Performance Measurement and Analysis Tools for Cray XE/XK Systems

Heidi Poxon
Cray Inc.
Topics

- Analysis performed by the Cray performance tools
- Visualization of performance data
Analysis Performed by the Cray Performance Tools
Load Imbalance
Motivation for Load Imbalance Analysis

● Increasing system software and architecture complexity
  ● Current trend in high end computing is to have systems with tens of thousands of processors
    ● This is being accentuated with multi-core processors

● Applications have to be very well balanced in order to perform at scale on these MPP systems
  ● Efficient application scaling includes a balanced use of requested computing resources

● Desire to minimize computing resource “waste”
  ● Identify slower paths through code
  ● Identify inefficient “stalls” within an application
MPI Sync Time

- Measure load imbalance in programs instrumented to trace MPI functions to determine if MPI ranks arrive at collectives together
- Separates potential load imbalance from data transfer
- Sync times reported by default if MPI functions traced
- If desired, PAT_RT_MPI_SYNC=0 deactivates this feature
Imbalance Time

- Metric based on execution time
- It is dependent on the type of activity:
  - User functions
    \[ \text{Imbalance time} = \text{Maximum time} - \text{Average time} \]
  - Synchronization (Collective communication and barriers)
    \[ \text{Imbalance time} = \text{Average time} - \text{Minimum time} \]
- Identifies computational code regions and synchronization calls that could benefit most from load balance optimization
- Estimates how much overall program time could be saved if corresponding section of code had a perfect balance
  - Represents upper bound on “potential savings”
  - Assumes other processes are waiting, not doing useful work while slowest member finishes
Imbalance %

\[ \text{Imbalance\%} = 100 \times \frac{\text{Imbalance\ time}}{\text{Max\ Time}} \times \frac{N}{N - 1} \]

- Represents % of resources available for parallelism that is “wasted”
- Corresponds to % of time that rest of team is not engaged in useful work on the given function
- Perfectly balanced code segment has imbalance of 0%
- Serial code segment has imbalance of 100%
# Load Imbalance Example in Sampling

## 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>120.2</td>
<td>--</td>
<td>--</td>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>99.9%</td>
<td>120.0</td>
<td>--</td>
<td>--</td>
<td>USER</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>91.2%</td>
<td>109.6</td>
<td>77.4</td>
<td>41.6%</td>
<td>line.226</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8.6%</td>
<td>10.3</td>
<td>--</td>
<td>--</td>
<td>jacobi</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4.5%</td>
<td>5.5</td>
<td>5.5</td>
<td>50.6%</td>
<td>line.382</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.1%</td>
<td>0.2</td>
<td>--</td>
<td>--</td>
<td>ETC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Load Imbalance Example in Sampling

NCSA Workshop, February 2013

Cray Inc.
Call Tree with Discrete Unit of Help

Function `mpi_recv` has the highest load imbalance time (13.862401 seconds) and is therefore a candidate for further examination for performance optimization.

Imbalance time provides an estimate of how much time would be saved for the overall program if the corresponding area of the application had perfect balance.

To identify functions with notable load imbalance, look for "tail" or "wide" boxes with lots of yellow. Remember that an imbalance that shows up at one point in the call tree may be due to an imbalance that occurred further up in the call tree. To see values for the imbalance time and percentage, hover over the function.
Cache Utilization
The performance tools provide additional automatic HW counter analysis and observations for:

● **TLB utilization**
  - Measures how well the memory hierarchy is being utilized with regards to TLB
  - Depends on computation being single precision or double precision
  - Poor utilization indicates that not all entries on the page are being utilized between 2 TLB misses

● **cache utilization**
  - Poor utilization indicates that not all entries on the cache line are being utilized between 2 cache misses

● **D1 cache hit (or miss) ratios**

● **D1+D2 cache hit (or miss) ratios**
Example Cache Threshold Observations

<table>
<thead>
<tr>
<th></th>
<th>D1 cache utilization:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>61.7% of total execution time was spent in 1 functions with D1 cache hit ratios below</td>
</tr>
<tr>
<td></td>
<td>the desirable minimum of 90.0%. Cache utilization might be improved by modifying the</td>
</tr>
<tr>
<td></td>
<td>alignment or stride of references to data arrays in these functions.</td>
</tr>
<tr>
<td></td>
<td>D1    Time%   Function</td>
</tr>
<tr>
<td></td>
<td>cache                                         hit                                         ratio</td>
</tr>
<tr>
<td></td>
<td>74.3%  61.7%  calc3_</td>
</tr>
</tbody>
</table>

|                  | D1 + D2 cache utilization:                                                               |
|                  | 61.7% of total execution time was spent in 1 functions with combined D1 and D2 cache  |
|                  | hit ratios below the desirable minimum of 97.0%. Cache utilization might be improved  |
|                  | by modifying the alignment or stride of references to data arrays in these functions. |
|                  | D1+D2  Time%   Function                                                                 |
|                  | cache                                         hit                                         ratio |
|                  | 96.6%  61.7%  calc3_                           |
|                  | ...                                          |
Example Cache Threshold Observations (2)

================= Observations and suggestions =================

TLB utilization:
82.5% of total execution time was spent in 2 functions with fewer
than the desirable minimum of 512 data references per TLB miss. TLB
utilization might be improved by modifying the alignment or stride
of references to data arrays in these functions.

<table>
<thead>
<tr>
<th>LS</th>
<th>Time%</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>per</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.97</td>
<td>61.7%</td>
<td>calc3_</td>
</tr>
<tr>
<td>163.77</td>
<td>20.8%</td>
<td>calc2_</td>
</tr>
</tbody>
</table>

================= End Observations =================
View Profile Data 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:
...
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:
-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
### Notes for 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>Total</td>
</tr>
<tr>
<td>94.2%</td>
<td>730</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>USER</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>43.4%</td>
<td>336</td>
<td>8.75</td>
<td>2.6%</td>
<td>mlwxyz_</td>
<td></td>
</tr>
<tr>
<td>16.1%</td>
<td>125</td>
<td>6.28</td>
<td>4.9%</td>
<td>half_</td>
<td></td>
</tr>
<tr>
<td>8.0%</td>
<td>62</td>
<td>6.25</td>
<td>9.5%</td>
<td>full_</td>
<td></td>
</tr>
<tr>
<td>6.8%</td>
<td>53</td>
<td>1.88</td>
<td>3.5%</td>
<td>artv_</td>
<td></td>
</tr>
<tr>
<td>4.9%</td>
<td>38</td>
<td>1.34</td>
<td>3.6%</td>
<td>bnd_</td>
<td></td>
</tr>
<tr>
<td>3.6%</td>
<td>28</td>
<td>2.00</td>
<td>6.9%</td>
<td>currentf_</td>
<td></td>
</tr>
<tr>
<td>2.2%</td>
<td>17</td>
<td>1.50</td>
<td>8.6%</td>
<td>bndsf_</td>
<td></td>
</tr>
<tr>
<td>1.7%</td>
<td>13</td>
<td>1.97</td>
<td>13.5%</td>
<td>model_</td>
<td></td>
</tr>
<tr>
<td>1.4%</td>
<td>11</td>
<td>1.53</td>
<td>12.2%</td>
<td>cfl_</td>
<td></td>
</tr>
<tr>
<td>1.3%</td>
<td>10</td>
<td>0.75</td>
<td>7.0%</td>
<td>currentnh_</td>
<td></td>
</tr>
<tr>
<td>1.0%</td>
<td>8</td>
<td>5.28</td>
<td>41.9%</td>
<td>bndbo_</td>
<td></td>
</tr>
<tr>
<td>1.0%</td>
<td>8</td>
<td>8.28</td>
<td>53.4%</td>
<td>bndto_</td>
<td></td>
</tr>
<tr>
<td>5.4%</td>
<td>42</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>MPI</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.9%</td>
<td>15</td>
<td>4.62</td>
<td>23.9%</td>
<td>mpi_sendrecv_</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>ffte4_</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 Bytes</th>
<th>MPI Msg Count</th>
<th>MsgSz</th>
<th>4KB&lt;=&lt;16B</th>
<th>PE[mmm]</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>---------------</td>
<td>---------------</td>
<td>-------</td>
<td>-----------</td>
<td>---------</td>
<td>----------</td>
</tr>
<tr>
<td>15138076.0</td>
<td>4099.4</td>
<td>411.6</td>
<td>3687.8</td>
<td>Total</td>
<td></td>
</tr>
<tr>
<td>15138028.0</td>
<td>4093.4</td>
<td>405.6</td>
<td>3687.8</td>
<td>MPI_ISEND</td>
<td></td>
</tr>
<tr>
<td>8080500.0</td>
<td>2062.5</td>
<td>93.8</td>
<td>1968.8</td>
<td>calc2_</td>
<td></td>
</tr>
<tr>
<td>8216000.0</td>
<td>3000.0</td>
<td>1000.0</td>
<td>2000.0</td>
<td>pe.0</td>
<td></td>
</tr>
<tr>
<td>8208000.0</td>
<td>2000.0</td>
<td>--</td>
<td>2000.0</td>
<td>pe.9</td>
<td></td>
</tr>
<tr>
<td>6160000.0</td>
<td>2000.0</td>
<td>500.0</td>
<td>1500.0</td>
<td>pe.15</td>
<td></td>
</tr>
<tr>
<td>6285250.0</td>
<td>1656.2</td>
<td>125.0</td>
<td>1531.2</td>
<td>calc1_</td>
<td></td>
</tr>
<tr>
<td>8216000.0</td>
<td>3000.0</td>
<td>1000.0</td>
<td>2000.0</td>
<td>pe.0</td>
<td></td>
</tr>
<tr>
<td>6156000.0</td>
<td>1500.0</td>
<td>--</td>
<td>1500.0</td>
<td>pe.3</td>
<td></td>
</tr>
<tr>
<td>6156000.0</td>
<td>1500.0</td>
<td>--</td>
<td>1500.0</td>
<td>pe.5</td>
<td></td>
</tr>
</tbody>
</table>

...
Profile Visualization with Cray Apprentice2
Cray Apprentice

- 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 helps identify:
- Load imbalance
- Excessive communication
- Network contention
- Excessive serialization
- I/O Problems
Application Performance Summary
Statistics Overview

Switch Overview display
Load Balance View (Aggregated from Overview)

Min, Avg, and Max Values

-1, +1 Std Dev marks
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

- Select All
- Select None

Panel Actions

Panel Help
Apprentice$^2$ Call Tree View of Sampled Data
Load balance overview:
Height $\Leftrightarrow$ Max time
Middle bar $\Leftrightarrow$ Average time
Lower bar $\Leftrightarrow$ Min time
Yellow represents imbalance time

Width $\Leftrightarrow$ inclusive time
Height $\Leftrightarrow$ 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)

-1, +1 Std Dev marks

Min, Avg, and Max Values
Source Mapping from Call Tree

```plaintext
165  c angle pipelining loop (batches of mmi angles)
166  c
167      DO mo = 1, nmo
168          m0 = (mo-1)*mmi
169  c K-inflows (k=k0 boundary)
170      c
171      if (k2.lt.0 .or. kbc.eq.0) then
172          do mi = 1, mmi
173              do j = 1, jk
174                  do i = 1, ii
175                      phi(kb(i,j,mi)) = 0.0d0
176                  end do
177          end do
178      end do
179      end do
180      else
181      if (do_dsao) then
182          leak = 0.0
183              k = k0 - k2
184              do mi = 1, mmi
185                  m = mi + mi0
186                  do j = 1, jm
187                      do i = 1, it
188                          phi(kb(i,j,mi)) = phi(kb(i,j,mi))
189                          leak = leak
190                              + wtci(m)*phi(kb(i,j,mi))*di(i)*dj(j)
191                          & face(i,j,k+3,3) = face(i,j,k+3,3)
192                          & + wtci(m)*phi(kb(i,j,mi))
193                      end do
194                  end do
195              end do
196      end do
197      else
198          leakage(S) = leakage(S) + leak
199      end do
0.00  2.13  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)
Activity Report

Activity over Time

- Synchronization
- Data Transmission
- Read
- Write
- Other Traced
Mosaic View – Shows Communication Pattern

![Mosaic View Diagram](image)
HW Counters Overview
HW Counters Plot
Traffic Report – MPI Communication Timeline

[Diagram of MPI Communication Timeline]

Message matching finished. (0.00 secs)
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
Questions?