

VI-HPS

SOFTWARE

0.00 <<time step loop>>
0.00 updatedt
6.62 updatex
372.85 updatelen
0.00 gene
0.00 <<iteration loop>>
293.65 genbc

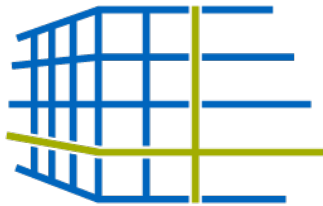


FAST SOLUTIONS

☒ PAPI_L1_DCM
☒ PAPI_L1_ICM
☐ PAPI_L2_DCM
☒ PAPI_L2_ICM
☒ PAPI_L1_TCM
☐ PAPI_L2_TCM

PRODUCTIVITY

Performance Analysis Basics



Munich Centre for
Advanced Computing

Michael Gerndt (gerndt@in.tum.de)

Felix Wolf (f.wolf@grs-sim.de)

October 31 2010



جامعة الملك عبد الله
للعلوم والتقنية
King Abdullah University of
Science and Technology



German Research School
for Simulation Sciences



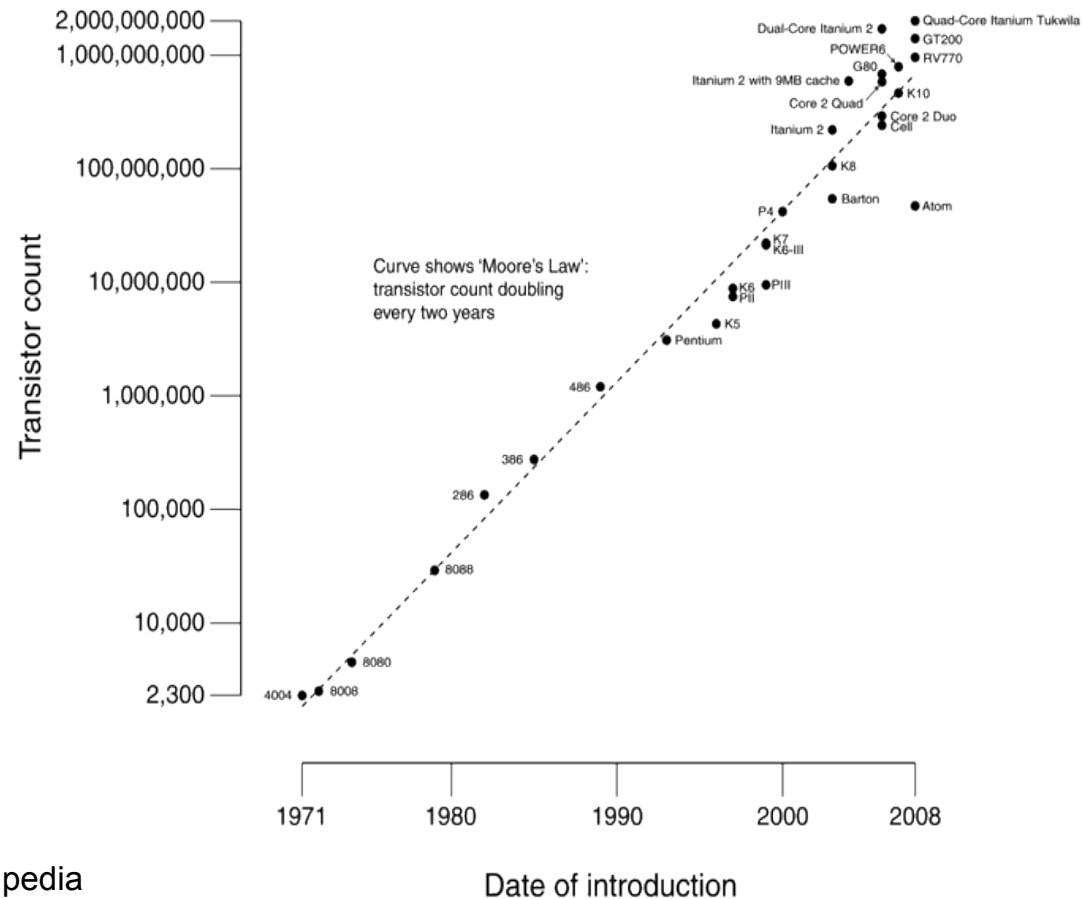
Universität Stuttgart



1. Motivation
2. Performance analysis basics
3. Measurement techniques
4. Analysis techniques
5. The tools

- Why performance analysis at all? Moore's Law still in charge

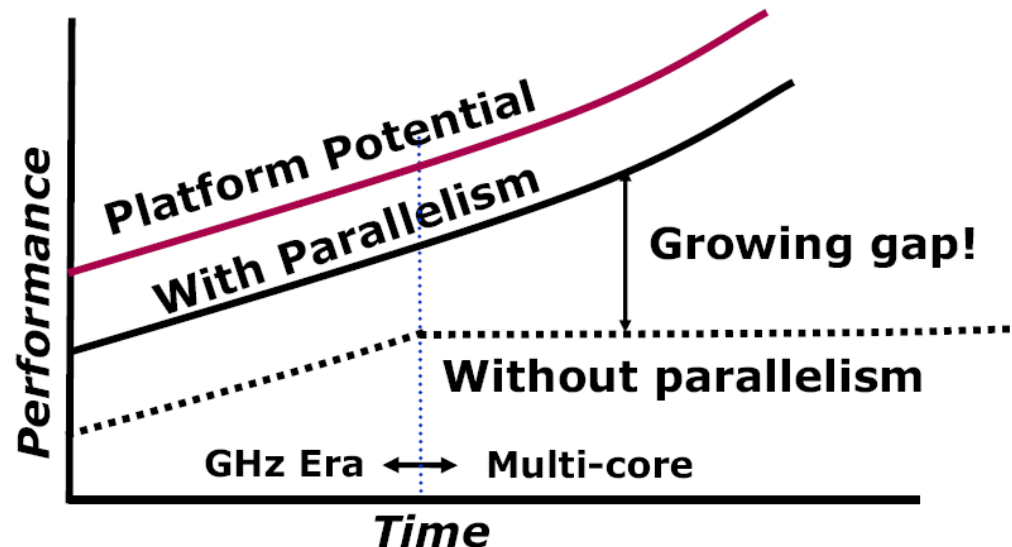
CPU Transistor Counts 1971-2008 & Moore's Law



Source: Wikipedia

- But *Free Lunch* is over

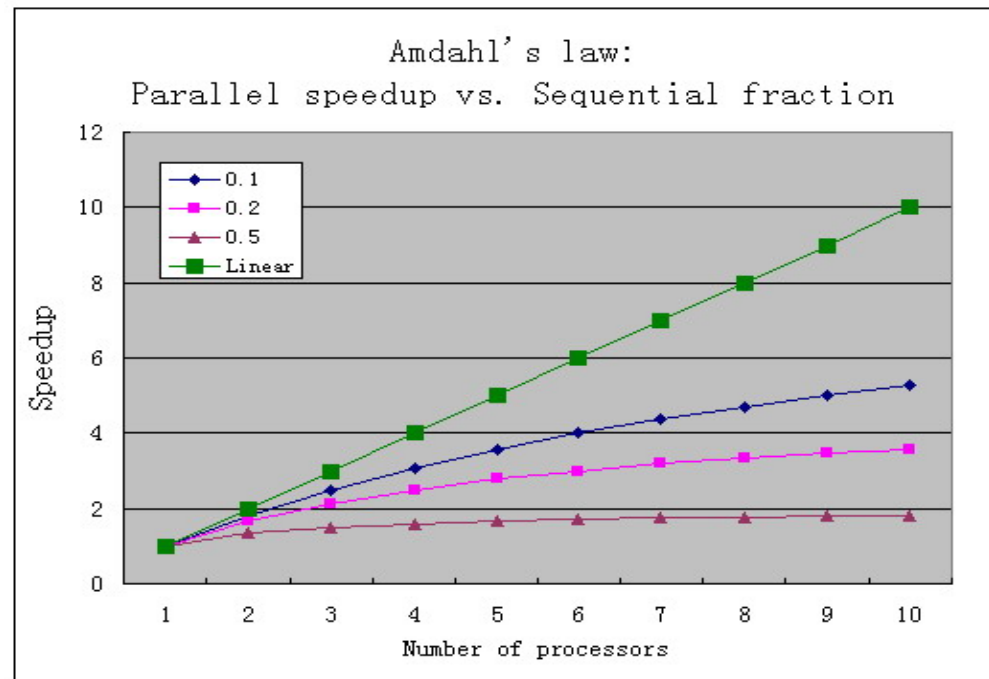
Need for Parallelism



Parallelism is crucial for optimal performance

- Two parallel architectures
 - Shared memory
 - OpenMP
 - Pthreads
 - Threading Building Blocks
 - ...
 - Distributed memory
 - MPI
 - Partitioned Global Address Space (e.g., UPC)
 - Sockets
- Today also heterogeneous systems
 - CUDA, OpenCL
 - Not covered here

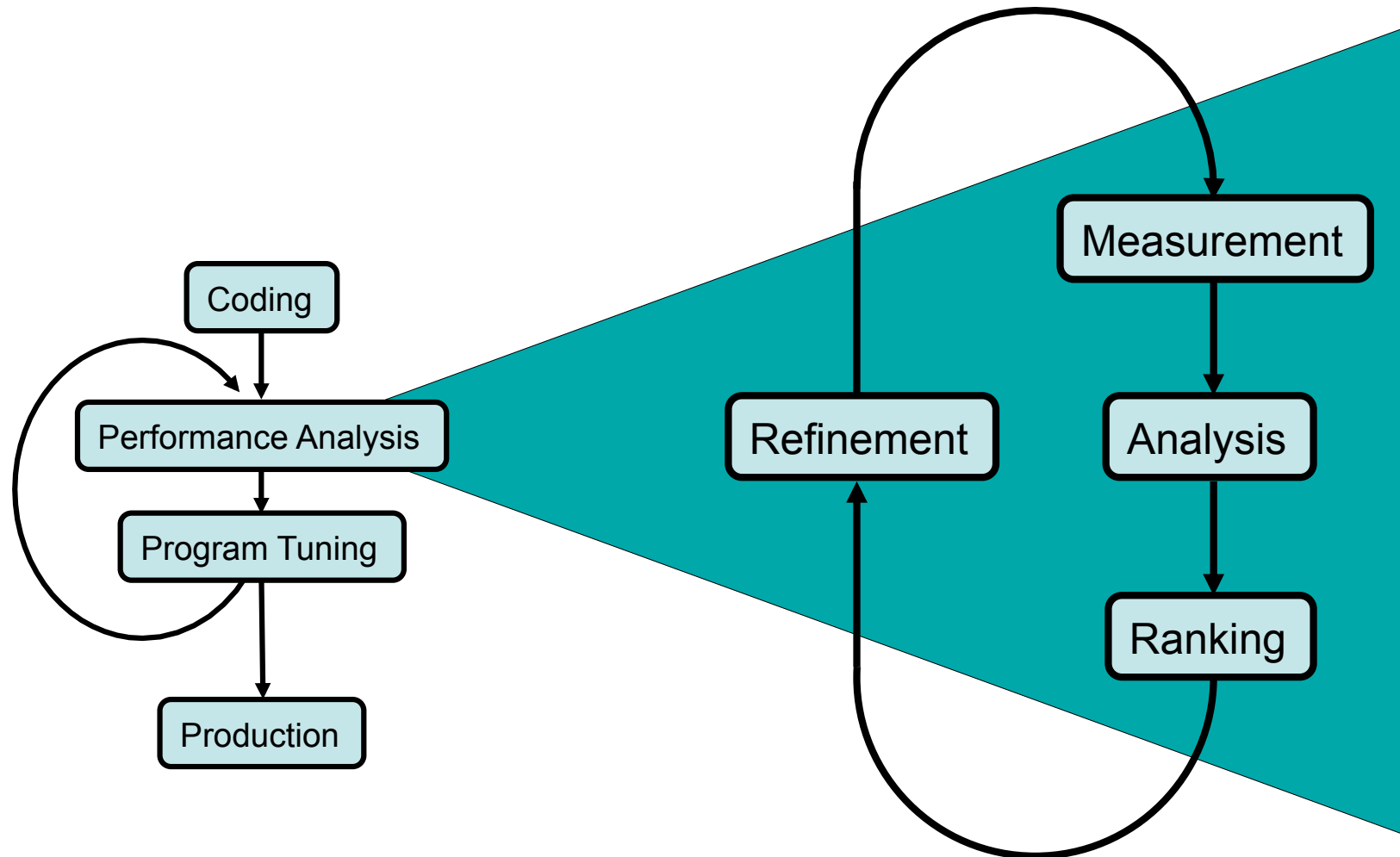
- Amdahl's Law:
 - The speedup of a program using multiple processors in parallel computing is limited by the time needed for the sequential fraction of the program.



Source: Wikipedia

- High complexity in parallel and distributed systems
 - Application
 - Algorithm, data structures
 - Parallel programming interface
 - Compiler, parallel libraries, communication, synchronization
 - Operating system
 - Process and memory management, IO
 - Hardware
 - CPU, memory, network

- Factors which influence performance of parallel programs
 - “Sequential” factors
 - Computation
 - Cache and memory
 - Input / output
 - “Parallel” factors
 - Communication (message passing)
 - Threading
 - Synchronization
 - Parallel I/O



- **Performance analysis** determines the performance on a given machine.
- **Performance prediction** allows to evaluate programs for a hypothetical machine. It is based on:
 - runtime data of an actual execution
 - machine model of the target machine
 - analytical techniques
 - simulation techniques
- **Benchmarking** determines the performance of a computer system on the basis of a set of typical applications.

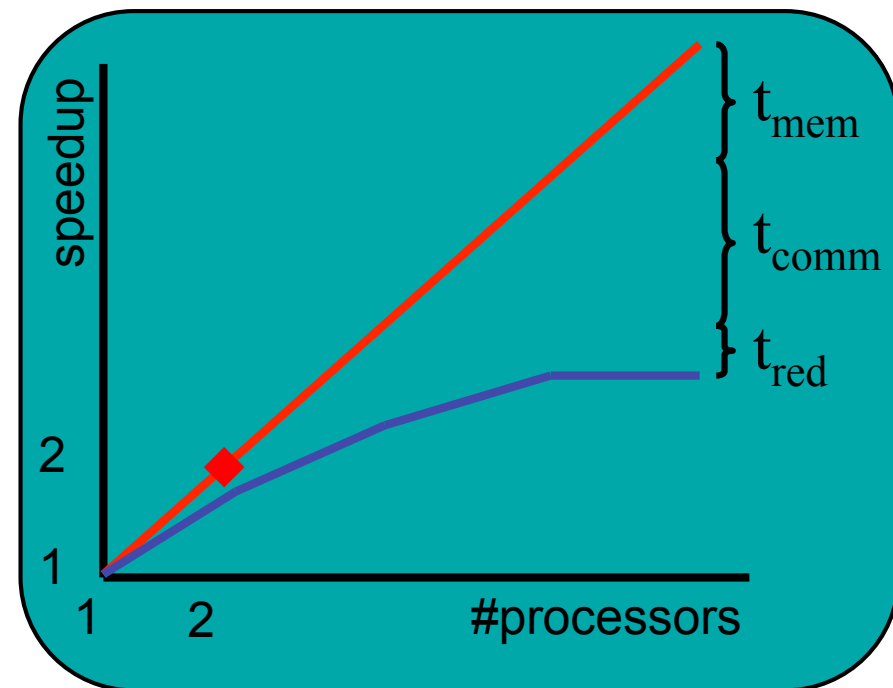
1. Motivation
2. Performance analysis basics
3. Measurement techniques
4. Analysis techniques
5. The tools

- How to decide whether a code performs well:
 - Comparison of measured FLOPS with peak performance
 - Comparison with a sequential version

$$\text{speedup}(p) = \frac{t_s}{t_p}$$

- Estimate distance to ideal time via overhead classes

- t_{mem}
- t_{comm}
- t_{sync}
- t_{red}
- ...



- Scaled speedup

- Problem size grows with machine size

$$\text{scaled_speedup}(p) = \frac{t_s(n_p)}{t_p(n_p)}$$

- Speedup can not be defined as $\text{Time}(1) / \text{Time}(p)$ for scaled up problem since $\text{time}(1)$ is hard to measure and inappropriate
- Insert $\text{performance} = \text{work} / \text{time}$ in speedup formula gives

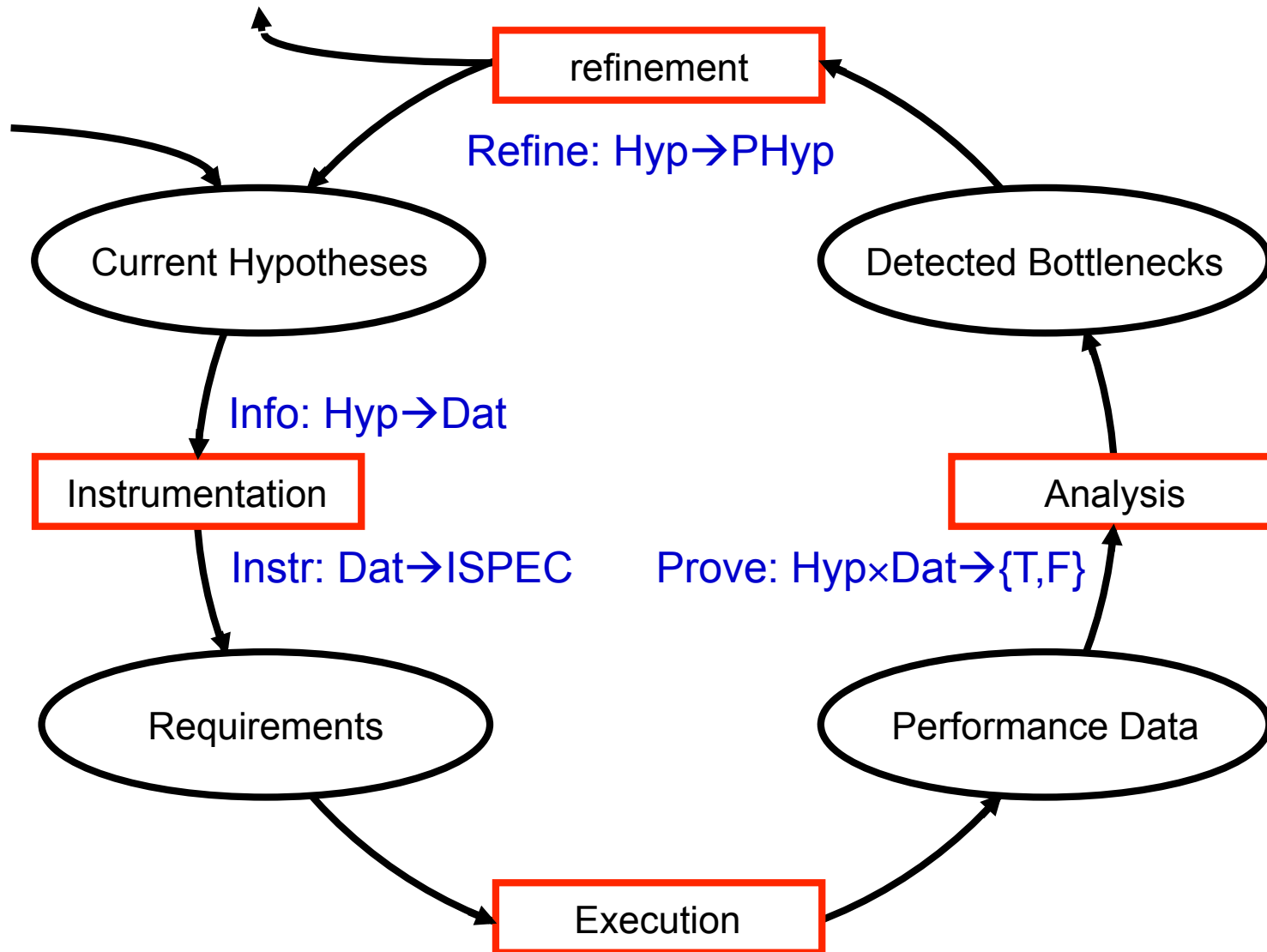
$$\begin{aligned} \text{Speedup}_{MC}(p) &= \frac{\text{Work}(p)}{\text{Time}(p)} / \frac{\text{Work}(1)}{\text{Time}(1)} \\ &= \frac{\text{Work}(p)}{\text{Work}(1)} / \frac{\text{Time}(p)}{\text{Time}(1)} \\ &= \frac{\text{Increase in Work}}{\text{Increase in Time}} \end{aligned}$$

- Parallel efficiency: Percentage of ideal speedup

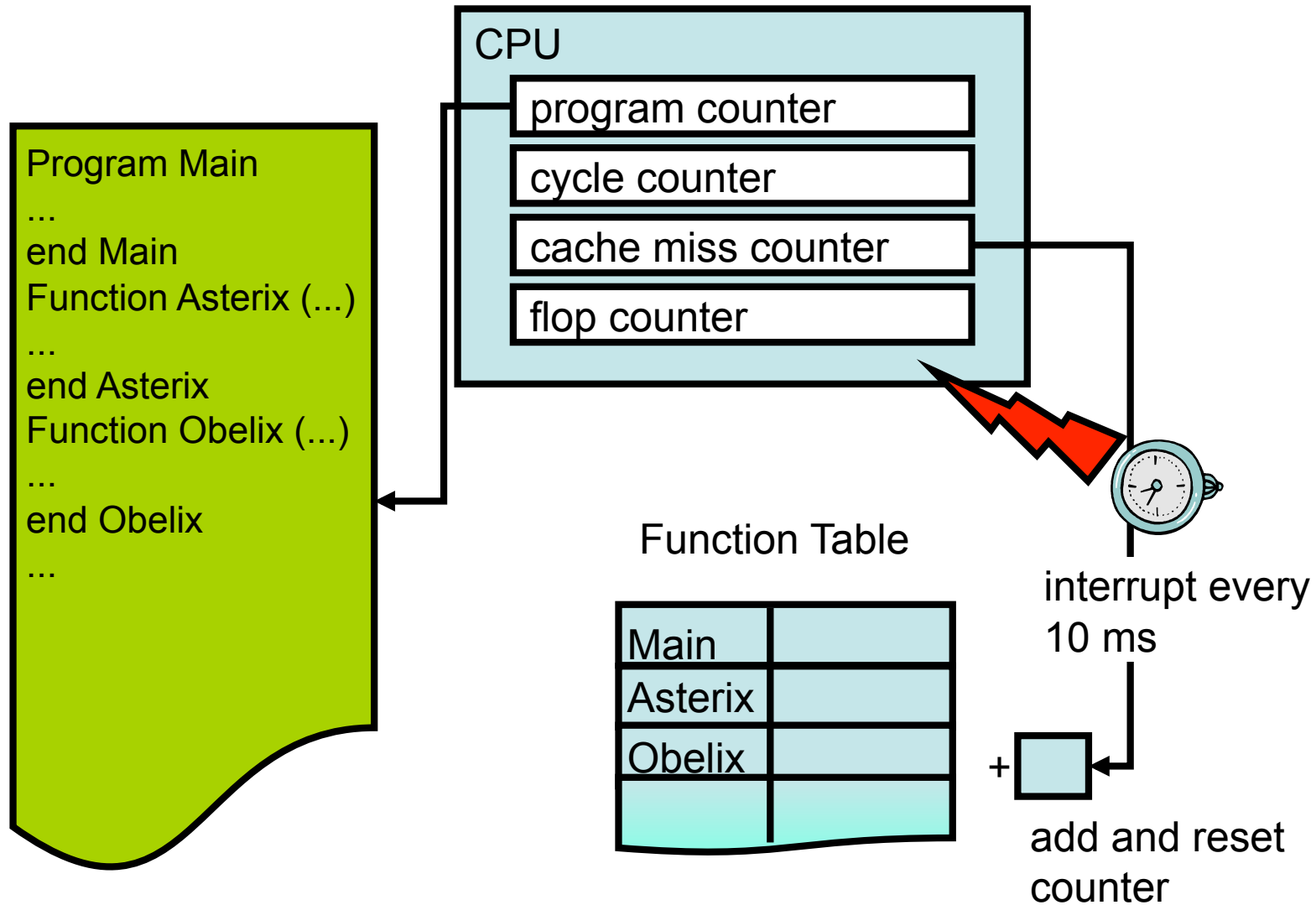
$$\text{efficiency}(p) = \text{speedup}(p) / p = \frac{t_s}{t_p * p}$$

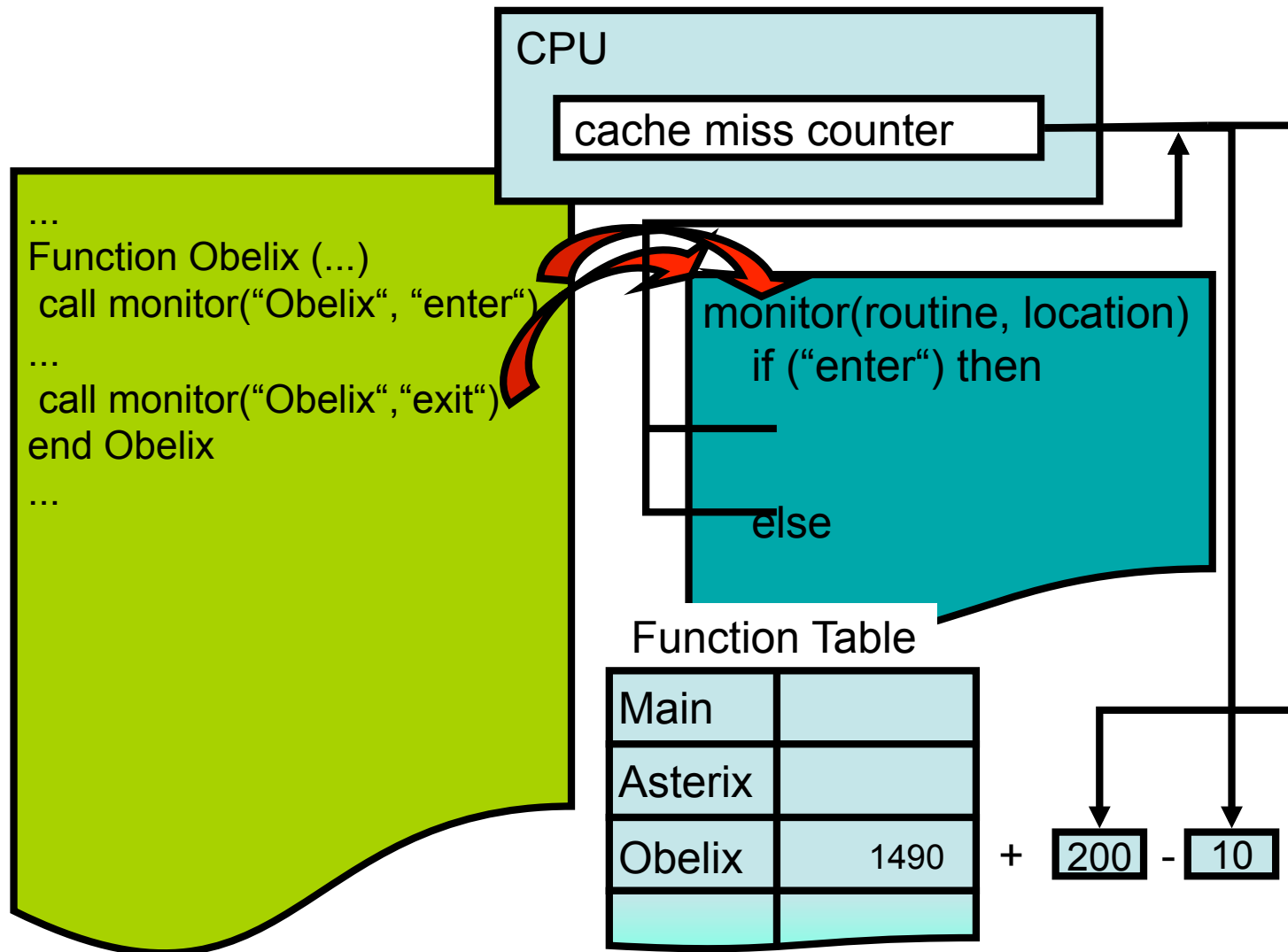
- Successful tuning is a combination of
 - right algorithms and libraries
 - compiler flags and directives
 - thinking!
- Measurement is better than guessing:
 - to determine performance problems
 - to validate tuning decisions and optimization
- Measurement should be repeated after each significant code modification and optimizations

- Do I have a performance problem at all?
 - Compare MFlops/MOps to typical rate
 - Speedup measurements
- What are the hot code regions?
 - Flat profiling
- Is there a bottleneck in those regions?
 - Single node: Hardware counter profiling
 - Parallel: Synchronization and communication analysis profiling
- Does the bottleneck vary over time or processor space?
 - Profiling individual processes and/or threads
 - Tracing
- Does the code behave similar for different configurations?
 - Analyze runs with different processor counts
 - Analyze different input configurations



- Event model of the execution
 - Events occur at a processor at a specific point in time
 - Events belong to event types
 - clock cycles
 - cache misses
 - remote references
 - start of a send operation
 - ...
- Profiling: Recording accumulated performance data for events
 - Sampling: Statistical approach
 - Instrumentation: Direct measurement
- Tracing: Recording performance data of individual events



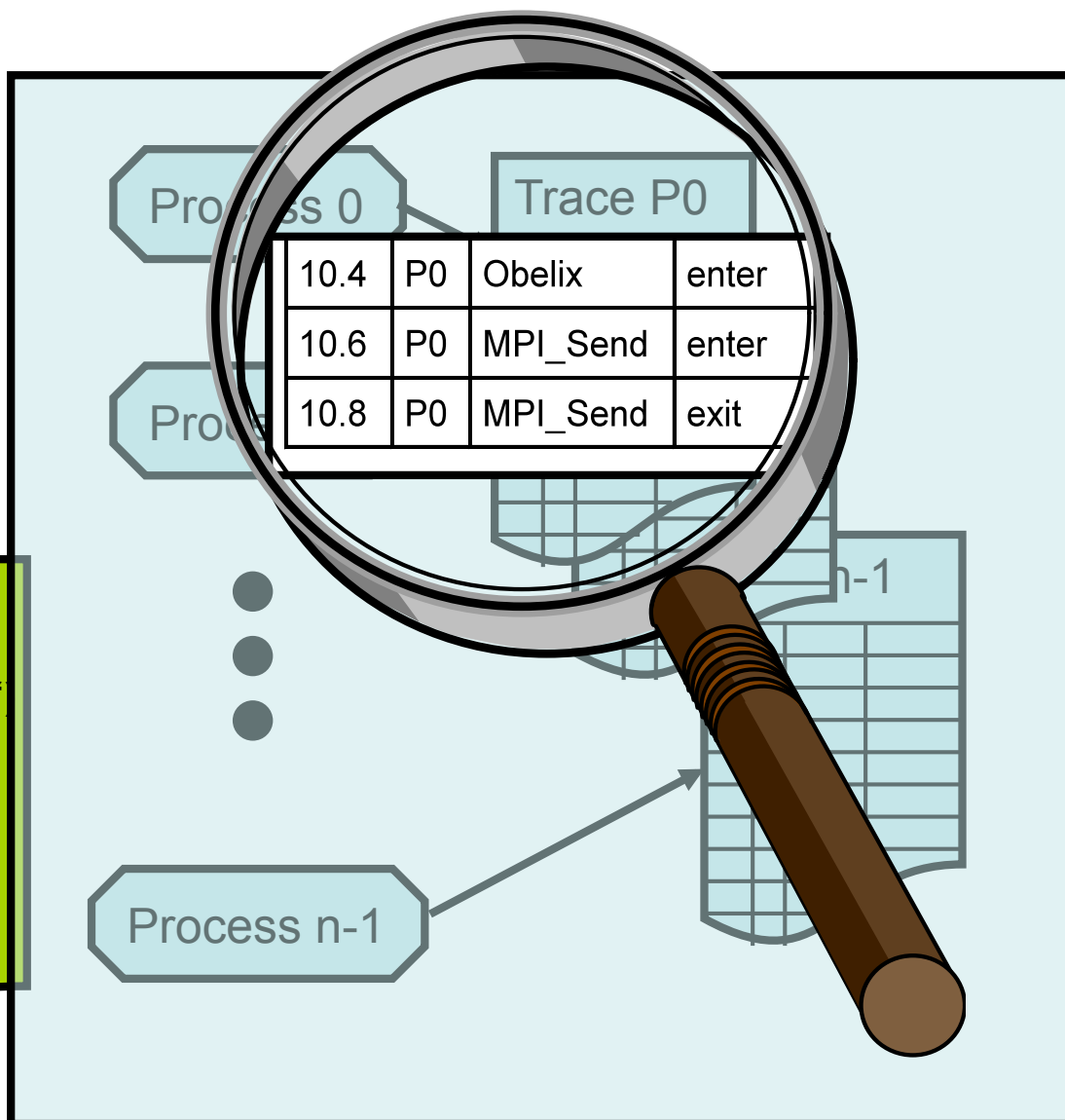


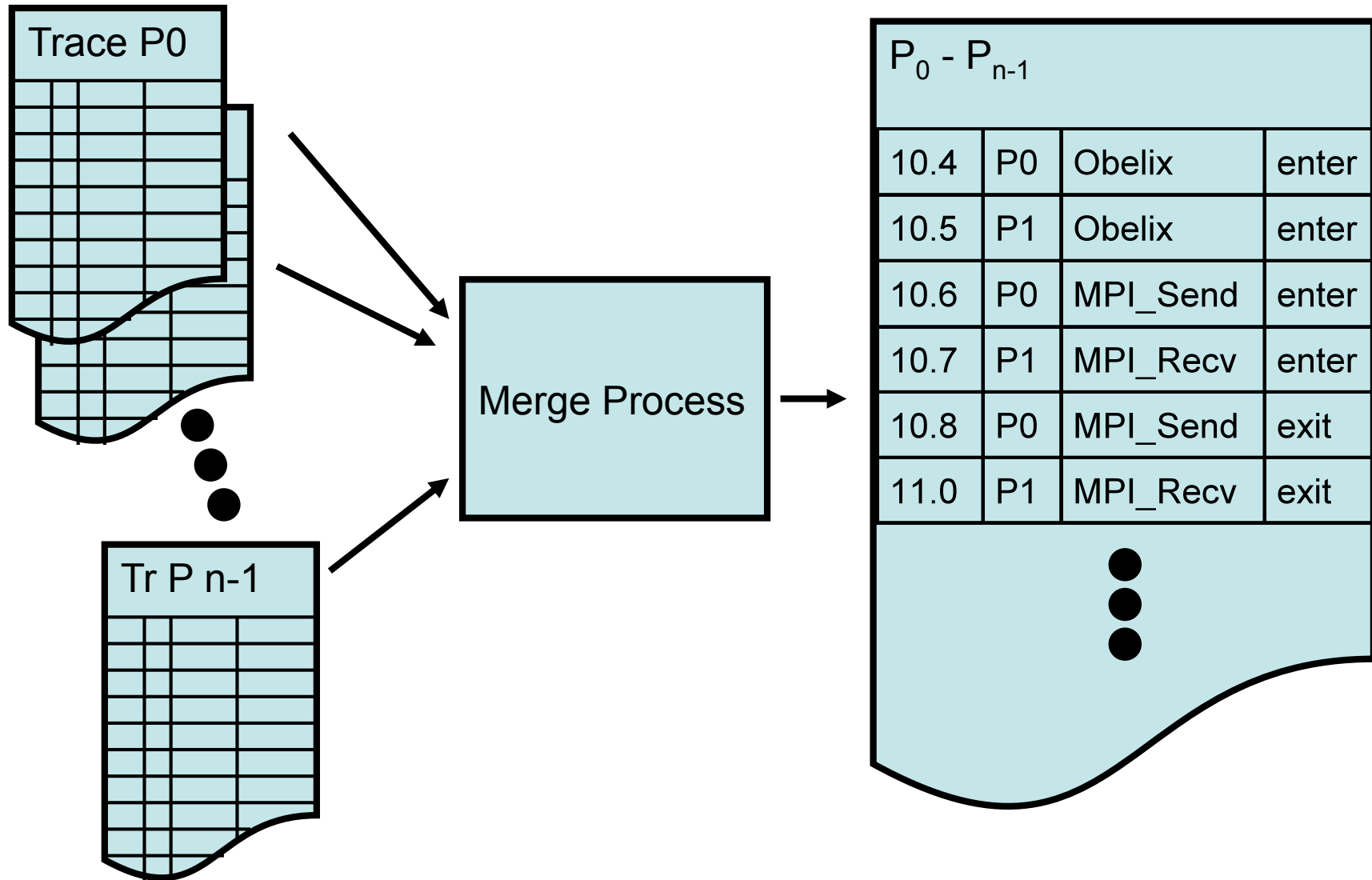
- Source code instrumentation
 - done by the compiler, source-to-source tool, or manually
 - + portability
 - + link back to source code easy
 - re-compile necessary when instrumentation is changed
 - difficult to instrument mixed-code applications
 - cannot instrument system or 3rd party libraries or executables
- Object code instrumentation
 - „patching“ the executable to insert hooks (like a debugger)
 - inverse pros/cons
 - Offline
 - Online

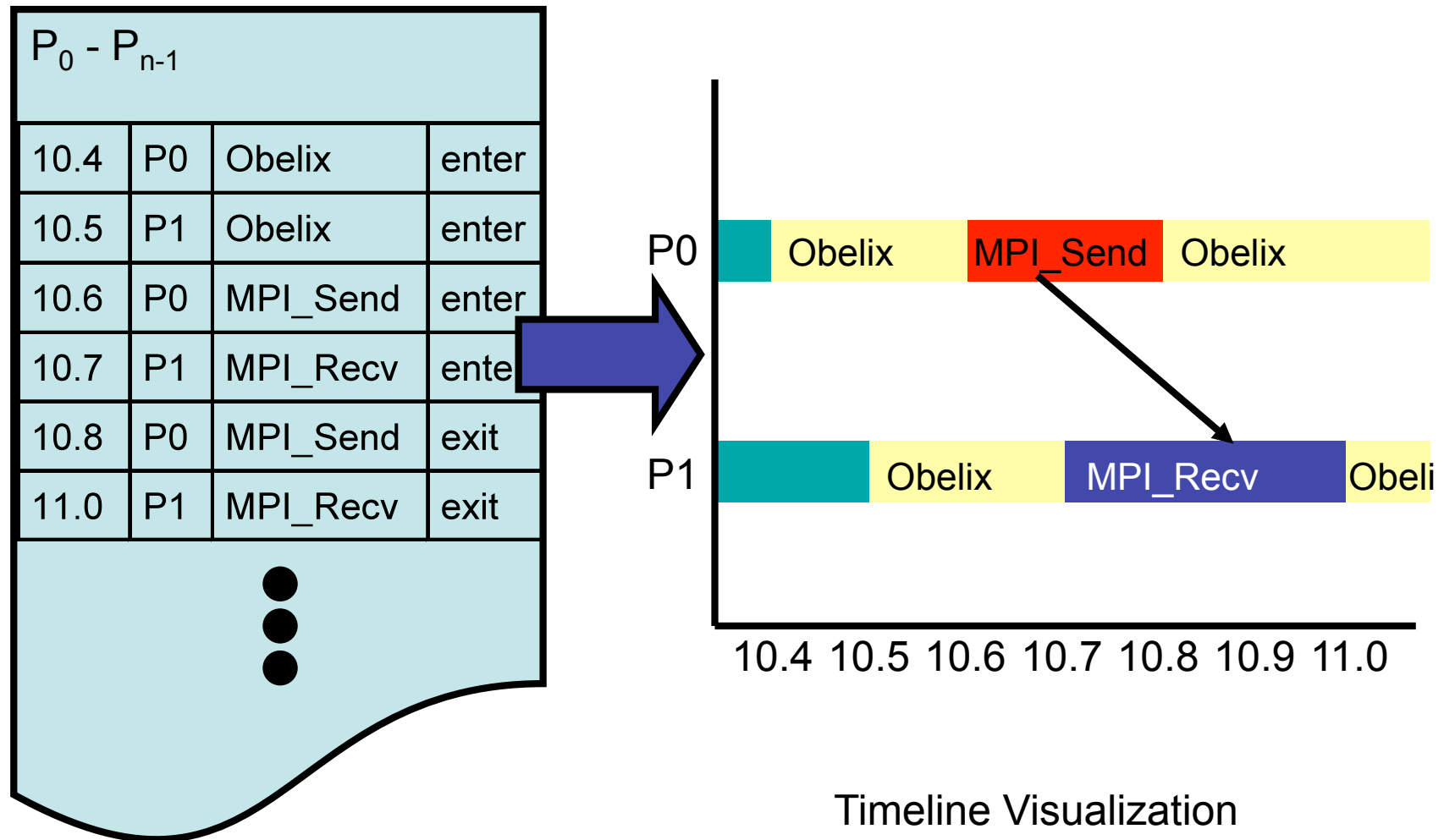
```
...
Function Obelix (...)
  call monitor("Obelix", "enter")
...
  call monitor("Obelix", "exit")
end Obelix
...
```

MPI Library

```
Function MPI_send (...)
  call monitor("MPI_send", "enter")
...
  call PMPI_send(...)
...
  call monitor("MPI_send", "exit")
end MPI_send
...
```

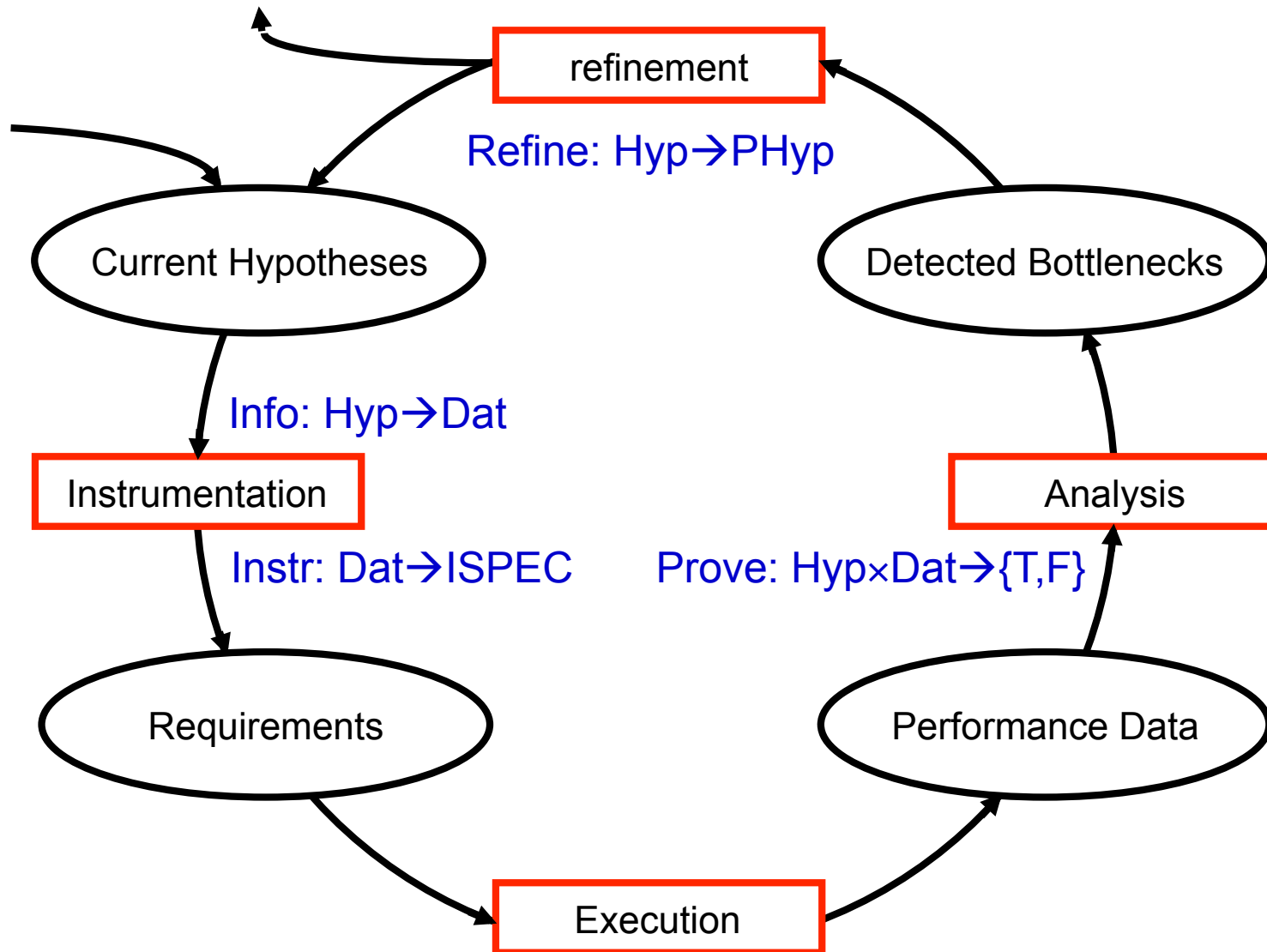






- Profiling
 - recording summary information (time, #calls, #misses...)
 - about program entities (functions, objects, basic blocks)
 - very good for quick, low cost overview
 - points out potential bottlenecks
 - implemented through sampling or instrumentation
 - moderate amount of performance data
- Tracing
 - recording information about events
 - trace record typically consists of timestamp, processid, ...
 - output is a trace file with trace records sorted by time
 - can be used to reconstruct the dynamic behavior
 - creates huge amounts of data
 - needs selective instrumentation

1. Motivation
2. Performance analysis basics
3. Measurement techniques
4. Analysis techniques
5. The tools



- Single node performance
 - Excessive number of 2nd-level cache misses
 - Low number of issued instructions
- IO
 - High data volume
 - Sequential IO due to IO subsystem or sequentialization in the program
- Excessive communication
 - Frequent communication
 - High data volume

- Frequent synchronization
 - All-to-all operations
 - Barrier operations
- Load balancing
 - Wrong data decomposition
 - Dynamically changing load

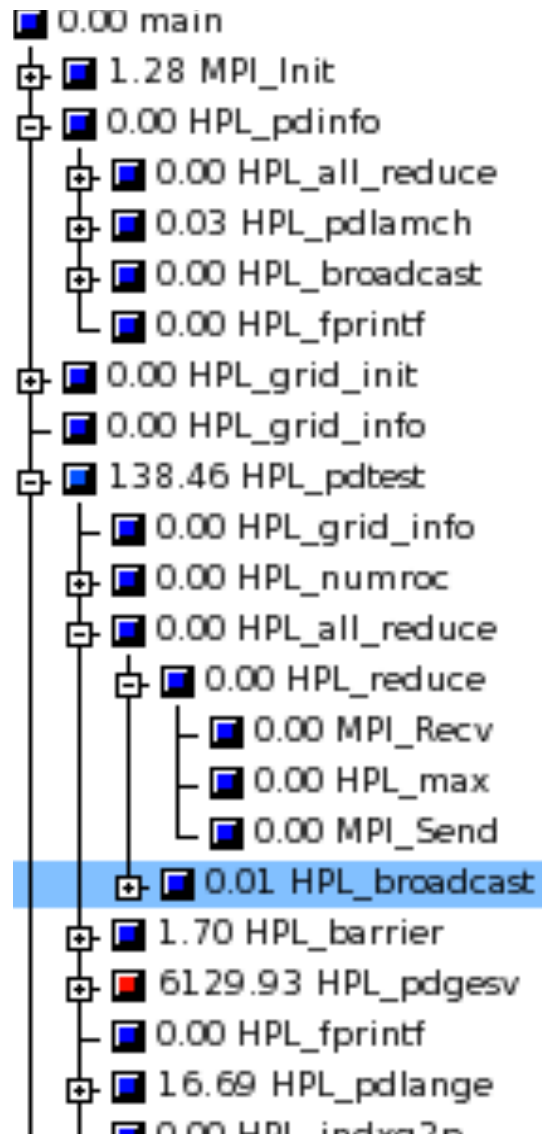
- Single node performance
 - ...
- IO
 - ...
- Excessive communication
 - Large number of remote memory accesses
 - False sharing
 - False data mapping
- Frequent synchronization
 - Implicit synchronization of parallel constructs
 - Barriers, locks, ...
- Load balancing
 - Uneven scheduling of parallel loops
 - Uneven work in parallel sections

- Offline vs online analysis
 - Offline: first generate data then analyse
 - Online: generate and analyze data while application is running
 - Online requires automation → limited to standard bottlenecks
 - Offline suffers more from size of measurement information
- Three techniques to support user in analysis
 - Source-level presentation of performance data
 - Graphical visualization
 - Ranking of high-level performance properties

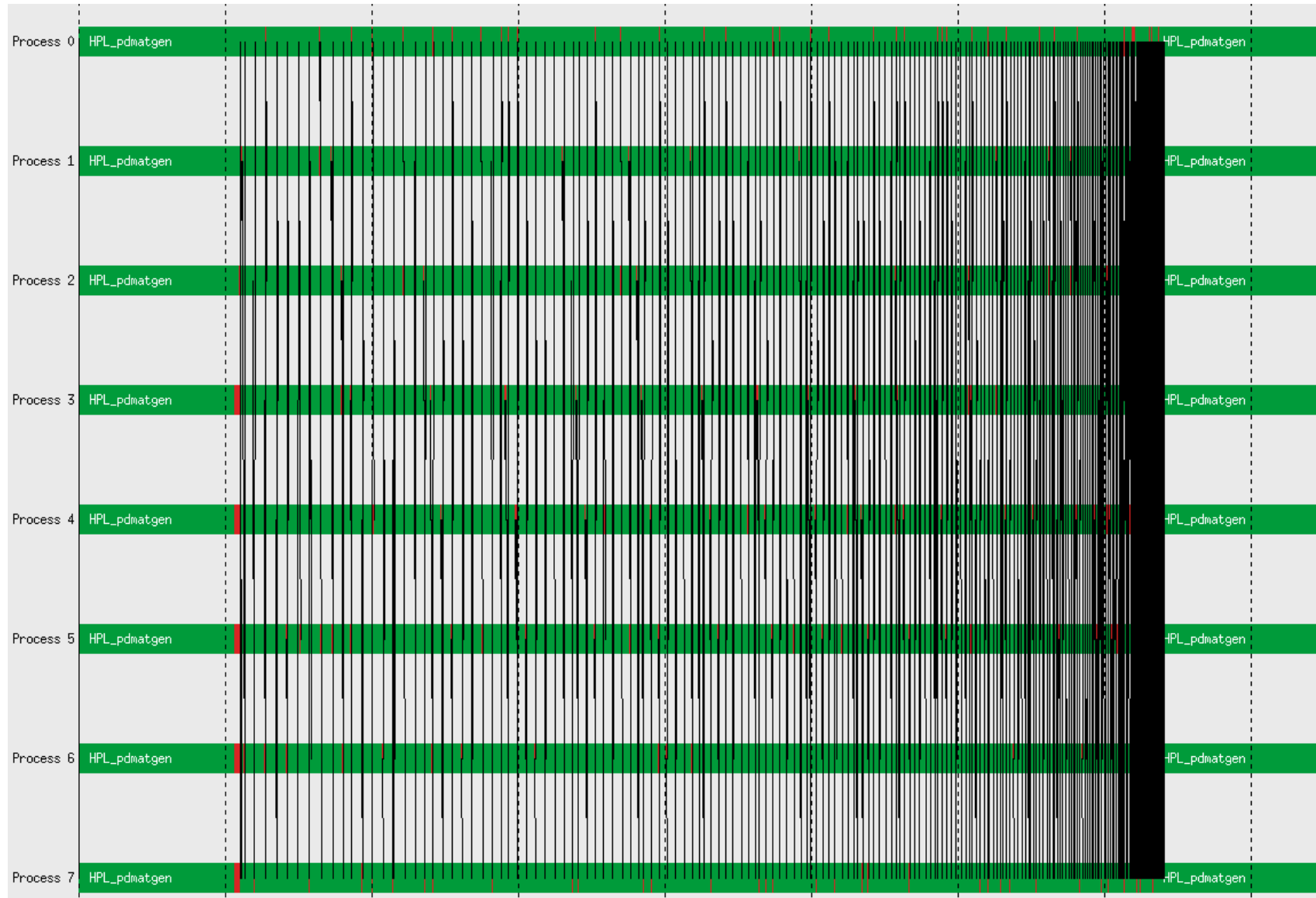
```
time      % region
6294.91  100.00 (summary) ALL
388.07   6.16  (summary) MPI
5748.72  91.32 (summary) USR
158.12   2.51  (summary) COM

1.89     0.03 HPL_bcast
2.06     0.03 HPL_bcast_1rinM
22.73    0.36 HPL_dtrsm
5618.52  89.25 HPL_dgemm
37.41    0.59 HPL_dlaswp00N
0.01     0.00 HPL_dlamc3
0.50     0.01 HPL_dcopy
11.54    0.18 HPL_dgemv
2.78     0.04 HPL_idamax
0.01     0.00 HPL_dlocmax
0.02     0.00 HPL_pdmxswp
0.21     0.00 HPL_dlocswpT
0.43     0.01 HPL_dscal
0.02     0.00 HPL_pdrpanrlT
0.08     0.00 HPL_dlatcpy
0.04     0.00 HPL_pdpancrT
304.69   4.84 MPI_Send
82.32    1.31 MPI_Recv
0.00     0.00 HPL_numrocI
1.89     0.03 HPL_pdupdateTT
0.02     0.00 HPL_pdpanel_init
```

excerpt from scalasca



excerpt from scalasca



excerpt from vampir

Analysis Techniques: Properties on Source Level



Fortran - psctests/aux_fields-psc.f90 - Eclipse

File Edit Refactor Navigate Search Project Run Window Help

Fortran Projects Navigator aux_fields-psc.f90 Outline SIR Outline V Make Targets

GENE
└─ COMPMOD
└─ cx_parallel
└─ INST
└─ INSTMOD
└─ mydir
└─ OBJ
└─ PRE
└─ .project
└─ 128cpus.out
└─ appl
└─ appl.sir
└─ aux_fields.F90
└─ aux_fields-psc.f90
└─ aux_func.F90
└─ aux_func-psc.f90
└─ boundary.F90
└─ boundary-psc.f90
└─ calc_rhs_helper.F90
└─ calc_rhs_helper-psc.f90
└─ calc_rhs.F90
└─ calc_rhs-psc.f90
└─ cheese.F90
└─ cheese-psc.f90
└─ checkpoint.F90
└─ checkpoint-psc.f90
└─ circular.F90
└─ circular-psc.f90
└─ collisions.F90
└─ collisions-psc.f90
└─ comm.F90
└─ comm-psc.f90
└─ debug_output.F90
└─ debug_output-psc.f90
└─ diag.F90
└─ diag-psc.f90
└─ dummyperf.F90

```
34 subroutine calc_rest(g_in,emfields_out,f_1_out)
35   complex,dimension(l1:l2,lj1:lj2,lk1:lk2,ll1:ll2,lm1:lm2)
36   complex,dimension(lbx:ubx,lj1:lj2,lbz:ubz,lbv:ubv,lbw:ubw)
37   complex,dimension(lbx:ubx,lj1:lj2,lbz:ubz,1:n_fields),int
38
39   call perfon('calcrest')
40
41   if (x_local) then
42     CALL field_solve_kkxy(g_in,emfields_out)
43   else
44     CALL field_solve_xky(g_in,emfields_out)
45   endif
46
47   if (n_fields.eq.1) then
48     if(perf_vec(2).eq.1) then
49       call calc_f_es(g_in,f_1_out)
50     else
51       call calc_f_es_perf(g_in,f_1_out)
52     endif
53   else
54     if(perf_vec(2).eq.1) then
55       call calc_f_em(g_in,emfields_out(:, :, 2), f_1_out)
56     else
57       call calc_f_em_perf(g_in,gy_pre,emfields_out(:, :, 2), f_1_out)
58     endif
59   endif
60
61   call exchange_5df(f_1_out)
62
63   if (n_procs v.gt.1) call exchange_v(f_1_out)
64   if (collision op.ne.'none') call exchange_mu(f_1_out)
```

SIR File: /home/rainy/workspaces/LRR/psctests/GEN
└─ subroutine: CALC_REST (54/220) (aux_fields-psc.f90)
└─ call: FIELD_SOLVE_KKXY (166/166) (aux_fields-psc.f90)
└─ subroutine: CALFULLRHS_KKXY_1 (40/119) (calc_rhs-psc.f90)
└─ call: MY_REAL_MAX_TO_ALL (79/79) (calc_rhs-psc.f90)
└─ subroutine: MY_REAL_MAX_TO_ALL (58/172) (comm-psc.f90)
└─ call: MPI_ALLREDUCE (114/114) (comm-psc.f90)
└─ subroutine: MY_COMPLEX_SUM_VWSPEC (492/89) (comm-psc.f90)
└─ call: MPI_ALLREDUCE (406/406) (comm-psc.f90)
└─ subroutine: FIELD_SOLVE_KKXY (131/320) (field_solve_kkxy-psc.f90)
└─ call: MY_COMPLEX_SUM_VWSPEC (189/189) (field_solve_kkxy-psc.f90)
└─ program: GENE (0/334) (gene-psc.f90:21)
└─ loop: (0/334) (gene-psc.f90:147)
└─ userRegion: (203/334) (gene-psc.f90:149)
└─ call: CALC_EXPLICIT_TIMESTEP (131/131) (time_scheme-psc.f90:122)
└─ subroutine: CALC_EXPLICIT_TIMESTEP (142/145) (time_scheme-psc.f90:122)
└─ loop: (0/3) (time_scheme-psc.f90:122)
└─ call: CALC_REST (3/3) (time_scheme-psc.f90:122)

Problems Console Fortran Declaration Periscope Properties View

Name	Filename	RFL	Severity	Region	Process	Thread
Stalls due to waiting for integer loads			70.09	Group		
└─ L3 misses			70.01	Group		
└─ IA64 Pipeline Stall Cycles			65.04	Group		
└─ Stalls due to waiting for data delivery to register			62.56	Group		
└─ Stalls due to branch misprediction flush			55.45	Group		
└─ Stalls due to pipeline flush			47.89	Group		
└─ L2 misses			34.36	Group		
└─ L2 misses	aux_fields-psc.f90	42	49.20	CALL_REGION	46	0
└─ L2 misses	field_solve_kkxy-psc.f90	37	46.61	SUB_REGION	83	0

Filter: Search: RE 0 Loaded - 88 Shown - 1 Selected - Sort: [Severity (REV)]

More later at this workshop

1. Motivation
2. Performance analysis basics
3. Measurement techniques
4. Analysis techniques
5. The tools

- Callgrind / Kcachegrind
 - Cache analysis via simple cache simulation
- Periscope
 - Automatic detection of performance properties
 - Profile data
 - Online
- Scalasca
 - Call-path profiling & tracing library
 - Automatic trace analysis
- VAMPIR
 - Visual trace analysis
- IBM HPC Toolkit
 - Profiling and tracing

1. Motivation
2. Performance analysis basics
3. Measurement techniques
4. Analysis techniques
5. The tools