Preparing for Performance Analysis at Exascale

Jonathon Anderson, Yumeng Liu, John Mellor-Crummey

Rice University

STW 2022

June 20, 2022

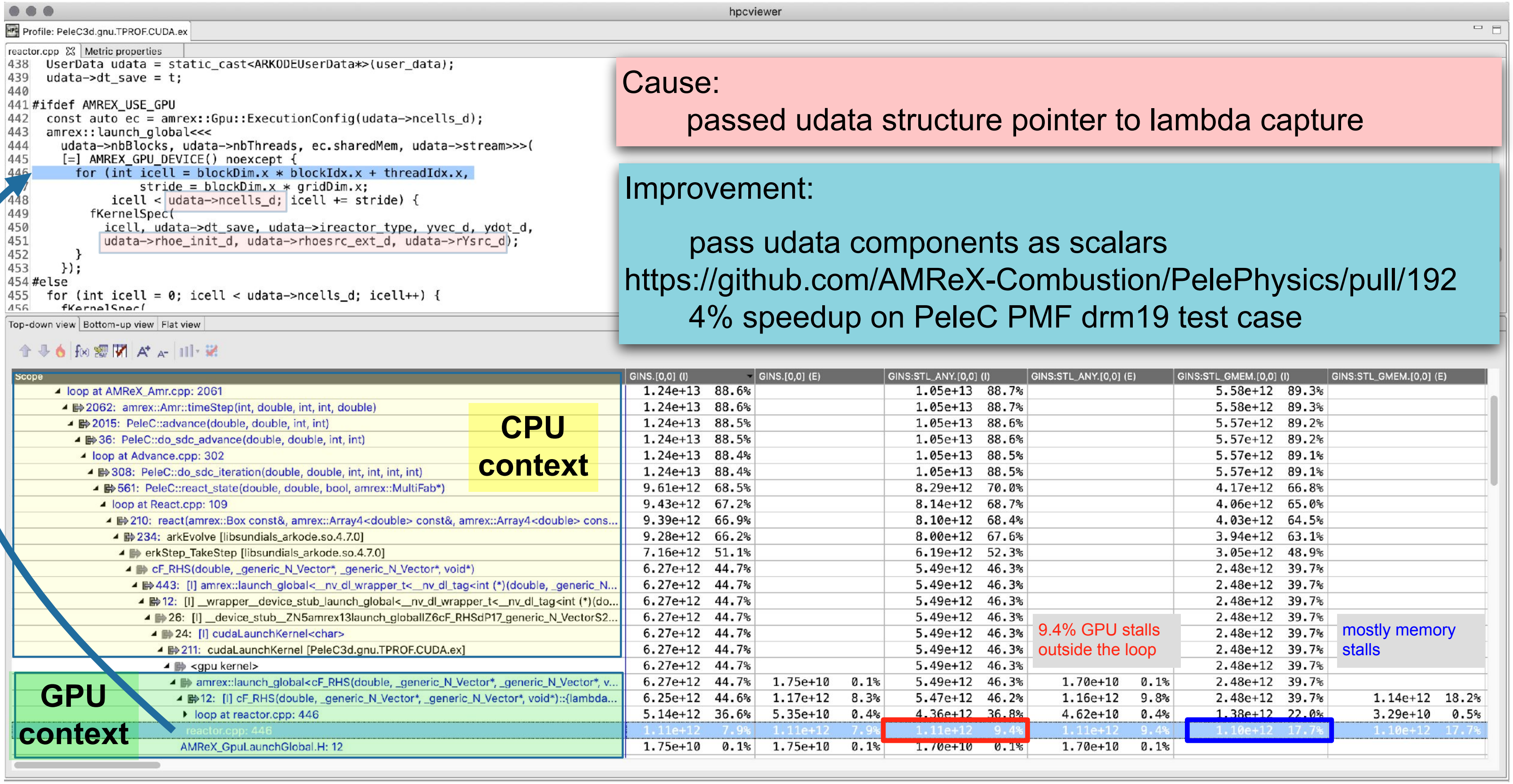
Performance Analysis in the Exascale Era

- Forthcoming exascale systems pose new challenges
  - Loss may only appear at the full scale of the machine, 10000s of compute nodes
    - Performance must be gathered from every application thread: very large data
  - Rich performance data must be gathered for a complete picture, over 130 GPU metrics
    - Some metrics are only present for GPU code regions: very sparse data
- Performance tools must handle these issues…
  - …But still provide detailed, useful analysis results!

Outline

- HPCToolkit at a glance
- Exploiting natural sparsity in performance data
- Streaming aggregation for highly-parallel processing of performance data
- Evaluation
- Conclusions
- Ongoing work: exploiting distributed object storage

HPCToolkit: Fine-grain Measurement and Attribution within Kernels



Cause:

passed udata structure pointer to lambda capture

Improvement:

pass udata components as scalars

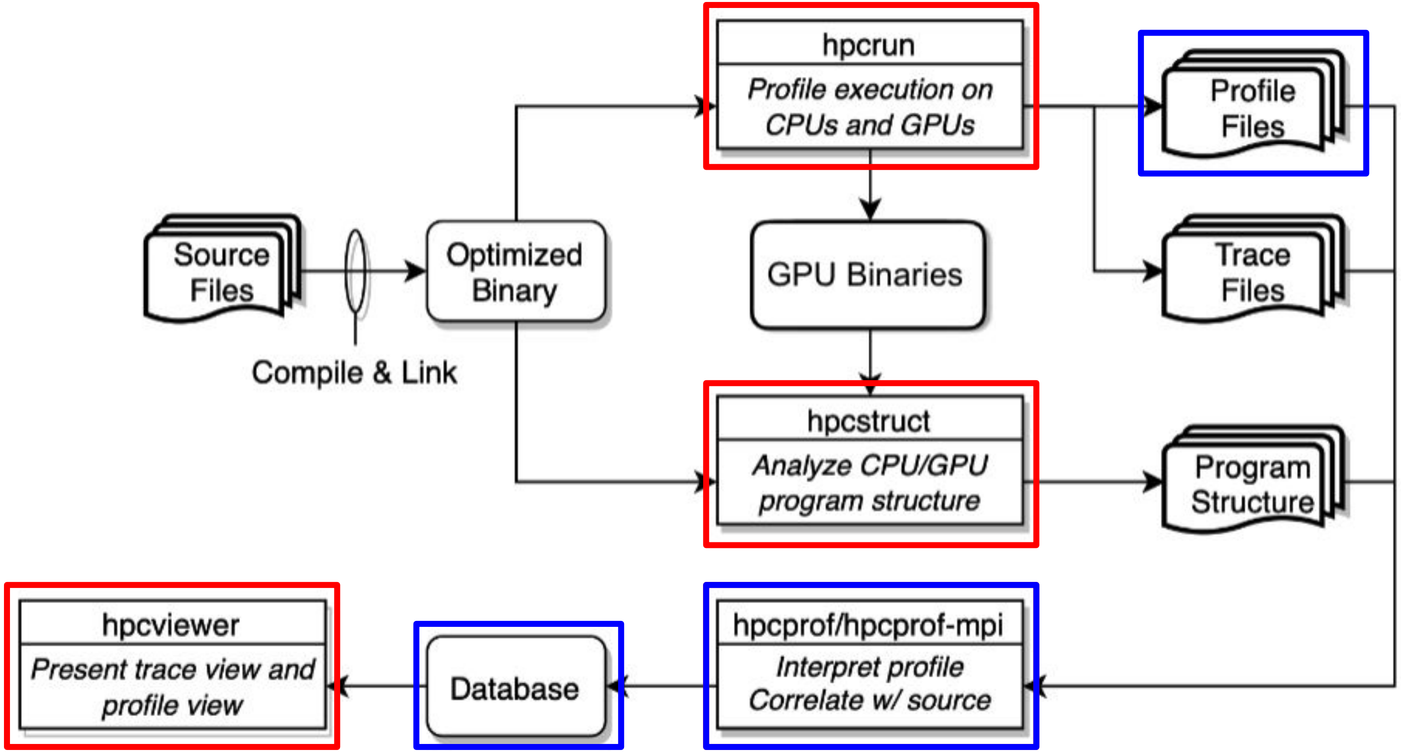
https://github.com/AMReX-Combustion/PelePhysics/pull/192

4% speedup on PeleC PMF drm19 test case

Experiments on Summit (NVIDIA GPUs)

EXASCALE COMPUTING PROJECT

HPCToolkit Our Work



Performance Data is Sparse

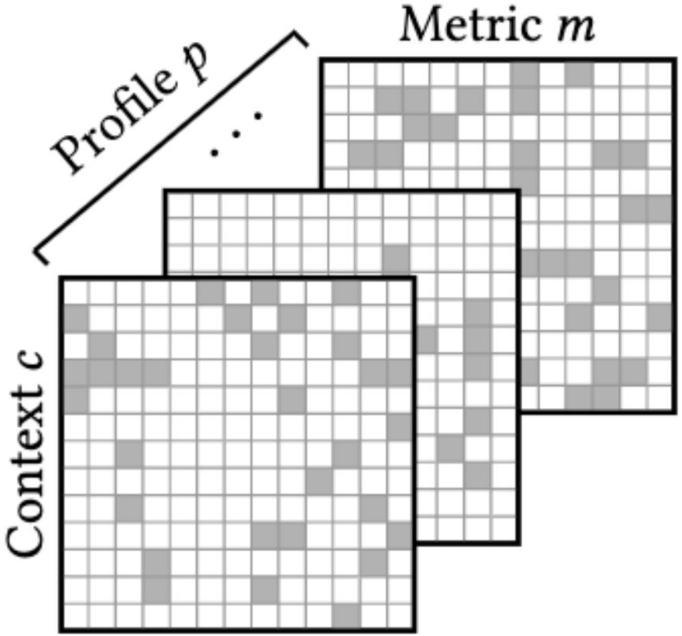
- Threads have different roles, zeros for contexts only in other threads
  - MPI helper threads, OpenMP worker threads, GPU streams…
- Metric costs accumulate in expensive leaf functions
  - Distant callers often have no exclusive metrics
- Many metrics apply only to certain instructions or code regions
  - TLB miss only on memory access; GPU cycles only in GPU code
- Some metrics only apply to very specific contexts
  - Kernel launch parameters only for GPU kernels, data motion volume for GPU copy
- All these factors contribute to sparsity!

Performance data

For each value we record, we need to know:

- which metric
- which calling context
- which thread

≈ 3-dimensional tensor



Sparsity in Practice

	Application	Density (%) Contexts	Metrics
	Measurement data (In)		
CPU	AMG 8K	69.1	100.0
CPU	AMG 8K†	22.7	20.7
GPU	LAMMPS 1K	17.7	1.8
GPU	PeleC 1K	15.8	2.0
	Analysis results (Out)		
CPU	AMG 8K	0.301	0.182
CPU	AMG 8K†	0.059	0.017
GPU	LAMMPS 1K	2.360	1.390
GPU	PeleC 1K	0.599	0.635

Measurement data

With 7 CPU metrics:

- 23% of the contexts have metric values
- 21% of the values for contexts are non-zero

With GPU metrics:

- 16% of the contexts have metric values
- 2% of the values for contexts are non-zero

Analysis results

In our experiments, at most:

- 2.3% of the contexts have metric values
- 1.4% of the values for contexts are non-zero

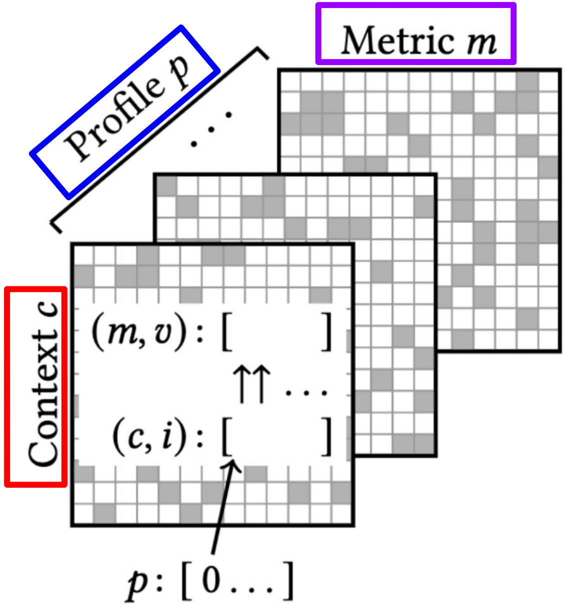
EXASCALE COMPUTING PROJECT

Exploiting Sparsity in Measurement Data

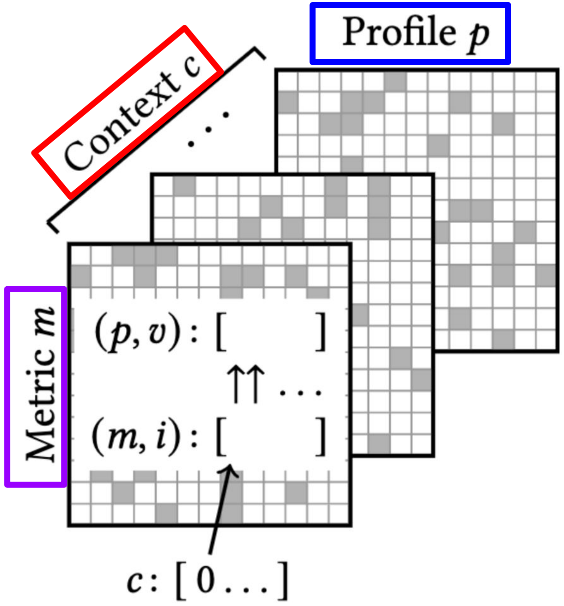
$$
\begin{array}{cc}
 & \text{Metric } m \\
\text{Context } c & \begin{bmatrix} 0 & 5 & 0 & 0 \\ 2 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \end{bmatrix}
\end{array}
\Longrightarrow
\begin{array}{l}
(m, v):\ [\ (1,5)\ \ (0,2)\ \ (2,6)\ \ (1,3)\ ] \\
(c, i):\ [\ (0,0)\ \ (1,1)\ \ (3,3)\ \ (\top,4)\ ]
\end{array}
$$

- Each value v measured for a metric m in a calling context c
  - (m, v) vector lists non-zero metric values
  - (c, i) vector maps contexts to contiguous ranges of the (m, v) vector
- Exploit sparsity in both metric and context dimensions to reduce space
- Ensure logarithmic access time to values

Exploiting Sparsity in Analysis Results



Profile-Major-Sparse (PMS):
data for each profile is contiguous

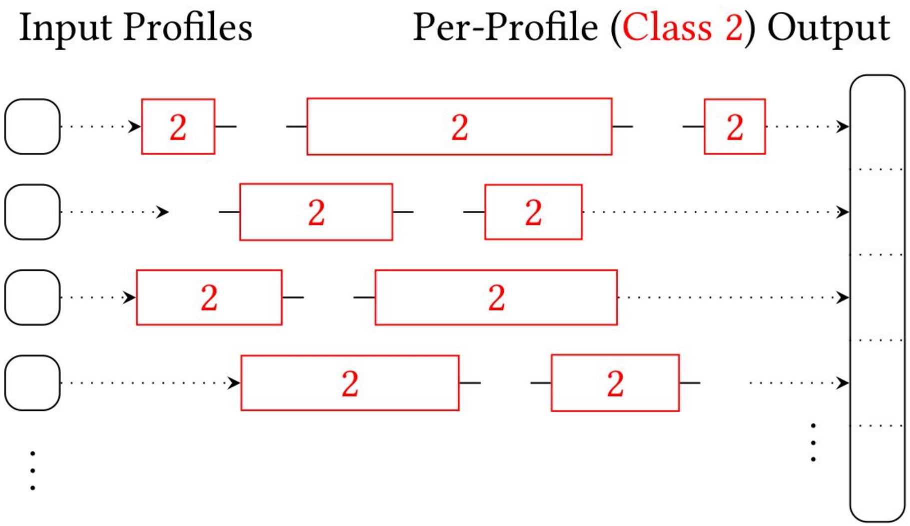


Context-Major-Sparse (CMS):
data for each context is contiguous

EXASCALE COMPUTING PROJECT

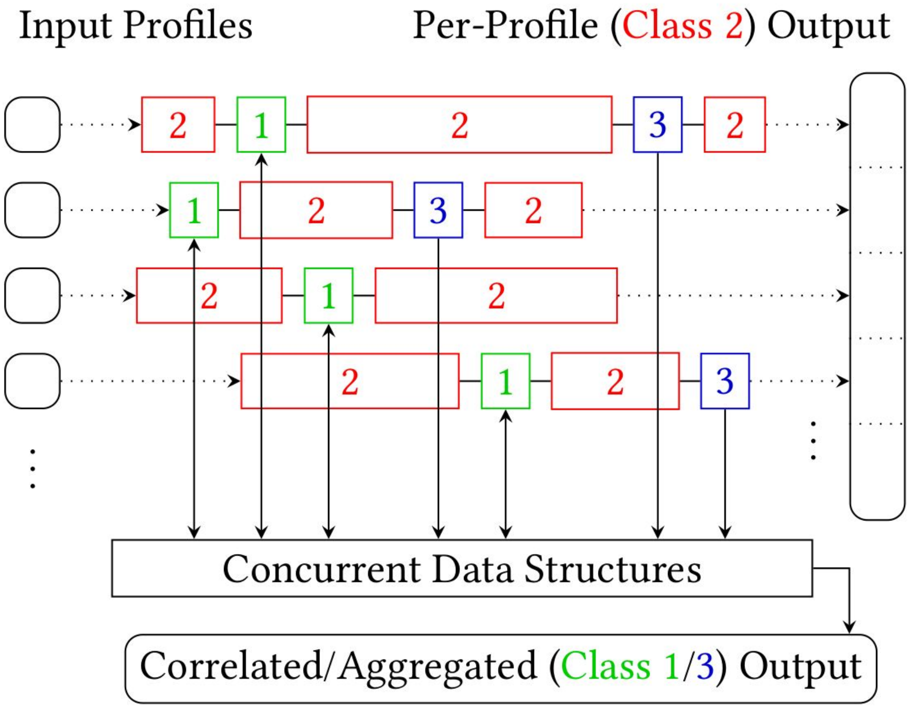
Streaming Aggregation: 3 Classes of Operations

- Data dependency based on number
  - In order: class 1, then 2, then 3
- Class 2 operations run in parallel
  - Independent, per-profile analysis
  - e.g. calculating inclusive cost (per-thread)
  - Read inputs and write outputs directly
  - No significant synchronization
- “Streaming” approach reduces memory
  - Only in-flight profiles in memory!



Streaming Aggregation: 3 Classes of Operations

- Class 1 operations as needed by Class 2
  - Associative, commutative, idempotent
    - e.g. correlate calling contexts across threads
- Class 3 ops aggregate Class 2 results
  - Associative and commutative
    - e.g. summary statistics for entire execution
- Update concurrent data structures
- Streaming parallel approach to “aggregation”
  - No defined order, no phases or barriers
  - Fine-grained, data-centric synchronization
  - Outperforms classic reduction approach!



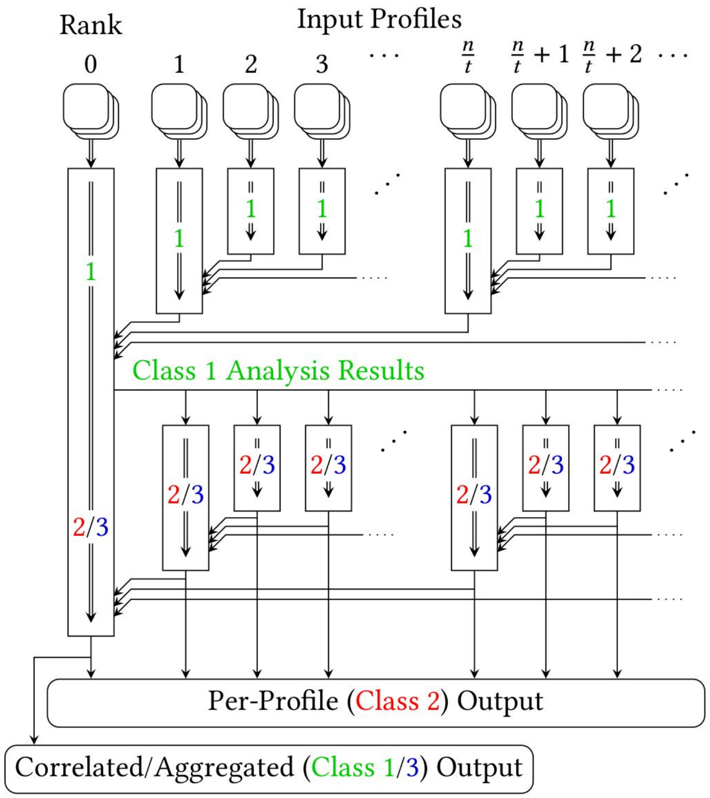
Lessons from a Proxy Application

- Reduction phase causes large serialization: phases are costly
- Lots of allocations also takes time: high footprint costly
- Choice data structure affects performance: choose/tune wisely

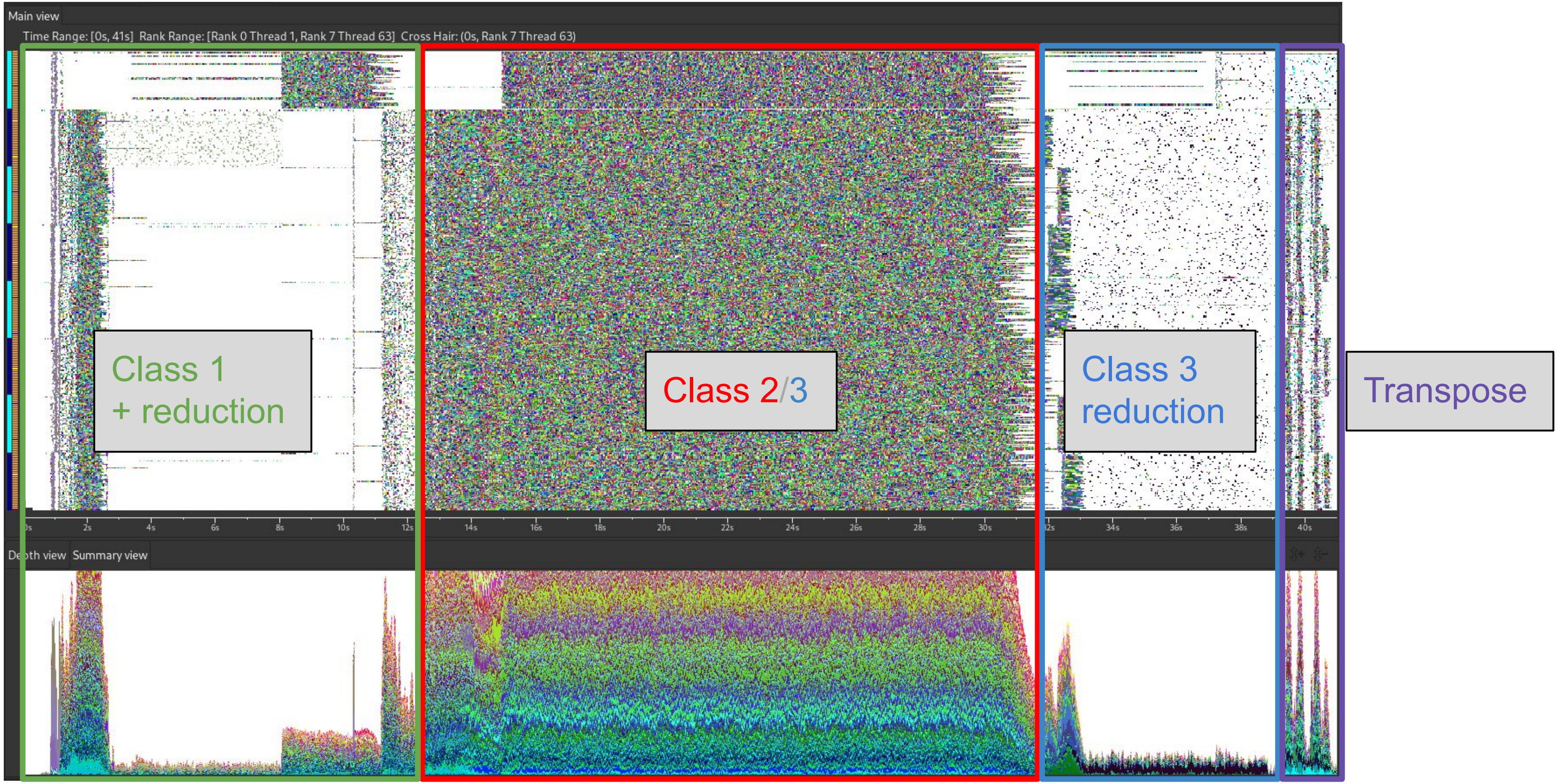
EXASCALE COMPUTING PROJECT

Extension to Multiple Nodes

- Hybrid approach
  - MPI across nodes
  - Streaming Aggregation within
- Class 1/3 ops use reductions across nodes
  - Shared-memory parallelism within a node
- Class 2 ops distributed across nodes
  - Independent, no communication needed
- All nodes perform I/O individually
  - Exploit distributed file system

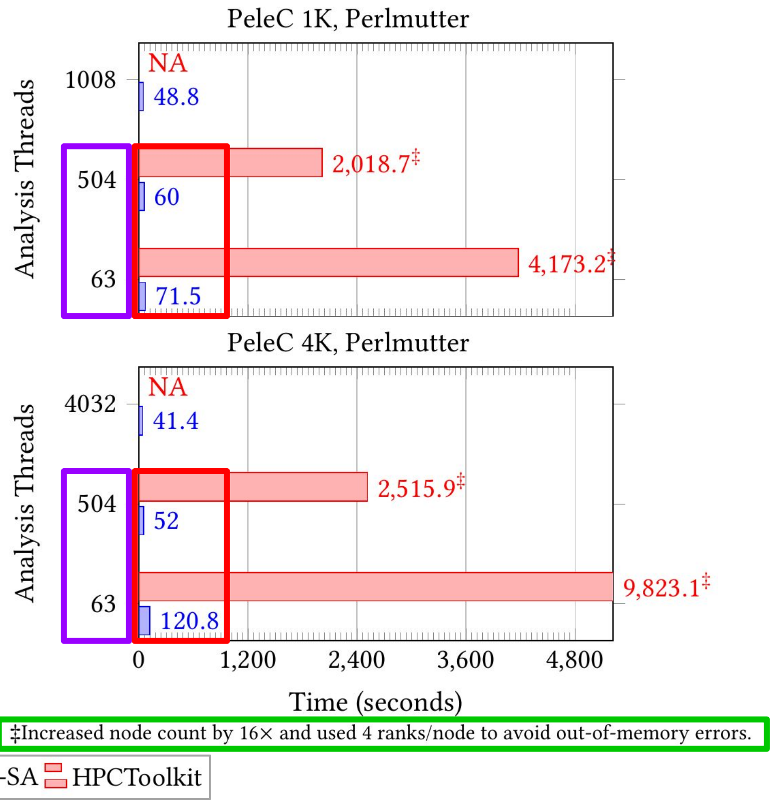


Dense Parallelism from Streaming Aggregation



Performance Improvements Analyzing GPU Measurements

- Post-processing performance from PeleC
  - 512+512 or 2K+2K (threads + GPUs)
  - Analysis uses 63 threads/node
- **30-80x performance improvement**
- **Smaller memory footprint is crucial**
  - Original `hpcprof` exceeded 256GB/node
  - Required 16x total compute resources
- **Minimal resources vs. application**
  - Ran PeleC on 128 and 512 nodes
    - Up to ⅓ of Perlmutter
  - Analyzed measurements on 1 and 8 nodes
- **Results in 1-2 mins**



See ICS paper for CPU-only results for AMG

Storage Improvements

- Performance data from real-world apps
  - Measurements (In) and results (Out)
- **Significant space reductions**
  - Up to 10x compression in measurements
  - Up to 1254x compression in results
- **Results in GBs… not TBs!**

Tool	Size (GiB)			
	AMG 65K		AMG 262K	
	In	Out	In	Out
HPCToolkit	**5.88**	195	**36.9**	1250
HPCToolkit-SA	6.14	**5.55**	38.2	**36.6**
	PeleC 1K		PeleC 4K	
	In	Out	In	Out
HPCToolkit	21.2	4800	50.7	14300
HPCToolkit-SA	**2.85**	**5.27**	**4.8**	**11.4**

Conclusions

- “Less is more”: Efficient use of compute resources is key
  - Exploit sparsity to reduce storage and I/O
  - Exploit multithreading to efficiently use each node’s cores and threads
  - Exploit distributed memory parallelism for scale
- Novel sparse formats for performance measurements and analysis results
  - Up to 1254x compression in analysis results for our experiments
- Novel highly-parallel streaming aggregation approach to performance analysis
  - Results for large-scale executions in minutes!
- End result: better prepared for performance analysis at exascale!
- Improvements actively being integrated into HPCToolkit
  - Will become widely available in a future release

Ongoing Work: Accelerating I/O

Problems

- Metadata server is slow:
  - Our approach of recording 2 files per thread may be costly
- Unnecessary overhead for maintaining page cache consistency between multiple processes on different nodes
  - Our writes don’t overlap

Approach

- When available, exploit object-based solid state storage for ultimate performance

Ongoing work: **D**istributed **A**synchronous **O**bject **S**torage (DAOS)

- Designed for massively distributed NVM
- Affordable, fast, large-capacity PM
    - SCM (store metadata, latency-sensitive small data)
    - NVMe SSDs (bulk data)

Properties

- High throughput and IOPS at arbitrary alignment and size
- Fine-grained I/O operations with true zero-copy I/O to SCM
- Non-blocking data and metadata operations to allow I/O and computation to overlap
- Scalable distributed transactions with guaranteed data consistency and automated recovery

An I/O abstraction layer for HPCToolkit

- Freedom to choose any I/O option (DAOS, POSIX, Lustre …) for any file
  - Example:
    - `hpcprof -o daos://<POOL>/<CONT>/database-dir measurement-dir`
    - We use DAOS for database and use POSIX for measurement files at the same time
- Easy integration of other I/O options in the future
  - Example: HPE Rabbit Near Node Storage (Livermore’s El Capitan), Lustre ...
- Details of I/O are invisible to other parts of the software
  - Example: just call `io->write(...)` without concern for underlying I/O implementations

HPCToolkit Funding Acknowledgments

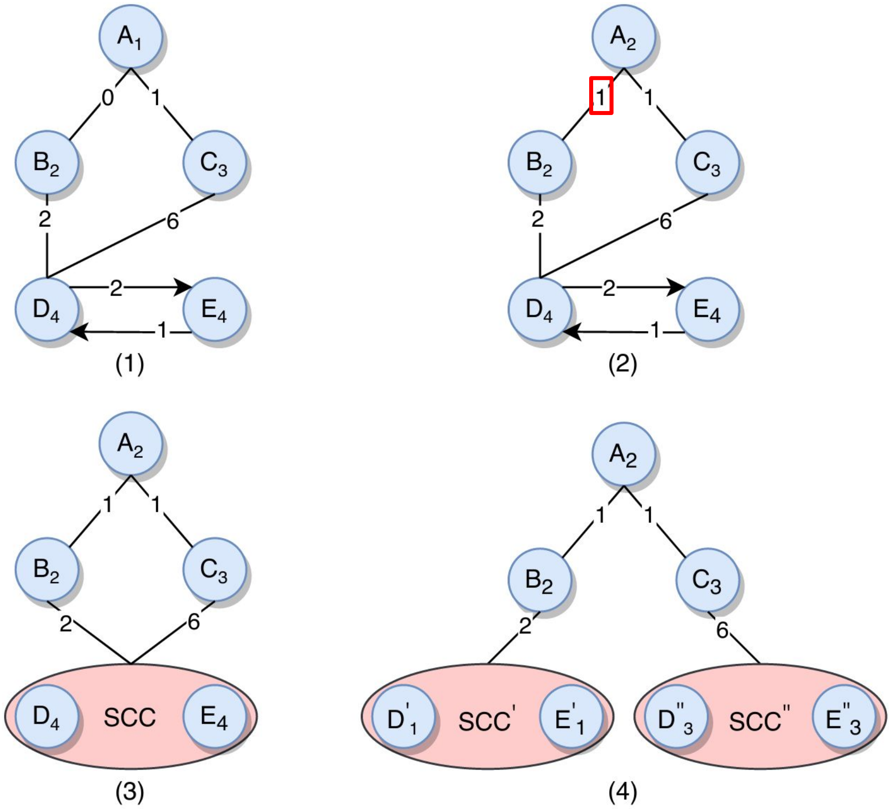
- Exascale Computing Project 17-SC-20-SC
- Lawrence Livermore National Laboratory Subcontract B645220
- Argonne National Laboratory Subcontract 9F-60073
- Advanced Micro Devices
- Intel Corporation
- TotalEnergies EP Research & Technology USA, LLC.

EXASCALE COMPUTING PROJECT

Backup Slides

Adapting Algorithms to New Constraints

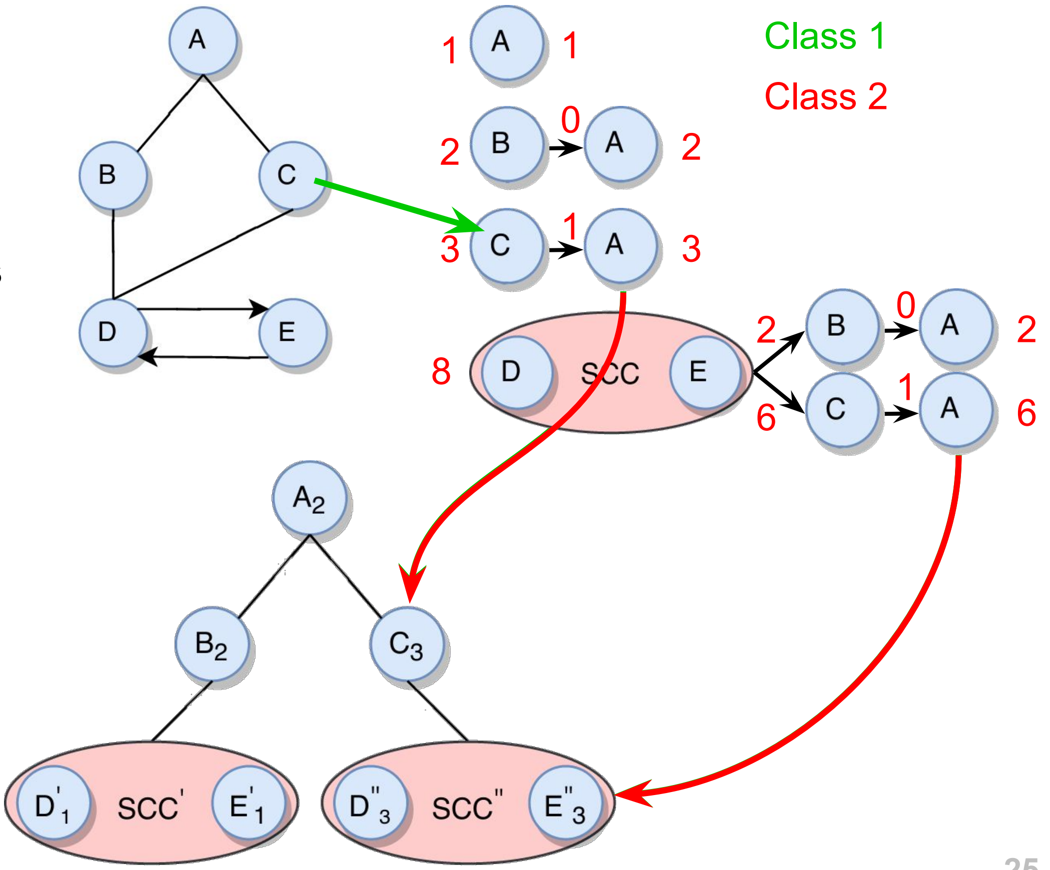
- E.g. GPU calling context reconstruction
  - GPU PC sampling is always flat
  - Reconstruct calling context based on static CFG from binary analysis
- 4-step algorithm:
  - Attribute flat samples to call graph
  - Fixup missing edge weights
  - Convert to DAG by SCC
  - Expand DAG into calling context tree
- Problem: does not obey S.A. constraints!
  - (4) must be class 1 for final CCT…
  - …but (1) must be class 2 for values!



Source: *Measurement and analysis of GPU-accelerated applications with HPCToolkit*, https://doi.org/10.1016/j.parco.2021.102837

Adapting Algorithms to New Constraints

- Generate full possible CCT (Class 1)
  - Iterate reverse static call paths
  - Create full context for each path
- Reattribute metric values (Class 2)
  - Attribute flat samples to heads
  - Distribute among branching paths
  - Assign path values to CCT
- Reconstruction mixed with normal ops
  - Generate when a context in need of reconstruction is parsed
  - Reattribute just before inclusive metric propagation



DAOS Integration Experiences - I

No matching DAOS function for every POSIX function

- `fread, fwrite, fseek` ...
  - `fread`: We reworked our code to avoid it by using `read_at`
  - `fwrite`: DAOS only has `write_at`, manage our own buffer and cursor
  - `fseek`: We reworked our code to avoid it by using `read_at` and `write_at`
- `write`
  - Multiple processes append to the same log file (debug info for developers)
  - With `write_at`, we need to share a buffer or a cursor between processes
  - We reworked so that each process creates its own log file
- C++ filesystem abstractions (`directory_iterator`, `exists`, `remove_all` ...)
  - We needed to reimplement these abstractions using our I/O abstraction layer

DAOS Integration Experiences - II

- Extra initialization and finalization steps
  - Each process initializes its own DAOS pool and container handles
  - We set up the output directory and begin recording trace data BEFORE MPI initialization
- Need to manage file system accesses carefully
  - Before: use POSIX anywhere without any direct coordination
  - Now: pass around I/O abstractions to access the correct file system
- Caching semantics
  - Objects created were only visible through command line after cache refreshed
  - Using `--disable-caching` can help, but may hurt performance