



Analyzing the Performance of IWAVE on a Cluster using HPCToolkit

John Mellor-Crummey and Laksono Adhianto

Department of Computer Science

Rice University

{johnmc,laksono}@rice.edu



Challenges for HPC Practitioners

- **Execution environments and applications are rapidly evolving**
 - **architecture**
 - rapidly changing multicore microprocessor designs
 - increasing scale of parallel systems
 - growing use of accelerators, e.g. GPGPU
 - **applications**
 - MPI everywhere to threaded implementations
 - adding additional scientific capabilities to existing applications
 - maintaining multiple variants or configurations for particular problems
- **Steep increase in application development effort to attain performance, evolvability, and portability**
- **Application developers need to**
 - assess weaknesses in algorithms and their implementations
 - improve scalability and performance within and across nodes
 - adapt to changes in emerging architectures
 - overhaul algorithms & data structures as needed

Challenges for HPC Practitioners

- **Execution environments and applications are rapidly evolving**
 - **architecture**
 - rapidly changing multicore microprocessor designs
 - increasing scale of parallel systems
 - growing use of accelerators, e.g. GPGPU
 - **applications**
 - MPI everywhere to threaded implementations
 - adding additional scientific capabilities to existing applications
 - maintaining multiple variants or configurations for particular problems
- **Steep increase in application development effort to attain performance, evolvability, and portability**
- **Application developers need to**
 - assess weaknesses in algorithms and their implementations
 - improve scalability and performance within and across nodes
 - adapt to changes in emerging architectures
 - overhaul algorithms & data structures as needed

Performance tools can play an important role as a guide

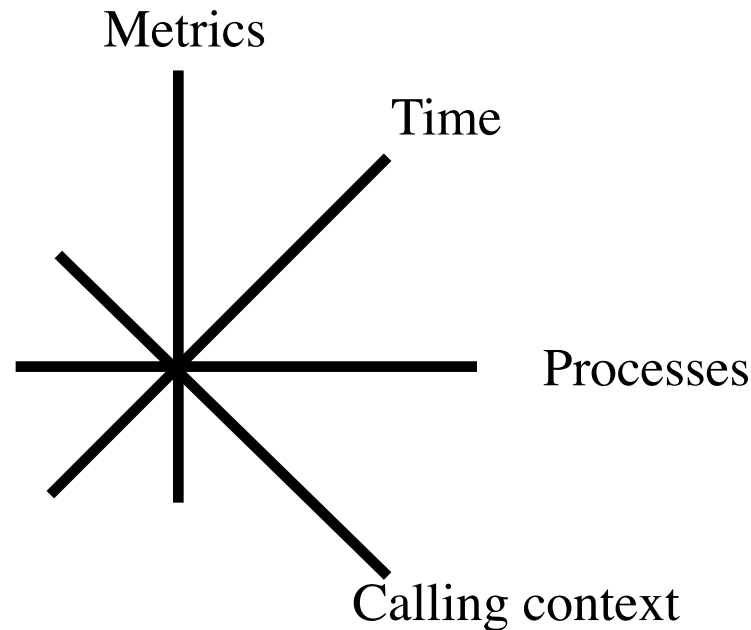
Motivation

- In December 2011, a member of CRAY Chapel team was able to achieve about 20x speedup
 - multi-threaded program compiled for a single locale
 - less than a day's work
- In January 2011, Rice Coarray Fortran team detected performance bottleneck in their HPCC FFT benchmark
 - majority of the time was spent in executing communication to perform a bit-reversal permutation
 - changing the algorithm and using coarse-grain all-to-all communication reduced the cost to only about 6%
- In December 2011, HPCToolkit team identified several performance bottlenecks in a DOD procurement benchmark
 - inefficient use of Posix I/O
 - load imbalance when not power of 2 processors
- And so on ...

HPCToolkit: Why it's Cool

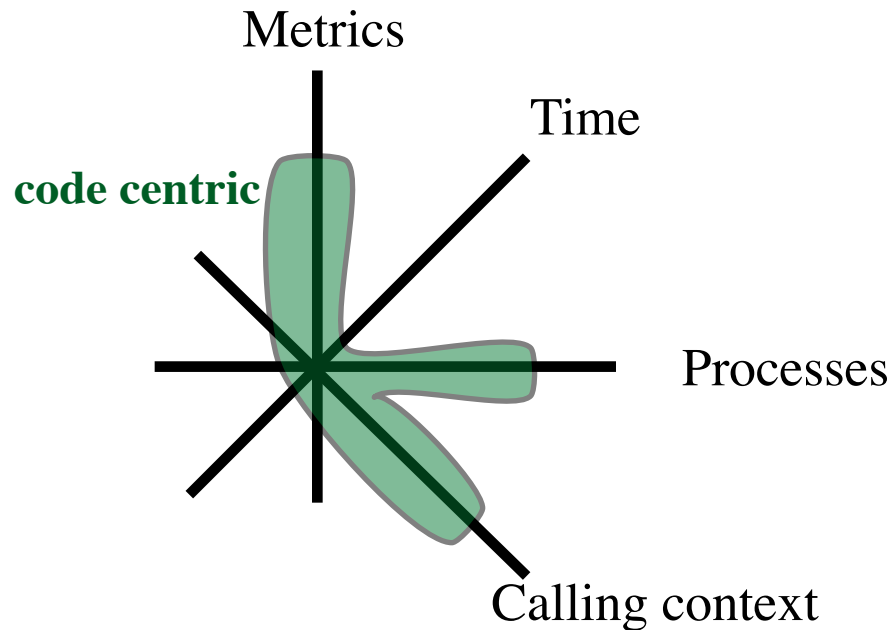
- **It runs (almost) anywhere, anytime by anyone**
 - language independent (C, C++, Fortran, ...)
 - programming model independent (MPI, OpenMP, UPC, ...)
 - operating systems independent (any Linux flavor)
 - architecture independent (x86_64, ppc64, MIPS)
 - compiler independent (Intel, PGI, GNU, Pathscale, ...)
 - runtime independent (MPICH, OpenMPI, GASNet, ...)
- **Usable on production executions**
 - low overhead: sampling rather than instrumentation
 - large number of processors
- **It's easy to use**
 - no need to rebuild code
 - work for fully optimized code
- **Effective performance analysis**
 - fine-grain attribution (lines, loops, procedures, call chains, ...)
 - correlate measurements with code for actionable results

Understanding Performance Measurements



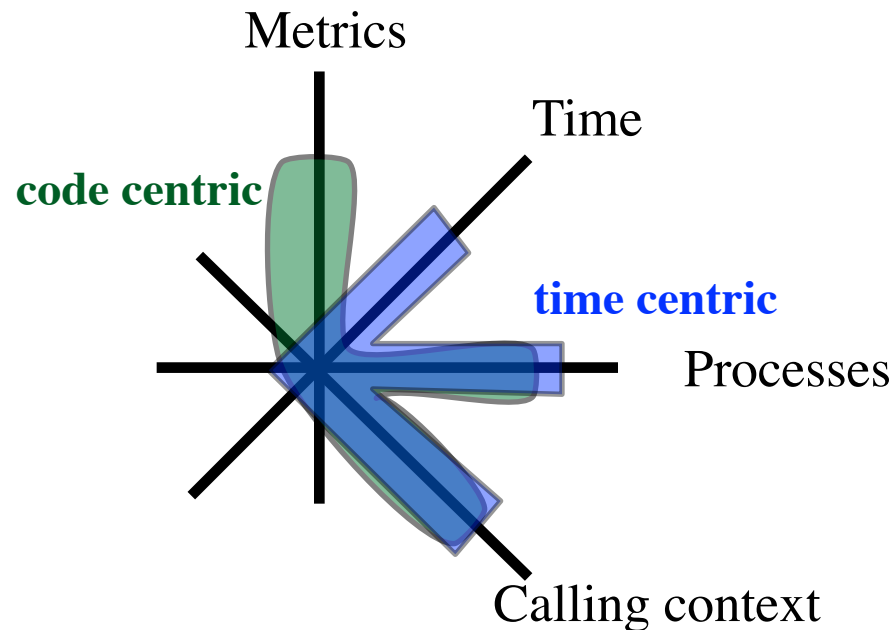
- **Four dimensions of performance data in HPCToolkit**
 - **metrics:** wallclock, L2 cache miss, cycles, flops, ...
 - **calling context:** main -> a -> b -> ...
 - **processes or ranks:** 0, 1, ..., P
 - **time:** from the beginning of measurement to the end
- **Warning: finding performance bottlenecks can be challenging**

Understanding Performance Measurements



- **Four dimensions of performance data in HPCToolkit**
 - **metrics:** wallclock, L2 cache miss, cycles, flops, ...
 - **calling context:** main -> a -> b -> ...
 - **processes or ranks:** 0, 1, ..., P
 - **time:** from the beginning of measurement to the end
- **Warning: finding performance bottlenecks can be challenging**

Understanding Performance Measurements



- **Four dimensions of performance data in HPCToolkit**
 - **metrics:** wallclock, L2 cache miss, cycles, flops, ...
 - **calling context:** main -> a -> b -> ...
 - **processes or ranks:** 0, 1, ..., P
 - **time:** from the beginning of measurement to the end
- **Warning: finding performance bottlenecks can be challenging**

Code-centric Analysis with hpcviewer

The screenshot displays the hpcviewer interface for a project named 'MOAB: mbperf_iMesh 200 B (Barcelona 2360 SE)'. The top pane shows the source code for 'mbperf_iMesh.cpp', with lines 22-28 visible. The code defines a 'SequenceCompare' class with a public operator() method that compares the end handle of one entity sequence with the start handle of another.

The bottom pane shows the 'Calling Context View' of the performance analysis. It includes a toolbar with navigation and analysis icons. The main table displays performance metrics for various scopes, including 'main', 'testB', and several nested loops and inlined functions. The table has columns for 'Scope', 'PAPI_L1_DCM (I)', 'PAPI_TOT_CYC (I)', and 'P'.

Scope	PAPI_L1_DCM (I)	PAPI_TOT_CYC (I)	P
main	8.63e+08 100 %	1.13e+11 100 %	
testB(void*, int, double const*, int const*)	8.35e+08 96.7%	1.10e+11 97.6%	
inlined from mbperf_iMesh.cpp: 261	6.81e+08 78.9%	0.98e+11 86.5%	
loop at mbperf_iMesh.cpp: 280-313	3.43e+08 39.8%	3.37e+10 29.9%	
imesh_getvtxarrcoords_	3.20e+08 37.1%	2.18e+10 19.3%	
MBCore::get_coords(unsigned long const*, int, double*) c	3.20e+08 37.1%	2.16e+10 19.1%	
loop at MBCore.cpp: 681-693	3.20e+08 37.1%	2.16e+10 19.1%	
inlined from stl_tree.h: 472	2.04e+08 23.7%	9.38e+09 8.3%	
loop at stl_tree.h: 1388	2.04e+08 23.6%	9.37e+09 8.3%	
inlined from TypeSequenceManager.hpp: 27	1.78e+08 20.6%	8.56e+09 7.6%	
TypeSequenceManager.hpp: 27	1.78e+08 20.6%	8.56e+09 7.6%	

Code-centric Analysis with hpcviewer

The screenshot displays the hpcviewer interface for a project named 'MOAB: mbperf_iMesh 200 B (Barcelona 2360 SE)'. The top pane shows the source code for 'mbperf_iMesh.cpp', with a red box highlighting the 'source pane' label. The code defines a 'SequenceCompare' class with a public method 'operator()' that returns a comparison result. The bottom pane shows the 'Calling Context View' with a red box highlighting the 'navigation pane' label. This view includes a table of performance metrics for various scopes, with a red box highlighting the 'metric pane' label. The table columns are 'Scope', 'PAPI_L1_DCM (I)', 'PAPI_TOT_CYC (I)', and 'P'. The 'Scope' column is expanded to show a call stack from 'main' down to 'TypeSequenceManager.hpp: 27'.

```
22 * Define less-than comparison for EntitySequence pointers as a comparison
23 * of the entity handles in the pointed-to EntitySequences.
24 */
25 class SequenceCompare {
26 public: bool operator() (EntitySequence* a, const EntitySequence* b ) const
27     { return a->start_handle() < b->start_handle(); }
28 };
```

Scope	PAPI_L1_DCM (I)	PAPI_TOT_CYC (I)	P
main	8.63e+08 100 %	1.13e+11 100 %	
testB(void*, int, double const*, int const*)	8.35e+08 96.7%	1.10e+11 97.6%	
inlined from mbperf_iMesh.cpp: 261	6.81e+08 78.9%	0.98e+11 86.5%	
loop at mbperf_iMesh.cpp: 280-313	3.43e+08 39.9%	4.00e+10 35.4%	
imesh_getvtxarrcoords_	3.20e+08 37.1%	2.18e+10 19.3%	
MBCore::get_coords(unsigned long const*, int, double*)	3.20e+08 37.1%	2.16e+10 19.1%	
loop at MBCore.cpp: 681-693	3.20e+08 37.1%	2.16e+10 19.1%	
inlined from stl_tree.h: 472	2.04e+08 23.7%	9.38e+09 8.3%	
loop at stl_tree.h: 1388	2.04e+08 23.6%	9.37e+09 8.3%	
inlined from TypeSequenceManager.hpp: 27	1.78e+08 20.6%	8.56e+09 7.6%	
TypeSequenceManager.hpp: 27	1.78e+08 20.6%	8.56e+09 7.6%	

Code-centric Analysis with hpcviewer

hpcviewer: MOAB: mbperf_iMesh 200 B (Barcelona 2360 SE)

```
mbperf_iMesh.cpp TypeSequenceManager.hpp stl_tree.h
```

```
22  * Define less-than comparison for EntitySequence pointers as a comparison
23  * of the entity handles in the pointed-to EntitySequences.
24  */
25  class SequenceCompare {
26  public: bool operator()( const EntitySequence* a, const EntitySequence* b ) const
27         { return a->end_handle() < b->start_handle(); }
28  };
```

Calling Context View Callers View Flat View

view control

metric display

Scope	PAPI_L1_DCM (I)	PAPI_TOT_CYC (I)	P
main	8.63e+08 100 %	1.13e+11 100 %	
testB(void*, int, double const*, int const*)	8.35e+08 96.7%	1.10e+11 97.6%	
inlined from mbperf_iMesh.cpp: 261	6.81e+08 78.9%	0.98e+11 86.5%	
loop at mbperf_iMesh.cpp: 280-313	3.43e+08 39.8%	3.37e+10 29.9%	
imesh_getvtxarrcoords_	3.20e+08 37.1%	2.18e+10 19.3%	
MBCore::get_coords(unsigned long const*, int, double*) c	3.20e+08 37.1%	2.16e+10 19.1%	
loop at MBCore.cpp: 681-693	3.20e+08 37.1%	2.16e+10 19.1%	
inlined from stl_tree.h: 472	2.04e+08 23.7%	9.38e+09 8.3%	
loop at stl_tree.h: 1388	2.04e+08 23.6%	9.37e+09 8.3%	
inlined from TypeSequenceManager.hpp: 27	1.78e+08 20.6%	8.56e+09 7.6%	
TypeSequenceManager.hpp: 27	1.78e+08 20.6%	8.56e+09 7.6%	

Code-centric Analysis with hpcviewer

hpcviewer: MOAB: mbperf_iMesh 200 B (Barcelona 2360 SE)

mbperf_iMesh.cpp TypeSequenceManager.hpp stl_tree.h

```
22 * Define less-than comparison for EntitySequence pointers as a comparison
23 * of the entity handles in the pointed-to EntitySequence
24 */
25 class SequenceCompare {
26 public: bool operator()( const EntitySequence* a, const EntitySequence* b ) const
27 { return a->end_handle() < b->start_handle();
28 };
```

costs for

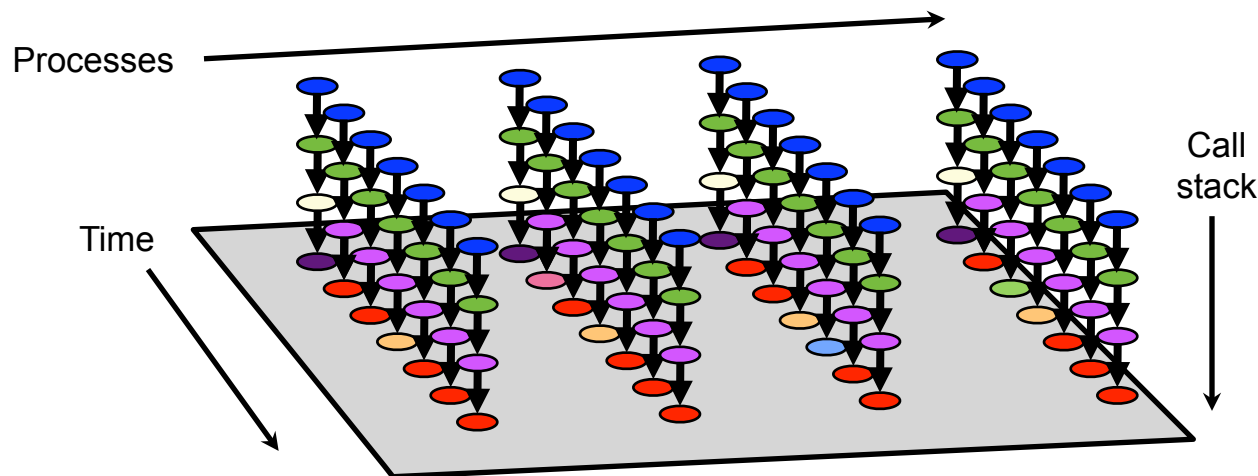
- inlined procedures
- loops
- function calls in full context

Calling Context View Callers View Flat View

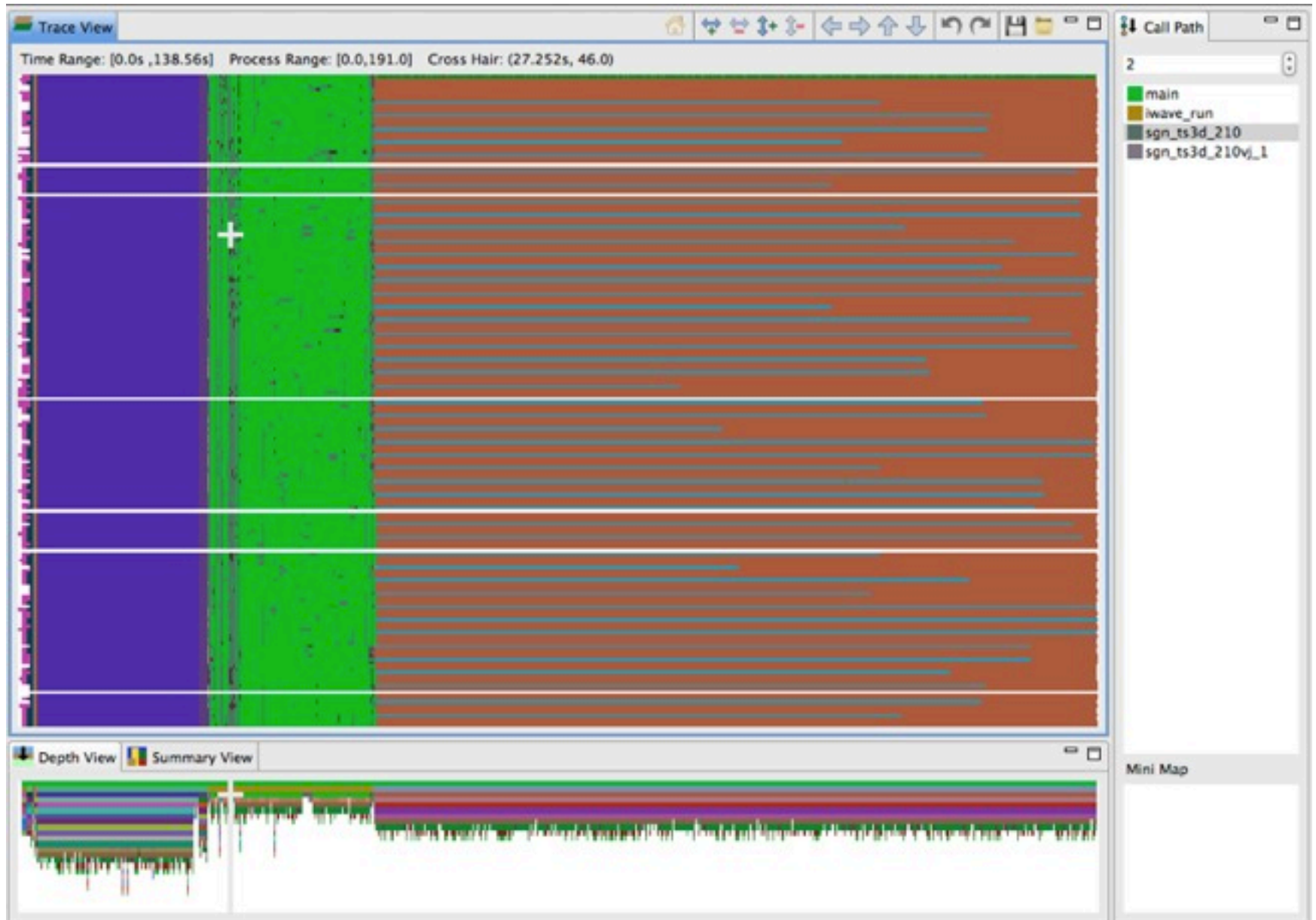
Scope	PAPI_L1_DCM (I)	PAPI_TOT_CYC (I)	P
main	8.63e+08 100 %	1.13e+11 100 %	
testB(void*, int, double const*, int const*)	8.35e+08 96.7%	1.10e+11 97.6%	
inlined from mbperf_iMesh.cpp: 261	6.81e+08 78.9%	0.98e+11 86.5%	
loop at mbperf_iMesh.cpp: 280-313	3.43e+08 39.8%	3.37e+10 29.9%	
imesh_getvtxarrcoords_	3.20e+08 37.1%	2.18e+10 19.3%	
MBCore::get_coords(unsigned long const*, int, double*)	3.20e+08 37.1%	2.16e+10 19.1%	
loop at MBCore.cpp: 681-693	3.20e+08 37.1%	2.16e+10 19.1%	
inlined from stl_tree.h: 472	2.04e+08 23.7%	9.38e+09 8.3%	
loop at stl_tree.h: 1388	2.04e+08 23.6%	9.37e+09 8.3%	
inlined from TypeSequenceManager.hpp: 27	1.78e+08 20.6%	8.56e+09 7.6%	
TypeSequenceManager.hpp: 27	1.78e+08 20.6%	8.56e+09 7.6%	

Understanding Executions over Time

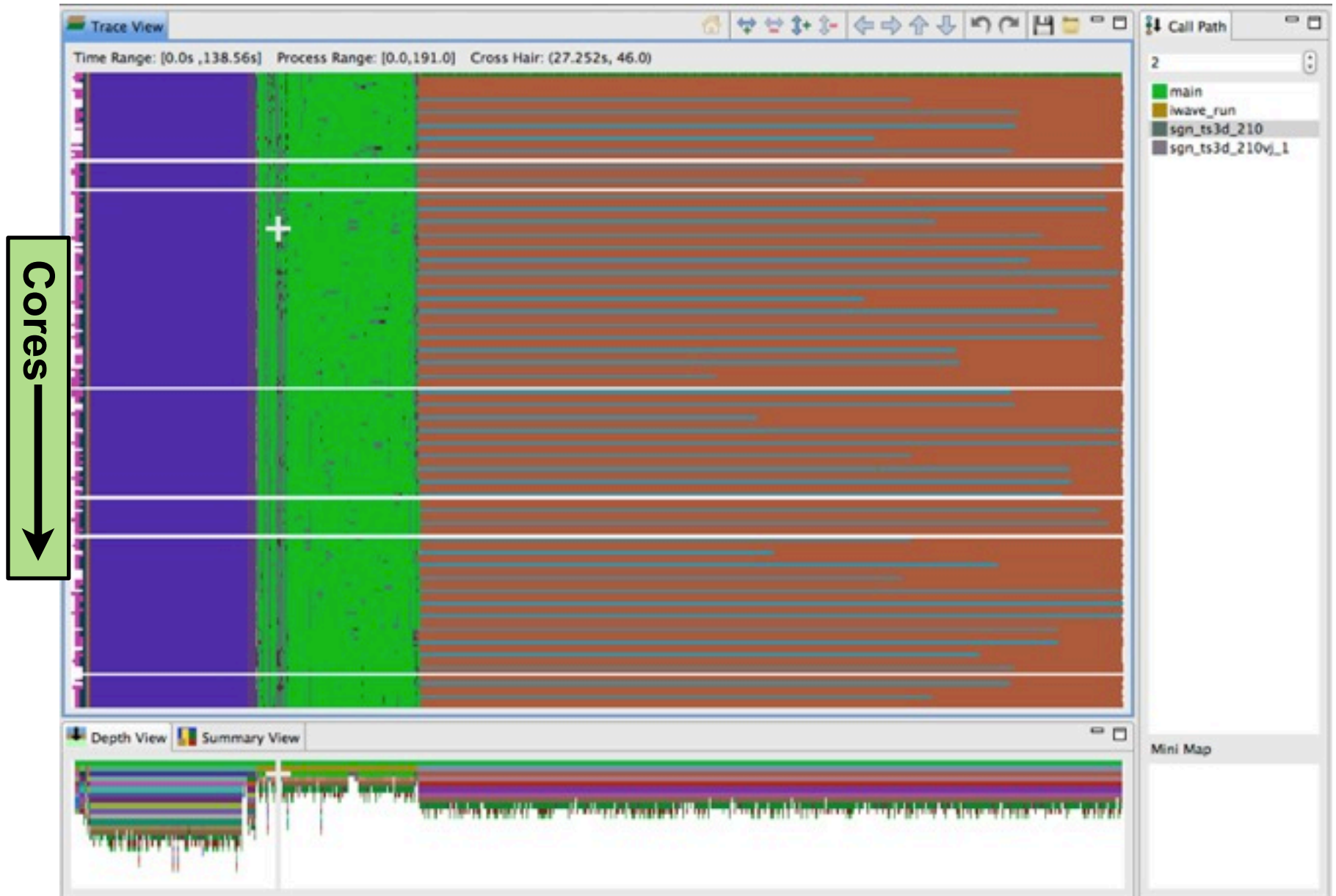
- Profiling compresses out the temporal dimension
 - temporal patterns, e.g. serialization, are invisible in profiles
- What can we do? Trace call path samples
 - sketch:
 - N times per second, take a call path sample of each thread
 - organize the samples for each thread along a time line
 - view how the execution evolves left to right
 - what do we view?
 - assign each procedure a color; view a depth slice of an execution



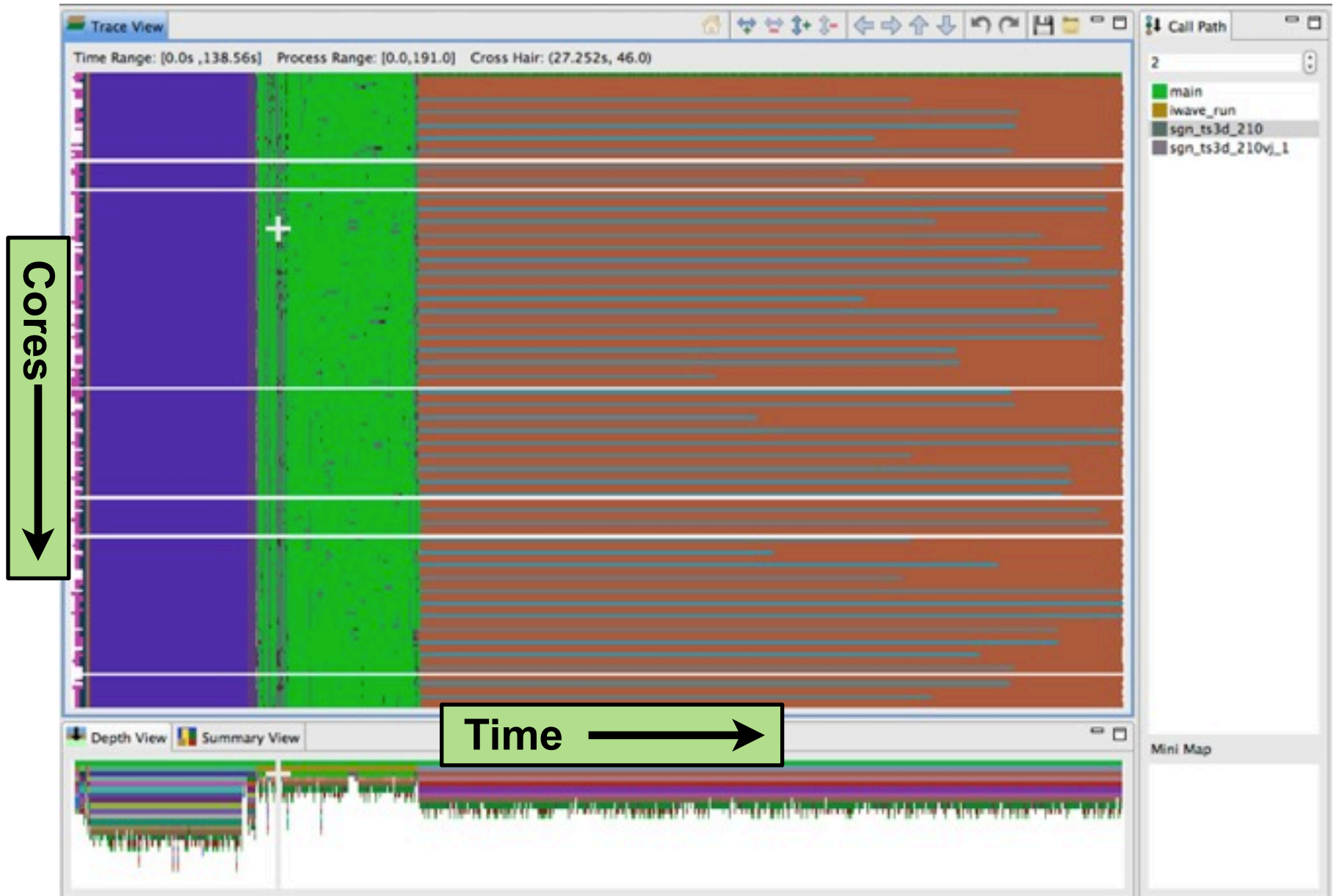
HPCToolkit's Time-centric View



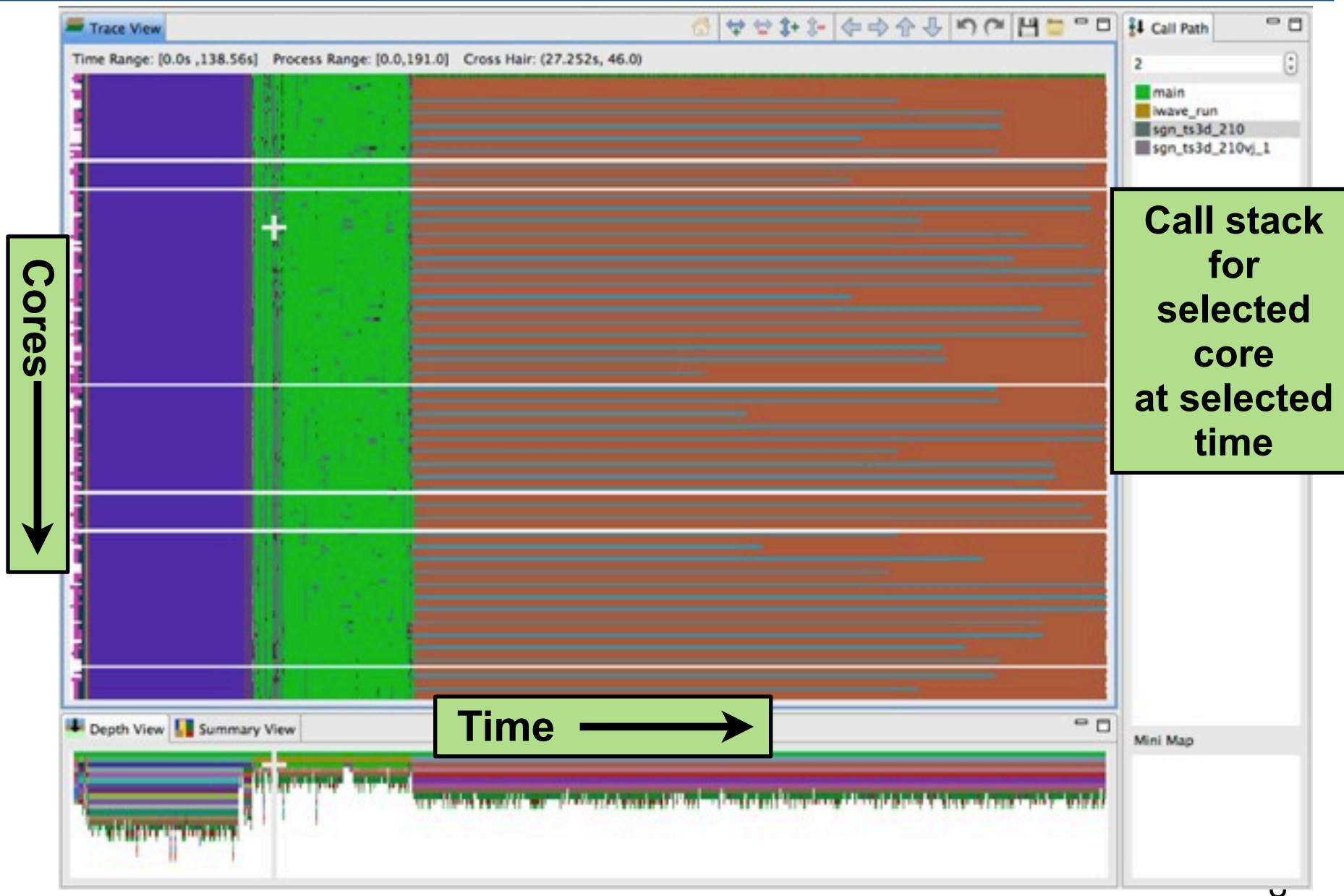
HPCToolkit's Time-centric View



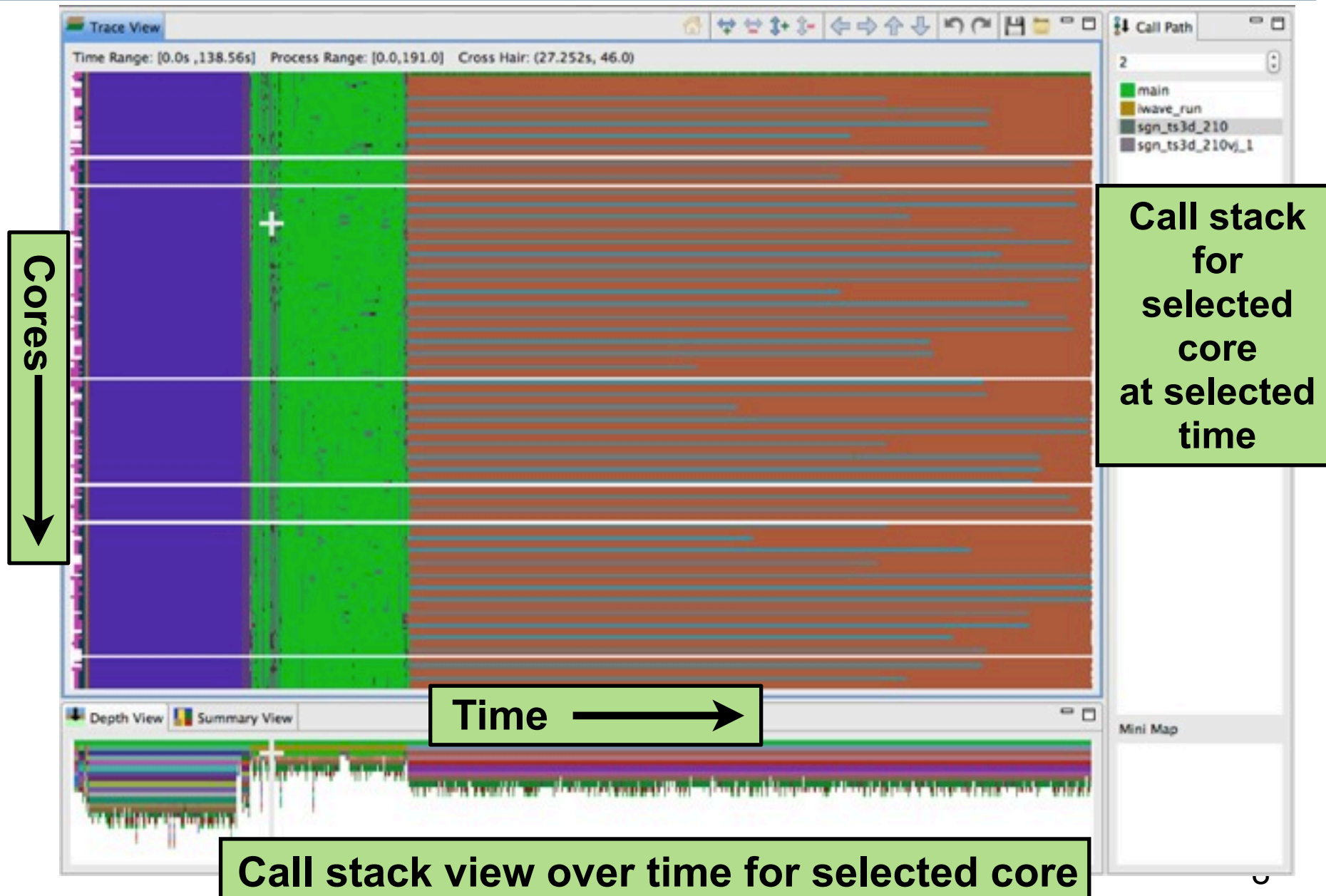
HPCToolkit's Time-centric View



HPCToolkit's Time-centric View



HPCToolkit's Time-centric View



IWAVE - Rice Inversion Project (Symes, PI)

- **Framework for finite difference simulation**
 - common services - memory, communication, I/O, job control
 - prescribed interfaces - problem description, numerical schemes
- **Applications written to the framework**
 - staggered grid acoustics with PML
 - staggered grid isotropic elasticity with PML
- **Portable - ISO C99, MPI, OpenMP**
- **Modeling engine for migration and inversion**

IWAVE on a Cluster

- **Experimental Platforms**

- **DaVinci**

- node: two 2.83 GHz Intel Westmere (6-core) processors, 48GB RAM
 - interconnect: 4x QDR Infiniband (40Gb/s)
 - GPFS fast scratch

- **STIC**

- node: two 2.4 GHz Intel Nehalem (4-core), 12GB RAM
 - interconnect: 2 4x DDR Infiniband links per node (20Gb/s each)
 - 11TB Panassas scratch

- **IWAVE configuration studied**

- **asg package**

- staggered grid finite difference (pressure, velocity) acoustic modeling

- **3D finite difference configuration**

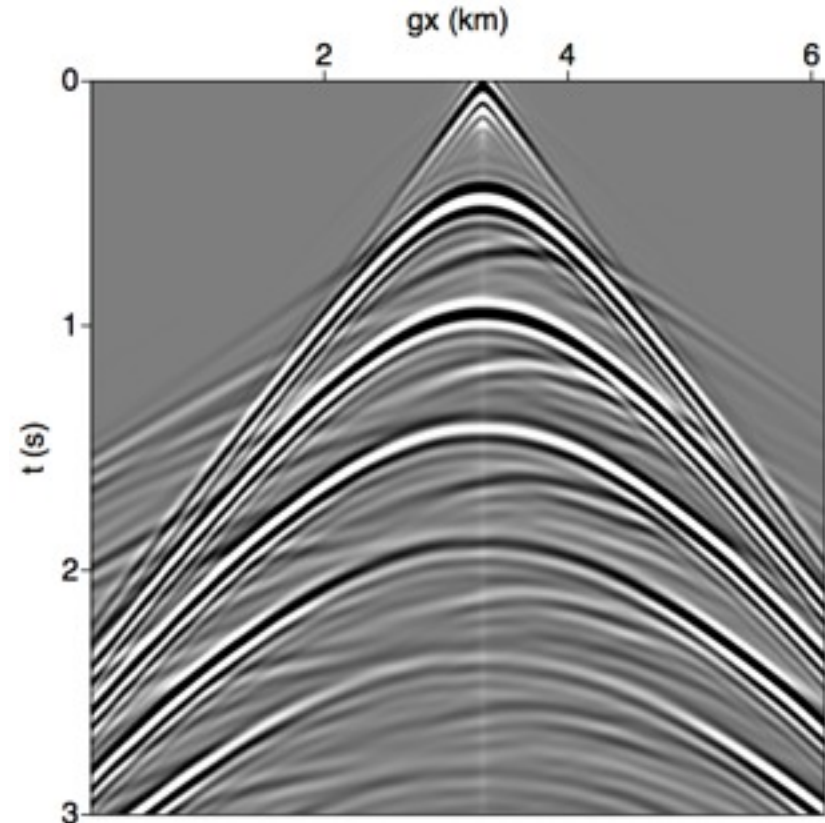
- **compiled with icc, version 12.0.0**

- **-O3 -std=c99 -g**

- **SEAM 20M GRID, FOR SHOT S020433**

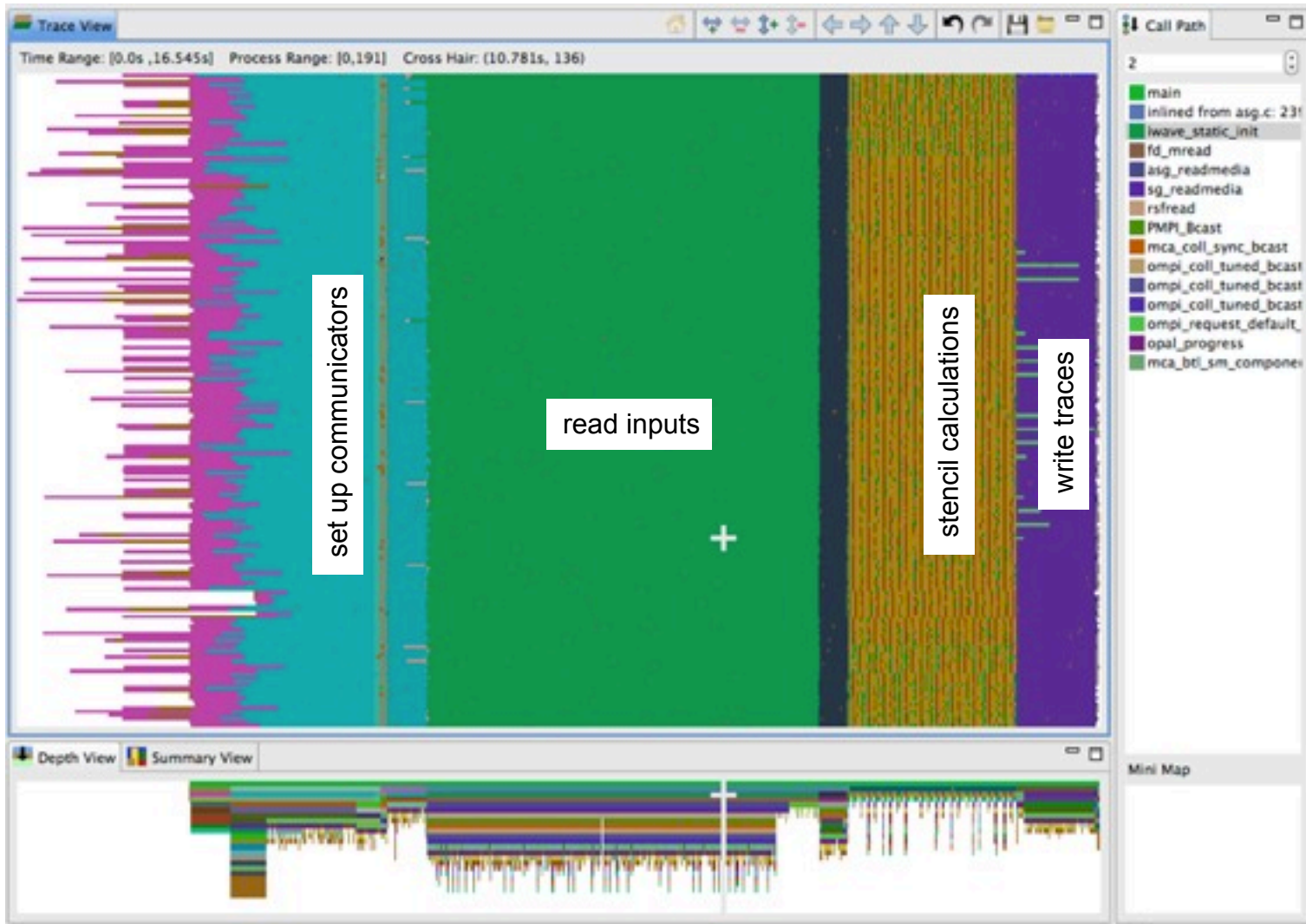
IWAVE Execution

- **8 x 6 x 4 MPI decomposition**
- **Model info**
 - **SEAMHALF20M**
 - **density info 808MB**
 - **velocity info 808MB**
- **IWAVE run**
 - **read velocity model**
 - **broadcast to all processors**
 - **read density model**
 - **broadcast to all processors**
 - **perform stencil calculations to compute pressures and velocities**
 - **write traces to disk**



Time-centric view of IWAVE

- MPI decomposition 8 x 6 x 4
- 32 nodes, 6 cores each (192 processor cores), OpenMPI



IWAVE Stencil - Overall Performance

The screenshot displays the source code for the IWAVE stencil computation in the top pane and its performance metrics in the bottom pane. The code is written in C and includes a loop over the x-axis, a stencil calculation, and updates to the x, y, and z coordinates.

```
256 #pragma ivdep
257 for ( _pxend = _px + nx; _px < _pxend; )
258 {
259     vx8 = _vx9[-1]; vx7 = _vx9[-2]; vx6 = _vx9[-3];
260     vx5 = _vx9[-4]; vx4 = _vx9[-5]; vx3 = _vx9[-6];
261     vx2 = _vx9[-7]; vx1 = _vx9[-8]; vx0 = _vx9[-9];
262
263     vx9 = *_vx9++;
264
265     delta = ((vx9 - vx0) * C5x + (vx8 - vx1) * C4x + (vx7 - vx2) * C3x + (vx6 - vx3) * C2x + (vx5 - vx4) * C1x +
266             ((*_vy9++) - (*_vy0++)) * C5y + ((*_vy8++) - (*_vy1++)) * C4y + ((*_vy7++) - (*_vy2++)) * C3y +
267             ((*_vy6++) - (*_vy3++)) * C2y + ((*_vy5++) - (*_vy4++)) * C1y +
268             ((*_vz9++) - (*_vz0++)) * C5z + ((*_vz8++) - (*_vz1++)) * C4z + ((*_vz7++) - (*_vz2++)) * C3z +
269             ((*_vz6++) - (*_vz3++)) * C2z + ((*_vz5++) - (*_vz4++)) * C1z) * (*_mpx++);
270
271     (*_px) = (*_px) * (*ratio_x++) + delta * (*recip_x++);
272     _px++;
273
274     (*_py) = (*_py) * ratio_y + delta * recip_y;
275     _py++;
276
277     (*_pz) = (*_pz) * ratio_z + delta * recip_z;
278     _pz++;
279 }
```

The performance metrics table below shows the following data:

Scope	PAPI_TOT_CYC-Sum (l)	PAPI_L2_TCM-Sum (l)	PAPI_TLB_DM-Sum (l)	PAPI_FP_INS-Sum (l)
Experiment Aggregate Metrics	1.36e+12 100 %	1.42e+10 100 %	4.10e+08 100 %	5.01e+11 100 %
main	1.36e+12 100.0	1.42e+10 100.0	4.10e+08 100 %	5.01e+11 100 %
loop at asg.c: 133	1.36e+12 100.0	1.42e+10 100.0	4.10e+08 100 %	5.01e+11 100 %
loop at asg.c: 239	1.36e+12 100.0	1.42e+10 100.0	4.10e+08 100 %	5.01e+11 100 %
loop at asg.c: 239	1.36e+12 100.0	1.42e+10 100.0	4.10e+08 100 %	5.01e+11 100 %
hwave_run	1.36e+12 100.0	1.42e+10 99.9%	4.10e+08 99.9%	5.01e+11 100.0
sgn_ts3d_210	8.83e+11 64.7%	1.20e+10 84.3%	2.15e+08 52.4%	5.00e+11 99.8%
sgn_ts3d_210p012	3.66e+11 26.8%	4.87e+09 34.2%	9.54e+07 23.3%	1.82e+11 36.2%
loop at sgn210_3d.c: 242	3.65e+11 26.8%	4.86e+09 34.1%	9.50e+07 23.2%	1.82e+11 36.2%
loop at sgn210_3d.c: 242	3.65e+11 26.8%	4.86e+09 34.1%	9.50e+07 23.2%	1.82e+11 36.2%
loop at sgn210_3d.c: 242	3.55e+11 26.0%	4.77e+09 33.4%	9.50e+07 23.2%	1.81e+11 36.0%
sgn210_3d.c:...	6.69e+10 4.9%	7.69e+08 5.4%	1.28e+07 3.1%	4.13e+10 8.2%
sgn210_3d.c:...	5.36e+10 3.9%	9.42e+08 6.6%	3.20e+06 0.8%	2.55e+10 5.1%

.51 FLOP per cycle

IWAVE Stencil - Why Low Performance?

- Look at LLC misses to see demand fetch from memory
- Survey resource stalls from any source

The screenshot displays the hpcviewer interface for the application 'asg.x'. The top pane shows the source code for 'sgn210_3d.c', with lines 237-261 visible. The code includes calculations for ratios and reciprocals, and nested loops for z, y, and x dimensions. The bottom pane shows the 'Calling Context View' with a table of performance metrics. A red box highlights the row for 'loop at sgn210_3d.c: 257'.

```
237
238 ratio_x[ix] = (1.0 - etaxdt) / (1.0 + etaxdt);
239 recip_x[ix] = 1.0 / (1.0 + etaxdt);
240 }
241
242 for ( iz = tid; iz < nz; iz += tsz )
243 {
244   etazdt = (*_epz) * dt2;
245
246   ratio_z = (1.0 - etazdt) / (1.0 + etazdt);
247   recip_z = 1.0 / (1.0 + etazdt);
248
249   for ( iy = 0; iy < ny; ++iy )
250   {
251     etaydt = (*_epy++) * dt2;
252
253     ratio_y = (1.0 - etaydt) / (1.0 + etaydt);
254     recip_y = 1.0 / (1.0 + etaydt);
255
256 #pragma ivdep
257   for ( _pxend = _px + nx; _px < _pxend; )
258   {
259     vx8 = _vx9[-1]; vx7 = _vx9[-2]; vx6 = _vx9[-3];
260     vx5 = _vx9[-4]; vx4 = _vx9[-5]; vx3 = _vx9[-6];
261     vv7 = _vv9[-7]; vv1 = _vv9[-8]; vv8 = _vv9[-9];
```

Scope	RAPL_TOT_CYC:Sum (l)	MEM_LOAD_RETIRED-LLC_MISS:Sum (l)	RESOURCE_STALLS-ANY:Sum (l)
Experiment Aggregate Metrics	1.38e+12 100 %	1.04e+09 100 %	7.64e+11 100 %
main	1.38e+12 100.0	1.04e+09 100.0	7.64e+11 100.0
loop at asg.c: 133	1.38e+12 100.0	1.04e+09 100.0	7.64e+11 100.0
loop at asg.c: 239	1.38e+12 100.0	1.04e+09 100.0	7.64e+11 100.0
loop at asg.c: 239	1.38e+12 100.0	1.04e+09 100.0	7.64e+11 100.0
loop at sgn210_3d.c: 242	1.38e+12 100.0	1.04e+09 99.8%	7.64e+11 100.0
loop at sgn210_3d.c: 249	8.83e+11 63.8%	6.74e+08 64.8%	4.82e+11 63.1%
loop at sgn210_3d.c: 257	3.66e+11 26.5%	2.09e+08 20.1%	2.32e+11 30.4%
loop at sgn210_3d.c: 268	3.46e+11 24.4%	2.09e+08 20.1%	2.32e+11 30.4%
loop at sgn210_3d.c: 268	3.46e+11 24.4%	2.09e+08 20.1%	2.32e+11 30.4%

- LLC misses 3 orders of magnitude lower than cycles
- Resource stalls on par with total cycles

IWAVE - Looking at Resource Stall Causes

Dominant resource stalls

- LOADS
- CPU reservation station full
 - can't issue instructions until operands available

```
250 {
251   etaydt = (*_epy++) * dt2;
252
253   ratio_y = (1.0 - etaydt) / (1.0 + etaydt);
254   recip_y = 1.0 / (1.0 + etaydt);
255
256 #pragma ivdep
257   for ( _pxend = _px + nx; _px < _pxend; )
258   {
259     vx8 = _vx9[-1]; vx7 = _vx9[-2]; vx6 = _vx9[-3];
260     vx5 = _vx9[-4]; vx4 = _vx9[-5]; vx3 = _vx9[-6];
261     vx2 = _vx9[-7]; vx1 = _vx9[-8]; vx0 = _vx9[-9];
262
263     vx9 = *_vx9++;
264
265     delta = ((vx9 - vx0) * C5x + (vx8 - vx1) * C4x + (vx7 - vx2) * C3x + (vx6 - vx3) *
266             ((*_vy9++) - (*_vy0++)) * C5y + ((*_vy8++) - (*_vy1++)) * C4y + ((*_vy7++) - (*_vy2++)) * C3y +
267             ((*_vy6++) - (*_vy3++)) * C2y + ((*_vy5++) - (*_vy4++)) * C1y +
268             ((*_vz9++) - (*_vz0++)) * C5z + ((*_vz8++) - (*_vz1++)) * C4z + ((*_vz7++) - (*_vz2++)) * C3z +
269             ((*_vz6++) - (*_vz3++)) * C2z + ((*_vz5++) - (*_vz4++)) * C1z) * (*_mpx++);
270
271     (*_px) = (*_px) * (*ratio_x++) + delta * (*recip_x++);
272     _px++;

```

Scope	RESOURCE_STALLS-LOAD-Sum (l)	RESOURCE_STALLS-STORE-Sum (l)	RESOURCE_STALLS-RS_FULL-Sum (l)	RESOURCE_STALLS-ROB_FULL-Sum (l)
Experiment Aggregate Metrics	1.76e+11 100 %	6.46e+10 100 %	3.81e+11 100 %	1.56e+11 100 %
main	1.76e+11 100.0	6.45e+10 100.0	3.81e+11 100.0	1.56e+11 100.0
loop at asg.c: 133	1.76e+11 100.0	6.45e+10 100.0	3.81e+11 100.0	1.56e+11 100.0
loop at asg.c: 239	1.76e+11 100.0	6.45e+10 100.0	3.81e+11 100.0	1.56e+11 100.0
loop at asg.c: 239	1.76e+11 100.0	6.45e+10 100.0	3.81e+11 100.0	1.56e+11 100.0
@iwave_run	1.76e+11 100.0	6.45e+10 99.9%	3.81e+11 100.0	1.56e+11 99.9%
@sgn_ts3d_210	1.75e+11 99.4%	3.16e+09 4.9%	2.17e+11 56.8%	1.14e+11 73.3%
@sgn_ts3d_210p012	1.35e+11 76.7%	3.12e+09 4.8%	1.07e+11 28.0%	9.50e+07 0.1%
loop at sgn210_3d.c: 242	1.35e+11 76.7%	3.12e+09 4.8%	1.07e+11 28.0%	9.50e+07 0.1%
loop at sgn210_3d.c: 268	1.35e+11 76.7%	3.11e+09 4.8%	1.07e+11 28.0%	9.50e+07 0.1%
loop at sgn210_3d.c: 268	1.35e+11 76.6%	1.79e+09 2.8%	1.05e+11 27.6%	9.00e+07 0.1%
sgn210_3d.c: 268	3.69e+10 21.0%		1.45e+10 3.8%	4.00e+07 0.0%

IWAVE - Looking at Memory System Usage

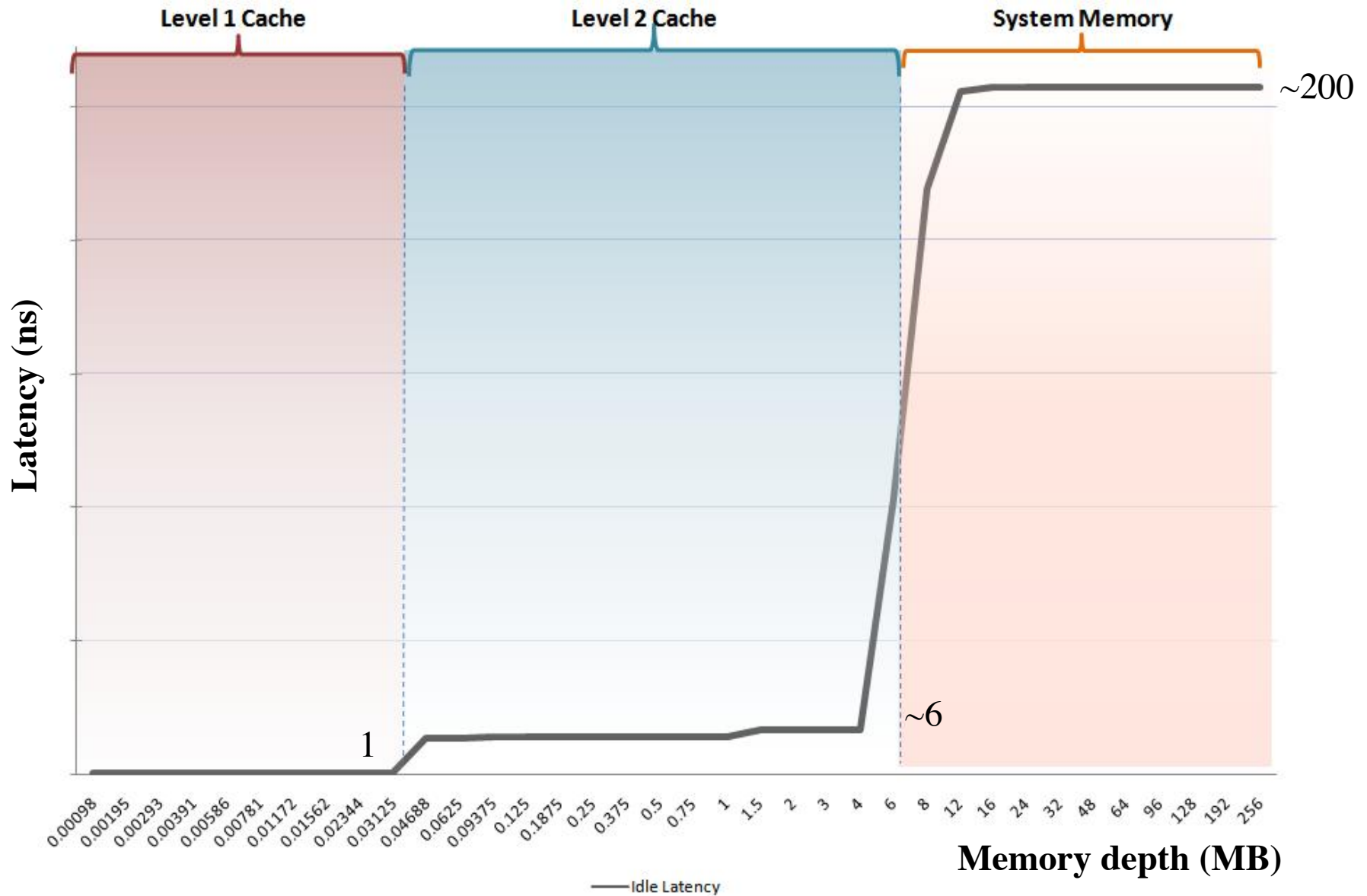
Analyze where the loads go

- L2 Hit - 1.51×10^{10}
- L3 Hit - 1.08×10^9
- Memory - 2.07×10^8

The screenshot shows the hpcviewer interface with the source code for iwave.c and a performance table. The code includes nested loops for iy and ix, with vector operations and a complex delta calculation. The performance table below highlights the 'loop at sgn210_3d.c: 249' with a red box, showing its contribution to the total memory system usage.

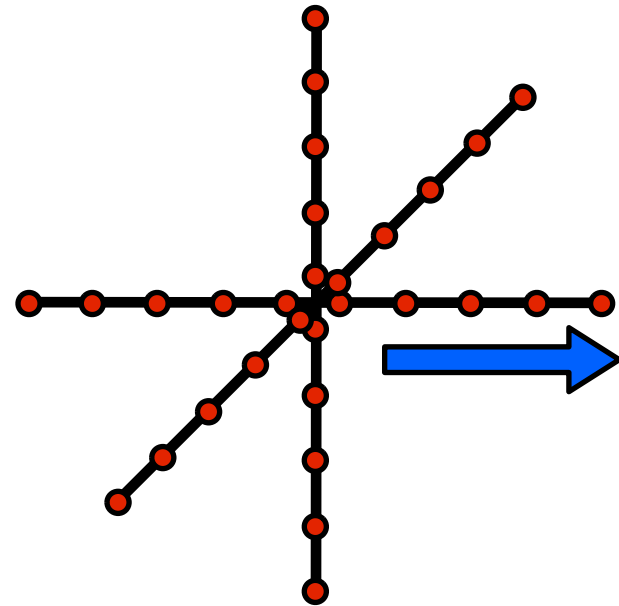
Scope	PAPI_TOT_CYC Sum (I)	MEM_LOAD_RETIRED:LLC_MISS Sum (I)	MEM_LOAD_RETIRED:L2_HIT Sum (I)	MEM_LOAD_RETIRED:LLC_UNSHARED_HIT Sum (I)
loop at asg.c: 239	1.85e+12 100.0	1.04e+09 100.0	4.64e+10 100 %	3.05e+09 100.0
loop at asg.c: 239	1.85e+12 100.0	1.04e+09 100.0	4.64e+10 100 %	3.05e+09 100.0
iwave_run	1.85e+12 99.9%	1.03e+09 99.8%	4.64e+10 100 %	3.05e+09 100.0
sgn_ts3d_210	8.86e+11 47.9%	6.66e+08 64.3%	1.92e+10 41.4%	2.82e+09 92.5%
sgn_ts3d_210p012	3.69e+11 19.9%	2.08e+08 20.1%	1.51e+10 32.4%	1.08e+09 35.4%
loop at sgn210_3d.c: 242	3.69e+11 19.9%	2.07e+08 20.0%	1.51e+10 32.4%	1.08e+09 35.4%
loop at sgn210_3d.c: 249	3.69e+11 19.9%	2.07e+08 20.0%	1.51e+10 32.4%	1.08e+09 35.4%
sgn210_3d.c: 244	4.00e+07 0.0%			
sgn210_3d.c: 247	3.40e+07 0.0%			
sgn210_3d.c: 246	4.00e+06 0.0%	4.00e+04 0.0%		
free	4.60e+07 0.0%	1.20e+05 0.0%		
loop at sgn210_3d.c: 152	2.00e+07 0.0%	8.00e+04 0.0%		
malloc	1.80e+07 0.0%	4.00e+04 0.0%		

Memory Latency on Intel 5100 MCH



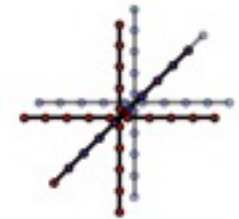
Principal Stencil Pattern

- Execution under study
 - `sgn_ts3d_210p012`
 - 10 points along each coordinate axis
 - sweep through memory along the X coordinate dimension

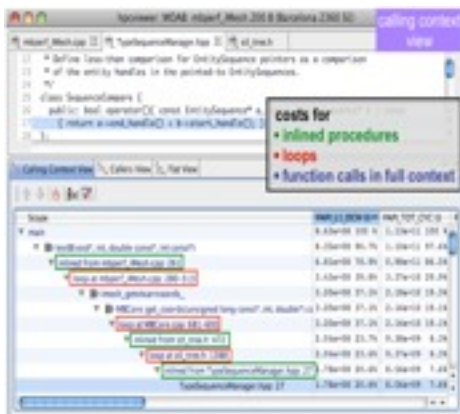


IWAVE Tuning Recommendations

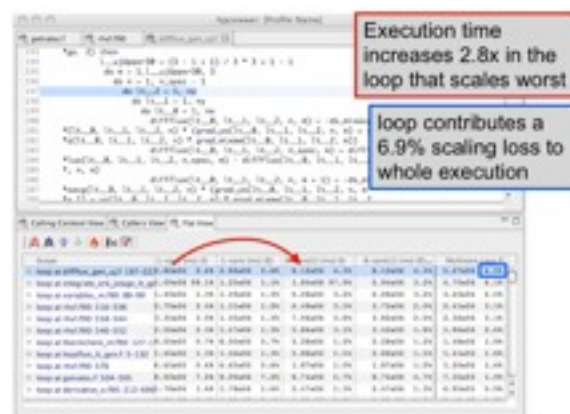
- **Computation vs. communication**
 - communication for the example studied is ~27% of `iwave_run`
 - compute on more data per core for higher parallel efficiency
 - no communication/computation overlap
- **I/O**
 - IWAVE uses serial Posix I/O for its input
 - using HDF5 and parallel I/O would be a higher performance choice
- **Stencil calculations**
 - IWAVE's stencil calculations are latency bound
 - spend most of their time waiting for data from L2 cache
 - need to make better use of the memory hierarchy
 - unrolling once in Y and Z coordinate dimensions will reuse data values immediately
 - currently, temporal reuse along Y and Z axis is long distance
 - unrolling in Y and Z: immediately reuse 9 of every 10 values loaded
 - pointer-based data access inhibits compiler-based tiling
 - tiling along Y and Z will be important for good cache reuse with large data



HPCToolkit Capabilities at a Glance



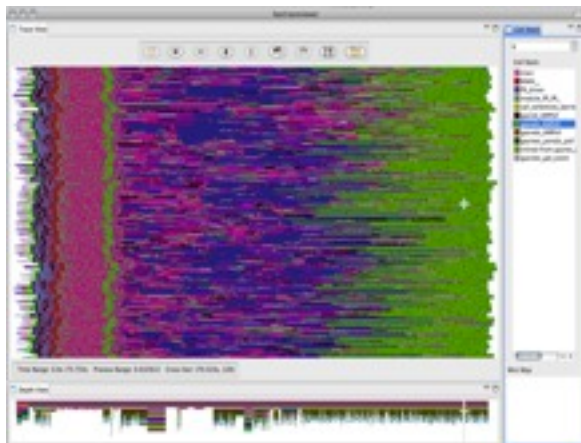
Attribute Costs to Code



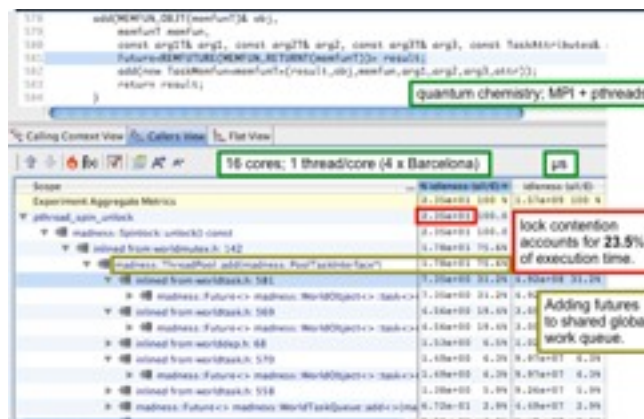
Pinpoint & Quantify Scaling Bottlenecks



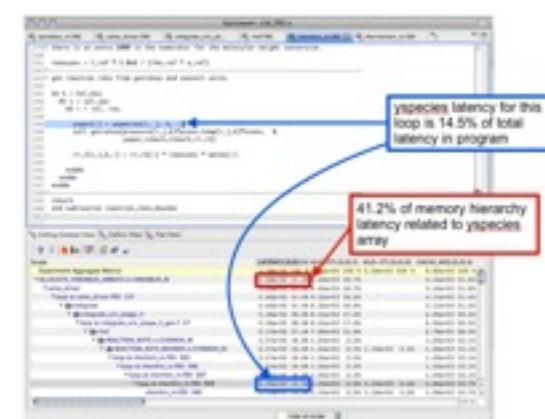
Assess Imbalance and Variability



Analyze Behavior over Time



Shift Blame from Symptoms to Causes



Associate Costs with Data

hpctoolkit.org



HPCToolkit Status

- **Operational today on**
 - 64- and 32-bit x86 systems running Linux (including Cray XT/E/K)
 - IBM Blue Gene/P
 - IBM Power7 systems running Linux
- **Emerging capabilities**
 - IBM Blue Gene/Q
 - NVIDIA GPU
 - measurement and reporting using GPU hardware counters
 - data centric analysis
- **Available as open source software at <http://hpctoolkit.org>**

Ongoing Work

- **Homogeneous nodes**
 - measurement and analysis for massive numbers of threads
 - “blame shifting” to pinpoint and quantify causes of idleness in OpenMP programs
- **Heterogeneous nodes**
 - “blame shifting” to pinpoint and quantify causes of CPU and GPU idleness in hybrid programs
 - derived metrics for GPU
- **Bandwidth monitoring of communication and I/O**
- **Future enhancements**
 - support for Intel MIC
 - provide higher-level prescriptive feedback

References

- **HPCToolkit Project: <http://hpctoolkit.org>**
- **David Levinthal. Performance Analysis Guide for Intel® Core™ i7 Processor and Intel® Xeon™ 5500 processors, Version 1.0, http://software.intel.com/sites/products/collateral/hpc/vtune/performance_analysis_guide.pdf**