



# The HPCToolkit

---

Introduction to Rosa Course  
Scientific Computing Group

---



## HPCToolkit: Motivation

---

- HPCToolkit's measurement tools collect performance data on each process and thread of an MPI program
    - Can be used with MPI or hybrid programs (MPI with OpenMP or pthreads)
  - Supports C, C++ and Fortran
  - The *hpcviewer* tool aids analysis of the application performance data
-

## HPCToolkit: Motivation

---

- HPCToolkit can provide **accurate fine-grain** measurements of production applications running at scale, through
  - *low measurement overhead*
  - *call path profiling*
  - mapping between *fully optimised object code* and *associated source code*

3

## HPCToolkit: Motivation

---

- Requirement: low measurement overhead
  - **Avoid instrumentation**
    - Tools such as *Tau* use source code instrumentation to insert profiling code into the source before compilation
      - Instrumentation can interfere with compiler optimisations such as inlining and loop transformations
      - Source instrumentation is not useful when using libraries
    - Binary instrumentors such as *gprof* may prevent full optimisation of code

4

## HPCToolkit: Motivation

---

- Requirement: low measurement overhead
  - Instead of instrumentation use **statistical sampling**
    - Sample events triggered by periodic interrupts from an interval timer or from the overflow of hardware performance counters
    - One can sample metrics that reflect work (e.g. floating point operations performed), resource consumption (e.g. cycles, memory bus transactions) or inefficiency (e.g. stalls)

## HPCToolkit: Motivation

---

- Call path profiling
  - Important to associate the costs incurred by a procedure with the context in which it is called
  - Particularly important for codes based on application frameworks and libraries

## HPCToolkit: Motivation

---

- Recovering static program structure
  - Since measurements are made with reference to executables, it is necessary to map measurements back to program source code
  - HPCToolkit constructs this mapping using binary analysis
    - Can thus accurately attribute performance metrics to calling contexts, procedures, loops, and inlined instances of procedures

7

## HPCToolkit: How to use on Rosa

---

- Load the hpct module
  - **module load hpct**
- Compile code with debugging support
  - *for PGI: -gopt; for PathScale: -g1; for GNU: -g*
    - Remember to add optimisation flags (eg. -O3) *after* -g to make sure compiler optimisation is not reduced
- Link your application using hpclink
  - **hpclink ftn -o code.exe code.o code1.o ...**
  - **hpclink cc -o code.exe code.o code1.o ...**

8

## HPCToolkit: How to use on Rosa

---

- In your PBS batch job script set the `CSPROF_OPT_EVENT` environment variable according to which events you wish to monitor, e.g.
  - `export CSPROF_OPT_EVENT="PAPI_TOT_CYC@4000000 PAPI_L2_TCM@400000"`
    - This would sample your executable every 4 million cycles and every 400k level 2 cache misses
  - Note: use “`papi_avail`” program to see which hardware counters are available on the XT5

9

## HPCToolkit: How to use on Rosa

---

- Launch code with **aprun** as normal
  - You will get a directory called `hpctoolkit-<exe>-measurements-<pbs_jobid>`
    - This will contain files with suffix `.hpcrun` (one for each MPI task) and logfiles (`.log`)
- Do a one-time binary analysis of your code using the `hpcstruct` command (interactively)
  - `hpcstruct <exe>`
    - This generates a file called `<exe>.hpcstruct`
    - May take some minutes!

10

## HPCToolkit: How to use on Rosa

---

- Finally, invoke “hpcprof” to join the information from the run to the information from “hpcstruct”
  - `hpcprof -S <exe>.hpcstruct -l <path-to-source> hpctoolkit-<exe>-measurements-<pbs_jobid>/*.hpcrun`
    - This produces a directory `hpctoolkit-<exe>-database`, which can be shipped elsewhere if you wish to analyse the results on another machine

11

## HPCToolkit: How to use on Rosa

---

- Java-based viewer “hpcviewer” provides a top-down visualisation of the experiment database
  - `module load hpct`
  - `hpcviewer hpctoolkit-<exe>-database`
- Viewer can also be installed and launched on your workstation
- Help can be found in a help pane
- Documentation on how to use the viewer at <http://hpctoolkit.org/documentation.html>

12

# HPCToolkit: hpcviewer

- Browser window divided in three panes
  - Source pane
    - Select an entity in the navigation pane and the source pane will load appropriate file and highlight corresponding line
  - Navigation pane
    - Hierarchical tree-based structure used to organise performance data. Display depends on whether a context, callers, or flat view is being shown
  - Metric pane
    - Displays performance metrics

13

The screenshot shows the hpcviewer application interface. The top pane is the Source pane, displaying Fortran code for a matrix multiplication routine. The middle pane is the Metric pane, showing a table of performance metrics. The bottom pane is the Navigation pane, showing a hierarchical tree of performance data.

**Source pane**

```

17 do j = 1, dim2
18   A(i, j) = rand()
19 enddo
20 enddo
21 do i = 1, dim2
22   do j = 1, dim3
23     B(i, j) = rand()
24   enddo
25 enddo
26c etime is not a part of the standard
27c call etime(tarray, start)
28 call cpu_time(start)
29 do i = 1, dim1
30   do j = 1, dim3
31     C(i, j) = 0.
32     do k = 1, dim2
33       C(i, j) = C(i, j) + A(i, k)*B(k, j)
34     enddo
35   enddo
36 enddo
37c call etime(tarray, finish)
    
```

**Metric pane**

Scope	PAPI_TOT_CYC (I)	PAPI_TOT_CYC (E)	PAPI_L2_TCM (I)	PAPI_L2_TCM (E)	PAPI_TOT_CYC (I)	PAPI_TOT_CYC (E)
Experiment Aggregate Metrics	3.49e+10 100 %	3.49e+10 100 %	7.80e+06 200 %	7.80e+06 100 %	3.48e+10 100 %	3.48
main	3.49e+10 100 %	3.45e+10 98.9%	7.80e+06 100 %	7.60e+06 97.4%	3.48e+10 100 %	3.45
MAIN	3.49e+10 100 %	3.45e+10 98.9%	7.80e+06 100 %	7.60e+06 97.4%	3.48e+10 100 %	3.45
loop at matrix2.f: 29	3.45e+10 98.7%	3.43e+10 98.3%	7.60e+06 97.4%	6.80e+06 87.2%	3.45e+10 99.0%	3.43
loop at matrix2.f: 30	3.45e+10 98.7%	1.48e+08 0.4%	7.60e+06 97.4%	8.00e+05 10.3%	3.45e+10 99.0%	1.20
loop at matrix2.f: 32	3.43e+10 98.3%	3.43e+10 98.3%	6.80e+06 87.2%	6.80e+06 87.2%	3.43e+10 98.7%	3.43
matrix2.f: 32	3.43e+10 98.3%	3.43e+10 98.3%	6.80e+06 87.2%	6.80e+06 87.2%	3.43e+10 98.7%	3.43
matrix2.f: 30	1.48e+08 0.4%	1.48e+08 0.4%	8.00e+05 10.3%	8.00e+05 10.3%	1.20e+08 0.3%	1.20
_wrap_mpi_init_	2.40e+08 0.7%				2.00e+08 0.6%	
_wrap_mpi_finalize_	7.60e+07 0.2%					
loop at matrix2.f: 21	7.20e+07 0.2%					
loop at matrix2.f: 16	6.80e+07 0.2%		2.00e+05 2.0%			
_f90io_ldw						

**Navigation pane**

14

# HPCToolkit: hpcviewer

- Supports three main views of performance data
  - Calling context view
    - Can see costs incurred by calls to a procedure in a particular calling context (call path)
  - Callers view
    - Bottom-up view enables you to look upward along call paths. Useful for understanding performance of procedures that are used in multiple contexts
  - Flat view
    - Costs incurred by a procedure in any calling context are aggregated together

The screenshot shows the hpcviewer application with a code editor at the top and a performance table below. The code editor displays a Fortran-like matrix multiplication routine. The performance table below has several columns: Scope, PAPI\_TOT\_CYC (I), PAPI\_TOT\_CYC (E), PAPI\_L2\_TCM (I), PAPI\_L2\_TCM (E), PAPI\_TOT\_CYC (I), and PAPI\_TOT\_CYC (E). The table is expanded to show a hierarchy of scopes from 'main' down to individual loops and MPI-related functions.

Scope	PAPI_TOT_CYC (I)	PAPI_TOT_CYC (E)	PAPI_L2_TCM (I)	PAPI_L2_TCM (E)	PAPI_TOT_CYC (I)	PAPI_TOT_CYC (E)
Experiment Aggregate Metrics	3.49e+10 100 %	3.49e+10 100 %	7.80e+06 100 %	7.80e+06 100 %	3.48e+10 100 %	3.48
main	3.49e+10 100 %		7.80e+06 100 %		3.48e+10 100 %	
MAIN	3.49e+10 100 %	3.45e+10 98.9%	7.80e+06 100 %	7.60e+06 97.4%	3.48e+10 100 %	3.45
loop at matrix2.f: 29	3.45e+10 98.9%				3.45e+10 99.0%	
loop at matrix2.f: 30	3.45e+10 98.9%			05 10.3%	3.45e+10 99.0%	1.20
loop at matrix2.f: 32	3.43e+10 98.3%			06 87.2%	3.43e+10 98.7%	3.43
matrix2.f: 32	3.43e+10 98.3%			06 87.2%	3.43e+10 98.7%	3.43
matrix2.f: 30	1.48e+08 0.4%	1.48e+08 0.4%	8.00e+05 10.3%	8.00e+05 10.3%	1.20e+08 0.3%	1.20
_wrap_mpi_init_	2.40e+08 0.7%				2.00e+08 0.6%	
_wrap_mpi_finalize_						
loop at matrix2.f: 21					7.20e+07 0.2%	
loop at matrix2.f: 16					7.20e+07 0.2%	
_f90io_ldw						



## HPCToolkit: How to analyse

---

- Values of individual metrics are of limited use by themselves
  - e.g. number of cache misses for a loop or routine is of little value in isolation – only useful when compared to the number of instructions executed or the number of cache accesses
  - hpcviewer can be used to calculate derived metrics via spreadsheet-like formulae (e.g. *cycles/instruction* from PAPI\_TOT\_CYC/PAPI\_TOT\_INS for a particular scope)

17

## HPCToolkit: How to analyse

---

- Derived metrics can be used to pinpoint and quantify inefficiencies
- Scalability bottlenecks can be pinpointed by differential analysis of two profiles run using different numbers of processes
- See *Effective Strategies for Analysis Program Performance with HPCToolkit* for more details
- <http://hpctoolkit.org>

18