



SC21

St. Louis, MO | science & beyond.

Identifying Scalability Bottlenecks with HPCToolkit

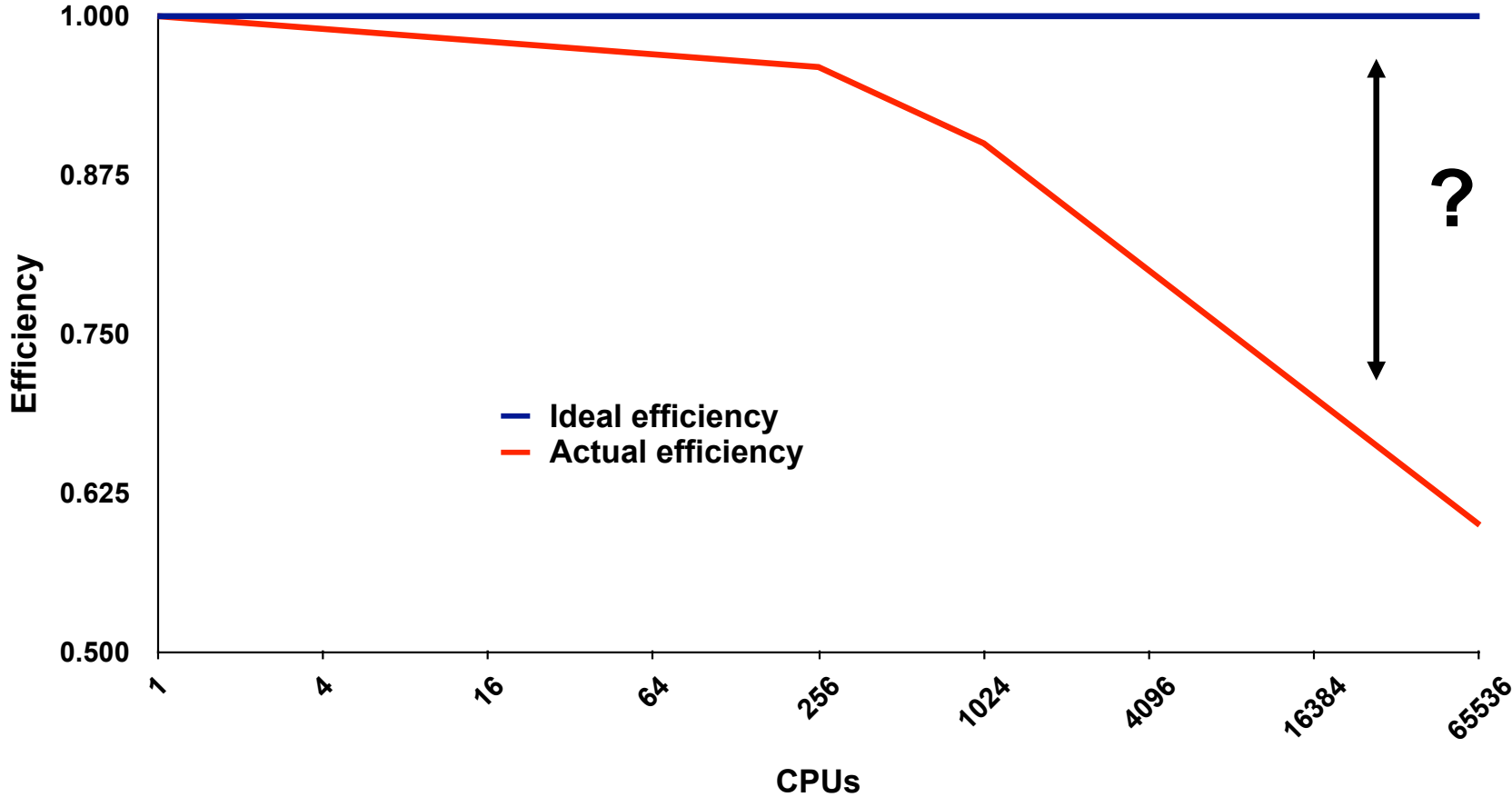
John Mellor-Crummey

Department of Computer Science
Rice University

15 November 2021



The Problem of Scaling

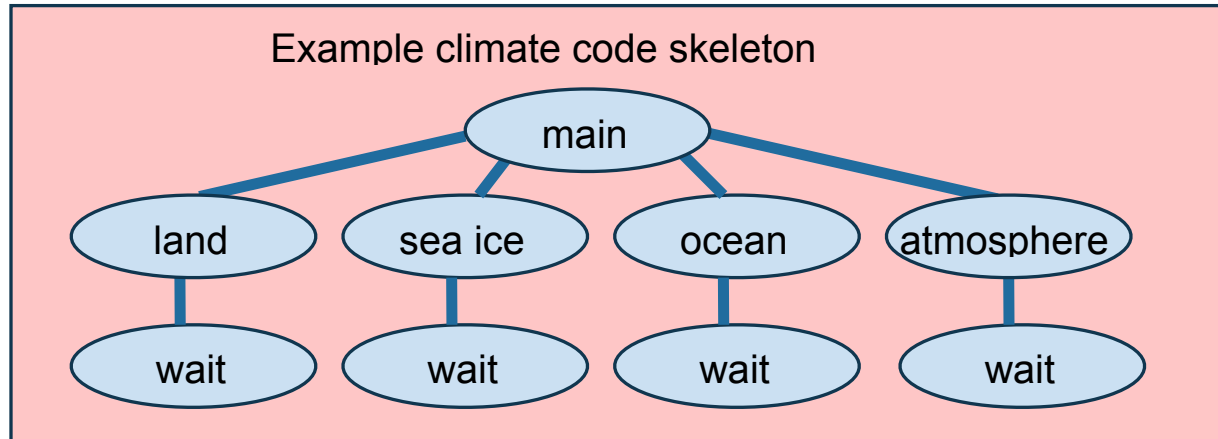


Goal: Automatic Scaling Analysis

- Pinpoint scalability bottlenecks
- Guide user to problems
- Quantify the magnitude of each problem
- Diagnose the nature of the problem

Challenges for Pinpointing Scalability Bottlenecks

- Parallel applications
 - modern software uses layers of libraries
 - performance is often context dependent

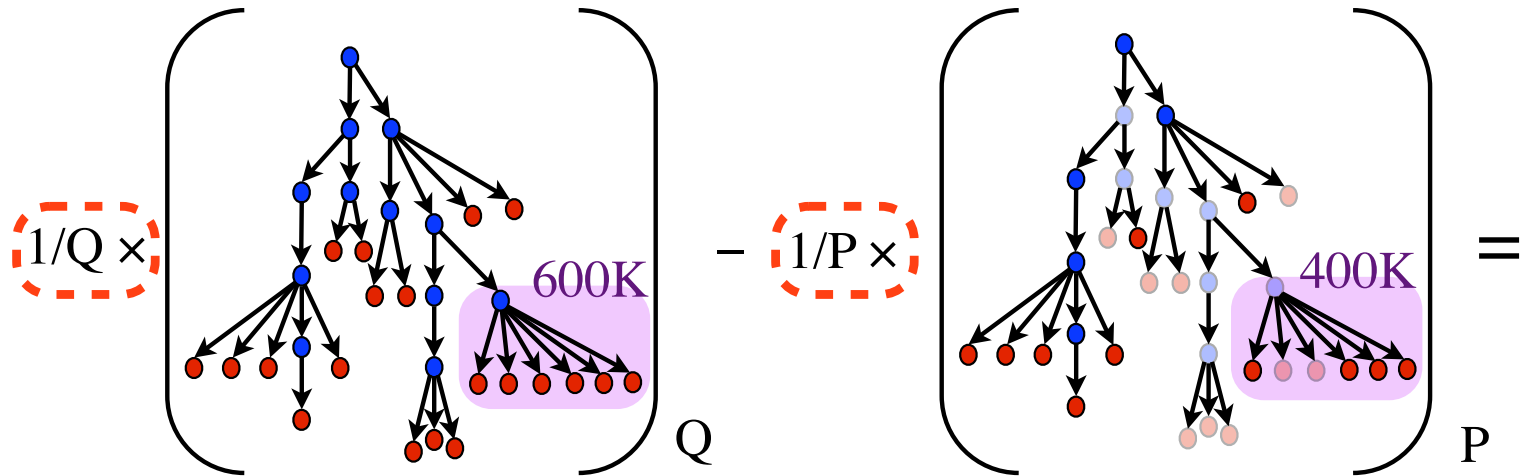


- Monitoring
 - bottleneck nature: computation, data movement, synchronization?
 - 2 pragmatic constraints (1) acceptable data volume, (2) low perturbation

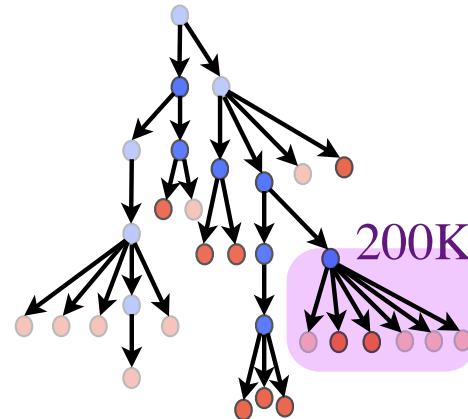
Scalability Analysis with Expectations

- **You have performance expectations for your parallel code**
 - strong scaling: linear speedup
 - weak scaling: constant execution time
- **Put your expectations to work**
 - measure performance under different conditions
 - e.g., different levels of parallelism and/or different inputs
 - express your expectations as an equation
 - compute the deviation from expectations for each calling context
 - for both inclusive and exclusive costs
 - correlate the metrics with the source code
 - explore the annotated call tree interactively

Pinpointing and Quantifying Scalability Bottlenecks

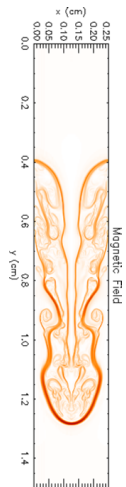


coefficients for analysis
of weak scaling

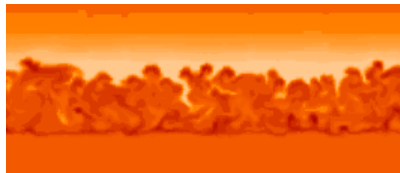


Scalability Analysis Demo

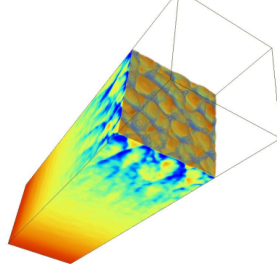
- Parallel, adaptive-mesh refinement (AMR) code
- Block structured AMR; a block is the unit of computation
- Designed for compressible reactive flows
- Can solve a broad range of (astro)physical problems
- Portable: runs on many massively-parallel systems
- Scales and performs well
- Fully modular and extensible: components can be combined to create many different applications



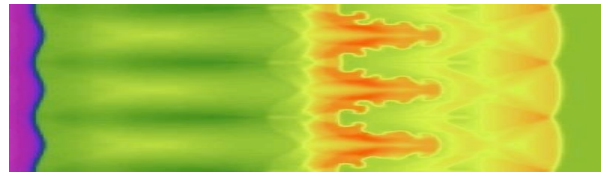
Magnetic
Rayleigh-Taylor



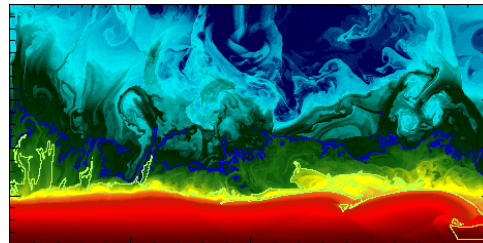
Nova outbursts on white dwarfs



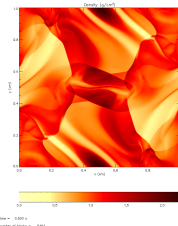
Cellular detonation



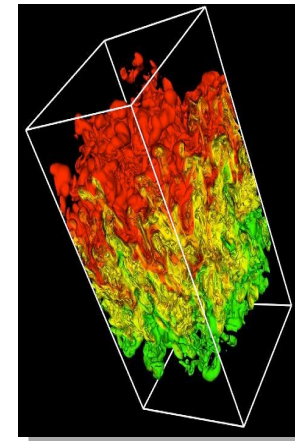
Laser-driven shock instabilities



Helium burning on neutron stars

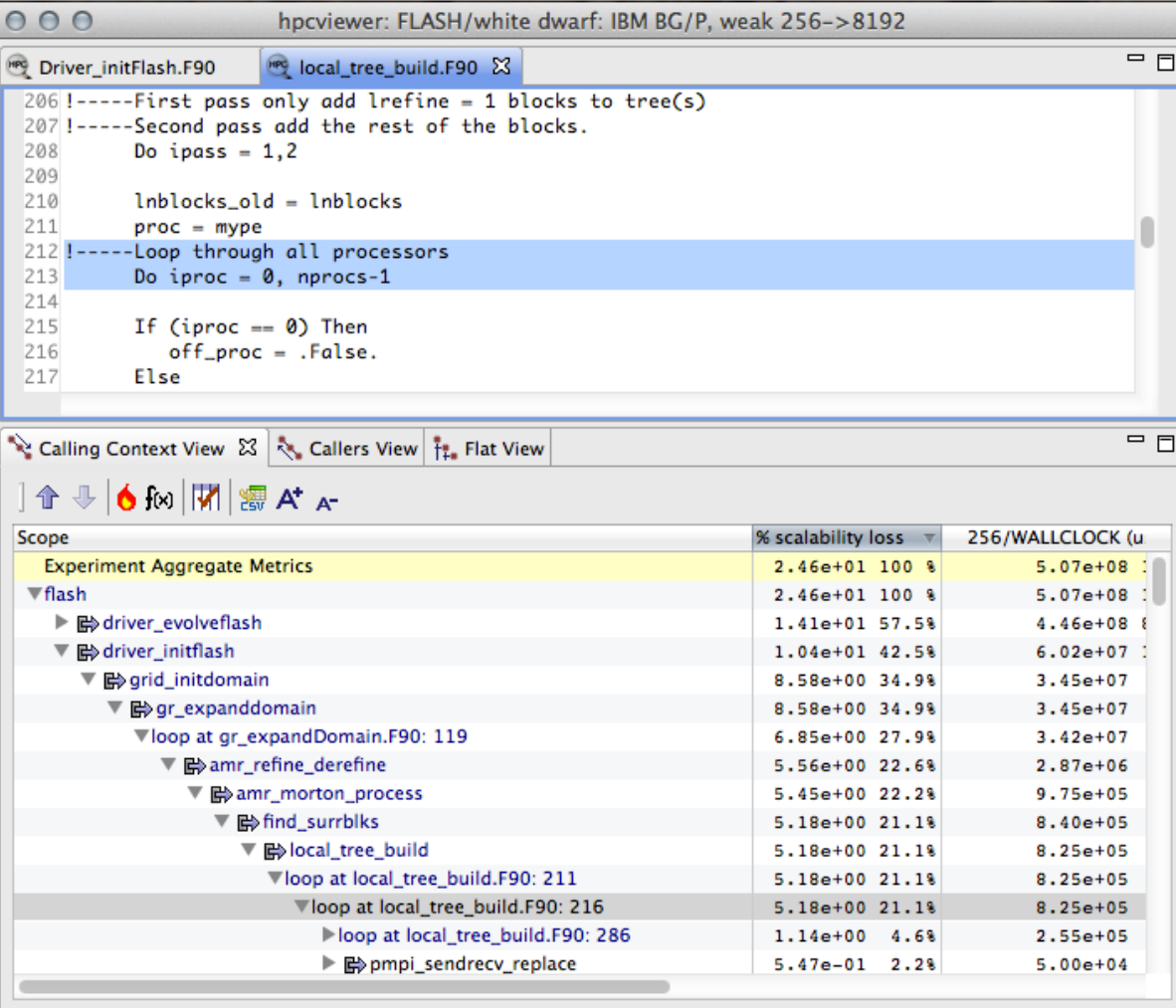


Orzag/Tang MHD
vortex



Rayleigh-Taylor instability

Scalability Analysis of Flash (Demo)



The screenshot displays the hpcviewer interface for the FLASH/white dwarf application. The top pane shows the source code for the `local_tree_build.F90` file, with lines 212-213 highlighted. The bottom pane shows a scalability analysis table with the following data:

Scope	% scalability loss	256/WALLCLOCK (u)
Experiment Aggregate Metrics	2.46e+01 100 %	5.07e+08
flash	2.46e+01 100 %	5.07e+08
driver_evolveflash	1.41e+01 57.5%	4.46e+08
driver_initflash	1.04e+01 42.5%	6.02e+07
grid_initdomain	8.58e+00 34.9%	3.45e+07
gr_expanddomain	8.58e+00 34.9%	3.45e+07
loop at gr_expandDomain.F90: 119	6.85e+00 27.9%	3.42e+07
amr_refine_derefine	5.56e+00 22.6%	2.87e+06
amr_morton_process	5.45e+00 22.2%	9.75e+05
find_surrblks	5.18e+00 21.1%	8.40e+05
local_tree_build	5.18e+00 21.1%	8.25e+05
loop at local_tree_build.F90: 211	5.18e+00 21.1%	8.25e+05
loop at local_tree_build.F90: 216	5.18e+00 21.1%	8.25e+05
loop at local_tree_build.F90: 286	1.14e+00 4.6%	2.55e+05
pmi_sendrecv_replace	5.47e-01 2.2%	5.00e+04

Scalability Analysis of FLASH

- Difference call path profile from two executions
 - different number of nodes
 - different number of threads
- Pinpoint and quantify scalability bottlenecks within or across nodes

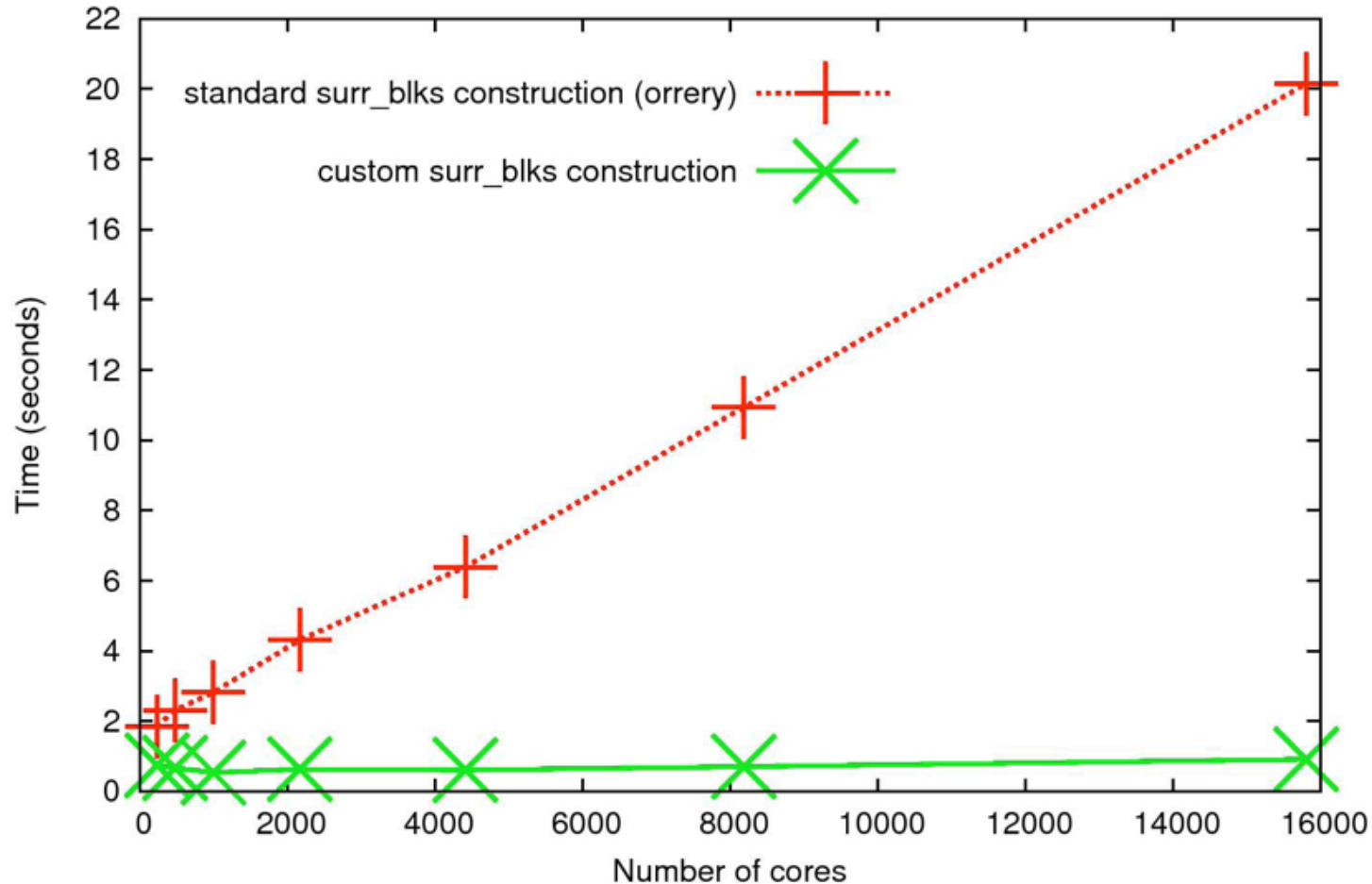
The screenshot shows the hpcviewer interface for the FLASH/white dwarf: IBM BG/P, weak 256->8192. The top pane displays code from Driver_initFlash.F90 and local_tree_build.F90. A red box highlights the following code block:

```
212 !-----Loop through all processors
213 Do iproc = 0, nprocs-1
214
215     If (iproc == 0) Then
216         off_proc = .False.
217     Else
```

An arrow points from this code block to a table in the bottom pane. The table shows the call path profile with columns for Scope, % scalability loss, and 256/WALLCLOCK (u). The highlighted row is 'loop at local_tree_build.F90: 216'.

Scope	% scalability loss	256/WALLCLOCK (u)
Experiment Aggregate Metrics	2.46e+01 100 %	5.07e+08
flash	2.46e+01 100 %	5.07e+08
driver_evolveflash	1.41e+01 57.5%	4.46e+08
driver_initflash	1.04e+01 42.5%	6.02e+07
grid_initdomain	8.58e+00 34.9%	3.45e+07
gr_expanddomain	8.58e+00 34.9%	3.45e+07
loop at gr_expandDomain.F90: 119	6.85e+00 27.9%	3.42e+07
amr_refine_derefine	5.56e+00 22.6%	2.87e+06
amr_morton_process	5.45e+00 22.2%	9.75e+05
find_surrblks	5.18e+00 21.1%	8.40e+05
local_tree_build	5.18e+00 21.1%	8.25e+05
loop at local_tree_build.F90: 211	5.18e+00 21.1%	8.25e+05
loop at local_tree_build.F90: 216	5.18e+00 21.1%	8.25e+05
loop at local_tree_build.F90: 286	1.14e+00 4.6%	2.55e+05
pmpi_sendrecv_replace	5.47e-01 2.2%	5.00e+04

Improved Flash Scaling of AMR Setup



Scalability Loss in Practice

- **Try computing scaling losses across nodes as you increase the node count**
- **Try computing scaling losses within nodes as you increase the thread count**

- **How?**
 - use hpcrun to monitor an execution at scale P
 - e.g. `mpirun -n 100 hpcrun -o foo.m ...`
 - use hpcrun to monitor an execution at scale Q
 - e.g. `mpirun -n 1000 hpcrun -o bar.m ...`
 - use hpcstruct to analyze binaries in each directory
 - invoke hpcprof passing both of the directories
 - e.g., `hpcprof foo.m bar.m`
 - open the resulting database in hpcviewer and use the equation to analyze your losses!

References

- Cristian Coarfa, John Mellor-Crummey, Nathan Froyd, and Yuri Dotsenko. 2007. Scalability analysis of SPMD codes using expectations. In Proceedings of the 21st annual International Conference on Supercomputing (ICS '07). Association for Computing Machinery, New York, NY, USA, 13–22. DOI:<https://doi.org/10.1145/1274971.1274976>
- Rice University, HPCToolkit Users Manual. <http://hpctoolkit.org/manual/HPCToolkit-users-manual.pdf>