• Normally it is about 5 times smaller than the “.xf” file
• Contains the information needed from the application binary
  • Can be reused, even if the application binary is no longer available or if it was rebuilt
• It is the only input format accepted by Cray Apprentice²
Program Instrumentation - Automatic Profiling Analysis

- Automatic profiling analysis (APA)
  - Provides simple procedure to instrument and collect performance data for novice users
  - Identifies top time consuming routines
  - Automatically creates instrumentation template customized to application for future in-depth measurement and analysis
Steps to Collecting Performance Data

• Access performance tools software
  
  % module load perftools

• Build application keeping .o files (CCE: -h keepfiles)
  
  % make clean ; make

• Instrument application for automatic profiling analysis
• You should get an instrumented program a.out+pat
  
  % pat_build -O apa a.out

• Run application to get top time consuming routines
Steps to Collecting Performance Data (2)

- You should get a performance file ("<sdatafile>.xf") or multiple files in a directory <sdatadir>

  \% aprun ... a.out+pat  (or  qsub <pat script>)

- Generate report and .apa instrumentation file

  \% pat_report <sdatafile>.xf > sampling_report

  \% pat_report -o sampling_report [ <sdatafile>.xf | <sdatadir> ]

- Inspect .apa file and sampling report
- Verify if additional instrumentation is needed
Generating Profile from APA

• Instrument application for further analysis (a.out+apa)

  % pat_build -O <apafilename>.apa

• Run application

  % aprun ... a.out+apa (or qsub <apa script>)

• Generate text report and visualization file (.ap2)

  % pat_report -o my_text_report.txt [<datafile>.xf | <datadir>]

• View report in text and/or with Cray Apprentice

  % app2 <datafile>.ap2
CPU HW Performance Counters
**PAPI Predefined Events**

- Common set of events deemed relevant and useful for application performance tuning
  - Accesses to the memory hierarchy, cycle and instruction counts, functional units, pipeline status, etc.
  - The “papi_avail” utility shows which predefined events are available on the system – execute on compute node

- PAPI also provides access to native events
  - The “papi_native_avail” utility lists all AMD native events available on the system – execute on compute node

- PAPI uses perf_events Linux subsystem

- Information on PAPI and AMD native events
  - pat_help counters
  - man intro_papi (points to PAPI documentation: [http://icl.cs.utk.edu/papi/](http://icl.cs.utk.edu/papi/))
Hardware Counters Selection

• HW counter collection enabled with PAT_RT_HWPC environment variable

• PAT_RT_HWPC <set number> | <event list>
  • A set number can be used to select a group of predefined hardware counters events (recommended)
    • CrayPat provides 23 groups on the Cray XT/XE systems
    • See pat_help(1) or the hwpc(5) man page for a list of groups
  • Alternatively a list of hardware performance counter event names can be used
  • Hardware counter events are not collected by default
Predefined Interlagos HW Counter Groups

See pat_help -> counters -> amd_fam15h -> groups

0: Summary with instructions metrics
1: Summary with TLB metrics
2: L1 and L2 Metrics
3: Bandwidth information
4: <Unused>
5: Floating operations dispatched
6: Cycles stalled, resources idle
7: Cycles stalled, resources full
8: Instructions and branches
9: Instruction cache
10: Cache Hierarchy (unsupported for IL)
Predefined Interlagos HW Counter Groups (cont’d)

11: Floating point operations dispatched
12: Dual pipe floating point operations dispatched
13: Floating point operations SP
14: Floating point operations DP
19: Prefetchs
20: FP, D1, TLB, MIPS
21: FP, D1, TLB, Stalls
22: D1, TLB, MemBW
23: FP, D1, D2, and TLB
default: group 23

Support for L3 cache counters coming in 3Q2013
New HW counter groups for Interlagos (6 counters)

- Group 20: FP, D1, TLB, MIPS
  - PAPI_FP_OPS
  - PAPI_L1_DCA
  - PAPI_L1_DCM
  - PAPI_TLB_DM
  - DATA_CACHE_REFILLS_FROM_NORTHBRIDGE
  - PAPI_TOT_INS

- Group 21: FP, D1, TLB, Stalls
  - PAPI_FP_OPS
  - PAPI_L1_DCA
  - PAPI_L1_DCM
  - PAPI_TLB_DM
  - DATA_CACHE_REFILLS_FROM_NORTHBRIDGE
  - PAPI_RES_STL
Example: HW counter data & Derived Metrics

- **PAPI_TLB_DM**: Data translation lookaside buffer misses
- **PAPI_L1_DCA**: Level 1 data cache accesses
- **PAPI_FP_OPS**: Floating point operations
- **DC_MISS**: Data Cache Miss
- **User_Cycles**: Virtual Cycles

---

**USER**

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (%)</td>
<td>98.3%</td>
</tr>
<tr>
<td>Time</td>
<td>4.434402 secs</td>
</tr>
<tr>
<td>Imb. Time</td>
<td>-- secs</td>
</tr>
<tr>
<td>Imb. Time (%)</td>
<td>--</td>
</tr>
<tr>
<td>Calls</td>
<td>0.001M/sec 4500.0 calls</td>
</tr>
<tr>
<td>PAPI_L1_DCM</td>
<td>14.820M/sec 65712197 misses</td>
</tr>
<tr>
<td>PAPI_TLB_DM</td>
<td>0.902M/sec 3998928 misses</td>
</tr>
<tr>
<td>PAPI_L1_DCA</td>
<td>333.331M/sec 147796162 refs</td>
</tr>
<tr>
<td>PAPI_FP_OPS</td>
<td>445.571M/sec 1975672594 ops</td>
</tr>
<tr>
<td>User time (approx)</td>
<td>4.434 secs 1197186993 cycles 100.0%Time</td>
</tr>
<tr>
<td>Average Time per Call</td>
<td>0.000985 sec</td>
</tr>
<tr>
<td>CrayPat Overhead : Time</td>
<td>0.1%</td>
</tr>
<tr>
<td>HW FP Ops / User time</td>
<td>445.571M/sec 1975672594 ops 4.1%peak(DP)</td>
</tr>
<tr>
<td>HW FP Ops / WCT</td>
<td>445.533M/sec</td>
</tr>
<tr>
<td>Computational intensity</td>
<td>0.17 ops/cycle 1.34 ops/ref</td>
</tr>
<tr>
<td>MFLOPS (aggregate)</td>
<td>1782.28M/sec</td>
</tr>
<tr>
<td>TLB utilization</td>
<td>369.60 refs/miss 0.722 avg uses</td>
</tr>
<tr>
<td>D1 cache hit,miss ratios</td>
<td>95.6% hits 4.4% misses</td>
</tr>
<tr>
<td>D1 cache utilization (misses)</td>
<td>22.49 refs/miss 2.811 avg hits</td>
</tr>
</tbody>
</table>

---

**PAT_RT_HWPC=1**

Flat profile data
Raw counts
Derived metrics
PAT_RT_HWPC=2 (L1 and L2 Metrics)

========================================================================

USER

------------------------------------------------------------------------

Time%                                             98.3%
Time                                           4.436808 secs
Imb. Time                                             -- secs
Imb. Time%                                             --
Calls                          0.001M/sec        4500.0 calls

DATA_CACHE_REFILLS:
L2_MODIFIED:L2_OWNED:
L2_EXCLUSIVE:L2_SHARED       9.821M/sec      43567825 fills

DATA_CACHE_REFILLS_FROM_SYSTEM:
ALL                         24.743M/sec     109771658 fills
PAPI_L1_DCM                  14.824M/sec      65765949 misses
PAPI_L1_DCA                  332.960M/sec    1477145402 refs

User time (approx)             4.436 secs   11978286133 cycles 100.0% Time
Average Time per Call                          0.000986 sec

CrayPat Overhead : Time         0.1%

D1 cache hit,miss ratios        95.5% hits    4.5% misses
D1 cache utilization (misses)    22.46 refs/miss  2.808 avg hits
D1 cache utilization (refills)   9.63 refs/refill  1.204 avg uses
D2 cache hit,miss ratio        28.4% hits    71.6% misses
D1+D2 cache hit,miss ratio      96.8% hits    3.2% misses
D1+D2 cache utilization        31.38 refs/miss   3.922 avg hits

System to D1 refill             24.743M/sec     109771658 lines
System to D1 bandwidth         1510.217MB/sec  7025386144 bytes
D2 to D1 bandwidth             599.398MB/sec   2788340816 bytes

========================================================================
Profile Visualization with pat_report
CrayPat/X: Version 5.2.3.8078 Revision 8078 (xf 8063) 08/25/11 ...

Number of PEs (MPI ranks): 16

Numbers of PEs per Node: 16

Numbers of Threads per PE: 1

Number of Cores per Socket: 12

Execution start time: Thu Aug 25 14:16:51 2011

System type and speed: x86_64 2000 MHz

Current path to data file:
/lus/scratch/heidi/ted_swim/mpi-openmp/run/swim+pat+27472-34t.ap2

Notes for table 1:
...

pat_report: Job Execution Information
Notes for table 1:

Table option:
-0 profile

Options implied by table option:
-d ti%@0.95,ti,imb_ti,imb_ti%,tr -b gr,fu,pe=HIDE

Other options:
-T

Options for related tables:
-0 profile_pe.th       -0 profile_th_pe
-0 profile+src        -0 load_balance
-0 callers            -0 callers+src
-0 calltree           -0 calltree+src

The Total value for Time, Calls is the sum for the Group values.
The Group value for Time, Calls is the sum for the Function values.
The Function value for Time, Calls is the avg for the PE values.
(To specify different aggregations, see: pat_help report options s1)

This table shows only lines with Time% > 0.

Percentages at each level are of the Total for the program.
(For percentages relative to next level up, specify:
-0 s percent=r[elative])
pat_report: Additional Information

Instrumented with:
pat_build -gmpi -u himenoBMTxpr.x

Program invocation:
../bin/himenoBMTxpr+pat.x

Exit Status:  0 for 256 PEs

CPU Family: 15h  Model: 01h  Stepping: 2
Core Performance Boost: Configured for 0 PEs
Capable for 256 PEs

Memory pagesize:  4096

Accelerator Model: Nvidia X2090 Memory: 6.00 GB Frequency: 1.15 GHz

Programming environment:  CRAY

Runtime environment variables:
OMP_NUM_THREADS=1
Sampling Output (Table 1)

Table 1: Profile by Function

<table>
<thead>
<tr>
<th>Samp %</th>
<th>Samp</th>
<th>Imb. Samp</th>
<th>Imb. Samp</th>
<th>Group Function</th>
<th>PE='HIDE'</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0%</td>
<td>775</td>
<td>--</td>
<td>--</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>94.2%</td>
<td>730</td>
<td>--</td>
<td>--</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th>mlwxyz_</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>43.4%</td>
<td>336</td>
<td>8.75</td>
<td>2.6%</td>
<td>half_</td>
<td></td>
</tr>
<tr>
<td>16.1%</td>
<td>125</td>
<td>6.28</td>
<td>4.9%</td>
<td>full_</td>
<td></td>
</tr>
<tr>
<td>8.0%</td>
<td>62</td>
<td>6.25</td>
<td>9.5%</td>
<td>artv_</td>
<td></td>
</tr>
<tr>
<td>6.8%</td>
<td>53</td>
<td>1.88</td>
<td>3.5%</td>
<td>bnd_</td>
<td></td>
</tr>
<tr>
<td>4.9%</td>
<td>38</td>
<td>1.34</td>
<td>3.6%</td>
<td>currentf_</td>
<td></td>
</tr>
<tr>
<td>3.6%</td>
<td>28</td>
<td>2.00</td>
<td>6.9%</td>
<td>bndsf_</td>
<td></td>
</tr>
<tr>
<td>2.2%</td>
<td>17</td>
<td>1.50</td>
<td>8.6%</td>
<td>model_</td>
<td></td>
</tr>
<tr>
<td>1.7%</td>
<td>13</td>
<td>1.97</td>
<td>13.5%</td>
<td>cfl_</td>
<td></td>
</tr>
<tr>
<td>1.4%</td>
<td>11</td>
<td>1.53</td>
<td>12.2%</td>
<td>currenth_</td>
<td></td>
</tr>
<tr>
<td>1.3%</td>
<td>10</td>
<td>0.75</td>
<td>7.0%</td>
<td>bndbo_</td>
<td></td>
</tr>
<tr>
<td>1.0%</td>
<td>8</td>
<td>5.28</td>
<td>41.9%</td>
<td>bndto_</td>
<td></td>
</tr>
<tr>
<td>1.0%</td>
<td>8</td>
<td>8.28</td>
<td>53.4%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th>mpi_sendrecv_</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.9%</td>
<td>15</td>
<td>4.62</td>
<td>23.9%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.8%</td>
<td>14</td>
<td>16.53</td>
<td>55.0%</td>
<td>mpi_bcast_</td>
<td></td>
</tr>
<tr>
<td>1.7%</td>
<td>13</td>
<td>5.66</td>
<td>30.7%</td>
<td>mpi_barrier_</td>
<td></td>
</tr>
</tbody>
</table>
### Table 1: Profile by Function Group and Function

<table>
<thead>
<tr>
<th>Time %</th>
<th>Time</th>
<th>Imb. Time</th>
<th>Imb. Time %</th>
<th>Calls</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0%</td>
<td>104.593634</td>
<td>--</td>
<td>--</td>
<td>22649</td>
<td>Total</td>
</tr>
<tr>
<td>71.0%</td>
<td>74.230520</td>
<td>--</td>
<td>--</td>
<td>10473</td>
<td>MPI</td>
</tr>
<tr>
<td>69.7%</td>
<td>72.905208</td>
<td>0.508369</td>
<td>0.7%</td>
<td>125</td>
<td>mpi_allreduce_</td>
</tr>
<tr>
<td>1.0%</td>
<td>1.050931</td>
<td>0.030042</td>
<td>2.8%</td>
<td>94</td>
<td>mpi_alltoall_</td>
</tr>
<tr>
<td>25.3%</td>
<td>26.514029</td>
<td>--</td>
<td>--</td>
<td>73</td>
<td>USER</td>
</tr>
<tr>
<td>16.7%</td>
<td>17.461110</td>
<td>0.329532</td>
<td>1.9%</td>
<td>23</td>
<td>selfgravity_</td>
</tr>
<tr>
<td>7.7%</td>
<td>8.078474</td>
<td>0.114913</td>
<td>1.4%</td>
<td>48</td>
<td>fft_e4_</td>
</tr>
<tr>
<td>2.5%</td>
<td>2.659429</td>
<td>--</td>
<td>--</td>
<td>435</td>
<td>MPI_SYNC</td>
</tr>
<tr>
<td>2.1%</td>
<td>2.207467</td>
<td>0.768347</td>
<td>26.2%</td>
<td>172</td>
<td>mpi_barrier_(sync)</td>
</tr>
<tr>
<td>1.1%</td>
<td>1.188998</td>
<td>--</td>
<td>--</td>
<td>11608</td>
<td>HEAP</td>
</tr>
<tr>
<td>1.1%</td>
<td>1.166707</td>
<td>0.142473</td>
<td>11.1%</td>
<td>5235</td>
<td>free</td>
</tr>
</tbody>
</table>
### Table 4: MPI Message Stats by Caller

<table>
<thead>
<tr>
<th>MPI Msg</th>
<th>MPI Msg</th>
<th>MsgSz</th>
<th>4KB&lt;=&lt;16B</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bytes</td>
<td>Count</td>
<td>&lt;64KB</td>
<td>PE[mmm]</td>
<td>Caller</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Count</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

15138076.0 | 4099.4 | 411.6 | 3687.8 | Total

---

15138028.0 | 4093.4 | 405.6 | 3687.8 | MPI_SEND

---

8080500.0 | 2062.5 | 93.8  | 1968.8 | calc2_

3

---

8216000.0 | 3000.0 | 1000.0 | 2000.0 | pe.0
4

8208000.0 | 2000.0 | --     | 2000.0 | pe.9
4

6160000.0 | 2000.0 | 500.0  | 1500.0 | pe.15
4

---

6285250.0 | 1656.2 | 125.0  | 1531.2 | calc1_

3

---

8216000.0 | 3000.0 | 1000.0 | 2000.0 | pe.0
4

6156000.0 | 1500.0 | --     | 1500.0 | pe.3
4

6156000.0 | 1500.0 | --     | 1500.0 | pe.5
4

---
Profile Visualization with Cray Apprentice2
Cray Apprentice helps identify:

- Load imbalance
- Excessive communication
- Network contention
- Excessive serialization
- I/O Problems

- Call graph profile
- Communication statistics
- Time-line view
  - Communication
  - I/O
- Activity view
- Pair-wise communication statistics
- Text reports
- Source code mapping

- Runs on login node
- Supported on Mac OS and Windows also
Application Performance Summary

Profile

CPU
- Computation: 61.6%
- Programming Model: 38.4%
- IO: 0.0%

Function/Region Profile
- 61.3% = main
- 19.1% = _pgas_get_nbi
- 14.0% = _pgas_put_nbi

Load Imbalance
- 0.12s = main
- 0.34s = _pgas_get_nbi
- 0.18s = _pgas_put_nbi

Memory Utilization
- L1 cache hit ratio: 94.3%
- Process HMem (MB/sec): 1683
- LD + ST per TLE miss: 166530, 95 ref/miss

Data Movement
- No data collected.
- Run per_bautil etc

Wallclock time: 24.978773s
Statistics Overview

Switch Overview display
Load Balance View (Aggregated from Overview)

Min, Avg, and Max Values

-1, +1 Std Dev marks
pat_report Tables in Cray Apprentice2

• Complimentary performance data available in one place

• Drop down menu provides quick access to most common reports

• Ability to easily generate different views of performance data

• Provides mechanism for more in depth explanation of data presented
Example of `pat_report` Tables in Cray Apprentice2

New text table icon

Right click for table generation options
Generating New pat_report Tables

- Profile
- Custom...
- Source
- Calltree
- Callers
- Show Notes
- Show All PE's
- Show HWPC
- Use Thresholds

Panel Actions
Panel Help
Load balance overview:
Height ⇔ Max time
Middle bar ⇔ Average time
Lower bar ⇔ Min time

Yellow represents imbalance time

Width ⇔ inclusive time
Height ⇔ exclusive time

Filtered nodes or sub tree

DUH Button: Provides hints for performance tuning

Zoom

Function List
Discrete Unit of Help (DUH Button)
Load Balance View (from Call Tree)

Min, Avg, and Max Values

-1, +1 Std Dev marks
Source Mapping from Call Tree

```c
165  \* angle pipelining loop (batches of mmi angles)
166  \*
167      DO mg = 1, mmo
168          mli = (mg-1)*mmi
169  \*
170  \* K-inflows (k=k0 boundary)
171  \*
172      IF (k2.lt.0 .OR. kbc.eq.0) THEN
173          DO ml = 1, mmi
174              j = 1, jtt
175              DO i = 1, itt
176                  phikb(i,j,ml) = 0.0d0
177              END DO
178          END DO
179  \* else if (do,dsa) then
180      leakage = 0.0
181          k = k0 - k2
182          DO ml = 1, mmi
183              n = ml + mio
184              j = 1, jtt
185              DO i = 1, itt
186                  phikb(i,j,ml) = phikbc(i,j,m)
187                  leak = leak +
188                  & wtsi(m)*phikb(i,j,ml)*di(i)*dj(j)
189                  & face(i,j,k+k3,3) = face(i,j,k+k3,3)
190                  & wtsi(m)*phikb(i,j,ml)
191          END DO
192      ELSE
193          leakage = leakage + leak
194      END ELSE
195  \*
```

0.00  2.10  4.27  6.40  8.53
Full Trace Visualization with Cray Apprentice2
Trace Overview – Additional Views

- HW counters overview (counter histogram by function)
- HW counters plot (counters in timeline)
- Mosaic (shows communication pattern)
- Traffic report (MPI timeline)

Activity report (Synchronization, data movement, etc. over time)
Mosaic View – Shows Communication Pattern
HW Counters Overview

10.170 DATA CACHE REFILLS FROM NORTHBRIDGE
- MPI_Recv
- MPI_Send
- MPI_BARRIER(sync)
- Others

9.319M DATA_CACHE_REFILLS_FROM_L2_OR_NORTHBRIDGE
- ALL
  - MPI_Recv
  - MPI_BARRIER(sync)
  - MPI_Send
  - MPI_Wait
  - Others

9.276M PAPI_L1_DCM
- MPI_Recv
- MPI_BARRIER(sync)
- MPI_Send
- MPI_Wait
- Others

3.078 PAPI_TLB_DM
- MPI_Recv
- MPI_Send
- MPI_BARRIER(sync)
- Others

2.628G PAPI_L1_DCA
- MPI_Recv
- MPI_BARRIER(sync)
- MPI_Send
- Others

4 PAPI_FP_OPS
- MPI_Wtime

4.882G CYCLES_RTC
- MPI_Recv
- MPI_BARRIER(sync)
- MPI_Send
- Others

13.998M LS_L1D_INV_TLB_DM
- MPI_Recv
- MPI_BARRIER(sync)
- MPI_Wait
Traffic Report – MPI Communication Timeline
Man pages

• intro_craypat(1)
  Introduces the craypat performance tool

• pat_build(1)
  Instrument a program for performance analysis

• pat_help(1)
  Interactive online help utility

• pat_report(1)
  Generate performance report in both text and for use with GUI

• app2 (1)
  Describes how to launch Cray Apprentice2 to visualize performance data
Man pages (2)

- **hwpc(5)**
  - describes predefined hardware performance counter groups

- **nwpc(5)**
  - Describes predefined network performance counter groups

- **accpc(5) / accpc_k20(5)**
  - Describes predefined GPU performance counter groups

- **intro_papi(3)**
  - Lists PAPI event counters
  - Use `papi_avail` or `papi_native_avail` utilities to get list of events when running on a specific architecture
MPI Rank Order
MPI Rank Order

Is your nearest neighbor really your nearest neighbor?

And do you want them to be your nearest neighbor?
MPI Rank Placement

- Change default rank ordering with:
  - MPICH_RANK_REORDER_METHOD

- Settings:
  - 0: **Round-robin** placement – Sequential ranks are placed on the next node in the list. Placement starts over with the first node upon reaching the end of the list.
  - 1: **SMP-style** placement – Sequential ranks fill up each node before moving to the next. - DEFAULT
  - 2: **Folded** rank placement – Similar to round-robin placement except that each pass over the node list is in the opposite direction of the previous pass.
  - 3: **Custom ordering** - The ordering is specified in a file named MPICH_RANK_ORDER.
When Is Rank Re-ordering Useful?

- Maximize on-node communication between MPI ranks

- Grid detection and rank re-ordering is helpful for programs with significant point-to-point communication

- Relieve on-node shared resource contention by pairing threads or processes that perform different work (for example computation with off-node communication) on the same node
Automatic Communication Grid Detection

• Cray performance tools produce a custom rank order if it’s beneficial based on grid size, grid order and cost metric

• Heuristics available for:
  • MPI sent message statistics
  • User time (time spent in user functions) – can be used for PGAS codes
  • Hybrid of sent message and user time

• Summarized findings in report

• Available with sampling or tracing

• Describe how to re-run with custom rank order
### Table 1: Profile by Function Group and Function

<table>
<thead>
<tr>
<th>Time%</th>
<th>Time</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Calls</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100.0%</td>
<td>463.147240</td>
<td>--</td>
<td>--</td>
<td>21621.0</td>
<td>Total</td>
</tr>
<tr>
<td>52.0%</td>
<td>240.974379</td>
<td>--</td>
<td>--</td>
<td>21523.0</td>
<td>MPI</td>
</tr>
<tr>
<td>47.7%</td>
<td>221.142266</td>
<td>36.214468</td>
<td>14.1%</td>
<td>10740.0</td>
<td>mpi_recv</td>
</tr>
<tr>
<td>4.3%</td>
<td>19.829001</td>
<td>25.849906</td>
<td>56.7%</td>
<td>10740.0</td>
<td>MPI_SEND</td>
</tr>
<tr>
<td>43.3%</td>
<td>200.474690</td>
<td>--</td>
<td>--</td>
<td>32.0</td>
<td>USER</td>
</tr>
<tr>
<td>41.0%</td>
<td>189.897060</td>
<td>58.716197</td>
<td>23.6%</td>
<td>12.0</td>
<td>sweep_</td>
</tr>
<tr>
<td>1.6%</td>
<td>7.579876</td>
<td>1.899097</td>
<td>20.1%</td>
<td>12.0</td>
<td>source_</td>
</tr>
<tr>
<td>4.7%</td>
<td>21.698147</td>
<td>--</td>
<td>--</td>
<td>39.0</td>
<td>MPI_SYNC</td>
</tr>
<tr>
<td>4.3%</td>
<td>20.091165</td>
<td>20.005424</td>
<td>99.6%</td>
<td>32.0</td>
<td>mpi_allreduce_(sync)</td>
</tr>
<tr>
<td>0.0%</td>
<td>0.000024</td>
<td>--</td>
<td>--</td>
<td>27.0</td>
<td>SYSCALL</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>------</td>
<td>------</td>
<td>-------</td>
<td>-------</td>
</tr>
</tbody>
</table>
MPI Grid Detection:

There appears to be point-to-point MPI communication in a 96 X 8 grid pattern. The 52% of the total execution time spent in MPI functions might be reduced with a rank order that maximizes communication between ranks on the same node. The effect of several rank orders is estimated below.

A file named MPICH_RANK_ORDER.Grid was generated along with this report and contains usage instructions and the Custom rank order from the following table.

<table>
<thead>
<tr>
<th>Rank</th>
<th>On-Node Bytes/PE</th>
<th>Bytes/PE% of Total</th>
<th>MPICH_RANK_REORDER_METHOD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom</td>
<td>2.385e+09</td>
<td>95.55%</td>
<td>3</td>
</tr>
<tr>
<td>SMP</td>
<td>1.880e+09</td>
<td>75.30%</td>
<td>1</td>
</tr>
<tr>
<td>Fold</td>
<td>1.373e+06</td>
<td>0.06%</td>
<td>2</td>
</tr>
<tr>
<td>RoundRobin</td>
<td>0.000e+00</td>
<td>0.00%</td>
<td>0</td>
</tr>
</tbody>
</table>
MPICH_RANK_ORDER File

# The 'Custom' rank order in this file targets nodes with multi-core
# processors, based on Sent Msg Total Bytes collected for:
#
# Program:   /lus/nid00030/heidi/sweep3d/mod/sweep3d.mpi
# Ap2 File:  sweep3d.mpi+pat+27054-89t.ap2
# Number PEs: 48
# Max PEs/Node: 4
#
# To use this file, make a copy named MPICH_RANK_ORDER, and set the
# environment variable MPICH_RANK_REORDER_METHOD to 3 prior to
# executing the program.
#
# The following table lists rank order alternatives and the grid_order
# command-line options that can be used to generate a new order.
...

### Auto-Generated MPI Rank Order File

To use this file, make a copy named MPICH_RANK_ORDER, and set the environment variable `MPICH_RANK_REORDER_MET HOD` to 3 prior to executing the program.

| # The 'USER_Time_hybrid' rank order in this file targets nodes with multi-core |
| # processors, based on Sent order in this file targets nodes with |
| Total Bytes collected for: |
| Program: |
| /lus/nil00023/malice/craypat/WOR |
| KSHOP/bh2o |
| demo/Rank/sweep3d/src/sweep3d | |
| # Ap2 File: sweep3d.gmpi |
| # Number PEs: 768 |
| # Max PEs/Node: 16 |

<table>
<thead>
<tr>
<th>#</th>
<th>Bytes</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>175,363,159,323,143,355,255,291</td>
<td>155,550,171,518,219,582,147,61</td>
<td>646,298,750,322,718,354,758,290</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>129,563,193,531,161,571,225,539</td>
<td>646,701,737,652,705,668,745,692</td>
<td>734,662,686,670,726,702,694,65</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>177,515,145,579,209,547,217,61</td>
<td>682,288,680,624,720,512,696,632</td>
<td>765,661,709,663,741,653,711,669</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>113,397,641,55,429,87,421,23</td>
<td>728,584,680,624,720,512,696,632</td>
<td>677,727,751,693,647,701,717,687</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>7,405,71,146,93,437,103,413,47,149,422,213,454,181,494,221,48</td>
<td>705,526,736,536,704,560,744,520</td>
<td>326,303,327,367,366,335,302,33</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Additional data is available in the full file.
grid_order Utility

• Can use grid_order utility without first running the application with the Cray performance tools if you know a program’s data movement pattern

• Originally designed for MPI programs, but since reordering is done by PMI, it can be used by other programming models (since PMI is used by MPI, SHMEM and PGAS programming models)

• Utility available if perftools modulefile is loaded

• See grid_order(1) man page or run grid_order with no arguments to see usage information
Reorder Example for Bisection Bandwidth

• Assume 32 ranks

• Decide on row or column ordering:

  • $\textit{grid\_order} \ -R \ -g \ 2,16$

    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

  • $\textit{grid\_order} \ -C \ -g \ 2,16$

    0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30
    1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31

• Since rank 0 talks to rank 16, and not with rank 1, we choose Row ordering
Reorder Example for Bisection Bandwidth (2)

• Specify cell (or chunk) to make sure rank pairs live on same node (but don’t care how many pairs live on a node)

• $\text{grid\_order -R \_g 2,16 \_c 2,1}$

0,16
1,17
2,18
3,19
4,20
5,21
6,22
7,23
8,24
9,25
10,26
11,27
12,28
13,29
14,30
15,31

Fills a Magny-Cours node
Using New Rank Order

• Save grid_order output to file called MPICH_RANK_ORDER

• Export MPICH_RANK_REORDER_METHOD=3

• Run non-instrumented binary with and without new rank order to check overall wallclock time for improvement
Example Performance Results

- Default thread ordering
  - Application 8538980 resources: utime ~126s, stime ~108s

- Maximized on-node data movement with reordering
  - Application 8538982 resources: utime ~38s, stime ~106s
Thank You
Abnormal Termination Processing (ATP)

- Useful when a user wants to know where his/her application crashes (i.e. stack backtrace when crashing).
- ATP gathers all of the stack backtraces into a merged stack backtrace tree and (1) output to job stderr output file, (2) writes to disk as the file "atpMergedBT.dot".
- To use ATP:
  - At compile time:
    - module load atp  # this is default
    - Compile code with "-g"
  - In job script, before aprun command, add:
    - export ATP_ENABLED=1
ATP (cont.)

- atpMergedBT.dot can be viewed with statview GUI tool
Stack Trace Analysis Tool on Blue Waters

Craig Steffen   SEAS group
csteffen@ncsa.illinois.edu
How to Use Stack Trace Analysis Tool (STAT)

• Find the MOM node where your job is running
• Ssh to that MOM node
  • > ssh nidXXXX
• > module load stat
• > cd /scratch/my/job/dir/
• > mkdir stat_info_$PBS_JOBID
• > cd stat_info_$PBS_JOBID
How to use STAT (continued)

• > ps -aux | grep my_login_name
  • [ find the pid of your aprun command ]
• > STAT 123456
Attaching to job launcher and launching tool daemons...

....
Results written to
/scratch/my/job/dir/stat_info_222333/...
To Visualize STAT data:

- Log onto login node with X forwarding
- > module load stat
- > cd [to where data is]
- STATview XXXYYY.dot
Usage information for STAT (after “module load stat”)

- csteffen@h2ologin2 23:33 ~ $ man STAT
- Man: find all matching manual pages
- * STAT (1)
- stat (1+)
- stat (2)
- Man: What manual page do you want?
- Man:
- csteffen@h2ologin2 23:33 ~ $
How to find MOM node for your job:  
Have the job script “phone home”:

echo 'about to run solver'

touch running_on_host_`hostname`'

date

export PAT_RT_HWPC="PAPI_FP_OPS,PAPI_TOT_INS"
aprun -n 1536 -N 32 -cc 0-7,8-15,16-23,24-31 ./$solver_exec_name
Start job and wait for it to run

- csteffen@h2ologin2 00:04 ~/specfem3d/SF3DG_csteffen $ qsub run_1536.sh
- 364139.nid00221
- csteffen@h2ologin2 00:04 ~/specfem3d/SF3DG_csteffen $ qstat | grep csteffen
- 364132.nid00221 specfem3d_globe csteffen 00:00:02 C batch
- 364139.nid00221 specfem3d_globe csteffen 0 R batch
SSH to the running MOM node

csteffen@h2ologin2 00:05 ~/specfem3d/SF3DG_csteffen $ ls -l rt running_on*
-rw------- 1 csteffen bw_staff 0 May 19 23:53 running_on_host_nid23054
-rw------- 1 csteffen bw_staff 0 May 20 00:05 running_on_host_nid25261
csteffen@h2ologin2 00:05 ~/specfem3d/SF3DG_csteffen $ ssh nid25261
On MOM node, find PID of my aprun

csteffen@nid25261 00:06 ~ $ ps -aux | grep csteffen | grep aprun
Warning: bad ps syntax, perhaps a bogus '‐'? See
http://procps.sf.net/faq.html

csteffen 23786 0.0 0.0 22316 4360 ? S 00:05 0:00
   /usr/bin/perl /sw/xe/altd/bin/aprun -n 1536 -N 32 -cc 0-7,8-15,16-23,24-31 ./xspecfem3D

csteffen 23807 0.0 0.0 28804 2168 ? S 00:05 0:00
   /usr/bin/aprun -n 1536 -N 32 -cc 0-7,8-15,16-23,24-31
      ./xspecfem3D

csteffen 23957 0.0 0.0 5624 864 pts/0 S+ 00:06 0:00
grep aprun

csteffen@nid25261 00:06 ~ $ module load stat

csteffen@nid25261 00:06 ~ $ STAT 23807
Attaching to job launcher (null):23807 and launching tool daemons...
Trace files available for later analysis

Results written to /mnt/a/u/staff/csteffen/stat_results/xspecfem3D.0000

csteffen@h2ologin2 00:10 ~/stat_results/xspecfem3D.0000 $ ls -lrt
total 124
-rw-r--r-- 1 csteffen bw_staff 12 May 20 00:07 xspecfem3D.0000.top
-rw-r--r-- 1 csteffen bw_staff 48057 May 20 00:07 xspecfem3D.0000.ptab
-rw-r--r-- 1 csteffen bw_staff 634 May 20 00:07 xspecfem3D.0000.fulltop
-rw-r--r-- 1 csteffen bw_staff 1265 May 20 00:07 xspecfem3D.0000.perf
-rw-r--r-- 1 csteffen bw_staff 64140 May 20 00:07 xspecfem3D.0000.3D.dot

csteffen@h2ologin2 00:10 ~/stat_results/xspecfem3D.0000 $
csteffen@h2ologin2 00:13 ~/stat_results/xspecfem3D.0000 $ module load stat
csteffen@h2ologin2 00:13 ~/stat_results/xspecfem3D.0000 $ STATview
  xspecfem3D.0000.3D.dot
STATview
STATview displays call trees and occupancies
Distributed Debugging Tool (DDT) on Blue Waters

Craig Steffen  SEAS group
csteffen@ncsa.illinois.edu
Why DDT?

- Complete Graphical Debugger
  - Traps data values
  - Across-execution visualizers
  - Drops user into source automatically
  - Synchronization and deadlock checking
- Only requires symbols (-g) to work
- Very parallel (hundreds of thousands of ranks)
- Has useful annotations for optimized-out source
HowTo 0:

> Module load ddt
> ddt
HowTo 1: click on “Run and Debug Program”

- DDT understands qsub and job submission
- It launches job and executable
- DDT will start a debug session automatically as soon as the job starts.
HowTo 2: Manual Launch

• To launch a program manually click on *Manually Launch a Program* button.

• Select how many processes you want to debug and click on *Listen*. At this point start a program or programs using the following command:
  
  • > ddt-client <path-to-program-binary>

• Debugging begins when all executables are running

• Used for applications with multiple, separate, communicating executables
HowTo 3: Attach to Running Executable

• click on Attach to a Running Program button.
• DDT will start scanning each of 64 mom nodes to locate active jobs owned by you. If there are more than one active job DDT will find all of them. Once DDT finds the desired job select it and click on Attach to listed processes button.
HowTo 4: Start DDT as an interactive job

- > qsub -l –X [interactive job with X forwarding]
- > module load ddt
- > ddt --noqueue
- Click on Run and Debug a Program. A new window with expandable tabs will appear.

- Click on Run button to start a debug session.
DDT: debugging displays

- Points out faults in crashes
- Memory debugging tool detects leaks and out-of-bounds
- “sparklines” graphical data value summary across all ranks
- Good source code navigator and controls
Congestion Protection and Balanced Injection
What is Congestion Protection?

- Network congestion is a condition that occurs when the volume of traffic on the high-speed network (HSN) exceeds the capacity to handle it.
- To "protect" the network from data loss, congestion protection (CP) globally “throttles” injection bandwidth per-node.
- If CP happens often, application performance degrades.

At job completion you might see the following message reported to stdout:

Application 61435 network throttled: 4459 nodes throttled, 25:31:21 node-seconds
Application 61435 balanced injection 100, after throttle 63

The throttling event lasts for 20 seconds each time CP is triggered.
Types of congestion events

- There are two main forms of congestion: many-to-one and long-path. The former is easy to detect and correct. The latter is harder to detect and may not be correctable.
- Many-to-one congestion occurs in some algorithms and can be corrected. uGNI and DMAPP based codes doing All-to-one operations are common case. See “Modifying Your Application to Avoid Gemini Network Congestion Errors” on balanced injection section on the portal.
- Long-path congestion is typically due to a combination of communication pattern and node allocation. It can also be due to a combination of jobs running on the system.
- We monitor for cases of congestion protection and try to determine the most likely cause.
Balanced Injection

- Balanced Injection (BI) is a mechanism that attempts to reduce compute node injection bandwidth in order to prevent throttling and which may have the effect of improving application performance for certain communication patterns.
- BI can be applied “per-job” using an environment variable or with user accessible API.
- `export APRUN_BALANCED_INJECTION=64`
- Can be set from 1-100 (100 = no BI).
- There isn’t a linear relation of BI to application performance.
- MPI-based applications have “balanced injection” enabled in collective MPI calls that locally “throttle” injection bandwidth.