A Comparison of Performance Analysis Tools on the NERSC SP

Jonathan Carter
NERSC User Services
Performance Tools on the IBM SP

- **PE Benchmarker**
  - IBM PSSP
  - Trace and visualize hardware counters values or MPI and user-defined events

- **Paraver**
  - European Center for Parallelism at Barcelona (CEPBA)
  - Trace and visualize program states, hardware counters values, and MPI and user-defined events

- **Vampir**
  - Pallas GmbH
  - Trace and visualize MPI and user-defined events

- **PAPI, HPMlib, poe+, etc.**
  - Tools based on pmapi hardware-counter interface
  - Summaries over entire execution
Example Program

• Solve Poisson equation on a 2D grid using Jacobi iteration

$$\nabla^2 u = f(x, y)$$

$$u(x, y) = g(x, y)$$

$$u_{i,j}^{k+1} = \frac{1}{4} (u_{i-1,j}^k + u_{i,j+1}^k + u_{i,j-1}^k + u_{i+1,j}^k - h^2 f_{i,j})$$
Example Program

- Decompose 2D grid into blocks, giving a block to each MPI process
- Assuming 4 MPI processes:
Example Program

- At the edges, information must be exchanged between MPI processes:
Example Program

- Flow chart of execution:

  initialize
  while ! converged
    exchange boundary data
    jacobi
    collect global sum of differences
    update
  finalize

- Each MPI process is parallelized using OpenMP for the `jacobi` and `update` phases.
PE Benchmarker

• Performance Collection Tool (pct)
  – Application is run under the control of pct, uses DPCL to insert
    probes into executable before it runs.
  – Collects hardware/OS profiles at a thread level on a routine by
    routine basis.
  – Collects MPI event statistics.

• Performance Visualization Tool (pvt)
  – Graphical display of execution statistics.

• MPI events
  – Summaries only (utestats), relies on ANL *jumpshot* to visualize MPI
    timelines.
Performance Collection Tool

- Available with a graphical or command line interface
- May collect MPI event statistics or hardware/OS profiles (hardware counters, cpu time, etc.)
- Limitations:
  - Instrumentation takes place after program has been launched by POE. For a large number of source files and routines, this can take a considerable amount of time.
  - Requires 512 MB of shared memory segments
Performance Collection Tool

- For MPI event statistics
  - Select subset of MPI processes
  - Select subset of source files or routines
  - Select subset of MPI events
  - Add user-defined events

- For hardware/OS profiles
  - Select subset of processes
  - Select subset of timers, memory usage, hardware-counter group
  - Define hardware-counter groups (several already predefined)
Performance Collection Tool

- Must use "threaded" compilers for MPI tracing
- Must set environment variable MP_UTE=YES before linking MPI application if tracing of MPI events is required
- To run
  
  module load java
  
  pct
Performance Collection Tool
Performance Collection Tool – Profiles

Select processes, routines, data to be collected:

- SourceTree:
  - SourceRoot
  - ProbeRoot
  - poisson2
    - poisson2.f90
    - ProbeRoot
    - communicate
    - init
    - jscobi
    - jscobi@OL@1
    - jscobi@RV@1
    - mapping
    - poisson
    - procmem
    - update
    - update@OL@2
    - xifalloc.f

Information:
function task OfileId 11
waiting to receive source code function nodes
children received.
There were 10 function(s) added to poisson2.f90.
Performance Collection Tool – Profiles

- After the application completes:
  - Files named `basename.cdf.xx` created, one per process
  - Contain hardware profiles that can be viewed with the Performance Visualization Tool
  - Generally files are not too large
Performance Collection Tool – MPI Events

May need to increase Maximum trace file size, decide on Event Types to be monitored:
Performance Collection Tool – MPI Events

Selecting MPI Events:

![Performance Collection Tool interface]

- SourceRoot
  - ProbeRoot
  - poisson2
    - ProbeRoot
    - poisson2_f90
    - xifalloc.f

Information
- select trace
- trace set path "/u4/jcarter/poisson/af"
- select command succeeded.
- trace command succeeded.
Performance Collection Tool – MPI Events

Selecting User Markers:

![Screenshot of the Performance Collection Tool showing user markers and options for selecting MPI Events.]
Performance Collection Tool – MPI Events
Performance Collection Tool – MPI Events

• After the application completes:
  – Files named basename.xx created, one per node (xx is rank of one random task per node)
  – Files are generally large (AIX trace files)
    • Depends on what is traced: mpi, process, idle
  – Need to preprocess files with the uteconvert command to produce UTE files
  – Can collect statistics directly from UTE files using utestats command, and use the slogmerge in order to view with jumpshot
Command Line Interface

- Invoke with `pct -c`
- Provide a script file with `pct -c -s scriptfile`

```plaintext
load poe exec /usr/common/homes/j/jcarter/poisson/poisson2 poeargs "-procs 4
-nodes 1 -retry 30 -retrycount 30"
select trace
trace set path "/usr/common/homes/j/jcarter/poisson/tmp"
trace set logsize 50
trace set event mpi
trace add mpiname all to file "*"
trace add beginmarker "jacobi" to file "poisson2.f90" funcid 2 pointid 0
trace add endmarker "jacobi" to file "poisson2.f90" funcid 2 pointid 5
trace add beginmarker "update" to file "poisson2.f90" funcid 8 pointid 0
trace add endmarker "update" to file "poisson2.f90" funcid 8 pointid 5
start
wait
```
Performance Visualization Tool

- View hardware/OS profiles
- To run
  
  module load java
  
  pvt basename.cdf.*
Performance Visualization Tool

Select data to be displayed from "Data View" menu:
Performance Visualization Tool

Expand processes to show functions and threads:

<table>
<thead>
<tr>
<th>Source View</th>
<th>Data View</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root node (function-centric)</td>
<td>Wall Clock Time</td>
</tr>
<tr>
<td>Process: taskid=0, pid=32744</td>
<td>5821.411</td>
</tr>
<tr>
<td>File: poisson2.f90</td>
<td>727.160</td>
</tr>
<tr>
<td>Function: communicate</td>
<td>727.159</td>
</tr>
<tr>
<td>Function: init</td>
<td>6.620</td>
</tr>
<tr>
<td>Function: jacobi</td>
<td>1.559</td>
</tr>
<tr>
<td>Function: jacobi@OL@1</td>
<td>1.500</td>
</tr>
<tr>
<td>Thread: id=1</td>
<td>413.167</td>
</tr>
<tr>
<td>Thread: id=1286</td>
<td>102.860</td>
</tr>
<tr>
<td>Thread: id=1543</td>
<td>103.515</td>
</tr>
<tr>
<td>Thread: id=1800</td>
<td>103.423</td>
</tr>
<tr>
<td>Function: jacobi@OL@1</td>
<td>103.368</td>
</tr>
<tr>
<td>Function: mapping</td>
<td>0.080</td>
</tr>
<tr>
<td>Function: poisson</td>
<td>3.3378E-5</td>
</tr>
<tr>
<td>Function: procmap</td>
<td>3.563</td>
</tr>
<tr>
<td>Function: update</td>
<td>0.005</td>
</tr>
<tr>
<td>Function: update@OL@1</td>
<td>0.752</td>
</tr>
<tr>
<td>File: xfaloc.f</td>
<td>299.914</td>
</tr>
<tr>
<td>Process: taskid=1, pid=45470</td>
<td>2.95464E-4</td>
</tr>
<tr>
<td>Process: taskid=2, pid=30288</td>
<td>726.633</td>
</tr>
<tr>
<td>Process: taskid=3, pid=41700</td>
<td>726.933</td>
</tr>
<tr>
<td>Process: taskid=4, pid=43324</td>
<td>727.225</td>
</tr>
<tr>
<td>Process: taskid=5, pid=40822</td>
<td>728.255</td>
</tr>
<tr>
<td>Process: taskid=6, pid=33772</td>
<td>728.296</td>
</tr>
<tr>
<td>Process: taskid=7, pid=10756</td>
<td>728.599</td>
</tr>
<tr>
<td>Process: taskid=8, pid=10756</td>
<td>728.311</td>
</tr>
</tbody>
</table>
Performance Visualization Tool

Select Thread-Centric View for another way of grouping results:
Performance Visualization Tool

Produce various text reports:
Performance Visualization Tool

Text reports can be saved as tab-separated text files.
Performance Visualization Tool

Show detailed statistics on one process, function or thread:
## Performance Visualization Tool

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
<th>Units</th>
<th>N</th>
<th>Summary</th>
<th>Average</th>
<th>StdDev</th>
<th>Max</th>
<th>Max ID</th>
<th>Min</th>
<th>Min ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wall Clock Time</td>
<td></td>
<td>s</td>
<td>1</td>
<td>16000</td>
<td>2000</td>
<td>413.313</td>
<td>0.578</td>
<td>408.460</td>
<td>7</td>
<td>407.590</td>
</tr>
<tr>
<td>User CPU Usage</td>
<td></td>
<td>s</td>
<td>1</td>
<td>3263.15</td>
<td>407.894</td>
<td>0.272</td>
<td>408.460</td>
<td>7</td>
<td>407.590</td>
<td></td>
</tr>
<tr>
<td>System CPU U...</td>
<td></td>
<td>s</td>
<td>2</td>
<td>31.86</td>
<td>3.983</td>
<td>0.247</td>
<td>4.410</td>
<td>6</td>
<td>3.640</td>
<td></td>
</tr>
<tr>
<td>Maximum Res...</td>
<td></td>
<td>s</td>
<td>2</td>
<td>300.183</td>
<td>2401.464</td>
<td>17.176</td>
<td>3.00228</td>
<td>0</td>
<td>300.172</td>
<td></td>
</tr>
<tr>
<td>Page Fault Wt...</td>
<td></td>
<td>s</td>
<td>3</td>
<td>6114</td>
<td>195592</td>
<td>24449</td>
<td>4.330</td>
<td>24456</td>
<td>5</td>
<td>24442</td>
</tr>
<tr>
<td>Page Fault Wt...</td>
<td></td>
<td>s</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Voluntary Con...</td>
<td></td>
<td>s</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Involuntary Co...</td>
<td></td>
<td>s</td>
<td>6</td>
<td>33</td>
<td>26337</td>
<td>3292.125</td>
<td>647.622</td>
<td>4651</td>
<td>4</td>
<td>2488</td>
</tr>
<tr>
<td>Floating-Point...</td>
<td></td>
<td>s</td>
<td>7</td>
<td>0.324</td>
<td>2.594</td>
<td>0.324</td>
<td>2.9058E-4</td>
<td>0.325</td>
<td>2</td>
<td>0.324</td>
</tr>
<tr>
<td>Fixed-Point In...</td>
<td></td>
<td>s</td>
<td>8</td>
<td>0.004</td>
<td>0.035</td>
<td>0.004</td>
<td>2.12258E-4</td>
<td>0.005</td>
<td>5</td>
<td>0.004</td>
</tr>
<tr>
<td>Branch instru...</td>
<td></td>
<td>s</td>
<td>9</td>
<td>N/A</td>
<td>1.63</td>
<td>1.06</td>
<td>0.163</td>
<td>1.21356E-4</td>
<td>0.163</td>
<td>2</td>
</tr>
<tr>
<td>Load instru...</td>
<td></td>
<td>s</td>
<td>10</td>
<td>0.041</td>
<td>0.329</td>
<td>0.041</td>
<td>3.06346E-5</td>
<td>0.041</td>
<td>4</td>
<td>0.041</td>
</tr>
<tr>
<td>Store instru...</td>
<td></td>
<td>s</td>
<td>11</td>
<td>6.28825E9</td>
<td>2.01311E11</td>
<td>2.51639E10</td>
<td>8170277.127</td>
<td>2.51748E10</td>
<td>5</td>
<td>2.51546E10</td>
</tr>
<tr>
<td>Floating...</td>
<td></td>
<td>s</td>
<td>12</td>
<td>6.00123E9</td>
<td>1.92042E11</td>
<td>2.40052E10</td>
<td>505732.519</td>
<td>2.40058E10</td>
<td>5</td>
<td>2.40045E10</td>
</tr>
<tr>
<td>Number Of Lo...</td>
<td></td>
<td>s</td>
<td>13</td>
<td>1.58338E9</td>
<td>5.07191E10</td>
<td>6.33988E9</td>
<td>4307560.447</td>
<td>6.34556E9</td>
<td>5</td>
<td>6.33485E9</td>
</tr>
<tr>
<td>Processor Cl...</td>
<td></td>
<td>s</td>
<td>14</td>
<td>3.85545E10</td>
<td>1.23337E12</td>
<td>1.54171E11</td>
<td>1.38131E8</td>
<td>1.54408E11</td>
<td>7</td>
<td>1.53941E11</td>
</tr>
<tr>
<td>FPUC (Floating...</td>
<td></td>
<td>s</td>
<td>15</td>
<td>6.49821E9</td>
<td>2.07942E11</td>
<td>2.59928E10</td>
<td>49761.1849</td>
<td>2.59935E10</td>
<td>0</td>
<td>2.59922E10</td>
</tr>
<tr>
<td>FXU2 (Fixed...</td>
<td></td>
<td>s</td>
<td>16</td>
<td>1.88941E7</td>
<td>6.39244E8</td>
<td>7.99056E7</td>
<td>3856520.521</td>
<td>8.49423E7</td>
<td>5</td>
<td>7.54959E7</td>
</tr>
<tr>
<td>FXU0 (Fixed...</td>
<td></td>
<td>s</td>
<td>17</td>
<td>9.57332E7</td>
<td>2.95926E9</td>
<td>3.69907E8</td>
<td>6.52351E7</td>
<td>4.32752E8</td>
<td>5</td>
<td>2.45477E8</td>
</tr>
<tr>
<td>FXU1 (Fixed...</td>
<td></td>
<td>s</td>
<td>18</td>
<td>4.59100E7</td>
<td>1.82913E9</td>
<td>2.28641E8</td>
<td>5.93726E7</td>
<td>3.42402E8</td>
<td>4</td>
<td>1.83807E8</td>
</tr>
</tbody>
</table>
Visualizing MPI Events

• Convert AIX tracefiles to UTE tracefiles
  
  \texttt{uteconvert -n n basename}.

• Merge UTE tracefiles
  
  \texttt{utemerge -n n basename.ute}.

• Generate statistics file
  
  \texttt{utestats -o basename.stats basename.ute.ute}

• Generate slog file for \texttt{jumpshot} tool
  
  \texttt{slogmerge -n n basename.ute}.

• Run \texttt{jumpshot} (limit is 64 MPI processes)
  
  \texttt{module load java mpe}
  
  \texttt{jumpshot basename.slog}
jumpshot

Entire profile is divided into frames, select frame, select MPI-Process or Thread, then click "Display".
jumpshot

Basic MPI process view, use zoom buttons to see detail
jumpshot
jumpshot

Clicking on messages (red circles) or blocks produces a detailed text description:

```
Message: 'Backward Arrow'
Starts at: task ID = 0:0 & time = 3.248908 sec.
Ends  at: task ID = 1:0 & time = 3.248866 sec.
```

```
User Marker: 'Jacobi'
Starts at: task ID = 1:0 & time = 3.258939 sec.
Ends  at: task ID = 1:0 & time = 4.417081 sec.
Instruction Address = 1003e064
```
jumpshot

Thread level view:
jumpshot

To visualize statistics file. Go back to "View and Frame Selector Window", use "File->View Statistics" menu to open the "Statistics Viewer" window, and from there open previously generated "stats" file.
jumpshot
Further Reading

• **IBM PE for AIX, Operation and Use, Volume 2**
  – Chapter 3 and Appendix A
  – [http://hpcf.nersc.gov/vendor_docs/ibm/pe/am103mst.html](http://hpcf.nersc.gov/vendor_docs/ibm/pe/am103mst.html)

• **Tour of Jumpshot**
  – `/usr/common/usg/mpe/1.2.2/share/jumpshot-3/doc/TourStepByStep.pdf`
CEPBA Tools

- Run application under the control of `ompitrace`
  - Uses DPCL technology to insert probes into executable
- Merge trace files with `omp2prv`
- Visualize results with `paraver`
ompitrace

- ompitrace [-v] [-locks] [-nosw] [-r] [functions:[all|funcfile]]
  [-stderr:stderr] [-counters] FullPathPOE
  FullPathApplName [ApplPOEParams]

- Options:
  - -v         Verbose mode
  - -locks     Trace lock calls
  - -nosw      Not use the switch clock.
  - -r         Must be set if application uses threaded MPI library
  - -function  Extra functions to be traced (user code)
  - -calls     Extra calls to be traced (system code)
ompitrace

- -counters:[option,...] Trace counters automatically, options:
  - user/nouser
  - parallel/noparallel
  - calls/nocalls
  - mpi/nompi

- ompitrace -list -functions:funcfile -calls:callfile
  FullPathPOE FullPathApplName [ApplPOEParams]
  - -list List all traceable application functions/calls.

- ompitrace -hwc [counter]
  - -hwc [counter] List the event counters available for each counter.
  If a counter is selected, event are listed only for the selected ones.
Related environment variables

- **MPTRACE_ADDFUNCTIONS**: Filename containing additional functions to trace.
- **MPTRACE_DIR**: Output temporary/tracefile directory.
- **MPTRACE_BUFFER_SIZE**: Number of events allowed in thread event buffers.
- **MPTRACE_FILE_SIZE**: Maximum trace file size in Mbytes
- **MPTRACE_COUNTERS**: Hardware event traced for each hardware counter. Can use $MPTRACE_FPHWC to use same set as hpmcount command.
- **MPTRACE_LABELS**: User manually event traced labels.
ompitrace

- ompitrace is a powerful tool to capture the states and events of a program:
  - Each thread passes through many states as it executes. The two basic states are *running* and *idle*. Also recognized are states that are associated with OpenMP (thread) or MPI actions, e.g. scheduling a thread, waiting for a message, collective operation.
  - Data can be recorded at certain execution events. Typical events are the entry and exit of a threaded-routine, the entry and exit of an MPI routine. You can add the entry and exit of any user-written routines. When an event occurs, data related to event are saved along with a time stamp.
ompitrace

$ module load cepba
$ export MPTRACE_COUNTERS=${MPTRACE_FPHWC}
$ ompitrace -r -nosw -counters:parallel poisson2 -procs 4 -nodes 1
OMPItrace tool (Version 1.1)

Tracing application: poisson2

Tracing Parameters:
* Local clock
* Tracing counters: user,parallel,calls,nompi

Parameters:
  0: /usr/bin/poe
  1: /usr/common/homes/j/jcarter/poisson/poisson2
  2: -procs
  3: 4
  4: -nodes
  5: 1
ompitrace

-> Creating the target application <-
subfilter: default repo mpccc will be charged
llsubmit: Processed command file through Submit Filter:
    "/usr/common/nsgeq/etccsubfilter".
-> Expanding source files <-
    -> ( 0 ) Expanding: poisson2.f90
    -> ( 1 ) Expanding: xlfalloc.f
-> Loading module (32-bit): /usr/common/usgeqcepba2etc/openmp+mpix32_nsw <-
Loading module in MPI application ...
Module loaded.
-> Searching probes <-
-> Installing probes ( 33 ) <-
-> Activating probes <-
EndProbeCount: 2
-> Starting application <-
. . .
ompitrace: application has been traced
ompitrace: remember to merge intermediate traces by using ompir2prv tool:
    ompir2prv *.mpit -s *.sym -o poisson242434.prv
$ ompi2prv *.mpit -s *.sym -o poisson245306.prv
Getting input arguments :
   * file (0) poisson2000004530600000000000.mpit will be APPL 1 TASK 1 THREAD 1
   (Pid 45306)
   * file (1) poisson20000045306000100000.mpit will be APPL 1 TASK 2 THREAD 1
   (Pid 45306)
   * file (2) poisson2000004530600200000.mpit will be APPL 1 TASK 3 THREAD 1
   (Pid 45306)
   * file (3) poisson2000004530600300000.mpit will be APPL 1 TASK 4 THREAD 1
   (Pid 45306)
SymFile = poisson20000045306.sym (ApplNum = 0)

ompi2prv tool (Version 1.1)

Generating paraver tracefile poisson245306.prv... (Number of applications = 1 Number
   of traces = 4)

Search_CommunicatorDefinitions ...
Rewind traces ...
     ...
Paraver trace file poisson245306.prv has been generated
Paraver

- Paraver is a powerful tool to visualize states and events in tracefiles.
- Each user thread has a trace, the values of states (or events) in several thread traces can be merged into a trace for an MPI task.
- MPI tasks can be merged into an application.
paraver

Three windows (and the display window) are created.
The default display window is usually too cluttered to make out any detail. Zoom in by right clicking and choose "Zoom" from the menu. Or click the magnifying glass in the Global Controller window. Two additional clicks define the area to be displayed.
paraver

From "paraver" window, Use "Tracefiles->Trace Information->Defined States window" to bring up a window to show the state to color mapping.
paraver
paraver

- Paraver enables you to construct customized displays, but you have to become familiar with the Visualizer, Semantic and Filter modules.
- For example:
paraver

- If you graph something new, click on $F$ in the Visualizer window, and then $\text{Apply}$ to redraw with the correct scale.
paraver
paraver

- Semantic Module has many options to
  - present State Values, Event Types, and Event States to the Visualizer Module
  - merge these values going from threads to tasks to applications
paraver

- Use Paraver window "Tracefiles->Trace Information->Defined Event Type/Values Window" to see all events recorded in the tracefile.
- Suppose we are interested in displaying the two main user functions and the OpenMP versions of those functions.
paraver

- Use Filter Module to remove all Communication Events, and all User Events not relevant to functions
- Use Semantic Module to display the Function value
paraver
paraver

• The 1D and 2D Analyzer modules can be used to evaluate statistics on state values and event values.
• The 1D Analyzer evaluates a set of functions applied to the output of the semantic module.
• The 2D Analyzer produces a matrix of values, paraver objects vs. states, events, or event values.
1D Analyzer example:

- How much time per thread is spent actually computing (not thread management, not MPI, not idle)?
- Turn off all filters in Filter Module, set for thread-level Display window in Visualizer Module, use following Semantic Module settings:
- Click on 1D Analyzer button, then click on "All Trace" button in the new window
1D Analyzer

<table>
<thead>
<tr>
<th>Flow</th>
<th>Avg Semantic Val</th>
<th># Sends</th>
<th># Receives</th>
<th># Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>THREAD 1.1.1</td>
<td>0.93</td>
<td>200</td>
<td>0</td>
<td>8211</td>
</tr>
<tr>
<td>THREAD 1.1.2</td>
<td>0.90</td>
<td>0</td>
<td>0</td>
<td>3600</td>
</tr>
<tr>
<td>THREAD 1.1.3</td>
<td>0.90</td>
<td>0</td>
<td>0</td>
<td>3600</td>
</tr>
<tr>
<td>THREAD 1.1.4</td>
<td>0.90</td>
<td>0</td>
<td>0</td>
<td>3600</td>
</tr>
<tr>
<td>THREAD 1.2.1</td>
<td>0.93</td>
<td>200</td>
<td>0</td>
<td>8211</td>
</tr>
<tr>
<td>THREAD 1.2.2</td>
<td>0.90</td>
<td>0</td>
<td>0</td>
<td>3600</td>
</tr>
<tr>
<td>THREAD 1.2.3</td>
<td>0.90</td>
<td>0</td>
<td>0</td>
<td>3600</td>
</tr>
<tr>
<td>THREAD 1.2.4</td>
<td>0.90</td>
<td>0</td>
<td>0</td>
<td>3600</td>
</tr>
<tr>
<td>THREAD 1.3.1</td>
<td>0.93</td>
<td>200</td>
<td>200</td>
<td>8211</td>
</tr>
<tr>
<td>THREAD 1.3.2</td>
<td>0.90</td>
<td>0</td>
<td>0</td>
<td>3600</td>
</tr>
<tr>
<td>THREAD 1.3.3</td>
<td>0.90</td>
<td>0</td>
<td>0</td>
<td>3600</td>
</tr>
<tr>
<td>THREAD 1.3.4</td>
<td>0.90</td>
<td>0</td>
<td>0</td>
<td>3600</td>
</tr>
<tr>
<td>THREAD 1.4.1</td>
<td>0.93</td>
<td>200</td>
<td>200</td>
<td>8211</td>
</tr>
<tr>
<td>THREAD 1.4.2</td>
<td>0.90</td>
<td>0</td>
<td>0</td>
<td>3600</td>
</tr>
<tr>
<td>THREAD 1.4.3</td>
<td>0.90</td>
<td>0</td>
<td>0</td>
<td>3600</td>
</tr>
<tr>
<td>THREAD 1.4.4</td>
<td>0.90</td>
<td>0</td>
<td>0</td>
<td>3600</td>
</tr>
<tr>
<td>Total</td>
<td>14.49</td>
<td>600</td>
<td>600</td>
<td>76044</td>
</tr>
<tr>
<td>Average</td>
<td>0.91</td>
<td>50</td>
<td>50</td>
<td>4752.75</td>
</tr>
<tr>
<td>Maximum</td>
<td>0.93</td>
<td>200</td>
<td>200</td>
<td>8211</td>
</tr>
<tr>
<td>Minimum</td>
<td>0.90</td>
<td>0</td>
<td>0</td>
<td>3600</td>
</tr>
</tbody>
</table>

Results show OpenMP threads are about 90% efficient, MPI threads about 93% efficient.
Derived Views

- Power 3 hardware counter values can be visualized over the whole run.
- Mflop/s is given approximately by adding the values of the `PM_FPU0_CMPL`, `PM_FPU1_CMPL`, `PM_FPU_FMA` events.
- Create three display windows each showing the values of one counter and then add them.
Displaying First Event

- PM_FPU1_CMPL is event 42000135, so filter for this event only

- Choose "Avg Next Evt Val" for the thread Semantic Module
  - Computes last value minus first value divided by time multiplied by "Factor"
  - Units of time are indicated in Visualizer Module
Displaying First Event

- Clicking on the "Color" checkbox plots a graph instead of color coding the value.
- Repeat this procedure with PM_FPU0_CMPL and PM_FPU_FMA
Creating Derived Window

• Visualizer Module should look like this:
  – Three windows have been given a label
• Select a window, then click on "Derived"
Creating Derived Window

- Semantic Module window changes during creation of a derived view.
- Select "add" as the Operation to combine the two windows. Default is "product".
Creating Derived Window

- Add FMA window to FPU0+FPU1 window
Mflop/s 1D Analyzer View

Using the 1D Analyzer on the Mflop window produces the same kind of output as hpmcount.
Further Reading

• **CEPBA Tools**
  - Not completely updated for IBM SP
おわり