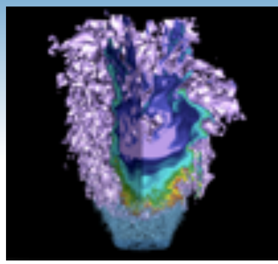
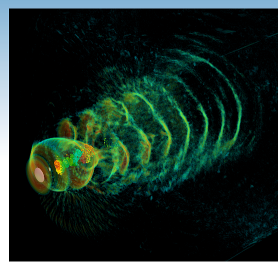
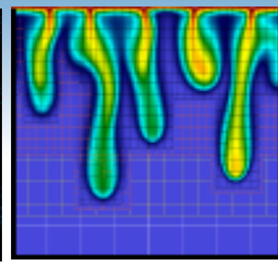
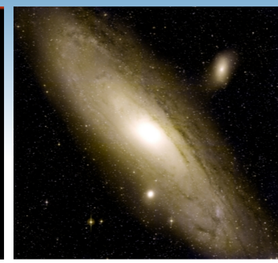
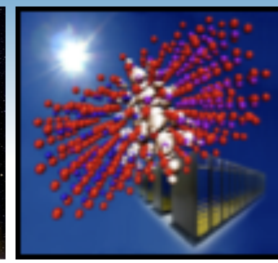
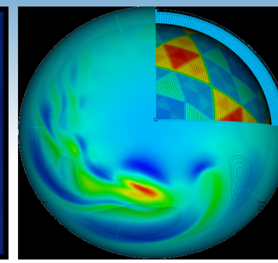
     

Cray Optimization and Performance Tools

Harvey Wasserman
Woo-Sun Yang
NERSC User Services Group

February 7-8, 2011

Cray XE6 Workshop
NERSC Oakland Scientific Facility

U.S. DEPARTMENT OF ENERGY | Office of Science
NERSC National Energy Research Scientific Computing Center
BERKELEY LAB

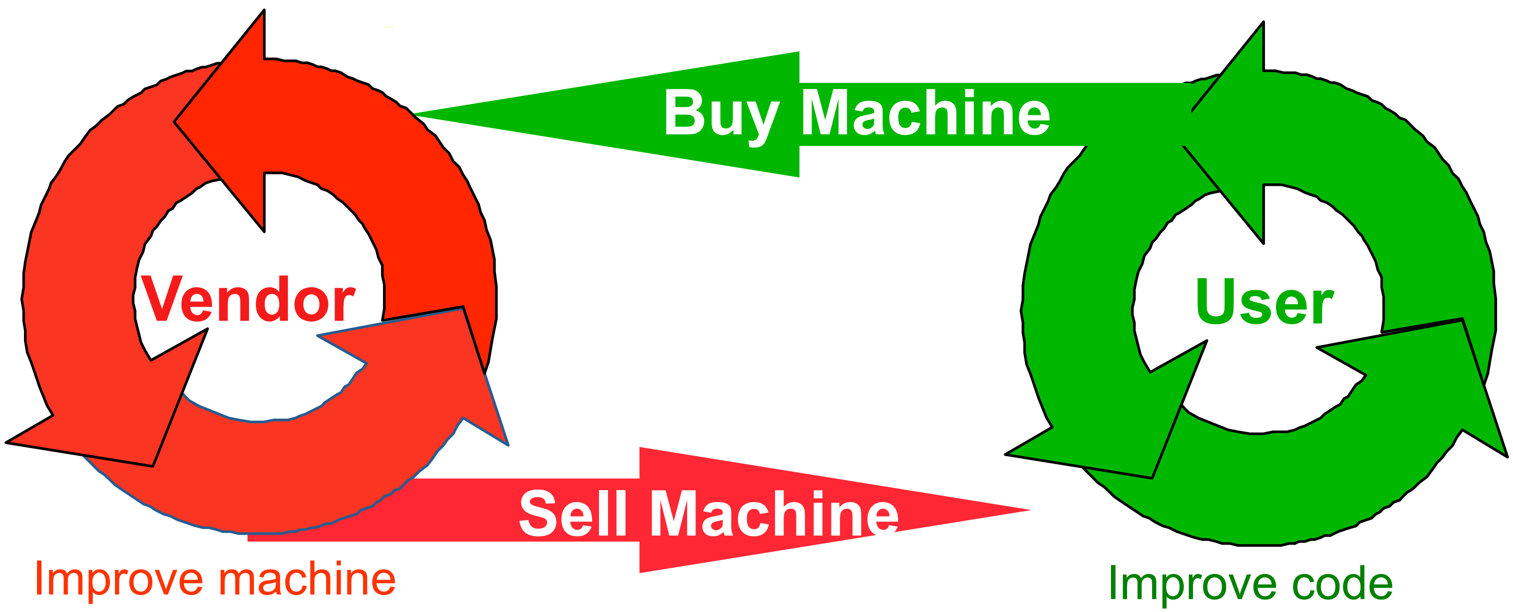
Outline

- Introduction, motivation, some terminology
- Using CrayPat
- Using Apprentice2
- Hands-on lab

Why Analyze Performance?

- **Improving performance on HPC systems has compelling economic and scientific rationales.**
  - Dave Bailey: Value of improving performance of a single application, 5% of machine's cycles by 20% over 10 years: $1,500,000
  - Scientific benefit probably much higher
- **Goal: solve problems faster; solve larger problems**
- **Accurately state computational need**
- **Only that which can be measured can be improved**
- **The challenge is mapping the application to an increasingly more complex system architecture**
  - or set of architectures

Performance Evaluation as an Iterative Process



Overall goal: more / better science results

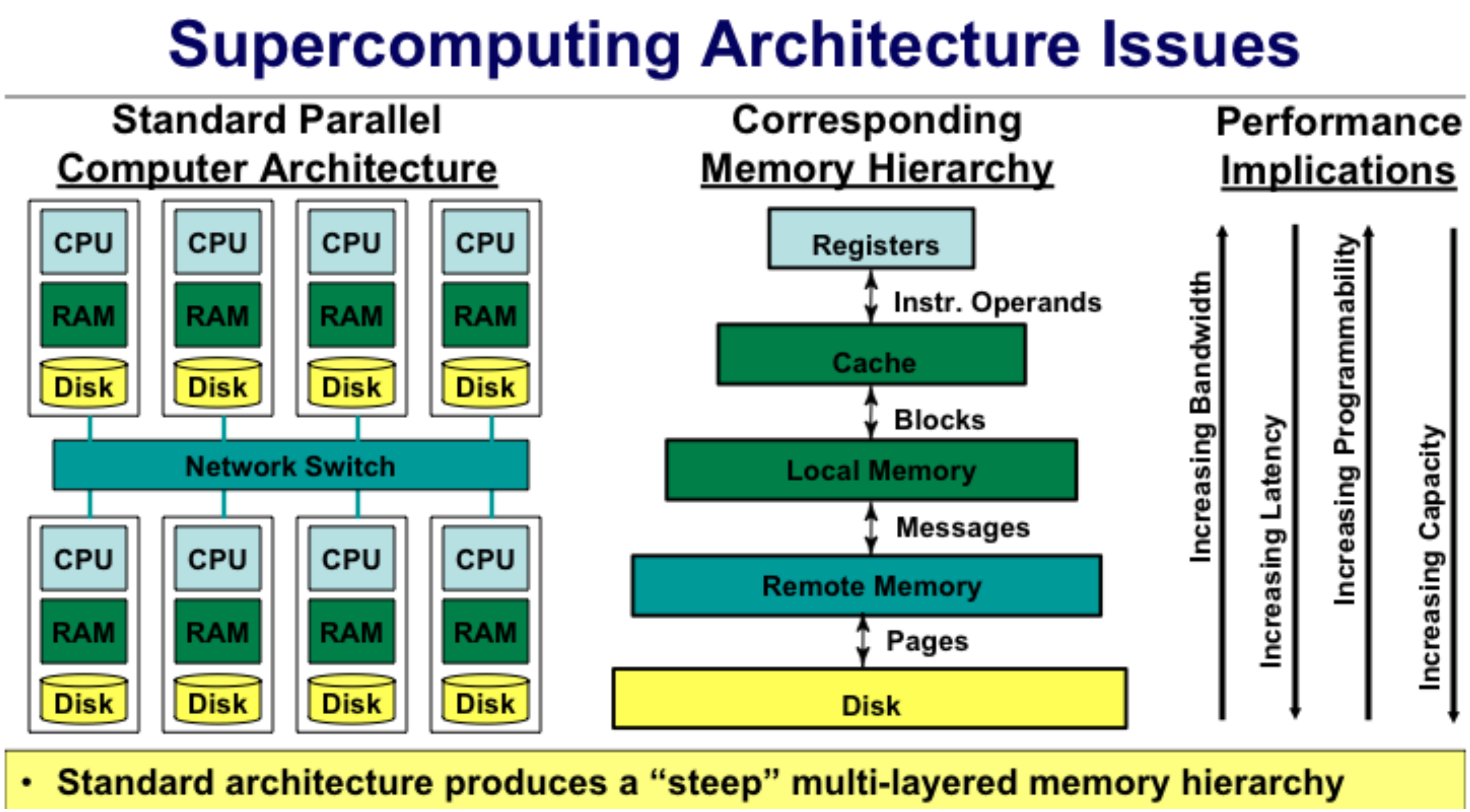
Performance Analysis Issues

- **Difficult process for real codes**
- **Many ways of measuring, reporting**
- **Very broad space: Not just time on one size**
  - for fixed size problem (same memory per processor): Strong Scaling
  - scaled up problem (fixed execution time): Weak Scaling
- **A variety of pitfalls abound**
  - Must compare parallel performance to best *uniprocessor* algorithm, not just parallel program on 1 processor (unless it's best)
  - Be careful relying on any single number
- **Amdahl's Law**

Performance Questions

- How can we tell if a program is performing well?
- Or isn't?
- If performance is not "good," how can we pinpoint why?
- How can we identify the causes?
- What can we do about it?

Supercomputer Architecture



Performance Metrics

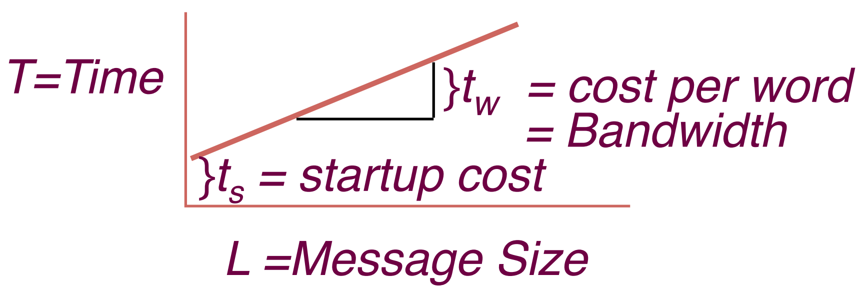
- **Primary metric: application time**
  - but gives little indication of efficiency

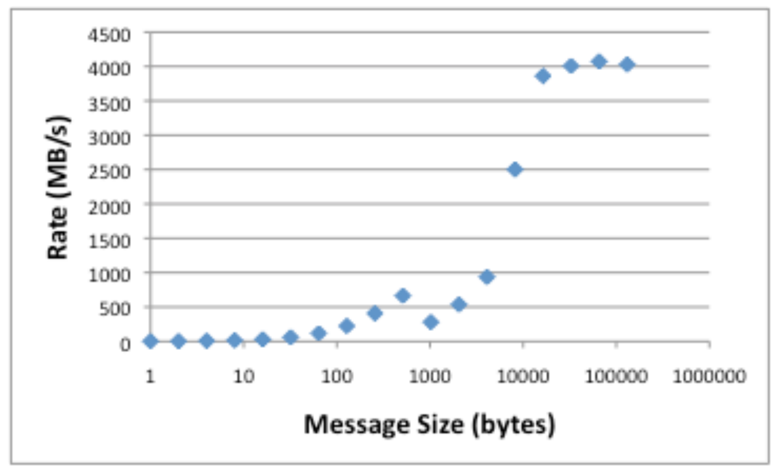
- **Derived measures:**
  - rate (Ex.: messages per unit time, Flops per Second, clocks per instruction), cache utilization

- **Indirect measures:**
  - speedup, efficiency, scalability

Performance Metrics

- **Most basic:**
  - *counts*: how many **`MPI_Send`** calls?
  - *duration*: how much time in **`MPI_Send`** ?
  - *size*: what size of message in **`MPI_Send`**?
- **(MPI performance as a function of message size)**





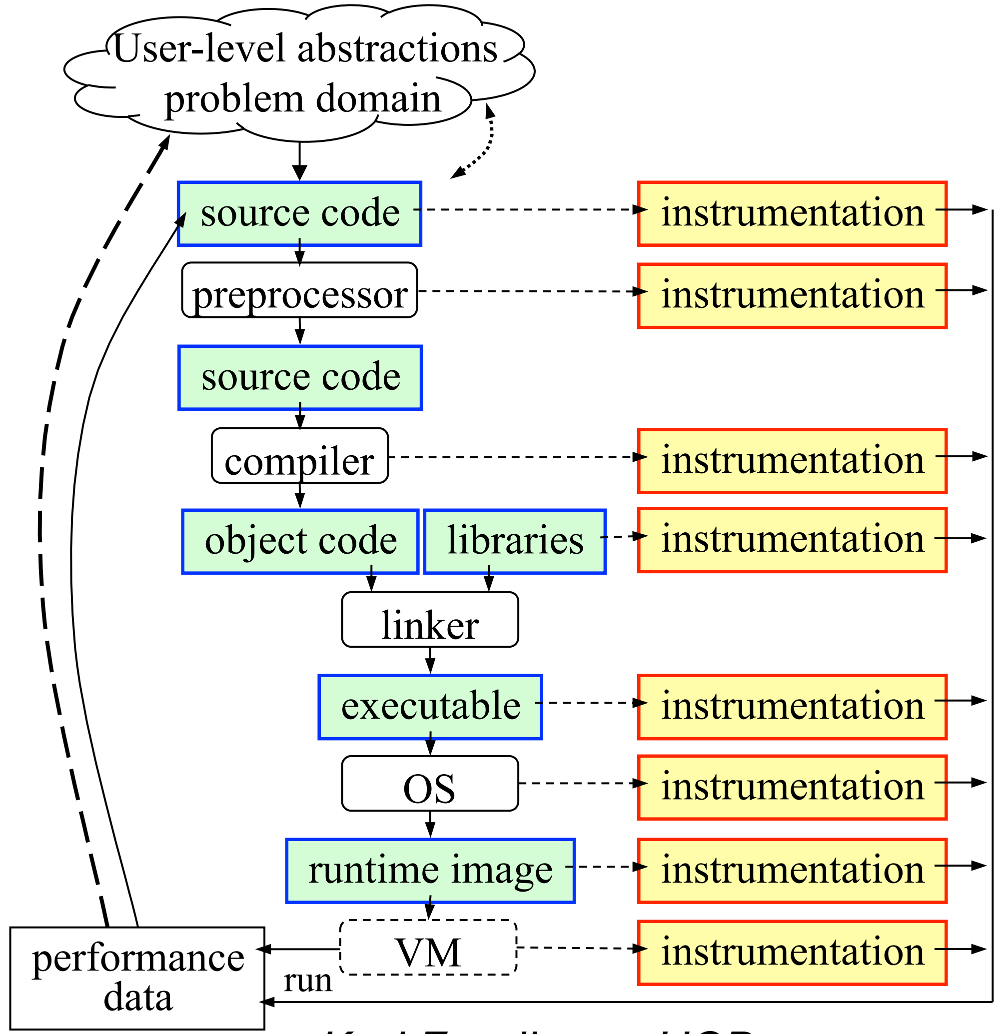
$T_{msg} = t_s + t_w L$

Performance Data Collection

- **Two dimensions:**
- **When data collection is triggered:**
  - Externally (asynchronous): Sampling
    - OS interrupts execution at regular intervals and records the location (program counter) (and / or other event(s))
  - Internally (synchronous): Tracing
    - Event based
    - Code instrumentation, Automatic or manual

Instrumentation

- **Instrumentation: adding measurement probes to the code to observe its execution.**
- **Different techniques depending on where the instrumentation is added.**
- **Different overheads and levels of accuracy with each technique**



Karl Fuerlinger, UCB

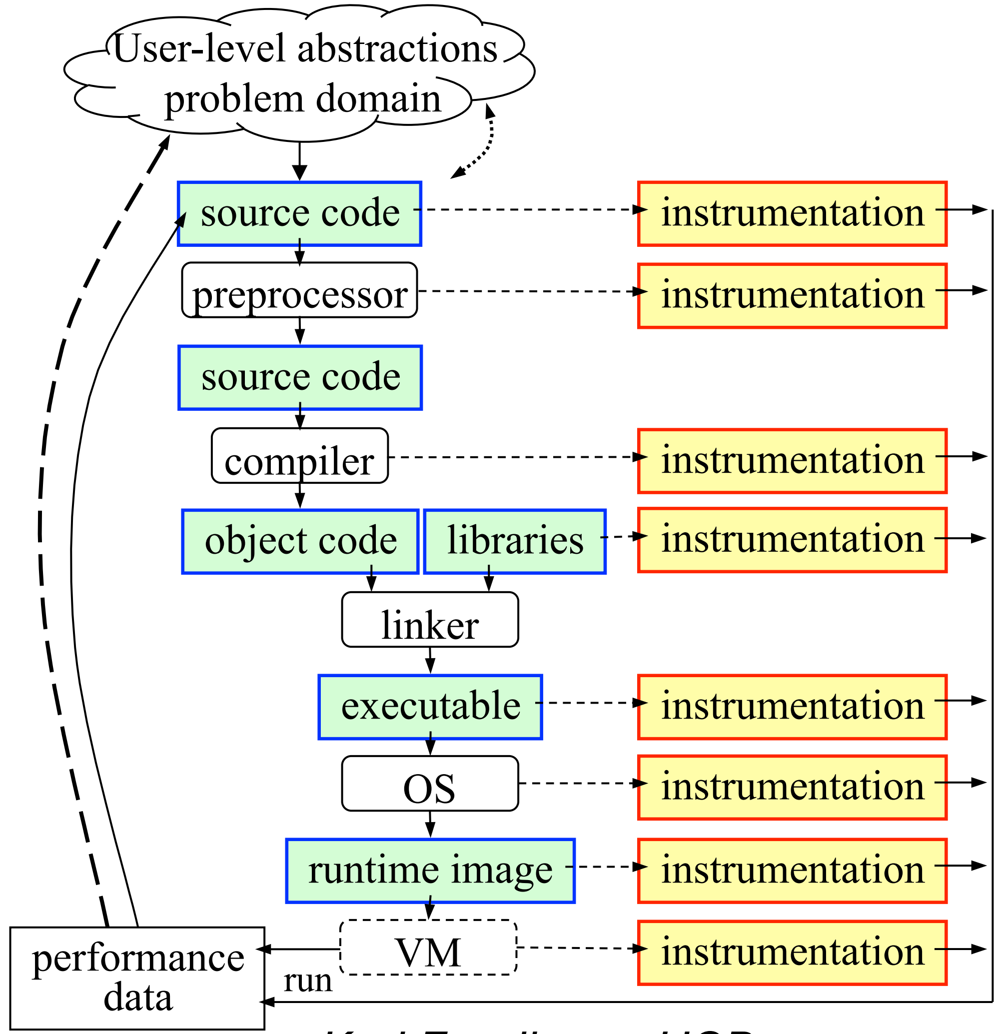
Source-Level Instrumentation

- **Goal is to allow performance measurement *without* modification of user source code**

```
 Timing information:

 Time in timer: IMPVMIXT
Timer number  1 =    10.04 seconds
 Time in timer: IMPVMIXU
Timer number  2 =     0.51 seconds
 Time in timer: TOTAL
Timer number  3 =    17.88 seconds
 Time in timer: STEP
Timer number  4 =    17.88 seconds
 Time in timer: BAROCLINIC
Timer number  5 =    13.88 seconds
 Time in timer: BAROTROPIC
Timer number  6 =     1.27 seconds
-----------------------------------


 POP exiting...
 Successful completion of POP run


-----------------------------------
real 18.09
user 17.97
sys 0.10
```

Instrumentation

- **Instrumentation: adding measurement probes to the code to observe its execution.**
- **Different techniques depending on where the instrumentation is added.**
- **Different overheads and levels of accuracy with each technique**



Karl Fuerlinger, UCB

Performance Instrumentation

- **Approach: use a tool to “instrument” the code**
  1. Transform a binary executable before executing
  2. Include “hooks” for important events
  3. Run the instrumented executable to capture those events, write out raw data file
  4. Use some tool(s) to interpret the data

Performance Data Collection

- **How performance data are presented:**
  - Profile: combine sampled events over time
    - Reflects runtime behavior of program entities
      - functions, loops, basic blocks
      - user-defined “semantic” entities
    - Good for low-overhead performance assessment
    - Helps to expose performance hotspots (“bottleneckology”)
  - Trace file: Sequence of events over time
    - Gather individual time-stamped events (and arguments)
    - Learn when (and where?) events took place on a global timeline
    - Common for message passing events (sends/receives)
    - Large volume of performance data generated; generally intrusive
    - Becomes very difficult at large processor counts, large numbers of events
      - Example in Apprentice section at end of tutorial

Performance Analysis Difficulties

- **Tool overhead**
- **Data overload**
- **User knows the code better than the tool**
- **Choice of approaches**
- **Choice of tools**
- **CrayPat is an attempt to overcome several of these**
  - By attempting to include intelligence to identify problem areas
  - However, in general the problems remain

Performance Tools @ NERSC

- **IPM: Integrated Performance Monitor**
- **Vendor Tools:**
  - CrayPat
- **Community Tools (Not all fully supported):**
  - TAU (U. Oregon via ACTS)
  - OpenSpeedShop (DOE/Krell)
  - HPCToolKit (Rice U)
  - PAPI (Performance Application Programming Interface)

Profiling: Inclusive vs. Exclusive

- **Inclusive time for main:**
  - 100 secs
- **Exclusive time for main:**
  - 100-20-50-20=10 secs
  - Exclusive time sometimes called “self”

```
int main( )
{ /* takes 100 secs */
  f1(); /* takes 20 secs */
 /* other work */
  f2(); /* takes 50 secs */
  f1(); /* takes 20 secs */

 /* other work */
}

/* similar for other metrics,
such as hardware performance
counters, etc. */
```

Woo-Sun Yang

USING CRAYPAT

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

CrayPat Outline

- Introduction
- Sampling (and example)
- Tracing
- .xf files
- pat_report
- Tracing examples: Heap, MPI, OpenMP
- APA (Automatic Program Analysis)
- CrayPat API
- Monitoring hardware performance counters

- Exercises provided: `Exercise info in this box`

Introduction to CrayPat

- **Suite of tools to provide a wide range of performance-related information**
- **Can be used for both sampling and tracing user codes**
  - with or without hardware or network performance counters
- **Supports Fortran, C, C++, UPC, MPI, Coarray Fortran, OpenMP, Pthreads, SHMEM**

Access to Cray Tools

- **Access via module utility**
- **Old:**
  - `module load xt-craypat`
  - `module load apprentice2`
- **Now:**
  - `module load perftools`
  - `xt-craypat`, `apprentice2`, and `xt-papi` (via `xt-craypat`) are loaded

Using CrayPat

1. **Access the tools**
   - `module load perftools`
2. **Build your application; keep .o files**
   - `make clean`
   - `make`
3. **Instrument application**
   - `pat_build ... a.out`
   - Result is a new file, `a.out+pat`
4. **Run instrumented application to get top time consuming routines**
   - `aprun ... a.out+pat`
   - Result is a new file XXXXX.xf (or a directory containing .xf files)

   Adjust script for +pat
5. **Run pat_report on that new file; view results**
   - `pat_report  XXXXX.xf  > my_profile`
   - `vi my_profile`
   - Result is also a new file: XXXXX.ap2

CrayPat Notes

- **Key points to remember:**
  - MUST load module prior to building your code
    - Error message is obscure!

```
ERROR: Missing required ELF section 'link
information'
```

  - MUST load module prior to looking at man pages
  - MUST run your application in $SCRATCH
  - Module name change: xt-craypat → perftools
  - MUST leave relocatable binaries (*.o) when compiling

pat_build for Sampling

- To sample the program counter (PC) at a given time interval or when a specified hardware counter overflows; runs faster than tracing

- To build, use –S or simply without any tracing flag for pat_build
  - `pat_build –S a.out` or
  - `pat_build a.out`

Running a Sampling Experiment

- **To run**
  - Set **`PAT_RT_EXPERIMENT`** to a type
    - Default: **`samp_pc_time`** with default time interval (`PAT_RT_INTERVAL`) of 10,000 microseconds
    - Others: **`samp_pc_ovfl`**, **`samp_cs_time`**, **`samp_cs_ovfl`** (see pat man page)
- **pat_report on .xf from a sampling experiment generates .apa file (later on this)**

Sampling Example

```
> module load perftools
> ftn -c jacobi_serial.f90
> ftn -o jacobi_serial
  jacobi_serial.o
> pat_build jacobi_serial
```

Run jacobi_serial+pat in a batch job.

```
> pat_report
  jacobi_serial+pat+5511-2558sot.xf
```

```
Table 1:  Profile by Function

 Samp % | Samp |Group
        |      | Function

 100.0% | 3598 |Total
|-----------------------
|  99.9% | 3596 |USER
||----------------------
||  73.8% | 2654 |MAIN_
||  24.8% |  891 |compute_diff_
||   1.4% |   50 |init_fields_
|=======================
```

Function Group

Exercise in perftools/sampling

```
Table 2:  Profile by Group, Function, and Line

 Samp % | Samp |Group
        |      | Function
        |      |  Source
        |      |   Line

 100.0% | 3598 |Total
|-----------------------
|  99.9% | 3596 |USER
||----------------------
||  73.8% | 2654 |MAIN_
3|        |      | perftools/sampling/./jacobi_serial.f90
||||--------------------
4|||  48.8% | 1757 |line.36
4|||  24.9% |  897 |line.54
||||====================
||  24.8% |  891 |compute_diff_
3|        |      | perftools/sampling/./jacobi_serial.f90
||||--------------------
4|||  24.7% |  890 |line.147
||||====================
||   1.4% |   50 |init_fields_
3|        |      | perftools/sampling/./jacobi_serial.f90
|=======================
```

```
Table 3:  Wall Clock Time, Memory High Water Mark

   Process |   Process |Total
      Time |     HiMem |
           | (MBytes) |

 57.593670 |      1153 |Total
|===========================
```

More on Sampling

- **Binary built for sampling doesn't work for a tracing experiment**
- **Binary built for tracing can be used for sampling ("trace-enhanced sampling")**
  - Set PAT_RT_EXPERIMENT to a sampling type
  - set PAT_RT_SAMPLING_MODE to 1 (raw user sampling) or 3 (bubble user sampling: aggregate all samples inside a call to a traced function)

pat_build for Tracing

- **To trace entries and returns from functions**
  - Type of experiment: ‘`trace`’; no need to set PAT_RT_EXPERIMENT to it in general
- **`–w`: trace functions specified by `–t` and `–T`**
  - If none is specified, “trace” the ‘main’ (i.e., entire code as a whole)
- **`–u`: trace all user functions routine by routine**
  - For source files owned and writable by the user
  - Use care: binary runs longer and can fail

```
WARNING: Tracing small, frequently called
functions can add excessive overhead
WARNING: To set a minimum size, say 800
bytes, for traced functions, use:
-D trace-text-size=800
```

pat_build for Tracing

- **`-T` *`function`*: trace *function***
  - `pat_build –w –T field_,grad_ a.out`
  - Learn the Unix `nm` or `readelf`:
    `nm mycode.o |grep " T "`
- **`-T !`*`function`*: do not trace *function***
  - `pat_build -u -T \!field_ a.out`
    - trace all user functions except field_
    - '\' to escape the '`!`' character in csh/tcsh
- **`-t` *`tracefile`*: trace all functions listed in the file *`tracefile`*.**

CrayPat Trace Function Groups

- **`–g`: trace all functions in certain function groups (e.g., MPI):**
  - **`pat_build -g mpi,heap -u a.out`**
  - **`pat_build -g mpi -T \!MPI_Barrier a.out`**
    trace all MPI calls except MPI_Barrier
- **See `$CRAYPAT_ROOT/lib/Trace*` for files that list what routines are traced**

- **mpi**
- **omp**
- **pthreads**
- **caf**
- **upc**
- **shmem**
- **ga**
- **heap**
- **blas**
- **blacs**
- **lapack**

- **scalapack**
- **fftw**
- **petsc**
- **io**
- **netcdf**
- **hdf5**
- **lustre**
- **adios**
- **sysio**
- **dmapp**
- **…**

Other pat_build Options

- **`-f`: overwrite an existing instrumented program**
- **`-o` *`instr_prog`*:**
  - use a different name for the instrumented executable instead of a.out+pat
  - can put ***`instr_prog`*** at the end of the command line without '-o'
- **`-O` *`optfile`*: use the pat_build options in the file *`optfile`*.**
  - Special argument '`-O apa`' will be discussed later

`Exercise in perftools/pat_build_examples`

Instrumenting Programs Using Compiler Options

- **Available in Pathscale, GNU, and Cray compilers at NERSC**
- **Requires recompile, link (selected files)**
- **Alternative to pat_build –u**
  - GNU, PathScale:

```
cc –finstrument-functions -c pgm.c
cc –o pgm pgm.c
```

  - Cray compiler:

```
cc –h func_trace -c pgm.c
cc –o pgm pgm.o
```

  - Then

```
pat_build –w pgm
```

.xf Files

- **Experiment data files; binary files**
- **Number of .xf files**
  - Single file (for ≤ 256 PEs) or directory containing multiple (~√PEs) files
  - Can be changed with `PAT_RT_EXPFILE_MAX`
- **Name convention**
  - **a.out+pat+<UNIX_PID>-<NODE_ID>[st][dfot].xf** (or a.out+apa+….xf; see APA)
  - [st][dfot]: [sampling, tracing], [distributed memory, forked process, OpenMP, Pthreads]
- **New one each time you run your application**
- **Can create .xf file(s) in other than the current location by setting `PAT_RT_EXPFILE_DIR`**

pat_report

- **Generates from .xf data file(s) ASCII text report and .ap2 file (to be viewed with Apprentice2)**
  - Create .ap2 file right after .xf file becomes available!
    - .xf file requires the instrumented executable in the original directory (not portable)
    - .ap2 doesn't (self-contained and portable)
- **pat_report on .xf file (or directory containing multiple .xf files) generates**
  - text report to stdout (terminal)
  - .ap2 file
  - .apa file, in case of a sampling experiment
- **Running on .ap2 file generates text report to stdout**

pat_report Options

- `-d`: data items to display (time data, heap data, counter data,…)
- `-b`: how data is aggregated or labeled (group, function, pes, thread, …)
- `-s`: details of report appearance (aggregation, format,…)
- `-O|-b|-d|-s –h`: list all available cases for the option

pat_report Options

- **–O: predefined report types; this is what we should use**
  - `profile`, `callers` (`ca`), `calltree` (`ct`), `ca+src`, `ct+src`, `heap`, `load_balance`, `mpi_callers`, `hwpc`, `nids`, …
  - `heap` and `load_balance` have a few "sub-reports"
    - `load_balance` = `load_balance_program` (for entire program)
      + `load_balance_group` (per function group)
      + `load_balance_function` (per function)
  - Examples:

```
-O profile: -d ti%@0.95,ti,imb_ti,imb_ti%,tr
            -b gr,fu,pe=HIDE
-O callers+src: -d ti%@0.95,ti,tr -b gr,fu,ca,pe=HIDE
                —s show_ca='fu,so,li'
-O load_balance_function: -d ti%@0.95,ti,tr
                          -b gr,fu,pe=[mmm]
```

pat_report Options

- Without `–d`, `-b` or `-O`, a few reports appear by default; dependent on the used trace groups

- `-i` *`instr_prog`*: specify the path for the instrumented executable (if not in the same directory as the .xf file)
- `-o` *`output_file`*: specify the output file name
- `-T`: disable all thresholds (5%)

- pat_report lists the options used in the report – a good place to learn options; try adding an option to the existing ones
- By default, all reports (-O) show either no individual PE values or only the PEs having the maximum, median, and minimum values.
- The suffix `_all` can be appended to any of the pat_report keyword options to show the data for all PEs

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

Heap Memory Example

```
> module load perftools
> ftn -c jacobi_serial.f90
> ftn -o jacobi_serial
  jacobi_serial.o
> pat_build -g heap -u
  jacobi_serial
```

Run jacobi_serial+pat in a batch job.

```
> pat_report
  jacobi_serial+pat+15243-18tot.xf
```

```
Table 1:  Profile by Function Group and Function

 Time % |      Time |  Calls |Group
        |           |        | Function

 100.0% | 0.576433 |   64.0 |Total
|------------------------------------
|  99.9% | 0.575942 |   46.0 |USER
||-----------------------------------
...
||===================================
|   0.1% | 0.000491 |   18.0 |HEAP
||-----------------------------------
||   0.1% | 0.000464 |    7.0 |free
||   0.0% | 0.000022 |   10.0 |malloc
||   0.0% | 0.000005 |    1.0 |calloc
|====================================
```

Heap function group

```
Table 3:  Heap Stats during Main Program

 Tracked |  Total | Total | Tracked | Tracked |Tota
    Heap | Allocs | Frees | Objects |  MBytes |
 HiWater |        |       |     Not |     Not |
  MBytes |        |       |   Freed |   Freed |

  12.466 |     11 |     7 |       4 |   4.836 |Tota
|===================================================
```

```
Table 4:  Heap Leaks during Main Program

 Tracked |  Tracked |  Tracked |Caller
  MBytes |   MBytes |  Objects |
     Not |      Not |      Not |
 Freed % |    Freed |    Freed |

  100.0% |    4.836 |        4 |Total
|------------------------------------
|  99.8% |    4.826 |        3 |(N/A)
|   0.2% |    0.010 |        1 |__hpf_malloc_without_abort
===================================
```

Sometimes not easy to understand

Exercise in perftools/heap

MPI Code Example

- Profiling by MPI functions
- MPI message stats
- load imbalance among MPI tasks

```
> module load perftools
> ftn -c jacobi_mpi.f90
> ftn -o jacobi_mpi
  jacobi_mpi.o
> pat_build -g mpi -u jacobi_mpi
```

Run jacobi_mpi+pat in a batch job.

```
> pat_report
  jacobi_mpi+pat+15207-18tdt.xf
```

```
Exercise in perftools/mpi
```

MPI Code Example

```
Table 1:  Profile by Function Group and Function

 Time % |      Time |Imb. Time |   Imb.  | Calls |Experiment=1
        |           |          | Time %  |       |Group
        |           |          |         |       | Function
        |           |          |         |       |  PE='HIDE'

 100.0% | 0.078172 |        -- |      -- | 124.0 |Total
|------------------------------------------------------------
|  83.2% | 0.065023 |        -- |      -- |  26.0 |USER
||-----------------------------------------------------------
||  60.0% | 0.046878 | 0.003254 |   6.5% |   1.0 |jacobi_mpi_
||  18.7% | 0.014595 | 0.001870 |  11.4% |  10.0 |compute_diff_
...
||===========================================================
|  12.9% | 0.010061 |        -- |      -- |  14.0 |MPI_SYNC
||-----------------------------------------------------------
||   7.9% | 0.006150 | 0.006052 |  49.2% |   4.0 |mpi_bcast_(sync)
||   5.0% | 0.003912 | 0.002366 |  22.5% |  10.0 |mpi_allreduce_(s
||===========================================================
|   3.9% | 0.003088 |        -- |      -- |  84.0 |MPI
||-----------------------------------------------------------
||   3.0% | 0.002330 | 0.011095 |  83.0% |  20.0 |MPI_SENDRECV
||   0.5% | 0.000413 | 0.000051 |  11.0% |  10.0 |MPI_ALLREDUCE
||   0.2% | 0.000173 | 0.000317 |  65.0% |   1.0 |mpi_finalize
||                              |  41.8% |   4.0 |mpi_bcast
||                              |  48.6% |  24.0 |MPI_COMM_SIZE
||   0.0% | 0.000017 | 0.000022 |  57.3% |  24.0 |mpi_comm_rank
||   0.0% | 0.000000 | 0.000001 |  54.6% |   1.0 |MPI_INIT
|============================================================
```

imb = max – avg

imb% = imb/max * npes/(npes-1) * 100%

No per-PE info

Function Groups

per PE

PEs with max, min, median values

```
Table 2:  Load Balance with MPI Message Stats

 Time % |      Time |    MPI |  MPI Msg |   Avg MPI |Experiment=1
        |           |    Msg |    Bytes |  Msg Size |Group
        |           |  Count |          |           | PE[mmm]

 100.0% | 0.078209 |   33.9 | 764697.0 |  22546.35 |Total
|------------------------------------------------------------
|  83.1% | 0.065031 |     -- |       -- |        -- |USER
||-----------------------------------------------------------
||   0.4% | 0.068331 |    -- |       -- |        -- |pe.154
||   0.3% | 0.065538 |    -- |       -- |        -- |pe.93
||   0.3% | 0.053429 |    -- |       -- |        -- |pe.24
||===========================================================
|  12.9% | 0.010065 |     -- |       -- |        -- |MPI_SYNC
||-----------------------------------------------------------
||   0.1% | 0.019155 |    -- |       -- |        -- |pe.68
||   0.1% | 0.009859 |    -- |       -- |        -- |pe.51
||   0.0% | 0.001855 |    -- |       -- |        -- |pe.155
||===========================================================
|   4.0% | 0.003113 |   33.9 | 764697.0 |  22546.35 |MPI
||-----------------------------------------------------------
||   0.1% | 0.014530 |  34.0 | 767896.0 |  22585.18 |pe.24
||   0.0% | 0.002374 |  34.0 | 767896.0 |  22585.18 |pe.132
||   0.0% | 0.001671 |  34.0 | 767896.0 |  22585.18 |pe.75
||===========================================================
```

MPI Code Example

```
Table 3:  MPI Message Stats by Caller

  MPI Msg |    MPI | MsgSz | 4KB<= |Experiment=1
    Bytes |    Msg |  <16B | MsgSz |Function
          |  Count | Count | <64KB | Caller
          |        |       | Count |  PE[mmm]

 764697.0 |   33.9 |   14.0 |   19.9 |Total
|-----------------------------------------------
| 764641.0 |   19.9 |     -- |   19.9 |MPI_SENDRECV
|          |        |        |        | jacobi_mpi_
|||---------------------------------------------
3|| 767840.0 |   20.0 |     -- |   20.0 |pe.32
3|| 767840.0 |   20.0 |     -- |   20.0 |pe.7
3|| 383920.0 |   10.0 |     -- |   10.0 |pe.239
||==============================================
|     40.0 |   10.0 |   10.0 |     -- |MPI_ALLREDUCE
|          |        |        |        | compute_diff_
3          |        |        |        |  jacobi_mpi_
||||--------------------------------------------
4|||     40.0 |   10.0 |   10.0 |     -- |pe.32
4|||     40.0 |   10.0 |   10.0 |     -- |pe.21
4|||     40.0 |   10.0 |   10.0 |     -- |pe.171
||||============================================
|     16.0 |    4.0 |    4.0 |     -- |mpi_bcast
|          |        |        |        | read_params_
3          |        |        |        |  jacobi_mpi_
||||--------------------------------------------
4|||     16.0 |    4.0 |    4.0 |     -- |pe.32
4|||     16.0 |    4.0 |    4.0 |     -- |pe.21
4|||     16.0 |    4.0 |    4.0 |     -- |pe.171
|===============================================
```

Bins by message size

per PE

callers

Level of depth

MPI_SYNC Function Group

- **Time spent waiting at a barrier before entering a collective can be a significant indication of load imbalance.**
- **MPI_SYNC group: for time spent waiting at the barrier before entering the collectives**
- **Actual time spent in the collectives go to the MPI function group**
- **Not to separate these groups, set PAT_RT_MPI_SYNC to 0 before aprun**

```
Table 1:  Profile by Function Group and Function

 Time % |      Time |Imb. Time |   Imb.  | Calls |Experiment=1
        |           |          | Time %  |       |Group
        |           |          |         |       | Function
        |           |          |         |       |  PE='HIDE'

 100.0% | 0.078172 |        -- |      -- | 124.0 |Total
|-------------------------------------------------------------
|  83.2% | 0.065023 |        -- |      -- |  26.0 |USER
||------------------------------------------------------------
||  60.0% | 0.046878 | 0.003254 |   6.5% |   1.0 |jacobi_mpi_
||  18.7% | 0.014595 | 0.001870 |  11.4% |  10.0 |compute_diff_
...
||============================================================
|  12.9% | 0.010061 |        -- |      -- |  14.0 |MPI_SYNC
||------------------------------------------------------------
||   7.9% | 0.006150 | 0.006052 |  49.2% |   4.0 |mpi_bcast_(sync)
||   5.0% | 0.003912 | 0.002366 |  22.5% |  10.0 |mpi_allreduce_(sync)
||============================================================
|   3.9% | 0.003088 |        -- |      -- |  84.0 |MPI
||------------------------------------------------------------
||   3.0% | 0.002330 | 0.011095 |  83.0% |  20.0 |MPI_SENDRECV
||   0.5% | 0.000413 | 0.000051 |  11.0% |  10.0 |MPI_ALLREDUCE
||   0.2% | 0.000173 | 0.000317 |  65.0% |   1.0 |mpi_finalize
||   0.1% | 0.000108 | 0.000077 |  41.8% |   4.0 |mpi_bcast
||   0.1% | 0.000046 | 0.000043 |  48.6% |  24.0 |MPI_COMM_SIZE
||   0.0% | 0.000017 | 0.000022 |  57.3% |  24.0 |mpi_comm_rank
||   0.0% | 0.000000 | 0.000001 |  54.6% |   1.0 |MPI_INIT
|=============================================================
```

U.S. DEPARTMENT OF ENERGY | Office of Science

MPI Rank Order Suggestion

- '`-O mpi_sm_rank_order [-s rank_grid_dim=M,N] [-s rank_cell_dim=m,n] [-s mpi_dest=d]`':
  - Based on sent messages
  - `-s rank_*`: specify a different MPI process topology
    - Global topology, `M×N`
    - topology per node, `m×n`
  - consider `d` busiest partners (default, 8)

```
> pat_report -O mpi_sm_rank_order
  jacobi_mpi+pat+30971-19tdot.xf
> ls MPICH_RANK_ORDER.*
MPICH_RANK_ORDER.d ←
MPICH_RANK_ORDER.u ←
```

Examined the cases 0, 1, 2, and 3 ('d' and 'u') for MPICH_RANK_REORDER_METHOD; provided MPICH_RANK_ORDER file for the 'd' and 'u' cases.

MPI Rank Order Suggestion

```
Table 1:  Sent Message Stats and Suggested MPI Rank Order

                Communication Partner Counts
        Number   Rank
      Partners  Count       Ranks
             1      2       0  239
             2    238       1  2  3  4  ...
--------------------------------------------
                   Sent Msg Total Bytes per
            Max            Avg   Max
      Total Bytes   Total Bytes   Rank

         1.54e+06      1.53e+06  1
------------------------------------------------------------
        24 cores per node:  Sent Msg Total Bytes per node
  Rank          Max       Max/           Avg        Avg/   Max Node
 Order  Total Bytes        SMP   Total Bytes         SMP   Ranks
     1     1.54e+06     100.0%      1.38e+06      100.0%   24 25 26 27 28 29
     d     6.14e+06     400.0%      5.53e+06      400.0%   6
     u     6.14e+06     400.0%      5.53e+06      400.0%   1
     2     3.69e+07    2400.0%      3.32e+07     2400.0%   1
     0     3.69e+07    2400.0%      3.67e+07     2655.6%   1
```

Best: smallest ratios wrt to SMP ord.

```
> cat MPICH_RANK_ORDER.d
# The custom rank placement in this file is the one labeled 'd'
# in the report from:
#
#   pat_report -O mpi_sm_rank_order \
#       -s mpi_dests=8 \
#      /scratch/scratchdirs/wyang/mpi/jacobi_mpi+pat+30971-19td
...
# To use this file, copy it to MPICH_RANK_ORDER and set the
# environment variable MPICH_RANK_REORDER_METHOD to 3 prior
# to executing the program.
#
116,117,118,119,6,7,228,229,4,5,230,231,236,237,1,0,2,3,232,233,2
3,60,61,174,175,64,65,170,171,58,59,176,177,66,67,16
71 164 165 52 53 182 183 72 73 16
```

MPI Rank Order Suggestion

- **'-O mpi_rank_order [-s mro_metric=…]': based on specified metric**
  - 'time', if no metric is specified
  - HWPC if **a.out+pat** was run with this set (later on this)

> pat_report -O mpi_rank_order jacobi_mpi+pat+30971-19tdot.ap2
> ls MPICH_RANK_ORDER.*
MPICH_RANK_ORDER.d

Examined the cases 0, 1, 2, and 3 ('d') for MPICH_RANK_REORDER_METHOD; provided MPICH_RANK_ORDER file for 'd' case.

```
Table 1:  Suggested MPI Rank Order

                 USER Time per MPI rank
           Max          Avg   Max
      USER Time  USER Time   Rank
       2.68e+08   2.55e+08   23
------------------------------------------------------------
       24 cores per node:  USER Time per node
 Rank         Max     Max/          Avg      Avg/   Max Node
Order   USER Time      SMP    USER Time       SMP   Ranks
    0    6.13e+09    99.9%     6.11e+09    100.0%   3,13,23,33,43,53,63,73,83,9
    1    6.13e+09   100.0%     6.11e+09    100.0%   0,1,2,3,4,5,6,7,8,9,10,11,1
    2    6.13e+09   100.0%     6.11e+09    100.0%   3,16,23,36,43,56,63,76,83,9
    d    6.13e+09    99.9%     6.11e+09    100.0%   143,113,64,63,79,105,26,54,
```

OpenMP Code Example

```
> module load perftools
> ftn -mp=nonuma -c jacobi_omp.f90
> ftn -mp=nonuma -o jacobi_omp
   jacobi_omp.o
> pat_build -g omp -u jacobi_omp
```

Run jacobi_omp+pat in a batch job

```
> pat_report
   jacobi_omp+pat+15307-18tot.xf
```

`Exercise in perftools/openmp`

OpenMP Code Example

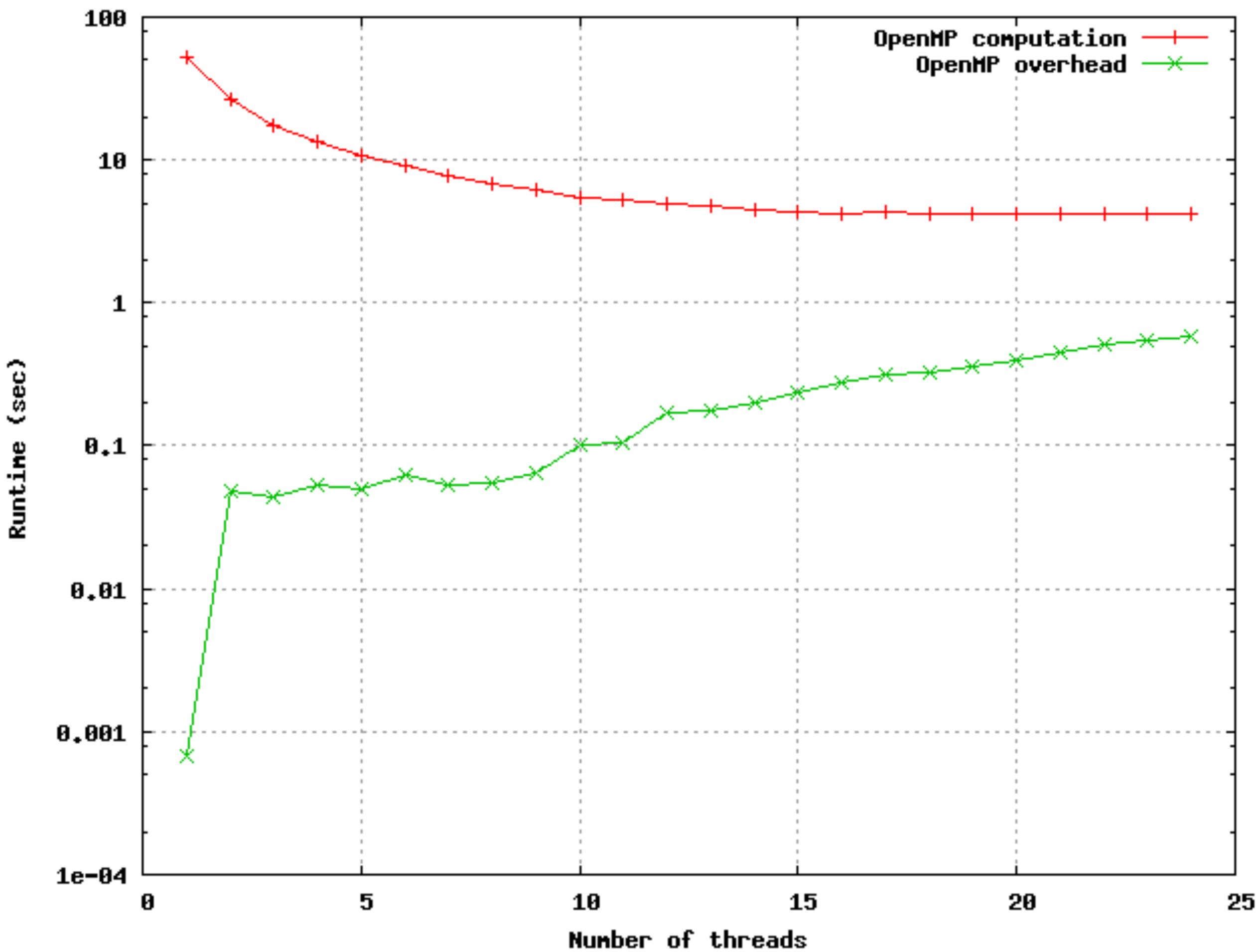
```
Table 1:  Profile by Function Group and Function

 Time % |      Time |Imb. Time |   Imb. | Calls |Group
        |           |          | Time % |       | Function
        |           |          |        |       |  Thread='HIDE'

 100.0% | 4.128724 |        -- |     -- | 379.0 |Total
|---------------------------------------------------------------
|  88.9% | 3.671375 |        -- |     -- | 253.0 |USER
||--------------------------------------------------------------
||  36.8% | 1.519932 | 0.007403 |   0.5% |  20.0 |MAIN_.LOOP@li.35
||  14.1% | 0.582412 | 0.001967 |   0.4% |  20.0 |compute_diff_.LOOP@li.166
||  13.7% | 0.565131 | 0.012983 |   2.4% |  20.0 |MAIN_.LOOP@li.56
||  11.4% | 0.471882 | 0.000751 |   0.2% |   1.0 |init_fields_.LOOP@li.104
||  11.4% | 0.470750 | 0.000434 |   0.1% |   1.0 |init_fields_.LOOP@li.108
||   1.2% | 0.049400 | 0.047252 | 100.0% |   1.0 |MAIN_
...
||==============================================================
|  11.0% | 0.453033 |        -- |     -- | 104.0 |OMP
||--------------------------------------------------------------
||  10.9% | 0.449830 | 0.430272 | 100.0% |   1.0 |init_fields_.REGION@li.104(ovhd)
||   0.0% | 0.001494 | 0.001429 | 100.0% |  20.0 |MAIN_.REGION@li.35(ovhd)
||   0.0% | 0.000885 | 0.000847 | 100.0% |  20.0 |compute_diff_.REGION@li.166(ovhd)
||   0.0% | 0.000343 | 0.000329 | 100.0% |   1.0 |omp_set_num_threads
||   0.0% | 0.000278 | 0.000266 | 100.0% |  21.0 |set_bc_.REGION@li.138(ovhd)
||   0.0% | 0.000123 | 0.000118 | 100.0% |  20.0 |MAIN_.REGION@li.56(ovhd)
||   0.0% | 0.000079 | 0.000076 | 100.0% |  21.0 |set_bc_.REGION@li.145(ovhd)
||==============================================================
|   0.1% | 0.004316 |        -- |     -- |  22.0 |PTHREAD
|        |          |          |        |       | pthread_create
|===============================================================
```

No per-thread info

- **24 thread case**
- **Good load balance in parallel regions!**
- **Overhead:**
  - Imb. =100%!, all in the master thread (pat_report's '–O load_balance' to see this more clearly)
  - largest in init_fields_

OpenMP Code Example

jacobi_omp.f90 with ngrind=9,600 and maxiter=20



Automatic Program Analysis (APA)

- One may not know in advance where large run time is spent; tracing all the functions can be overwhelming due to large overhead

1. Have the tool detect the most time consuming functions in the application with a sampling experiment

2. Feed this information back to the tool to instrument for focused data collection

3. Get performance information on the most significant parts of the application

- APA does this for you (you can do the same thing by hand)

Automatic Program Analysis (APA)

1. **pat_build -O apa a.out**
   - Produces the instrumented executable a.out+pat for sampling
2. **aprun -n … a.out+pat**
   - Produces data file, e.g., a.out+pat+4677-19sdot.xf
3. **pat_report a.out+pat+4571-19sdot.xf**
   - Produces a.out+pat+4571-19sdot.apa (suggested options for tracing)
   - Produces a.out+pat+4571-19sdot.ap2
4. **Edit a.out+pat+4571-19sdot.apa, if necessary (next slide)**
5. **pat_build -O a.out+pat+4571-19sdot.apa**
   - Produces a.out+apa for tracing
6. **aprun -n … a.out+apa**
   - Produces a.out+apa+4590-19tdot.xf
7. **pat_report a.out+apa+4590-19tdot.xf > out**

`Exercise in perftools/apa`

.apa File

- **Recommended pat_build options for tracing**
- **Customize it for your need**
  - Include/exclude functions
  - Add/change options
- **Note that the suggestions may not be valid for a very different task/thread configuration**

```
#  You can edit this file, if desired, and use it
#  to reinstrument the program for tracing like this:
#
#          pat_build -O jacobi_mpi+pat+24908-19sdot.apa
#
#  These suggested trace options are based on data from:
#
#    /scratch/scratchdirs/wyang/

# ------------------------------

#       HWPC group to collect by default.

  -Drtenv=PAT_RT_HWPC=1  # Summary with TLB metrics.

# ------------------------------

#       Libraries to trace.

  -g mpi

# ------------------------------------------------------------------------

#       User-defined functions to trace, sorted by % of samples.
...
  -w  # Enable tracing of user-defined functions.
      # Note: -u should NOT be specified as an additional option.

# 54.17%  1798 bytes
         -T MAIN_

# 16.67%  296 bytes
#         -T compute_diff_
...
# ------------------------

  -o jacobi_mpi+apa                              # New instrumented program.

  /scratch/scratchdirs/wyang/apa/jacobi_mpi  # Original program
```

Command to run

PAT_RT_HWPC set to 1

mpi trace group chosen

Trace ‘MAIN_’, but not ‘compute_diff_’

Instrumented program to create

Sampling run result

Program to instrument

U.S. DEPARTMENT OF ENERGY | Office of Science

CrayPat Application Program Interface (API)

- **Assume your code contains initialization and solution sections**
- **Want to analyze performance only of solution**
- **How to do this? Several approaches:**
  - Init section is only one routine (or just a few routines): eliminate it (or them) from the profile.
  - Init section is many routines: Use API to define a profile region that excludes init
  - What happens if some routines shared by init and solve? Use API to turn profiling on and off as needed

`Exercise in perftools/api`

Using the API

```
include 'mpif.h'
include 'pat_apif.h'
...
call pat_record(PAT_STATE_OFF, ierr)
call mpi_init(ierr)
...
! Main solver loop.

call pat_record(PAT_STATE_ON, ierr)

h = 1.0 / n

do k=1,maxiter
  call pat_region_begin(1, 'compute_unew', ierr)
  ...
  call pat_region_end(1, ierr)

  call set_bc(unew,n,js,je)
  ...
  call compute_diff(u,unew,n,js,je,diffnorm)
  ...
  call pat_region_begin(2, 'set_u_to_unew', ierr)
  u(:,:) = unew(:,:)
  call pat_region_end(2, ierr)
  ...
enddo
call pat_record(PAT_STATE_OFF, ierr)
...
```

Arbitrary, user-assigned id's

```
> module load perftools
> ftn -c jacobi_mpi_api.f90
> ftn -o jacobi_mpi_api jacobi_mpi_api.o
> pat_build -g mpi -u jacobi_mpi_api
```

Run jacobi_mpi_api+pat in a batch job.

```
> pat_report
  jacobi_mpi_api+pat+502-19tdot.xf
```

```
Time %  |     Time |Imb. Time |   Imb.  | Calls |Group
        |          |          | Time %  |       | Function
        |          |          |         |       |  PE='HIDE'

100.0% | 2.571502 |       -- |      -- | 122.0 |Total
|-----------------------------------------------------------
|  98.9% | 2.543296 |       -- |      -- |  42.0 |USER
||----------------------------------------------------------
||  45.1% | 1.158653 | 0.008723 |   0.8% |  10.0 |#1.compute_unew
||  22.0% | 0.566156 | 0.008736 |   1.6% |  10.0 |#2.set_u_to_unew
||  21.6% | 0.555785 | 0.005339 |   1.0% |  10.0 |compute_diff_
||  10.2% | 0.262097 | 0.174330 |  40.8% |   1.0 |MAIN_
||   0.0% | 0.000600 | 0.001415 |  71.7% |  10.0 |set_bc_
||   0.0% | 0.000006 | 0.000004 |  38.7% |   1.0 |main
||==========================================================
|   0.9% | 0.022937 |       -- |      -- |  10.0 |MPI_SYNC
|        |          |          |         |       | mpi_allreduce_(sync)
|   0.2% | 0.005269 |       -- |      -- |  70.0 |MPI
||----------------------------------------------------------
||   0.2% | 0.004835 | 0.007111 |  60.8% |  20.0 |MPI_SENDRECV
||   0.0% | 0.000402 | 0.000064 |  14.0% |  10.0 |MPI_ALLREDUCE
||   0.0% | 0.000025 | 0.000007 |  23.4% |  20.0 |MPI_COMM_SIZE
||   0.0% | 0.000007 | 0.000002 |  19.0% |  20.0 |mpi_comm_rank
|===========================================================
```

Hardware Performance Counters

- **Registers available on the processor that count certain events**
- **Minimal overhead**
  - They're running all the time
  - Typically one clock period to read
- **Potentially rich source of performance information**

Types of Counters

- Cycles
- Instruction count
- Memory references, cache hits/misses
- Floating-point instructions
- Resource utilization

PAPI Event Counters

- **PAPI (Performance API) provides a standard interface for use of the performance counters in major microprocessors**

- **Predefined actual and derived counters supported on the system**
  – To see the list, run ‘papi_avail’ on compute node via aprun:

```
module load perftools
aprun –n 1 papi_avail
```

- **AMD native events also provided; use ‘papi_native_avail’:**

```
aprun –n 1 papi_native_avail
```

Hardware Performance Monitoring

- **Specify hardware counters to be monitored during sampling or tracing**
  – Default is “off” (no HW counters measured)
  – Choose up to 4 events

- **Can specify individual events:**

```
setenv PAT_RT_HWPC “PAPI_FP_OPS,PAPI_L1_DCM”
aprun –n … a.out+pat
```
(or `a.out+apa`)

- **Or predefined event group number (next slide):**

```
setenv PAT_RT_HWPC 1
aprun –n … a.out+pat
```
(or `a.out+apa`)

- **Multiplexing (monitoring more than 4 events) to be supported in later versions (5.2?)**

`Exercise in perftools/hwpc`

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

Predefined Counter Groups for PAT_RT_HWPC

0 Summary with instruction metrics
1 Summary with translation lookaside buffer (TLB) metrics
2 L1 and L2 cache metrics
3 Bandwith information
4 *** DO NOT USE, not supported on Quad-core or later AMD Opteron processors ***
5 Floating point instructions
6 Cycles stalled and resources empty
7 Cycles stalled and resources full
8 Instructions and branches
9 Instruction cache values
10 Cache hierarchy
11 Floating point instructions (2)
12 Floating point instructions (vectorization)
13 Floating point instructions (single precision)
14 Floating point instructions (double precision)
15 L3 cache
16 L3 cache, core-level reads
17 L3 cache, core-level misses
18 L3 cache, core-level fills caused by L2 evictions
19 Prefetches

Hardware Performance Monitoring

PAT_RT_HWPC = 1

```
USER / MAIN_
------------------------------------------------------------------------
  Time%                                                72.3%
  Time                                              7.428213 secs
  Calls                               0.1 /sec           1.0 calls
  PAPI_L1_DCM                     4.392M/sec      32625209 misses
  PAPI_TLB_DM                     1.224M/sec       9091457 misses
  PAPI_L1_DCA                   996.063M/sec    7398982001 refs
  PAPI_FP_OPS                  2232.437M/sec   16583040487 ops
  User time (approx)              7.428 secs   15599292538 cycles  100.0%Time
  Average Time per Call                             7.428213 secs
  CrayPat Overhead : Time          0.0%
  HW FP Ops / User time        2232.437M/sec   16583040487 ops  26.6%peak(DP)
  HW FP Ops / WCT              2232.437M/sec
  Computational intensity           1.06 ops/cycle    2.24 ops/ref
  MFLOPS (aggregate)            2232.44M/sec
  TLB utilization                 813.84 refs/miss   0.795 avg uses
  D1 cache hit,miss ratios         99.6% hits          0.4% misses
  D1 cache utilization (misses)   226.79 refs/miss  14.174 avg hits
```

Measured

Derived

avg uses (or hits): per word per miss

`pat_report –s data_size=4 …`
(because single precision was used)

cacheline: 64 Bytes
L1 cache: 64 KB, dedicated for each core
L2 cache: 512 KB, dedicated for each core
page: 4 KB (2 MB if huge page)

Hardware Performance Monitoring

PAT_RT_HWPC = 2

```
USER / MAIN_
------------------------------------------------------------------------
  Time%                                               72.5%
  Time                                             7.515183 secs
  Calls                             0.1 /sec            1.0 calls
  DATA_CACHE_REFILLS:
    L2_MODIFIED:L2_OWNED:
    L2_EXCLUSIVE:L2_SHARED       30.787M/sec     231368270 fills
  DATA_CACHE_REFILLS_FROM_SYSTEM:
    ALL                          61.353M/sec     461083431 fills
  PAPI_L1_DCM                     4.383M/sec      32937797 misses
  PAPI_L1_DCA                   984.555M/sec    7399124478 refs
  User time (approx)              7.515 secs   15781931557 cycles  100.0%Time
  Average Time per Call                            7.515183 secs
  CrayPat Overhead : Time          0.0%
  D1 cache hit,miss ratios        99.6% hits          0.4% misses
  D1 cache utilization (misses)  224.64 refs/miss   14.040 avg hits
  D1 cache utilization (refills)  10.69 refs/refill  0.668 avg uses
  D2 cache hit,miss ratio         33.4% hits         66.6% misses
  D1+D2 cache hit,miss ratio      99.7% hits          0.3% misses
  D1+D2 cache utilization        337.36 refs/miss   21.085 avg hits
  System to D1 refill            61.353M/sec     461083431 lines
  System to D1 bandwidth       3744.720MB/sec  29509339584 bytes
  D2 to D1 bandwidth           1879.073MB/sec  14807569280 bytes
```

Hardware Performance Monitoring

PAT_RT_HWPC = 5

```
USER / MAIN_
------------------------------------------------------------------------
  Time%                                          72.9%
  Time                                        7.654353 secs
  Calls                        0.1 /sec            1.0 calls
  RETIRED_MMX_AND_FP_INSTRUCTIONS:
    PACKED_SSE_AND_SSE2   964.770M/sec    7384702007 instr
  PAPI_FML_INS            241.562M/sec    1849000244 ops
  PAPI_FAD_INS            301.927M/sec    2311057161 ops
  PAPI_FDV_INS                   1 /sec            6 ops
  User time (approx)        7.654 secs   16074189254 cycles   100.0%Time
  Average Time per Call                       7.654353 secs
  CrayPat Overhead : Time   0.0%
  HW FP Ops / Cycles                              0.26 ops/cycle
  HW FP Ops / User time   543.488M/sec    4160057405 ops   6.5%peak(DP)
  HW FP Ops / WCT         543.488M/sec
  FP Multiply / FP Ops                           44.4%
  FP Add / FP Ops                                55.6%
  MFLOPS (aggregate)        543.49M/sec
```

Relative ratios for multiplies and adds

Hardware Performance Monitoring

PAT_RT_HWPC = 12

```
USER / MAIN_
------------------------------------------------------------------------
  Time%                                               72.5%
  Time                                             7.501585 secs
  Calls                             0.1 /sec            1.0 calls
  RETIRED_SSE_OPERATIONS:
    SINGLE_ADD_SUB_OPS:
    SINGLE_MUL_OPS              553.802M/sec     4154398320 ops
  RETIRED_SSE_OPERATIONS:
    DOUBLE_ADD_SUB_OPS:
    DOUBLE_MUL_OPS                    1 /sec              7 ops
  RETIRED_SSE_OPERATIONS:
    SINGLE_ADD_SUB_OPS:
    SINGLE_MUL_OPS:OP_TYPE     2210.601M/sec    16583040480 ops
  RETIRED_SSE_OPERATIONS:
    DOUBLE_ADD_SUB_OPS:
    DOUBLE_MUL_OPS:OP_TYPE            1 /sec              7 ops
  User time (approx)              7.502 secs    15753376049 cycles  100.0%Time
  Average Time per Call                            7.501585 secs
  CrayPat Overhead : Time         0.0%
```

Add and multiply instructions issued

Adds and multiplies performed

vector length (for sp) for 128-bit wide SSE2 vector operation

= 16583040480 / 4154398320 = 3.99

Compiled with ‘ftn –fastsse …’

Guidelines to Identify the Need for Optimization

Derived metric	Optimization needed when*	PAT_RT_HWPC
Computational intensity	< 0.5 ops/ref	0, 1
L1 cache hit ratio	< 90%	0, 1, 2
L1 cache utilization (misses)	< 1 avg hit	0, 1, 2
L1+L2 cache hit ratio	< 92%	2
L1+L2 cache utilization (misses)	< 1 avg hit	2
TLB utilization	< 0.9 avg use	1
(FP Multiply / FP Ops) or (FP Add / FP Ops)	< 25%	5
Vectorization	< 1.5 for dp; 3 for sp	12 (13, 14)

* Suggested by Cray

Monitoring Network Performance Counters

- **Use PAT_RT_NWPC instead of PAT_RT_HWPC**

- **See ‘Overview of Gemini Hardware Counters’, S-0025-10**
  – http://docs.cray.com

Harvey Wasserman

USING CRAY'S APPRENTICE TOOL

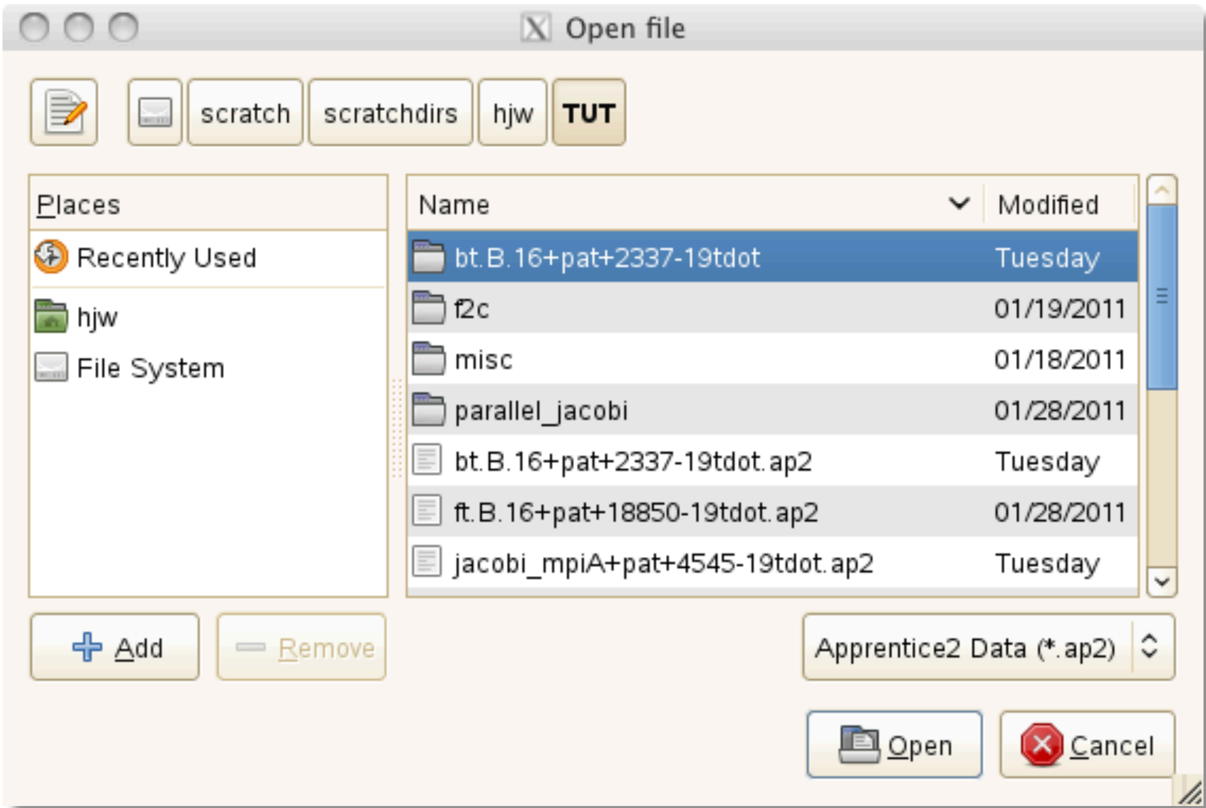
Using Apprentice

- Optional visualization tool for Cray's perftools data
- Use it in a X Windows environment
- Uses a data file as input (`XXX.ap2`) that is prepared by `pat_report`

```
1. module load perftools
2. ftn -c mpptest.f
3. ftn -o mpptest mpptest.o
4. pat_build -u -g mpi mpptest
5. aprun -n 16 mpptest+pat
6. pat_report mpptest+pat+PID.xf >
   my_report
7. app2 [--limit_per_pe tags] [XXX.ap2]
```

Opening Files

- **Identify files on the command line or via the GUI:**



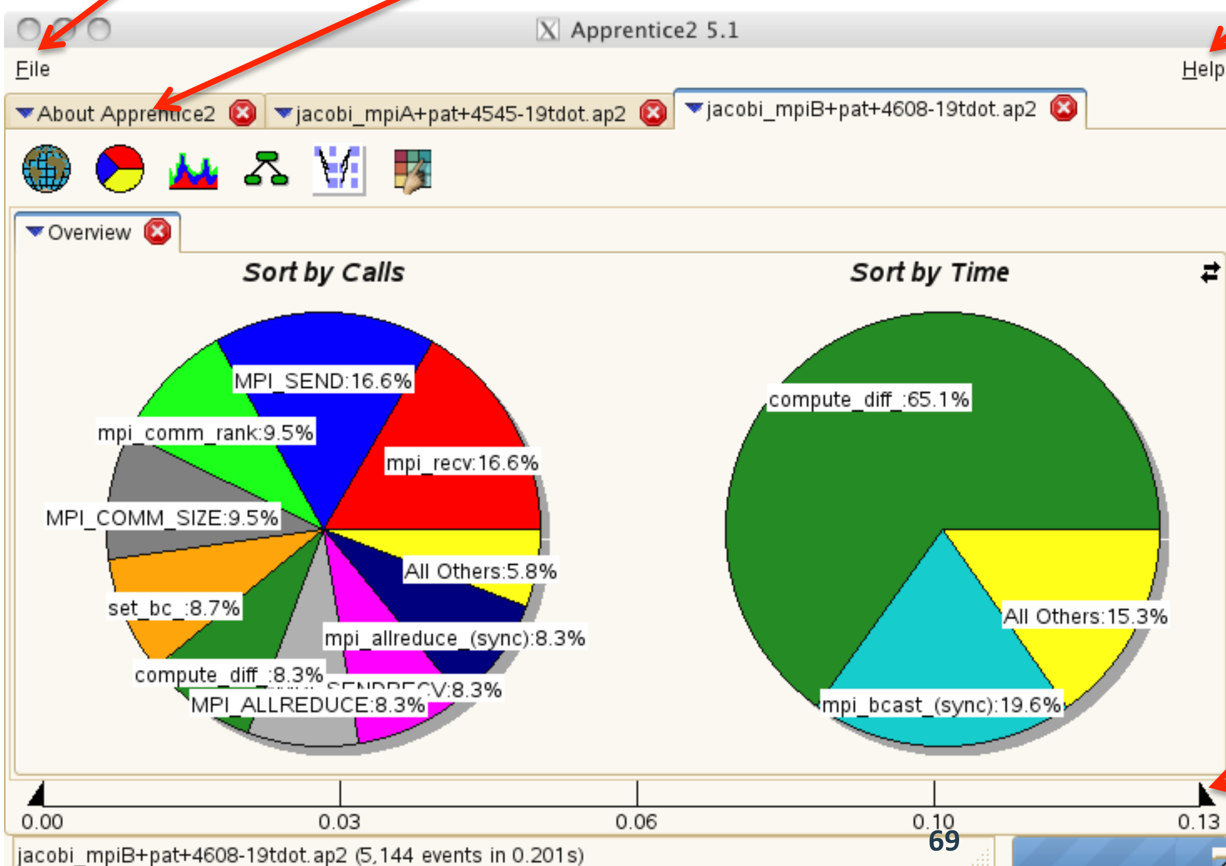
Apprentice Basic View

Can select new (additional) data file and do a screen dump

Worthless

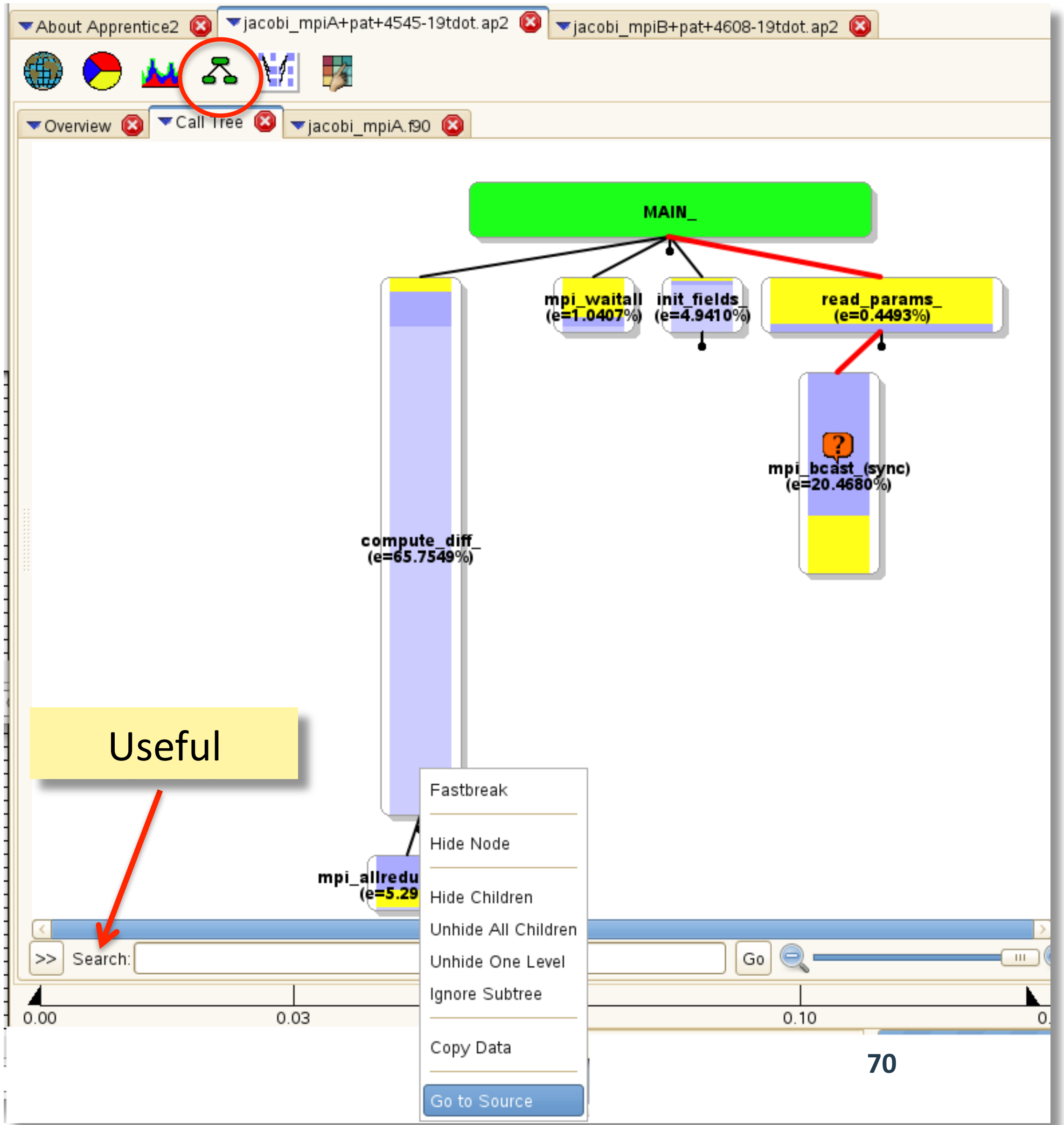
Useful

Can select other views of the data



Can drag the "calipers" to focus the view on portions of the run

Apprentice Call Tree Report



Useful

Horizontal size = cumulative time in node's children

Vertical size = time in computation

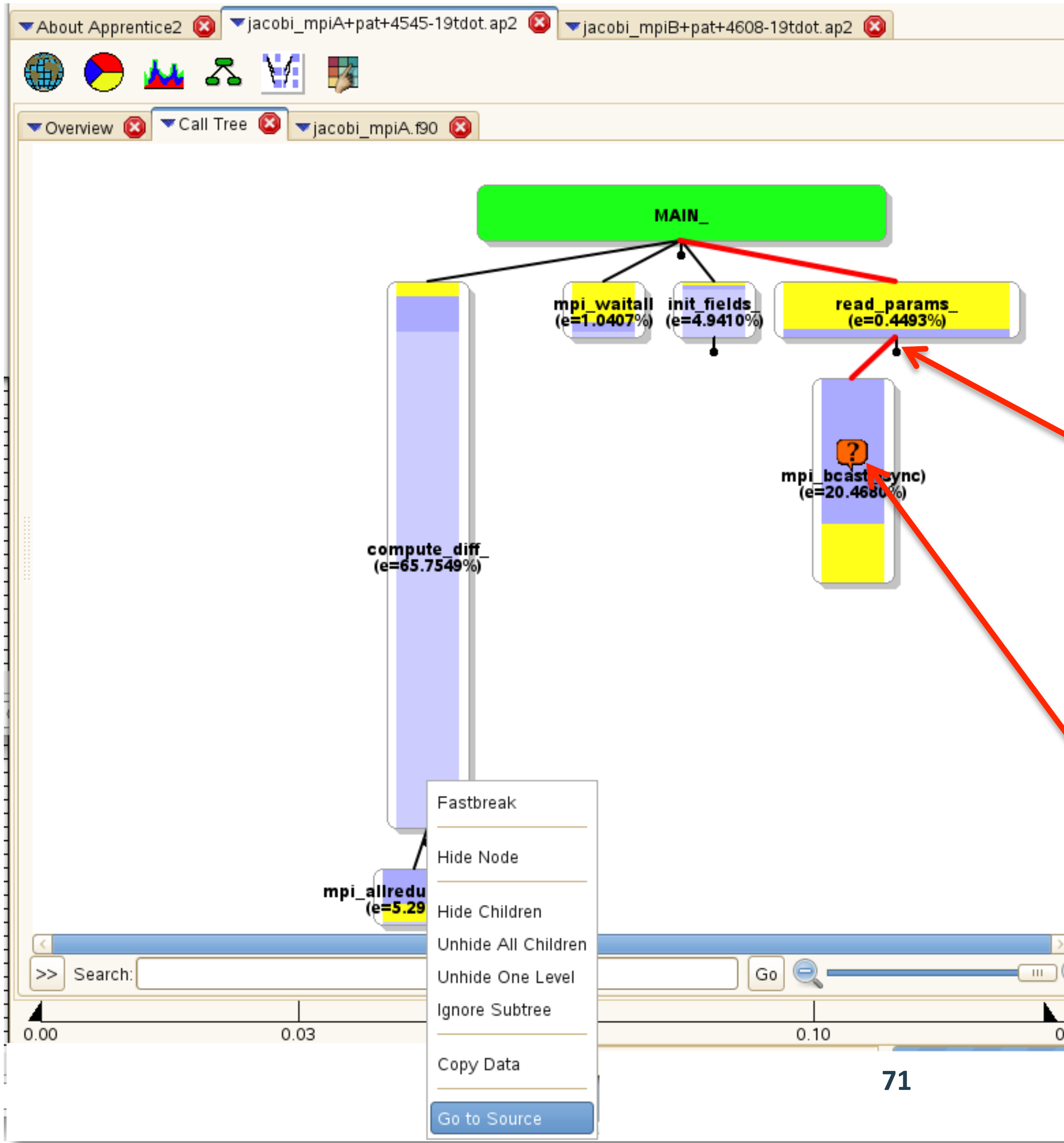
Green nodes: no callees

Stacked bar charts: load balancing info.
Yellow=Max
purple=Average
Light Blue=Minimum

Calipers work

Right-click to view source

Apprentice Call Tree Report



Red arc identifies path to the highest detected load imbalance.

Call tree stops there because nodes were filtered out. To see the hidden nodes, right-click on the node attached to the marker and select "unhide all children" or "unhide one level".

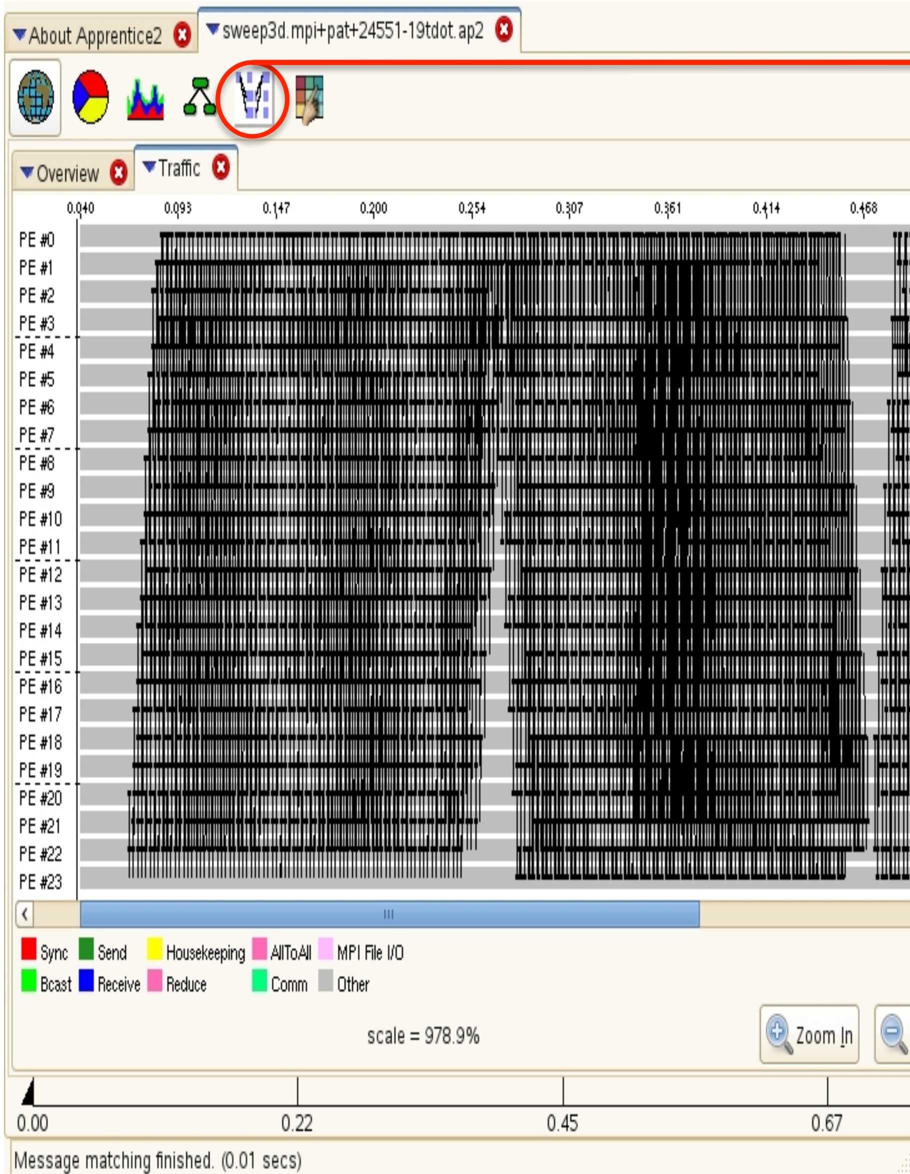
Double-click on [?] for more info about load imbalance.

Apprentice Event Trace Views

- Run code with
  `setenv PAT_RT_SUMMARY 0`

- Caution: Can generate enormous data files and take *forever*

Apprentice Traffic Report



Click here to select this report

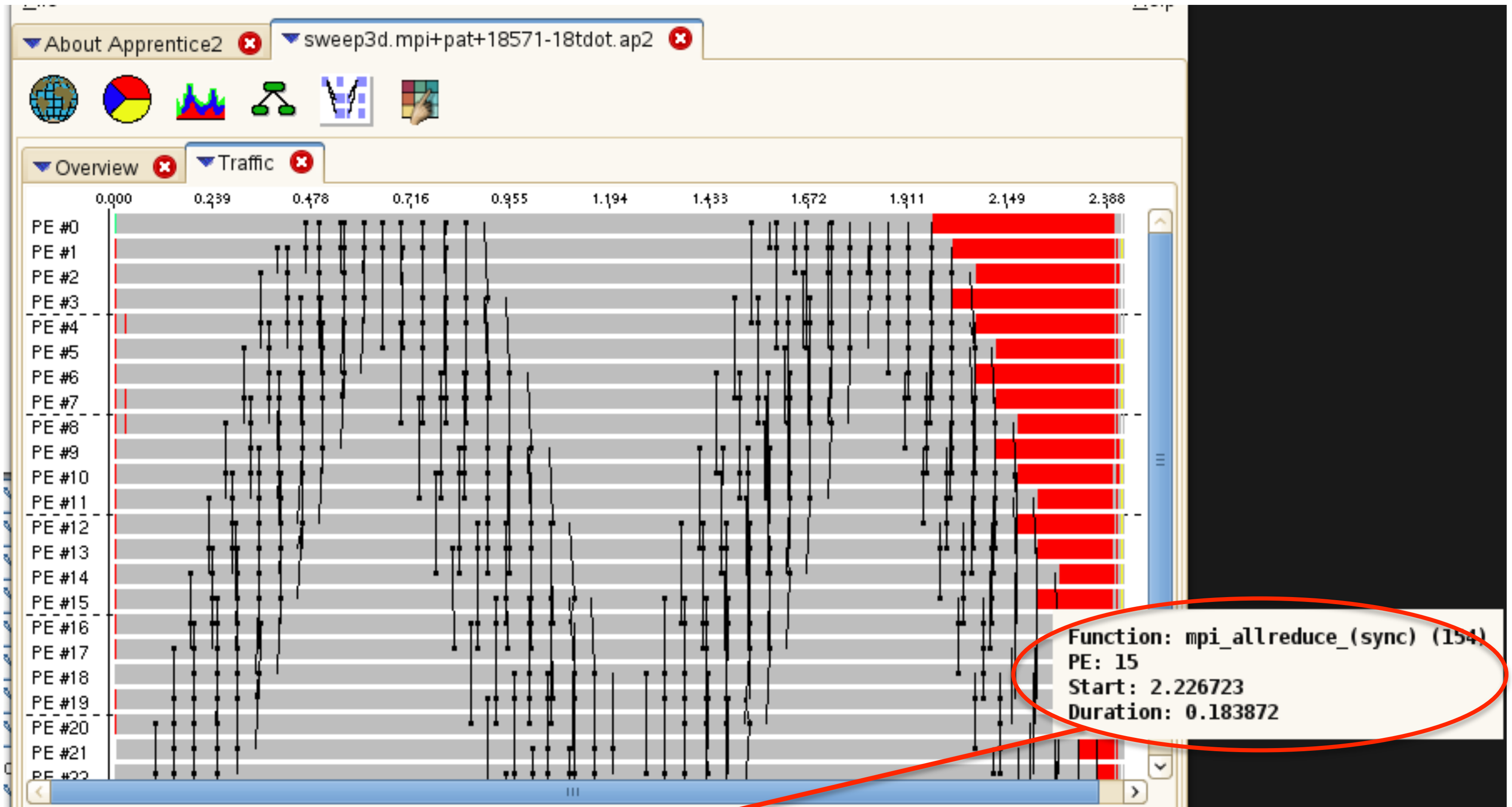
Shows message traces as a function of time

Look for large blocks of barriers held up by a single processor

Zoom is important; also, run just a portion of your simulation

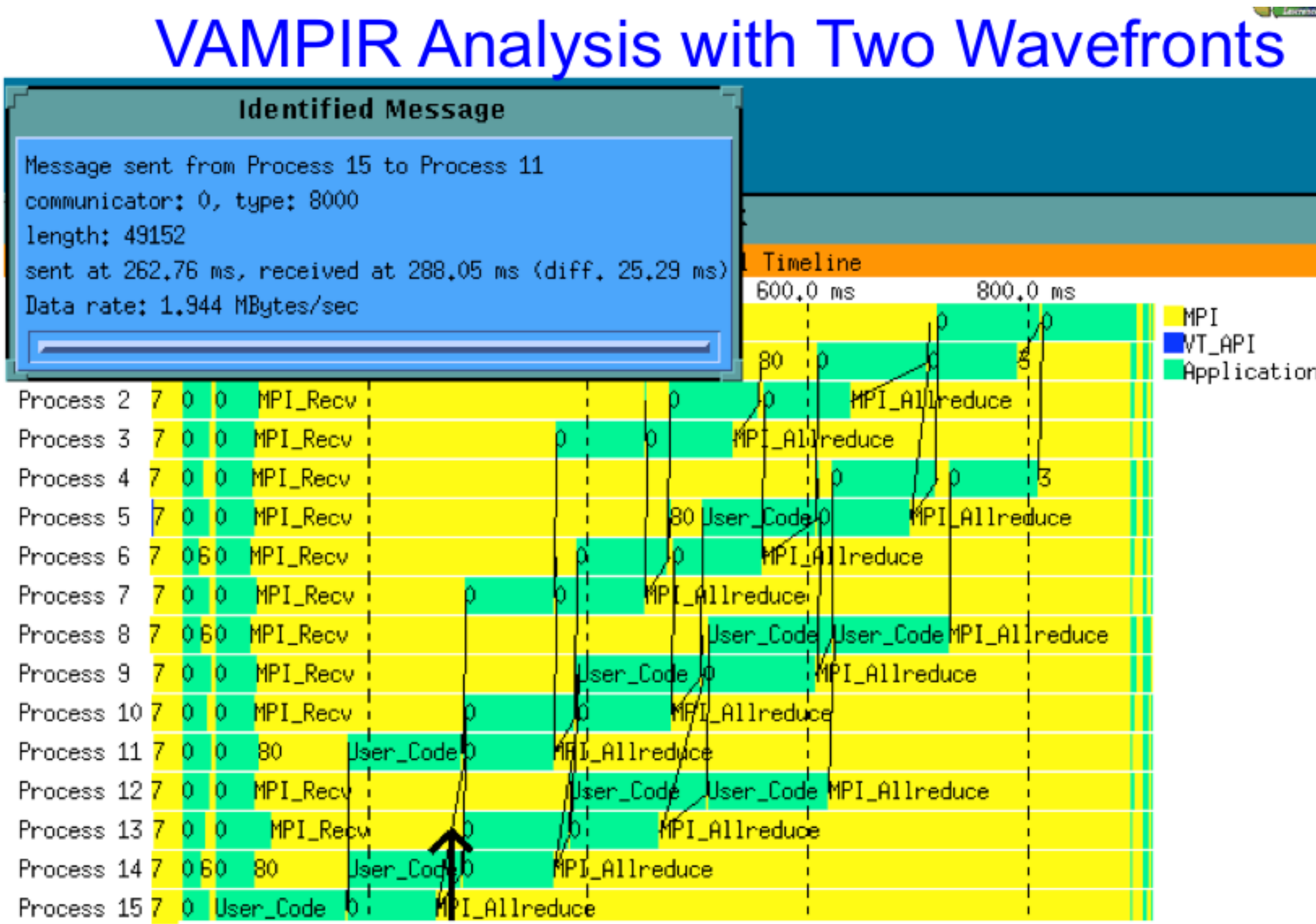
Scroll, zoom, filter: right-click on trace

Apprentice Traffic Report: Zoomed

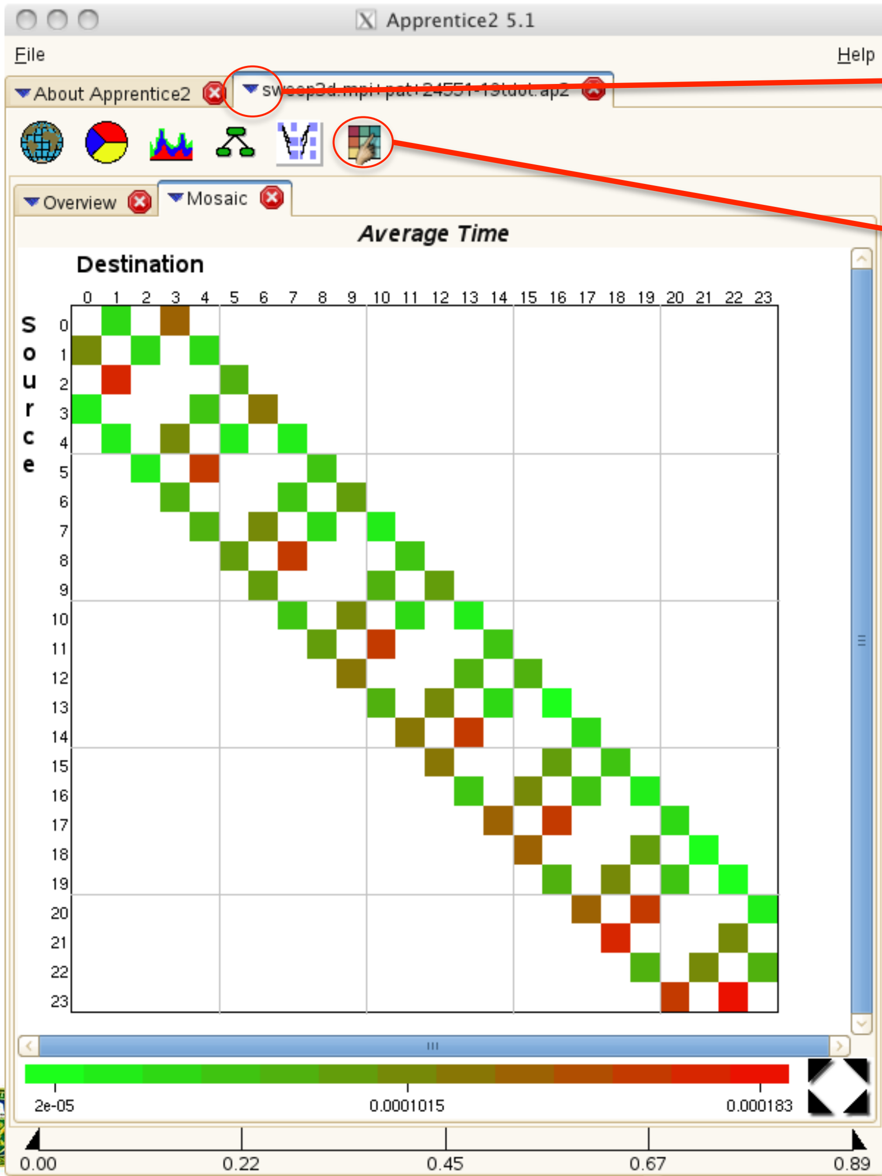


- **Mouse hover pops up window showing source location.**

Tracing Analysis Example



Mosaic View



Can right-click here for more options

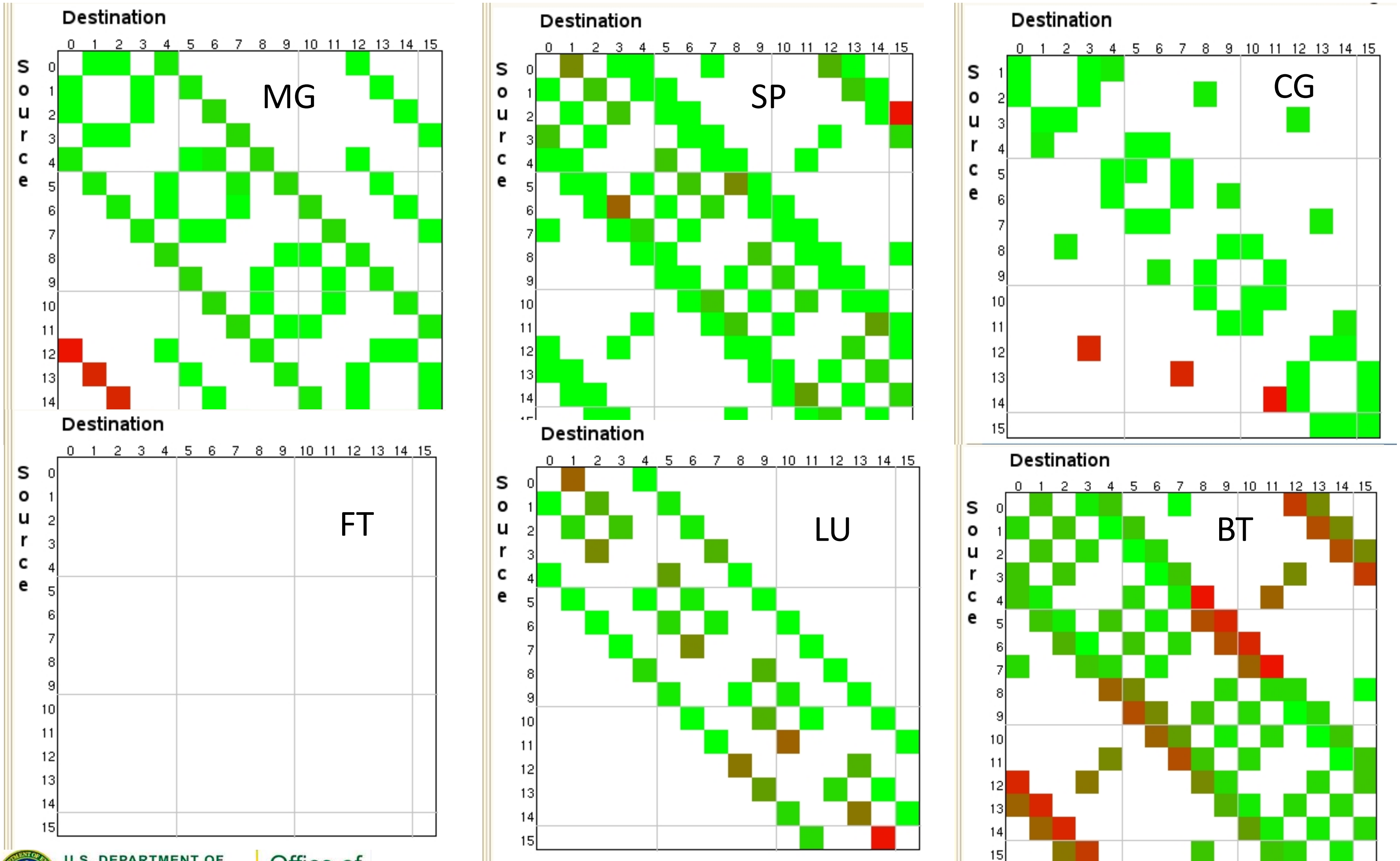
Click here to select this report

Shows Interprocessor communication topology and color-coded intensity

Colors show average time (green=low, red=high)

Very difficult to interpret by itself – use the Craypat message statistics with it.

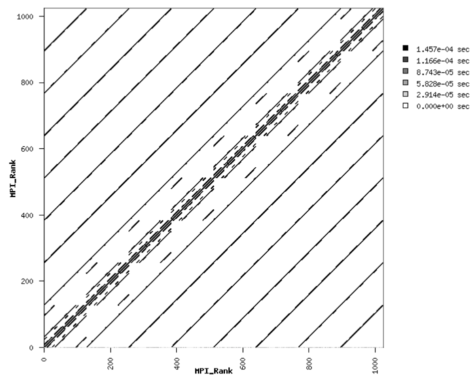
Mosaic View



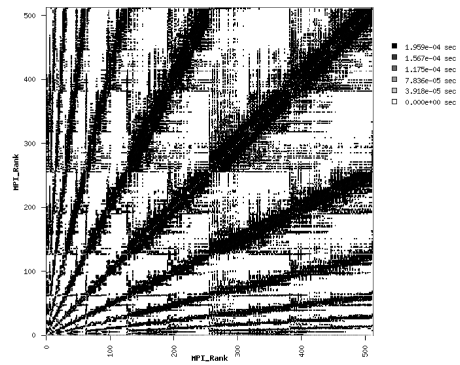
NERSC6 Application Benchmark Characteristics

Benchmark	*Science Area*	*Algorithm Space*	*Base Case Concurrency*	*Problem Description*
CAM	Climate (BER)	Navier Stokes CFD	56, 240 Strong scaling	D Grid, (~.5 deg resolution); 240 timesteps
GAMESS	Quantum Chem (BES)	Dense linear algebra	384, 1024 (Same as Ti-09)	DFT gradient, MP2 gradient
GTC	Fusion (FES)	PIC, finite difference	512, 2048 Weak scaling	100 particles per cell
IMPACT-T	Accelerator Physics (HEP)	PIC, FFT component	256,1024 Strong scaling	50 particles per cell
MAESTRO	Astrophysics (HEP)	Low Mach Hydro; block structured-grid multiphysics	512, 2048 Weak scaling	16 32^3 boxes per proc; 10 timesteps
MILC	Lattice Gauge Physics (NP)	Conjugate gradient, sparse matrix; FFT	256, 1024, 8192 Weak scaling	8x8x8x9 Local Grid, ~70,000 iters
PARATEC	Material Science (BES)	DFT; FFT, BLAS3	256, 1024 Strong scaling	686 Atoms, 1372 bands, 20 iters

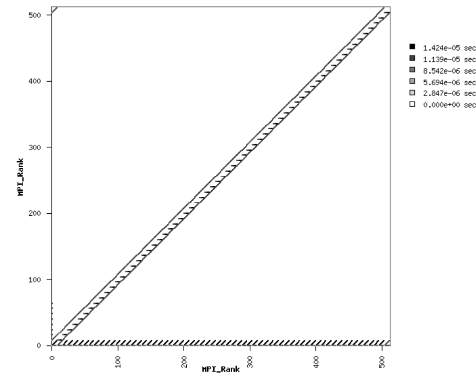
NERSC6 Benchmarks Communication Topology*



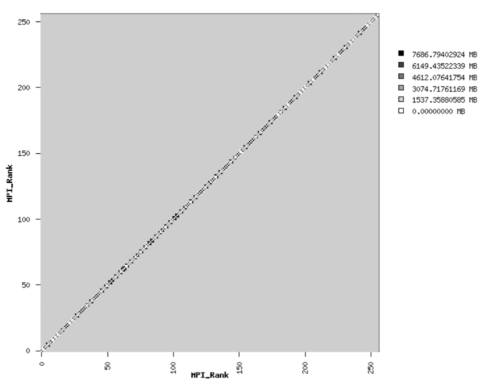
MILC



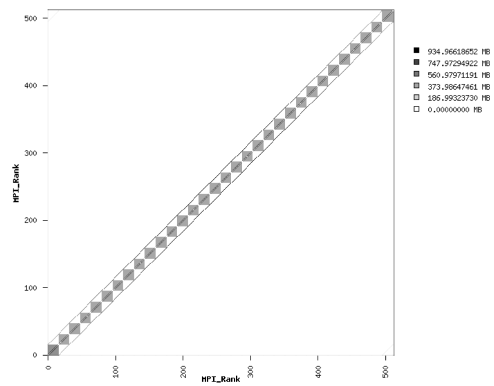
MAESTRO



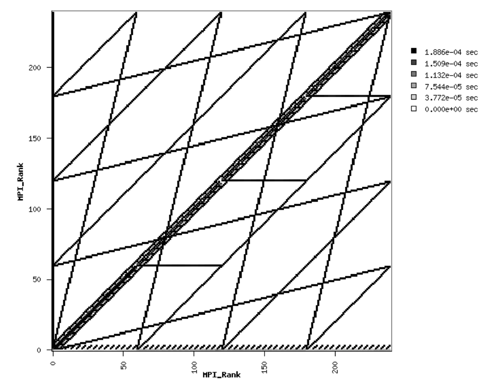
GTC



PARATEC



IMPACT-T



CAM

*From IPM

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB

Sample of CI & %MPI

	Code						
	MAESTRO	GAMESS	CAM	IMPACT-T	GTC	MILC	PARATEC
CI*	0.24	0.61	0.67	0.77	1.15	1.39	1.50
Cray XT4 % peak per core (largest case)	5%	12%	13%	14%	24%	14%	44%
Cray XT4 % MPI medium	20%		29%	9%	4%	12%	27%
Cray XT4 % MPI large			35%	40%	6%	23%	64%
Cray XT4 % MPI extra large	n/a	n/a	n/a	n/a	n/a	30%	n/a
Cray XT4 avg msg size med	2 K	n/a	113 K	35 KB	1 MB	16 KB	34 KB

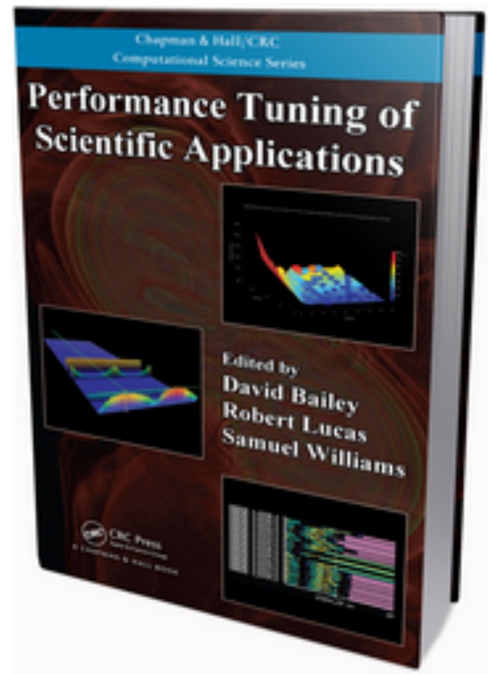
*CI is the computational intensity, the ratio of # of Floating Point Operations to # of memory operations.

For More Information

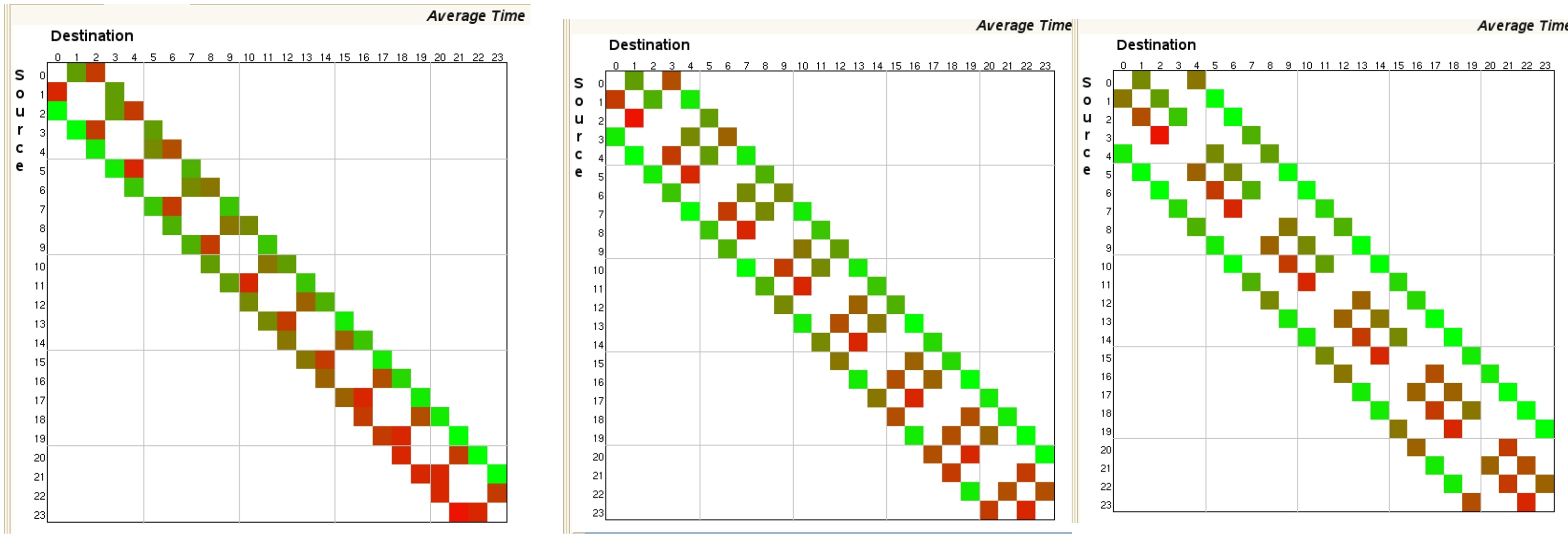
- **Using Cray Performance Analysis Tools, S–2376–51**
  - http://docs.cray.com/books/S-2376-51/S-2376-51.pdf
- **man craypat**
- **man pat_build**
- **man pat_report**
- **man pat_help ← very useful tutorial program**
- **man app2**
- **man hwpc**
- **man intro_perftools**
- **man papi**
- **man papi_counters**

For More Information

- **“Performance Tuning of Scientific Applications,” CRC Press 2010**



Exercise



Same code, same problem size, run on the same 24 cores. What is different? Why might one perform better than the other? What performance characteristics are different?

Exercise

- **Get the sweep3d code. Untar**
- **To build: type 'make mpi'**
- **Instrument for mpi, user**
- **Get an interactive batch session, 24 cores**
- **Run 3 sweep3d cases on 24 cores creating Apprentice traffic/mosaic views:**
  - `cp input1 input; aprun –n 24 …`
  - `cp input2 input; aprun –n 24 …`
  - `cp input3 input; aprun –n 24 …`
- **View the results from each run in Apprentice and try to explain what you see.**

Performance Metrics

Application

Compiler

Architecture

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

Instruction Set

Technology

$$\text{CPU Time} = N_{inst} * \text{CPI} * \text{Clock rate}$$