Performance Tools on Blue Waters

Manisha Gajbe + others
Performance Tools

- Cray Performance Tools
- Perfsuite (NCSA)
- TAU
Topics

- Cray performance tools overview
- Steps to using the tools
- Performance measurement on the Cray XE system
- Using HW performance counters
  - Access to Northbridge: L3 and memory
- Profiling applications
- Visualization of performance data through pat_report
- Visualization of performance data through Cray Apprentice2
- MPI Rank Order
- PerfSuite (NCSA)
- TAU : Tuning and Analysis Utilities
- Congestion Protection and Balanced Injection
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
    o 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
  o Indication of causes of problems
  o 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:
  o 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
**Sampling with Line Number information**

Table 2: Profile by Group, Function, and Line

<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></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 a file system that supports record locking, such as Lustre (/mnt/sn3x3/..., /lus/..., /scratch/..., etc.).
  - Can use PAT_RT_EXPFILE_DIR to point to an existing directory that resides on a high-performance file system if not the 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 (Collect data in detail than aggregate)
  • View trace file with Cray Apprentice²

• Request hardware performance counter information:
  • export PAT_RT_HWPC=<HWPC Group>
  • export PAT_RT_ACCPC=<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
- shmem 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 –g netcdf,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)
  o Provides simple procedure to instrument and collect performance data for novice users
  o Identifies top time consuming routines
  o 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
  o You should get an instrumented program a.out+pat

  % pat_build -O apa a.out

• Run application to get top time consuming routines
  
  % aprun ... a.out+pat (or qsub <pat script>)
Steps to Collecting Performance Data (2)

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

- 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 Apprentice²

  % 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/)
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, TLB
default: group 23

Support for L3 cache counters now available.
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`
AMD North-Bridge events

**L3_CACHE_MISSES:** type: core
- READ_BLOCK_EXCLUSIVE, READ_BLOCK_SHARED, READ_BLOCK_MODIFY, PREFETCH, ALL
- CORE_0, CORE_1 ... CORE_7, ALL_CORES

**DRAM_ACCESSSES:** type
- DCT0_PAGE_HIT, DCT0_PAGE_MISS, DCT0_PAGE_CONFLICT
- same for DCT1, ALL

To see all NB events (start with craynb:::)

> aprun papi_native_avail
## Example: HW counter data & Derived Metrics

<table>
<thead>
<tr>
<th>Counter Type</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAPI_TLB_DM</td>
<td>Data translation lookaside buffer misses</td>
<td>0.902M/sec, 3998928 misses</td>
</tr>
<tr>
<td>PAPI_L1_DCA</td>
<td>Level 1 data cache accesses</td>
<td>14.820M/sec, 65712197 misses</td>
</tr>
<tr>
<td>PAPI_FP_OPS</td>
<td>Floating point operations</td>
<td>333.331M/sec, 1477996162 refs</td>
</tr>
<tr>
<td>DC_MISS</td>
<td>Data Cache Miss</td>
<td>445.571M/sec, 1975672594 ops</td>
</tr>
<tr>
<td>User_Cycles</td>
<td>Virtual Cycles</td>
<td>4.43402 secs, 11971868993 cycles 100.0% Time</td>
</tr>
</tbody>
</table>

### Derived Metrics

- **Time%**: 98.3%
- **Imb. Time**: -- secs
- **Calls**: 0.001M/sec, 4500.0 calls
- **PAPI_L1_DCM**: 14.820M/sec, 65712197 misses
- **PAPI_TLB_DM**: 0.902M/sec, 3998928 misses
- **PAPI_L1_DCA**: 333.331M/sec, 1477996162 refs
- **PAPI_FP_OPS**: 445.571M/sec, 1975672594 ops
- **User time (approx)**: 4.434 secs, 11971868993 cycles, 100.0% Time
- **Average Time per Call**: 0.000985 sec
- **CrayPat Overhead : Time**: 0.1%
- **HW FP Ops / User time**: 445.571M/sec, 1975672594 ops, 4.1% peak (DP)
- **Computational intensity**: 0.17 ops/cycle, 1.34 ops/ref
- **MFLOPS (aggregate)**: 1782.28M/sec
- **TLB utilization**: 369.60 refs/miss, 0.722 avg uses
- **D1 cache hit,miss ratios**: 95.6% hits, 4.4% misses
- **D1 cache utilization (misses)**: 22.49 refs/miss, 2.811 avg hits

### Notes

- **PAT_RT_HWPC=1**
- Flat profile data
- Raw counts
- Derived metrics
<table>
<thead>
<tr>
<th>Time%</th>
<th>98.3%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>4.436808 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>
</tbody>
</table>

**DATA_CACHE_REFILLS:**

<table>
<thead>
<tr>
<th>L2_MODIFIED:L2_OWNED:</th>
<th>9.821M/sec 43567825 fills</th>
</tr>
</thead>
<tbody>
<tr>
<td>L2_EXCLUSIVE:L2_SHARED</td>
<td>9.821M/sec 43567825 fills</td>
</tr>
</tbody>
</table>

**DATA_CACHE_REFILLS_FROM_SYSTEM:**

<table>
<thead>
<tr>
<th>ALL</th>
<th>24.743M/sec 109771658 fills</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAPI_L1_DCM</td>
<td>14.824M/sec 65765949 misses</td>
</tr>
<tr>
<td>PAPI_L1_DCA</td>
<td>332.960M/sec 1477145402 refs</td>
</tr>
<tr>
<td>User time (approx)</td>
<td>4.436 secs 11978286133 cycles 100.0%Time</td>
</tr>
</tbody>
</table>

**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 …

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of PEs (MPI ranks):</td>
<td>16</td>
</tr>
<tr>
<td>Numbers of PEs per Node:</td>
<td>16</td>
</tr>
<tr>
<td>Numbers of Threads per PE:</td>
<td>1</td>
</tr>
<tr>
<td>Number of Cores per Socket:</td>
<td>12</td>
</tr>
<tr>
<td>Execution start time:</td>
<td>Thu Aug 25 14:16:51 2011</td>
</tr>
<tr>
<td>System type and speed:</td>
<td>x86_64 2000 MHz</td>
</tr>
<tr>
<td>Current path to data file:</td>
<td>/lus/scratch/heidi/ted_swim/mpi-openmp/run/swim+pat+27472-34t.ap2</td>
</tr>
</tbody>
</table>

Notes for table 1:
...

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 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
### Table 1: Profile by Function

<table>
<thead>
<tr>
<th>Samp %</th>
<th>Samp</th>
<th>Imb.</th>
<th>Imb.</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0%</td>
<td>775</td>
<td>--</td>
<td>--</td>
<td>Total</td>
</tr>
<tr>
<td>94.2%</td>
<td>730</td>
<td>--</td>
<td>--</td>
<td>USER</td>
</tr>
<tr>
<td>43.4%</td>
<td>336</td>
<td>8.75</td>
<td>2.6%</td>
<td>mlwxyz_</td>
</tr>
<tr>
<td>16.1%</td>
<td>125</td>
<td>6.28</td>
<td>4.9%</td>
<td>half_</td>
</tr>
<tr>
<td>8.0%</td>
<td>62</td>
<td>6.25</td>
<td>9.5%</td>
<td>full_</td>
</tr>
<tr>
<td>6.8%</td>
<td>53</td>
<td>1.88</td>
<td>3.5%</td>
<td>artv_</td>
</tr>
<tr>
<td>4.9%</td>
<td>38</td>
<td>1.34</td>
<td>3.6%</td>
<td>bnd_</td>
</tr>
<tr>
<td>3.6%</td>
<td>28</td>
<td>2.00</td>
<td>6.9%</td>
<td>currenf_</td>
</tr>
<tr>
<td>2.2%</td>
<td>17</td>
<td>1.50</td>
<td>8.6%</td>
<td>bndsf_</td>
</tr>
<tr>
<td>1.7%</td>
<td>13</td>
<td>1.97</td>
<td>13.5%</td>
<td>model_</td>
</tr>
<tr>
<td>1.4%</td>
<td>11</td>
<td>1.53</td>
<td>12.2%</td>
<td>cfl_</td>
</tr>
<tr>
<td>1.3%</td>
<td>10</td>
<td>0.75</td>
<td>7.0%</td>
<td>currenh_</td>
</tr>
<tr>
<td>1.0%</td>
<td>8</td>
<td>5.28</td>
<td>41.9%</td>
<td>bndbo_</td>
</tr>
<tr>
<td>1.0%</td>
<td>8</td>
<td>8.28</td>
<td>53.4%</td>
<td>bndto_</td>
</tr>
<tr>
<td>5.4%</td>
<td>42</td>
<td>--</td>
<td>--</td>
<td>MPI</td>
</tr>
<tr>
<td>1.9%</td>
<td>15</td>
<td>4.62</td>
<td>23.9%</td>
<td>mpi_sendrecv_</td>
</tr>
<tr>
<td>1.8%</td>
<td>14</td>
<td>16.53</td>
<td>55.0%</td>
<td>mpi_bcast_</td>
</tr>
<tr>
<td>1.7%</td>
<td>13</td>
<td>5.66</td>
<td>30.7%</td>
<td>mpi_barrier_</td>
</tr>
</tbody>
</table>

Notes for table 1:

...
### 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>Function</th>
<th>PE='HIDE'</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0%</td>
<td>104.593634</td>
<td>--</td>
<td>--</td>
<td>22649</td>
<td>Total</td>
<td></td>
</tr>
<tr>
<td>71.0%</td>
<td>74.230520</td>
<td>--</td>
<td>--</td>
<td>10473</td>
<td>MPI</td>
<td></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>
<td></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>
<td></td>
</tr>
<tr>
<td>25.3%</td>
<td>26.514029</td>
<td>--</td>
<td>--</td>
<td>73</td>
<td>USER</td>
<td></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>
<td></td>
</tr>
<tr>
<td>7.7%</td>
<td>8.078474</td>
<td>0.114913</td>
<td>1.4%</td>
<td>48</td>
<td>ffte4_</td>
<td></td>
</tr>
<tr>
<td>2.5%</td>
<td>2.659429</td>
<td>--</td>
<td>--</td>
<td>435</td>
<td>MPI_SYNC</td>
<td></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>
<td></td>
</tr>
<tr>
<td>1.1%</td>
<td>1.188998</td>
<td>--</td>
<td>--</td>
<td>11608</td>
<td>HEAP</td>
<td></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>
<td></td>
</tr>
</tbody>
</table>
Table 4: MPI Message Stats by Caller

<table>
<thead>
<tr>
<th>MPI(Msg) Bytes</th>
<th>MPI(Msg) Count</th>
<th>MsgSz &lt;16B Count</th>
<th>4KB&lt;=MsgSz Count</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>15138076.0</td>
<td>4099.4</td>
<td>411.6</td>
<td>3687.8</td>
<td>Total</td>
</tr>
<tr>
<td>15138028.0</td>
<td>4093.4</td>
<td>405.6</td>
<td>3687.8</td>
<td>MPI_ISEND</td>
</tr>
<tr>
<td>8080500.0</td>
<td>2062.5</td>
<td>93.8</td>
<td>1968.8</td>
<td>calc2_3</td>
</tr>
<tr>
<td>8216000.0</td>
<td>3000.0</td>
<td>1000.0</td>
<td>2000.0</td>
<td>pe.0</td>
</tr>
<tr>
<td>8208000.0</td>
<td>2000.0</td>
<td>--</td>
<td>2000.0</td>
<td>pe.9</td>
</tr>
<tr>
<td>6160000.0</td>
<td>2000.0</td>
<td>500.0</td>
<td>1500.0</td>
<td>pe.15</td>
</tr>
<tr>
<td>6285250.0</td>
<td>1656.2</td>
<td>125.0</td>
<td>1531.2</td>
<td>calc1_3</td>
</tr>
<tr>
<td>8216000.0</td>
<td>3000.0</td>
<td>1000.0</td>
<td>2000.0</td>
<td>pe.0</td>
</tr>
<tr>
<td>6156000.0</td>
<td>1500.0</td>
<td>--</td>
<td>1500.0</td>
<td>pe.3</td>
</tr>
<tr>
<td>6156000.0</td>
<td>1500.0</td>
<td>--</td>
<td>1500.0</td>
<td>pe.5</td>
</tr>
</tbody>
</table>
Using L3 NB counters with CrayPat

Load CrayPat module

$ module load perftools

Compile and Instrument as usual:

$ cc -o stream stream.c
$ pat_build stream

Define counters (up to 4)

$ export PAT_RT_PERFCTR=L3_CACHE_MISSES,...

Run

$ aprun -ss -cc 0 stream+pat

Generate the report

$ pat_report -O hwpc stream+pat+2313073-25560s.xf
Limited derived metrics at the moment. Can be used to calculate bandwidth to main memory.

Table 3: Program HW Performance Counter Data

<table>
<thead>
<tr>
<th>L3_CACHE_MISSES:ALL:CORE_0</th>
<th>L3_CACHE_MISSES</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>207185723</td>
<td>207502576</td>
<td>Total</td>
</tr>
</tbody>
</table>

Table 2:

<table>
<thead>
<tr>
<th>L3_CACHE_MISSES:ALL:CORE_0</th>
<th>L3_CACHE_MISSES</th>
<th>PE=SHOW</th>
</tr>
</thead>
<tbody>
<tr>
<td>206606606</td>
<td>206887763</td>
<td>Total</td>
</tr>
<tr>
<td>---------------------------</td>
<td>----------------</td>
<td>----------</td>
</tr>
<tr>
<td>207128361</td>
<td>207421774</td>
<td>pe.0</td>
</tr>
<tr>
<td>206084850</td>
<td>206353752</td>
<td>pe.1</td>
</tr>
</tbody>
</table>
Profile Visualization with Cray Apprentice2
Cray Apprentice$^2$

- 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

Cray Apprentice$^2$ helps identify:
- Load imbalance
- Excessive communication
- Network contention
- Excessive serialization
- I/O Problems
Application Performance Summary

Profile

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

Memory Utilization
- L1 cache hit ratio: 94.3% hits
- Process HMem (MB/sec): 1683
- LD + ST per TLE miss: 160530.98 ref/miss

Data Movement
- No data collected.
- Run perf_info etc

Load Imbalance
- 0.17s = main
- 0.34s = pgas_get_pct
- 0.18s = pgas_put_pct

Function/Region Profile
- 61.3% = main
- 19.1% = _pgas_get_pct
- 14.0% = _pgas_put_pct

Wallclock time: 24.978773s

upcbw+pat:8538972-127t_ap2 (4,800 events in 2.801s)
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

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of PEs (MPI ranks)</td>
<td>16</td>
</tr>
<tr>
<td>Numbers of PEs per Node</td>
<td>16</td>
</tr>
<tr>
<td>Numbers of Threads per PE</td>
<td>1</td>
</tr>
<tr>
<td>Number of Cores per Socket</td>
<td>12</td>
</tr>
<tr>
<td>Execution start time</td>
<td>Thu Apr 7 09:50:13 2011</td>
</tr>
<tr>
<td>System type and speed</td>
<td>x86_64 2000 MHz</td>
</tr>
<tr>
<td>Current path to data file</td>
<td>swim+pat+10302-0t.ap2</td>
</tr>
</tbody>
</table>

Notes for table 1:
Table option:
-0 profile
Options implied by table option:
-tr @0.95,ti,imb_t1,imb_t1.tr -b gr, fu, pe-HIDE, th-HIDE

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.
The PE value for Time, Calls is the max for the Thread values.
(To specify different aggregations, see: pat_help report options s1)

<table>
<thead>
<tr>
<th>Time</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>49.30</td>
</tr>
<tr>
<td>98.59</td>
<td>147.89</td>
</tr>
<tr>
<td>197.18</td>
<td></td>
</tr>
</tbody>
</table>

swim+pat+10302-0t.ap2 (1.373s)
Generating New `pat_report` Tables

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

Panel Actions

Panel Help
Call Tree View

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
Call Tree Visualization
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

```plaintext
155  c angle pipelining loop (batches of mni angles)
156  c
157  do mc = 1, nmo
158         mli = (mc-1)*mni
159  end do
160  c K-inflows (k-k0 boundary)
161  c
162  if (k2.lt.0 .or. kbc.eq.0) then
163      do ml = 1, mni
164      do j = 1, jf
165      do i = 1, it
166         phi(kb(i,j,ml)) = 0.0d0
167      end do
168      end do
169  end do
170  else
171      if (do, dsa) then
172          leak = 0.0
173          k = k0 - k2
174          do ml = 1, mni
175          do j = 1, jf
176          do i = 1, it
177              phi(kb(i,j,ml)) = phi(kb(i,j,ml)) - phi(kb(i,j,ml))
178              leak = leak + wts(i)*phi(kb(i,j,ml))
179              & face(i,j,k+k3,j) = face(i,j,k+k3,j)
180              & + wts(i)*phi(kb(i,j,ml))
181          end do
182          end do
183          end do
184      else
185          leakage(i) = leakage(i) + leak
186      end if
187  end if
188
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
HW Counters Plot
Man pages

- \texttt{intro_craypat(1)}
  Introduces the craypat performance tool

- \texttt{pat_build(1)}
  Instrument a program for performance analysis

- \texttt{pat_help(1)}
  Interactive online help utility

- \texttt{pat_report(1)}
  Generate performance report in both text and for use with GUI

- \texttt{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. Time</th>
<th>Imb. Time%</th>
<th>Calls</th>
<th>Group Function</th>
</tr>
</thead>
<tbody>
<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>
</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 Order</th>
<th>On-Node Bytes/PE</th>
<th>On-Node 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>
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.

...
| # The 'USER_Time_hybrid' rank order in this file targets nodes with multi-core |
| # processors, based on Sent Msg Total Bytes collected for: |
| # Program: /lus/nid00023/malice/craypat/WORKSHOP/bh2o-demo/Rank/sweep3d/src/sweep3d |
| # Ap2 File: sweep3d.gmpi-u.ap2 |
| # Number PEs: 768 |
| # Max PEs/Node: 16 |
| # To use this file, make a copy named MPIRANK_ORDER, and set the environment variable MPIRANK_REORDER_METHOD to 3 prior to executing the program. |
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:

  • $\texttt{grid\_order } – \texttt{R } – \texttt{g 2,16}$
  
  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

  • $\texttt{grid\_order } – \texttt{C } – \texttt{g 2,16}$
  
  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)

- `$ 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 \sim 126s, stime \sim 108s

• Maximized on-node data movement with reordering
  • Application 8538982 resources: utime \sim 38s, stime \sim 106s
PerfSuite Background

- Active development at NCSA since 2001
- UI/NCSA Open Source license
- Targeted to users of all levels of expertise
  - The intent is to provide an easy-to-use mechanism for measuring application performance, and to expose problem areas for further exploration
- Low measurement overhead
Features

• Counting and profiling using hardware performance event counters on CPUs, GPUs, networks, and using interval timers
• Executes un-modified dynamically-linked applications
• Easy to use XML files for configuration and output
  o Easy to change the events to count/profile, just change the XML file
• Metadata (such as processor information, pid, memory and time usage) stored in the output
• Functionality available through both command-line tools and library API
Provides

• Three performance counter-related utilities:
  o psrun - generate raw counter or statistical profiling data from an unmodified binary
  o psprocess - pre- and post-process data
  o psinv - query events and machine information

• Three libraries
  o libpshwpc – HardWare Performance Counter library
    - This is the one that you are most likely to use, if you ever need to
      - Insert API calls in the source code for finer control of start/end locations
  o libperfsuite – the “core” library
  o libpshwpc_mpi – a convenience library to capture MPI calls
Using PerfSuite on BW

# First, load the perfsuite module

% module load perfsuite

# By default, psrun does event counting on the given program, then use psprocess to produce psrun’s output XML files

% aprun -n <num> psrun -f -p myprog myprog_args
% psprocess myprog.0.12345.nid01234.xml

# Use a profiling config file to do profiling instead of counting

% aprun -n <num> psrun -C -c papi_profile_cycles.xml -f -p myprog myprog_args
% psprocess myprog.1.67890.nid12345.xml
psinv: Processor Inventory

- Lists information about the characteristics of the computer
- This same information is also stored in PerfSuite XML output and is useful for later generating derived
- Lists available hardware performance events

```
titan:~3% psinv -v
System Information -
Processors: 2
Total Memory: 2007.16 MB
System Page Size: 16.00 KB

Processor Information -
Vendor: Intel
Processor family: IPF
Model (Type): Itanium
Revision: 6
Clock Speed: 800.136 MHz

Cache and TLB Information -
Cache levels: 3
Caches/TLBs: 7

Cache Details -
Level 1:
  Type: Data
  Size: 16 KB
  Line size: 32 bytes
  Associativity: 4-way set associative

  Type: Instruction
  Size: 16 KB
  Line size: 32 bytes
  Associativity: 4-way set associative```
psrun: Performance Measurement

• Hardware performance counting and profiling with unmodified dynamically-linked executables

• Supports:
  • x86, x86-64, and powerpc
  • MPI and OpenMP
  • resource usage (memory, CPU time) collection
  • Both PAPI standard and CPU/GPU native events

• Configuration = XML, Output = XML or text

• Use “-p” for OpenMP programs, “-f” for MPI, and “-f -p” for MPI+OpenMP programs
Example Configuration

• For counting: set “ps_hwpc_eventlist” as the XML root element
• For profiling: set “ps_hwpc_profile” as the XML root element

<?xml version="1.0" encoding="UTF-8" ?>
<ps_hwpc_eventlist class="PAPI">
  <ps_hwpc_event type="preset" name="PAPI_FP_OPS" />
  <ps_hwpc_event type="preset" name="PAPI_TOT_CYC" />
  <ps_hwpc_event type="preset" name="PAPI_L1_DCM" />
  <ps_hwpc_event type="preset" name="PAPI_L2_DCM" />
  <ps_hwpc_event type="native">NOPS_RETIRED</ps_hwpc_event>
  <ps_hwpc_event type="native">BACK_END_BUBBLE_ALL</ps_hwpc_event>
</ps_hwpc_eventlist>

<?xml version="1.0" encoding="UTF-8" ?>
<ps_hwpc_profile class="PAPI">
  <ps_hwpc_event type="preset" name="PAPI_BR_MSP" threshold="100000" />
</ps_hwpc_profile>
PerfSuite Hardware Performance Summary Report
Version : 1.0
Created : Mon Dec 30 11:31:53 AM Central Standard Time 2002
Generator : psprocess 0.5
XML Source : /u/ncsa/anyuser/performance/psrun-ia64.xml

Execution Information
====================================
Date : Sun Dec 15 21:01:20 2002
Host : user01

Processor and System Information
====================================
Node CPUs : 2
Vendor : Intel
Family : IPF
Model : Itanium
CPU Revision : 6
Clock (MHz) : 800.136
Memory (MB) : 2007.16
Pagesize (KB): 16
The reports (text or HTML) generated by psprocess have several sections, covering:

- Report creation details
- Run details
- Machine information
- Raw counter listings
- Counter explanations and index
- Derived metrics
- Run annotation defined by you

Derived metrics are evaluated at run-time and can be extended (text mode only)

---

**Cache Information**

<table>
<thead>
<tr>
<th>Cache levels</th>
<th>3</th>
</tr>
</thead>
</table>

**Level 1**

<table>
<thead>
<tr>
<th>Type</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size (KB)</td>
<td>16</td>
</tr>
<tr>
<td>Linesize (B)</td>
<td>32</td>
</tr>
<tr>
<td>Assoc</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size (KB)</td>
<td>16</td>
</tr>
<tr>
<td>Linesize (B)</td>
<td>32</td>
</tr>
<tr>
<td>Assoc</td>
<td>4</td>
</tr>
</tbody>
</table>

**Level 2**

<table>
<thead>
<tr>
<th>Type</th>
<th>unified</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size (KB)</td>
<td>96</td>
</tr>
<tr>
<td>Linesize (B)</td>
<td>64</td>
</tr>
<tr>
<td>Assoc</td>
<td>6</td>
</tr>
<tr>
<td>Index Description</td>
<td>Counter Value</td>
</tr>
<tr>
<td>------------------------------------------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>1 Conditional branch instructions mispredicted.</td>
<td>4831072449</td>
</tr>
<tr>
<td>4 Floating point instructions</td>
<td>86124489172</td>
</tr>
<tr>
<td>5 Total cycles</td>
<td>594547754568</td>
</tr>
<tr>
<td>6 Instructions completed</td>
<td>1049339828741</td>
</tr>
</tbody>
</table>

Statistics

| Graduated instructions per cycle                           | 1.765                |
| Graduated floating point instructions per cycle            | 0.145                |
| Level 3 cache miss ratio (data)                            | 0.957                |
| Bandwidth used to level 3 cache (MB/s)                     | 385.087              |
| % cycles with no instruction issue                         | 10.410               |
| % cycles stalled on memory access                          | 43.139               |
| MFLOPS (cycles)                                            | 115.905              |
| MFLOPS (wallclock)                                         | 114.441              |
# Profile Information

- **Class**: PAPI
- **Version**: 3.6.2
- **Event**: PAPI TOT_CYC (Total cycles)
- **Period**: 100000
- **Samples**: 200471
- **Domain**: user
- **Run Time**: 27.19 (seconds)
- **Min Self %**: (all)

## Module Summary

<table>
<thead>
<tr>
<th>Samples</th>
<th>Self %</th>
<th>Total %</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>186068</td>
<td>92.82%</td>
<td>92.82%</td>
<td>/home/rkufrin/apps/aspcg/aspcg</td>
</tr>
<tr>
<td>14182</td>
<td>7.07%</td>
<td>99.89%</td>
<td>/opt/intel/cc/9.0/lib/libguide.so</td>
</tr>
<tr>
<td>187</td>
<td>0.09%</td>
<td>99.98%</td>
<td>/lib/ld-2.3.6.so</td>
</tr>
<tr>
<td>18</td>
<td>0.01%</td>
<td>99.99%</td>
<td>/lib/tls/libc-2.3.6.so</td>
</tr>
<tr>
<td>15</td>
<td>0.01%</td>
<td>100.00%</td>
<td>/lib/tls/libpthread-2.3.6.so</td>
</tr>
<tr>
<td>1</td>
<td>0.00%</td>
<td>100.00%</td>
<td>/tmp/perfsuite/lib/libpsrun_r.so.0.0.1</td>
</tr>
</tbody>
</table>

## File Summary

<table>
<thead>
<tr>
<th>Samples</th>
<th>Self %</th>
<th>Total %</th>
<th>File</th>
</tr>
</thead>
<tbody>
<tr>
<td>154346</td>
<td>76.99%</td>
<td>76.99%</td>
<td>/home/rkufrin/apps/aspcg/pc_jac2d_blk3.f</td>
</tr>
<tr>
<td>14506</td>
<td>7.24%</td>
<td>84.23%</td>
<td>/home/rkufrin/apps/aspcg/cg3_blk.f</td>
</tr>
<tr>
<td>14505</td>
<td>7.24%</td>
<td>91.46%</td>
<td>??</td>
</tr>
<tr>
<td>10185</td>
<td>5.08%</td>
<td>96.54%</td>
<td>/home/rkufrin/apps/aspcg/matvec2d_blk3.f</td>
</tr>
<tr>
<td>3042</td>
<td>1.52%</td>
<td>98.06%</td>
<td>/home/rkufrin/apps/aspcg/dot_prod2d_blk3.f</td>
</tr>
<tr>
<td>2366</td>
<td>1.18%</td>
<td>99.24%</td>
<td>/home/rkufrin/apps/aspcg/add_exchange2d_blk3.f</td>
</tr>
<tr>
<td>834</td>
<td>0.42%</td>
<td>99.66%</td>
<td>/home/rkufrin/apps/aspcg/main3.f</td>
</tr>
<tr>
<td>687</td>
<td>0.34%</td>
<td>100.00%</td>
<td>/home/rkufrin/apps/aspcg/cs_jac2d_blk3.f</td>
</tr>
</tbody>
</table>
Function Summary

<table>
<thead>
<tr>
<th>Samples</th>
<th>Self %</th>
<th>Total %</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>154346</td>
<td>76.99%</td>
<td>76.99%</td>
<td>pc_jac2d_blk3</td>
</tr>
<tr>
<td>14506</td>
<td>7.24%</td>
<td>84.23%</td>
<td>cg3_blk</td>
</tr>
<tr>
<td>10185</td>
<td>5.08%</td>
<td>89.31%</td>
<td>matxvec2d_blk3</td>
</tr>
<tr>
<td>6937</td>
<td>3.46%</td>
<td>92.77%</td>
<td>__kmp_x86_pause</td>
</tr>
<tr>
<td>4711</td>
<td>2.35%</td>
<td>95.12%</td>
<td>__kmp_wait_sleep</td>
</tr>
<tr>
<td>3042</td>
<td>1.52%</td>
<td>96.64%</td>
<td>dot_prod2d_blk3</td>
</tr>
<tr>
<td>2366</td>
<td>1.18%</td>
<td>97.82%</td>
<td>add_exchange2d_blk3</td>
</tr>
</tbody>
</table>

Function:File:Line Summary

<table>
<thead>
<tr>
<th>Samples</th>
<th>Self %</th>
<th>Total %</th>
<th>Function:File:Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>39063</td>
<td>19.49%</td>
<td>19.49%</td>
<td>pc_jac2d_blk3:/home/rkufrin/apps/aspcg/cracy2d_blk3.f:20</td>
</tr>
<tr>
<td>24134</td>
<td>12.04%</td>
<td>31.52%</td>
<td>pc_jac2d_blk3:/home/rkufrin/apps/aspcg/cracy2d_blk3.f:19</td>
</tr>
<tr>
<td>15626</td>
<td>7.79%</td>
<td>39.32%</td>
<td>pc_jac2d_blk3:/home/rkufrin/apps/aspcg/cracy2d_blk3.f:21</td>
</tr>
<tr>
<td>15028</td>
<td>7.50%</td>
<td>46.82%</td>
<td>pc_jac2d_blk3:/home/rkufrin/apps/aspcg/cracy2d_blk3.f:33</td>
</tr>
<tr>
<td>13878</td>
<td>6.92%</td>
<td>53.74%</td>
<td>pc_jac2d_blk3:/home/rkufrin/apps/aspcg/cracy2d_blk3.f:24</td>
</tr>
<tr>
<td>11880</td>
<td>5.93%</td>
<td>59.66%</td>
<td>pc_jac2d_blk3:/home/rkufrin/apps/aspcg/cracy2d_blk3.f:31</td>
</tr>
<tr>
<td>8896</td>
<td>4.44%</td>
<td>64.10%</td>
<td>pc_jac2d_blk3:/home/rkufrin/apps/aspcg/cracy2d_blk3.f:22</td>
</tr>
<tr>
<td>7863</td>
<td>3.92%</td>
<td>68.02%</td>
<td>matxvec2d_blk3:/home/rkufrin/apps/aspcg/matrixvec2d_blk3.f:19</td>
</tr>
<tr>
<td>7145</td>
<td>3.56%</td>
<td>71.59%</td>
<td>pc_jac2d_blk3:/home/rkufrin/apps/aspcg/cracy2d_blk3.f:32</td>
</tr>
</tbody>
</table>
TAU’s ParaProf can display PerfSuite profiles after being mapped to source and stored as XML (psprocess –x)

Development version of psprocess produces Cube XML files directly
libpshwpc: Performance Collection API

C / C++

ps_hwpc_init (void)
ps_hwpc_start (void)
ps_hwpc_read (long long *values)
ps_hwpc_suspend (void)
ps_hwpc_stop (char *prefix)
ps_hwpc_shutdown (void)

Fortran

call psf_hwpc_init (ierr)
call psf_hwpc_start (ierr)
call psf_hwpc_read (integer*8 values,ierr)
call psf_hwpc_suspend (ierr)
call psf_hwpc_stop (prefix, ierr)
call psf_hwpc_shutdown (ierr)

• Call “init” once, call “start”, “read” and “suspend” as many times as you like. Call “stop” (supplying a file name prefix of your choice) to get the performance data XML document
• Optionally, call “shutdown”
• Example programs demonstrating use are installed in PerfSuite “examples” subdirectory
• Additional routines ps_hwpc_numevents() and ps_hwpc_eventnames() allow querying current configuration
FORTRAN API Example

```fortran
include 'fperfsuite.h'
call PSF_hwpc_init(ierr)
call PSF_hwpc_start(ierr)
do j = 1, n
   do i = 1, m
      do k = 1, l
         c(i,j) = c(i,j) + a(i,k)*b(k,j)
      end do
   end do
end do
call PSF_hwpc_stop('perf', ierr)
call PSF_hwpc_shutdown(ierr)
```

```bash
% ftn -c matmult.f -I /sw/xe/perfsuite/1.1.2/
cnl4.1_gnu4.7.2_papi5.1.0.2/include

% ftn matmult.o -L /sw/xe/perfsuite/1.1.2/cnl4.1_gnu4.7.2_papi5.1.0.2/
lib -L /opt/cray/papi/5.1.0.2/perf_events/no-cuda/lib -lpshwpc -lperfsuite -lpapi
```
Java-based Performance Measurement

- PerfSuite 1.0.0+ supports measuring unmodified Java applications using PAPI similar to `psrun`
- Implemented using JVMTI (Java Virtual Machine Tool Interface)
- Usage:
  ```
  java -agentlib:psjrun MyClass
  ```
- Results are contained in XML documents that can be post-processed using `psprocess`
For More Information

• PerfSuite web sites:
  • https://bluewaters.ncsa.illinois.edu/perfsuite
  • http://perfsuite.ncsa.illinois.edu
TAU Performance System®

- Tuning and Analysis Utilities (18+ year project)
- Developed at University of Oregon, Eugene
- Performance problem solving framework for HPC
  - Integrated, scalable, flexible, portable
  - Target all parallel programming / execution paradigms
- Integrated performance toolkit (open source)
  - Instrumentation, measurement, analysis, visualization
  - Widely-ported performance profiling / tracing system
  - Performance data management and data mining
- Broad application use (NSF, DOE, DOD, …)
Instrumentation and Sampling

• Supports both instrumentation (direct performance observation) and sampling (indirect performance observation)

• Instrumentation can be:
  • Manually insert into the source code
  • Automatically by compiler (use the “-optCompInst” option)
  • Automatically by TAU’s PDT tool (by choosing a TAU makefile containing the “-pdt” string)

• Supports selective instrumentation:
  • Selects the files, functions/routines, loops
  • Supports patterns: “*”, “?” (for file names), “#” (for loops)
**Instrumentation**

- event selection
  - source code
  - object code
  - library wrapper
  - binary code
  - virtual machine

**Measurement**

**Event creation and management**

- event identifier
- entry/exit events
- atomic events
- event mapping
- event control

**Profiling**

- statistics
- atomic profiles
- entry/exit profiles
- I/O profiles
- profile sampling

**Tracing**

- trace buffering
- record creation
- trace I/O
- timestamp generation
- trace filtering
- trace merging

**Performance data sources**

- timing
- hardware counters
- system counters
- kernel

**OS and runtime system modules**

- threading
- interrupts
- runtime system
- I/O
Using TAU on BW

• First, load the tau module
• Two methods to run to generate TAU profile or trace files
  • Choose a TAU makefile or use a TAU compiler script, to instrument the source code, build an instrumented executable, then execute it
  • Use “tau_exec” to run an uninstrumented executable
• Analyze the generated files, using “pprof” for quick text output, or “paraprof” for richer visualization
TAU Makefiles on BW

- To see all installed TAU versions
  module avail tau
- To load the tau module, use one of the following:
  module load tau
  module load tau/<specific version>
- The TAU makefiles are located at:
  /sw/xe/tau/<tau_version>/<build>/craycnl/lib/Makefile.tau-*
  For example, for TAU 2.21.4 using PrgEnv-cray, they are:
  /sw/xe/tau/2.21.4/cnl4.1_cray8.1.1/craycnl/lib/Makefile.tau-*
- For an MPI+F90 application, you may want to start with
  Makefile.tau-cray-mpi-pdt, which supports MPI instrumentation
  & PDT for automatic source instrumentation
  setenv TAU_MAKEFILE
  /sw/xe/tau72.21.4/cnl4.1_cray8.1.1/craycnl/lib/
  Makefile.tau-cray-mpi-pdt
### Parallel Profile Analysis – pprof

<table>
<thead>
<tr>
<th>%Time</th>
<th>Exclusive msec</th>
<th>Inclusive msec</th>
<th>#Call</th>
<th>#Subs</th>
<th>Inclusive Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0</td>
<td>1</td>
<td>3:11.293</td>
<td>1</td>
<td>15</td>
<td>applu</td>
</tr>
<tr>
<td>99.6</td>
<td>3,667</td>
<td>3:10.463</td>
<td>3</td>
<td>37517</td>
<td>bcast_inputs</td>
</tr>
<tr>
<td>67.1</td>
<td>491</td>
<td>2:08.328</td>
<td>37200</td>
<td>37200</td>
<td>3450 exchange_1</td>
</tr>
<tr>
<td>44.5</td>
<td>6,461</td>
<td>1:25.159</td>
<td>9300</td>
<td>18600</td>
<td>9157 buts</td>
</tr>
<tr>
<td>41.0</td>
<td>1:18.436</td>
<td>1:18.436</td>
<td>18600</td>
<td>0</td>
<td>4217 MPI_Recv()</td>
</tr>
<tr>
<td>29.5</td>
<td>6,778</td>
<td>56,407</td>
<td>9300</td>
<td>18600</td>
<td>6065 bcts</td>
</tr>
<tr>
<td>26.2</td>
<td>50,142</td>
<td>50,142</td>
<td>19204</td>
<td>0</td>
<td>2611 MPI_Send()</td>
</tr>
<tr>
<td>16.2</td>
<td>24,451</td>
<td>31,031</td>
<td>301</td>
<td>602</td>
<td>103096 rhs</td>
</tr>
<tr>
<td>3.9</td>
<td>7,501</td>
<td>7,501</td>
<td>9300</td>
<td>0</td>
<td>807 jacl</td>
</tr>
<tr>
<td>3.4</td>
<td>838</td>
<td>6,594</td>
<td>604</td>
<td>1812</td>
<td>10918 exchange_3</td>
</tr>
<tr>
<td>3.4</td>
<td>6,590</td>
<td>6,590</td>
<td>9300</td>
<td>0</td>
<td>709 jacl</td>
</tr>
<tr>
<td>2.6</td>
<td>4,989</td>
<td>4,989</td>
<td>608</td>
<td>0</td>
<td>8266 MPI_Wait()</td>
</tr>
<tr>
<td>0.2</td>
<td>0.44</td>
<td>400</td>
<td>1</td>
<td>4</td>
<td>400051 init_comm</td>
</tr>
<tr>
<td>0.2</td>
<td>398</td>
<td>398</td>
<td>1</td>
<td>39</td>
<td>399634 MPI_Init()</td>
</tr>
<tr>
<td>0.1</td>
<td>140</td>
<td>247</td>
<td>1</td>
<td>47616</td>
<td>247068 setiv</td>
</tr>
<tr>
<td>0.1</td>
<td>131</td>
<td>131</td>
<td>57252</td>
<td>0</td>
<td>2 exact</td>
</tr>
<tr>
<td>0.1</td>
<td>89</td>
<td>103</td>
<td>1</td>
<td>2</td>
<td>103168 erhs</td>
</tr>
<tr>
<td>0.0</td>
<td>866</td>
<td>95</td>
<td>1</td>
<td>2</td>
<td>964458 read_input</td>
</tr>
<tr>
<td>0.0</td>
<td>95</td>
<td>95</td>
<td>9</td>
<td>0</td>
<td>10603 MPI_Bcast()</td>
</tr>
<tr>
<td>0.0</td>
<td>26</td>
<td>44</td>
<td>1</td>
<td>7937</td>
<td>44878 error</td>
</tr>
<tr>
<td>0.0</td>
<td>24</td>
<td>24</td>
<td>608</td>
<td>0</td>
<td>40 MPI_Irecv()</td>
</tr>
<tr>
<td>0.0</td>
<td>15</td>
<td>15</td>
<td>1</td>
<td>5</td>
<td>15630 MPI_Finalize()</td>
</tr>
<tr>
<td>0.0</td>
<td>4</td>
<td>12</td>
<td>1</td>
<td>1700</td>
<td>12335 setbv</td>
</tr>
<tr>
<td>0.0</td>
<td>7</td>
<td>8</td>
<td>3</td>
<td>3</td>
<td>2893 12norm</td>
</tr>
<tr>
<td>0.0</td>
<td>3</td>
<td>3</td>
<td>8</td>
<td>0</td>
<td>491 MPI_Allreduce()</td>
</tr>
<tr>
<td>0.0</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>6</td>
<td>3874 pintgr</td>
</tr>
<tr>
<td>0.0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1007 MPI_BARRIER()</td>
</tr>
<tr>
<td>0.0</td>
<td>0.116</td>
<td>0.837</td>
<td>1</td>
<td>4</td>
<td>837 exchange_4</td>
</tr>
<tr>
<td>0.0</td>
<td>0.512</td>
<td>0.512</td>
<td>1</td>
<td>0</td>
<td>512 MPI_Keyval_create()</td>
</tr>
<tr>
<td>0.0</td>
<td>0.333</td>
<td>0.333</td>
<td>1</td>
<td>0</td>
<td>383 exchange_5</td>
</tr>
<tr>
<td>0.0</td>
<td>0.024</td>
<td>0.191</td>
<td>1</td>
<td>2</td>
<td>191 exchange_6</td>
</tr>
<tr>
<td>0.0</td>
<td>0.103</td>
<td>0.103</td>
<td>6</td>
<td>0</td>
<td>17 MPI_Type_contiguous()</td>
</tr>
</tbody>
</table>
Parallel Profile Analysis – ParaProf

Raw files
PerfDMF managed (database)
Application
Experiment
Trial

HPMToolkit
Metadata
MpiP
TAU
ParaProf – Flat Profile

node, context, thread

8K processors

Miranda

- hydrodynamics
- Fortran + MPI
- LLNL BG/L
For More Information

• TAU web sites:
  • https://bluewaters.ncsa.illinois.edu/tau
  • http://tau.uoregon.edu
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.
Congestion on a Shared, Torus Network

- HSN uses dimension ordered routing: x-then-y-then-z between two locations on the torus. Note that A → B ≠ B → A.
- Shortest route can sometimes cause traffic to pass through geminis used by other jobs.
- Non-convex node allocations can have traffic that passes through geminis used by other jobs.
- Non-local I/O traffic and Lustre striping can lead to network hot-spots.
- We are working with Adaptive and Cray on eliminating some of the above causes of congestion with better node allocation: shape, location, etc.
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.
Thank You