Profiling

Yaakoub El Khamra

Texas Advanced Computing Center
Disclaimer: counting flops

- Lonestar: Cannot count FLOPS accurately, but can make reasonable estimates
- Stampede (Sandy Bridge): Cannot count FLOPS accurately
  - Results always overcount, by amounts that depend on how busy the memory system is.
- Stampede (Xeon Phi): Cannot count FLOPS at all
Lonestar != Stampede

Lonestar
- Command line timing
- gprof
- Perfexpert (fairly reliable)
- IPM
- mpiP
- Tau
- HPCToolkit

Stampede
- Command line timing
- gprof
- Perfexpert (ignore metrics relying on flops)
- Tau
- Vtune Amplifier
Timers: Command Line

- The command time is available in most Unix systems.
- It is simple to use (no code instrumentation required).
- Gives total execution time of a process and all its children in seconds.

```
% /usr/bin/time -p ./exeFile
real 9.95
user 9.86
sys 0.06
```

Leave out the -p option to get additional information:
```
% time ./exeFile
% 9.860u 0.060s 0:09.95 99.9% 0+0k 0+0io 0pf+0w
```
Timers: Code Section

```fortran
INTEGER :: rate, start, stop
REAL    :: time

CALL SYSTEM_CLOCK(COUNT_RATE = rate)
CALL SYSTEM_CLOCK(COUNT = start)

! Code to time here

CALL SYSTEM_CLOCK(COUNT = stop)
time = REAL( ( stop - start )/ rate )
```

```c
#include <time.h>

double start, stop, time;
start = (double)clock()/CLOCKS_PER_SEC;

/* Code to time here */

stop = (double)clock()/CLOCKS_PER_SEC;
time = stop - start;
```
About GPROF

GPROF is the GNU Project PROFiler.

• Requires recompilation of the code.

• Compiler options and libraries provide wrappers for each routine call and periodic sampling of the program.

• Provides three types of profiles
  - Flat profile
  - Call graph
  - Annotated source

[Link to GPROF website: gnu.org/software/binutils/]
Types of Profiles

• Flat Profile
  – CPU time spend in each function (self and cumulative)
  – Number of times a function is called
  – Useful to identify most expensive routines

• Call Graph
  – Number of times a function was called by other functions
  – Number of times a function called other functions
  – Useful to identify function relations
  – Suggests places where function calls could be eliminated

• Annotated Source
  – Indicates number of times a line was executed
Profiling with gprof

Use the `-pg` flag during compilation:

```bash
% gcc -g -pg ./srcFile.c
% icc -g -p ./srcFile.c
% pgcc -g -pg ./srcFile.c
```

Run the executable. An output file `gmon.out` will be generated with the profiling information.

Execute `gprof` and redirect the output to a file:

```bash
% gprof ./exeFile gmon.out > profile.txt
% gprof -l ./exeFile gmon.out > profile_line.txt
% gprof -A ./exeFile gmon.out > profile_anotated.txt
```
Flat profile

In the flat profile we can identify the most expensive parts of the code (in this case, the calls to matSqrt, matCube, and sysCube).

<table>
<thead>
<tr>
<th>% cumulative</th>
<th>self</th>
<th>self</th>
<th>total</th>
<th>calls</th>
<th>s/call</th>
<th>s/call name</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>seconds</td>
<td>seconds</td>
<td>calls</td>
<td>s/call</td>
<td>s/call</td>
<td>name</td>
</tr>
<tr>
<td>50.00</td>
<td>2.47</td>
<td>2.47</td>
<td>2</td>
<td>1.24</td>
<td>1.24</td>
<td>matSqrt</td>
</tr>
<tr>
<td>24.70</td>
<td>3.69</td>
<td>1.22</td>
<td>1</td>
<td>1.22</td>
<td>1.22</td>
<td>matCube</td>
</tr>
<tr>
<td>24.70</td>
<td>4.91</td>
<td>1.22</td>
<td>1</td>
<td>1.22</td>
<td>1.22</td>
<td>sysCube</td>
</tr>
<tr>
<td>0.61</td>
<td>4.94</td>
<td>0.03</td>
<td>1</td>
<td>0.03</td>
<td>4.94</td>
<td>main</td>
</tr>
<tr>
<td>0.00</td>
<td>4.94</td>
<td>0.00</td>
<td>2</td>
<td>0.00</td>
<td>0.00</td>
<td>vecSqrt</td>
</tr>
<tr>
<td>0.00</td>
<td>4.94</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>1.24</td>
<td>sysSqrt</td>
</tr>
<tr>
<td>0.00</td>
<td>4.94</td>
<td>0.00</td>
<td>1</td>
<td>0.00</td>
<td>0.00</td>
<td>vecCube</td>
</tr>
</tbody>
</table>
## Call Graph Profile

<table>
<thead>
<tr>
<th>index</th>
<th>% time</th>
<th>self</th>
<th>children</th>
<th>called</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>1/1</td>
<td></td>
<td>&lt;hicore&gt; (8)</td>
<td></td>
</tr>
<tr>
<td>100.0</td>
<td>0.03</td>
<td>4.91</td>
<td>1</td>
<td>main [1]</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>1.24</td>
<td>1/1</td>
<td></td>
<td>sysSqrt [3]</td>
<td></td>
</tr>
<tr>
<td>1.24</td>
<td>0.00</td>
<td>1/2</td>
<td></td>
<td>matSqrt [2]</td>
<td></td>
</tr>
<tr>
<td>1.22</td>
<td>0.00</td>
<td>1/1</td>
<td></td>
<td>sysCube [5]</td>
<td></td>
</tr>
<tr>
<td>1.22</td>
<td>0.00</td>
<td>1/1</td>
<td></td>
<td>matCube [4]</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>1/2</td>
<td></td>
<td>vecSqrt [6]</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>1/1</td>
<td></td>
<td>vecCube [7]</td>
<td></td>
</tr>
</tbody>
</table>

---

| 1.24 | 0.00 | 1/2 | main [1] |
| 1.24 | 0.00 | 1/2 | sysSqrt [3] |
| 50.0 | 2.47 | 0.00 | 2 | matSqrt [2] |

---

| 0.00 | 1.24 | 1/1 | main [1] |
| 25.0 | 0.00 | 1.24 | 1 | sysSqrt [3] |
| 1.24 | 0.00 | 1/2 | matSqrt [2] |
| 0.00 | 0.00 | 1/2 | vecSqrt [6] |

---
Visual Call Graph

- main
- sysSqrt
- matSqrt
- vecSqrt
- matCube
- vecCube
- sysCube
## Call Graph Profile

<table>
<thead>
<tr>
<th>index</th>
<th>% time</th>
<th>self</th>
<th>children</th>
<th>called</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>1/1</td>
<td></td>
<td>&lt;hicore&gt; (8)</td>
<td></td>
</tr>
<tr>
<td>[1]</td>
<td>100.0</td>
<td>0.03</td>
<td>4.91</td>
<td>1</td>
<td>main [1]</td>
</tr>
<tr>
<td>0.00</td>
<td>1.24</td>
<td>1/1</td>
<td></td>
<td>sysSqrt [3]</td>
<td></td>
</tr>
<tr>
<td>1.24</td>
<td>0.00</td>
<td>1/2</td>
<td></td>
<td>matSqrt [2]</td>
<td></td>
</tr>
<tr>
<td>1.22</td>
<td>0.00</td>
<td>1/1</td>
<td></td>
<td>sysCube [5]</td>
<td></td>
</tr>
<tr>
<td>1.22</td>
<td>0.00</td>
<td>1/1</td>
<td></td>
<td>matCube [4]</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>1/2</td>
<td></td>
<td>vecSqrt [6]</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>1/1</td>
<td></td>
<td>vecCube [7]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1.24</td>
<td>0.00</td>
<td>1/2</td>
<td></td>
<td>main [1]</td>
<td></td>
</tr>
<tr>
<td>1.24</td>
<td>0.00</td>
<td>1/2</td>
<td></td>
<td>sysSqrt [3]</td>
<td></td>
</tr>
<tr>
<td>[2]</td>
<td>50.0</td>
<td>2.47</td>
<td>0.00</td>
<td>2</td>
<td>matSqrt [2]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>1.24</td>
<td>1/1</td>
<td></td>
<td>main [1]</td>
<td></td>
</tr>
<tr>
<td>[3]</td>
<td>25.0</td>
<td>0.00</td>
<td>1.24</td>
<td>1</td>
<td>sysSqrt [3]</td>
</tr>
<tr>
<td>1.24</td>
<td>0.00</td>
<td>1/2</td>
<td></td>
<td>matSqrt [2]</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>1/2</td>
<td></td>
<td>vecSqrt [6]</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Visual Call Graph

```
main
   |sysSqrt
   |vecSqrt
   |matCube
   |vecCube
|   |sysCube
|   |matSqrt
```

- `main`
- `sysSqrt`
- `vecSqrt`
- `matCube`
- `vecCube`
- `sysCube`
- `matSqrt`
## Call Graph Profile

<table>
<thead>
<tr>
<th>index</th>
<th>% time</th>
<th>self</th>
<th>children</th>
<th>called</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>1/1</td>
<td>&lt;hicore&gt;</td>
<td>(8)</td>
</tr>
<tr>
<td>[1] 100.0</td>
<td>0.03</td>
<td>4.91</td>
<td>1</td>
<td>main</td>
<td>[1]</td>
</tr>
<tr>
<td>0.00</td>
<td>1.24</td>
<td>0.00</td>
<td>1/1</td>
<td>sysSqrt</td>
<td>[3]</td>
</tr>
<tr>
<td>1.24</td>
<td>0.00</td>
<td>1/2</td>
<td>matSqrt</td>
<td>[2]</td>
<td></td>
</tr>
<tr>
<td>1.22</td>
<td>0.00</td>
<td>1/1</td>
<td>sysCube</td>
<td>[5]</td>
<td></td>
</tr>
<tr>
<td>1.22</td>
<td>0.00</td>
<td>1/1</td>
<td>matCube</td>
<td>[4]</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>1/2</td>
<td>vecSqrt</td>
<td>[6]</td>
<td></td>
</tr>
<tr>
<td>0.00</td>
<td>0.00</td>
<td>1/1</td>
<td>vecCube</td>
<td>[7]</td>
<td></td>
</tr>
</tbody>
</table>

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

| 1.24  | 0.00 | 1/2 | main   | [1]    |
| 1.24  | 0.00 | 1/2 | sysSqrt| [3]   |
| [2] 50.0 | 2.47 | 0.00 | 2     | matSqrt| [2]   |

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

| 0.00  | 1.24  | 1/1 | main   | [1]    |
| [3] 25.0 | 0.00 | 1.24 | 1    | sysSqrt| [3]   |
| 1.24  | 0.00  | 1/2 | matSqrt| [2]   |
| 0.00  | 0.00  | 1/2 | vecSqrt| [6]   |

-----------------------------------------------
Visual Call Graph

- main
- sysSqrt
- matSqrt
- vecSqrt
- matCube
- vecCube
- sysCube
PERF-EXPERT
About PerfExpert

• Brand new tool, locally developed at UT
• Easy to use and understand
• Great for quick profiling and for beginners
• Provides recommendation on “what to fix” in a subroutine
• Collects information from PAPI using HPCToolkit
• No MPI specific profiling, no 3D visualization, no elaborate metrics
• Combines ease of use with useful interpretation of gathered performance data
• Optimization suggestions!!!
Profiling with PerfExpert: Compilation

• Load the java, papi, and perfexpert modules:
  – module load papi perfexpert

• Compile the code with full optimization and with the -g flag:
  – mpicc -g -O3 source.c
  – mpif90 -g -O3 source.f90

• In your job submission script:
  \texttt{perfexpert\_run\_exp ./<executable name> <executable args>}
  \texttt{perfexpert 0.1 experiment-*.xml}

Threshold of 0.1 lists only functions and loops that represent $\geq 10\%$ of total runtime
### PerfExpert Analysis Output

**Loop in function main() at Integrator.c:81 (98.9% of the total runtime)**

<table>
<thead>
<tr>
<th>Ratio to total instrns</th>
<th>%</th>
<th>0 ........ .25 ........... .50 ........... .75 ........... 100</th>
</tr>
</thead>
<tbody>
<tr>
<td>- floating point</td>
<td></td>
<td>30 **                    **</td>
</tr>
<tr>
<td>- data accesses</td>
<td></td>
<td>71  *******************************************************</td>
</tr>
<tr>
<td>* GFLOPS (% max)</td>
<td></td>
<td>1 *</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Performance assessment</th>
<th>LCPI</th>
<th>good . . . . . . okay . . . . . . fair . . . . . . poor . . . . . . bad . . . . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>* overall</td>
<td>4.0</td>
<td>&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Upper bound estimates</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>* data accesses</td>
<td>33.1 &gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;</td>
</tr>
<tr>
<td>- L1d hits</td>
<td>2.2 &gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;</td>
</tr>
<tr>
<td>- L2d hits</td>
<td>2.8 &gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;</td>
</tr>
<tr>
<td>- L2d misses</td>
<td>28.1 &gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;&gt;</td>
</tr>
<tr>
<td>* instruction accesses</td>
<td>0.4 &gt;&gt;&gt;</td>
</tr>
<tr>
<td>- L1i hits</td>
<td>0.4 &gt;&gt;&gt;</td>
</tr>
<tr>
<td>- L2i hits</td>
<td>0.0 &gt;</td>
</tr>
<tr>
<td>- L2i misses</td>
<td>0.0 &gt;</td>
</tr>
<tr>
<td>* data TLB</td>
<td>0.0 &gt;</td>
</tr>
<tr>
<td>* instruction TLB</td>
<td>0.0 &gt;</td>
</tr>
<tr>
<td>* branch instructions</td>
<td>0.1 &gt;&gt;</td>
</tr>
<tr>
<td>- correctly predicted</td>
<td>0.1 &gt;&gt;</td>
</tr>
<tr>
<td>- mispredicted</td>
<td>0.0 &gt;</td>
</tr>
<tr>
<td>* floating-point instr</td>
<td>1.1 &gt;&gt;&gt;</td>
</tr>
<tr>
<td>- fast FP instr</td>
<td>1.1 &gt;&gt;&gt;</td>
</tr>
<tr>
<td>- slow FP instr</td>
<td>0.0 &gt;</td>
</tr>
</tbody>
</table>

**Overall loop performance is bad. The biggest problem is data accesses that miss in the L2 cache. Remaining performance categories are good.**
PerfExpert Summary

• Load the papi, java, and perfexpert modules:
  \% module load papi perfexpert

• In your job submission script, make sure you have:
  perfexpert_run_exp ./<executable name> <executable args>
  perfexpert 0.1 experiment-*.xml

• Send output to AutoSCOPE for optimization suggestions:
  perfexpert 0.1 experiment-integrator.xml | autoscope

• Apply suggestions from autoscope and run again. Check to see if the wall clock time
  is reduced or not
Optimization Suggestions

Code Section: Loop in function main() at Integrator.c:81 (98.9% of the total runtime)

- Change the order of loops
  
  `loop i { loop j {...} } → loop j { loop i {...} }

- Employ loop blocking
  
  `loop i {loop k {loop j {c[i][j] = c[i][j] + a[i][k] * b[k][j];}}}
  
  `loop k step s {loop j step s {loop i {
    for (kk = k; kk < k + s; kk++) {
      for (jj = j; jj < j + s; jj++) {
        c[i][jj] = c[i][jj] + a[i][kk] * b[kk][jj];
      }
    }
  }}}

- Apply loop fission so every loop accesses just a couple of different arrays
  
  `loop i {a[i] = a[i] * b[i] - c[i];}
  
  `loop i {a[i] = a[i] * b[i];} loop i {a[i] = a[i] - c[i];}
IPM: INTEGRATED PERFORMANCE MONITORING
IPM: Integrated Performance Monitoring

• “IPM is a portable profiling infrastructure for parallel codes. It provides a low-overhead performance summary of the computation and communication in a parallel program”

• IPM is a quick, easy and concise profiling tool

• The level of detail it reports is smaller than TAU, PAPI or HPCToolkit
IPM: Integrated Performance Monitoring

• IPM features:
  – easy to use
  – has low overhead
  – is scalable

• Requires no source code modification, just adding the “-g” option to the compilation

• Produces XML output that is parsed by scripts to generate browser-readable html pages
IPM: Integrated Performance Monitoring

• Available on Ranger and Lonestar with the mvapich libraries
• Create ipm environment with module command before running code: “module load ipm”
• In your job script, set up the following ipm environment before the ibrun command:

  module load ipm
  export LD_PRELOAD=$TACC_IPM_LIB/libipm.so
  export IPM_REPORT=full

  ibrun <my executable> <my arguments>
IPM: Integrated Performance Monitoring

- Export LD_PRELOAD=$TACC_IPM_LIB/libipm.so
  - must be inside job script
- **IPM_REPORT**: full, terse or none are the levels of information
- **IPM_MPI_THRESHOLD**: Reports only routines using this percentage (or more) of MPI time.
  - e.g. "**IPM_MPI_THRESHOLD 0.3**" report subroutines that consume more than 30% of the total MPI time.
- Important details: "module help ipm"
IPM: Text Output

command: /work/01125/yye00/ICAC/cactus_SandTank SandTank.par
host: i101-x86_64_Linux
mpi_tasks: 32 on 2 nodes
start: 05/26/09/11:49:06 wallclock: 2.758892 sec
stop: 05/26/09/11:49:09 %comm: 2.01
gbytes: 4.38747e+00 total
gflop/sec: 9.39108e-02 total

region: *

entries	[tasks] = 32

[total] <avg> min max

wallclock: 88.2742 2.75857 2.75816 2.75889
user: 5.51634 0.172386 0.148009 0.200012
system: 1.771 0.0553438 0.0536683 0.056717
%comm: 2.00602 1.94539 2.05615
gflop/sec: 0.0939108 0.00293471 0.00293338 0.002952

gbytes: 4.38747 0.137109 0.136581 0.144985

PAPI_FP_OPS 2.5909e+08 8.09655e+06 8.09289e+06 8.14685e+06
PAPI_TOT_CYC 6.80291e+09 2.12591e+08 2.02236e+08 2.19109e+08
PAPI_VEC_INS 5.95596e+08 1.86124e+07 1.85964e+07 1.8756e+07
PAPI_TOT_INS 4.16377e+09 1.30118e+08 1.0987e+08 1.35676e+08

MPI_Allreduce 0.978938 53248 55.28 1.11
MPI_Comm_rank 0.316355 6002 17.86 0.36
MPI_Barrier 0.247135 3680 13.95 0.28
MPI_Allgatherv 0.16621 2848 9.39 0.19
MPI_Bcast 0.0217298 576 1.23 0.02
MPI_Allgather 0.0216982 672 1.23 0.02
MPI_Recv 0.0186796 32 1.05 0.02
MPI_Comm_size 0.000139921 2112 0.01 0.00
MPI_Send 0.000115622 32 0.01 0.00
# IPM: Integrated Performance Monitoring

**Command:**
```
command: /work/01125/yje004/IPM/Benchmark/Defiant.exe perurb -runpermurbed -da_grid_x 50 -da_grid_y 50 -da_grid_z 50 -seed_phi 3454345 -seed_k11 56756756 -seed_k22 235759 -seed_k33 234656 -seed_flowmask 3222111 -percentage_phi 0.1 -percentage_k11 0.1 -percentage_k22 0.1 -percentage_k33 0.1 -percentage_flowmask 0.15 -endtime 1.0 -ksp_type bicg -pc_type bjacobi
```

- **Codename:** unknown
- **State:** running
- **Username:** yje004
- **Group:** G-801077
- **Host:** 1115-108 (x86_64 Linux)
- **MPI Tasks:** 64 on 4 hosts
- **Start:** 07/13/10:00:28:10
- **Wallclock:** 3.80580e+00 sec
- **Stop:** 07/13/10:00:28:13
- **Percent Comm:** 11.575290360397
- **Total Memory:** 10.85/65 gbytes
- **Total GFlop/sec:** 1.0219290310653
- **Switch(Send):** 0 gbytes
- **Switch(Recv):** 0 gbytes

## Computation

<table>
<thead>
<tr>
<th>Event</th>
<th>Count</th>
<th>Pop</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAPI_FP_OPS</td>
<td>3889256866</td>
<td>*</td>
</tr>
<tr>
<td>PAPI_TOT_CYC</td>
<td>93055641837</td>
<td>*</td>
</tr>
<tr>
<td>PAPI_TOT_INS</td>
<td>82058705179</td>
<td>*</td>
</tr>
<tr>
<td>PAPI_VEC_INS</td>
<td>8293137711</td>
<td>*</td>
</tr>
</tbody>
</table>

## Communication

% of MPI Time

<table>
<thead>
<tr>
<th>Event</th>
<th>Ntasks</th>
<th>Avg</th>
<th>Min(rank)</th>
<th>Max(rank)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAPI_FP_OPS</td>
<td>*</td>
<td>607696348.53</td>
<td>53254674 (63)</td>
<td>68822066 (21)</td>
</tr>
<tr>
<td>PAPI_TOT_CYC</td>
<td>*</td>
<td>1453994403.70</td>
<td>1346848050 (23)</td>
<td>1646491906 (12)</td>
</tr>
<tr>
<td>PAPI_TOT_INS</td>
<td>*</td>
<td>1208217628.42</td>
<td>1134309464 (0)</td>
<td>1477795580 (12)</td>
</tr>
<tr>
<td>PAPI_VEC_INS</td>
<td>*</td>
<td>12958276.73</td>
<td>113002002 (63)</td>
<td>146915633 (21)</td>
</tr>
</tbody>
</table>
### IPM: Event Statistics

<table>
<thead>
<tr>
<th>Communication Event Statistics (100.00% detail, 9.9012e-06 error)</th>
<th>Buffer Size</th>
<th>Ncalls</th>
<th>Total Time</th>
<th>Min Time</th>
<th>Max Time</th>
<th>*%MPI</th>
<th>*%Wall</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI_Allreduce</td>
<td>8</td>
<td>79680</td>
<td>4.178</td>
<td>8.225e-06</td>
<td>8.882e-04</td>
<td>14.82</td>
<td>1.72</td>
</tr>
<tr>
<td>MPI_Bcast</td>
<td>4</td>
<td>1024</td>
<td>4.047</td>
<td>5.914e-08</td>
<td>6.413e-02</td>
<td>14.35</td>
<td>1.66</td>
</tr>
<tr>
<td>MPI_Allreduce</td>
<td>512</td>
<td>39336</td>
<td>3.803</td>
<td>1.660e-05</td>
<td>1.170e-01</td>
<td>13.49</td>
<td>1.56</td>
</tr>
<tr>
<td>MPI_Allreduce</td>
<td>4</td>
<td>25472</td>
<td>2.250</td>
<td>6.012e-07</td>
<td>1.552e-02</td>
<td>7.98</td>
<td>0.92</td>
</tr>
<tr>
<td>MPI_Barrier</td>
<td>0</td>
<td>64</td>
<td>1.176</td>
<td>1.014e-02</td>
<td>1.865e-02</td>
<td>4.17</td>
<td>0.48</td>
</tr>
<tr>
<td>MPI_Isend</td>
<td>8</td>
<td>630</td>
<td>1.028</td>
<td>3.427e-07</td>
<td>1.647e-02</td>
<td>3.65</td>
<td>0.42</td>
</tr>
<tr>
<td>MPI_Isend</td>
<td>4</td>
<td>4556</td>
<td>0.943</td>
<td>2.738e-07</td>
<td>1.833e-02</td>
<td>3.34</td>
<td>0.39</td>
</tr>
<tr>
<td>MPI_Send</td>
<td>14976</td>
<td>144</td>
<td>0.722</td>
<td>1.030e-03</td>
<td>7.308e-03</td>
<td>2.56</td>
<td>0.30</td>
</tr>
<tr>
<td>MPI_Comm_rank</td>
<td>0</td>
<td>106948</td>
<td>0.620</td>
<td>3.725e-08</td>
<td>9.872e-03</td>
<td>2.20</td>
<td>0.25</td>
</tr>
<tr>
<td>MPI_Waitany</td>
<td>0</td>
<td>6093</td>
<td>0.542</td>
<td>4.615e-07</td>
<td>1.358e-02</td>
<td>1.92</td>
<td>0.22</td>
</tr>
<tr>
<td>MPI_Waitany</td>
<td>1248</td>
<td>27462</td>
<td>0.519</td>
<td>5.183e-07</td>
<td>2.723e-04</td>
<td>1.84</td>
<td>0.21</td>
</tr>
<tr>
<td>MPI_Send</td>
<td>16224</td>
<td>144</td>
<td>0.517</td>
<td>4.283e-04</td>
<td>7.129e-03</td>
<td>1.83</td>
<td>0.21</td>
</tr>
<tr>
<td>MPI_Waitany</td>
<td>1352</td>
<td>20370</td>
<td>0.496</td>
<td>5.197e-07</td>
<td>5.783e-03</td>
<td>1.76</td>
<td>0.20</td>
</tr>
<tr>
<td>MPI_Start</td>
<td>0</td>
<td>269196</td>
<td>0.396</td>
<td>3.623e-07</td>
<td>3.685e-05</td>
<td>1.40</td>
<td>0.16</td>
</tr>
<tr>
<td>MPI_Send</td>
<td>13824</td>
<td>48</td>
<td>0.298</td>
<td>4.935e-03</td>
<td>7.227e-03</td>
<td>1.06</td>
<td>0.12</td>
</tr>
<tr>
<td>MPI_Waitany</td>
<td>1152</td>
<td>10890</td>
<td>0.243</td>
<td>5.383e-07</td>
<td>2.310e-04</td>
<td>0.86</td>
<td>0.10</td>
</tr>
<tr>
<td>MPI_Bcast</td>
<td>216</td>
<td>576</td>
<td>0.231</td>
<td>2.302e-06</td>
<td>5.843e-03</td>
<td>0.82</td>
<td>0.09</td>
</tr>
<tr>
<td>MPI_Allgather</td>
<td>4</td>
<td>9080</td>
<td>0.215</td>
<td>5.118e-07</td>
<td>1.793e-03</td>
<td>0.76</td>
<td>0.09</td>
</tr>
<tr>
<td>MPI_Waitall</td>
<td>184</td>
<td>11</td>
<td>0.210</td>
<td>1.633e-02</td>
<td>2.135e-02</td>
<td>0.74</td>
<td>0.09</td>
</tr>
<tr>
<td>MPI_Scan</td>
<td>4</td>
<td>384</td>
<td>0.144</td>
<td>2.259e-05</td>
<td>1.600e-03</td>
<td>0.51</td>
<td>0.06</td>
</tr>
<tr>
<td>MPI_Waitany</td>
<td>147</td>
<td>453</td>
<td>0.141</td>
<td>4.866e-07</td>
<td>1.406e-02</td>
<td>0.50</td>
<td>0.06</td>
</tr>
<tr>
<td>MPI_Waitany</td>
<td>4</td>
<td>448</td>
<td>0.132</td>
<td>4.345e-07</td>
<td>5.805e-03</td>
<td>0.47</td>
<td>0.05</td>
</tr>
<tr>
<td>MPI_Waitall</td>
<td>320</td>
<td>18</td>
<td>0.120</td>
<td>3.002e-06</td>
<td>1.284e-02</td>
<td>0.42</td>
<td>0.05</td>
</tr>
<tr>
<td>MPI_Send</td>
<td>17576</td>
<td>42</td>
<td>0.108</td>
<td>6.682e-05</td>
<td>6.406e-03</td>
<td>0.38</td>
<td>0.04</td>
</tr>
<tr>
<td>MPI_Waitall</td>
<td>72</td>
<td>6</td>
<td>0.103</td>
<td>1.547e-02</td>
<td>2.038e-02</td>
<td>0.36</td>
<td>0.04</td>
</tr>
<tr>
<td>MPI_Waitall</td>
<td>96</td>
<td>38</td>
<td>0.091</td>
<td>2.882e-06</td>
<td>1.563e-02</td>
<td>0.32</td>
<td>0.04</td>
</tr>
<tr>
<td>MPI_Waitany</td>
<td>624</td>
<td>140</td>
<td>0.088</td>
<td>1.126e-06</td>
<td>7.880e-03</td>
<td>0.31</td>
<td>0.04</td>
</tr>
<tr>
<td>MPI_Recv</td>
<td>8</td>
<td>9</td>
<td>0.085</td>
<td>8.373e-07</td>
<td>7.969e-03</td>
<td>0.30</td>
<td>0.03</td>
</tr>
</tbody>
</table>
IPM Buffer Size Distribution: % of Comm Time
Buffer Size Distribution: Ncalls

Message Buffer Size Distributions: Ncalls

Buffer size (bytes) vs. Ncalls for various MPI operations. The graph shows the cumulative distribution of buffer sizes for different MPI operations, indicating how often messages of various sizes are exchanged. Cumulative values are shown at the bottom of the graph.
Communication Topology: point to point data flow
IPM: Integrated Performance Monitoring

• When to use IPM?
  – To quickly find out *where your code is spending most of its time* (in both computation and communication)
  – For performing *scaling studies* (both strong and weak)
  – When you suspect you have *load imbalance* and want to verify it quickly
  – For a quick look at the *communication pattern*
  – To find out how much *memory* you are using *per task*
  – To find the *relative communication & compute time*
IPM: Integrated Performance Monitoring

• When IPM is NOT the answer
  – When you already know where the performance issues are
  – When you need detailed performance information on exact lines of code
  – When want to find specific information such as cache misses
Advanced Profiling Tools
: the next level
Advanced Profiling Tools

• Can be intimidating:
  – Difficult to install
  – Many dependences
  – Require kernel patches

• Useful for serial and parallel programs

• Extensive profiling and scalability information

• Analyze code using:
  – Timers
  – Hardware registers (PAPI)
  – Function wrappers

} Not your problem!!
PAPI

PAPI is a Performance Application Programming Interface (PAPI)

icl.cs.utk.edu/papi

- API to use hardware counters
- Behind Tau, HPCToolkit
- Multiplatform:
  - Most Intel & AMD chips
  - IBM POWER 4/5/6
  - Cray X/XD/XT
  - Sun UltraSparc I/II/III
  - MIPS
  - SiCortex
  - Cell
- Available as a module on Lonestar and Stampede (mind the hardware counters issue with sandybridge xeon)
Tau

TAU is a suite of Tuning and Analysis Utilities
www.cs.uoregon.edu/research/tau

• 12+ year project involving
  – University of Oregon Performance Research Lab
  – LANL Advanced Computing Laboratory
  – Research Centre Julich at ZAM, Germany

• Integrated toolkit
  – Performance instrumentation
  – Measurement
  – Analysis
  – Visualization
Tau: Measurements

• Parallel profiling
  – Function-level, block (loop)-level, statement-level
  – Supports user-defined events
  – TAU parallel profile data stored during execution
  – Hardware counter values (multiple counters)
  – Support for callgraph and callpath profiling

• Tracing
  – All profile-level events
  – Inter-process communication events
  – Trace merging and format conversion
PDT is used to instrument your code.

Replace `mpicc` and `mpif90` in make files with `tau_f90.sh` and `tau_cc.sh`

It is necessary to specify all the components that will be used in the instrumentation (mpi, openmp, profiling, counters [PAPI], etc. However, these come in a limited number of combinations.)

Combinations: First determine what you want to do (profiling, PAPI counters, tracing, etc.) and the programming paradigm (mpi, openmp), and the compiler. PDT is a required component:

<table>
<thead>
<tr>
<th>Instrumentation</th>
<th>Parallel Paradigm</th>
<th>Collectors</th>
<th>Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDT</td>
<td>MPI</td>
<td>PAPI</td>
<td>Intel</td>
</tr>
<tr>
<td>Hand-coded</td>
<td>OMP</td>
<td>Callpath</td>
<td>PGI</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>GNU (gcc)</td>
<td></td>
</tr>
</tbody>
</table>
Tau: Instrumentation

You can view the available combinations
(alias tauTypes 'ls -C1 $TAU | grep Makefile ').

Your selected combination is made known to the compiler wrapper through the TAU_MAKEFILE environment variable.

E.g. the PDT instrumentation (pdt) for the Intel compiler (icpc) with MPI (mpi) is set with this command:

    setenv TAU_MAKEFILE   $TAU/Makefile.tau-icpc-mpi-pdt

Other run-time and instrumentation options are set through TAU_OPTIONS. For verbose:

    setenv TAU_OPTIONS   ‘-optVerbose’
Tau Paraprof Overview

- Raw files
- PerfDMF managed (database)
- Application
- Experiment
- Trials

HPMToolkit
Metadata
MpiP
TAU
Tau Paraprof Manager Window

Provides Machine Details
Organizes Runs as: Applications, Experiments and Trials.
Routine Time Experiment

Profile Information is in “GET_TIME_OF_DAY” metric
Mean and Standard Deviation Statistics given.
Multiply_Matrices Routine Results

Function Data Window gives a closer look at a single function:

<table>
<thead>
<tr>
<th>Name: MULTIPLY_MATRICES [(matmulf90) (25,18)]</th>
<th>Metric Name: GET_TIME_OF_DAY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value: Exclusive</td>
<td>Units: seconds</td>
</tr>
</tbody>
</table>

```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>n.c.t 12.0</td>
<td>n.c.t 13.0</td>
<td>n.c.t 14.0</td>
<td>n.c.t 15.0</td>
<td>n.c.t 19.0</td>
<td>n.c.t 19.0</td>
<td>mean</td>
<td>n.c.t 15.0</td>
<td>n.c.t 15.0</td>
<td>n.c.t 17.0</td>
<td>n.c.t 10.0</td>
<td>n.c.t 8.0</td>
<td>n.c.t 6.0</td>
<td>n.c.t 11.0</td>
<td>n.c.t 14.0</td>
</tr>
<tr>
<td>Std. Dev</td>
<td>0.051</td>
<td>0.051</td>
<td>0.051</td>
<td>0.051</td>
<td>0.051</td>
<td>0.051</td>
<td>0.051</td>
<td>0.051</td>
<td>0.051</td>
<td>0.051</td>
<td>0.051</td>
<td>0.051</td>
<td>0.051</td>
<td>0.051</td>
<td>0.051</td>
</tr>
</tbody>
</table>
```
Float Point OPS trial

Hardware Counters provide Floating Point Operations (Function Data view).
L1 Data Cache Miss trial

Hardware Counters provide L1 Cache Miss Operations.
Call Path

Call Graph Paths (Must select through “thread” menu.)
Call Path

TAU_MAKEFILE =
…Makefile.tau-callpath-icpc-mpi-pdt
Derived Metrics

Select Argument 1 (green ball); Select Argument 2 (green ball); Select Operation; then Apply. Derived Metric will appear as a new trial.
Be careful→ even though ratios are constant, cores may do different amounts of work/operations per call.

Since FP/Miss ratios are constant—must be memory access problem.
VTUNE AMPLIFIER
Vtune

- This is an Intel product
- Free for non-commercial use
- Available only on Stampede: `module load vtune`
- Will support profiling on the Xeon Phi for Stampede
- Two part interface: command line and GUI
- MPI routines are like any other routines (i.e., knows about threading and OpenMP, not MPI) but it does understand MPI ranks (for per-rank profile result output)
How to use vtune

• Three step process:
  – Compile your code with “-g” and full optimization
  – Run with amplxe-cl to collect performance data
  – Visualize performance data with amplxe-gui

• For non-MPI codes with amplxe-cl:
  amplxe-cl -collect hotspots ./ex1

• For MPI codes, you can run amplxe-cl with ibrun:
  ibrun amplxe-cl -collect hotspots -result-dir "./$(hostname)" ./ex1
# Analysis Types on Stampede

## Host
- concurrency: Concurrency
- frequency: CPU Frequency
- hotspots: Hotspots
- lightweight-hotspots: Lightweight Hotspots
- locksandwaits: Locks and Waits
- sleep: CPU Sleep States
- snb-access-contention: Access Contention
- snb-bandwidth: Bandwidth
- snb-branch-analysis: Branch Analysis
- snb-client: Client Analysis
- snb-core-port-saturation: Core Port Saturation
- snb-cycles-uops: Cycles and uOps
- snb-general-exploration: General Exploration
- snb-memory-access: Memory Access
- snb-port-saturation: Port Saturation

## Xeon Phi
- knc-bandwidth: Bandwidth (Knights Corner Platform)
- knc-general-exploration: General Exploration (Knights Corner Platform)
- knc-lightweight-hotspots: Lightweight Hotspots (Knights Corner Platform)
Useful command line tools

• To see what analysis types are available, and options for their collection:
  `amplxe-cl -help collect`
  `amplxe-cl -help collect < insert analysis type >`

• General purpose, 1st pass analysis type: hotspots
• 2nd pass: concurrency to check for sync overhead, identify potential candidates for parallelization
Sample job submission script

#!/bin/zsh
#SBATCH -A TG-STA110013S
#SBATCH -p development
#SBATCH -t 00:50:00
set +x
module swap impi mvapich2
module load vtune

ibrun amplxe-cl -collect hotspots -result-dir "./hotspots_$(hostname)" ./ex1

ibrun amplxe-cl -collect concurrency -result-dir "./concurrency_$(hostname)" ./ex1
Sample output from amplxe-cl

<table>
<thead>
<tr>
<th>Result Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>Application Command Line</td>
</tr>
<tr>
<td>CPU Name</td>
</tr>
<tr>
<td>Computer Name</td>
</tr>
<tr>
<td>Frequency</td>
</tr>
<tr>
<td>Logical CPU Count</td>
</tr>
<tr>
<td>MPI Process Rank</td>
</tr>
<tr>
<td>Operating System</td>
</tr>
<tr>
<td>Result Size</td>
</tr>
<tr>
<td>User Name</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Elapsed Time: 4.555</td>
</tr>
<tr>
<td>CPU Time: 2.330</td>
</tr>
<tr>
<td>CPU Usage: 0.473</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Result Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter</td>
</tr>
<tr>
<td>Application Command Line</td>
</tr>
<tr>
<td>CPU Name</td>
</tr>
<tr>
<td>Computer Name</td>
</tr>
<tr>
<td>Environment Variables</td>
</tr>
<tr>
<td>Frequency</td>
</tr>
<tr>
<td>Logical CPU Count</td>
</tr>
<tr>
<td>MPI Process Rank</td>
</tr>
<tr>
<td>Operating System</td>
</tr>
<tr>
<td>Result Size</td>
</tr>
<tr>
<td>User Name</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Concurrency: 0.650</td>
</tr>
<tr>
<td>Elapsed Time: 3.748</td>
</tr>
<tr>
<td>CPU Time: 2.440</td>
</tr>
<tr>
<td>Wait Time: 3.876</td>
</tr>
<tr>
<td>CPU Usage: 0.636</td>
</tr>
</tbody>
</table>
**Hotspots**

**Elapsed Time:** 6.467s

- **Total Thread Count:** 2
- **Overhead Time:** 0s
- **Spin Time:** 0.140s
- **CPU Time:** 2.550s
- **Paused Time:** 0s

**Top Hotspots**

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

<table>
<thead>
<tr>
<th>Function</th>
<th>CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>PetscIsinFV</td>
<td>0.230s</td>
</tr>
<tr>
<td>VecMDot_Norm</td>
<td>0.220s</td>
</tr>
<tr>
<td>MatMulti_SeqAll</td>
<td>0.210s</td>
</tr>
<tr>
<td>PetscMallocV</td>
<td>0.180s</td>
</tr>
<tr>
<td>VecMAXPY_Seq</td>
<td>0.140s</td>
</tr>
<tr>
<td><strong>[Others]</strong></td>
<td>1.570s</td>
</tr>
</tbody>
</table>

**CPU Usage Histogram**

This histogram represents a breakdown of the Elapsed Time. It visualizes what percentage of the wall time the specific number of CPUs were running simultaneously. CPU Usage may be higher than the thread concurrency if a thread is executing code on a CPU while it is logically waiting.

**Collection and Platform Info**

This section provides information about this collection, including result set size and collection platform data.

- **Application Command Line:** ex3 -m "*640"
- **Frequency:** 2.7 GHz
- **Logical CPU Count:** 16
- **CPU Name:** Intel(R) Xeon(R) E5 processor
- **Operating System:** 2.6.32-358.el6.x86_64 CentOS release 6.4 (Final)
- **MPI Process Rank:** 10
- **Computer Name:** c558-404.stampede.tacc.utexas.edu
- **Result Site:** TACC
<table>
<thead>
<tr>
<th>Function / Call Stack</th>
<th>CPU Time by Utilization</th>
<th>Over</th>
<th>Spine Time</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>PetscSolveSeqNonSca</td>
<td>229.93ms</td>
<td>0ms</td>
<td>0ms</td>
<td>libpetsc.so</td>
</tr>
<tr>
<td>VecMDot_Seq</td>
<td>220.153ms</td>
<td>0ms</td>
<td>0ms</td>
<td>VecMDot_Seq</td>
</tr>
<tr>
<td>MatMult_SeqAij</td>
<td>210.064ms</td>
<td>0ms</td>
<td>0ms</td>
<td>MatMult_SeqAij</td>
</tr>
<tr>
<td>PetscMIntValidate</td>
<td>179.939ms</td>
<td>0ms</td>
<td>0ms</td>
<td>PetscMIntValidate</td>
</tr>
<tr>
<td>VecMAXPY_Seq</td>
<td>149.023ms</td>
<td>0ms</td>
<td>0ms</td>
<td>VecMAXPY_Seq</td>
</tr>
<tr>
<td>_isinf</td>
<td>129.936ms</td>
<td>0ms</td>
<td>0ms</td>
<td>libpetsc.so</td>
</tr>
<tr>
<td>MatSolve_SeqAijNatural</td>
<td>120.054ms</td>
<td>0ms</td>
<td>0ms</td>
<td>MatSolve_SeqAij</td>
</tr>
<tr>
<td>vyscall[]</td>
<td>109.997ms</td>
<td>0ms</td>
<td>0ms</td>
<td>vyscall[]</td>
</tr>
<tr>
<td>MPIDI_CH3I_SMP_readpro</td>
<td>100.009ms</td>
<td>0ms</td>
<td>0ms</td>
<td>libmich.so.8.0.1</td>
</tr>
<tr>
<td>MPIDI_CH3I_SMP_pullhead</td>
<td>89.887ms</td>
<td>0ms</td>
<td>0ms</td>
<td>libmich.so.8.0.1</td>
</tr>
<tr>
<td>MPIDI_CH3I_SMP_init</td>
<td>80.002ms</td>
<td>0ms</td>
<td>0ms</td>
<td>libmich.so.8.0.1</td>
</tr>
<tr>
<td>PCApply</td>
<td>79.973ms</td>
<td>0ms</td>
<td>0ms</td>
<td>libpetsc.so</td>
</tr>
<tr>
<td>PCApplyBAORB</td>
<td>69.983ms</td>
<td>0ms</td>
<td>0ms</td>
<td>libpetsc.so</td>
</tr>
<tr>
<td>look_sysfscpu</td>
<td>50.005ms</td>
<td>0ms</td>
<td>0ms</td>
<td>libmich.so.8.0.1</td>
</tr>
<tr>
<td>[libmlx4-rdma2v2.so]</td>
<td>50.002ms</td>
<td>0ms</td>
<td>19.999ms</td>
<td>libmlx4-rdma2v2.so</td>
</tr>
<tr>
<td>_IO_file_read</td>
<td>50.000ms</td>
<td>0ms</td>
<td>50.000ms</td>
<td>libc-2.3.4.so</td>
</tr>
<tr>
<td>_video_gettimeofday</td>
<td>49.002ms</td>
<td>0ms</td>
<td>0ms</td>
<td>[video]</td>
</tr>
<tr>
<td>_video_gettimeofday</td>
<td>49.002ms</td>
<td>0ms</td>
<td>0ms</td>
<td>[video]</td>
</tr>
</tbody>
</table>

Selected 1 row(s): 229.93ms 0ms 0ms
**Elapsed Time:** 4.036s

- Total Thread Count: 2
- Overhead Time: 0s
- Spin Time: 0.030s
- CPU Time: 2.429s
- Paused Time: 0s

**Top Hotspots**

This section lists the most active functions in your application. Optimizing these hotspot functions typically results in improving overall application performance.

<table>
<thead>
<tr>
<th>Function</th>
<th>CPU Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>VecMDot_Seq</td>
<td>0.280s</td>
</tr>
<tr>
<td>PetscIsinorNanScalar</td>
<td>0.200s</td>
</tr>
<tr>
<td>MatSolve_SeqAI_NaturalOrdering</td>
<td>0.170s</td>
</tr>
<tr>
<td>PetscMallocValidate</td>
<td>0.160s</td>
</tr>
<tr>
<td>MatMult_SeqAI</td>
<td>0.130s</td>
</tr>
<tr>
<td>[Others]</td>
<td>1.489s</td>
</tr>
</tbody>
</table>

**Thread Concurrency Histogram**

This histogram represents a breakdown of the Elapsed Time. It visualizes the percentage of the wall time the specific number of threads were running simultaneously. Threads are considered running if they are either actually running on a CPU or are in the runnable state in the OS scheduler. Essentially, Thread Concurrency is a measurement of the number of threads that were not waiting. Thread Concurrency may be higher than CPU usage if threads are in the runnable state and not consuming CPU time.

**CPU Usage Histogram**

This histogram represents a breakdown of the Elapsed Time. It visualizes what percentage of the wall time the specific number of CPUs were running simultaneously. CPU Usage may be higher than the thread concurrency if a thread is executing code on a CPU while it is logically waiting.
### Concurrency Hotspots by Thread

#### Concurrency viewpoint (change)

#### Source: Assembly

<table>
<thead>
<tr>
<th>Address</th>
<th>So. Line</th>
<th>Assembly</th>
<th>CPU Time by Utilization</th>
<th>Ove. Spi.</th>
<th>Wait Time by Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Idle</td>
<td>Poor</td>
<td>Ok</td>
</tr>
<tr>
<td>0x2e7048</td>
<td>0</td>
<td>mulsd wmm1, wmm10</td>
<td>30.044ms</td>
<td>0ms</td>
<td>0ms</td>
</tr>
<tr>
<td>0x2e7079</td>
<td>0</td>
<td>mulsd wmm5, wmm12</td>
<td>20.021ms</td>
<td>0ms</td>
<td>0ms</td>
</tr>
<tr>
<td>0x2e7078</td>
<td>0</td>
<td>add wmm1, wmm11, wmm6</td>
<td>20.020ms</td>
<td>0ms</td>
<td>0ms</td>
</tr>
<tr>
<td>0x2e7094</td>
<td>0</td>
<td>add wmm1, wmm13, wmm7</td>
<td>20.017ms</td>
<td>0ms</td>
<td>0ms</td>
</tr>
<tr>
<td>0x2e7125</td>
<td>0</td>
<td>mul wmm2, wmm12</td>
<td>20.014ms</td>
<td>0ms</td>
<td>0ms</td>
</tr>
<tr>
<td>0x2e70dc</td>
<td>0</td>
<td>add wmm1, wmm11</td>
<td>20.006ms</td>
<td>0ms</td>
<td>0ms</td>
</tr>
<tr>
<td>0x2e70b7</td>
<td>0</td>
<td>movsq (wrcx), wmm8</td>
<td>10.017ms</td>
<td>0ms</td>
<td>0ms</td>
</tr>
<tr>
<td>0x2e7104</td>
<td>0</td>
<td>add wmm11, wmm11</td>
<td>10.013ms</td>
<td>0ms</td>
<td>0ms</td>
</tr>
<tr>
<td>0x2e703d</td>
<td>0</td>
<td>add wmm5, wmm11</td>
<td>10.009ms</td>
<td>0ms</td>
<td>0ms</td>
</tr>
<tr>
<td>0x2e7220</td>
<td>0</td>
<td>movsq 0x38(wrdx), wmm2</td>
<td>10.009ms</td>
<td>0ms</td>
<td>0ms</td>
</tr>
<tr>
<td>0x2e700e</td>
<td>0</td>
<td>movsq 0x38(wrdx), wmm2</td>
<td>10.006ms</td>
<td>0ms</td>
<td>0ms</td>
</tr>
<tr>
<td>0x2e70c6</td>
<td>0</td>
<td>movsq 0x38(wrdx), wmm2</td>
<td>10.005ms</td>
<td>0ms</td>
<td>0ms</td>
</tr>
<tr>
<td>0x2e55ff</td>
<td>0</td>
<td>mul wmm5, wmm5</td>
<td>10.005ms</td>
<td>0ms</td>
<td>0ms</td>
</tr>
<tr>
<td>0x2e708e</td>
<td>0</td>
<td>movsq 0x38(wrdx), wmm14</td>
<td>10.004ms</td>
<td>0ms</td>
<td>0ms</td>
</tr>
<tr>
<td>0x2e713e</td>
<td>0</td>
<td>mov wrdx, 0x60(%rsp)</td>
<td>10.000ms</td>
<td>0ms</td>
<td>0ms</td>
</tr>
</tbody>
</table>

---

**Intel VTune Amplifier XE 2013**

---

**CPU Function/CPU Stack - CPU**

**Viewing 1 of 1 selected stack**

0.000% (0.000s of 0.000s)

**libpetsc.so**

libpetsc.so:VecMDDot_Seq - [U...

libpetsc.so:VecMDDot_MPI+0x31...

libpetsc.so:VecMDDot_t+0x477 - ...

libpetsc.so:KSPSolve Classical...

libpetsc.so:KSPSolveClassicCycle+

libpetsc.so:KSPSolve_GMRES...

libpetsc.so:KSPSolve+0x9a0 - ...

4x3main+0xd77 - [Unknown]...

libc.so.1.2.3.so:libc_start_main...
Profiling dos and don’ts

**DO**

- Test every change you make
- Profile typical cases
- Compile with optimization flags
- Test for scalability

**DO NOT**

- Assume a change will be an improvement
- Profile atypical cases
- Profile *ad infinitum*
  - Set yourself a goal or
  - Set yourself a time limit
Other tools

• Valgrind* [valgrind.org] – Powerful instrumentation framework, often used for debugging memory problems

• MPIP [mpip.sourceforge.net] – Lightweight, scalable MPI profiling tool

• Tau [www.cs.uoregon.edu/research/tau] – Suite of Tuning and Analysis Utilities

• Scalasca [www.fz-juelich.de/jsc/scalasca] – Similar to Tau, complete suit of tuning and analysis tools.

• HPCToolkit [www.hpctoolkit.org] – Interesting tool with a lot of promise
HPCToolkit

- “HPCToolkit is an open-source suite of multi-platform tools for profile-based performance analysis of applications. The figure below provides an overview of the toolkit components and their relationships.”

- HPCToolkit is under intense development and shows great potential

- HPCToolkit is a mid-range tool: providing detailed profiling information without the need to rewrite code
HPCToolkit: Overview
HPCToolkit: Overview

• To prepare the executable for profiling, compile the code with full optimization as usual but include the “-g” option

• Using HPCToolkit involves four stages:
  – We start by performing a static binary analysis of the compiled executable (hence the need for the “-g” option) using “hpcstruct”
  – You run the code with “hpcrun” to collect the stack profile
  – You interpret the stack profile and correlate it with the source using “hpcprof” to generate the profiling database
  – You view the profiling database with the “hpcviewer”
## HPCToolkit: Available Events

<table>
<thead>
<tr>
<th>Counter/Event Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAPI_L1_DCM</td>
<td>Level 1 data cache misses</td>
</tr>
<tr>
<td>PAPI_L1_ICM</td>
<td>Level 1 instruction cache misses</td>
</tr>
<tr>
<td>PAPI_L2_DCM</td>
<td>Level 2 data cache misses</td>
</tr>
<tr>
<td>PAPI_L2_ICM</td>
<td>Level 2 instruction cache misses</td>
</tr>
<tr>
<td>PAPI_L2_TCM</td>
<td>Level 2 cache misses</td>
</tr>
<tr>
<td>PAPI_L3_TCM</td>
<td>Level 3 cache misses</td>
</tr>
<tr>
<td>PAPI_FPU_JDL</td>
<td>Cycles floating point units are idle</td>
</tr>
<tr>
<td>PAPI_TLB_DM</td>
<td>Data translation lookaside buffer misses</td>
</tr>
<tr>
<td>PAPI_TLB.IM</td>
<td>Instruction translation lookaside buffer misses</td>
</tr>
<tr>
<td>PAPI_STL.ICY</td>
<td>Cycles with no instruction issue</td>
</tr>
<tr>
<td>PAPI_HW.INT</td>
<td>Hardware interrupts</td>
</tr>
<tr>
<td>PAPI_BR_TKN</td>
<td>Conditional branch instructions taken</td>
</tr>
<tr>
<td>PAPI_BR_MSP</td>
<td>Conditional branch instructions mispredicted</td>
</tr>
<tr>
<td>PAPI_TOT_INS</td>
<td>Instructions completed</td>
</tr>
<tr>
<td>PAPI_FP_INS</td>
<td>Floating point instructions</td>
</tr>
<tr>
<td>PAPI_BR_INS</td>
<td>Branch instructions</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Counter/Event Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAPI_VEC_INS</td>
<td>Vector/SIMD instructions</td>
</tr>
<tr>
<td>PAPI_RES_STL</td>
<td>Cycles stalled on any resource</td>
</tr>
<tr>
<td>PAPI_TOT_CYC</td>
<td>Total cycles</td>
</tr>
<tr>
<td>PAPI_L1_DCA</td>
<td>Level 1 data cache accesses</td>
</tr>
<tr>
<td>PAPI_L2_DCA</td>
<td>Level 2 data cache accesses</td>
</tr>
<tr>
<td>PAPI_L2_ICH</td>
<td>Level 2 instruction cache hits</td>
</tr>
<tr>
<td>PAPI_L1_ICA</td>
<td>Level 1 instruction cache accesses</td>
</tr>
<tr>
<td>PAPI_L2_ICA</td>
<td>Level 2 instruction cache accesses</td>
</tr>
<tr>
<td>PAPI_L1_ICR</td>
<td>Level 1 instruction cache reads</td>
</tr>
<tr>
<td>PAPI_L2_TCA</td>
<td>Level 2 total cache accesses</td>
</tr>
<tr>
<td>PAPI_L3_TCR</td>
<td>Level 3 total cache reads</td>
</tr>
<tr>
<td>PAPI_FML_INS</td>
<td>Floating point multiply instructions</td>
</tr>
<tr>
<td>PAPI_FAD_INS</td>
<td>Floating point add instructions (Also includes subtract instructions)</td>
</tr>
<tr>
<td>PAPI_FDV_INS</td>
<td>Floating point divide instructions (Counts both divide and square root instructions)</td>
</tr>
<tr>
<td>PAPI_FSQ_INS</td>
<td>Floating point square root instructions (Counts both divide and square root instructions)</td>
</tr>
<tr>
<td>PAPI_FP_OPS</td>
<td>Floating point operations</td>
</tr>
</tbody>
</table>