TACC
TEXAS ADVANCED COMPUTING CENTER
WWW.TACC.UTEXAS.EDU

TEXAS
The University of Texas at Austin

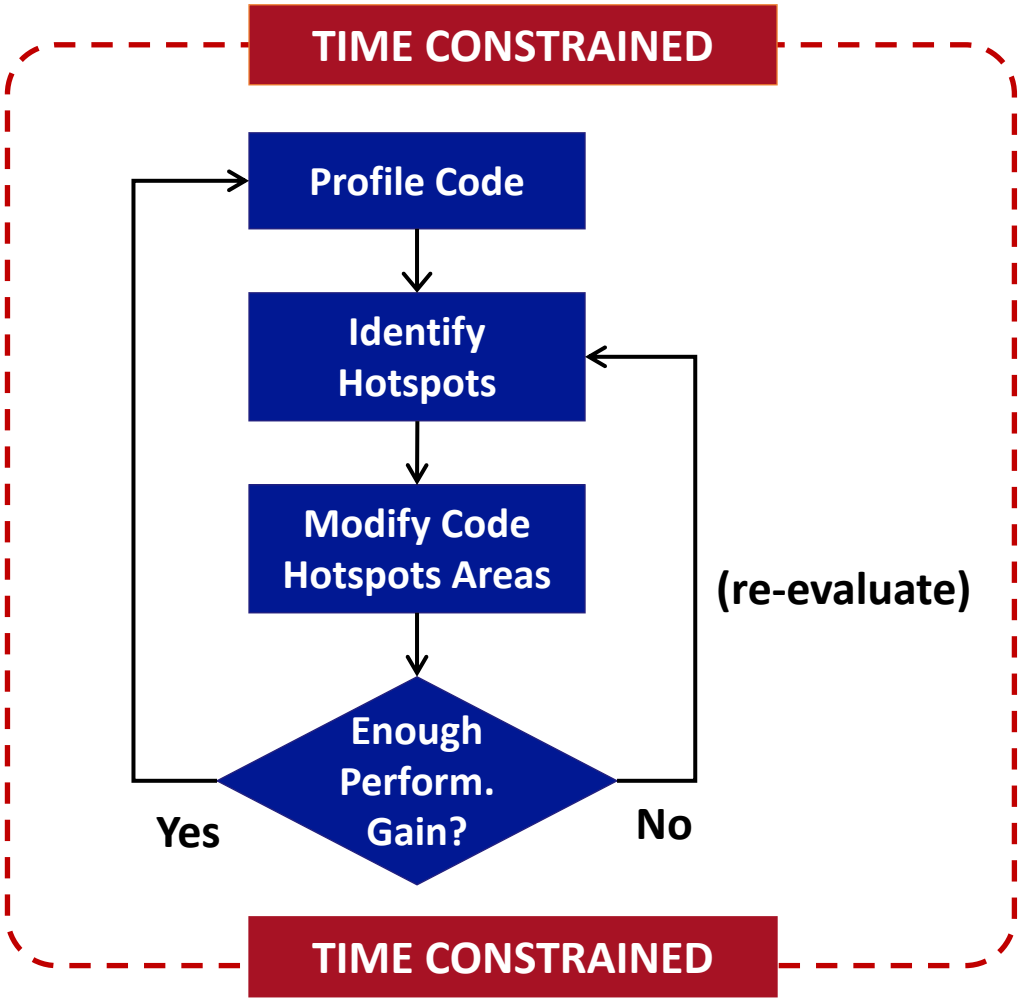
Optimizing for Multicore: Profiling Tools

PRESENTED BY:

Virginia Trueheart, Ian Wang, Cyrus Proctor, John Cazes

https://tinyurl.com/tacc-pearc-2019

Optimization Process



- Iterative process
- Application dependent
- Different levels
  - Compiler Options
  - Performance Libraries
  - Code Optimizations

Performance Analysis Basics

Controlled measurements are essential

- ***Control***

  Don't let the O/S decide process & memory affinity – **control it** !

- ***Repeat***

  No single measurement is reliable

- ***Automate***

  Don't try to remember how you did something – **script it** !

- ***Document***

  - Save all these:
    - ✓ **Code versions**
    - ✓ **Compilers / flags**
    - ✓ **User inputs**
    - ✓ **System environment**
    - ✓ **Timers / counters**
    - ✓ **Code output**

  Record Details:

  Important routines should track work done and time taken

Basic Tools

CML Timer | Code Timer | gprof

Timer: Command Line

The command **time** is available in most Unix systems.

It is simple to use (no code instrumentation required).

Gives total execution time of a process and all its children in seconds.

- `$ /usr/bin/time -p ./exe`

POSIX format

Elapsed wall clock time — `real 9.95`

Time spent in:
- user mode — `user 9.86`
- kernel mode — `sys 0.06`

Leave out the `-p` option to get additional information:

```
$ /usr/bin/time ./exe
$ 9.860u 0.060s 0:09.95 99.9%      0+0k 0+0io 0pf+0w
```

User | System | Real | %CPU | Shared Unshared | Input Output | Page faults Swap

The format option (-f)

Example: `$ /usr/bin/time -f "\t%U user,\t%S system,\t%x status" ./exe`

	Tag	Description	Tag	Description	
Time	%E	Elapsed real time in [hours:]minutes:seconds	%F	Number of major page faults that occurred while the process was running	MEM
	%e	Elapsed real time in seconds	%R	Number of minor, or recoverable, page faults	
	%S	Total number of CPU-seconds that the process spent in kernel mode	%W	Number of times the process was swapped out of main memory	
	%U	Total number of CPU-seconds that the process spent in user mode	%c	Number of times the process was context-switched involuntarily	
	%P	Percentage of the CPU that this job got, computed as (%U + %S) / %E	%w	Number of waits: times that the program was context-switched voluntarily	
MEM	%M	Maximum resident set size of the process during its lifetime, in Kbytes	%I	Number of filesystem inputs by the process	IO
	%t	Average resident set size of the process, in Kbytes	%O	Number of filesystem outputs by the process	
	%K	Average total (data+stack+text) memory use of the process, in Kbytes	%r	Number of socket messages received by the process	
	%D	Average size of the process's unshared data area, in Kbytes	%s	Number of socket messages sent by the process	
	%p	Average size of the process's unshared stack space, in Kbytes	%k	Number of signals delivered to the process	
	%X	Average size of the process's shared text space, in Kbytes	%C	Name and command-line arguments of the com- mand being timed	
	%Z	System's page size, in bytes. This is a per-system constant, but varies between systems	%x	Exit status of the command	

Timer: Code Section

C

```c
#include <time.h>

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

 /* Code to time here */

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

FORTRAN

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

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

! Code to time here

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

C (MPI)

```c
#include <mpi.h>

double start, stop, time;
start = MPI_Wtime();

 /* Code to time here */

stop = MPI_Wtime();
time = stop - start;
```

Python

```python
import time

start=time.time()

# Code to time here

end=time.time()
time = end - start
```

About gprof

- **gprof** is the **GNU Project PROFiler**
  - Requires recompilation of the code
  - Compiler options and libraries provide wrappers for each routine call and periodic sampling of the program.
  - Provides three types of profiles
    - **Flat profile**
    - **Call graph**
    - **Annotated source**

gprof: gnu.org/software/binutils/

gprof: Types of Profiles

- **Flat Profile**
  - CPU time spend in each function (self and cumulative)
  - Number of times a function is called
  - **Useful to identify most expensive routines**
- **Call Graph**
  - Number of times a function was called by other functions
  - Number of times a function called other functions
  - **Useful to identify function relations**
  - Suggests places where function calls could be eliminated
- **Annotated Source**
  - **Indicates number of times a line was executed**

Profiling with gprof

Use the `-pg` flag during compilation:

```
$ gcc -g -pg -o exeFile ./srcFile.c
$ gfortran -g -pg -o exeFile ./srcFile.f90
```

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

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

```
$ gprof    ./exeFile gmon.out > profile.txt
$ gprof -l ./exeFile gmon.out > profile_line.txt
$ gprof -A ./exeFile gmon.out > profile_anotated.txt
```

The code must be compiled with "-g"

Enable line-by-line profiling

Print annotated source code

gprof: Flat Profile

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

% time	cumulative seconds	self seconds	calls	self s/call	total s/call	name
50.00	2.47	2.47	2	1.24	1.24	matSqrt
24.70	3.69	1.22	1	1.22	1.22	matCube
24.70	4.91	1.22	1	1.22	1.22	sysCube
0.61	4.94	0.03	1	0.03	4.94	main
0.00	4.94	0.00	2	0.00	0.00	vecSqrt
0.00	4.94	0.00	1	0.00	1.24	sysSqrt
0.00	4.94	0.00	1	0.00	0.00	vecCube

gprof: Call Graph Profile

```
index % time    self  children    called     name
                0.00    0.00       1/1          <hicore> (8)
[1]    100.0    0.03    4.91       1          main [1]
                0.00    1.24       1/1          sysSqrt [3]
                1.24    0.00       1/2          matSqrt [2]
                1.22    0.00       1/1          sysCube [5]
                1.22    0.00       1/1          matCube [4]
                0.00    0.00       1/2          vecSqrt [6]
                0.00    0.00       1/1          vecCube [7]
-----------------------------------------------
                1.24    0.00       1/2          main [1]
                1.24    0.00       1/2          sysSqrt [3]
[2]     50.0    2.47    0.00       2           matSqrt [2]
-----------------------------------------------
                0.00    1.24       1/1          main [1]
[3]     25.0    0.00    1.24       1           sysSqrt [3]
                1.24    0.00       1/2          matSqrt [2]
                0.00    0.00       1/2          vecSqrt [6]
-----------------------------------------------
```

Called by

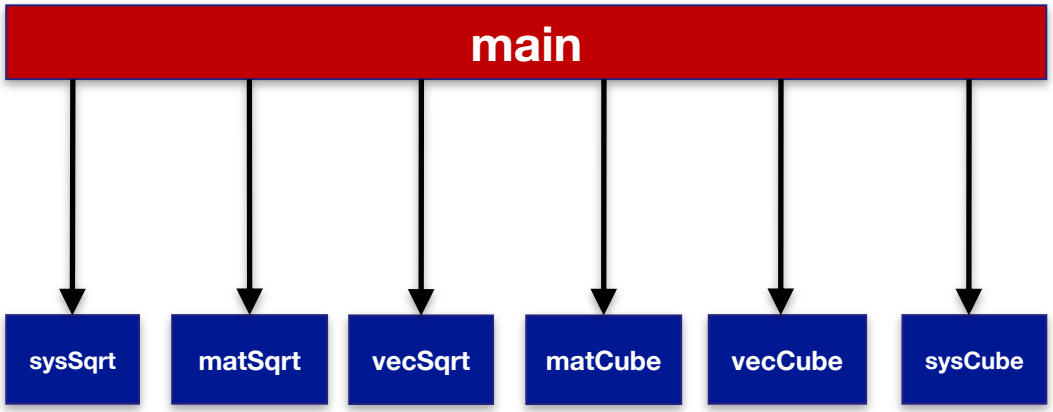
Called

This table describes the call tree of the program, and was sorted by the total amount of **time** spent in each function and its children.

The lines above it list the functions that called this function, and the lines below it list the functions this one called.

gprof: Call Graph Profile

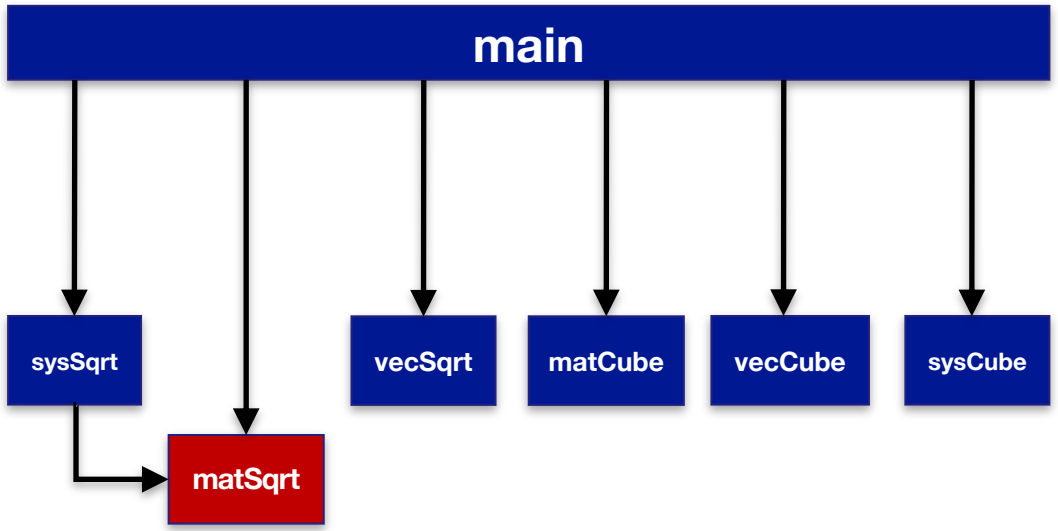
```
index % time    self  children    called     name
                0.00    0.00       1/1          <hicore> (8)
[1]    100.0    0.03    4.91       1          main [1]
                0.00    1.24       1/1          sysSqrt [3]
                1.24    0.00       1/2          matSqrt [2]
                1.22    0.00       1/1          sysCube [5]
                1.22    0.00       1/1          matCube [4]
                0.00    0.00       1/2          vecSqrt [6]
                0.00    0.00       1/1          vecCube [7]
-----------------------------------------------
                1.24    0.00       1/2          main [1]
                1.24    0.00       1/2          sysSqrt [3]
[2]     50.0    2.47    0.00       2           matSqrt [2]
-----------------------------------------------
                0.00    1.24       1/1          main [1]
[3]     25.0    0.00    1.24       1           sysSqrt [3]
                1.24    0.00       1/2          matSqrt [2]
                0.00    0.00       1/2          vecSqrt [6]
-----------------------------------------------
```



gprof: Call Graph Profile

```
index % time    self  children    called     name
                0.00    0.00       1/1          <hicore> (8)
[1]    100.0    0.03    4.91       1          main [1]
                0.00    1.24       1/1          sysSqrt [3]
                1.24    0.00       1/2          matSqrt [2]
                1.22    0.00       1/1          sysCube [5]
                1.22    0.00       1/1          matCube [4]
                0.00    0.00       1/2          vecSqrt [6]
                0.00    0.00       1/1          vecCube [7]
-----------------------------------------------
                1.24    0.00       1/2          main [1]
                1.24    0.00       1/2          sysSqrt [3]
[2]     50.0    2.47    0.00       2           matSqrt [2]
-----------------------------------------------
                0.00    1.24       1/1          main [1]
[3]     25.0    0.00    1.24       1           sysSqrt [3]
                1.24    0.00       1/2          matSqrt [2]
                0.00    0.00       1/2          vecSqrt [6]
-----------------------------------------------
```

Called by

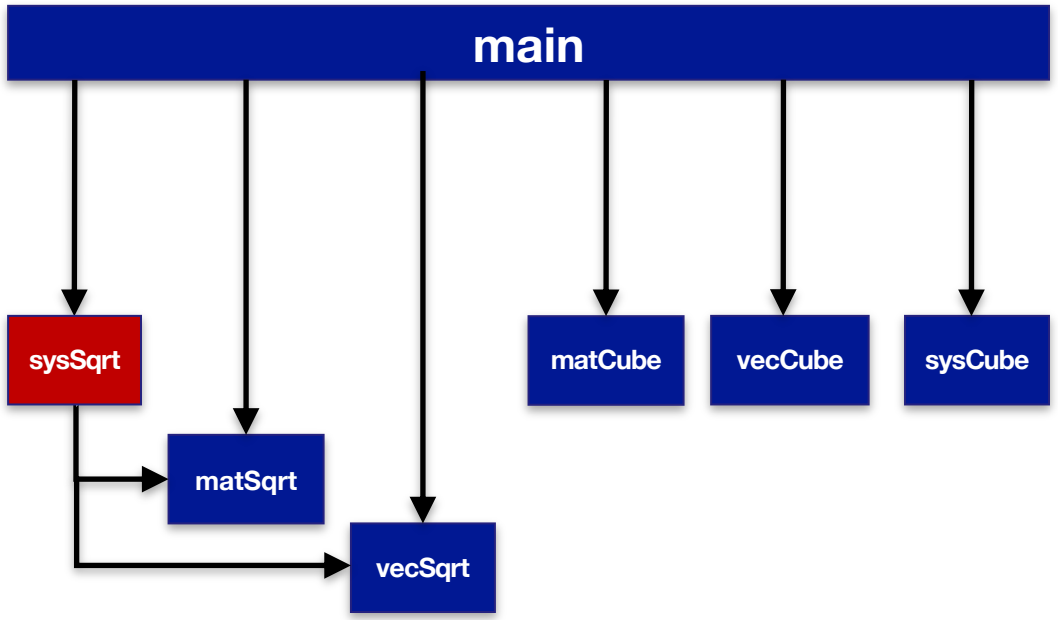


gprof: Call Graph Profile

```
index % time    self  children    called     name
                0.00    0.00       1/1          <hicore> (8)
[1]    100.0    0.03    4.91       1          main [1]
                0.00    1.24       1/1           sysSqrt [3]
                1.24    0.00       1/2           matSqrt [2]
                1.22    0.00       1/1           sysCube [5]
                1.22    0.00       1/1           matCube [4]
                0.00    0.00       1/2           vecSqrt [6]
                0.00    0.00       1/1           vecCube [7]
-----------------------------------------------
                1.24    0.00       1/2           main [1]
                1.24    0.00       1/2           sysSqrt [3]
[2]     50.0    2.47    0.00       2           matSqrt [2]
-----------------------------------------------
                0.00    1.24       1/1           main [1]
[3]     25.0    0.00    1.24       1           sysSqrt [3]
                1.24    0.00       1/2           matSqrt [2]
                0.00    0.00       1/2           vecSqrt [6]
-----------------------------------------------
```

Called by

Called



gprof: -l and -A

```
$ gprof -l ./exeFile gmon.out
```
Enable line-by-line profiling

```
$ gprof -A ./exeFile gmon.out
```
Print annotated source code

The code must be compiled with "-g"

Line-by-line profiling (-l)

```
Flat profile:

Each sample counts as 0.01 seconds.

  %   cumulative   self              self     total
 time   seconds   seconds    calls  Ts/call  Ts/call  name
100.00     47.21    47.21                             StaticFunc
(gprof_test.c:23 @ 40195a)

  0.00     47.21     0.00      100     0.00     0.00  StaticFunc
(gprof_test.c:18 @ 401927)

  0.00     47.21     0.00        1     0.00     0.00  TestFunc
(gprof_test.c:7 @ 4018c8)
```

Line number

Annotated source listing (-A)

```
                #include<stdio.h>
                void TestFunc();
                static void StaticFunc();
                void TestFunc()
          1 -> {
                        int i = 0;
                        printf("In TestFunc\n");
                        for (i=0; i<100; i++)
                                StaticFunc();
                }
                static void StaticFunc()
        100 -> {
                        int i = 0;
                        printf("In StaticFunc\n");
                        for (i=0; i<100000000; i++);
                }
                int main(void)
      ##### -> {
                        printf("In main\n");
                        TestFunc();
                        return 0;
                }
Top 10 Lines:
     Line      Count
       18        100
        7          1
Execution Summary:
        3   Executable lines in this file
        3   Lines executed
   100.00   Percent of the file executed
      101   Total number of line executions
    33.67   Average executions per line
```

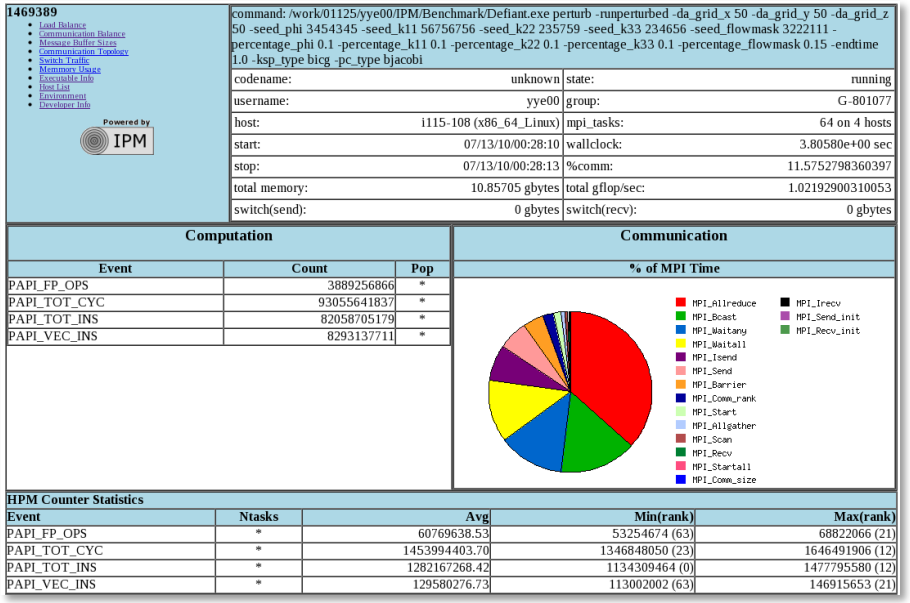
Intermediate Tools

IPM | IMPI Stats

IPM: Integrated Performance Monitoring

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

- IPM is a **quick**, **easy** and **concise** profiling tool. The level of detail it reports is smaller than **TAU**, **PAPI** or **HPCToolkit**.
- Requires **NO source code modification**, just adding the "-g" option to the compilation.
- Produces XML output that is parsed by scripts to generate browser-readable html pages.



1469389

- Load Balance
- Communication Balance
- Message Buffer Sizes
- Communication Topology
- Switch Traffic
- Memmory Usage
- Executable Info
- Host List
- Environment
- Developer Info

Powered by IPM

command: /work/01125/yye00/IPM/Benchmark/Defiant.exe perturb -runperturbed -da_grid_x 50 -da_grid_y 50 -da_grid_z 50 -seed_phi 3454345 -seed_k11 56756756 -seed_k22 235759 -seed_k33 234656 -seed_flowmask 3222111 -percentage_phi 0.1 -percentage_k11 0.1 -percentage_k22 0.1 -percentage_k33 0.1 -percentage_flowmask 0.15 -endtime 1.0 -ksp_type bicg -pc_type bjacobi

codename:	unknown	state:	running
username:	yye00	group:	G-801077
host:	i115-108 (x86_64_Linux)	mpi_tasks:	64 on 4 hosts
start:	07/13/10/00:28:10	wallclock:	3.80580e+00 sec
stop:	07/13/10/00:28:13	%comm:	11.5752798360397
total memory:	10.85705 gbytes	total gflop/sec:	1.02192900310053
switch(send):	0 gbytes	switch(recv):	0 gbytes

Computation

Event	Count	Pop
PAPI_FP_OPS	3889256866	*
PAPI_TOT_CYC	93055641837	*
PAPI_TOT_INS	82058705179	*
PAPI_VEC_INS	8293137711	*

Communication

% of MPI Time

HPM Counter Statistics

Event	Ntasks	Avg	Min(rank)	Max(rank)
PAPI_FP_OPS	*	60769638.53	53254674 (63)	68822066 (21)
PAPI_TOT_CYC	*	1453994403.70	1346848050 (23)	1646491906 (12)
PAPI_TOT_INS	*	1282167268.42	1134309464 (0)	1477795580 (12)
PAPI_VEC_INS	*	129580276.73	113002002 (63)	146915653 (21)

http://ipm-hpc.sourceforge.net/

IPM: How to Use

- **Compile the code:**

```
•$ icc -g -o exe mpi_code.c
```

- **Define collection type:**

```
•$ export IPM_REPORT=full
```

— Use “full”, “terse” (default), or ”none”

- **Define collection threshold (optional):**

```
•$ export IPM_MPI_THRESHOLD=3
```

— Reports only routines using ≥ this % of MPI time

- **Run the executable:**

```
•$ ibrun ./exe
```

— Will produce an XML output file

- **Generate HTML output:**

```
$ ipm_parse -html output.xml
```

— html files will be generated in a new folder

IPM:

Example Text Output

```
##IPMv0.983############################################################################
#
# command : /home1/01157/carlos/TEST/profiling/mm  (completed)
# host    : c404-504/x86_64_Linux          mpi_tasks : 16 on 1 nodes
# start   : 11/11/14/16:25:04              wallclock : 0.836220 sec
# stop    : 11/11/14/16:25:05              %comm     : 20.49
# gbytes  : 5.25372e+00 total              gflop/sec : NA
#
############################################################################
# region  : *       [ntasks] =     16
#
#                           [total]         <avg>           min           max
# entries                        16             1             1             1
# wallclock                 13.3793      0.836207      0.836197       0.83622
# user                      18.5732       1.16082       1.05884       1.18082
# system                    1.48777     0.0929854      0.074988      0.197969
# mpi                       2.74098      0.171311      0.133593      0.650309
# %comm                                   20.4864       15.9759       77.7683
# gbytes                    5.25372      0.328358      0.328205      0.330112
#
#                           [time]       [calls]        <%mpi>      <%wall>
# MPI_Bcast                  1.6727         32000         61.03         12.50
# MPI_Recv                 0.725848          4015         26.48          5.43
# MPI_Send                 0.320127          4015         11.68          2.39
# MPI_Barrier             0.0223054            16          0.81          0.17
# MPI_Comm_rank         2.97837e-06            16          0.00          0.00
# MPI_Comm_size         1.43796e-06            16          0.00          0.00
############################################################################
```

IPM: HTML Summary

1469389

- Load Balance
- Communication Balance
- Message Buffer Sizes
- Communication Topology
- Switch Traffic
- Memmory Usage
- Executable Info
- Host List
- Environment
- Developer Info

Powered by IPM

command: /work/01125/yye00/IPM/Benchmark/Defiant.exe perturb -runperturbed -da_grid_x 50 -da_grid_y 50 -da_grid_z 50 -seed_phi 3454345 -seed_k11 56756756 -seed_k22 235759 -seed_k33 234656 -seed_flowmask 3222111 -percentage_phi 0.1 -percentage_k11 0.1 -percentage_k22 0.1 -percentage_k33 0.1 -percentage_flowmask 0.15 -endtime 1.0 -ksp_type bicg -pc_type bjacobi

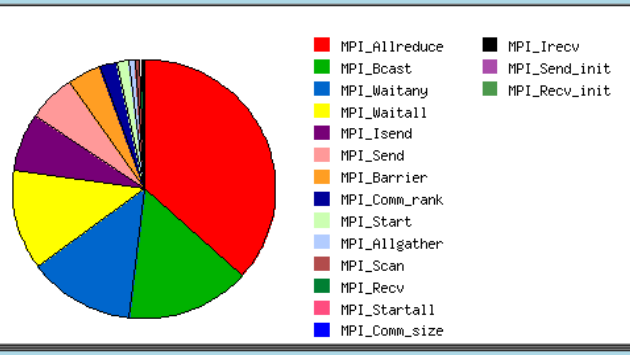
codename:	unknown	state:	running
username:	yye00	group:	G-801077
host:	i115-108 (x86_64_Linux)	mpi_tasks:	64 on 4 hosts
start:	07/13/10/00:28:10	wallclock:	3.80580e+00 sec
stop:	07/13/10/00:28:13	%comm:	11.5752798360397
total memory:	10.85705 gbytes	total gflop/sec:	1.02192900310053
switch(send):	0 gbytes	switch(recv):	0 gbytes

Computation

Event	Count	Pop
PAPI_FP_OPS	3889256866	*
PAPI_TOT_CYC	93055641837	*
PAPI_TOT_INS	82058705179	*
PAPI_VEC_INS	8293137711	*

Communication

% of MPI Time



HPM Counter Statistics

Event	Ntasks	Avg	Min(rank)	Max(rank)
PAPI_FP_OPS	*	60769638.53	53254674 (63)	68822066 (21)
PAPI_TOT_CYC	*	1453994403.70	1346848050 (23)	1646491906 (12)
PAPI_TOT_INS	*	1282167268.42	1134309464 (0)	1477795580 (12)
PAPI_VEC_INS	*	129580276.73	113002002 (63)	146915653 (21)

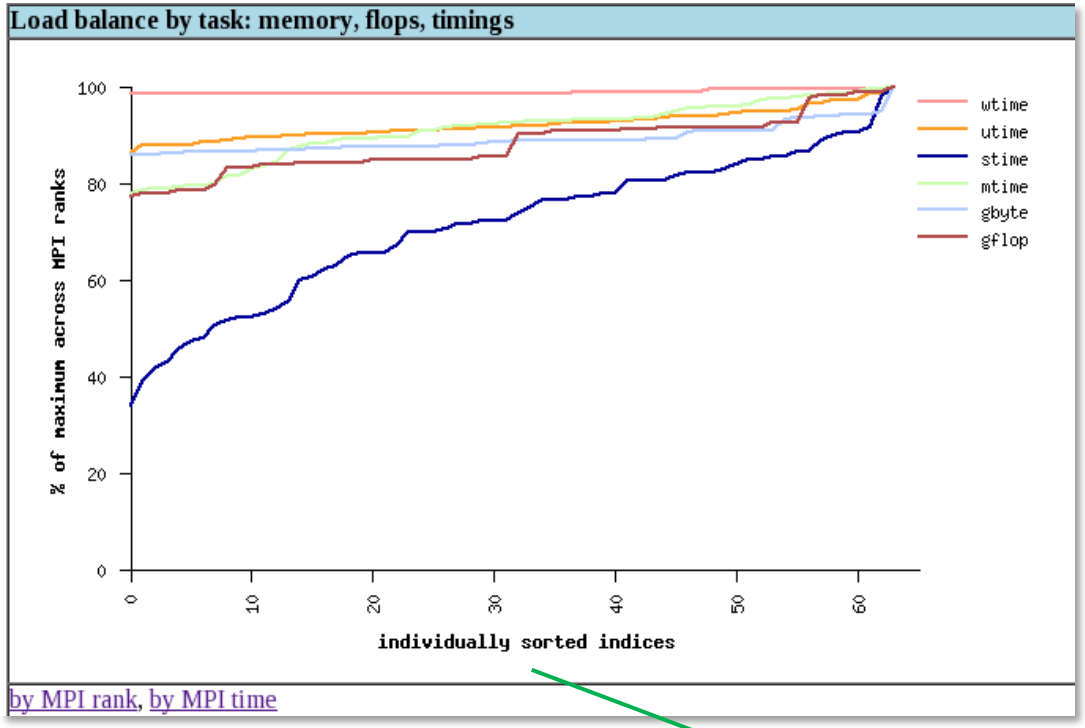
IPM: Event Statistics

Buffer size, Ncalls, Total Time, Min Time, Max Time, %MPI, %Wall

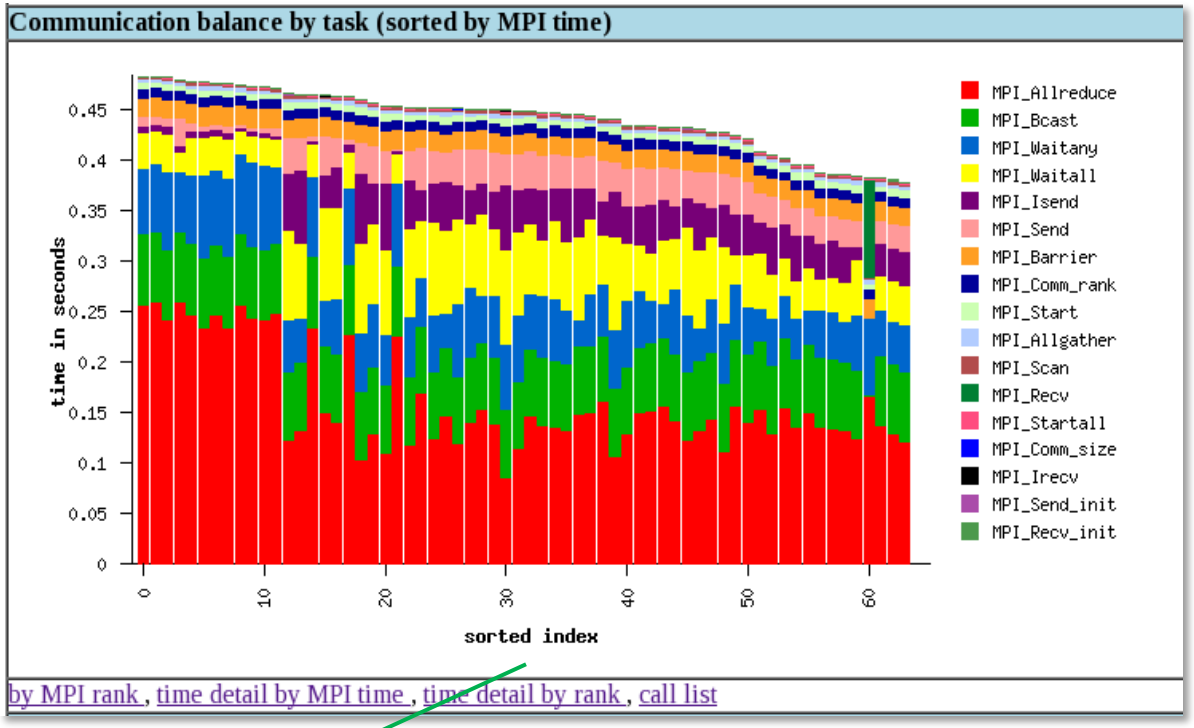
Communication Event Statistics (100.00% detail, 9.9012e-06 error)							
	Buffer Size	Ncalls	Total Time	Min Time	Max Time	%MPI	%Wall
MPI_Allreduce	8	79680	4.178	8.225e-06	8.882e-04	14.82	1.72
MPI_Bcast	4	1024	4.047	5.914e-08	6.413e-02	14.35	1.66
MPI_Allreduce	512	39936	3.803	1.660e-05	1.170e-01	13.49	1.56
MPI_Allreduce	4	25472	2.250	6.012e-07	1.552e-02	7.98	0.92
MPI_Barrier	0	64	1.176	1.814e-02	1.865e-02	4.17	0.48
MPI_Isend	8	630	1.028	3.427e-07	1.647e-02	3.65	0.42
MPI_Isend	4	4556	0.943	2.738e-07	1.833e-02	3.34	0.39
MPI_Send	14976	144	0.722	1.030e-03	7.308e-03	2.56	0.30
MPI_Comm_rank	0	106948	0.620	3.725e-08	9.872e-03	2.20	0.25
MPI_Waitany	0	6093	0.542	4.615e-07	1.358e-02	1.92	0.22
MPI_Waitany	1248	27462	0.519	5.183e-07	2.723e-04	1.84	0.21
MPI_Send	16224	144	0.517	4.283e-04	7.129e-03	1.83	0.21
MPI_Waitany	1352	20370	0.496	5.197e-07	5.783e-03	1.76	0.20
MPI_Start	0	269196	0.396	3.623e-07	3.685e-05	1.40	0.16
MPI_Send	13824	48	0.298	4.035e-03	7.227e-03	1.06	0.12
MPI_Waitany	1152	10980	0.243	5.383e-07	2.310e-04	0.86	0.10
MPI_Bcast	216	576	0.231	2.302e-06	5.843e-03	0.82	0.09
MPI_Allgather	4	9088	0.215	5.118e-07	1.793e-03	0.76	0.09
MPI_Waitall	184	11	0.210	1.633e-02	2.135e-02	0.74	0.09
MPI_Scan	4	384	0.144	2.259e-05	1.600e-03	0.51	0.06
MPI_Waitany	147	453	0.141	4.866e-07	1.406e-02	0.50	0.06
MPI_Waitany	4	448	0.132	4.345e-07	5.805e-03	0.47	0.05
MPI_Waitall	320	18	0.120	3.002e-06	1.284e-02	0.42	0.05
MPI_Send	17576	42	0.108	6.682e-05	6.406e-03	0.38	0.04
MPI_Waitall	72	6	0.103	1.547e-02	2.038e-02	0.36	0.04
MPI_Waitall	96	38	0.091	2.882e-06	1.563e-02	0.32	0.04
MPI_Waitany	624	140	0.088	1.126e-06	7.880e-03	0.31	0.04
MPI_Recv	8	9	0.085	8.373e-07	7.696e-02	0.30	0.03

IPM: Load and Communication Balance

Load balance by task



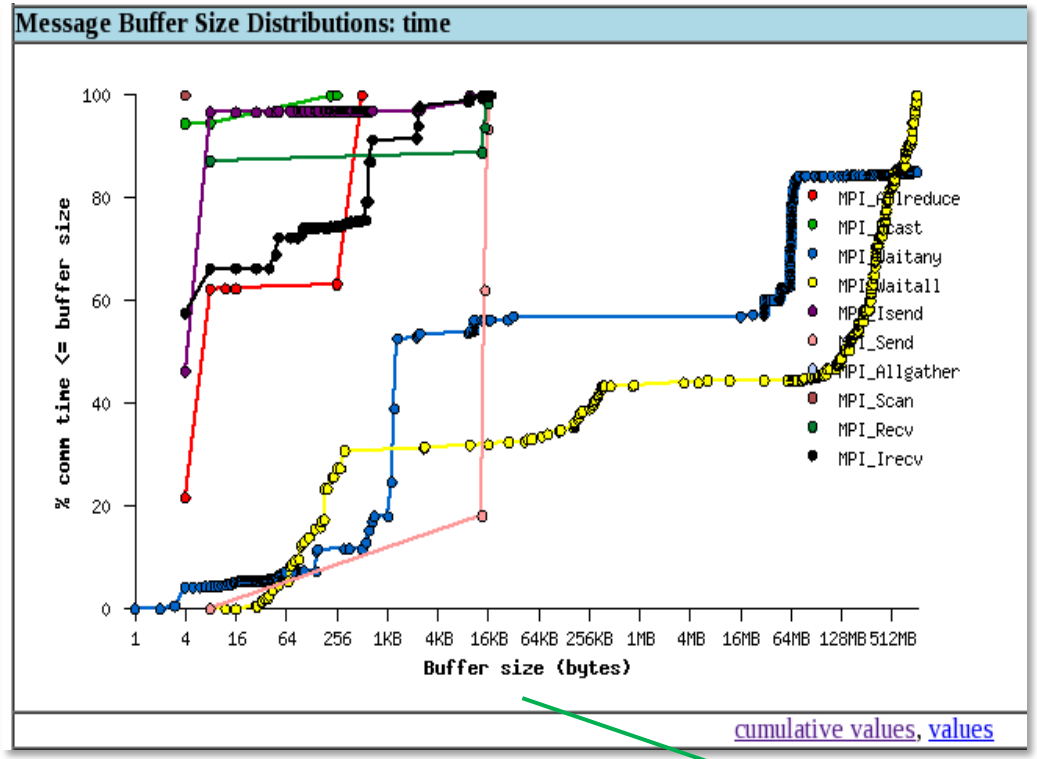
Communication balance by task



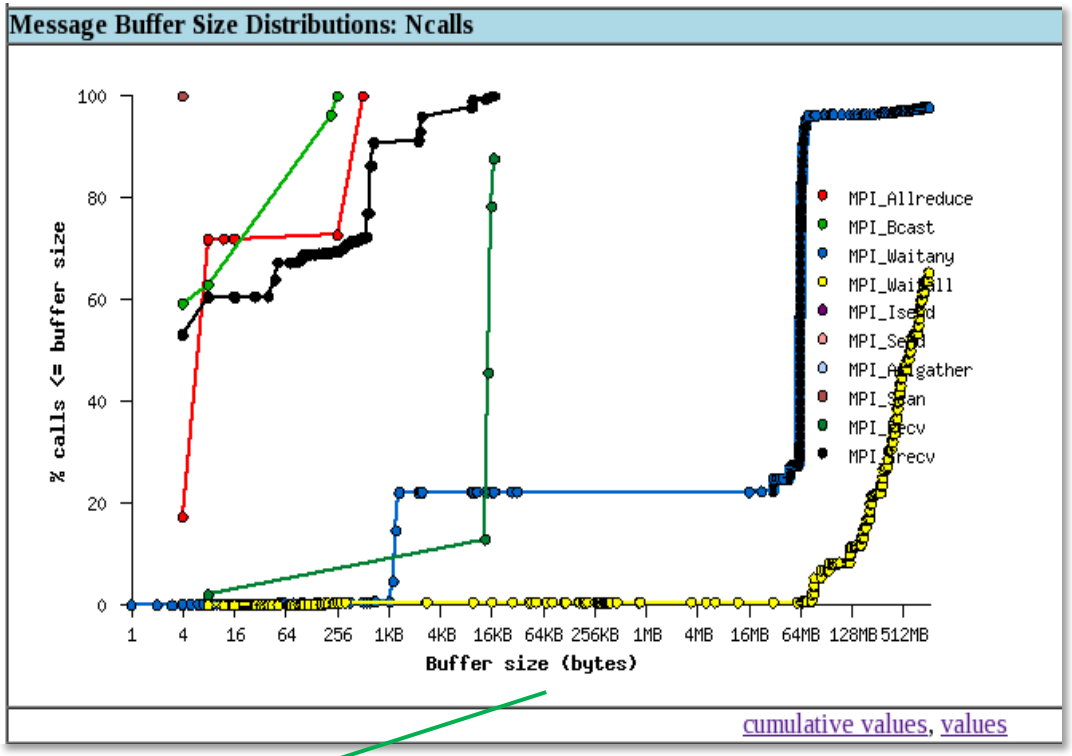
Sorted Indices

IPM: Buffer Size Distribution

% of Comm Time



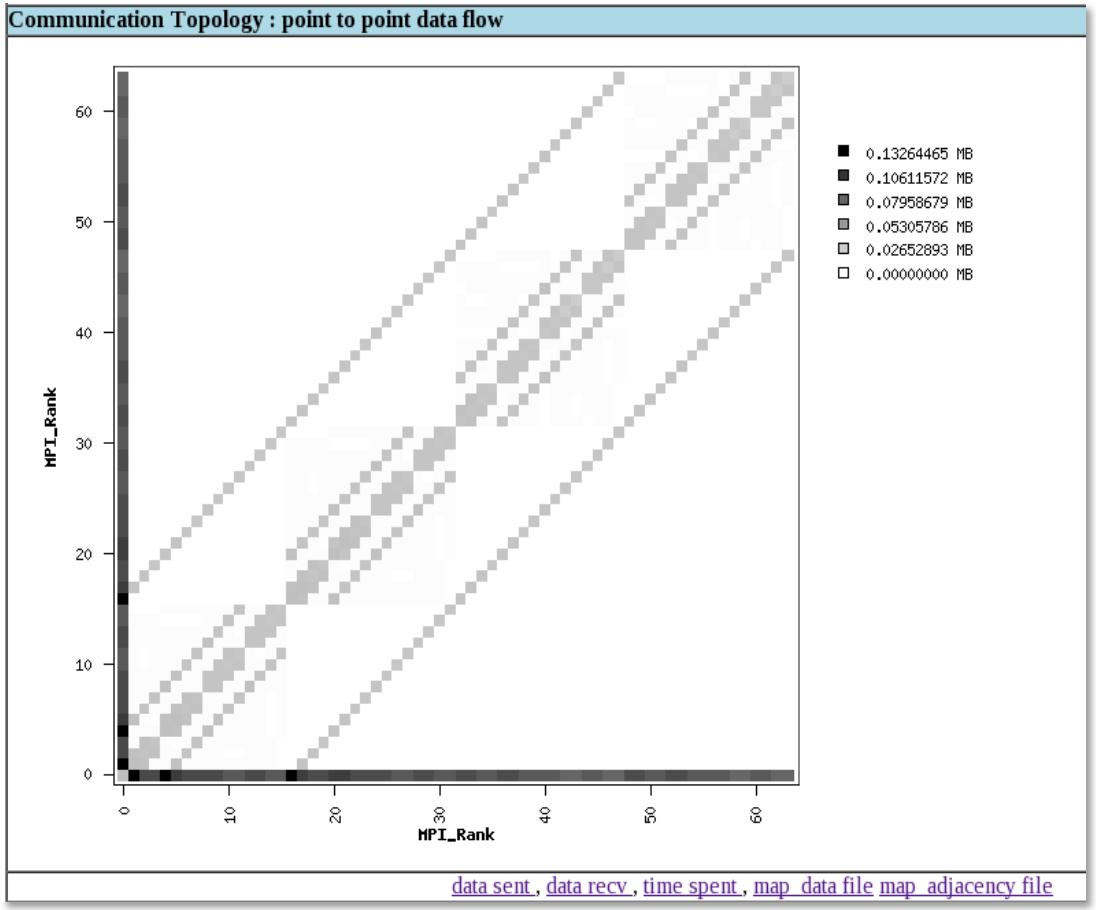
Ncalls



Buffer size

IPM: Communication Topology

MPI_Rank



MPI_Rank

IPM: Integrated Performance Monitoring

When to use IPM?

- To quickly find out **where your code is spending most of its time** (computation & communication)
- When you suspect you have **load imbalance** and want to verify it quickly
- For performing **scaling studies** (strong & weak)
- For a quick look at the **communication pattern**
- To find out **memory usage per task**
- To find the **relative communication & compute time**

When IPM is NOT the answer

- When you **already know** where the performance issues are
- When you need detailed performance information on **exact** lines of code
- When you want to find specific information such as **cache misses**

Profiling with IMPI Stats – ipm style

- **Enable IPM-style statistics collection :**
- `$ export I_MPI_STATS=ipm` — Summary data through all regions defined by "MPI_Pcontrol" in the code or use "ipm:terse" for basic summary
- **Define the output file name :**
  - `$ export I_MPI_STATS_FILE=mystats.dat`
- **Define collection threshold :**
  - `$ export I_MPI_STATS_ACCURACY=3` percentage
- **Define a subset or subsets of MPI functions for statistics gathering :**
  - `$ export I_MPI_STATS_SCOPE="all2all;send;time"` can be one or more from the table below

all2all	all2one	attr	comm	err	group	init	io
on2all	recv	req	rma	scan	send	sendrecv	serv
spawn	status	sync	time	topo	type		

References: https://software.intel.com/en-us/mpi-developer-guide-linux-gathering-statistics
https://software.intel.com/en-us/mpi-developer-reference-linux-ipm-statistics

Profiling with IMPI Stats – ipm style

I_MPI_STATS=ipm

```
##############################################################################
#
# command : ./a.out (completed)
# host : node01/x86_64_Linux mpi_tasks : 4 on 1 nodes
# start : 05/25/11/05:44:13 wallclock : 0.092012 sec
# stop : 05/25/11/05:44:13 %comm : 98.94
# gbytes : 0.00000e+00 total gflop/sec : NA
#
##############################################################################
# region : * [ntasks] = 4
#          └──────────────────  * = main region  (from MPI_Init to MPI_finalize)
# [total] <avg> min max # entries 4 1 1 1
# wallclock 0.332877 0.0832192 0.0732641 0.0920119
# user 0.047992 0.011998 0.006999 0.019996
# system 0.013997 0.00349925 0.002999 0.004
# mpi 0.329348 0.082337 0.0723064 0.0912335
# %comm 98.9398 98.6928 99.154
# gflop/sec NA NA NA NA
# gbytes 0 0 0 0
#
#
# [time] [calls] <%mpi> <%wall>
# MPI_Init 0.236192 4 71.71 70.95
# MPI_Reduce 0.0608737 8000 18.48 18.29
# MPI_Barrier 0.027415 800 8.32 8.24
# MPI_Recv 0.00483489 1 1.47 1.45
# MPI_Send 1.50204e-05 1 0.00 0.00
# MPI_Wtime 1.21593e-05 8 0.00 0.00
# MPI_Finalize 3.33786e-06 4 0.00 0.00
# MPI_Comm_rank 1.90735e-06 4 0.00 0.00
# MPI_TOTAL 0.329348 8822 100.00 98.94
##############################################################################
```

Advanced Tools

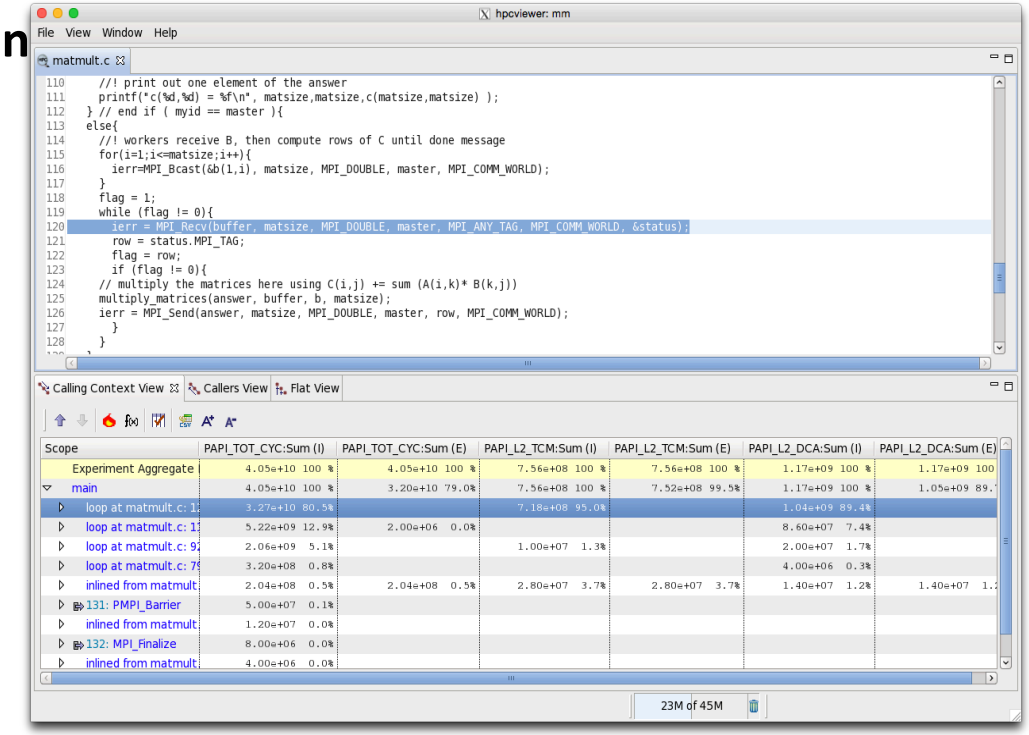
HPCToolKits | TAU | VTune

Advanced Profiling Tools

- Can be intimidating
  - **Difficult to install**
  - **Many dependences**
  - **Require kernel patches**
- Useful for serial and parallel programs
- Extensive profiling and scalability information
- Analyze code using
  - **Timers**
  - **Hardware registers (PAPI)**
  - **Function wrappers**

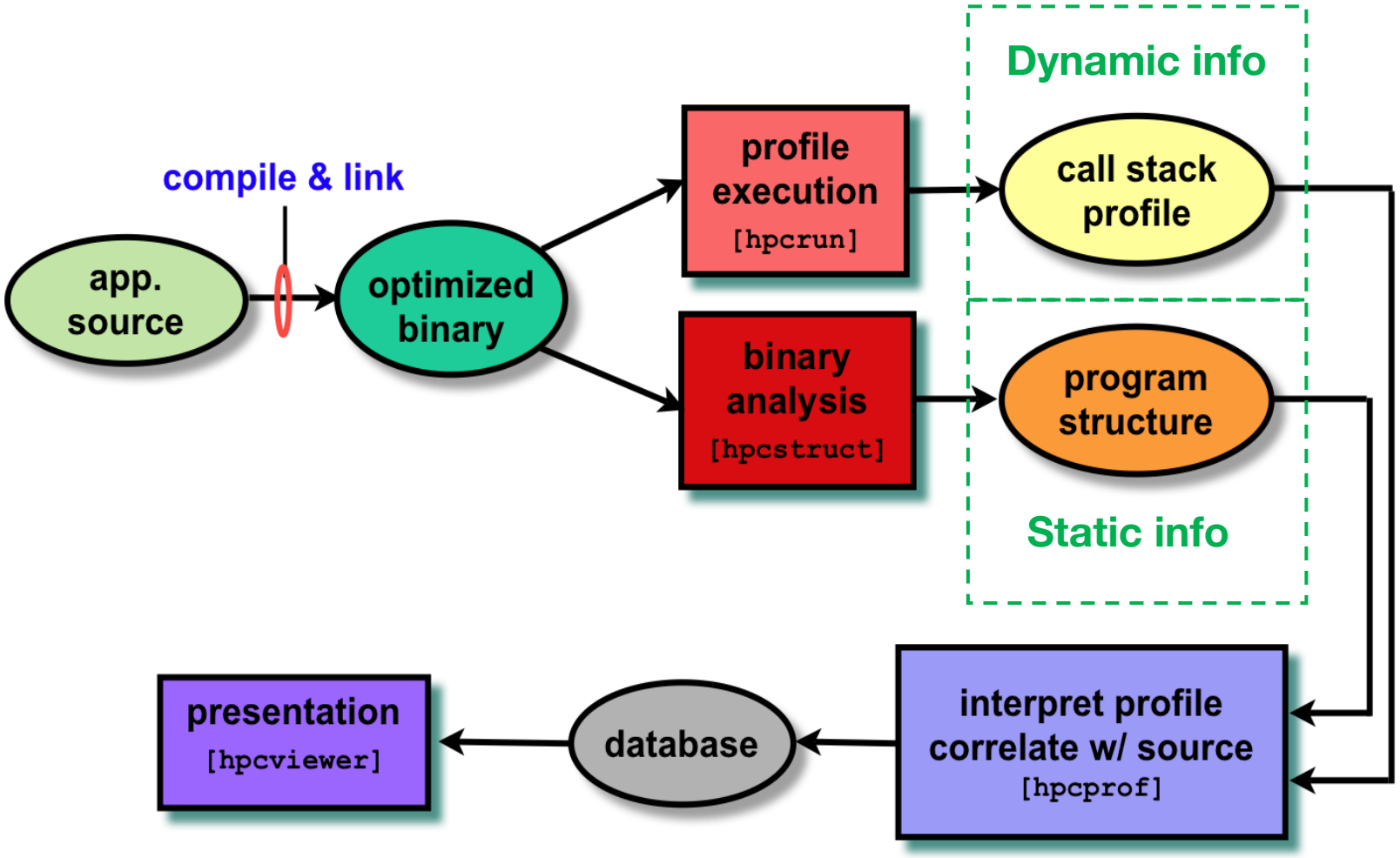
HPCToolkit

- **Binary-level measurement and analysis – n**
  recompilation! (binary compiled with -g -O3 is recommended)
- **Uses sample-based measurements**
  - Controlled overhead (typically 3-5%)
  - Good scalability for large scale parallelism analysis
- **Good analysis tools**
  - Ability to compare multiple runs at different scales
  - Scalable interface for trace viewer
  - Good derived metrics support



hpctoolkit.org

HPCToolkit : Overview



hpctoolkit.org

HPCToolkit : How to Use

- Compile with symbols and optimization:
  - `$ icc -g -O3 -o exe File.c`

Any level of optimization is supported by HPCToolkit. But we should turn off interprocedural optimization (IPA) to avoid possible problems

Collect dynamic info

- Collect profiling information:
  - `$ hpcrun ./exe` (Serial / OMP)
  - `$ ibrun hpcrun ./exe` (MPI parallel)

Collect static info

- Perform static binary analysis of the executable:

```
$ hpcstruct ./exe
```

Analyze data

- Interpret profile and correlate it with source code:

```
$ hpcprof -S exe.hpcstruct hpctoolkit-exe-measurement-dir
```

static info

dynamic info

View results

- View the results

```
$ hpcviewer hpctoolkit-exe-database-dir
```

report

HPCToolkit : Specifying Sample Sources

- Find all available events and a brief description using:

```
$ hpcrun --list-events
```

- Specify what events to measure during the collection :

```
$ hpcrun -e e[@p] ./exe
```

After every ‘p’ instances of event ‘e’ a sample is generated

- Multiple events can be specified (no multiplexing)

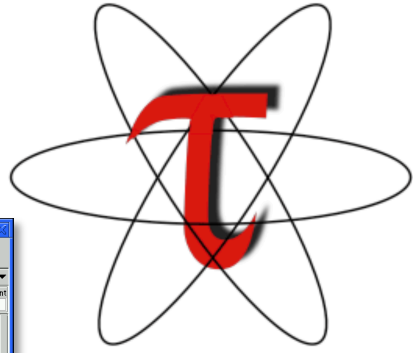
```
$ hpcrun -e PAPI_L2_TCM -e PAPI_L2_DCA -e PAPI_TOT_CYC ./exe
```

event 1 | event 2 | event 3

- WALLCLOCK event cannot be used with other events

```
$ hpcrun -e WALLCLOCK ./exe
```

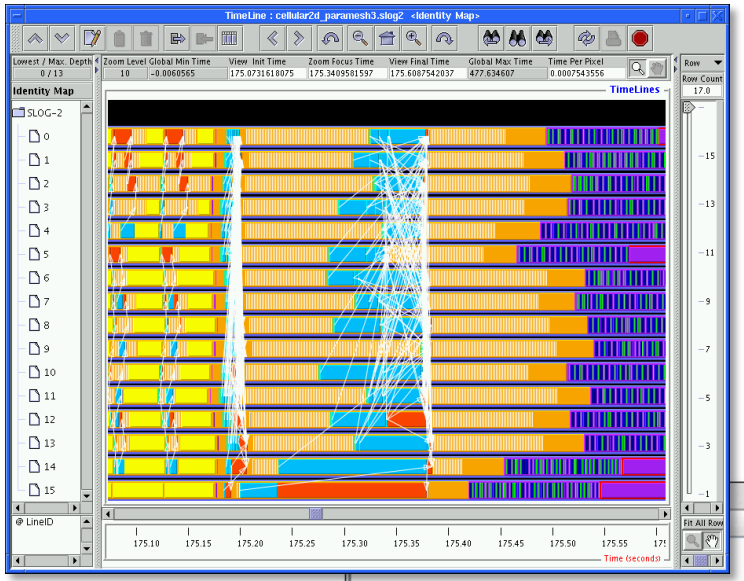
TAU: Tuning and Analysis Utilities



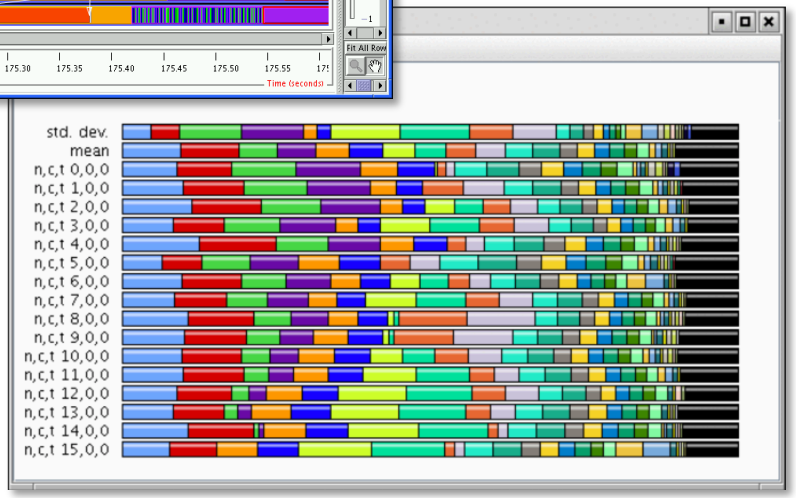
- **12+ year project involving**

  Performance Research Lab, Univ of Oregon
  Advanced Computing Laboratory, LANL
  Research Centre Julich at ZAM, Germany

- **Integrated toolkit**
  - ✓ Performance instrumentation
  - ✓ Measurement
  - ✓ Analysis
  - ✓ Visualization

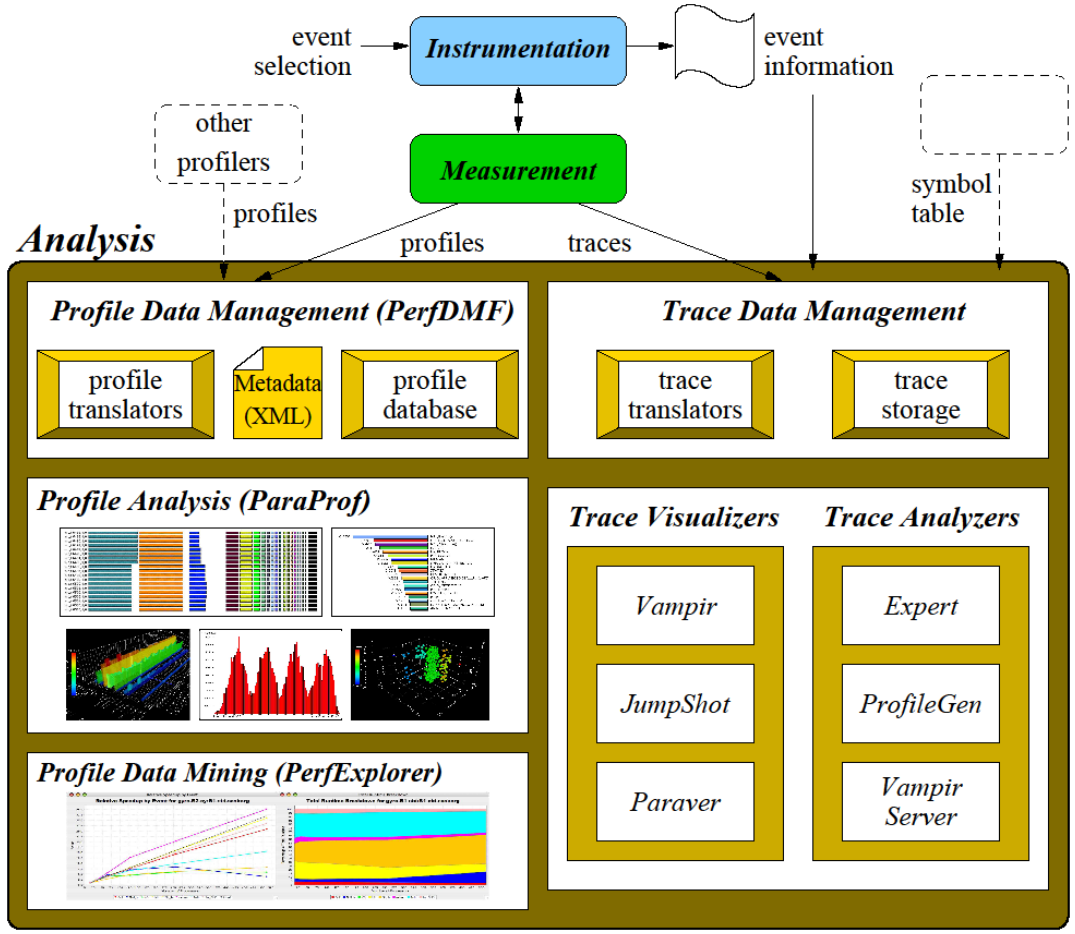


Jumpshot-4 viewer
ANL



www.cs.uoregon.edu/research/tau

TAU: Performance System Architecture



- **Parallel profiling**
  - ✓ Function-level, block (loop)-level, statement-level
  - ✓ Supports user-defined events
  - ✓ TAU parallel profile data stored during execution
  - ✓ Hardware counter values (multiple counters)
  - ✓ Support for **callgraph** and **callpath** profiling
- **Tracing**
  - ✓ All profile-level events
  - ✓ Inter-process communication events
  - ✓ Trace merging and format conversion

TAU: Instrumenting your code

https://portal.tacc.utexas.edu/software/tau

To use TAU, you need to recompile your code using some TAU scripts as compilers. The TAU compilation scripts are:

- `tau_f90.sh (F90), tau_cc.sh (C), tau_cxx.sh (C++)`

Call these scripts directly or place in a makefile:

```
$ tau_cc.sh -o exe file.c
```

Control and configure TAU output with the following environment variables:

TAU_PROFILE	Set to 1 to turn on **profiling** (statistics) information.
PROFILEDIR	Set to the name of a directory; otherwise output goes to the current directory.
TAU_TRACE	Set to 1 to turn on **tracing** (timeline) information.
TRACEDIR	Set to the name of a directory. You can safely use the 'PROFILEDIR' value.

TAU: Examples

Set up your environment in a batch script or idev session:

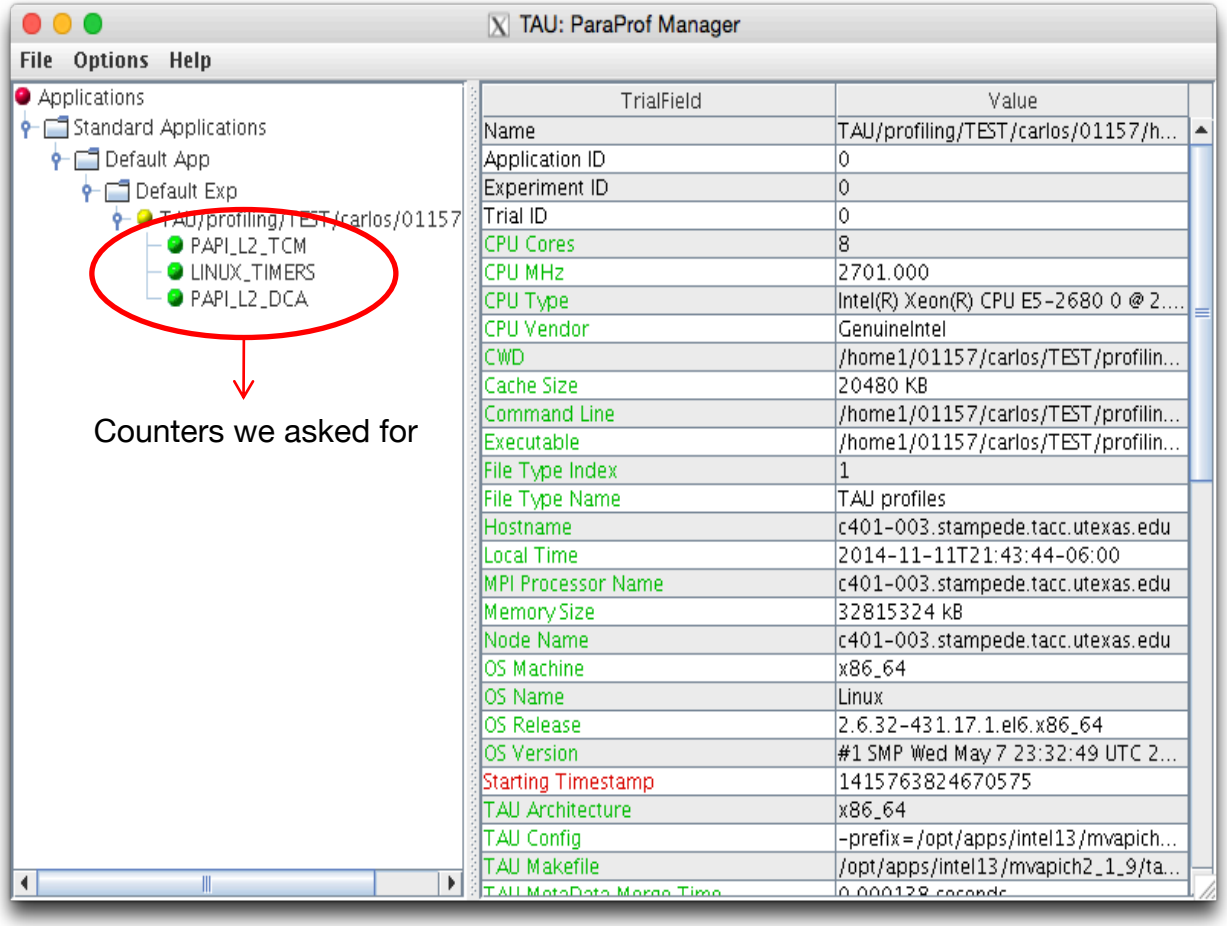
Example batch script

```
• #SBATCH directives
• ...
• export PROFILEDIR=mytaudir
• export TRACEDIR=mytaudir
• export TAU_PROFILE=1
• export TAU_TRACE=1
• ...
• ibrun myprogram
```

Example idev session

```
login1$ idev
...
c455-073[knl]$ mkdir -p mytaudir; cd mytaudir
c455-073[knl]$ export PROFILEDIR=mytaudir
c455-073[knl]$ export TRACEDIR=mytaudir
c455-073[knl]$ export TAU_PROFILE=1
c455-073[knl]$ export TAU_TRACE=1
...
ibrun myprogram
```

TAU: Paraprof Manager Window

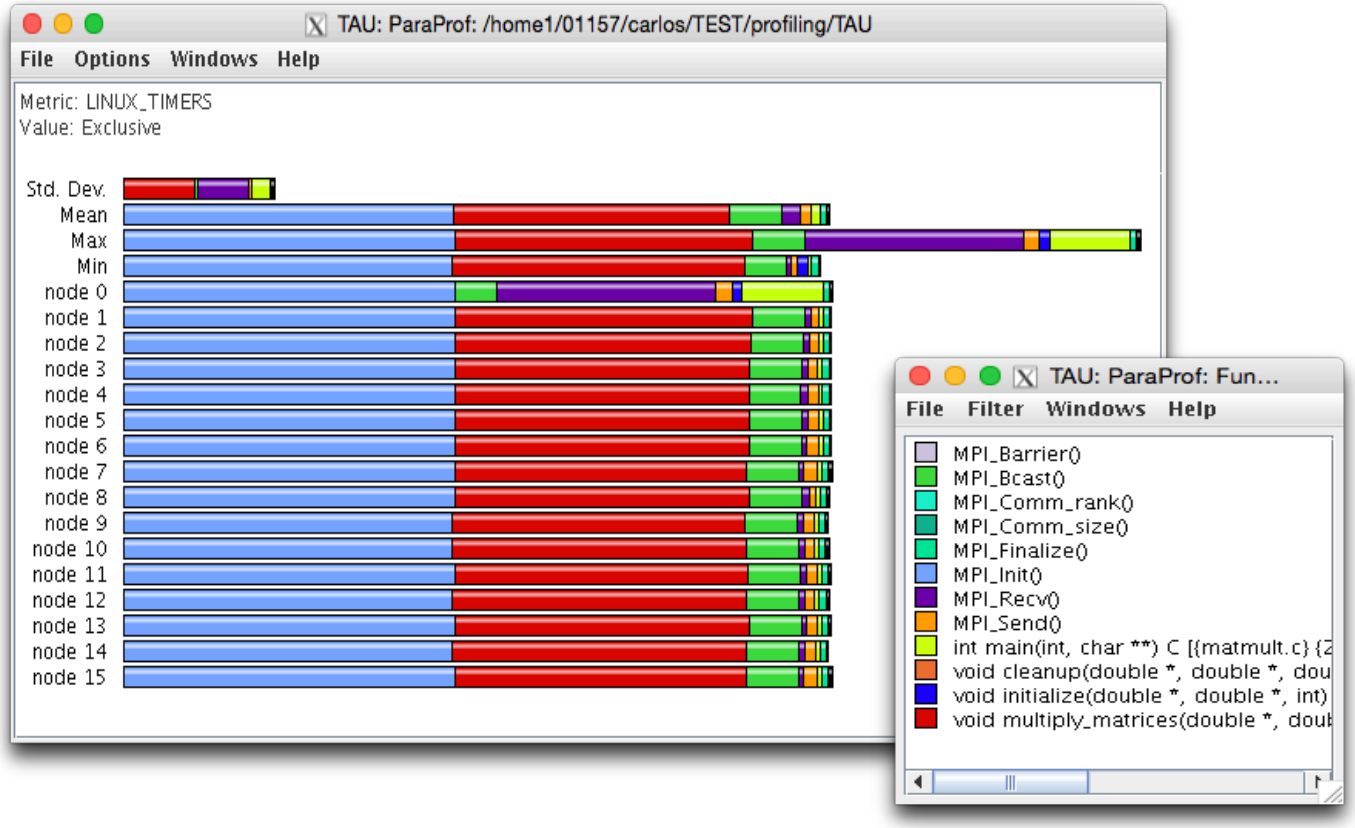


- Display global statistics with TAU's 3D profile browser, ParaProf

```
$ paraprof ${PROFILEDIR}
```

- Provides Machine Details
- Organizes Runs as
  - Applications
  - Experiments
  - Trials

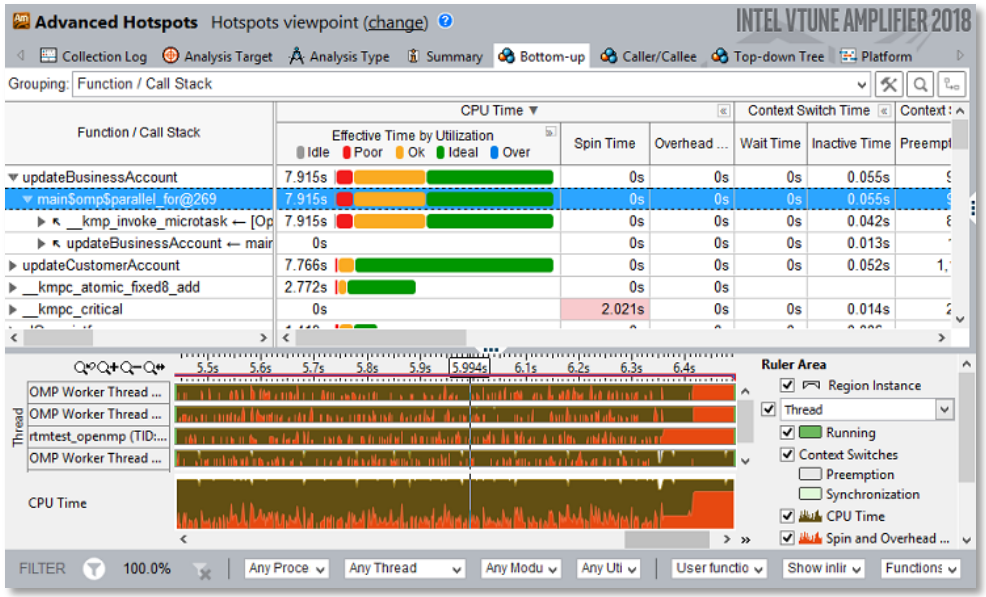
TAU: Routine Time Experiment



- Each Event collected is viewed separately
- Statistics provided across MPI tasks
- Click on a given color to see more detail (new window)

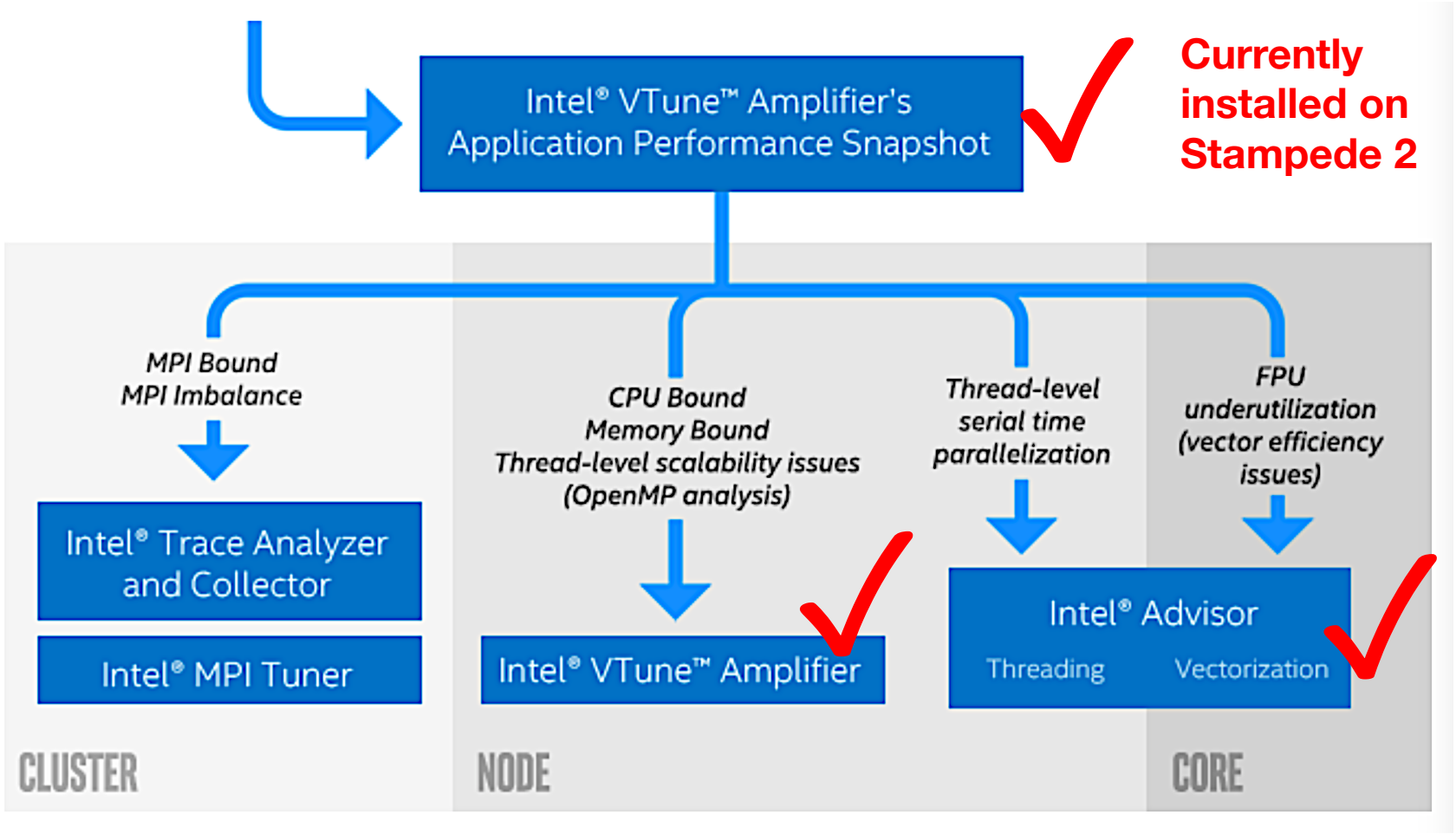
Intel VTune Amplifier

- VTune is Intel's signature performance profiling tool
- Supports serial, threaded, MPI, and hybrid applications
- Two part interface : command line and GUI
- Much improved support for KNL, including memory access details
  - ✓ Memory bandwidth to DDR4 and MCDRAM (Flat or Cache)
  - ✓ Fine grained control of memory allocation using libmemkind



software.intel.com/en-us/intel-vtune-amplifier-xe

Tuning Workflow



https://www.alcf.anl.gov/files/velesko_vtune_may.pdf

VTune : Basics

- Compile your code with **-g** and full optimization (e.g. on KNL node)
- `$ mpicc -g -O3 -xMIC-AVX512 -qopenmp -o exe ./file.c`
- Load VTune module
- `$ module load vtune`
- Run with **amplxe-cl** to collect performance data
- `$ amplxe-cl -collect hotspots ./exe` **(Serial / OMP)**
- `$ ibrun amplxe-cl -collect hotspots ./exe` **(MPI Parallel)**
- Generate text-only report or work with GUI
- `$ amplxe-cl -report hotspots -r <result_dir>` **(text-only)**
- `$ amplxe-gui <result_dir>` **(GUI)**

VTune : in a Batch Job

```
#!/bin/bash
#SBATCH -J knl-run.%j.err
#SBATCH -o knl-run.%j.out
#SBATCH -N 1
#SBATCH -n 1
#SBATCH -p normal
#SBATCH -t 00:30:00
#SBATCH -A <Allocation>
export OMP_NUM_THREADS=68

module load vtune
amplxe-cl -c hotspots -no-auto-finalize -- ./exe
```

Launch amplxe-cl with proper options

VTune: Pre-defined Collections

For information on a specific analysis type: `$ amplxe-cl -help collect <analysis type>`

Collection	Information Provided
hotspots	Identify most time-consuming code sections (user mode)
advanced-hotspots	Adds CPI, higher frequency low overhead sampling
concurrency	CPU utilization, threading synchronization overhead (user mode)
disk-io	Disk IO preview, not working in Stampede (requires root)
memory-access	Memory access details and memory bandwidth utilization. May include specific arrays that are heavy BW users (useful for fine-grained MCDRAM use)
hpc-performance	Performance characterization, including floating point unit and memory bandwidth utilization

https://software.intel.com/en-us/vtune-amplifier-help-running-command-line-analysis

VTune :

Example: Summary (hotspots/OpenMP)

Elapsed Time: 21.492s

CPU Time:	1426.740s
Effective Time:	164.571s
Spin Time:	1233.155s
Imbalance or Serial Spinning:	1218.450s
Lock Contention:	0s
Other:	14.705s
Overhead Time:	29.015s
Total Thread Count:	159
Paused Time:	0s

OpenMP Analysis. Collection Time: 21.492

- Serial Time (outside parallel regions): 0s (0.0%)
- Parallel Region Time: 0s (0.0%)

Time

Top Hotspots

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

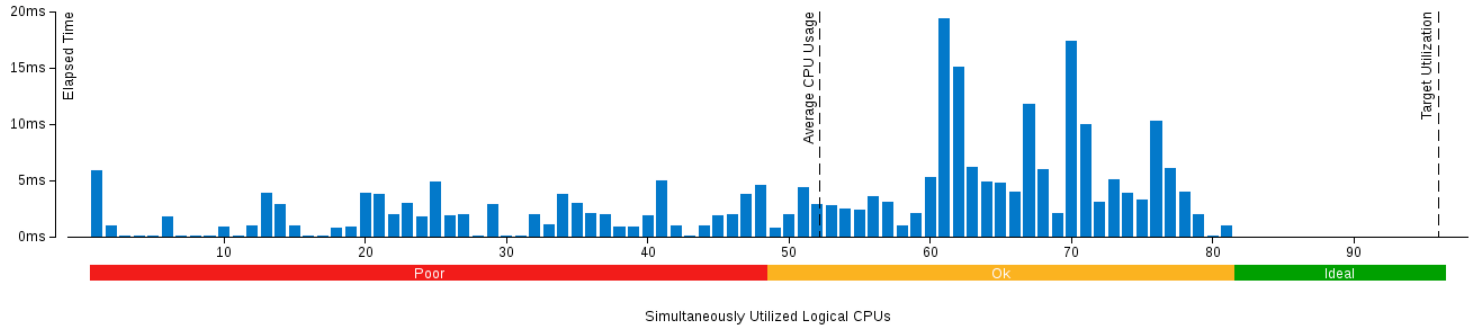
Function	Module	CPU Time
__kmp_fork_barrier	libiomp5.so	1219.933s
_INTERNAL_26_____src_kmp_dispatch_cpp_de46bbb0::[OpenMP dispatcher]	libiomp5.so	23.734s
hilbert_index	art	19.611s
smoothompparallel@508	art	16.714s
hilbert_coords	art	10.157s
[Others]		136.590s

Hotspots

OpenMP Region CPU Usage Histogram

This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously in an OpenMP region. Spin and Overhead time adds to the Idle CPU usage value. OpenMP regions in the drop-down list are sorted by Potential Gain (Elapsed Time) so it is recommended to start exploration from the top.

OpenMP Region: smoothompparallel:96@unknown:508:733



CPU usage

VTune: Online Resources

Profiling your application with Intel® Vtune™ Amplifier and Intel® Advisor

https://www.alcf.anl.gov/files/velesko_vtune_may.pdf

Using Intel® Advisor and VTune™ Amplifier with MPI

https://software.intel.com/en-us/articles/using-intel-advisor-and-vtune-amplifier-with-mpi

Intel® VTune™ Amplifier Support - Training

https://software.intel.com/en-us/intel-vtune-amplifier-xe-support/training

Summary

Profiling Dos and Don’ts

DO

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

DO NOT

- ✗ Assume a change will be an improvement
- ✗ Profile atypical cases
- ✗ Profile *ad infinitum*

Instead, set yourself a **goal** or a **time limit**

Other Useful Profiling tools at TACC

MPIP mpip.sourceforge.net

Lightweight, scalable MPI profiling tool

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

Intel Advisor https://software.intel.com/en-us/advisor

SIMD vectorization optimization and shared memory threading assistance tool

REMORA https:github.com/TACC/remora

REsource **MO**nitoring for **R**emote **A**pplications

Valgrind valgrind.org

Powerful instrumentation framework, often used for debugging memory problems

Other Useful Profiling tools at TACC

- **TACC Stats** https://www.tacc.utexas.edu/research-development/tacc-projects/tacc-stats
  - Transparent resources and performance monitoring. Provides data through web interface

PerfexPert https://www.tacc.utexas.edu/research-development/tacc-projects/perfexpert

An easy-to-use performance diagnosis tool for HPC applications