Application Performance Tuning

M. D. Jones, Ph.D.

Center for Computational Research
University at Buffalo
State University of New York

High Performance Computing I, 2013

Performance Foundations

Three pillars of performance optimization:

- Algorithmic - choose the most effective algorithm that you can for the problem of interest
- Serial Efficiency - optimize the code to run efficiently in a non-parallel environment
- Parallel Efficiency - effectively use multiple processors to achieve a reduction in execution time, or equivalently, to solve a proportionately larger problem

Algorithmic Efficiency

Choose the best algorithm *before* you start coding (recall that good planning is an essential part of writing good software):

- Running on large number of processors? Choose an algorithm that **scales** well with increasing processor count
- Running a large system (mesh points, particle count, etc.)? Choose an algorithm that **scales** well with system size
- If you are going to run on a massively parallel machine, plan from the beginning on how you intend to decompose the problem (it may save you a lot of time later)

Serial Efficiency

Getting efficient code in parallel is made much more difficult if you have not optimized the sequential code, and in fact can lead to a misleading picture of parallel performance. Recall that our definition of parallel speedup,

$$S(N_p) = \frac{\tau_S}{\tau_p(N_p)},$$

involves the time, τ_S, for an **optimal** sequential implementation (**not** just $\tau_p(1)$!)

Establish a Performance Baseline

Steps to establishing a baseline for your own performance expectations:

- Choose a representative problem (or better still a suite of problems) that can be run under identical circumstances on a multitude of platforms/compilers
  Requires portable code!
- How fast is “fast”? You can utilize hardware performance counters to measure actual code performance
- Profile, profile, and then profile some more ... to find bottlenecks and spend **your** time more effectively in optimizing code

Parallel Performance Trap

Pitfalls when measuring the performance of parallel codes:

- For many, *speedup* or *linear scalability* is the ultimate goal.
- This goal is incomplete - a terribly inefficient code can scale well, but actually deliver poor **efficiency**.
- For example, consider a simple Monte Carlo based code that uses the most rudimentary uniform sampling (i.e. no importance sampling) - this can be made to scale perfectly in parallel, but the algorithmic efficiency (measured perhaps by the delivered variance per cpu-hour) is quite low.

time Command

Note that this is not the `time` built-in function in many shells (`bash` and `tcsh` included), but instead the one located in `/usr/bin`. This command is quite useful for getting an overall picture of code performance. The default output format:

```
%Uuser %Ssystem %Eelapsed %PCPU (%Xtext+%Ddata %Mmax)k
%Iinputs+%Ooutputs (%Fmajor+%Rminor)pagefaults %Wswaps
```

and using the `-p` option:

```
real %e
user %U
sys %S
```

time Example

```
[rush:~/d_laplace]$ /usr/bin/time ./laplace_s
:
  Max value in sol:   0.999992327961218
  Min value in sol:  -8.742278000372475E-008
75.82user 0.00system 1:17.72elapsed 97%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (0major+913minor)pagefaults 0swaps
[bono:~/d_laplace]$ /usr/bin/time -p ./laplace_s
:
  Max value in sol:   0.999992327961218
  Min value in sol:  -8.742278000372475E-008
real 75.73
user 74.68
sys 0.00
```

time MPI Example

```
[rush:~/d_laplace]$ /usr/bin/time mpiexec -np 2 ./laplace_mpi
:
  Max value in sol:   0.999992327961218
  Min value in sol:  -8.742278000372475E-008
Writing logfile....
Finished writing logfile.
28.43user 1.54system 0:31.95elapsed 93%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (0major+14920minor)pagefaults 0swaps
```

You will see the result of timing the `mpiexec` shell script, not the MPI code). Of course, having external timing is nice, but thankfully MPI gives up much better timing and profiling tools to use.

Code Section Timing (Calipers)

Timing sections of code requires a bit more work on the part of the programmer, but there are *reasonably* portable means of doing so:

Routine	Type	Resolution
times	user/sys	ms
gettimeofday	wall	μs
clock_gettime	wall	ns
system_clock (f90)	wall	system-dependent
cpu_time (f95)	cpu	compiler-dependent
MPI_Wtime*	wall	system-dependent
OMP_GET_WTIME*	wall	system-dependent

Generally I prefer the MPI and OpenMP timing calls whenever I can use them (*the MPI and OpenMP specifications call for their intrinsic timers to be high precision).

More information on code section timers (and code for doing so):

- LLNL Performance Tools:

  https://computing.llnl.gov/tutorials/performance_tools/#gettimeofday

- Stopwatch (nice F90 module, but you need to supply a low-level function for accessing a timer):
  http://math.nist.gov/StopWatch

GNU Tools: gprof

Tool that we used briefly before:

- Generic GNU profiler
- Requires recompiling code with **-pg** option
- Running subsequent instrumented code produces **gmon.out** to be read by **gprof**
- Use the environment variable **GMON_OUT_PREFIX** to specify a new **gmon.out** prefix to which the process ID will be appended (especially usefule for parallel runs) - this is a largely undocumented feature ...
- Line-level profiling is possible, as we will see in the following example

gprof Shortcomings

Shortcomings of **gprof** (which apply also to any statistical profiling tool):

- Need to recompile to instrument the code
- Instrumentation can affect the statistics in the profile
- Overhead can significantly increase the running time
- Compiler optimization can be affected by instrumentation

Types of gprof Profiles

gprof profiles come in three types:

1. **Flat Profile:** shows how much time your program spent in each function, and how many times that function was called
2. **Call Graph:** for each function, which functions called it, which other functions it called, and how many times. There is also an estimate of how much time was spent in the subroutines of each function
3. **Basic-block:** Requires compilation with the `-a` flag (supported only by GNU?) - enables gprof to construct an annotated source code listing showing how many times each line of code was executed

gprof example

```
gfortran -I. -O3 -ffast-math -g -pg -o rp rp_read.o initial.o en_gde.o \
          adwfns.o rpqmc.o evol.o dbxgde.o dbxtri.o gen_etg.o gen_rtg.o \
  gdewfn.o lib/*.o
[jonesm@rush ~/d_bench]$ qsub -qdebug -lnodes=1:ppn=2,walltime=00:30:00 -I
[jonesm@rush ~/d_bench]$ ./rp
 Enter runid:(<=9chars)
short
...
...skip copious amount of standard output ...
...
[jonesm@rush ~/d_bench]$ gprof rp gmon.out > & out.gprof
[jonesm@rush ~/d_bench]$ less out.gprof
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  s/call   s/call  name
 89.74    123.23   123.23   204008    0.00     0.00  triwfns_
  6.96    132.79     9.56        1    9.56   137.22  MAIN__
  1.18    134.41     1.62   200004    0.00     0.00  en_gde__
  1.05    135.86     1.44   204002    0.00     0.00  evol_
  0.71    136.83     0.97 14790551    0.00     0.00  ranf_
  0.27    137.20     0.37   204008    0.00     0.00  gdewfn_
...
```

```
[jonesm@rush ~/d_bench]$ gprof --line rp gmon.out > & out.gprof.line
[jonesm@rush ~/d_bench]$ less out.gprof.line
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds calls  ns/call  ns/call  name
 17.45     23.96    23.96                           triwfns_ (adwfns.f:129 @ 403c94)
 14.46     43.82    19.86                           triwfns_ (adwfns.f:130 @ 403ce6)
 12.87     61.50    17.68                           triwfns_ (adwfns.f:171 @ 404755)
 12.31     78.41    16.91                           triwfns_ (adwfns.f:172 @ 4047e2)
  0.67     79.33     0.92                           triwfns_ (adwfns.f:130 @ 403cd6)
  0.59     80.14     0.82                           MAIN__ (cc4WTuQH.f:308 @ 4070c6)
  0.51     80.84     0.70                           MAIN__ (cc4WTuQH.f:304 @ 4072da)
```

More gprof Information

More **gprof** documentation:

- **gprof** GNU Manual:
  `http://sourceware.org/binutils/docs/gprof/`
- **gprof** man page:
  man gprof

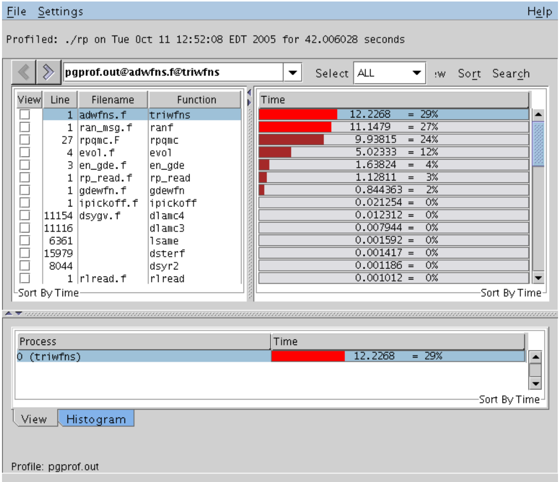
PGI Tools: pgprof

- PGI tools also have profiling capabilities (c.f. **man pgf95**)
- Graphical profiler, **pgprof**

pgprof example

```
pgf90  -tp p7-64 -fastsse -g77libs -Mprof=lines -o rp rp_read.o initial.o en_gde.o \
          adwfns.o rpqmc.o evol.o dbxgde.o dbxtri.o gen_etg.o gen_rtg.o \
  gdewfn.o lib/*.o
[jonesm@rush ~/d_bench]$ ./rp
 Enter runid:(<=9chars)
short
...
...skip copious amount of standard output ...
...
[jonesm@rush ~/d_bench]$ pgprof -exe ./rp pgprof.out
```

pgprof example [screenshot]



- N.B. you can also use the **-text** option to **pgprof** to make it behave more like **gprof**
- See the *PGI Tools Guide* for more information (should be a PDF copy in $PGI/doc)

mpiP: Statistical MPI Profiling

- `http://mpip.sourceforge.net`
- Not a tracing tool, but a lightweight interface to accumulate statistics using the MPI profiling interface
- Quite useful in conjunction with a tracefile analysis (e.g. using jumpshot)
- Installed on CCR systems - see **"module avail"** for **mpiP** availability and location

mpiP Compilation

To use mpiP you need to:

- Add a `-g` flag to add symbols (this will allow mpiP to access the source code symbols and line numbers)
- Link in the necessary mpiP profiling library and the binary utility libraries for actually decoding symbols (*There is a trick that you can use most of the time to avoid having to link with mpiP, though*).

Compilation examples (from `U2`) follow ...

mpiP Runtime Flags

You can set various mpiP runtime flags (e.g. `export MPIP="-t 10.0 -k 2"`):

Option	Description	Default
-c	Generate concise version of report, omitting callsite process-specific detail.	
-e	Print report data using floating-point format	
-f dir	Record output file in directory <dir>	.
-g	Enable mpiP debug mode	disabled
-k n	Sets callsite stack traceback depth to <n>	1
-n	Do not truncate full pathname of filename in callsites	
-o	Disable profiling at initialization. Application must enable profiling with MPI_Pcontrol()	
-s n	Set hash table size to <n>	256
-t x	Set print threshold for report, where <x> is the MPI percentage of time for each callsite	0.0
-v	Generates both concise and verbose report output	

mpiP Example Output

From an older version of mpiP, but still almost entirely the same - this one links directly with mpiP; first compile:

```
mpicc   -g -o atlas aij2_basis.o analyze.o atlas.o barrier.o byteflip.o \
    chordgn2.o cstrings.o io2.o map.o mutils.o numrec.o paramods.o proj.o \
    projAtlas.o sym2.o util.o -lm -L/Projects/CCR/jonesm/mpiP-2.8.2/gnu/ch_gm/lib \
    -lmpiP -lbfd -liberty -lm
```

then run (in this case using 16 processors) and examine the output file:

```
[jonesm@joplin d_derenzo]$ ls *.mpiP
atlas.gcc3-mpi-papi-mpiP.16.20578.1.mpiP
[jonesm@joplin d_derenzo]$ less atlas.gcc3-mpi-papi-mpiP.16.20578.1.mpiP
:
:
```

```
@ mpiP
@ Command : ./atlas.gcc3-mpi-papi-mpiP study.ini 0
@ Version                  : 2.8.2
@ MPIP Build date          : Jun 29 2005, 14:53:41
@ Start time               : 2005 06 29 15:18:52
@ Stop time                : 2005 06 29 15:28:34
@ Timer Used               : gettimeofday
@ MPIP env var             : [null]
@ Collector Rank           : 0
@ Collector PID            : 20578
@ Final Output Dir         : .
@ MPI Task Assignment      : 0 bb18n17.ccr.buffalo.edu
@ MPI Task Assignment      : 1 bb18n17.ccr.buffalo.edu
@ MPI Task Assignment      : 2 bb18n16.ccr.buffalo.edu
@ MPI Task Assignment      : 3 bb18n16.ccr.buffalo.edu
@ MPI Task Assignment      : 4 bb18n15.ccr.buffalo.edu
@ MPI Task Assignment      : 5 bb18n15.ccr.buffalo.edu
@ MPI Task Assignment      : 6 bb18n14.ccr.buffalo.edu
@ MPI Task Assignment      : 7 bb18n14.ccr.buffalo.edu
@ MPI Task Assignment      : 8 bb18n13.ccr.buffalo.edu
@ MPI Task Assignment      : 9 bb18n13.ccr.buffalo.edu
@ MPI Task Assignment      : 10 bb18n12.ccr.buffalo.edu
@ MPI Task Assignment      : 11 bb18n12.ccr.buffalo.edu
@ MPI Task Assignment      : 12 bb18n11.ccr.buffalo.edu
@ MPI Task Assignment      : 13 bb18n11.ccr.buffalo.edu
@ MPI Task Assignment      : 14 bb18n10.ccr.buffalo.edu
@ MPI Task Assignment      : 15 bb18n10.ccr.buffalo.edu
```

```
---------------------------------------------------------------------------
@--- MPI Time (seconds) ---------------------------------------------------
---------------------------------------------------------------------------
Task    AppTime    MPITime     MPI%
   0        582       44.7     7.69
   1        579       41.9     7.24
   2        579       40.7     7.03
   3        579       36.9     6.37
   4        579       22.3     3.84
   5        579       16.6     2.87
   6        579         32     5.53
   7        579       35.9     6.20
   8        579       28.6     4.93
   9        579       25.9     4.48
  10        579       39.2     6.76
  11        579       33.8     5.84
  12        579       35.3     6.10
  13        579         41     7.07
  14        579       29.9     5.16
  15        579       41.4     7.16
   *   9.27e+03        546     5.89
```

```
---------------------------------------------------------------------------
@--- Callsites: 13 --------------------------------------------------------
---------------------------------------------------------------------------
 ID Lev File/Address     Line Parent_Funct             MPI_Call
  1   0 util.c            833 gsync                    Barrier
  2   0 atlas.c          1531 readProjData             Allreduce
  3   0 projAtlas.c       745 backProjAtlas            Allreduce
  4   0 atlas.c          1545 readProjData             Allreduce
  5   0 atlas.c          1525 readProjData             Allreduce
  6   0 atlas.c          1541 readProjData             Allreduce
  7   0 atlas.c          1589 readProjData             Allreduce
  8   0 atlas.c          1519 readProjData             Allreduce
  9   0 util.c            789 mygcast                  Bcast
 10   0 projAtlas.c      1100 computeLoglikeAtlas      Allreduce
 11   0 atlas.c          1514 readProjData             Allreduce
 12   0 atlas.c          1537 readProjData             Allreduce
 13   0 projAtlas.c       425 fwdBackProjAtlas2        Allreduce
```

```
---------------------------------------------------------------------------
@--- Aggregate Time (top twenty, descending, milliseconds) ----------------
---------------------------------------------------------------------------
Call                 Site       Time    App%    MPI%      COV
Allreduce              13   3.09e+05    3.33   56.50     0.46
Barrier                 1   2.13e+05    2.30   38.97     0.35
Bcast                   9   1.69e+04    0.18    3.10     0.37
Allreduce               3   7.78e+03    0.08    1.42     0.11
Allreduce              10       62.7    0.00    0.01     0.20
Allreduce              11       2.42    0.00    0.00     0.09
Allreduce               7       2.17    0.00    0.00     0.26
Allreduce              12       1.15    0.00    0.00     0.20
Allreduce               6       1.14    0.00    0.00     0.19
Allreduce               5       1.13    0.00    0.00     0.15
Allreduce               8       1.12    0.00    0.00     0.18
Allreduce               2        1.1    0.00    0.00     0.13
Allreduce               4        1.1    0.00    0.00     0.12
```

```
---------------------------------------------------------------------------
@--- Aggregate Sent Message Size (top twenty, descending, bytes) ----------
---------------------------------------------------------------------------
Call                 Site      Count      Total       Avrg   Sent%
Allreduce              13      65536   2.28e+09   3.48e+04   83.69
Allreduce               3       8192   2.85e+08   3.48e+04   10.46
Bcast                   9     490784   1.59e+08        325    5.85
Allreduce              11         16   2.07e+04    1.3e+03    0.00
Allreduce              10        512    4.1e+03          8    0.00
Allreduce               7         16        256         16    0.00
Allreduce               2         16         64          4    0.00
Allreduce               6         16         64          4    0.00
Allreduce               5         16         64          4    0.00
Allreduce               4         16         64          4    0.00
Allreduce               8         16         64          4    0.00
Allreduce              12         16         64          4    0.00
...
...
```

Using mpiP at Runtime

Now let's examine an example using mpiP at runtime.
This example is solves a simple Laplace equation with Dirichlet boundary conditions using finite differences.

```bash
#!/bin/bash
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=8
#SBATCH --constraint=CPU-L5520
#SBATCH --partition=debug
#SBATCH --time=00:10:00
#SBATCH --mail-type=END
#SBATCH --mail-user=jonesm@buffalo.edu
#SBATCH --output=slurmMPIP.out
#SBATCH --job-name=mpip-test
module load intel
module load intel-mpi
module load mpip
module list
export I_MPI_DEBUG=4
# Use LD_PRELOAD trick to load mpiP wrappers at runtime
export LD_PRELOAD=$MPIPDIR/lib/libmpiP.so
export I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so
srun ./laplace_mpi<<EOF
2000
EOF
```

... and then run it and examine the resulting mpiP output file:

```
[rush:~/d_laplace/d_mpip]$ ls -l laplace_mpi.*
[rush:~/d_laplace/d_mpip]$ ls -l laplace_mpi.*
-rw-r--r-- 1 jonesm ccrstaff    17920 Dec  5  2012 laplace_mpi.16.11597.1.mpiP
-rw-r--r-- 1 jonesm ccrstaff    17698 Oct  7 15:45 laplace_mpi.16.31074.1.mpiP
-rw-r--r-- 1 jonesm ccrstaff 16032024 Oct  7 15:45 laplace_mpi.dat
lrwxrwxrwx 1 jonesm ccrstaff       18 Dec 21  2010 laplace_mpi.f90 -> ../laplace_mpi.f90
```

and again we will break it down by section:

```
@ mpiP
@ Command : /ifs/user/jonesm/d_laplace/d_mpip/./laplace_mpi
@ Version                    : 3.3.0
@ MPIP Build date            : Oct 14 2011, 16:16:34
@ Start time                 : 2013 10 07 15:43:14
@ Stop time                  : 2013 10 07 15:45:11
@ Timer Used                 : PMPI_Wtime
@ MPIP env var               : [null]
@ Collector Rank             : 0
@ Collector PID              : 31074
@ Final Output Dir           : .
@ Report generation          : Single collector task
@ MPI Task Assignment        : 0 d16n02
@ MPI Task Assignment        : 1 d16n02
@ MPI Task Assignment        : 2 d16n02
@ MPI Task Assignment        : 3 d16n02
@ MPI Task Assignment        : 4 d16n02
@ MPI Task Assignment        : 5 d16n02
@ MPI Task Assignment        : 6 d16n02
@ MPI Task Assignment        : 7 d16n02
@ MPI Task Assignment        : 8 d16n03
@ MPI Task Assignment        : 9 d16n03
@ MPI Task Assignment        : 10 d16n03
@ MPI Task Assignment        : 11 d16n03
@ MPI Task Assignment        : 12 d16n03
@ MPI Task Assignment        : 13 d16n03
@ MPI Task Assignment        : 14 d16n03
@ MPI Task Assignment        : 15 d16n03
```

```
---------------------------------------------------------------------------
@--- MPI Time (seconds) ---------------------------------------------------
---------------------------------------------------------------------------
Task    AppTime    MPITime     MPI%
   0        117       17.2    14.70
   1        117       17.6    15.08
   2        117       19.9    17.03
   3        117       17.9    15.33
   4        117       17.7    15.14
   5        117       14.6    12.53
   6        117       13.6    11.64
   7        117       13.4    11.49
   8        117       12.2    10.48
   9        117       16.7    14.28
  10        117         17    14.52
  11        117       17.2    14.70
  12        117       19.1    16.33
  13        117       17.5    14.97
  14        117       18.3    15.68
  15        117       17.5    14.96
   *   1.87e+03        267    14.30
```

```
---------------------------------------------------------------------------
@--- Callsites: 6 ----------------------------------------------------------
---------------------------------------------------------------------------
 ID Lev File/Address        Line Parent_Funct             MPI_Call
  1   0 laplace_mpi.f90      118 MAIN__                   Allreduce
  2   0 laplace_mpi.f90      143 MAIN__                   Recv
  3   0 laplace_mpi.f90       48 __paramod_MOD_xchange    Sendrecv
  4   0 laplace_mpi.f90       80 MAIN__                   Bcast
  5   0 laplace_mpi.f90       46 __paramod_MOD_xchange    Sendrecv
  6   0 laplace_mpi.f90      138 MAIN__                   Send
---------------------------------------------------------------------------
@--- Aggregate Time (top twenty, descending, milliseconds) ----------------
---------------------------------------------------------------------------
Call                 Site       Time    App%    MPI%     COV
Allreduce               1   2.45e+05   13.09   91.53    0.14
Sendrecv                5   1.12e+04    0.60    4.20    0.17
Sendrecv                3   1.11e+04    0.59    4.14    0.20
Send                    6        306    0.02    0.11    0.55
Bcast                   4       16.7    0.00    0.01    0.25
Recv                    2         14    0.00    0.01    0.00
```

```
---------------------------------------------------------------------------
@--- Callsite Time statistics (all, milliseconds): 80 ---------------------
---------------------------------------------------------------------------
Name                  Site Rank  Count      Max     Mean      Min   App%   MPI%
Allreduce                1    0  23845     32.2    0.667    0.037  13.57  92.36
Allreduce                1    1  23845     29.1    0.685    0.037  13.98  92.72
Allreduce                1    2  23845     33.9    0.779    0.037  15.92  93.45
Allreduce                1    3  23845     29.2    0.696     0.04  14.22  92.77
Allreduce                1    4  23845     32.3    0.687    0.036  14.03  92.65
Allreduce                1    5  23845     29.2     0.56    0.037  11.44  91.29
Allreduce                1    6  23845     28.5    0.511    0.035  10.43  89.60
Allreduce                1    7  23845     25.7    0.493    0.036  10.06  87.62
Allreduce                1    8  23845     26.1    0.444    0.038   9.07  86.55
Allreduce                1    9  23845     28.2    0.637    0.034  13.00  91.04
Allreduce                1   10  23845     33.6    0.649    0.036  13.26  91.30
Allreduce                1   11  23845     28.9    0.657    0.034  13.42  91.29
Allreduce                1   12  23845     32.3    0.737    0.039  15.06  92.23
Allreduce                1   13  23845       31     0.67    0.035  13.69  91.46
Allreduce                1   14  23845     33.7    0.704    0.038  14.38  91.73
Allreduce                1   15  23845     33.9    0.683    0.036  13.94  93.18
Allreduce                1    * 381520     33.9    0.641    0.034  13.09  91.53
```

```
---------------------------------------------------------------------------
@--- Callsite Message Sent statistics (all, sent bytes) --------------------
---------------------------------------------------------------------------
Name              Site Rank   Count       Max      Mean       Min       Sum
Allreduce            1    0   23845         8         8         8 1.908e+05
Allreduce            1    1   23845         8         8         8 1.908e+05
Allreduce            1    2   23845         8         8         8 1.908e+05
Allreduce            1    3   23845         8         8         8 1.908e+05
Allreduce            1    4   23845         8         8         8 1.908e+05
Allreduce            1    5   23845         8         8         8 1.908e+05
Allreduce            1    6   23845         8         8         8 1.908e+05
Allreduce            1    7   23845         8         8         8 1.908e+05
Allreduce            1    8   23845         8         8         8 1.908e+05
Allreduce            1    9   23845         8         8         8 1.908e+05
Allreduce            1   10   23845         8         8         8 1.908e+05
Allreduce            1   11   23845         8         8         8 1.908e+05
Allreduce            1   12   23845         8         8         8 1.908e+05
Allreduce            1   13   23845         8         8         8 1.908e+05
Allreduce            1   14   23845         8         8         8 1.908e+05
Allreduce            1   15   23845         8         8         8 1.908e+05
Allreduce            1    *  381520         8         8         8 3.052e+06
```

Intel Trace Analyzer/Collector (ITAC)

A commercial product for performing MPI trace analysis that has enjoyed a long history is **Vampir/Vampirtrace**, originally developed and sold by Pallas GmbH. Now owned by Intel and available as the Intel Trace Analyzer and Collector. We have a license on `U2` if someone wants to give it a try.
Note that **Vampir/Vampirtrace** has since been reborn as an entirely new product.

ITAC Example

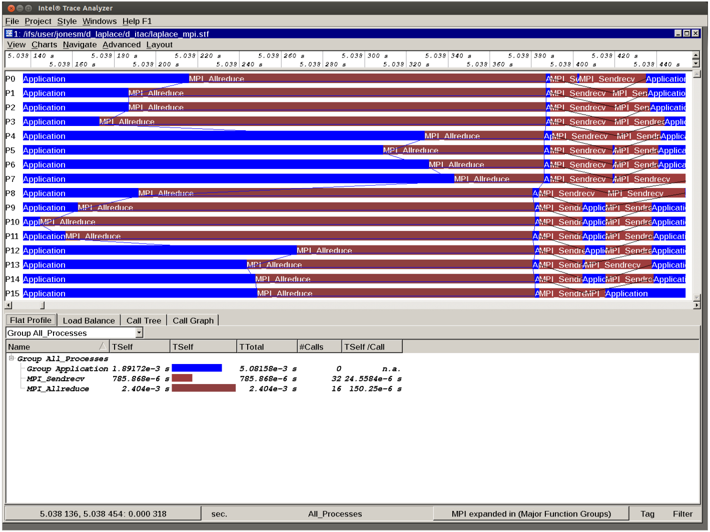
Note that you do not have to recompile your application to use ITAC (unless you are building it statically), you can just build it usual, using Intel MPI:

```
[rush:~/d_laplace/d_itac]$ module load intel-mpi
[rush:~/d_laplace/d_itac]$ make laplace_mpi
mpiifort -ipo -O3 -Vaxlib -g  -c laplace_mpi.f90
mpiifort -ipo -O3 -Vaxlib -g  -o laplace_mpi laplace_mpi.o
ipo: remark #11001: performing single-file optimizations
ipo: remark #11005: generating object file /tmp/ipo_ifortjOzYAn.o
[bono:~/d_laplace/d_itac]$
```

You can turn trace collection on at run-time ...

```
#!/bin/bash
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=8
#SBATCH --constraint=CPU-L5520
#SBATCH --partition=debug
#SBATCH --time=00:10:00
#SBATCH --mail-type=END
#SBATCH --mail-user=jonesm@buffalo.edu
#SBATCH --output=slurmITAC.out
#SBATCH --job-name=mpip-test
module load intel
module load intel-mpi
. /util/intel/itac/8.1.0.024/bin/itacvars.sh
module list
which mpiexec
# this is one of those times when srun is not going to work - need to
#   use Intel MPI's task launching to generate ITAC profile
MYHOSTFILE=tmp.hosts
srun -l hostname -s | sort -n | awk '{print $2}' > $MYHOSTFILE
NNODES=`cat $MYHOSTFILE | uniq | wc -l`
NPROCS=`cat $MYHOSTFILE | wc -l`
export I_MPI_DEBUG=4
mpdboot -n $NNODES -f "$MYHOSTFILE" -v
mpdtrace
mpiexec -trace -np $NPROCS ./laplace_mpi <<EOF
2000
EOF
mpdallexit
[ -e "$MYHOSTFILE" ] && \rm "$MYHOSTFILE"
```

Running the preceding batch job on U2 produces a bunch (many!)of profiling output files, the most important of which can be is the name of your binary with a **.stf** suffix, in this case *laplace_mpi.stf*, which we feed to the Intel Trace Analyzer using the **traceanalyzer** command ... and we should see a profile that looks very much like what you can see using **jumpshot** (MPICH2's trace file profiler).


Intel® Trace Analyzer
File Project Style Windows Help F1
1: /ifs/user/jonesm/d_laplace/d_itac/laplace_mpi.stf
View Charts Navigate Advanced Layout
Flat Profile
Load Balance
Call Tree
Call Graph
Group All_Processes
Name TSelf TSelf TTotal #Calls TSelf /Call
Group All_Processes
Group Application 1.89172e-3 s 5.08158e-3 s 0 n.a.
MPI_Sendrecv 785.868e-6 s 785.868e-6 s 32 24.5584e-6 s
MPI_Allreduce 2.404e-3 s 2.404e-3 s 16 150.25e-6 s
5.038 136, 5.038 454: 0.000 318
sec.
All_Processes
MPI expanded in (Major Function Groups)
Tag
Filter

More ITAC Documentation

Some helpful pointers to more ITAC documentation:

```
[rush:~/d_laplace/d_itac]$ which traceanalyzer
/util/intel/itac/8.1.0.024/bin/traceanalyzer
[rush:~/d_laplace/d_itac]$ ls -1 /util/intel/itac/8.1.0.024/doc
FAQ.pdf
Getting_Started.html
INSTALL.html
ITA_Reference_Guide
ITA_Reference_Guide.htm
ITA_Reference_Guide.pdf
ITC_Reference_Guide
ITC_Reference_Guide.htm
ITC_Reference_Guide.pdf
Release_Notes_Addendum_for_MIC_Architecture.txt
Release_Notes.txt
```

Generally a good idea to refer to the documentation for the same version that you are using (you can check with **”module show intel-mpi”**).

Introduction

- **P**erformance **A**pplication **P**rogramming **I**nterface
- Implement a portable(!) and efficient API to access existing **hardware** performance counters
- Ease the optimization of code by providing base infrastructure for cross-platform tools

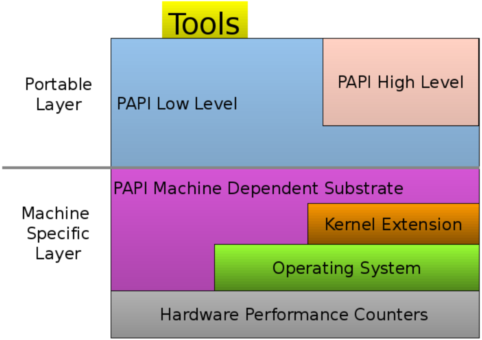
Pre-PAPI

Before PAPI came along, there were hardware performance counters, of course - but access to them was limited to proprietary tools and APIs. Some examples were SGI's **perfex** and Cray's **hpm**.

Now, as long as PAPI has been ported to a particular hardware substrate, the end-programmer (or tool developer) can just use the PAPI interface.

PAPI Schematic

Best summarized by the following schematic picture:



Behind the PAPI Curtain

- Linux - x86/x86_64 uses the **perfctr** kernel patches by Mikael Petterssen:

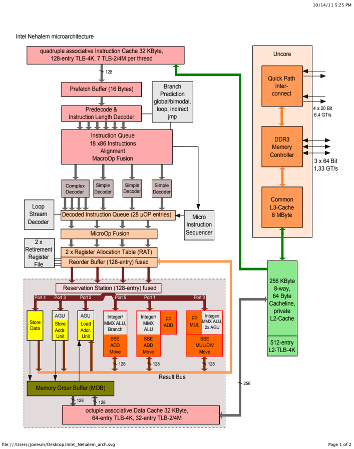
  http://user.it.uu.se/~mikpe/linux/perfctr/2.6

  - Headed for inclusion in mainstream Linux kernel (was a custom patch applied to CCR systems prior to Linux kernel 2.6.32)
  - low overhead
- IA64 - uses **PFM**, developed by HP and included in the linux kernel (for x86_64):
  - Full use of available IA64 monitoring capabilities
  - Quite a bit slower than **perfctr**, at least according to the **PAPI** developers
  - http://www.hpl.hp.com/research/linux/perfmon
  - libpfm lives on using perf events, but perfmon apparently ceased development for Linux as of kernel 2.6.30 or so
- "Perf Events" added to Linux kernel in 2.6.31, replacing both of the above, c.f.:

  http://web.eecs.utk.edu/~vweaver1/projects/perf-events/

Nehalem Xeon Block Diagram

Block diagram for Nehalem architecture, showing a single socket (repeat on QPI for dual sockets):



“Westmere” Xeons

Characteristics of the ’Westmere’ E5645 Xeons that form (part of) CCR’s **U2** cluster:

Clock Cycle	2.4 GHz
TPP	9.6 GFlop/s (per core)
Pipeline	14 stages
L2 Cache Size	256 kByte
L3 Cache Size	12 MByte
CPU-Memory Bandwidth	32 GByte/s (nonuniform!)

"Westmere" Xeon Memory Hierarchy Penalties

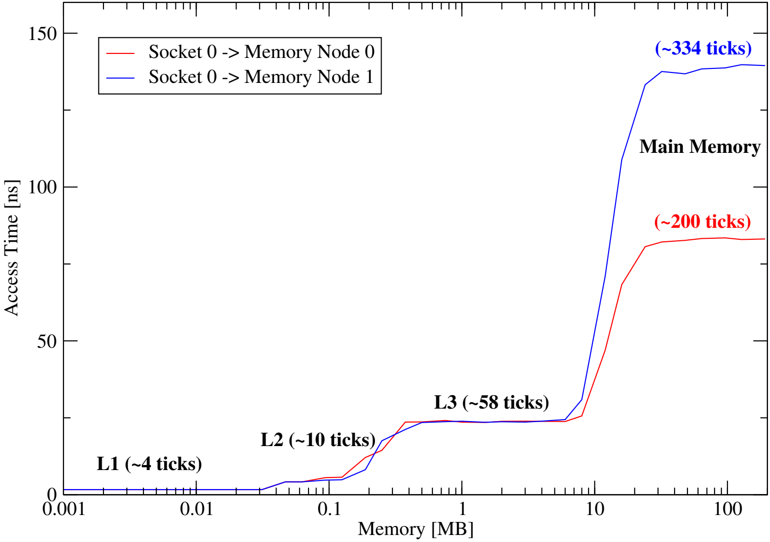
Consider the penalties in lost computation for **not** reusing data in the various caches (using U2's Intel *Westmere* Xeon E5645 processors):

Memory	Miss Penalty (cycles)
L1 Cache	4
L2 Cache	10
L3 Cache	58
Main	200 (on socket)
	334 (off socket)

as determined by the **lmbench** benchmark[1].

[1] `http://www.bitmover.com/lmbench`

Xeon E5645 Memory Latencies



Available PAPI Performance Data

- Cycle count
- Instruction count (including Integer, Floating point, load/store)
- Branches (including Taken/not taken, mispredictions)
- Pipeline stalls (due to memory, resource conflicts)
- Cache (misses for different levels, invalidation)
- TLB (Translation Lookaside Buffer) misses, invalidation, etc.

High-level PAPI

- Intended for coarse-grained measurements
- Requires little (or no) setup code
- Allows only PAPI preset
- Allows only aggregate counting (no statistical profiling)

Low-level PAPI

- More efficient (and functional) than high-level
- About 60 functions
- Thread-safe
- Supports presets and native events

Preset vs. Native Events

preset or pre-defined events, are those which have been considered useful by the PAPI community and developers:

http://icl.cs.utk.edu/projects/papi/presets.html

native events are those countable by the CPU's hardware. These events are highly platform specific, and you would need to consult the processor architecture manuals for the relevant native event lists

Low-level PAPI Functions

- Hardware counter multiplexing (time sharing hardware counters to allow more events to be monitored than can be conventionally supported)
- Processor information
- Address space information
- Memory information (static and dynamic)
- Timing functions
- Hardware event inquiry
- ... and many more

More PAPI Information

For more on PAPI, including source code, documentation, presentations, and links to third-party tools that utilize PAPI, see

http://icl.cs.utk.edu/projects/papi

How to Access PAPI at CCR

Consider a simple example code to measure Flop/s using the high-level PAPI API:

```
#include <stdio.h>
#include <stdlib.h>
#include "papi.h"


int main()
{
  float real_time, proc_time,mflops;
  long_long flpops;
  float ireal_time, iproc_time, imflops;
  long_long iflpops;
  int retval;
```

```c
  if((retval=PAPI_flops(&ireal_time,&iproc_time,&iflpops,&imflops)) < PAPI_OK)
  {
    printf("Could not initialise PAPI_flops \n");
    printf("Your platform may not support floating point operation event.\n");
    printf("retval: %d\n", retval);
    exit(1);
  }

  your_slow_code();

  if((retval=PAPI_flops( &real_time, &proc_time, &flpops, &mflops))<PAPI_OK)
  {
    printf("retval: %d\n", retval);
    exit(1);
  }

  printf("Real_time: %f Proc_time: %f Total flpops: %lld MFLOPS: %f\n",
         real_time, proc_time,flpops,mflops);
  exit(0);
}
int your_slow_code()
{
  int i;
  double  tmp=1.1;

  for(i=1; i<2000; i++)
  {
    tmp=(tmp+100)/i;
  }
  return 0;
}
```

How to Access PAPI at CCR

On U2, you access the **papi** module and compile accordingly:

```
[rush:/ifs/user/jonesm/d_papi]$ module list
Currently Loaded Modulefiles:
  1) null               2) modules             3) use.own             4) intel-mpi/4.1.1
  5) papi/v5.1.1        6) intel/13.1
[rush:~/d_papi]$ icc -I$PAPI/include -o PAPI_flops PAPI_flops.c -L$PAPI/lib -lpapi
[rush:~/d_papi]$ ./PAPI_flops
Real_time: 0.000017 Proc_time: 0.000002 Total flpops: 16 MFLOPS: 6.636506
```

... and on Lennon (SGI Altix):

```
[jonesm@lennon ~/d_papi]$ module load papi
 'papi/v3.2.1' load complete.
[jonesm@lennon ~/d_papi]$ echo $PAPI
/util/perftools/papi-3.2.1
[jonesm@lennon ~/d_papi]$ gcc -I$PAPI/include -o PAPI_flops PAPI_flops.c \
        -L$PAPI/lib -lpapi
[jonesm@lennon ~/d_papi]$ ldd PAPI_flops
        libpapi.so => /util/perftools/papi-3.2.1/lib/libpapi.so
                                                     (0x2000000000040000)
        libc.so.6.1 => /lib/tls/libc.so.6.1 (0x20000000000ac000)
        libpfm.so.2 => /usr/lib/libpfm.so.2 (0x2000000000318000)
        /lib/ld-linux-ia64.so.2 => /lib/ld-linux-ia64.so.2 (0x2000000000000000)
[jonesm@lennon ~/d_papi]$ ./PAPI_flops
Real_time: 0.000143 Proc_time: 0.000132 Total flpops: 48056 MFLOPS: 363.964111
```

N.B., The Altix is long dead, but this is a good example of the cross-platform portability of PAPI accessign the hardware performance counters.

papi_avail Command

You can use **papi_avail** to check event availability (different CPUs support various events):

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
#SBATCH --constraint=CPU-L5520
#SBATCH --partition=debug
#SBATCH --time=00:10:00
#SBATCH --mail-type=END
#SBATCH --mail-user=jonesm@buffalo.edu
#SBATCH --output=slurmQ.out
#SBATCH --job-name=papi-test
#
module load papi
module list
export | grep SLURM
papi_avail
```

```
Available events and hardware information.
--------------------------------------------------------------------------------
PAPI Version             : 5.1.1.0
Vendor string and code   : GenuineIntel (1)
Model string and code    : Intel(R) Xeon(R) CPU           L5520  @ 2.27GHz (26)
CPU Revision             : 5.000000
CPUID Info               : Family: 6  Model: 26  Stepping: 5
CPU Max Megahertz        : 2266
CPU Min Megahertz        : 2266
Hdw Threads per core     : 1
Cores per Socket         : 4
NUMA Nodes               : 2
CPUs per Node            : 4
Total CPUs               : 8
Running in a VM          : no
Number Hardware Counters : 7
Max Multiplex Counters   : 64
--------------------------------------------------------------------------------
```

Name	Code	Avail	Deriv	Description (Note)
PAPI_L1_DCM	0x80000000	Yes	No	Level 1 data cache misses
PAPI_L1_ICM	0x80000001	Yes	No	Level 1 instruction cache misses
PAPI_L2_DCM	0x80000002	Yes	Yes	Level 2 data cache misses
PAPI_L2_ICM	0x80000003	Yes	No	Level 2 instruction cache misses
PAPI_L3_DCM	0x80000004	No	No	Level 3 data cache misses
PAPI_L3_ICM	0x80000005	No	No	Level 3 instruction cache misses
PAPI_L1_TCM	0x80000006	Yes	Yes	Level 1 cache misses
PAPI_L2_TCM	0x80000007	Yes	No	Level 2 cache misses
PAPI_L3_TCM	0x80000008	Yes	No	Level 3 cache misses
PAPI_CA_SNP	0x80000009	No	No	Requests for a snoop
PAPI_CA_SHR	0x8000000a	No	No	Requests for exclusive access to shared cache line
PAPI_CA_CLN	0x8000000b	No	No	Requests for exclusive access to clean cache line
PAPI_CA_INV	0x8000000c	No	No	Requests for cache line invalidation
PAPI_CA_ITV	0x8000000d	No	No	Requests for cache line intervention
PAPI_L3_LDM	0x8000000e	Yes	No	Level 3 load misses
PAPI_L3_STM	0x8000000f	No	No	Level 3 store misses
PAPI_BRU_IDL	0x80000010	No	No	Cycles branch units are idle
PAPI_FXU_IDL	0x80000011	No	No	Cycles integer units are idle
PAPI_FPU_IDL	0x80000012	No	No	Cycles floating point units are idle
PAPI_LSU_IDL	0x80000013	No	No	Cycles load/store units are idle
PAPI_TLB_DM	0x80000014	Yes	No	Data translation lookaside buffer misses
PAPI_TLB_IM	0x80000015	Yes	No	Instruction translation lookaside buffer misses
PAPI_TLB_TL	0x80000016	Yes	Yes	Total translation lookaside buffer misses
PAPI_L1_LDM	0x80000017	Yes	No	Level 1 load misses
PAPI_L1_STM	0x80000018	Yes	No	Level 1 store misses
PAPI_L2_LDM	0x80000019	Yes	No	Level 2 load misses
PAPI_L2_STM	0x8000001a	Yes	No	Level 2 store misses
PAPI_BTAC_M	0x8000001b	No	No	Branch target address cache misses
PAPI_PRF_DM	0x8000001c	No	No	Data prefetch cache misses

```
PAPI_L3_DCH  0x8000001d No  No  Level 3 data cache hits
PAPI_TLB_SD  0x8000001e No  No  Translation lookaside buffer shootdowns
PAPI_CSR_FAL 0x8000001f No  No  Failed store conditional instructions
PAPI_CSR_SUC 0x80000020 No  No  Successful store conditional instructions
PAPI_CSR_TOT 0x80000021 No  No  Total store conditional instructions
PAPI_MEM_SCY 0x80000022 No  No  Cycles Stalled Waiting for memory accesses
PAPI_MEM_RCY 0x80000023 No  No  Cycles Stalled Waiting for memory Reads
PAPI_MEM_WCY 0x80000024 No  No  Cycles Stalled Waiting for memory writes
PAPI_STL_ICY 0x80000025 No  No  Cycles with no instruction issue
PAPI_FUL_ICY 0x80000026 No  No  Cycles with maximum instruction issue
PAPI_STL_CCY 0x80000027 No  No  Cycles with no instructions completed
PAPI_FUL_CCY 0x80000028 No  No  Cycles with maximum instructions completed
PAPI_HW_INT  0x80000029 No  No  Hardware interrupts
PAPI_BR_UCN  0x8000002a Yes No  Unconditional branch instructions
PAPI_BR_CN   0x8000002b Yes No  Conditional branch instructions
PAPI_BR_TKN  0x8000002c Yes No  Conditional branch instructions taken
PAPI_BR_NTK  0x8000002d Yes Yes Conditional branch instructions not taken
PAPI_BR_MSP  0x8000002e Yes No  Conditional branch instructions mispredicted
PAPI_BR_PRC  0x8000002f Yes Yes Conditional branch instructions correctly predicted
PAPI_FMA_INS 0x80000030 No  No  FMA instructions completed
PAPI_TOT_IIS 0x80000031 Yes No  Instructions issued
PAPI_TOT_INS 0x80000032 Yes No  Instructions completed
PAPI_INT_INS 0x80000033 No  No  Integer instructions
PAPI_FP_INS  0x80000034 Yes No  Floating point instructions
PAPI_LD_INS  0x80000035 Yes No  Load instructions
PAPI_SR_INS  0x80000036 Yes No  Store instructions
PAPI_BR_INS  0x80000037 Yes No  Branch instructions
PAPI_VEC_INS 0x80000038 No  No  Vector/SIMD instructions (could include integer)
PAPI_RES_STL 0x80000039 Yes No  Cycles stalled on any resource
PAPI_FP_STAL 0x8000003a No  No  Cycles the FP unit(s) are stalled
PAPI_TOT_CYC 0x8000003b Yes No  Total cycles
PAPI_LST_INS 0x8000003c Yes Yes Load/store instructions completed
PAPI_SYC_INS 0x8000003d No  No  Synchronization instructions completed
```

```
PAPI_L1_DCH  0x8000003e  Yes   Yes   Level 1 data cache hits
PAPI_L2_DCH  0x8000003f  Yes   Yes   Level 2 data cache hits
PAPI_L1_DCA  0x80000040  Yes   No    Level 1 data cache accesses
PAPI_L2_DCA  0x80000041  Yes   No    Level 2 data cache accesses
PAPI_L3_DCA  0x80000042  Yes   Yes   Level 3 data cache accesses
PAPI_L1_DCR  0x80000043  Yes   No    Level 1 data cache reads
PAPI_L2_DCR  0x80000044  Yes   No    Level 2 data cache reads
PAPI_L3_DCR  0x80000045  Yes   No    Level 3 data cache reads
PAPI_L1_DCW  0x80000046  Yes   No    Level 1 data cache writes
PAPI_L2_DCW  0x80000047  Yes   No    Level 2 data cache writes
PAPI_L3_DCW  0x80000048  Yes   No    Level 3 data cache writes
PAPI_L1_ICH  0x80000049  Yes   No    Level 1 instruction cache hits
PAPI_L2_ICH  0x8000004a  Yes   No    Level 2 instruction cache hits
PAPI_L3_ICH  0x8000004b  No    No    Level 3 instruction cache hits
PAPI_L1_ICA  0x8000004c  Yes   No    Level 1 instruction cache accesses
PAPI_L2_ICA  0x8000004d  Yes   No    Level 2 instruction cache accesses
PAPI_L3_ICA  0x8000004e  Yes   No    Level 3 instruction cache accesses
PAPI_L1_ICR  0x8000004f  Yes   No    Level 1 instruction cache reads
PAPI_L2_ICR  0x80000050  Yes   No    Level 2 instruction cache reads
PAPI_L3_ICR  0x80000051  Yes   No    Level 3 instruction cache reads
PAPI_L1_ICW  0x80000052  No    No    Level 1 instruction cache writes
PAPI_L2_ICW  0x80000053  No    No    Level 2 instruction cache writes
PAPI_L3_ICW  0x80000054  No    No    Level 3 instruction cache writes
PAPI_L1_TCH  0x80000055  No    No    Level 1 total cache hits
PAPI_L2_TCH  0x80000056  Yes   Yes   Level 2 total cache hits
PAPI_L3_TCH  0x80000057  No    No    Level 3 total cache hits
PAPI_L1_TCA  0x80000058  Yes   Yes   Level 1 total cache accesses
PAPI_L2_TCA  0x80000059  Yes   No    Level 2 total cache accesses
PAPI_L3_TCA  0x8000005a  Yes   No    Level 3 total cache accesses
PAPI_L1_TCR  0x8000005b  Yes   Yes   Level 1 total cache reads
PAPI_L2_TCR  0x8000005c  Yes   Yes   Level 2 total cache reads
```

```
PAPI_L3_TCR  0x8000005d  Yes   Yes  Level 3 total cache reads
PAPI_L1_TCW  0x8000005e  No    No   Level 1 total cache writes
PAPI_L2_TCW  0x8000005f  Yes   No   Level 2 total cache writes
PAPI_L3_TCW  0x80000060  Yes   No   Level 3 total cache writes
PAPI_FML_INS 0x80000061  No    No   Floating point multiply instructions
PAPI_FAD_INS 0x80000062  No    No   Floating point add instructions
PAPI_FDV_INS 0x80000063  No    No   Floating point divide instructions
PAPI_FSQ_INS 0x80000064  No    No   Floating point square root instructions
PAPI_FNV_INS 0x80000065  No    No   Floating point inverse instructions
PAPI_FP_OPS  0x80000066  Yes   Yes  Floating point operations
PAPI_SP_OPS  0x80000067  Yes   Yes  Floating point operations;
                                    optimized to count scaled single precision vector operatio
PAPI_DP_OPS  0x80000068  Yes   Yes  Floating point operations;
                                    optimized to count scaled double precision vector operatio
PAPI_VEC_SP  0x80000069  Yes   No   Single precision vector/SIMD instructions
PAPI_VEC_DP  0x8000006a  Yes   No   Double precision vector/SIMD instructions
PAPI_REF_CYC 0x8000006b  Yes   No   Reference clock cycles
-------------------------------------------------------------------------
Of 108 possible events, 64 are available, of which 17 are derived.
```

papi_even_chooser Command

Not all events can be simultaneously monitored (at least not without multiplexing):

```
[rush:~]$ papi_event_chooser PRESET PAPI_FP_OPS
Event Chooser: Available events which can be added with given events.
--------------------------------------------------------------------------------
PAPI Version              : 5.1.1.0
Vendor string and code    : GenuineIntel (1)
Model string and code     : Intel(R) Xeon(R) CPU E7- 4830  @ 2.13GHz (47)
CPU Revision              : 2.000000
CPUID Info                : Family: 6  Model: 47  Stepping: 2
CPU Max Megahertz         : 2127
CPU Min Megahertz         : 2127
Hdw Threads per core      : 1
Cores per Socket          : 8
NUMA Nodes                : 4
CPUs per Node             : 8
Total CPUs                : 32
Running in a VM           : no
Number Hardware Counters : 7
Max Multiplex Counters    : 64
--------------------------------------------------------------------------------

    Name        Code      Deriv Description (Note)
PAPI_L1_DCM  0x80000000  No    Level 1 data cache misses
PAPI_L1_ICM  0x80000001  No    Level 1 instruction cache misses
```

```
PAPI_L2_DCM  0x80000002  Yes  Level 2 data cache misses
PAPI_L2_ICM  0x80000003  No   Level 2 instruction cache misses
PAPI_L1_TCM  0x80000006  Yes  Level 1 cache misses
PAPI_L2_TCM  0x80000007  No   Level 2 cache misses
PAPI_L3_TCM  0x80000008  No   Level 3 cache misses
PAPI_L3_LDM  0x8000000e  No   Level 3 load misses
PAPI_TLB_DM  0x80000014  No   Data translation lookaside buffer misses
PAPI_TLB_IM  0x80000015  No   Instruction translation lookaside buffer misses
PAPI_TLB_TL  0x80000016  Yes  Total translation lookaside buffer misses
PAPI_L1_LDM  0x80000017  No   Level 1 load misses
PAPI_L1_STM  0x80000018  No   Level 1 store misses
PAPI_L2_LDM  0x80000019  No   Level 2 load misses
PAPI_L2_STM  0x8000001a  No   Level 2 store misses
PAPI_BR_UCN  0x8000002a  No   Unconditional branch instructions
PAPI_BR_CN   0x8000002b  No   Conditional branch instructions
PAPI_BR_TKN  0x8000002c  No   Conditional branch instructions taken
PAPI_BR_NTK  0x8000002d  Yes  Conditional branch instructions not taken
PAPI_BR_MSP  0x8000002e  No   Conditional branch instructions mispredicted
PAPI_BR_PRC  0x8000002f  Yes  Conditional branch instructions correctly predicted
PAPI_TOT_IIS 0x80000031  No   Instructions issued
PAPI_TOT_INS 0x80000032  No   Instructions completed
PAPI_FP_INS  0x80000034  No   Floating point instructions
PAPI_LD_INS  0x80000035  No   Load instructions
PAPI_SR_INS  0x80000036  No   Store instructions
PAPI_BR_INS  0x80000037  No   Branch instructions
PAPI_RES_STL 0x80000039  No   Cycles stalled on any resource
PAPI_TOT_CYC 0x8000003b  No   Total cycles
```

```
PAPI_LST_INS 0x8000003c  Yes  Load/store instructions completed
PAPI_L2_DCH  0x8000003f  Yes  Level 2 data cache hits
PAPI_L2_DCA  0x80000041  No   Level 2 data cache accesses
PAPI_L3_DCA  0x80000042  Yes  Level 3 data cache accesses
PAPI_L2_DCR  0x80000044  No   Level 2 data cache reads
PAPI_L3_DCR  0x80000045  No   Level 3 data cache reads
PAPI_L2_DCW  0x80000047  No   Level 2 data cache writes
PAPI_L3_DCW  0x80000048  No   Level 3 data cache writes
PAPI_L1_ICH  0x80000049  No   Level 1 instruction cache hits
PAPI_L2_ICH  0x8000004a  No   Level 2 instruction cache hits
PAPI_L1_ICA  0x8000004c  No   Level 1 instruction cache accesses
PAPI_L2_ICA  0x8000004d  No   Level 2 instruction cache accesses
PAPI_L3_ICA  0x8000004e  No   Level 3 instruction cache accesses
PAPI_L1_ICR  0x8000004f  No   Level 1 instruction cache reads
PAPI_L2_ICR  0x80000050  No   Level 2 instruction cache reads
PAPI_L3_ICR  0x80000051  No   Level 3 instruction cache reads
PAPI_L2_TCH  0x80000056  Yes  Level 2 total cache hits
PAPI_L2_TCA  0x80000059  No   Level 2 total cache accesses
PAPI_L3_TCA  0x8000005a  No   Level 3 total cache accesses
PAPI_L2_TCR  0x8000005c  Yes  Level 2 total cache reads
PAPI_L3_TCR  0x8000005d  Yes  Level 3 total cache reads
PAPI_L2_TCW  0x8000005f  No   Level 2 total cache writes
PAPI_L3_TCW  0x80000060  No   Level 3 total cache writes
PAPI_SP_OPS  0x80000067  Yes  Floating point operations; optimized to count scaled single prec
PAPI_DP_OPS  0x80000068  Yes  Floating point operations; optimized to count scaled double prec
PAPI_VEC_SP  0x80000069  No   Single precision vector/SIMD instructions
PAPI_VEC_DP  0x8000006a  No   Double precision vector/SIMD instructions
PAPI_REF_CYC 0x8000006b  No   Reference clock cycles
-------------------------------------------------------------------------
Total events reported: 57
event_chooser.c                    PASSED
```

PAPI Examples

In this section we will work through a few simple examples of using the PAPI API, mostly focused on using the high-level API. And we will steer clear of native events, and leave those to tool developers.

Accessing Counters Through PAPI

Include files for constants and routine interfaces:

- C: `papi.h`
- F77: `f77papi.h`
- F90: `f90papi.h`

PAPI Naming Scheme

The `C` interfaces:

PAPI C interface

(return type) PAPI_function_name(arg1, arg2, ...)

and `Fortran` interfaces

PAPI Fortran interfaces

PAPIF_function_name(arg1, arg2, ..., *check*)

note that the *check* parameter is the same type and value as the `C` return value.

Relation Between C and Fortran Types in PAPI

The following table shows the relation between the C and Fortran types used in PAPI:

Pseudo-type	Fortran type	Description
C_INT	INTEGER	Default Integer type
C_FLOAT	REAL	Default Real type
C_LONG_LONG	INTEGER*8	Extended size integer
C_STRING	CHARACTER*(PAPI_MAX_STR_LEN)	Fortran string
C_INT FUNCTION	EXTERNAL INTEGER FUNCTION	Fortran function returning integer result

High-level API Example in C

Let's consider the following example code for using the high-level API in C.

```c
#include "papi.h"
#include <stdio.h>
#define NUM_EVENTS 2
#define THRESHOLD 10000
#define ERROR_RETURN(retval) { fprintf(stderr, "Error %d %s:line %d: \n",  \
                              retval,__FILE__,__LINE__);  exit(retval); }
void computation_mult() { /* stupid codes to be monitored */
  double tmp=1.0;
  int i=1;
  for( i = 1; i < THRESHOLD; i++ ) {
    tmp = tmp*i;
  }
}
void computation_add() { /* stupid codes to be monitored */
  int tmp = 0;
  int i=0;

  for( i = 0; i < THRESHOLD; i++ ) {
    tmp = tmp + i;
  }
}
```

```c
int main()
{
  /*Declaring and initializing the event set with the presets*/
  int Events[2] = {PAPI_TOT_INS, PAPI_TOT_CYC};
  /*The length of the events array should be no longer than the
    value returned by PAPI_num_counters.*/

  /*declaring place holder for no of hardware counters */
  int num_hwcntrs = 0;
  int retval;
  char errstring[PAPI_MAX_STR_LEN];
  /*This is going to store our list of results*/
  long_long values[NUM_EVENTS];

/***************************************************************************
 *  This part initializes the library and compares the version number of the*
 * header file, to the version of the library, if these don't match then it *
 * is likely that PAPI won't work correctly.If there is an error, retval    *
 * keeps track of the version number.                                       *
 ***************************************************************************/
  if((retval = PAPI_library_init(PAPI_VER_CURRENT)) != PAPI_VER_CURRENT ) {
    fprintf(stderr, "Error: %d %s\n",retval, errstring);
    exit(1);
  }
```

```c
/*****************************************************************************
 * PAPI_num_counters returns the number of hardware counters the platform *
 * has or a negative number if there is an error                          *
 *****************************************************************************/
  if ((num_hwcntrs = PAPI_num_counters()) < PAPI_OK) {
    printf("There are no counters available. \n");
    exit(1);
  }

  printf("There are %d counters in this system\n",num_hwcntrs);

/*****************************************************************************
 * PAPI_start_counters initializes the PAPI library (if necessary) and    *
 * starts counting the events named in the events array. This function    *
 * implicitly stops and initializes any counters running as a result of   *
 * a previous call to PAPI_start_counters.                                *
 *****************************************************************************/

  if ( (retval = PAPI_start_counters(Events, NUM_EVENTS)) != PAPI_OK)
    ERROR_RETURN(retval);

  printf("\nCounter Started: \n");

  /* Your code goes here*/
  computation_add();
```

```c
/*********************************************************************
 * PAPI_read_counters reads the counter values into values array      *
 *********************************************************************/
  if ( (retval=PAPI_read_counters(values, NUM_EVENTS)) != PAPI_OK)
    ERROR_RETURN(retval);
  printf("Read successfully\n");

  printf("The total instructions executed for addition are %lld \n",values[0]);
  printf("The total cycles used are %lld \n", values[1] );

  printf("\nNow we try to use PAPI_accum to accumulate values\n");

  /* Do some computation here */
  computation_add();
/***********************************************************************
 * What PAPI_accum_counters does is it adds the running counter values  *
 * to what is in the values array. The hardware counters are reset and  *
 * left running after the call.                                         *
 ***********************************************************************/
  if ( (retval=PAPI_accum_counters(values, NUM_EVENTS)) != PAPI_OK)
    ERROR_RETURN(retval);

  printf("We did an additional %d times addition!\n", THRESHOLD);
  printf("The total instructions executed for addition are %lld \n",
         values[0] );
  printf("The total cycles used are %lld \n", values[1] );
```

```
/*********************************************************************
 * Stop counting events(this reads the counters as well as stops them  *
 *********************************************************************/

  printf("\nNow we try to do some multiplications\n");
  computation_mult();

/******************* PAPI_stop_counters **********************************/
  if ((retval=PAPI_stop_counters(values, NUM_EVENTS)) != PAPI_OK)
    ERROR_RETURN(retval);

  printf("The total instruction executed for multiplication are %lld \n",
         values[0] );
  printf("The total cycles used are %lld \n", values[1] );
  exit(0);
}
```

Running on CCR:

```
[rush:/ifs/user/jonesm/d_papi]$ icc -I$PAPI/include -o highlev highlev.c -L$PAPI/lib -lpapi
[rush:/ifs/user/jonesm/d_papi]$ module list
Currently Loaded Modulefiles:
  1) null              2) modules           3) use.own           4) intel-mpi/4.1.1
  5) papi/v5.1.1       6) intel/13.1
[rush:/ifs/user/jonesm/d_papi]$ ./highlev
There are 7 counters in this system

Counter Started:
Read successfully
The total instructions executed for addition are 4406
The total cycles used are 8482

Now we try to use PAPI_accum to accumulate values
We did an additional 10000 times addition!
The total instructions executed for addition are 10056
The total cycles used are 16570

Now we try to do some multiplications
The total instruction executed for multiplication are 6041
The total cycles used are 7328
```

PAPI Initialization

The preceding example used `PAPI_library_init` to initialize PAPI, which is also used for the low-level API, but you can also use the `PAPI_num_counters`, `PAPI_start_counters`, or one of the “rate” calls, `PAPI_flips`, `PAPI_flops`, or `PAPI_ipc`.

Events are counted, as we saw in the example, using `PAPI_accum_counters`, `PAPI_read_counters`, and `PAPI_stop_counters`.

Let’s look at an even simpler example just using one of the rate counters.

High-level Example in F90

For something a little different we can look at our old friend, matrix multiplication, this time in Fortran:

```
! A simple example for the use of PAPI, the number of flops you should
! get is about INDEX^3  on machines that consider add and multiply one flop
! such as SGI, and 2*(INDEX^3) that don't consider it 1 flop such as INTEL
! -Kevin London
program flops
  implicit none
  include "f90papi.h"

  integer,parameter :: i8=SELECTED_INT_KIND(16) ! integer*8
  integer,parameter :: index=1000
  real :: matrixa(index,index),matrixb(index,index),mres(index,index)
  real :: proc_time, mflops, real_time
  integer(kind=i8) :: flpins
  integer :: i,j,k, retval
```

```fortran
retval = PAPI_VER_CURRENT
CALL PAPIf_library_init(retval)
if ( retval.NE.PAPI_VER_CURRENT) then
   print*,'Failure in PAPI_library_init: ', retval
end if

CALL PAPIf_query_event(PAPI_FP_OPS, retval)
if (retval .NE. PAPI_OK) then
   print*,'Sorry, no PAPI_FP_OPS event: ',PAPI_ENOEVNT
end if

! Initialize the Matrix arrays
do i=1,index
   do j=1,index
      matrixa(i,j) = i+j
      matrixb(i,j) = j-i
      mres(i,j) = 0.0
   end do
end do

! Setup PAPI library and begin collecting data from the counters
call PAPIf_flops( real_time, proc_time, flpins, mflops, retval )
if ( retval.NE.PAPI_OK) then
   print*,'Failure on PAPIf_flops: ',retval
end if
```

```fortran
  ! Matrix-Matrix Multiply
  do i=1,index
     do j=1,index
        do k=1,index
           mres(i,j) = mres(i,j) + matrixa(i,k)*matrixb(k,j)
        end do
     end do
  end do

  ! Collect the data into the Variables passed in
  call PAPIf_flops( real_time, proc_time, flpins, mflops, retval)
  if ( retval.NE.PAPI_OK) then
     print*,'Failure on PAPIf_flops: ',retval
  end if
  print *, 'Real_time: ', real_time
  print *, ' Proc_time: ', proc_time
  print *, ' Total flpins: ', flpins
  print *, ' MFLOPS: ', mflops
end program flops
```

Compile and run on E5645 U2:

```
[rush:]$ ifort -I$PAPI/include -o flops flops.f90 -L$PAPI/lib -lpapi
[rush:/ifs/user/jonesm/d_papi]$ ./flops
 Real_time:   0.4914970
  Proc_time:   0.4900359
  Total flpins:              503306816
  MFLOPS:    1027.082
```

Low-level API

The low-level API is primarily intended for experienced application programmers and tool developers. It manages hardware events in user-defined groups called “event sets,” and can use both preset and native events. The low-level API can also interrogate the hardware and determine memory sizes of the executable itself.

The low-level API can also be used for **multiplexing**, in which more (virtual) counters can be used than the underlying hardware supports by timesharing the available (physical) hardware counters.

PAPI Low-level Example

A simple example using the low-level API:

```
#include <papi.h>
#include <stdio.h>

#define NUM_FLOPS 10000

main() {
int retval, EventSet=PAPI_NULL;
long_long values[1];

/* Initialize the PAPI library */
retval = PAPI_library_init(PAPI_VER_CURRENT);
if (retval != PAPI_VER_CURRENT) {
  fprintf(stderr, "PAPI library init error!\n");
  exit(1);
}

/* Create the Event Set */
if (PAPI_create_eventset(&EventSet) != PAPI_OK) handle_error(1);

/* Add Total Instructions Executed to our Event Set */

if (PAPI_add_event(EventSet, PAPI_TOT_INS) != PAPI_OK) handle_error(1);
```

```
/* Start counting events in the Event Set */
if (PAPI_start(EventSet) != PAPI_OK) handle_error(1);

/* Defined in tests/do_loops.c in the PAPI source distribution */
do_flops(NUM_FLOPS);

/* Read the counting events in the Event Set */
if (PAPI_read(EventSet, values) != PAPI_OK) handle_error(1);

printf("After reading the counters: %lld\n",values[0]);

/* Reset the counting events in the Event Set */
if (PAPI_reset(EventSet) != PAPI_OK) handle_error(1);

do_flops(NUM_FLOPS);

/* Add the counters in the Event Set */
if (PAPI_accum(EventSet, values) != PAPI_OK) handle_error(1);
printf("After adding the counters: %lld\n",values[0]);

do_flops(NUM_FLOPS);

/* Stop the counting of events in the Event Set */
if (PAPI_stop(EventSet, values) != PAPI_OK) handle_error(1);

printf("After stopping the counters: %lld\n",values[0]);
}
```

PAPI in Parallel

threads `PAPI_thread_init` enables PAPI's thread support, and should be called immediately after `PAPI_library_init`.

MPI codes are treated very simply - each process has its own address space, and potentially its own hardware counters.

High-level Tools

There are a bunch of open-source high-level tools that build on some of the simple approaches that we have been talking about. General characteristics found in most (not necessarily all):

- Ability to generate and view MPI trace files, leveraging MPI's built-in profiling interface,
- Ability to do statistical profiling (á la `gprof`) and code viewing for identifying hotspots,
- Ability to access performance counters, leveraging PAPI

Tool Examples

A list of such high-level tool examples (not exhaustive):

- **TAU**, Tuning and Analysis Utility,
  http://www.cs.uoregon.edu/Research/tau/home.php
- **Open|SpeedShop**,funded by U.S. DOE
  http://www.openspeedshop.org
- **IPM**, Integrated Performance Management
  http://ipm-hpc.sourceforge.net

Example: IPM

IPM is relatively simple to install and use, so we can easily walk through our favorite example. Note that IPM does:

- MPI
- PAPI
- I/O profiling
- Memory
- Timings: wall, user, and system

Run and Gather

```
#!/bin/bash
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=16
#SBATCH --constraint=CPU-E5-2660
#SBATCH --partition=debug
#SBATCH --time=00:15:00
#SBATCH --mail-type=END
#SBATCH --mail-user=jonesm@buffalo.edu
#SBATCH --output=slurmIPM.out
#SBATCH --job-name=ipm-test
module load intel
module load intel-mpi
module list
export I_MPI_DEBUG=4
# Use LD_PRELOAD trick to load ipm wrappers at runtime
export LD_PRELOAD=/projects/jonesm/ipm/src/ipm/lib/libipm.so
export I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so
srun ./laplace_mpi<<EOF
2000
EOF
```

... and the output is a big xml file plus some useful output to standard output:

```
[rush:~/d_laplace/d_ipm]$ file jonesm.1318862245.001449.0
jonesm.1318862245.001449.0: XML  document text
[rush:~/d_laplace/d_ipm]$ less subMPIP.out
...
##IPMv0.983####################################################################
#
# command : ./laplace_mpi  (completed)
# host    : d16n03/x86_64_Linux           mpi_tasks : 16 on 2 nodes
# start   : 10/17/11/10:37:25             wallclock : 116.170005 sec
# stop    : 10/17/11/10:39:21             %comm     : 13.94
# gbytes  : 2.24606e+00 total             gflop/sec : 5.02520e+00 total
#
##############################################################################
# region  : *       [ntasks] =     16
#
```

```
#                              [total]         <avg>           min           max
# entries                           16             1             1             1
# wallclock                    1853.71       115.857       115.816        116.17
# user                         1853.09       115.818       115.707       115.936
# system                       2.18066      0.136291      0.071989      0.198969
# mpi                          259.152        16.197       11.3859       19.1157
# %comm                                      13.9425       9.82914       16.5048
# gflop/sec                     5.0252      0.314075      0.311741      0.319497
# gbytes                       2.24606      0.140379      0.138138      0.170021
#
# PAPI_FP_OPS              5.83778e+11   3.64861e+10   3.62149e+10    3.7116e+10
# PAPI_FP_INS               5.8276e+11   3.64225e+10   3.62144e+10   3.69079e+10
# PAPI_DP_OPS              5.82764e+11   3.64228e+10   3.62144e+10   3.69079e+10
# PAPI_VEC_DP              4.00803e+06        250501             0   4.00803e+06
#
#                               [time]       [calls]        <%mpi>       <%wall>
# MPI_Allreduce                243.838        381520         94.09         13.15
# MPI_Sendrecv                 14.9598        763040          5.77          0.81
# MPI_Send                    0.339084            15          0.13          0.02
# MPI_Recv                   0.0143731            15          0.01          0.00
# MPI_Bcast                 0.00124932            16          0.00          0.00
# MPI_Comm_rank            1.58967e-05            16          0.00          0.00
# MPI_Comm_size            8.01496e-06            16          0.00          0.00
###############################################################################
```

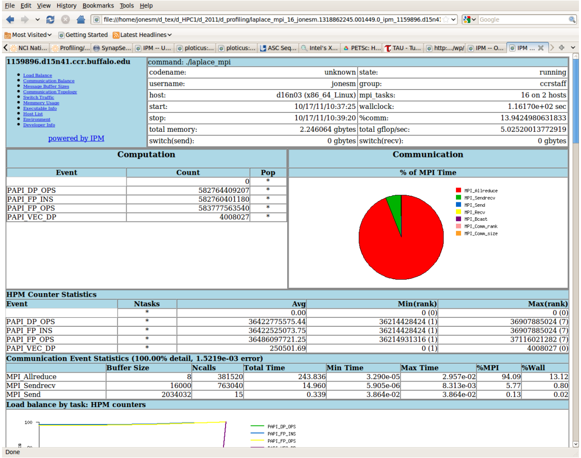
Script to Generate HTML from XML Results

```bash
#!/bin/bash

if [ $# -ne 1 ]; then
   echo "Usage: $0 xml_filename"
   exit
fi
XMLFILE=$1

export IPM_KEYFILE=/projects/jonesm/ipm/src/ipm/ipm_key
export PATH=${PATH}:/projects/jonesm/ipm/src/ipm/bin
/projects/jonesm/ipm/src/ipm/bin/ipm_parse -html $XMLFILE
```

```
[u2:~/d_laplace/d_ipm]$ ./genhtml.sh jonesm.1318862245.001449.0
# data_acquire     = 0 sec
# data_workup      = 0 sec
#  mpi_pie         = 1 sec
#  task_data       = 0 sec
#  load_bal        = 0 sec
#  time_stack      = 0 sec
#  mpi_stack       = 1 sec
#  mpi_buff        = 0 sec
#  switch+mem      = 0 sec
#  topo_tables     = 0 sec
#  topo_data       = 0 sec
#  topo_time       = 0 sec
# html_all         = 2 sec
# html_regions     = 0 sec
# html_nonregion   = 1 sec
[rush:~/d_laplace/d_ipm]$ ls -l \
laplace_mpi_16_jonesm.1318862245.001449.0_ipm_1159896.d15n41.ccr.buffalo.edu/
total 346
-rw-r--r-- 1 jonesm ccrstaff   994 Oct 17 16:07 dev.html
-rw-r--r-- 1 jonesm ccrstaff   104 Oct 17 16:07 env.html
-rw-r--r-- 1 jonesm ccrstaff   347 Oct 17 16:07 exec.html
-rw-r--r-- 1 jonesm ccrstaff   451 Oct 17 16:07 hostlist.html
drwxr-xr-x 2 jonesm ccrstaff   930 Oct 17 16:07 img
-rw-r--r-- 1 jonesm ccrstaff 10550 Oct 17 16:07 index.html
-rw-r--r-- 1 jonesm ccrstaff   387 Oct 17 16:07 map_adjacency.txt
-rw-r--r-- 1 jonesm ccrstaff  8961 Oct 17 16:07 map_calls.txt
-rw-r--r-- 1 jonesm ccrstaff  1452 Oct 17 16:07 map_data.txt
drwxr-xr-x 2 jonesm ccrstaff   803 Oct 17 16:07 pl
-rw-r--r-- 1 jonesm ccrstaff  2620 Oct 17 16:07 task_data
[rush:~/d_laplace/d_ipm]$ tar czf my-ipm-files.tgz \
  laplace_mpi_16_jonesm.1318862245.001449.0_ipm_1159896.d15n41.ccr.buffalo.edu/
[rush:~/d_laplace/d_ipm]$ ls -l my-ipm-files.tgz
-rw-r--r-- 1 jonesm ccrstaff 71509 Oct 17 16:48 my-ipm-files.tgz
```

Visualize Results in Browser

Transfer compressed tar file to your local machine, unpack, and browse the results:



Summary

Summary of high-level tools

- IPM is pretty easy to use, provides some good functionality
- TAU and Open|SpeedShop have steeper learning curves, much more functionality