Missing Pieces in the OpenMP Ecosystem

John Mellor-Crummey

Department of Computer Science
Rice University

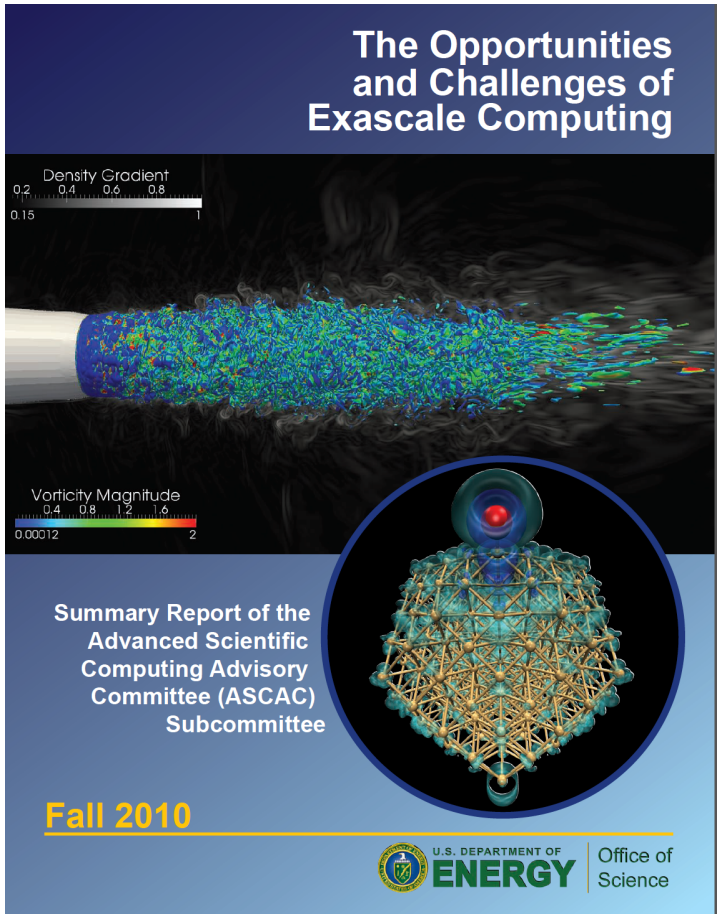
johnmc@rice.edu

Outline

- Forthcoming supercomputers in the US
- The missing performance tool interface in OpenMP
- Hpctoolkit: a sampling-based performance tool using OMPT
  - attributing performance to highly-optimized source
  - understanding scaling losses with threading
  - blame shifting to pinpoint causes of performance losses
  - OMPT TARGET
  - case study: commercial DRTM code
  - data-centric analysis to understand locality issues
- The missing support for data layout in OpenMP
  - data layout today in OpenMP
  - lessons from the past
  - key elements of a good solution
- Looking ahead

Planning for Exascale

- Enable
  - solution of vastly more accurate predictive models
  - analysis of massive quantities of data
- Quantum advances in areas of science and technology
  - adaptation to regional climate changes
  - carbon footprint for transportation
  - efficiency and safety of nuclear energy
  - innovative designs for cost-effective renewable energy resources
  - design of experimental facilities
  - understanding of fission and fusion
  - reverse engineering of the human brain
  - design, control and manufacture of advanced materials



2015-2016 DOE Systems

	Cori	Trinity	Theta
Laboratory	LBNL	LANL/SNL	ANL
Compute	9300+ Xeon Phi (KNL)	9500+ Xeon Phi (KNL)	2500 Xeon Phi (KNL)
Data Partition	1630 nodes Dual 16-core Haswell	9436 nodes Dual 16-core Haswell	—
Performance (RPEAK)	~30 PF	~40 PF	8.5 PF
System Interconnect	Cray Aries Dragonfly	Cray Aries Dragonfly	Cray Aries Dragonfly
Memory	96GB DDR4 16GB MCDRAM	96GB DDR4 16GB MCDRAM	96GB DDR4 16GB MCDRAM
Peak Power	10 MW	13 MW	1.7MW

DOE CORAL

- Collaboration of Oak Ridge, Argonne, and Livermore
  - joint procurement of leadership computer systems
- Goal
  - streamline procurement processes and reduce costs to develop supercomputers that will be five to seven times more powerful when fully deployed than today's fastest systems in the U.S.
- Procurements announced
  - Nov 2014
    - $325 million: two state-of-the-art supercomputers to ORNL, LLNL
  - April 2015
    - $200 million: next-generation supercomputer ALCF
- General availability expected in 2018

2018 DOE CORAL Systems

	Aurora	Summit	Sierra
Laboratory	ANL	ORNL	LLNL
Nodes	>50,000	~3400	N/A
Processor	Xeon Phi (KNH)	Multiple Power9	Multiple Power9
GPU	-	> 1 NVIDIA Volta	> 1 NVIDIA Volta
Performance (RPEAK)	180 PF - 450 PF	150 - 300 PF	100+ PF
System Interconnect	Intel Omni-Path-2 w/ silicon photonics	Mellanox dual rail EDR-IB (23 GB/s)	Mellanox dual rail EDR-IB (23 GB/s)
Memory	> 7PB (HBM+ local, NV memory)	> 512GB/node (HBM+DDR4)	> 512GB/node (HBM+DDR4)
Peak Power	13 MW	10 MW	N/A
Vendor	Cray	IBM	IBM

National Strategic Computing Initiative

Executive Order, July 29, 2015

United States Government must create a coordinated federal strategy for HPC research, development, and deployment

Strategic Objectives

- **Accelerating delivery of a capable exascale computing system**
- **Integrating technology for modeling and simulation with that for data analytic computing**
- **Establishing, over the next 15 years, a viable path forward for future HPC systems for the "post- Moore's Law era"**
- **Increasing the capacity and capability of an enduring national HPC ecosystem by employing a holistic approach**
- **Developing an enduring public-private collaboration to share benefits of HPC R&D across government, industry, and academia**

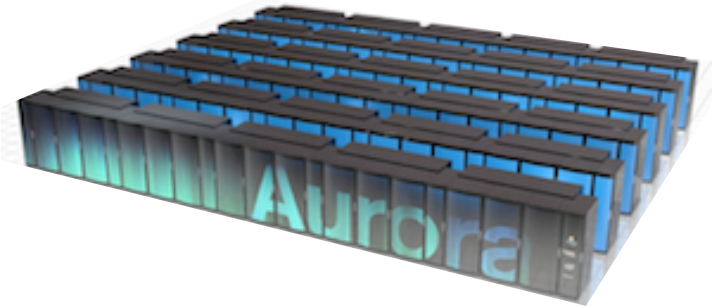
Barak Obama

Programming Models for Emerging Systems

MPI + X

- Self-hosted, manycore Intel Xeon Phi (KNL)
  - — OpenMP
- Accelerated IBM Power9 + NVIDIA Volta,
  - — ORNL recommends OpenMP 4.0 and OpenACC over low-level frameworks such as CUDA or OpenCL
  - — LLNL recommends RAJA Portability Layer
  - — SNL recommends Kokkos library

- Self-hosted, manycore Intel Xeon Phi (KNH)
  - — ANL recommends OpenMP 4.x



LLNL: RAJA Portability Layer

Goal: localize platform-specific code changes by decoupling loop body and traversal

C-style for-loop

```
double* x ; double*y ;
double a ;
// …
for ( int i = begin; i < end; ++i )  {
    y[ i ] += a * x[ i ] ;
}
```

RAJA-style loop

```
Real_ptr x;   Real_ptr y ;
Real_type a ;
// …
forall< exec_policy >( IndexSet, [&] (Index_type i)  {
    y[ i ] += a * x[ i ] ;
}  );
```

- **Data type encapsulation** hides non-portable compiler directives, data attributes, etc. (not required to use RAJA, but a good idea in general)
- **Traversal templates** encapsulate platform-specific scheduling & execution
- **Index sets** encapsulate loop iteration patterns & data placement
- **C++ lambda functions** enable decoupling (crucial for adoption of RAJA)

Important: Loop body is the same. Transformations can be adopted incrementally and each part can be specialized for a particular code.

Slide credit: Rich Hornung, Jeff Keasler. RAJA Portability Layer, July 25, 2014.

SNL: Kokkos Thread-parallel Programming Model

- Strategy for performance portability
  - map computations to threads: lambda, parallel patterns
  - map data to memory: abstract layouts for multidimensional arrays
  - access data through special hardware: e.g., atomics, texture cache
- Abstractions: spaces, policies, patterns
  - memory space: where data resides (perf, capacity, bandwidth)
  - execution space: encapsulates HW resources (cores, GPU, …)
  - execution policy: how (and where) a user function is executed
  - pattern: parallel_for, parallel_reduce, parallel_scan, task-dag, …
- Composition: parallel_pattern(Policy<Space>, Function)
- Extensible spaces, policies, and patterns

Challenges for Computational Scientists

- Rapidly evolving platforms and applications
  - node architecture
    - latency-optimized multicore, e.g. IBM Power
    - throughput-optimized manycore, e.g., Intel Xeon Phi
    - growing presence of accelerators, e.g., NVIDIA GPU
    - increasing scale of thread-level parallelism on nodes
  - applications
    - add threading to MPI everywhere implementations
    - enhance vector parallelism
    - refine predictive models and data analysis techniques
- Computational scientists need to
  - adapt to changes in emerging architectures
  - improve scalability within and across nodes
  - assess weaknesses in algorithms and their implementations

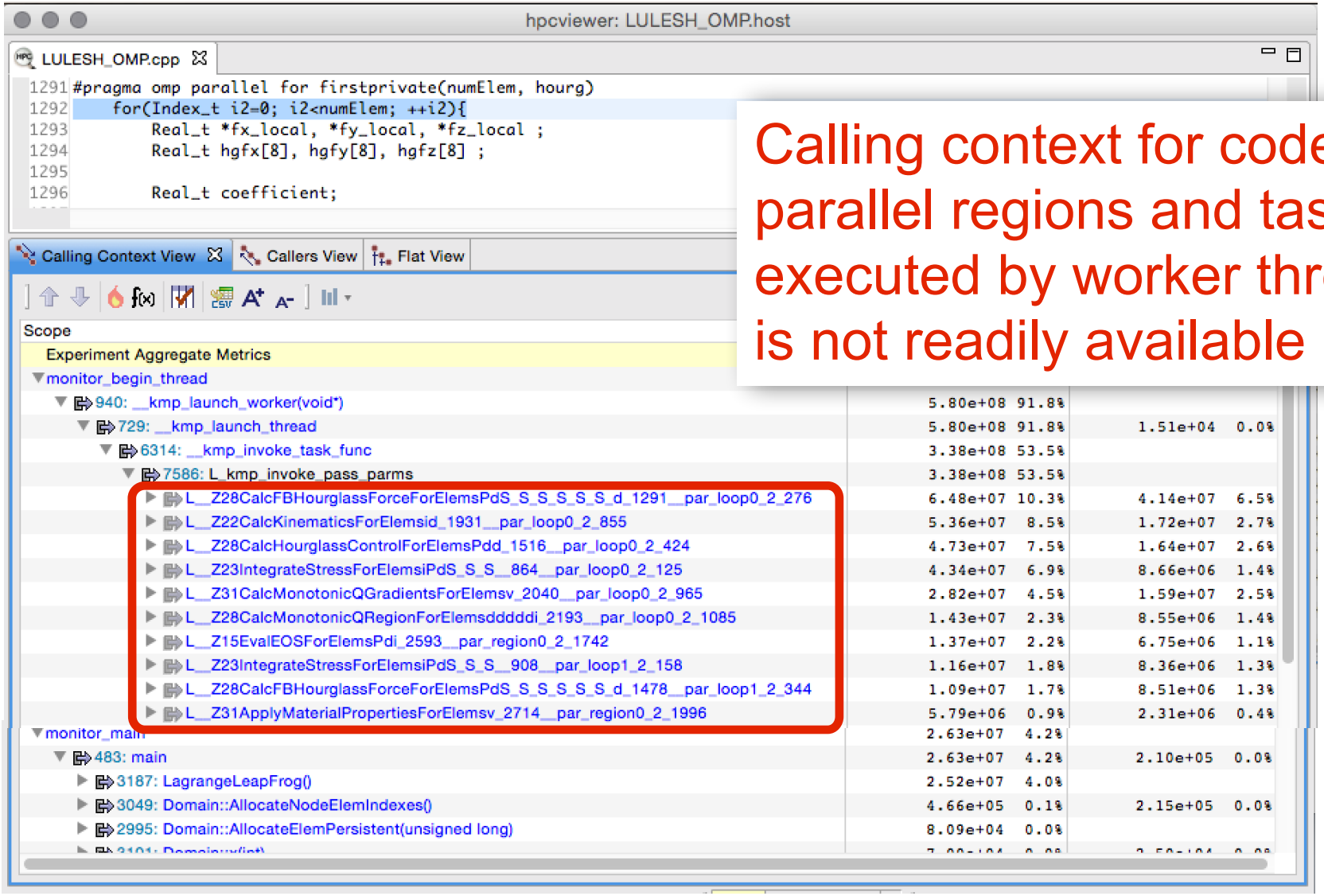
Multiplicity of Programming Environments

- Cray KNL-based manycore platforms
  - Intel, Cray, and GCC programming environments
- IBM Power9 + NVidia Volta systems
  - PGI, GCC, IBM XL, LLVM
- Goal: tool suite usable with all OpenMP implementations

Missing Piece: OpenMP Interfaces for Tools

Challenge for OpenMP Tools

- **Typically, large gap between OpenMP source and implementation**



Calling context for code in parallel regions and tasks executed by worker threads is not readily available

Obstacles for Tools

- Differences in OpenMP implementations
  - static vs. dynamic linking
    - Oracle's collector interface for tools supports only dynamic linking
    - static linking is still preferred for today's supercomputers
  - threads
    - Intel: extra *shepherd* thread
    - IBM: none
  - call stack
    - GOMP: master calls outlined function from user code
    - Intel and IBM: master calls outlined function from runtime
    - PGI: cactus stack
- No standard API for runtime inquiry

OMPT Design Objectives

- Enable tools to gather performance information and associate costs with application source and runtime system
  - construct low-overhead tools based on asynchronous sampling
  - identify application stack frames vs. runtime frames
  - associate a thread's activity at any point with a descriptive state
    - parallel work, idle, lock wait, ...
- Negligible overhead if OMPT interface is not in use
- Define support for trace-based performance tools
- Don't impose an unreasonable development burden
  - runtime implementers
  - tool developers

Major OMPT Functionality

- State tracking
  - threads maintain state at all times (e.g., working, waiting, idle)
  - a tool can query this state at any time (async signal safe)
- Call stack interpretation
  - inquiry functions enable tools to reconstruct application-level call stacks from implementation-level information
    - identify which frames on the call stack belong to the runtime system
- Event notification callbacks for predefined events
  - mandatory callbacks for threads, parallel regions, and tasks
  - optional callbacks for identifying idleness and attributing blame
  - optional callbacks for tracing activity for all OpenMP constructs
- Target device monitoring
  - collect event trace on target
  - inspect, process, and record target events on host

HPCToolkit's Support for OMPT & OpenMP

Simplified sketch

- Initialization: install callbacks
  - mandatory: thread begin/end, parallel region & task begin/end
  - blame shifting: idle & wait begin/end, mutex release
- When a profiling trigger fires
  - if thread is idle
    - apply blame shifting to attribute idleness to working threads
  - if thread is not idle
    - accept undirected blame for idleness of others
    - attribute work and blame to application-level calling context
- When a mutex release occurs
  - accept directed blame charged to that mutex
  - attribute blame to application-level calling context

Attribute costs to application-level calling context

- unwind call stack
- elide OpenMP runtime frames using OMPT frame information
- use info about nesting of tasks & regions to reconstruct full context

Program Structure Analysis

- Challenge: understanding performance of optimized code
  - template instantiation, function inlining, and code transformations
- Approach: recover program structure in detail using binary analysis in conjunction with information from the compiler
  - parse the machine code in an executable
  - for each instruction, recover its complete static call chain
  - build a CFG and identify loop nests with interval analysis
    - challenges: non-returning functions, tail calls, unreachable padding bytes

Tail Call Example from LLNL's LULESH

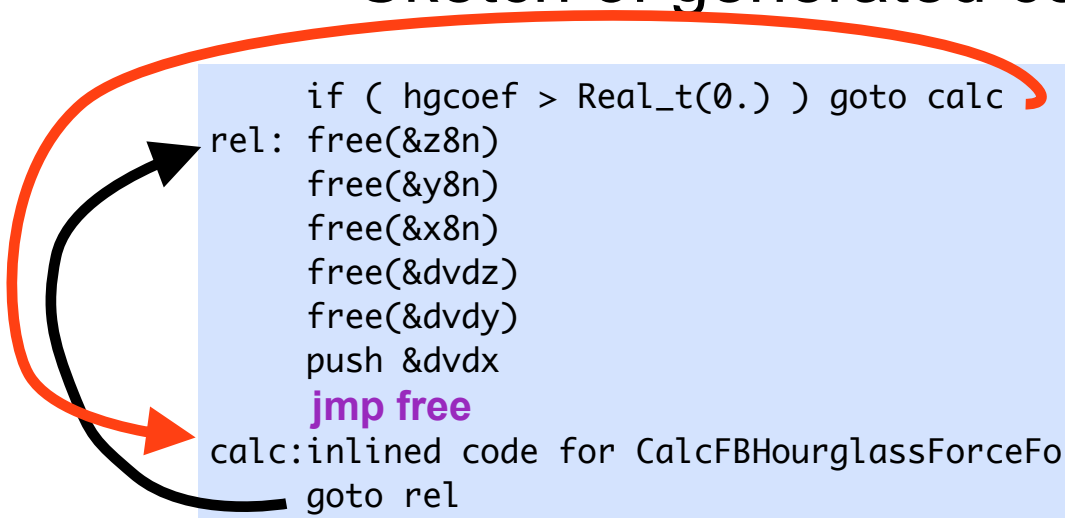
Fragment of source code

```
   if ( hgcoef > Real_t(0.) ) {
      CalcFBHourglassForceForElems(determ,x8n,y8n,z8n,dvdx,dvdy,dvdz,hgcoef);
   }

   Release(&z8n) ;
   Release(&y8n) ;
   Release(&x8n) ;
   Release(&dvdz) ;
   Release(&dvdy) ;
   Release(&dvdx) ;

   return ;
```

Sketch of generated code (gcc 4.4.6 -O3)



```
     if ( hgcoef > Real_t(0.) ) goto calc
rel: free(&z8n)
     free(&y8n)
     free(&x8n)
     free(&dvdz)
     free(&dvdy)
     push &dvdx
     jmp free
calc:inlined code for CalcFBHourglassForceForElems
     goto rel
```

Program Structure Analysis

- Challenge: understanding performance of optimized code
  - template instantiation, function inlining, and code transformations
- Approach: recover program structure in detail using binary analysis in conjunction with information from the compiler
  - parse the machine code in an executable
  - for each instruction, recover complete static call chains
  - build a CFG and identify loop nests with interval analysis
    - challenges: non-returning functions, tail calls, unreachable padding bytes
  - integrate information about loops and inlining
  - present information about highly optimized code similar to unoptimized executables

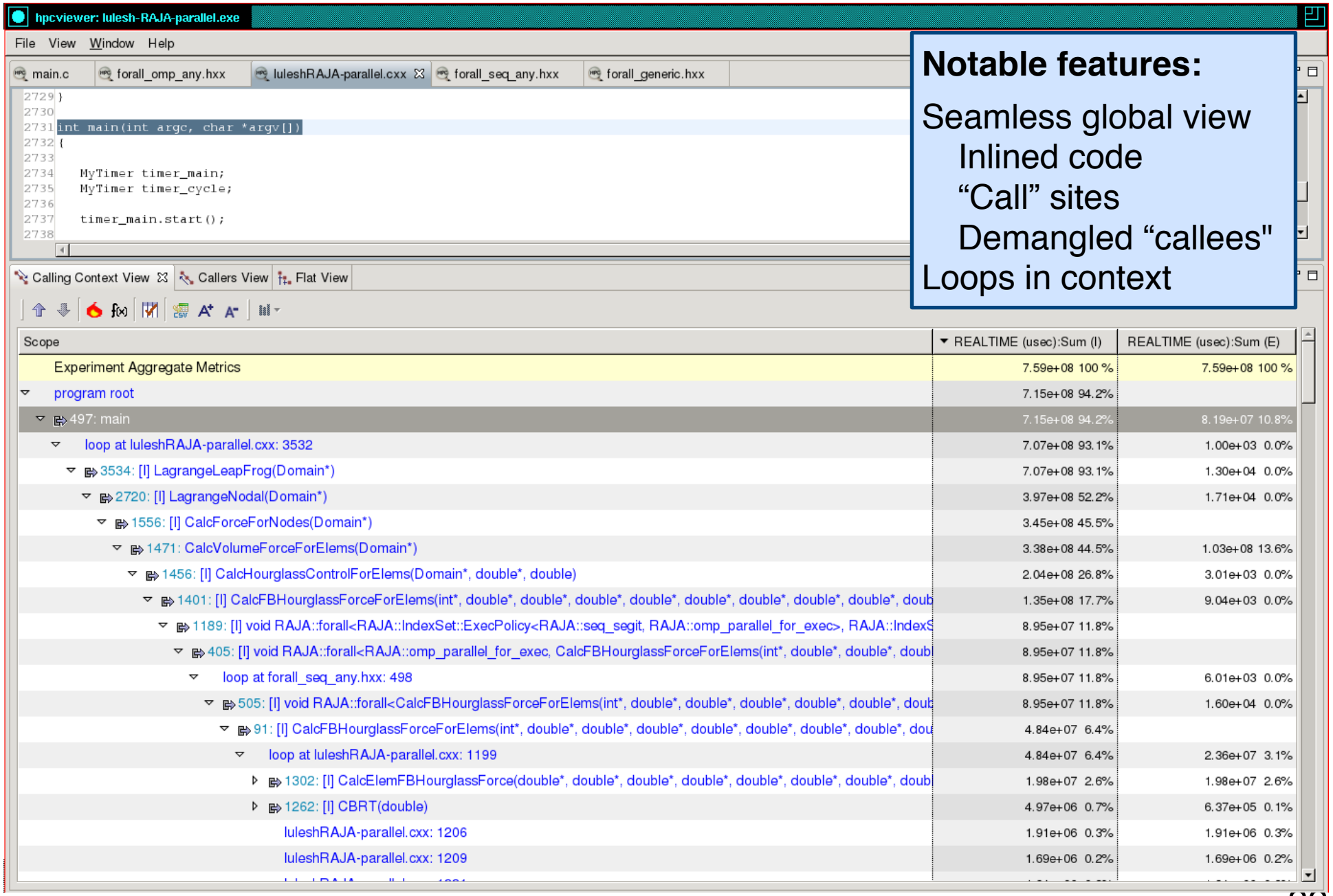
Case Study: LLNL's LULESH with RAJA

Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics

- Implementation using RAJA portability model
- Compiled with high optimization
  - icpc -g -O3 -msse4.1 -align -inline-max-total-size=20000 -inline-forceinline -ansi-alias -std=c++0x -openmp -debug inline-debug-info -parallel-source-info=2 -debug all
- Linked with OMPT-enabled LLVM OpenMP runtime
- Data collection
  - hpcrun -e REALTIME@1000 ./lulesh-RAJA-parallel.exe
    - implicitly uses the OMPT performance tools interface, which is enabled in our OMPT-enhanced version of the Intel LLVM OpenMP runtime

2 18-core Haswell
72+1 threads

Case Study: LLNL's LULESH with



Notable features:

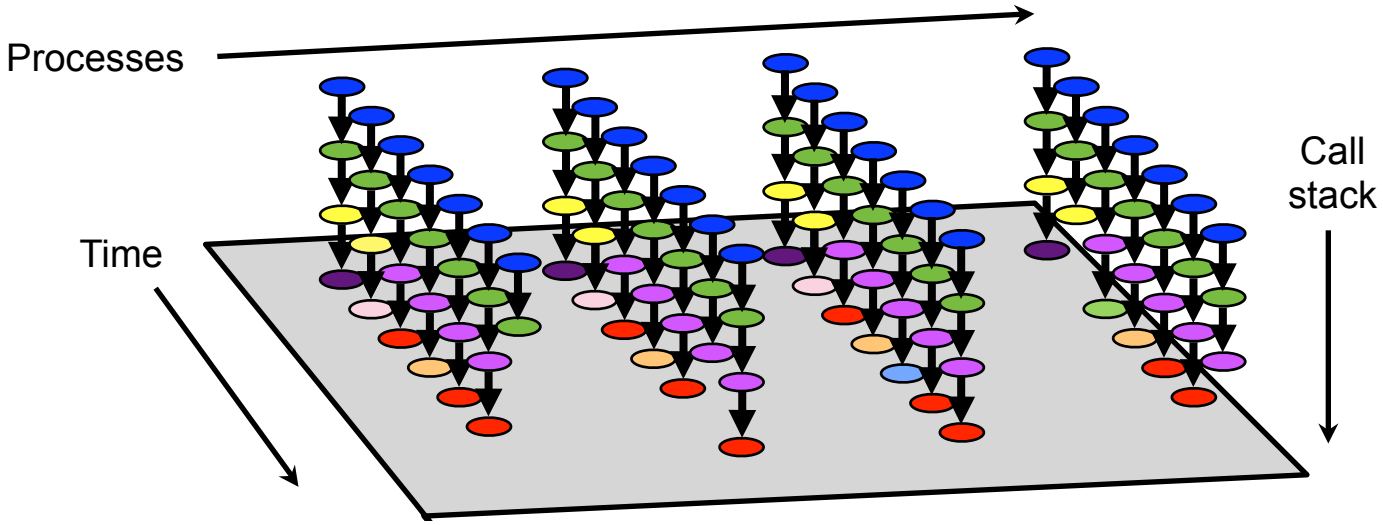
Seamless global view
- Inlined code
- "Call" sites
- Demangled "callees"

Loops in context

Understanding Temporal Behavior

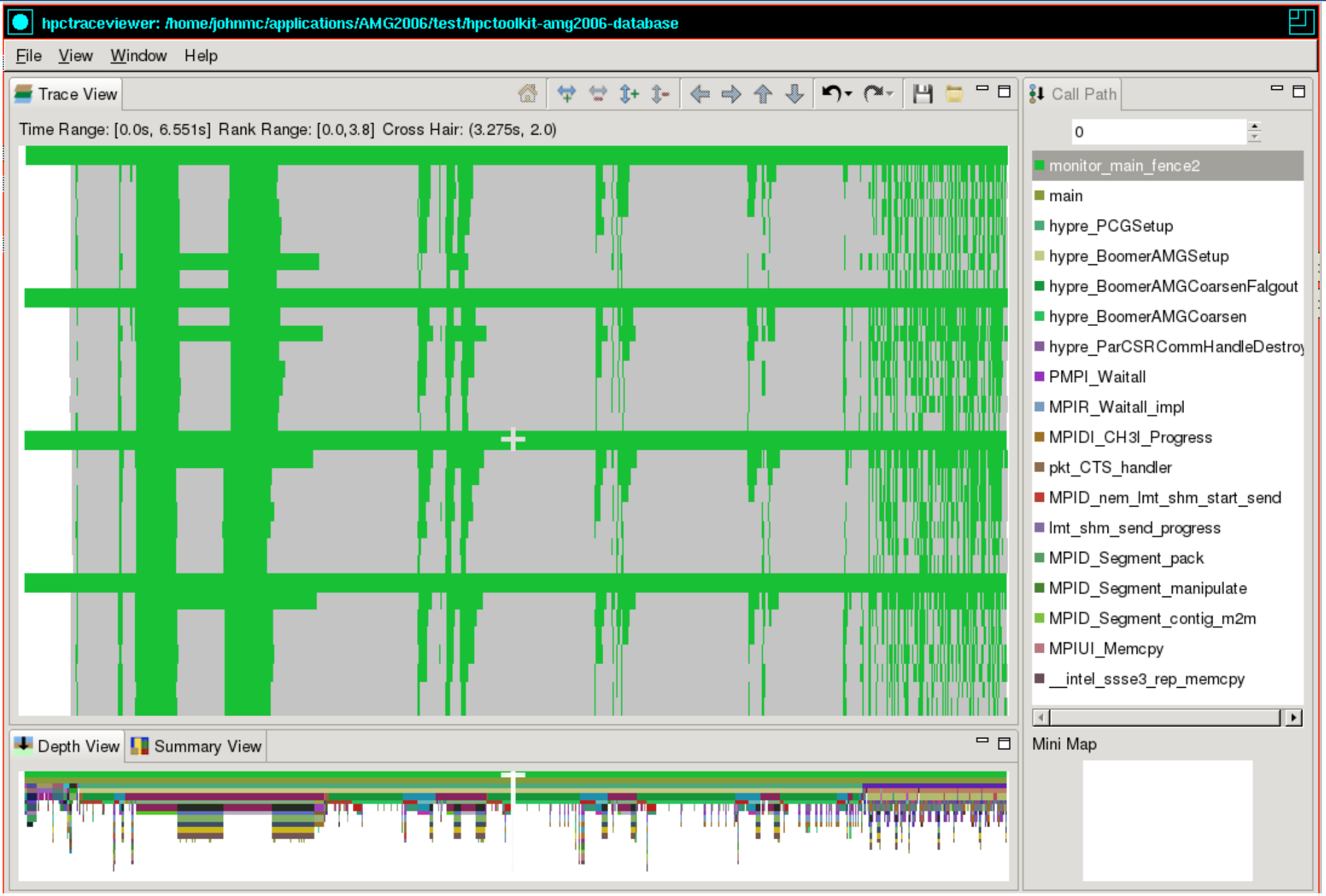
- Profiling compresses out the temporal dimension
  - temporal patterns, e.g. serialization, are invisible in profiles
- What can we do? Trace call path samples
  - sketch:
    - N times per second, take a call path sample of each thread
    - organize the samples for each thread along a time line
 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

      assign each procedure a color; view a depth slice of an execution



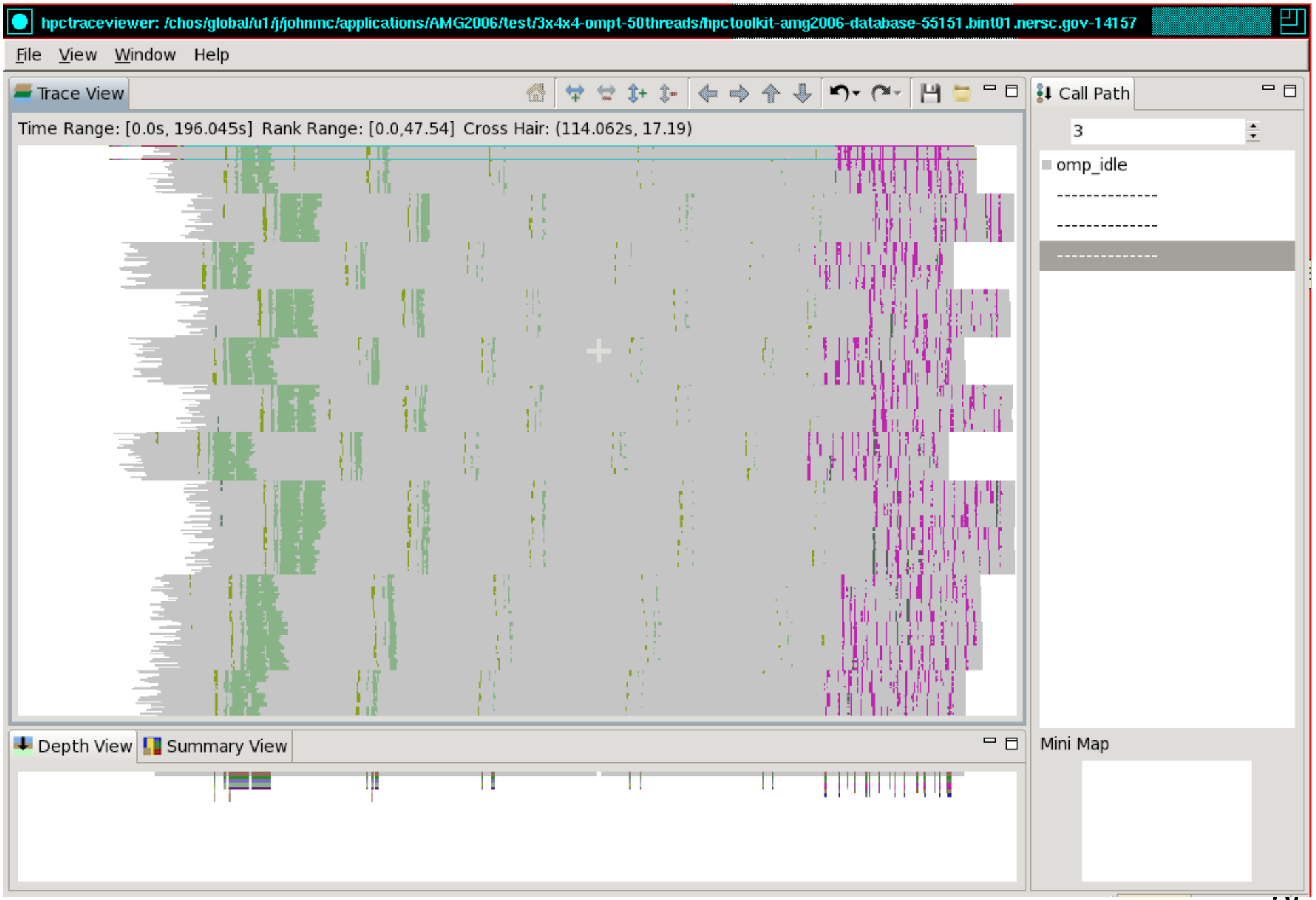
2 18-core Haswell
4 MPI ranks
6+3 threads per rank

Case Study: AMG2006



12 nodes on Babbage@NERSC
24 Xeon Phi
48 MPI ranks
50+5 threads per rank

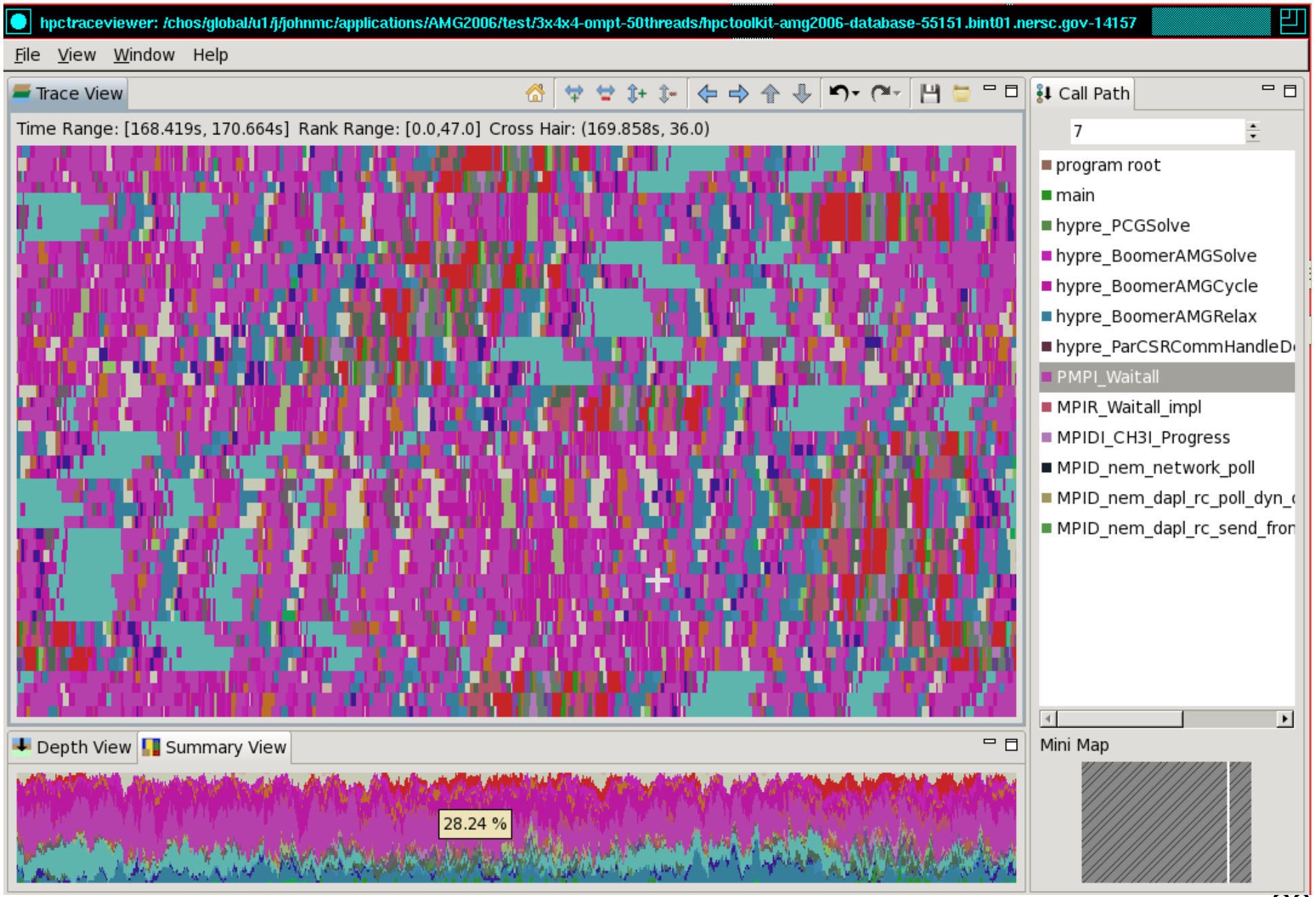
Case Study: AMG2006



12 nodes on Babbage@NERSC
24 Xeon Phi
48 MPI ranks
50+5 threads per rank

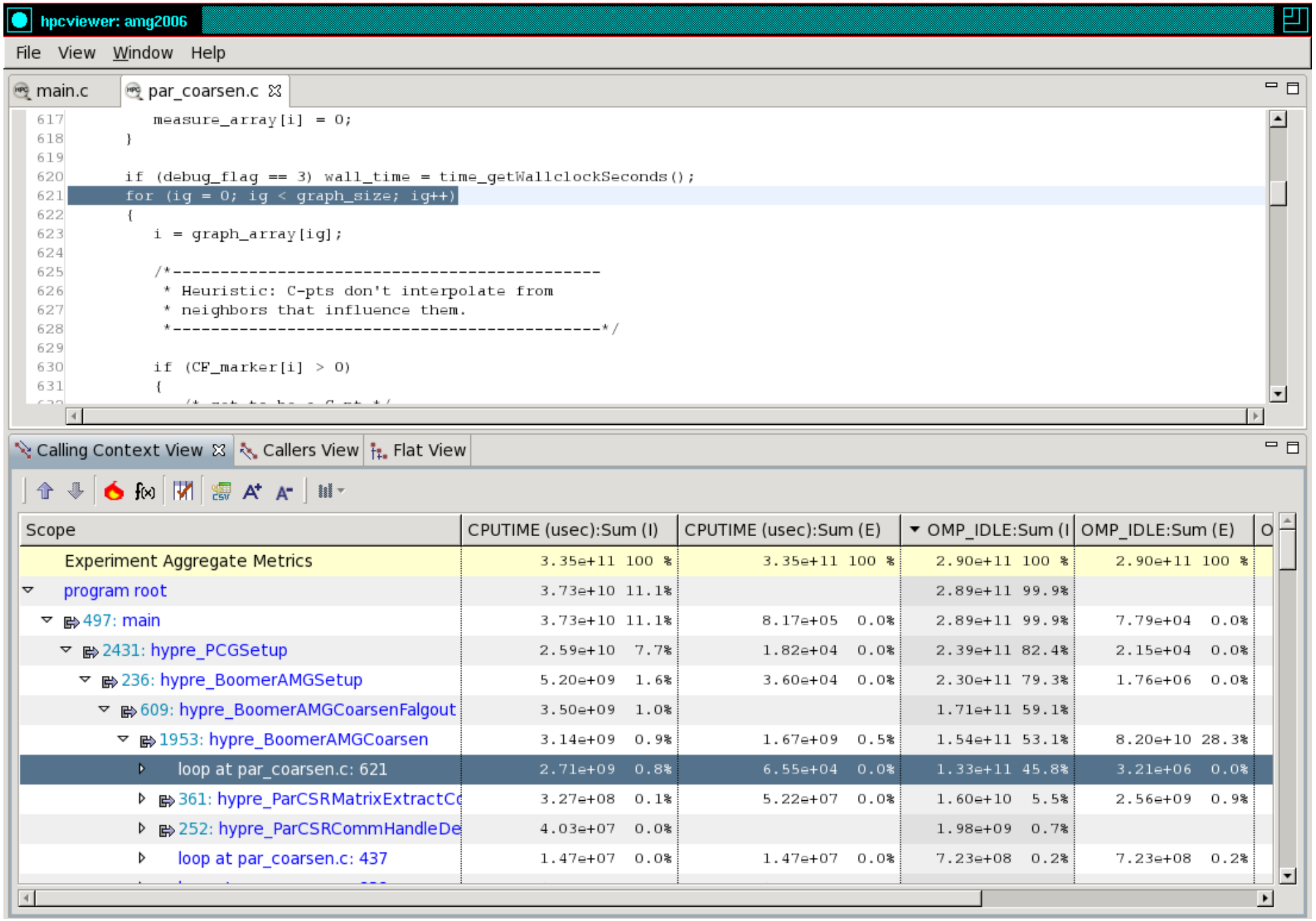
Case Study: AMG2006

Slice
Thread 0 from each MPI rank



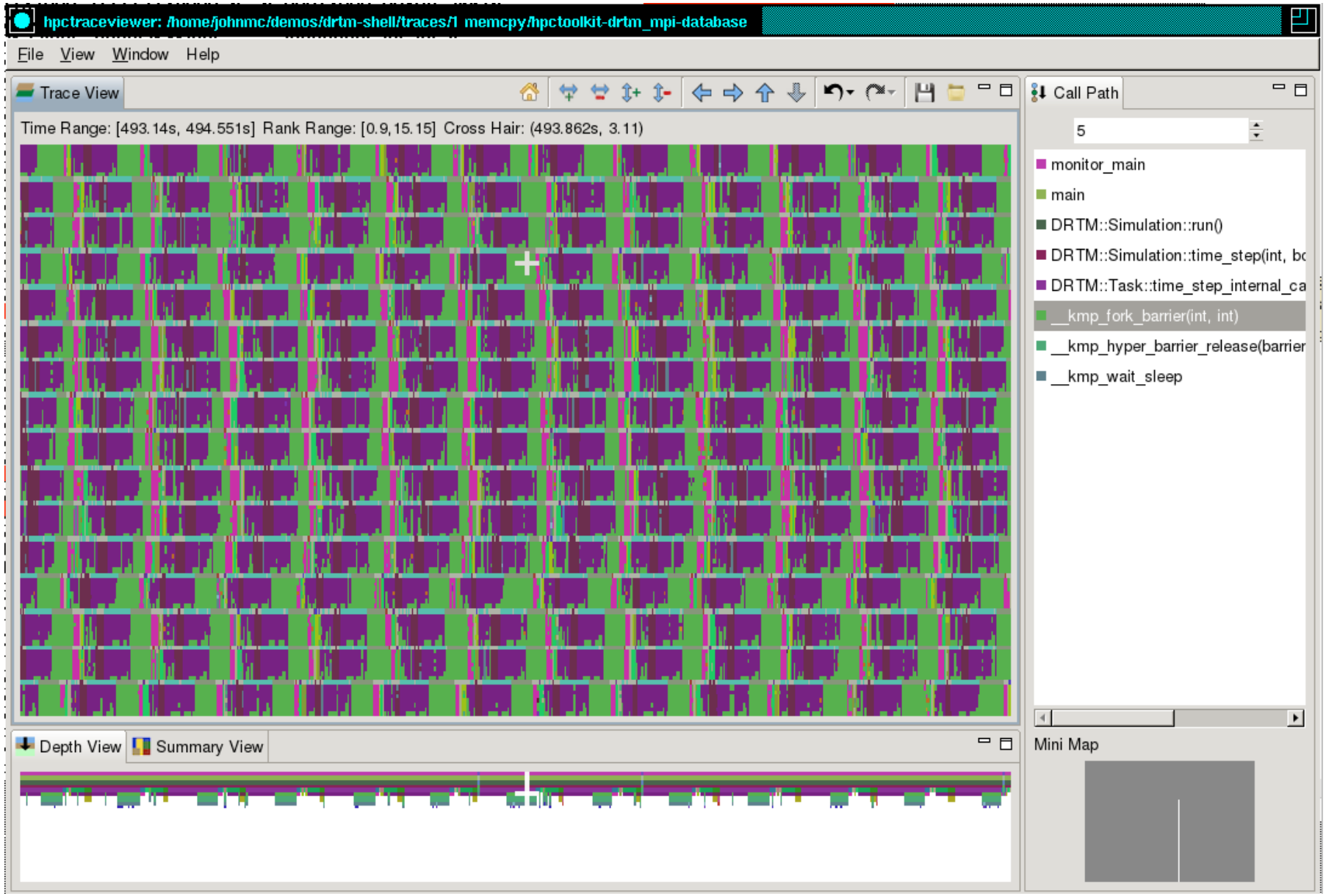
12 nodes on Babbage@NERSC
24 Xeon Phi
48 MPI ranks
50+5 threads per rank

Case Study: AMG2006



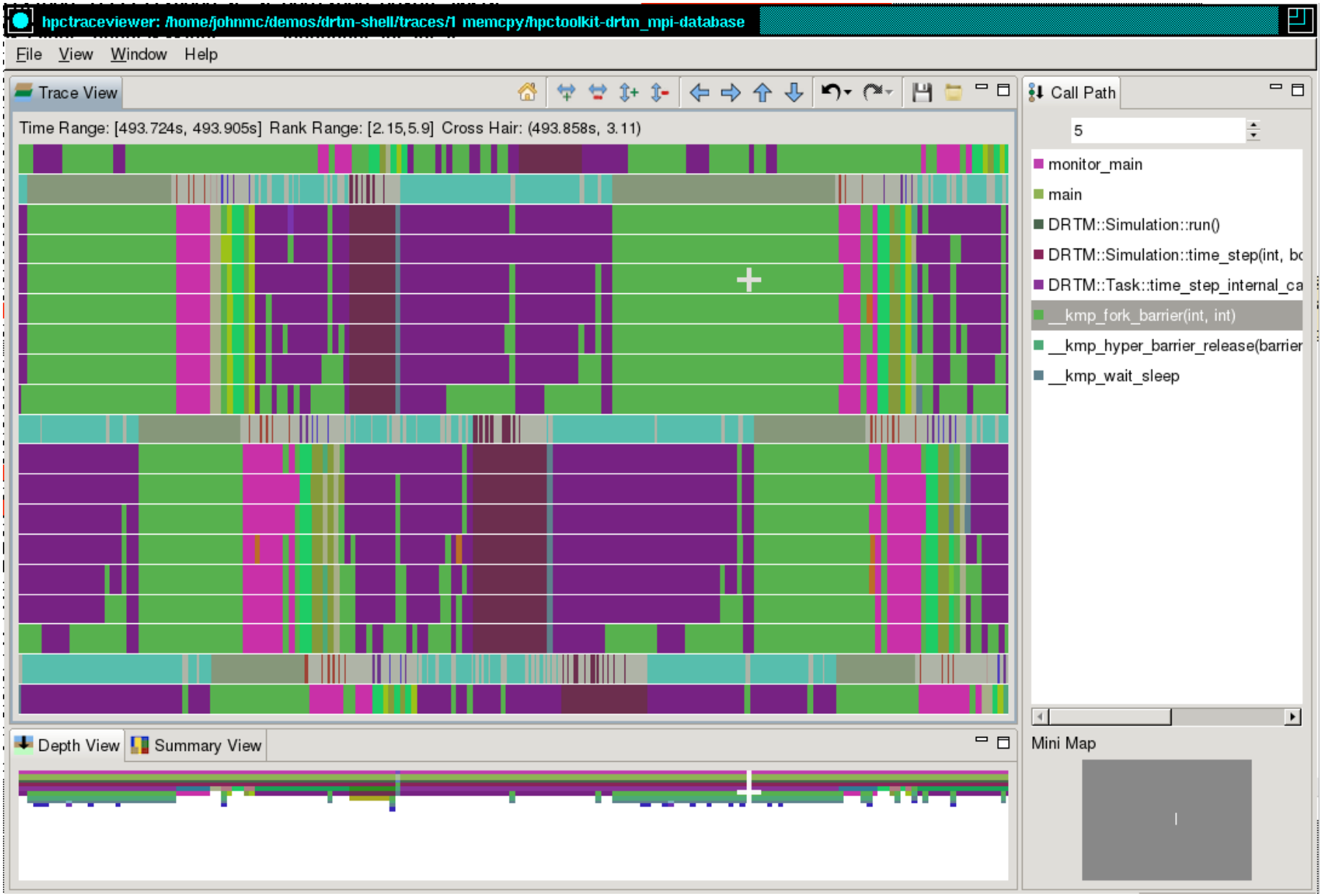
Sandy Bridge cluster
48 MPI ranks
6+3 threads/rank

Distributed RTM (Shell Research)



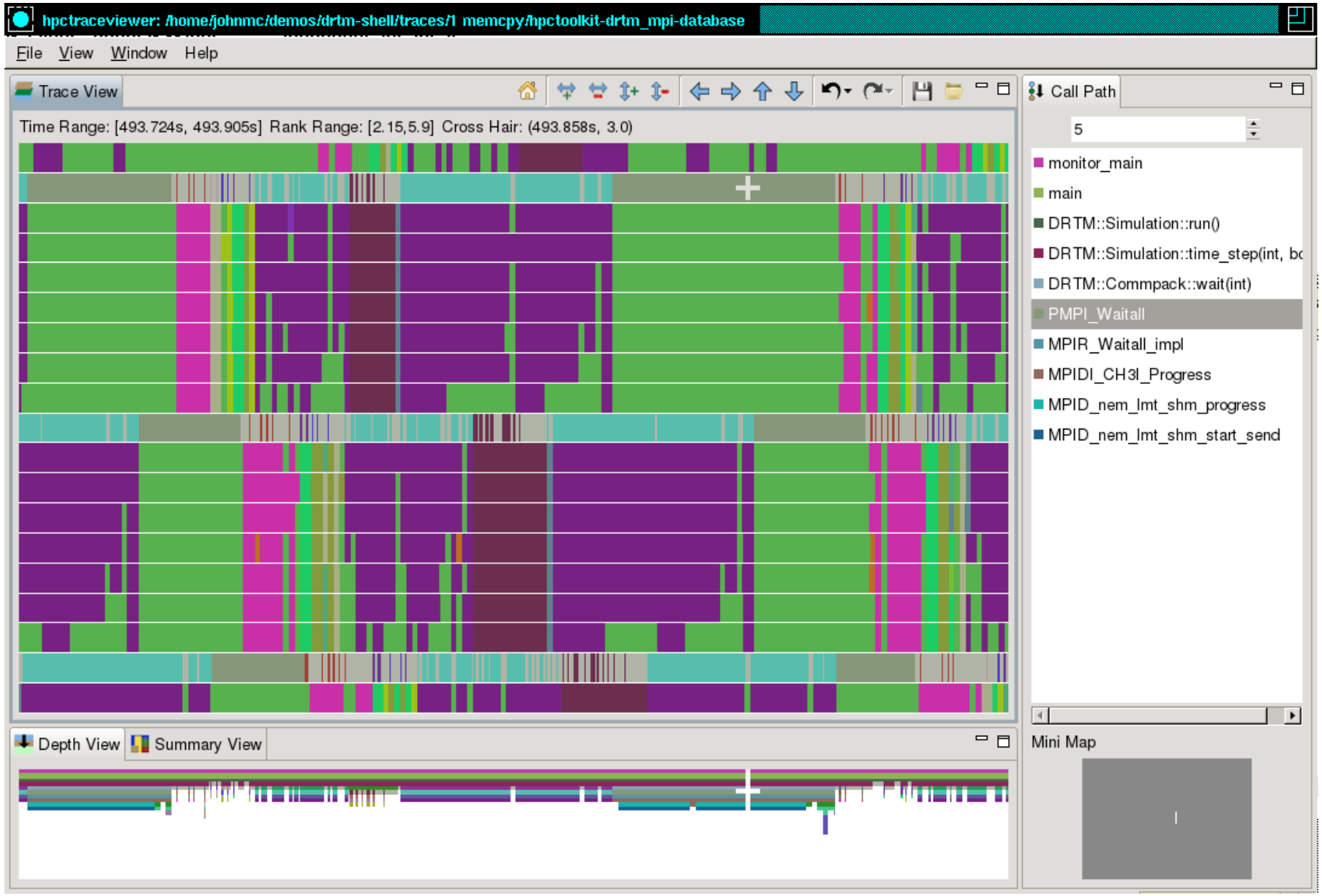
Sandy Bridge cluster
48 MPI ranks
6+3 threads/rank

Distributed RTM (Shell Research)



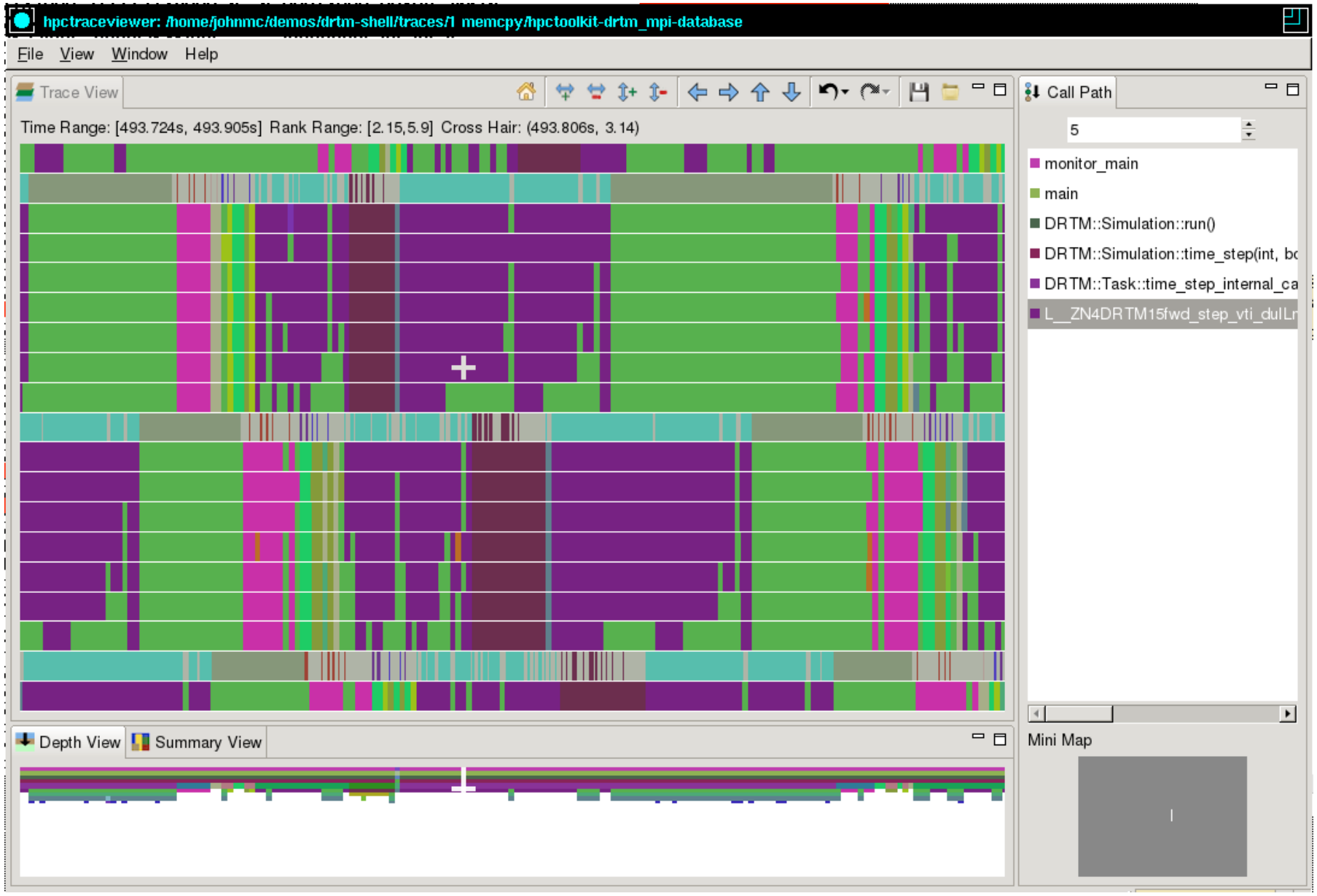
Sandy Bridge cluster
48 MPI ranks
6+3 threads/rank

Distributed RTM (Shell Research)



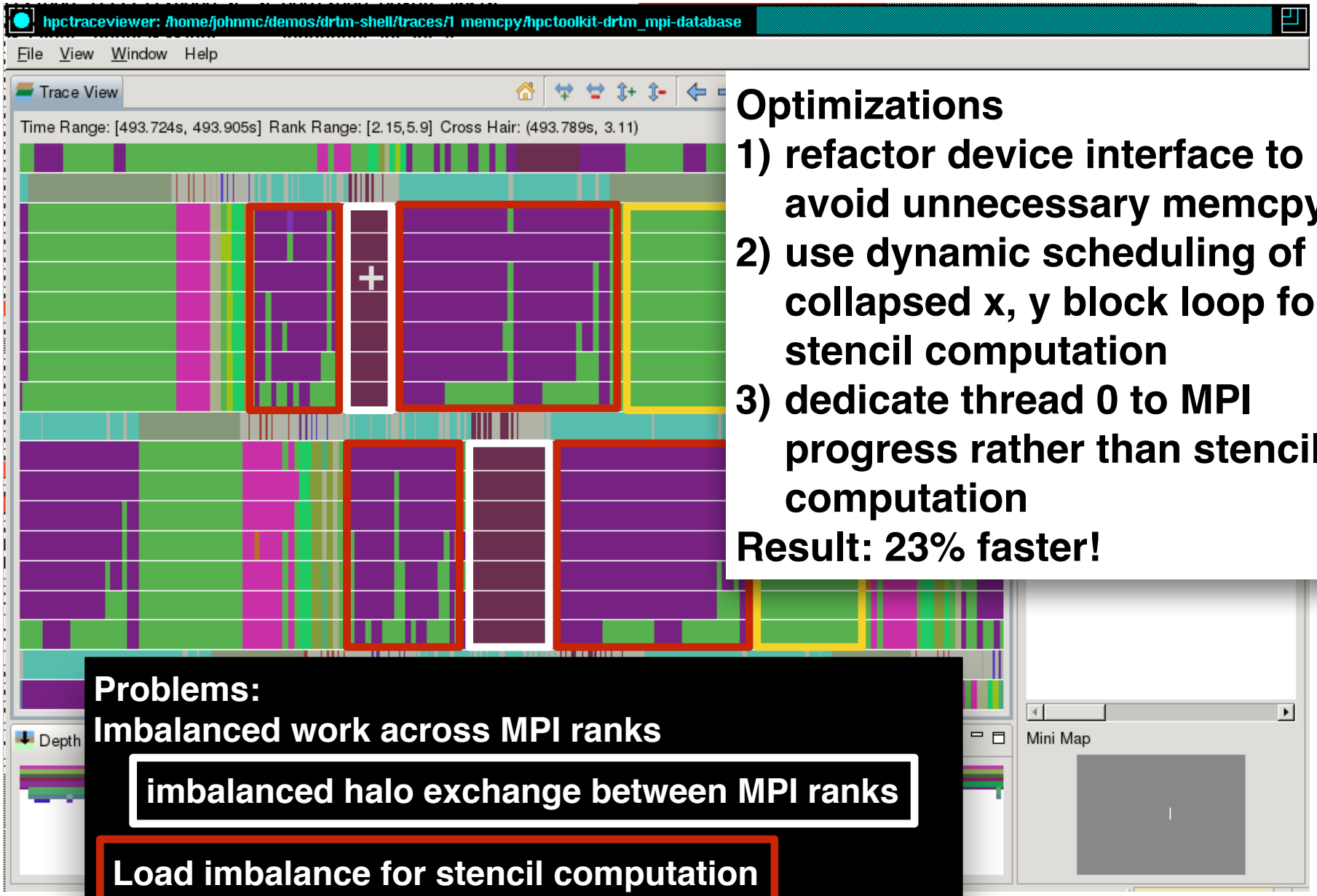
Sandy Bridge cluster
48 MPI ranks
6+3 threads/rank

Distributed RTM (Shell Research)



Sandy Bridge cluster
48 MPI ranks
6+3 threads/rank

Distributed RTM (Shell Research)



Optimizations

1) refactor device interface to avoid unnecessary memcpy
2) use dynamic scheduling of collapsed x, y block loop for stencil computation
3) dedicate thread 0 to MPI progress rather than stencil computation

Result: 23% faster!

Problems:
Imbalanced work across MPI ranks

- imbalanced halo exchange between MPI ranks
- Load imbalance for stencil computation
- Lots of thread idling waiting for communication

Target Device Monitoring

- Initialization
  - Host
    - inquires about the number and types of devices attached
    - specifies events to monitor on a target device
    - enables tracing on a target device
- Repeat until tracing disabled
  - Target device driver → host
    - invokes a callback on the host to request a new trace buffer
  - Device
    - records its events in a trace buffer
  - Target device driver → host
    - invokes a callback on the host to process and empty a trace buffer as necessary (full) or useful (flush)
- Finalization
  - Host
    - disables tracing, flushes events from device

Processing Device Trace Records

- Cursor advance: buffer, current_cursor, *next_cursor
- Get record type: OMPT, native, notype
- Extracting OMPT records
  - ompt_record_t *ompt_record_get(*buffer, cursor)
    - ompt_record_t: union type with space for any OMPT record
- OMPT record type

```
typedef struct ompt_record_s {
  ompt_event_t                    type;
  uint64_t                        time;
  ompt_thread_id_t                thread_id;
  ompt_dev_activity_id_t          dev_task_id;
  union {
    ompt_record_new_parallel_t new_parallel;
    ompt_record_task_t         task;

    …
  } record;
} ompt_record_t;
```

Processing Device Trace Records

- Extracting native records
  - void *ompt_record_native_get(*buffer, cursor)
  - ompt_record_native_abstract_t ompt_record_native_get_abstract(*native_record)
- Native record abstract type

```
typedef struct ompt_record_native_abstract_s {
    ompt_record_native_kind_t       kind,
    const char                     *type;
    uint64_t                        start_time;
    unit64_t                        end_time;
    uint64_t                        hwid;
    ompt_dev_task_id_t              dev_task_id;
} ompt_record_native_abstract_t;
```

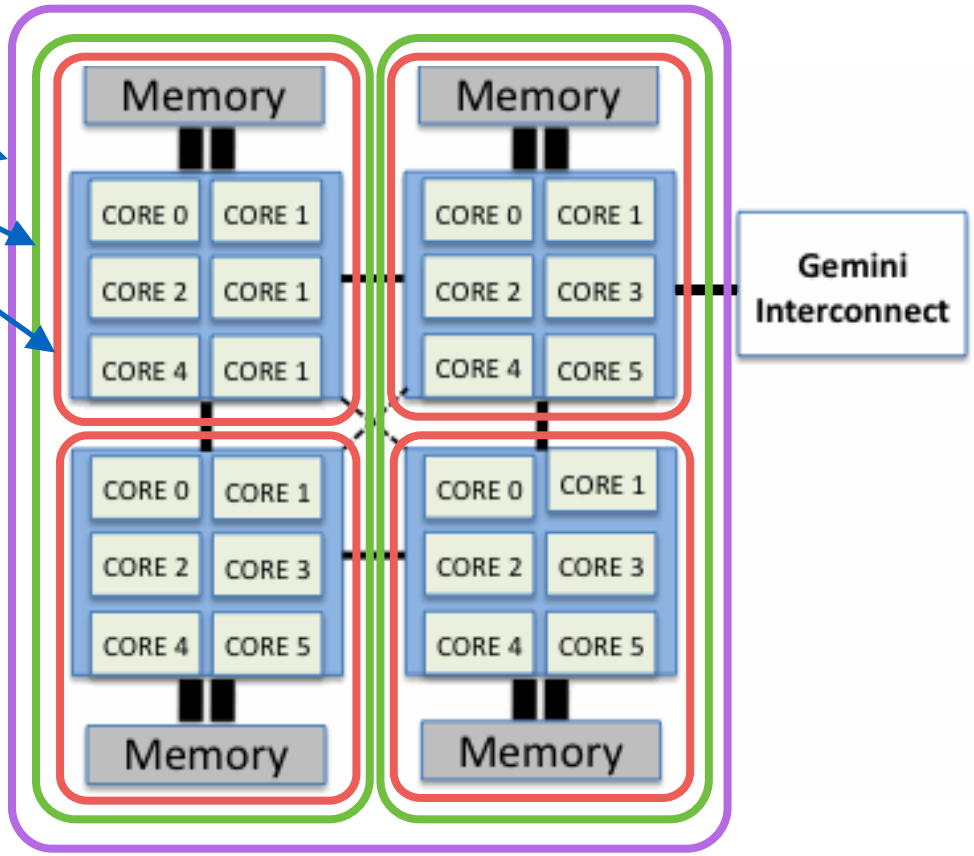
Data Centric Performance Analysis

- Memory latency and bandwidth: often bottlenecks for scientific codes on today's systems
- Users want tools that
  - understand memory hierarchy bottlenecks
    - attribute latency and interconnect traffic to code
    - attribute latency and interconnect traffic to data
  - explain how to correct them
    - explain what data threads touch
    - pinpoint where data mapping is determined by first-touch
- PhD research by Xu Liu (now Asst. Prof @ William and Mary)
  - new measurement techniques that support code-, data-, and address-centric attribution
  - new analysis techniques
    - new metrics to quantify bottlenecks
    - new analysis to identify where and how to optimize code

Example Hierarchical Node Architecture

A Hopper 24-core node

nested NUMA domains



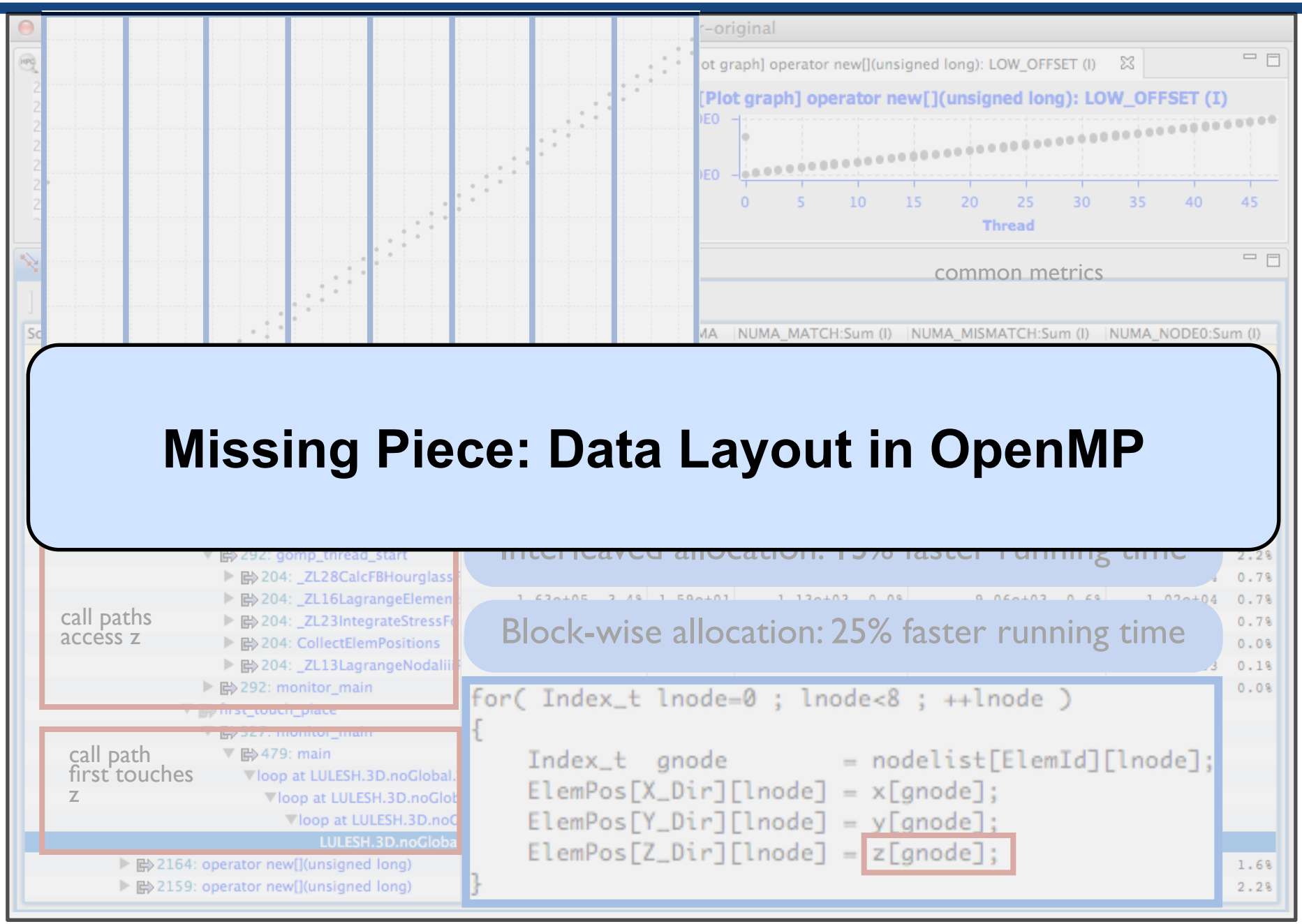
2 Magny-Cours cpus / node

2 6-core dies / socket

2 memory paths / die

4 HyperTransport3 links / die

LULESH on Platform with 8 NUMA Domains



Data Movement Costs will Dominate!

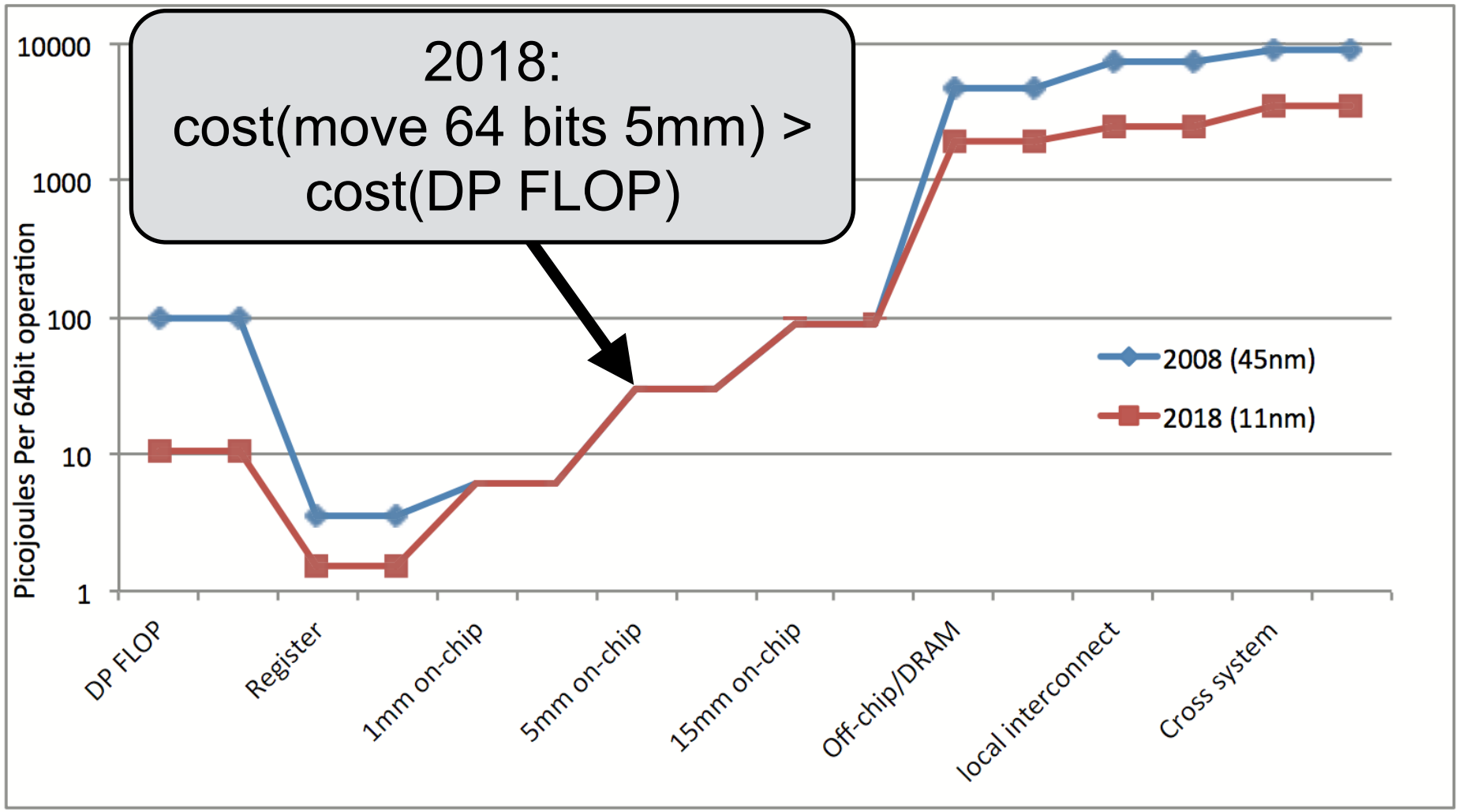


Figure credit: Peter M. Kogge and John Shalf. Exascale computing trends: Adjusting to the "new normal"' for computer architecture. Computing in Science and Engineering, 15(6):16–26, 2013.

The Data Layout Problem

- Data movement: critical optimization target for future systems
- Challenges
  - today's programming models are compute-centric
    - *default* usage of OpenMP assumes cores are equidistant to one another and to main memory
    - little for managing data locality or data movement
  - non-uniformity of data movement costs is growing
    - between nodes of a system
    - between processing elements within a chip
  - our current models are entirely misaligned with the underlying technological constraints to achieving efficient computation
  - modern programming environments offer few abstractions for managing data locality
  - without such abstractions, programmers must manually manage data locality using techniques such as loop-blocking

Important Types of Locality

- Vertical locality - within the cache hierarchy
  - spatial locality
  - temporal locality
  - opportunity: computation reordering to enhance reuse
- Horizontal locality - communication with others
  - opportunity: careful mapping of logical to physical topology

OpenMP Today: Data Layout with First Touch

- When a large data array is allocated via mmap, virtual pages are not bound to physical pages until they are first accessed
- Default: kernel binds a virtual page to a physical page local to the thread performing the first access
- Disadvantages
  - distribution of data is implicit in the accesses each thread performs when initializing data
    - easy to forget or get wrong
  - page granularity is a crude approximation of desired layout
    - quality is inversely proportional to object size
  - can't be adjusted on Linux
    - Solaris: MADV_ACCESS_LWP - hint to the kernel that the access pattern is changing; consider moving page to next touch

      Richard McDougall, Jim Mauro. Solaris Internals: Solaris 10 and OpenSolaris Kernel Architecture. Prentice Hall; 2 edition, 2006.

Alternatives to First-Touch Layouts

- Solaris: MADV_ACCESS_MANY - allocate data using random page placement
- Interleaved allocation using libnuma
  - http://linux.die.net/man/3/numa

Typical Approach Today

- Current programming models virtualize data movement using coherent caches
  - ignore topological locality for inter-processor communication
- Programming environments do not allow us to express locality information because in the past it could be ignored
- Need to evolve new programming environments
  - express information about locality that helps compilers and runtime systems to optimize data movement

Exploiting Data Affinity through Schedule Reuse

- Affinities between threads and data accesses
  - implicit in iteration schedules
- Reuse iteration schedules throughout execution
  - exploit temporal locality across parallel constructs

Dimitrios S. Nikolopoulos, Ernest Artiaga, Eduard Ayguadé, and Jesús Labarta. Exploiting memory affinity in OpenMP through schedule reuse. EWOMP 2001. SIGARCH Computer Architecture News 29(5): 49-55 (2001)

Computation & Data Transformations

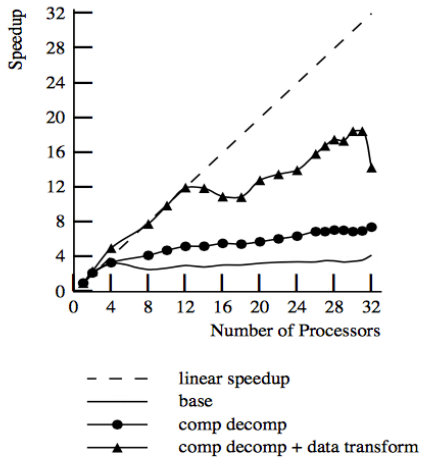


Figure 4: Vpenta Speedups

Program	Speedups (32 proc)		Critical Technique		Data Decompositions
	Base	Fully Optimized	Comp Decomp	Data Transform	
vpenta	4.2	14.3	√	√	F(*, BLOCK, *) A(*, BLOCK)
LU (1Kx1K)	19.5	33.5	√	√	A(*, CYCLIC)
stencil (512x512)	15.6	28.5	√	√	A(BLOCK,BLOCK)
ADI (1Kx1K)	8.0	22.9	√		A(*,BLOCK)
erlebacher	11.6	20.2	√	√	DUX(*,*,BLOCK) DUY(*,*,BLOCK) DUZ(*,BLOCK,*)
swm256	15.6	17.9			P(BLOCK,BLOCK)
tomcatv	4.9	18.0	√	√	AA(BLOCK,*)

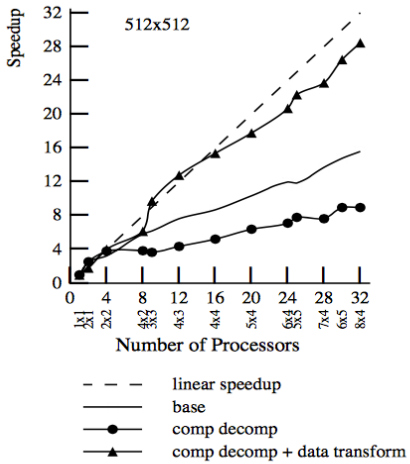


Figure 8: Five-Point Stencil Speedups

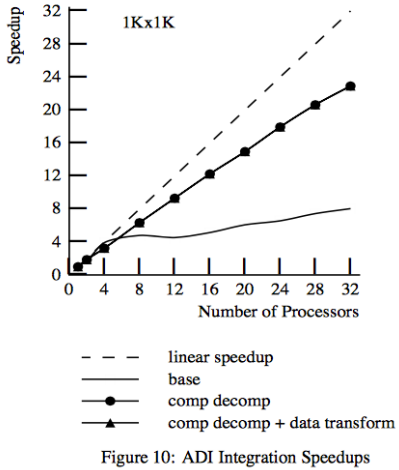


Figure 10: ADI Integration Speedups

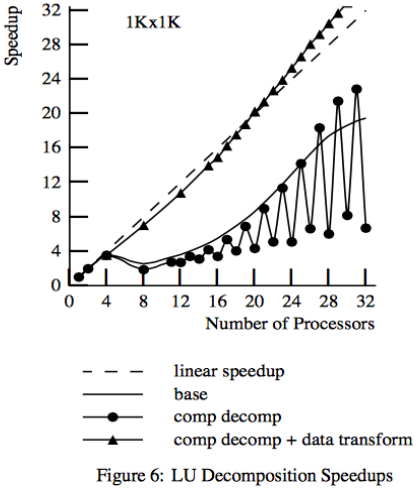


Figure 6: LU Decomposition Speedups

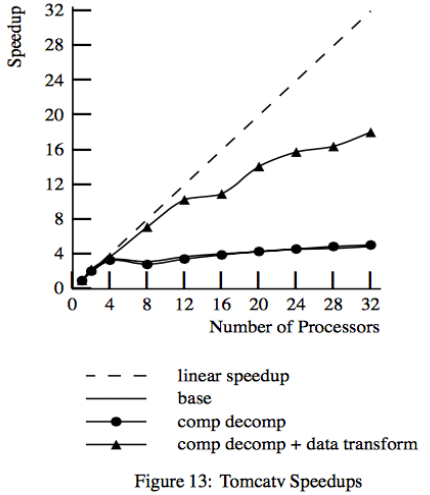


Figure 13: Tomcatv Speedups

Jennifer M. Anderson, Saman P. Amarasinghe, and Monica S. Lam. Data and computation transformations for multiprocessors. PPOPP '95, ACM, New York, NY, USA, 166-178. 1995.

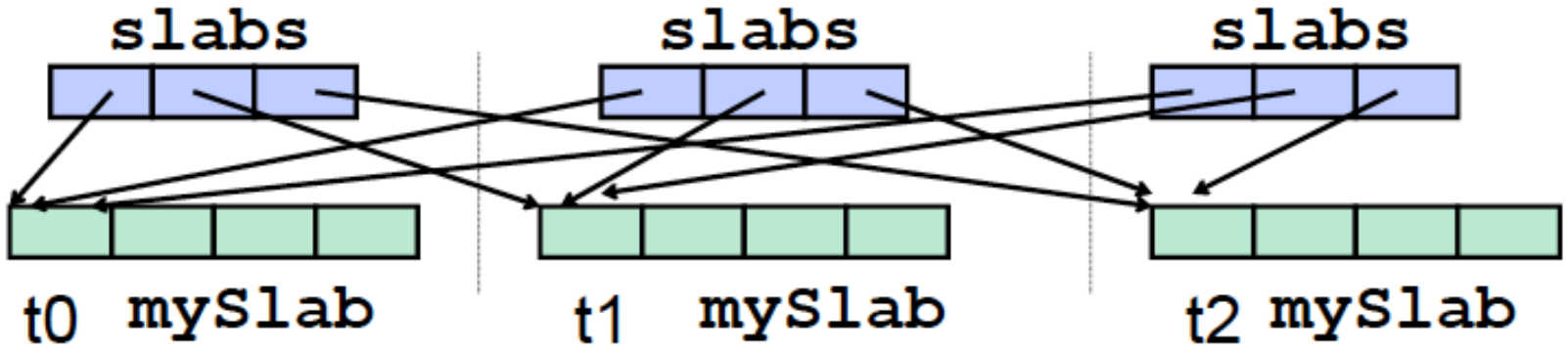
Shared Data in PGAS Languages

PGAS: programming model for parallel applications

- Provides a global memory address space that is partitioned such that a portion of the memory is local to each processor
- Locality information is encoded into addresses
  - can be exploited to schedule the tasks to the data
  - not vice versa
- Data exchange between regions of the global memory is based on one-sided, non-blocking, asynchronous and zero-copy communication
  - hiding and decoupling communication and synchronization are key requirements for efficient utilization of large machines
- Different approaches
  - global view: UPC, Chapel
  - local view: Titanium, Coarray Fortran

PGAS Data Layout Approaches

- UPC: data cut up into blocks (default size 1) and distributed cyclic among all of the nodes on a parallel system
- HPF uses global view, partitioned according to directives
- Titanium and Coarray Fortran: local view allocation
  - — Titanium uses explicit directories pointing to local views

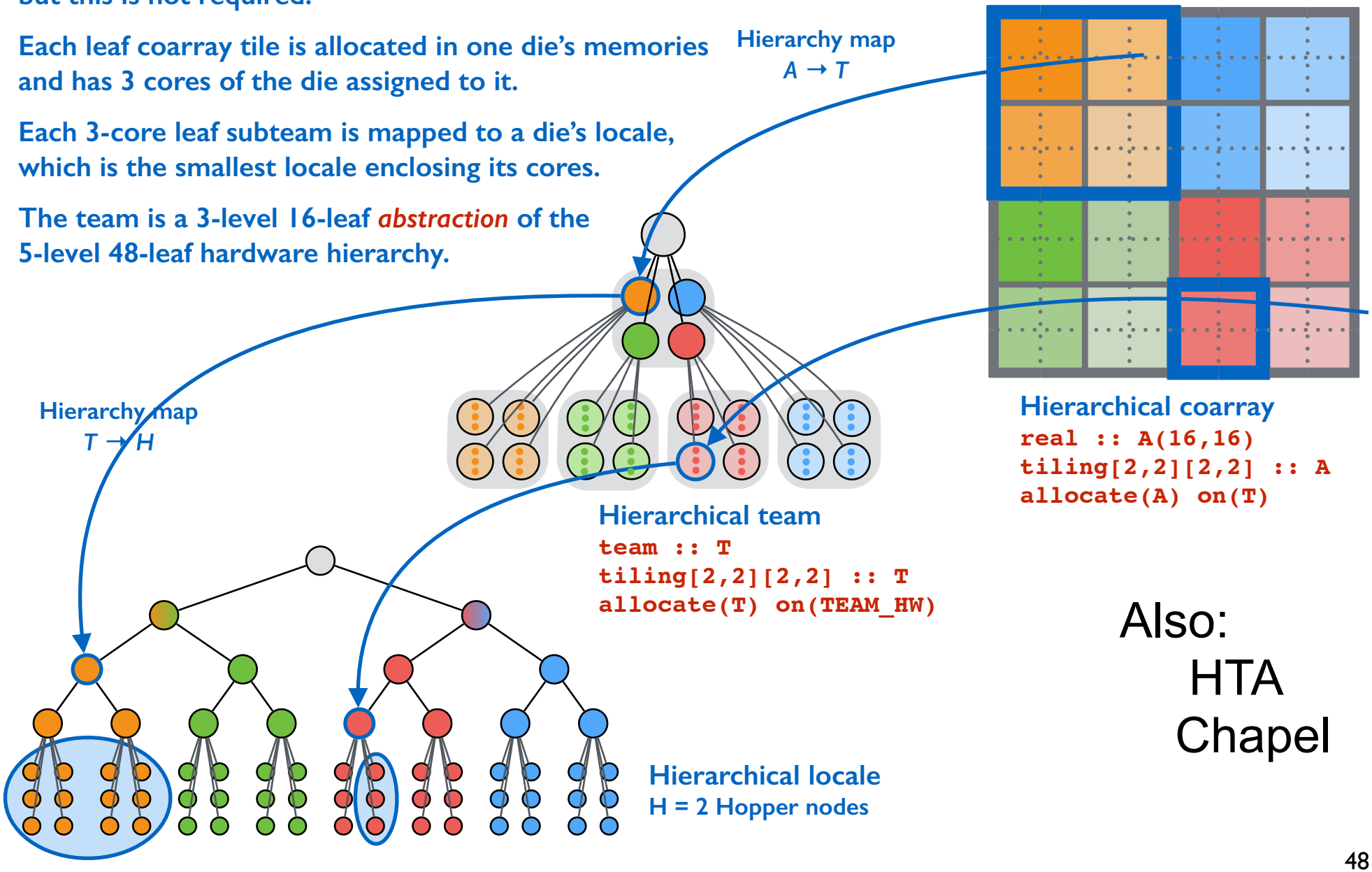


  - — CAF uses implicit, distributed hash tables for directories

Figure credit: Parallel Computing Laboratory, UC Berkeley. Titanium. Poster, SC10, 2010.

Hierarchical Data Layouts in HCAF

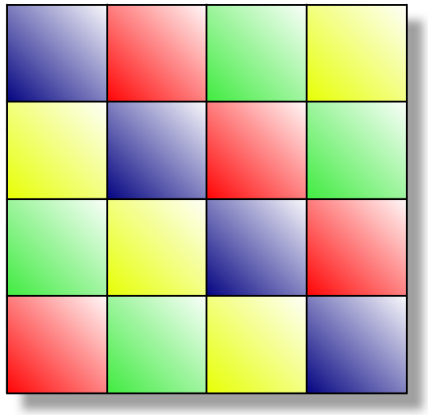
- Team and coarray hierarchies have same shape here, but this is not required.
- Each leaf coarray tile is allocated in one die's memories and has 3 cores of the die assigned to it.
- Each 3-core leaf subteam is mapped to a die's locale, which is the smallest locale enclosing its cores.
- The team is a 3-level 16-leaf *abstraction* of the 5-level 48-leaf hardware hierarchy.



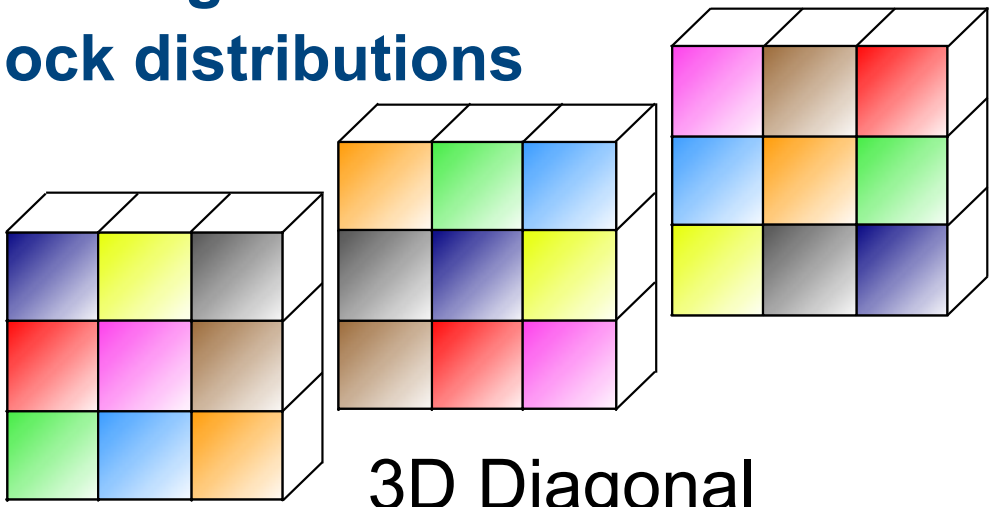
Also:
HTA
Chapel

Non-Hierarchical Layouts and Partitionings

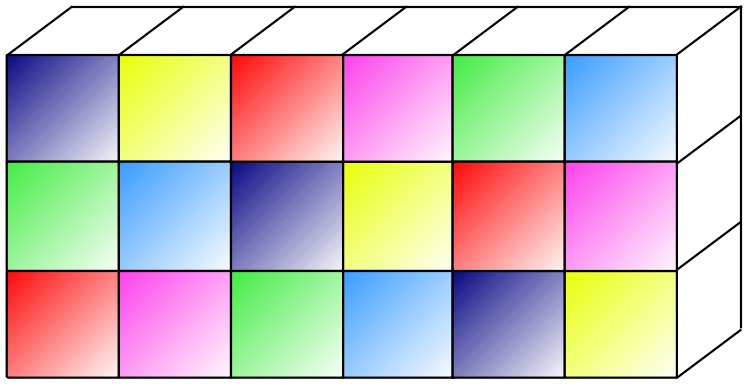
Multipartitionings: skewed-cyclic, block distributions

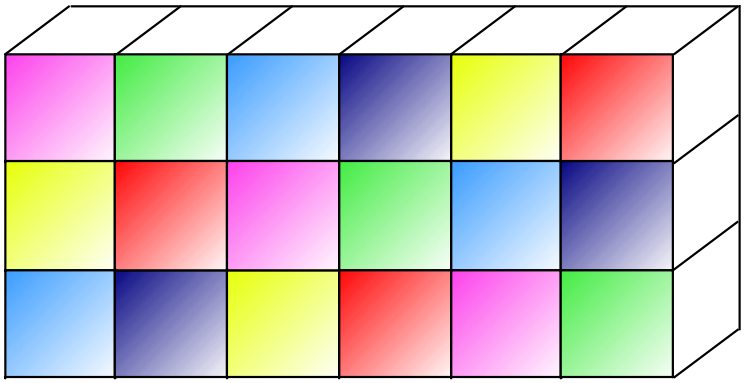


2D



3D Diagonal





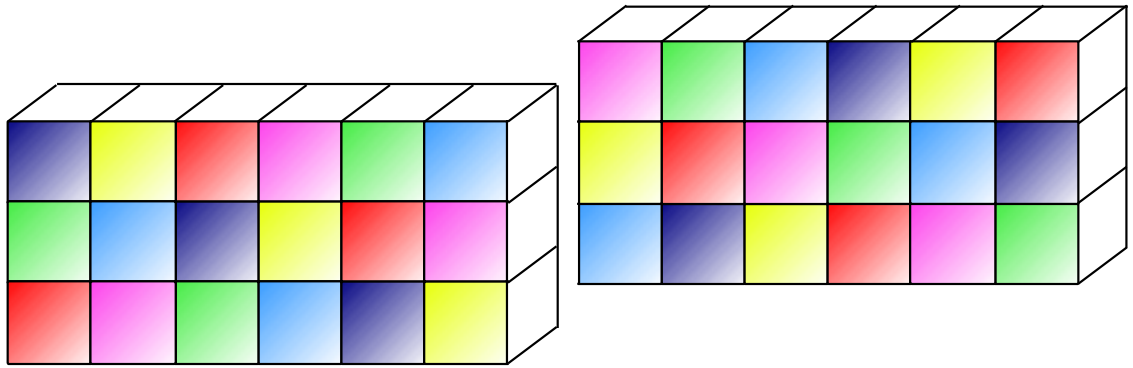
3D Generalized

A. Darte, J. Mellor-Crummey, R. Fowler, and D. Chavarria-Miranda. Generalized multipartitioning of multi-dimensional arrays for parallelizing line-sweep computations. *J. Parallel Distrib. Comput.* 63, 9 (September 2003), 887-911.

Handling Dependences

Computations with dependences between elements of a layout are common

- In RAJA, Keasler and Hornung used index segments, flags for each segment, and a segments waits for its inputs
- Multipartitioning is more complex than that, motivating something more
  - store each processors data blocks local to that processor
  - when generating code for a line-sweep computation with the dHPF compiler using a multi partitioned distribution
    - specify a tile schedule for each processor
    - a tile can't be scheduled until its predecessor is computed
    - iteration order within a tile depends on sweep direction



Language Level Abstractions for Data Locality

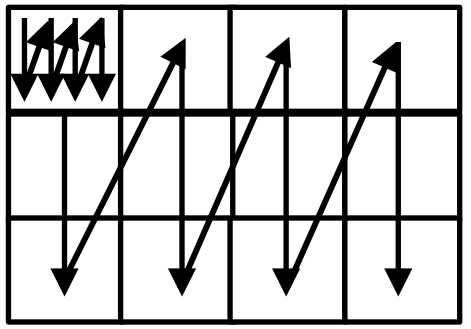
- Language-level abstractions for data locality offer many benefits
  - support for clearer syntax
    - document a program's locality decisions for a human reader
    - communicate developer's intent to a compiler
  - support for strong semantic checks
  - optimization opportunities inaccessible to library-based solutions
- Specify locality concerns as part of a program's declarations
  - declarations for distributed arrays
  - not annotations for each loop nest or array operation
- Multiresolution approach has many advantages
  - from high-level abstractions to lower-level imperative control as needed
  - author and deploy high-level abstractions using the lower-level concepts

Elements of a Solution

- Locality and communication should be evident in the source
  - CAF, UPC have this
  - HPF does not
- Primitives needed
  - move data to computation
  - move computation to data
- Program should not require rewriting when moving to a different architecture
  - machine model should be separate from user code
  - language and compiler automatically map code to machine structure
  - provide user with mechanisms for adapting to machine structure at execution time

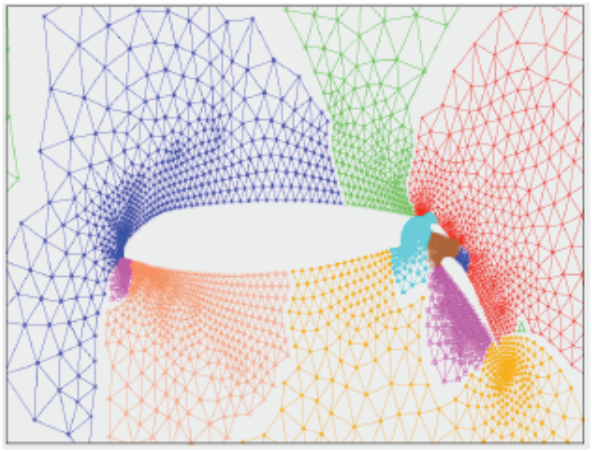
Some Building Blocks for a Solution

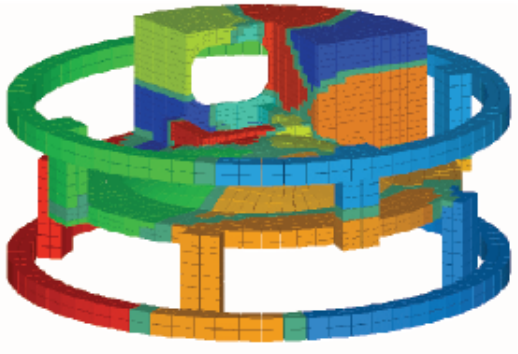
- Hierarchical iteration spaces in Chapel
  - index space of computation can be adjusted to match hierarchical data layouts
- Hierarchical place trees in Habanero
  - abstraction for runtime systems to reason about data locality for task-oriented parallelism
- Multidimensional array abstractions: tilings, array views, layouts and iterators to manage data locality
  - Kokkos: kayout mapping: a(i,j,k,l) → memory location
    - polymorphic, defined at compile time
      e.g., row-major, column-major, Morton ordering, padding, ...
    - user can specify Layout: View< ArrayType, Layout, Space >
  - Intel SIMD Building Blocks (C++ Library)
  - Chapel domain maps

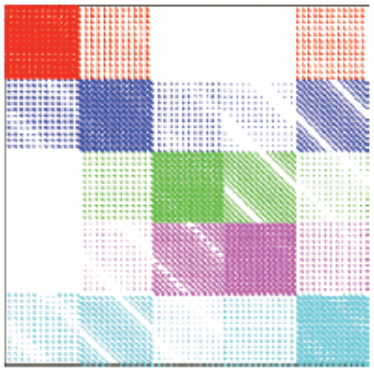


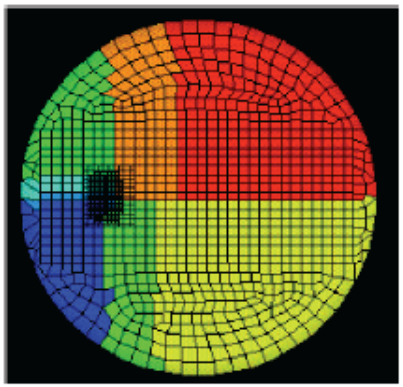
What about Irregular Problems?

- Broad class of problems have irregular structure
- Map application data to processors for computing in parallel
- Apply to grid points, elements, matrix rows, particles









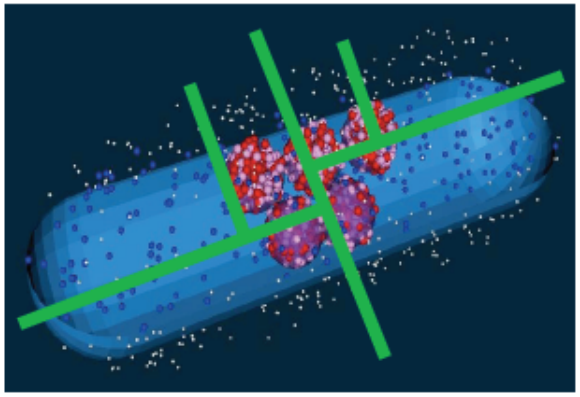
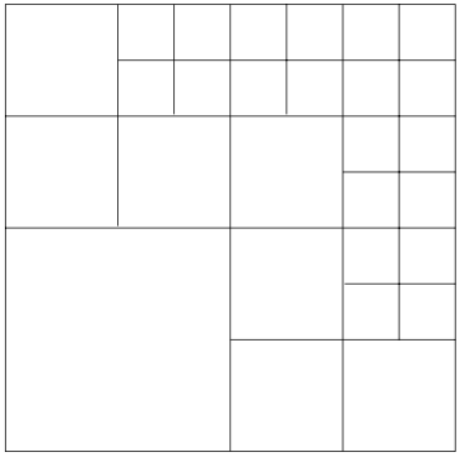
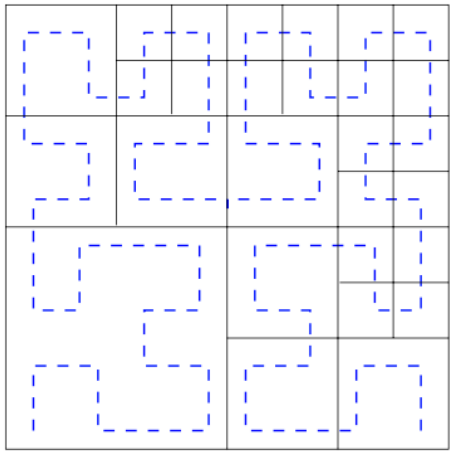


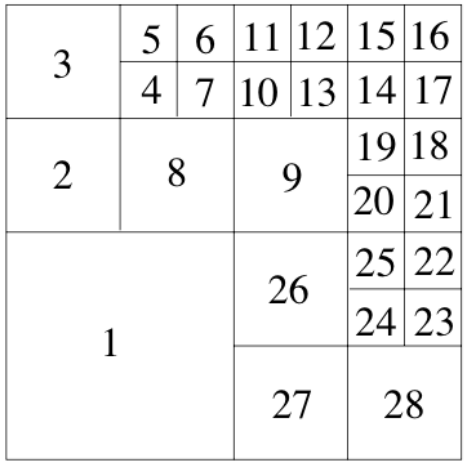
Figure credit: Erik Boman, Cedric Chevalier, Karen Devine. The Zoltan Toolkit – Partitioning, Ordering, and Coloring. Dagstuhl Tutorial. 2009.

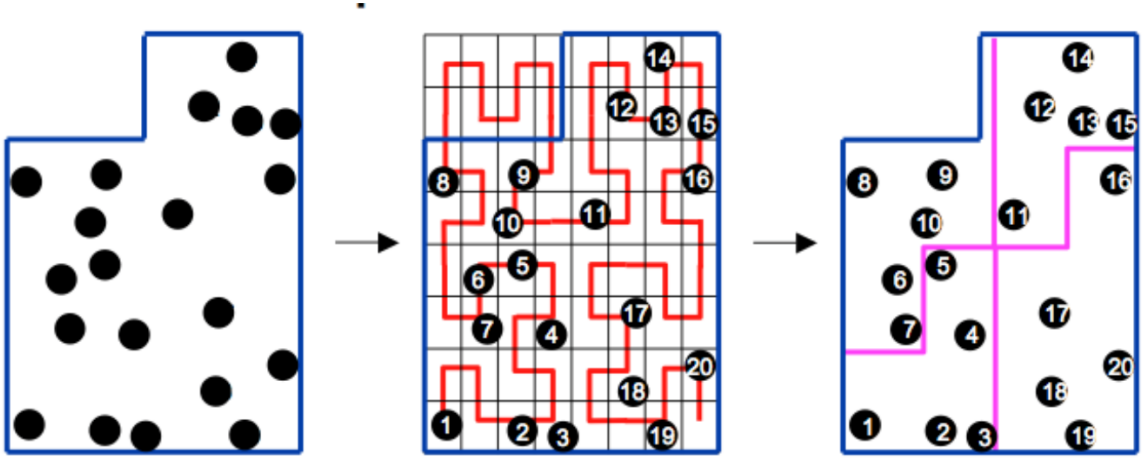
Space-filling Curves (SFC)

Applies to adaptively refined meshes, particles, …









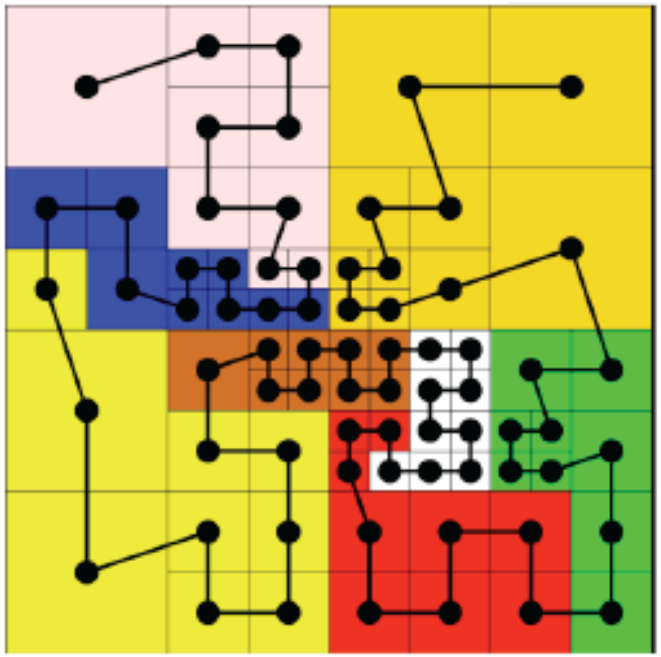


Figure credit: Figure Credit: K. Schloegel, et al. Graph Partitioning for High Performance Scientific Simulations. In CRPC Parallel Computing Handbook. Morgan Kauffman, 2000. Marsha Berger and Andreas Klöckner. Lecture 12: Load balancing and partitioning. G63.2011.002/G22.2945.001. NYU. November 16, 2010.

High Performance Requires Support at All Levels

- Programming model abstractions to
  - express parallelism
  - partition and map data
  - colocate computation with data
- Tools to pinpoint, quantify, and explain performance losses
- Runtime libraries
  - to manage data and computation
  - collect performance information
- Operating system interfaces that support
  - control of hardware performance monitoring units
  - introspection into all aspects of a program's performance
- Hardware performance monitoring units
  - measure and attribute computation, communication, data movement, I/O

Looking Forward

- Developers will need tools to succeed with emerging systems
  - getting tools right requires an approach that spans HW and SW
    - must consider these issues early to achieve a good result
  - a good starting point will be to incorporate useful solutions into the OpenMP standard
- Data layout problem is difficult
  - many pieces of solutions
  - a language-based approach is most likely to succeed
  - the alternative is only knowing information at runtime
- Codesign will be important to develop good solutions
  - consider the holistic design for computing systems – programming models, algorithms, and HW, monitoring, and tools

References

- Didem Unat et al. Programming Abstractions for Data Locality. 2014 Workshop on Programming Abstractions for Data Locality. Lugano, Switzerland April 28–29, 2014.
- Peter M. Kogge and John Shalf. Exascale computing trends: Adjusting to the ”new normal”’ for computer architecture. Computing in Science and Engineering, 15(6):16–26, 2013.
- H. Carter Edwards and Christian Trott. Kokkos: Performance Portable Thread-Parallel Execution and Data Structures On Next Generation Platforms. PADAL Workshop, June 24-25, 2015. SAND2015-4954C.
- Jeff Keasler, Rich Hornung. Partitioning and Scheduling for Enhanced Locality. PADAL Workshop, June 24-25, 2015. LLNL-PRES-673743.