What's New in Proxy Apps

ECP Annual Meeting 2019

David Richards
LLNL

Houston, TX
15 January 2019

ECP Proxy App Project: Objectives and Scope

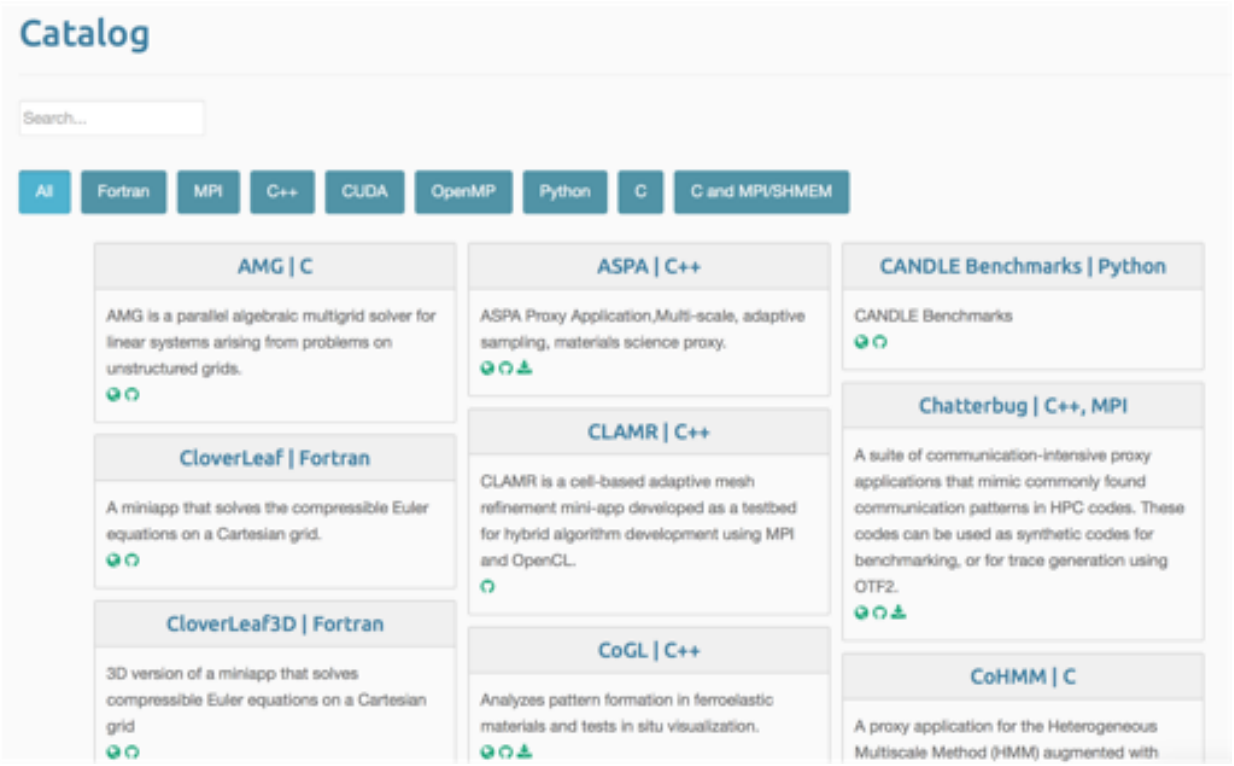
- Assemble and curate a proxy app suite that represents the most important features (especially performance) of exascale applications.
- Improve the quality of proxies created by ECP and maximize the benefit received from their use.
  - Set standards for documentation, build and test systems, performance models, etc.
- Assess the representativeness of proxies
- Project team members at LLNL, LANL, SNL, LBNL, ORNL, ANL

Over 50 proxy apps can be installed using Spack

- Builds multiple versions, configurations, and compilers
- Encourages “standard” build recipes that conform to common build systems (e.g. Autotools, CMake, Makefile)
- Ease of aggregation
  - `spack fetch <package_name>` to just download the vanilla source
  - `spack install --source <package_name>` to just download the possibly patched source
- Proxies can be build as a whole or separately the “Spack” way
  - Different compilers and with or without system libraries
- Currently using Spack for continuous integration

Spack

We maintain a catalog of over 50 proxy apps



- Work underway on collecting/distributing multiple versions of proxies
  - Issues with maintainability, consistency, releasability, etc.
- Contributions are welcome.
  - Submit your proxy information through the proxy app website

https://proxyapps.exascaleproject.org

EXASCALE COMPUTING PROJECT

We released version 2.0 of the ECP Proxy App Suite in Oct 2018

Proxy	Language	GPU		Proxy	Language	GPU
AMG	C	No public version		miniVite	C++	
CANDLE Benchmarks	Python			NEKbone	Fortran	OpenACC, OpenACC+CUDA
Ember	MPI/SHMEM	N/A		PICSARlite	Fortran	
ExaMiniMD	C++	Kokkos		SW4lite	C, C++, Fortran	CUDA, RAJA
Laghos	C++	CUDA, RAJA, OCCA		SWFFT	C	
MACSio	C	N/A		thornado-mini	Fortran	
miniAMR	C	No public version		XSBench	C	OpenACC, OpenCL, AMP
miniQMC	C++	Kokkos OpenMP				

Acknowledgements

Slides & information for discussion of new mini-apps provided by:

- miniVite: Mahantesh Halappanavar, Sayan Ghosh
- miniQMC: Paul Kent
- thornado-mini: Bronson Messer
- PICSARlite: Jean-Luc Vay
- Ember: Si Hammond

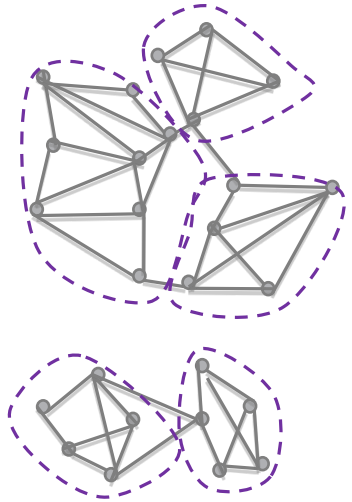
Pacific Northwest NATIONAL LABORATORY

Graph Clustering (community detection)

- Problem: Given $G(V,E,\omega)$, identify tightly knit groups of vertices that strongly correlate to one another within their group, and sparsely so, outside.

Input :

- $V = \{1,2,\ldots n\}$
- E: a set of M edges
- $\omega(e)$: weight of edge e (non-negative)
- $m = \sum_{\forall e \in E} \omega(e)$

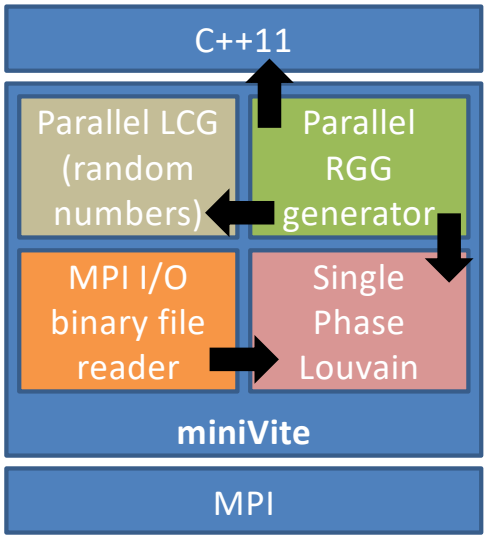


Output :

- A partitioning of V into k mutually disjoint clusters $P = \{C_1, C_2, \ldots C_k\}$ such that: … ?

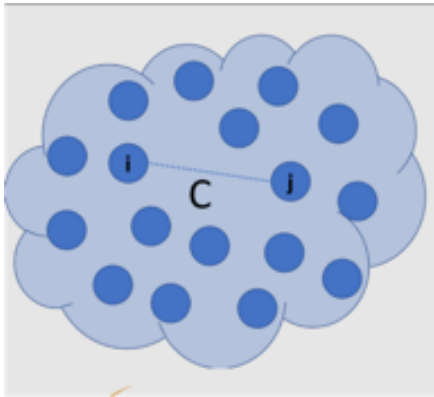
miniVite (/'vi:te/)

- Implements a single phase of Louvain method (without rebuilding the graph)
- Capable of generating synthetic Random Geometric Graphs (RGG) in parallel (needs random numbers)
  - Can also add random edges across processes
- Can also use real world graphs as input (have to convert to a binary format first)
- Parts of code has multiple communication options (can be selected at compile time) – Sendrecv, NB Isend/Irecv (default), MPI RMA and Collectives
- About 3K LoC



Graph clustering OR Community detection

- **Communities**: Nodes/vertices of most real-world network/graphs tend to be organized into tightly-knit modules known as *communities* or *clusters*
- In 2008, Blondel, et al. introduced a multi-phase, iterative heuristic for modularity optimization, called the ***Louvain method***
- Goodness of partitioning into communities is typically measured using a global metric called ***modularity***
  - Depends on sum of edge weights between vertices within a community (*e_ij*), and sum of weights incident upon a community (*a_c*)



$$Q = \sum_{c \in C} \left[\frac{e_{ij}}{2m} - \left(\frac{a_c}{2m} \right)^2 \right]$$

where:

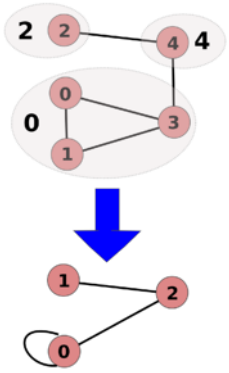
$$e_{ij} = \sum w_{ij} : \forall i, j \in c, \text{ and } \{i, j\} \in E$$

$$a_c = \sum_{i \in c} k_i$$

Louvain algorithm

Input: Graph $G = (V, E)$, threshold τ, Initial community assignment, C_{init}

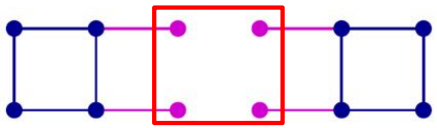
1: $Q_{prev} \leftarrow -\infty$
2: $C_{prev} \leftarrow$ Initialize each vertex in its own community
3: **while** true **do**
4: **for all** $v \in V$ **do**
5: $N(v) \leftarrow$ neighboring communities of v
6: $targetComm \leftarrow \arg\max_{t \in N_v} \Delta Q(v \text{ moving to } t)$
7: **if** $\Delta Q > 0$ **then**
8: Move v to $targetComm$ and update C_{curr}
9: $Q_{curr} \leftarrow ComputeModularity(V, E, C_{curr})$
10: **if** $Q_{curr} - Q_{prev} \leq \tau$ **then**
11: **break**
12: **else**
13: $Q_{prev} \leftarrow Q_{curr}$



Rebuilding is nontrivial, may takes ~1-10% of the overall phase time depending on input graph and volume of vertex movement

- Initially each vertex assigned to a separate community
- Within each iteration
  - Compute delta Q when a vertex migrates
  - Move vertex from current community to one that yields max delta Q
- Phase continues until delta Q between successive iterations is below a threshold
- At the end of each phase, graph is rebuilt

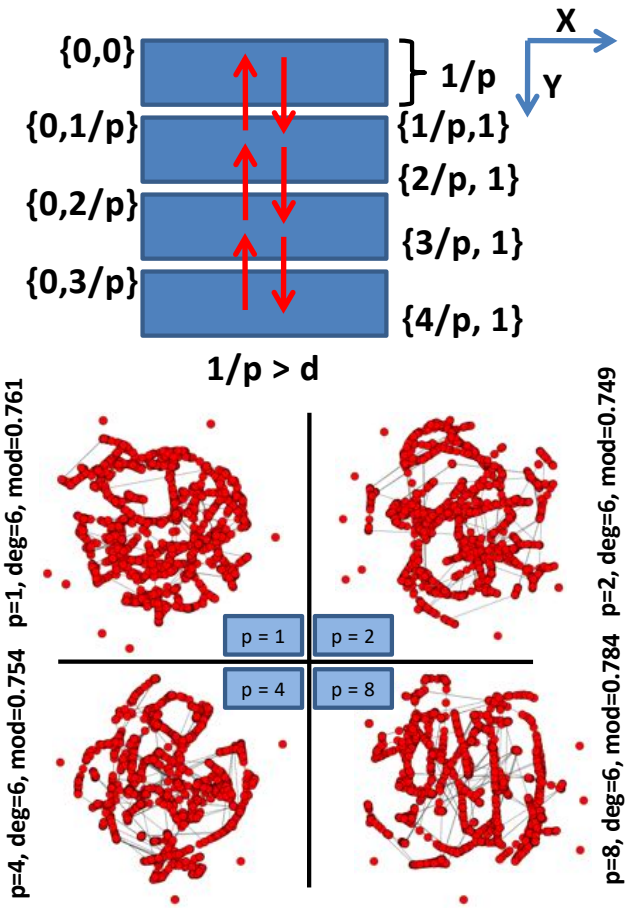
Simple vertex-based data distribution



ghosts/cross edges

Parallel Random Geometric Graph (RGG) generator

- Constructed by randomly placing *N* nodes within a unit square – only add an edge between two vertices if their distance is within a range *d*
  - Basically, generate random numbers within (0,1) and compute euclidean distance between two points, if it is less than *d*, add an edge
  - RGG is known to demonstrate community structure (high modularity)
- We distribute equal number of vertices across processes, each process may have (cross) edges with its up or down neighbor
- Option to add random number of (cross) edges across processes that are far apart



Results

FIRST PHASE OF LOUVAIN METHOD VS THE LAST PHASE FOR REAL WORLD INPUTS ON 1K PROCESSES OF NERSC CORI.

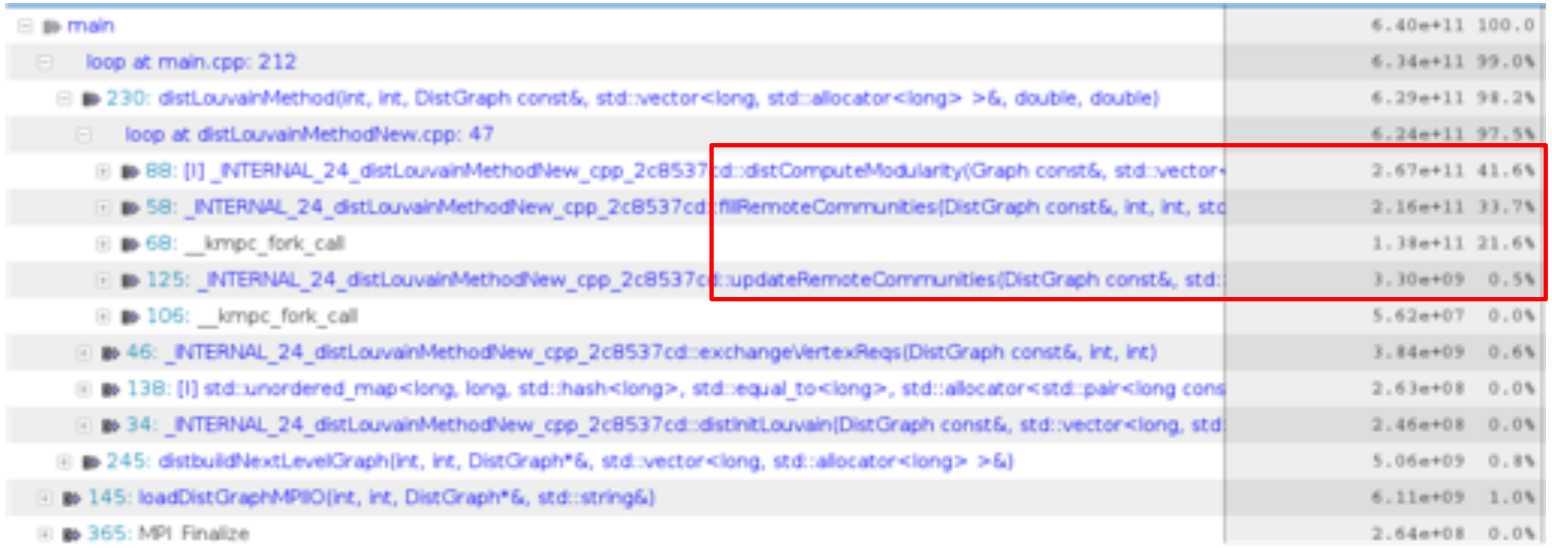
Graphs	#Vertices	#Edges	Modularity	First phase		Final		
				Iterations	Time (in s)	Phases	Iterations	Time (in s)
friendster	65.6M	1.8B	0.624	143	565.201	3	440	567.173
it-2004	41.3M	1.15B	0.973	14	45.064	4	91	45.849
nlpkkt	27.9M	401.2M	0.939	3	3.57	5	832	21.084
sk-2005	50.6M	1.9B	0.971	11	71.562	4	83	72.94
orkut	3M	117.1M	0.472	89	59.5	3	281	59.64
sinaweibo	58.6M	261.3M	0.482	3	270.254	4	108	281.216
twitter-2010	21.2M	265M	0.478	3	209.385	4	184	386.483
uk2007	105.8M	3.3B	0.972	9	35.174	6	139	37.9887
web-cc12-paylvladmin	42.8M	1.2B	0.687	31	140.493	4	159	146.92
webbase-2001	118M	1B	0.983	14	14.702	7	239	24.455

For a number of real world graphs, first phase of Louvain method does the most work (little difference between first and final phase)

Processes	NB Send/Recv	Collectives	Sendrecv	RMA
512	7.48492	7.35221	11.2029	7.39827
1024	6.52832	5.56177	13.2942	5.93101

RGG of 134.2M vertices and 1.6B edges on NERSC Cori

Performance analysis (I)



1. **HPCToolkit profiling shows over 60% of time spent in managing and communicating vertex-community information**
2. **About 40% is spent on global communication (MPI_Allreduce) for computing modularity**

QMCPACK https://qmcpack.org

Developed and released via github.com/QMCPACK/qmcpack

Pull request reviews, continuous integration & integration testing.

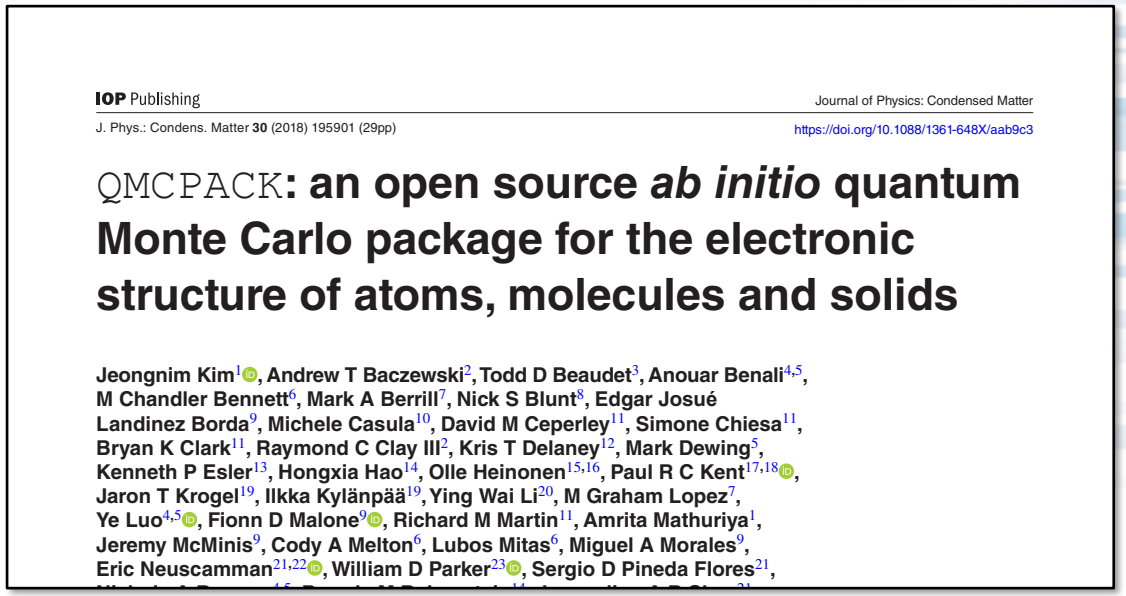
UIUC/NCSA open source license.

~330K code lines, <100K “core”

C++11 (14 soon), MPI, HDF5, XML, FFTW.

OpenMP (walker level only) and CUDA (old) on node. **Different vectorization / parallelization scheme for GPU & CPU**. Major development + maintenance headache.

Miniapps: miniqmc, miniafqmc under github.com/QMCPACK . Use for prototyping, experimenting with individual kernels etc.



IOP Publishing — Journal of Physics: Condensed Matter

J. Phys.: Condens. Matter **30** (2018) 195901 (29pp) https://doi.org/10.1088/1361-648X/aab9c3

QMCPACK: an open source *ab initio* quantum Monte Carlo package for the electronic structure of atoms, molecules and solids

Jeongnim Kim[1], Andrew T Baczewski[2], Todd D Beaudet[3], Anouar Benali[4,5], M Chandler Bennett[6], Mark A Berrill[7], Nick S Blunt[8], Edgar Josué Landinez Borda[9], Michele Casula[10], David M Ceperley[11], Simone Chiesa[11], Bryan K Clark[11], Raymond C Clay III[2], Kris T Delaney[12], Mark Dewing[5], Kenneth P Esler[13], Hongxia Hao[14], Olle Heinonen[15,16], Paul R C Kent[17,18], Jaron T Krogel[19], Ilkka Kylänpää[19], Ying Wai Li[20], M Graham Lopez[7], Ye Luo[4,5], Fionn D Malone[9], Richard M Martin[11], Amrita Mathuriya[1], Jeremy McMinis[9], Cody A Melton[6], Lubos Mitas[6], Miguel A Morales[9], Eric Neuscamman[21,22], William D Parker[23], Sergio D Pineda Flores[21],

Citation paper with high level method & algorithm descriptions
J. Kim et al, Journal of Physics: Condensed Matter **30** 195901 (2018)
https://doi.org/10.1088/1361-648X/aab9c3

Open access this year

High-level Algorithms

Today, GPU operates on walkers (Markov chains) in a “batch” to reduce kernel launch overhead, particularly for small problems (N<few 100). CUDA kernels operate on multiple walkers at once.

Drawbacks: Multiple drivers, Extra APIs through entire app, Code divergence, Memory bandwidth still stressed...

CPU Algorithm

```
do time step i   [ 1K-100K per MPI task ]
        do walker j  [ 1 per core, OpenMP ]
                do electron k [ 0.1-10K ]
                        do component l  [ 3-4 ]
                                advance WF
                        end l
                end k
                evaluate Hamiltonian
        end j
  spawn/kill walkers, load balance
end i
```

GPU Algorithm: 1 MPI task/GPU

```
do time step i

    do electron k (lock step)
        do component l
            advance WF of all the walkers [10-1000]
        end l
    end k
    evaluate Hamiltonian of all the walkers
    spawn/kill walkers, load balance
end i
```

miniQMC

- miniQMC is a mini-app for QMCPack, which is a Quantum Monte Carlo code
  - The goal of QMCPack is to provide highly accurate calculations of the properties of complex materials
- Computational motifs include: particle methods, dense and sparse linear algebra, and Monte Carlo
- The version we are using is based on version 3.1.0 of QMCPack

miniQMC computation

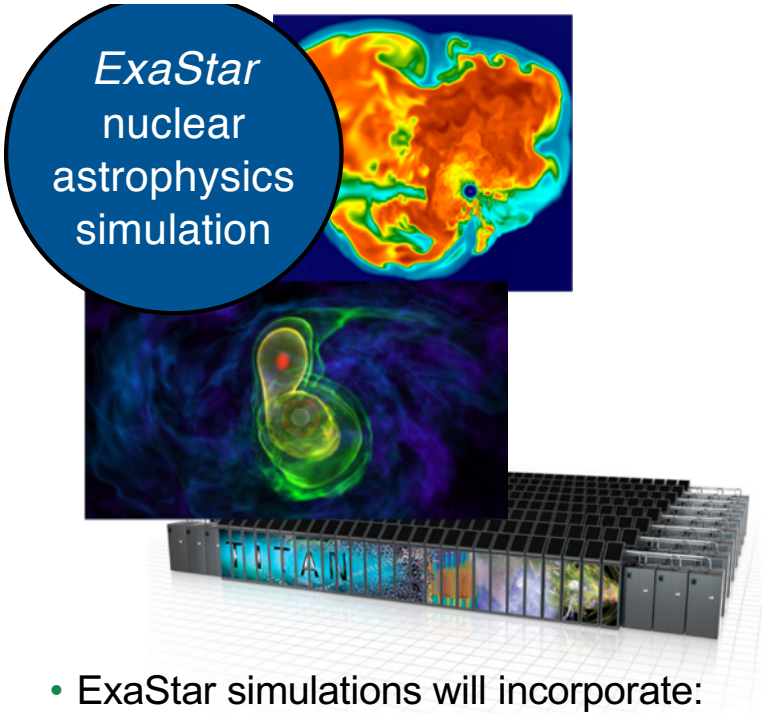
- QMCPack utilizes an ensemble of walkers which represent particle positions and are moved through space by a drift-diffusion process
- The miniQMC calculation utilizes one walker per rank with a predefined problem to focus on single node performance
- The size of the problem space for the walker in miniQMC can scaled at runtime by replicating the space by tiling
- There is no MPI communication in miniQMC since the mini-app is focused on the computational portion of the app

Significant kernels

- Update – Sherman-Morrison rank-1 NxN matrix inverse update

  Strongly memory bandwidth limited. Source of N^3 scaling.

- Spline – 3D spline value & gradients 4x4x4xN stencil, membw limited
- Jastrow functions – Small classical MD force-field like polynomial evaluations using distances+cutoffs from electron positions. One and two-body forms most common.
- Distance tables – Interparticle distances with minimum image convention/periodic boundary conditions applied.
- Others, depending on threadability of above.



- ExaStar simulations will incorporate:
  - experimental nuclear physics data
  - satellite observations of astrophysical phenomena
  - GW detections
  - neutrino experimental data, including solar and reactor experiments

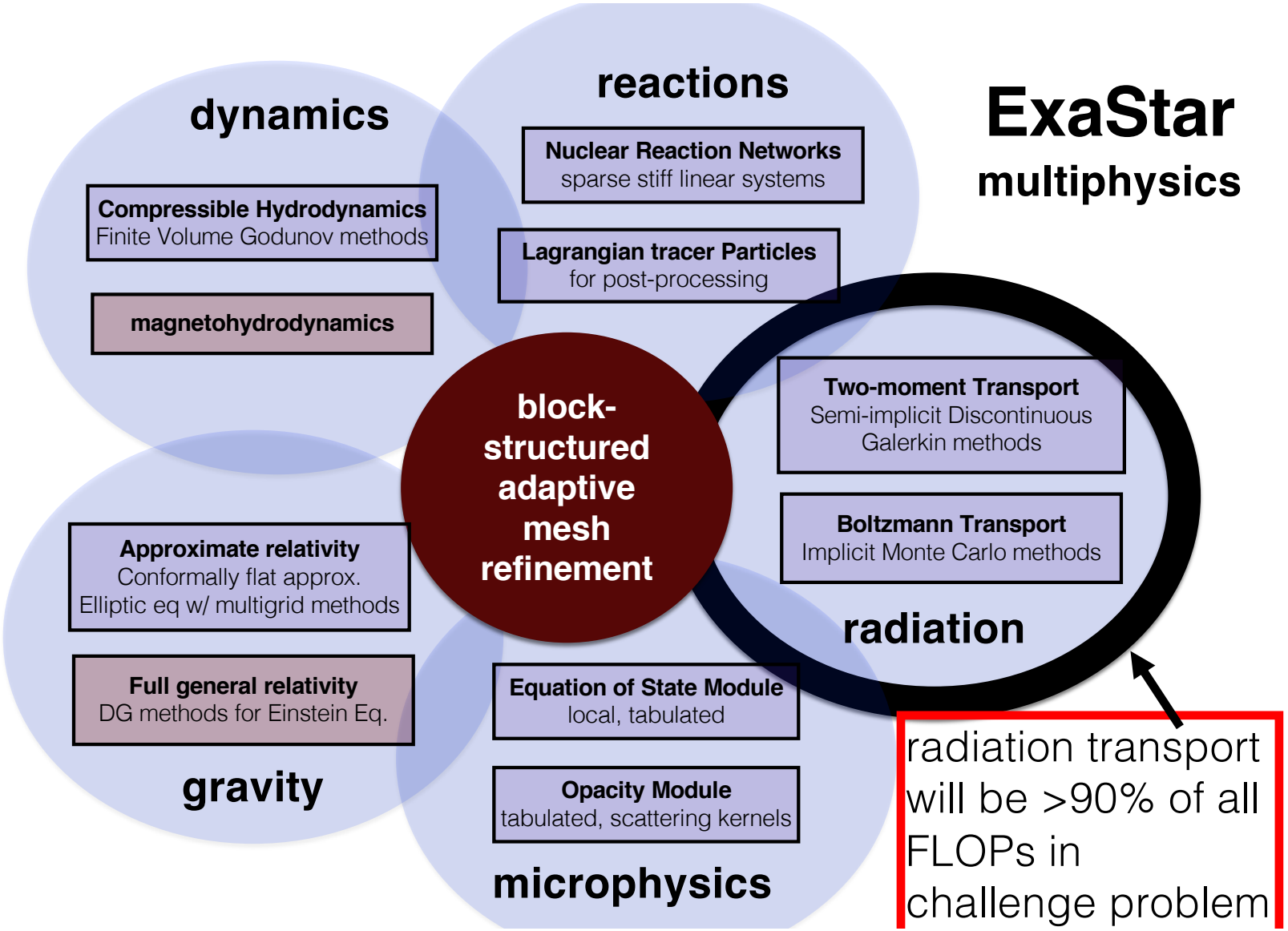
to improve predictive power

- ExaStar simulations are essential to:
  - Guide future nuclear physics experimental programs
    - siting the r-process directly impacts which rates are most important to measure
  - Provide reliable templates for gravitational wave and neutrino detectors
    - Low signal-to-noise requires templates for matching
  - Interpret X-ray and gamma-ray observations

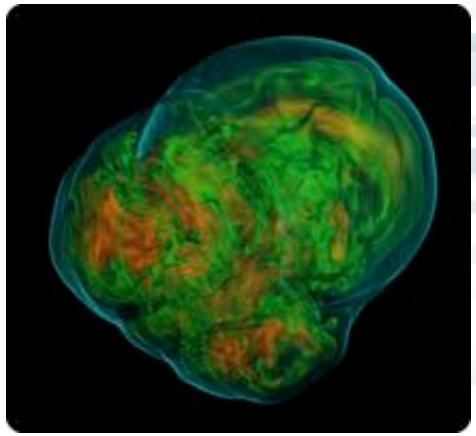
DUNE DEEP UNDERGROUND NEUTRINO EXPERIMENT

LIGO

FRIB

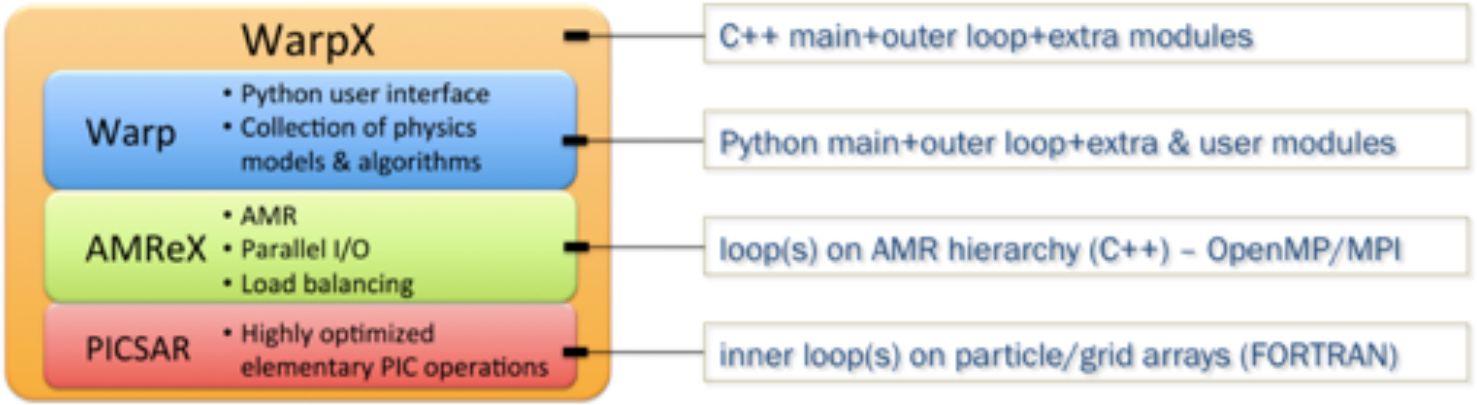

ExaStar
multiphysics
dynamics
Compressible Hydrodynamics
Finite Volume Godunov methods
magnetohydrodynamics
reactions
Nuclear Reaction Networks
sparse stiff linear systems
Lagrangian tracer Particles
for post-processing
block-structured adaptive mesh refinement
Two-moment Transport
Semi-implicit Discontinuous Galerkin methods
Boltzmann Transport
Implicit Monte Carlo methods
radiation
Approximate relativity
Conformally flat approx.
Elliptic eq w/ multigrid methods
Full general relativity
DG methods for Einstein Eq.
gravity
Equation of State Module
local, tabulated
Opacity Module
tabulated, scattering kernels
microphysics
radiation transport will be >90% of all FLOPs in challenge problem

thornado-mini represents neutrino radiation transport (80-90% of FLOPS in ExaStar challenge problem)



- Key computational motifs are structured mesh, dense linear algebra
- The problem size is defined by the number of elements used, the number of energy groups, and the resolution of the interaction table used.
  - The problem solved in the mini-app (Deleptonization) uses production values for elements and energy groups (2 and 20, respectively), but the table is about quarter-resolution in the three dimensions.
- The mini-app is, in fact, the development vehicle for the finite-element transport algorithms, the mini-app is slowly updated with new implementations. GPU-enabled version (1.1) will be available in March, 2019.

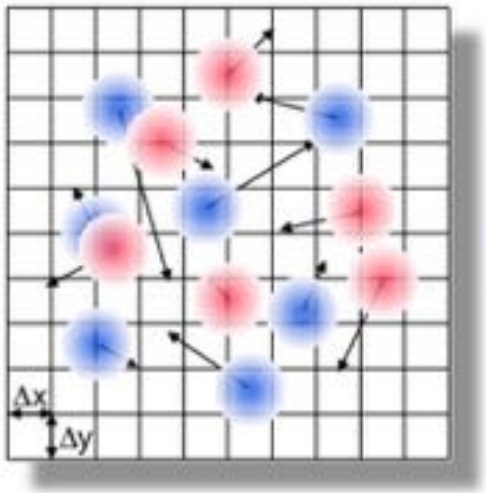
WarpX: Exascale Modeling of Advanced Particle Accelerators

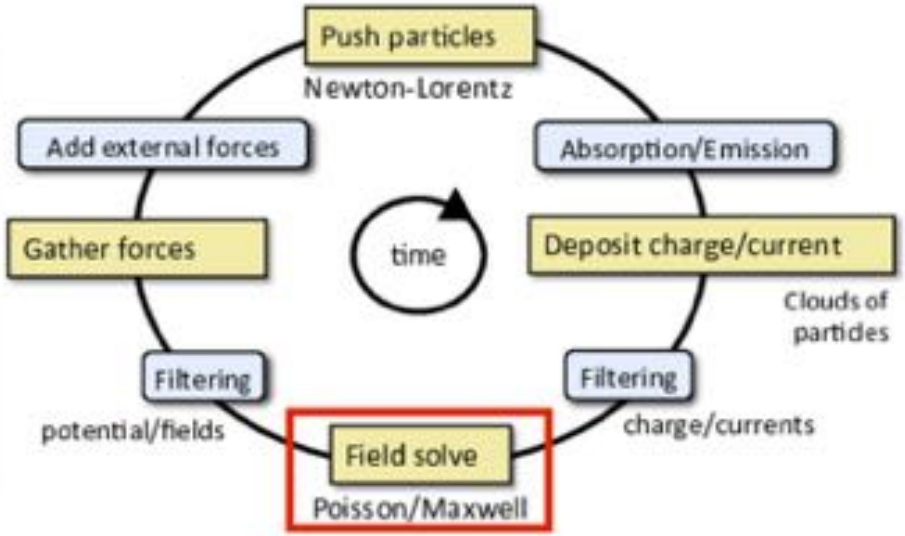


WarpX has code connecting the 3 components

- **Warp**: original Python code (no longer needed!)
- **AMReX**: data and communication
- **PICSAR**: kernels, also a self-contained mini-app

The Particle-In-Cell (PIC) method





Different regimes:

- Particle-dominated, >>1 particle/cell
- Commensurate, ~1 particle/cell
- Field-dominated, <<1 particle/cell

Time breakdown on 1 KNL node, 50 particles/cell

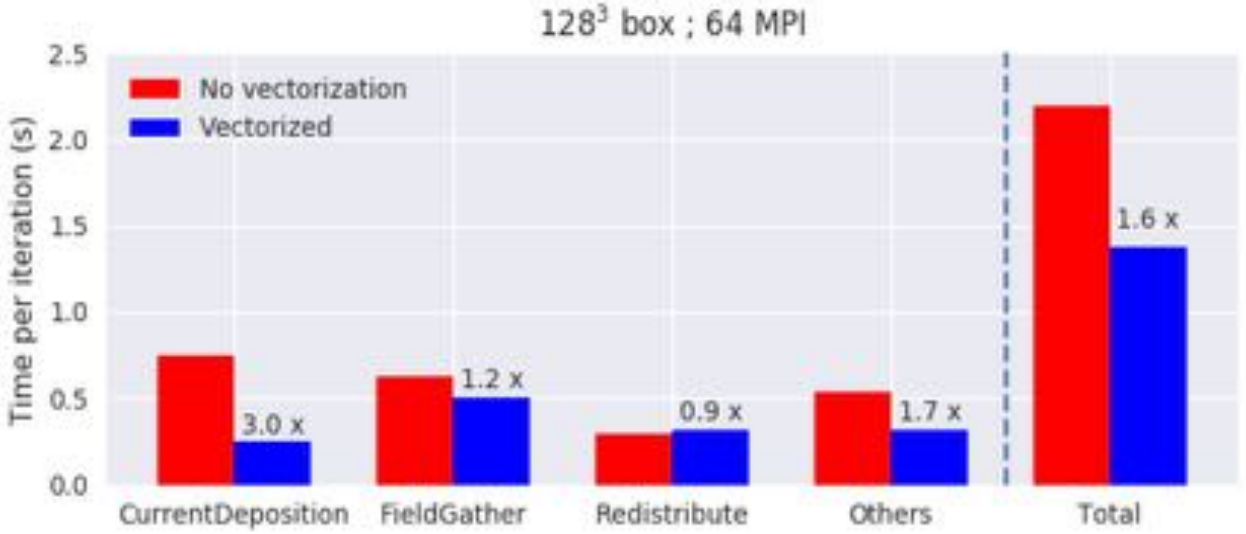


Figure 9: Time taken by the most time-consuming routines (current deposition, field gather, and particle redistribution), as well as by the lumped sum of the other routines ("Others"). The numbers above each blue bar indicates the speedup due to vectorization.

Report is on Confluence: **ADSE06-WarpX_Milestone_report_FY18.2.pdf**

Roofline model on 1 KNL node, 32 particles/cell

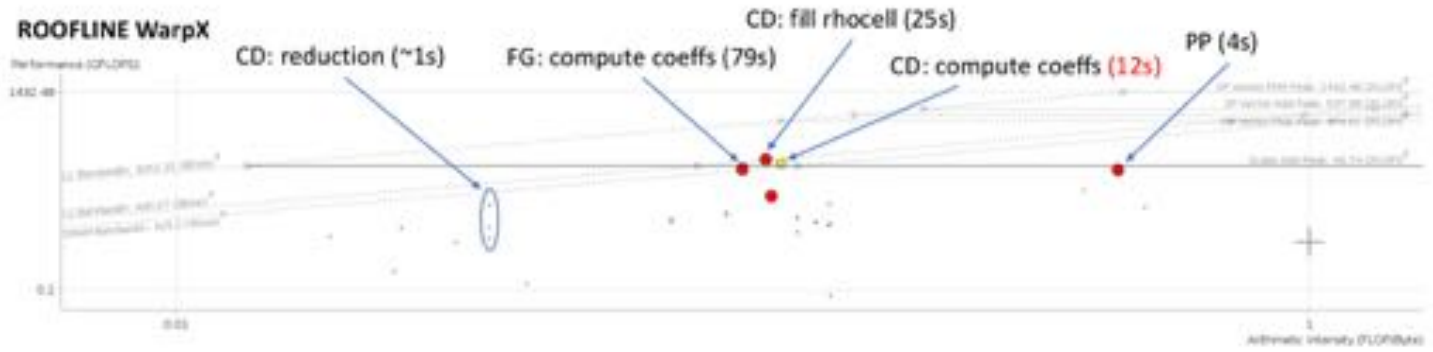


Figure 12: Roofline model for WarpX for a simulation with $128 \times 128 \times 128$ cells and 32 particles per cell on a single Cori KNL node with 64 OpenMP threads. The particle shape factor is 3. This image was obtained using Intel Advisor.

PICSAR kernels mostly memory-bound.
Shown here:
Charge Deposition, Field Gather, Particle Push.

PICSARlite was created to simplify looking a PIC challenges

- Reduced loc from over 80,000 to under 10,000
  - Most steps have multiple options or algorithms. Simplified to only one.
  - Led to script that will automatically reduce PICSAR to a smaller code base containing only the selected methods
- WarpX team is interested in reducing communication:
  - merging messages
  - overlapping communication with computation
- WarpX team has a mini version of WarpX (using only simplest version of each PIC kernel) implemented on SummitDev.

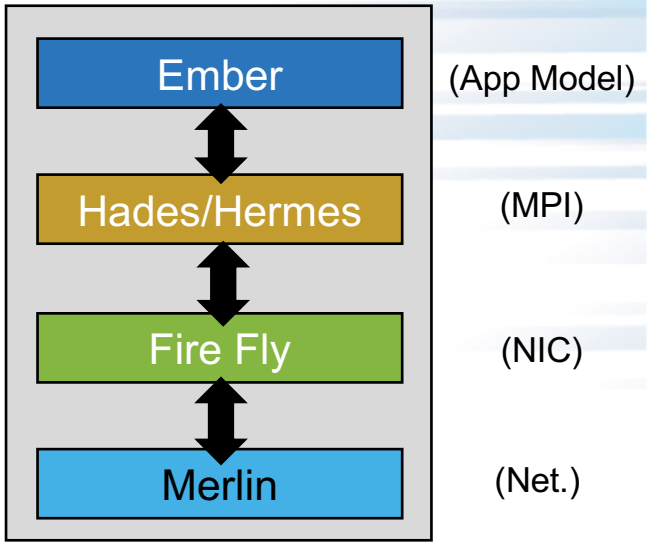
Modeling Communication Patterns for Exascale

- Exascale systems are likely to be some of the largest HPC machines ever built
  - Implies larger scale and higher performance demands for network interconnects
  - Additional opportunities to add hardware support for some communication operations to reduce latency, improve operation throughput and increase performance

- What we (HPC community, vendors etc) need are:
  - Good models of primitive communication patterns that HPC codes routinely execute (so we can analyze the impact of hardware changes on full system performance)
  - Methods to easily scale communication models so we can evaluate broad range of Exascale system options (traces lock us in to specific rank counts and configurations)
  - Flexible parameterization of communication patterns to reflect choices in decomposition that could be important (e.g. when using a GPU versus a CPU versus a …)
  - Note: we cannot represent *every* communication pattern in *every* application, we need to capture a small subset from which important characteristics of network performance can be modeled in order to support tractable analysis of future systems

Ember Communication Patterns

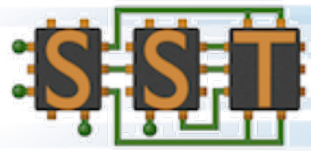
- Ember is a suite of communication patterns that have been developed since 2012 in collaboration with leading industry vendors (Cray, IBM, Intel, HPE, ..)
  - Designed originally to work in the SST Simulator as a scalable model of DOE workloads (https://github.com/sstsimulator)
  - Highly parameterizable to easily replicate behavior across range of DOE code bases
  - Flexible enough to scale from small node counts to over a million simulated MPI ranks (much easier to use than communication traces)
  - Can encode complex, dynamic behavior which traces cannot capture
  - Generic enough to work in any simulation environment



- **For ECP Proxy Applications several communication patterns have been turned into simple, equivalent MPI/SHMEM drivers to run on HPC systems**

http://github.com/sstsimulator/ember

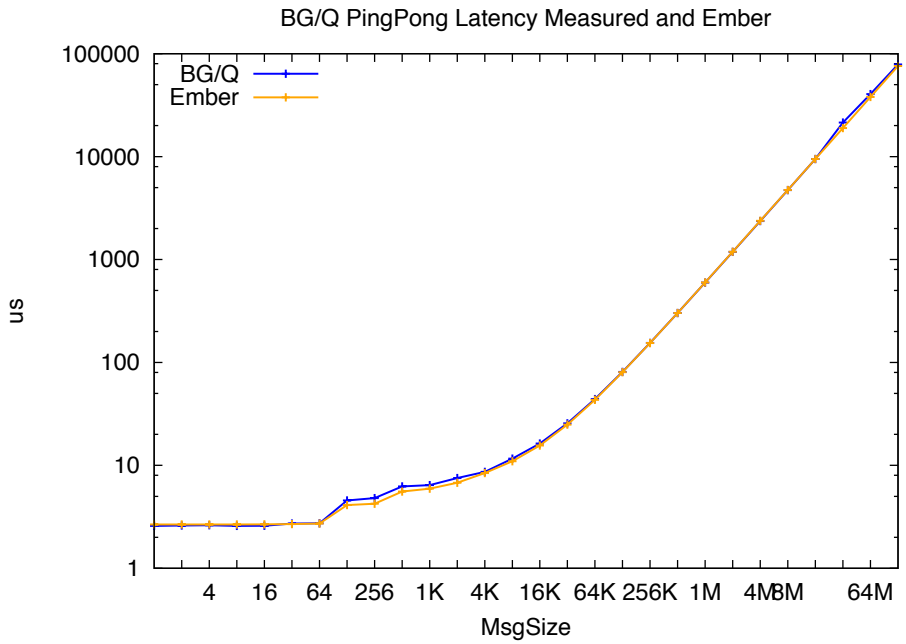
Using Ember Communication Patterns



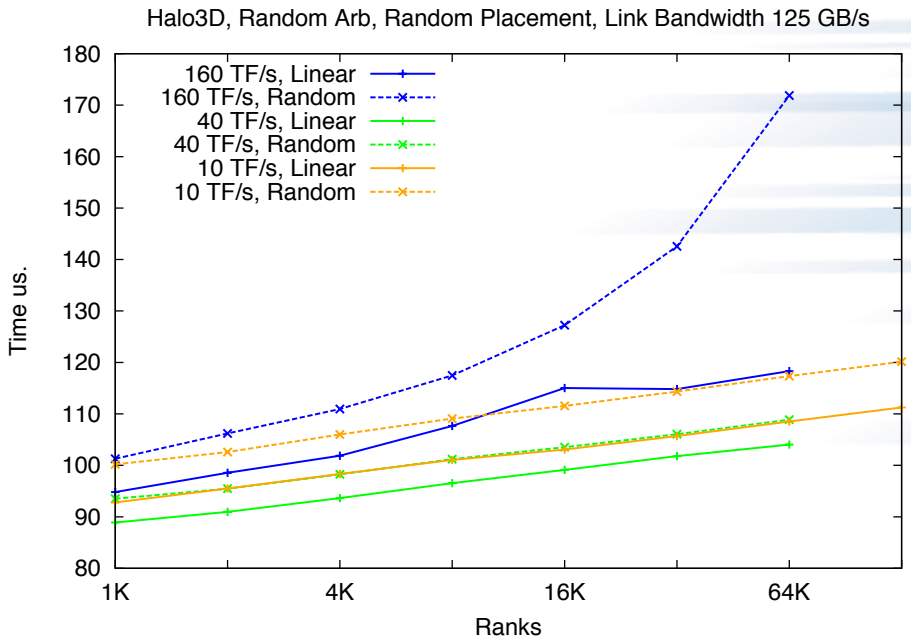
- **How can I use Ember Communication Patterns?**
  - Two main choices: (1) Run the MPI/SHMEM implementations, (2) Run the patterns on one of DOE's simulation environments (e.g. SST, CODES..)

- What is the SST Simulator?
  - Parallel, Conservative, Discrete-Event Hardware Simulation Environment
  - Can simulate cycle-accurate models of single compute nodes through to full-scale models of system interconnects
  - Validated against DOE machine installations
  - Used by industry vendors (Cray, Intel, IBM, HPE, etc)
  - Email: wg-sst@sandia.gov for more information
  - Go to: https://sst-simulator.org

Example Uses of Ember



Simulator/Performance Model Validation



Projection of Communication Patterns For Future Exascale Machine Configurations

EXASCALE COMPUTING PROJECT

Communication Patterns included in Ember

- Nearest neighbor (Structured, halo communication)
- Nearest neighbor (Unstructured, halo-like communication, varying message sizes)
- Wavefront sweeps
- MPI Collective operations (reduction, all-gather, scatter, etc)
- Random, fine-grained messaging (closer to large-scale graph analysis)
- FFTs (all-to-all messages)
- Network in-casts (many to few communication flows, also represent I/O transfers)

Assessment Activities

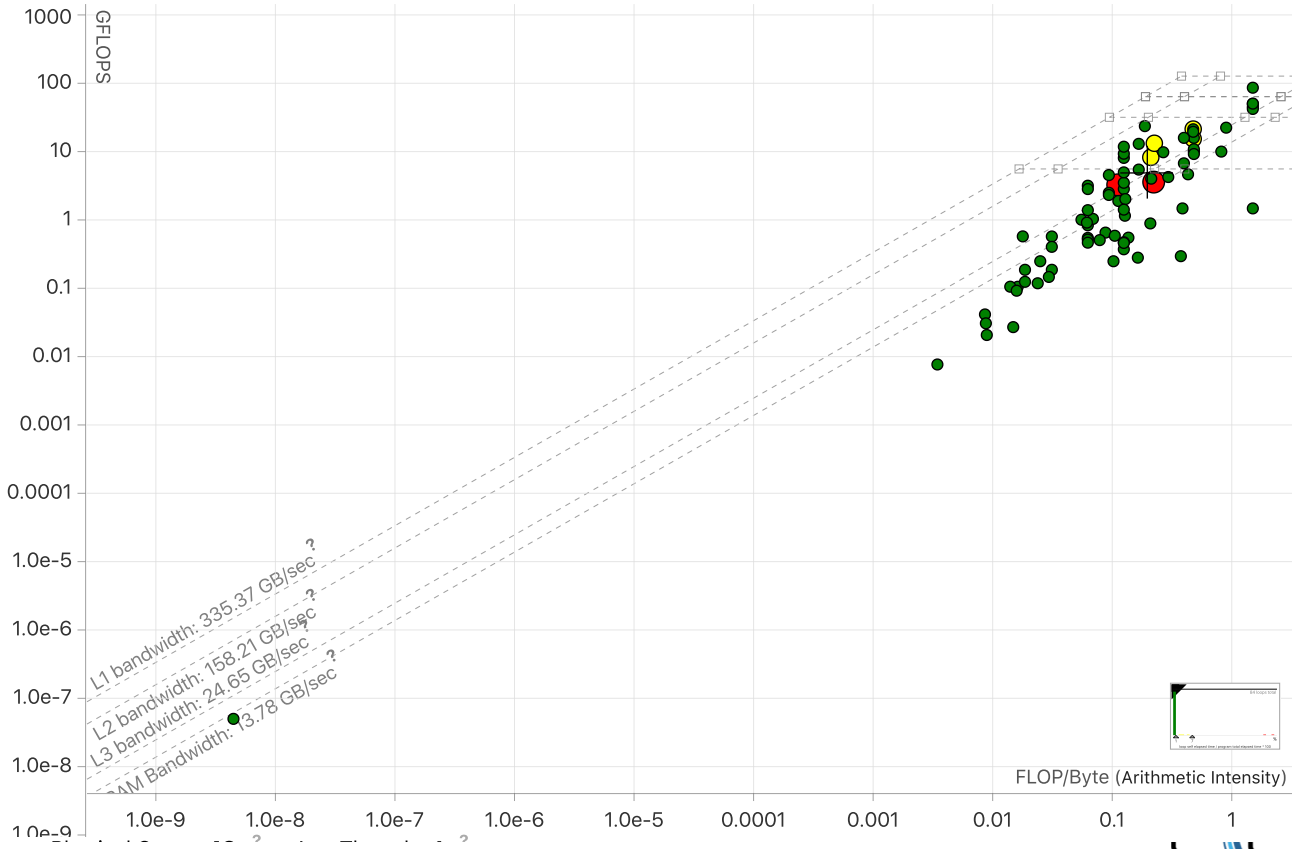
- Goal
  - Performance characterization and identification of underlying hardware bottlenecks that negatively impact performance of both proxy and parent application
    - On current generation platforms
    - Using Exascale challenge problems where possible, otherwise a challenge problem on current systems
  - Determine quantitatively if the proxy represents the parent as intended (e.g., memory, communication, computation)
  - Target 8 new proxy/parent pairs per year
  - Looking at both CPU and GPU-based performance

Standard Performance Characterization

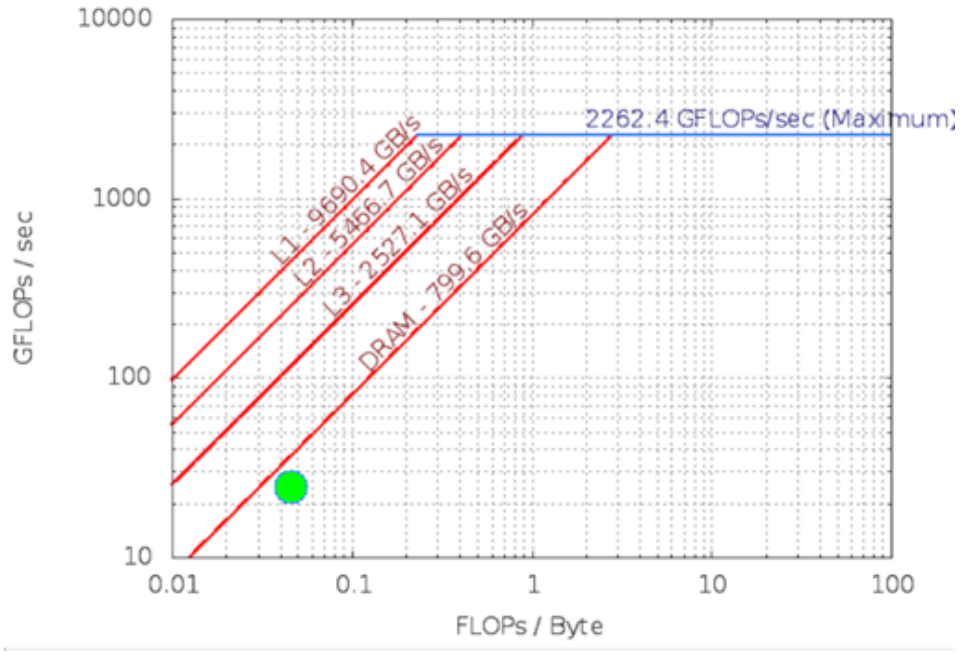
- Dynamic profiling to determine hot functions
- Roofline model to understand upper bound
  - Cache and memory bandwidths
  - FLOPS/arithmetic intensity
- For both proxy and parent and for whole execution and for each of top 10 functions (10 functions that account for largest percentage of total execution time)
  - Cache and memory bandwidths
  - FLOPS/arithmetic intensity

Baseline using Roofline Model

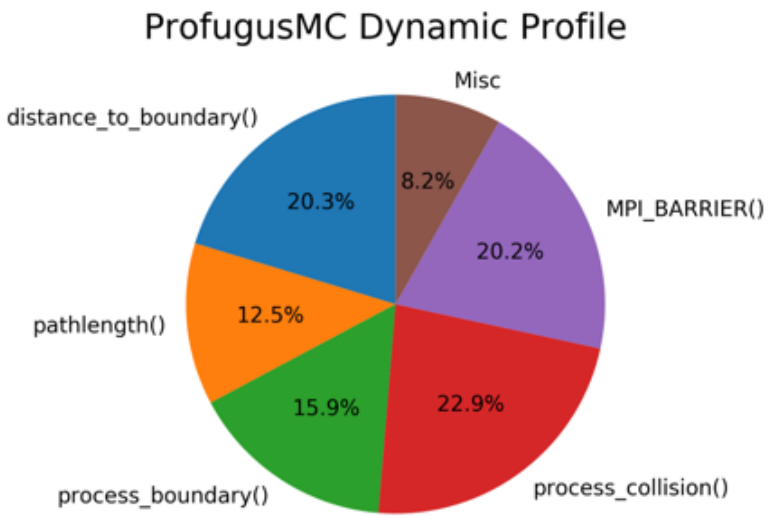
- miniQMC



Profugus roof line from Brian



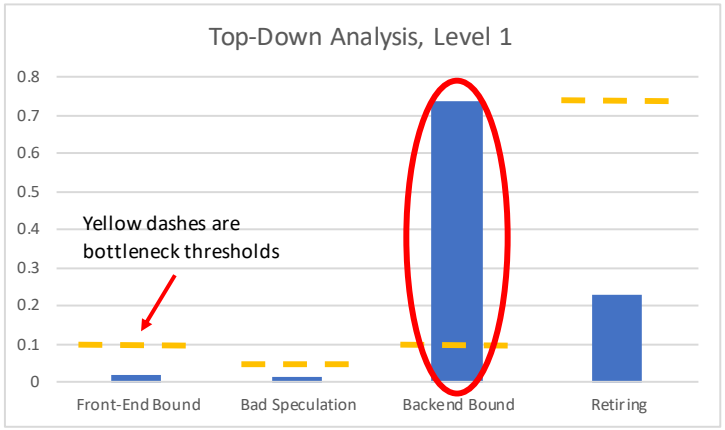
Profugus Dynamic Profile from Brian

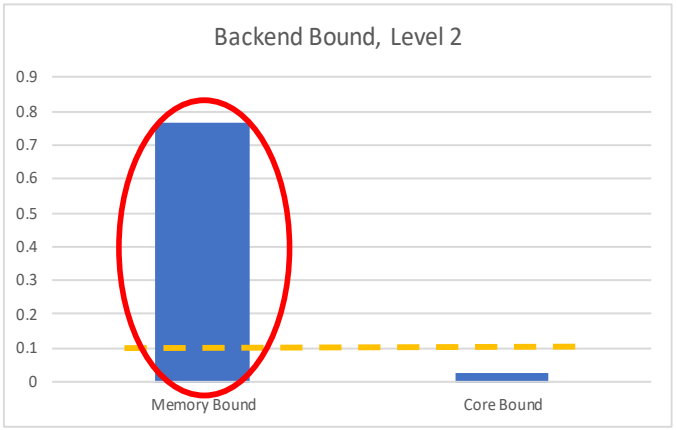


Identifying Hardware Bottlenecks

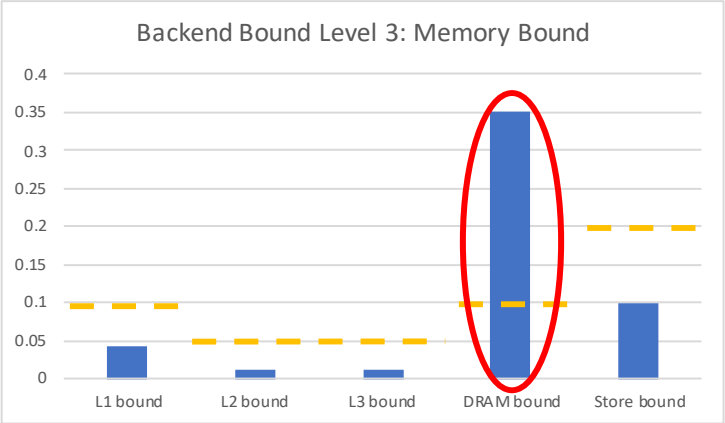
- Drill-down on memory bounded-ness
  - Implemented Intel's Top-Down Microarchitecture Analysis (TMA) into LDMS
    - Fast, flexible, and get per-process data
  - Currently validating TMA methodology
    - Does it really identify bottlenecks correctly?

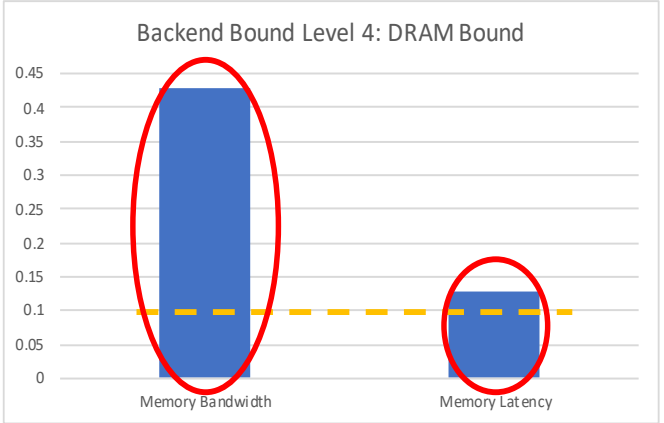
QMCPack, NiO256 atoms on Skylake

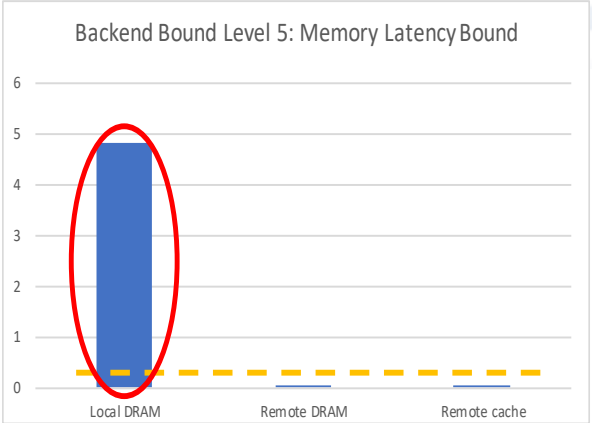




TMA identifies the same bottlenecks for miniQMC

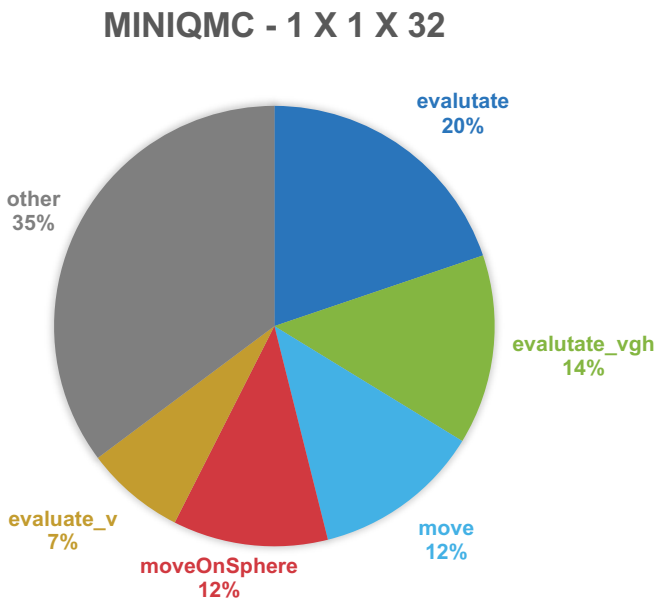


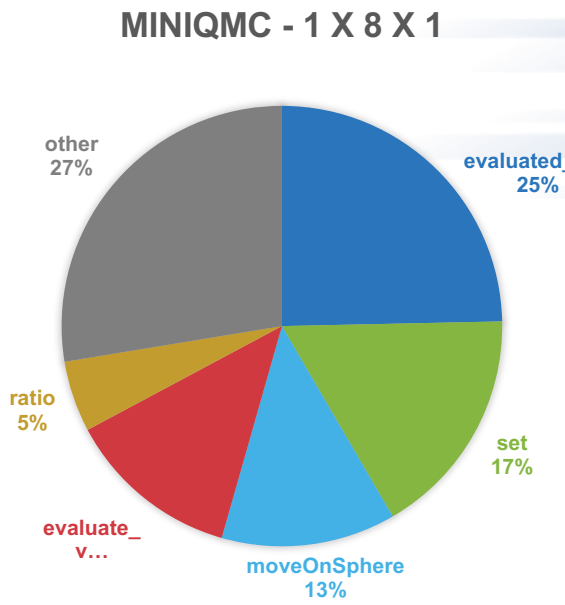




Function Drill-Down: Dynamic Profiles

- MiniQMC





Instrumented these functions, currently collecting TMA and other hardware performance counters to better understand composite behavior.

GPU Characterization

- Motivation
  - Large proportion of performance on exascale computers expected to come from accelerators such as GPUs.
  - Tradeoffs between performance and portability depending on GPU program model used.
  - Application may have different characteristics and bottlenecks on GPU vs. CPU architectures.
- Goal is to use proxy apps to answer the following questions:
  - What are the implications of GPU architecture trends for ECP applications?
  - What GPU programming models will enable the best performance portability across future accelerator architectures?
  - What software optimizations are key to achieving good performance on GPU architectures?

Interaction with ECP AD Teams

Best practices for proxy app development for GPUs (Proxy App team can help as resources allow)

- Provide proxy app configurations and inputs representative of exascale application problems (can be scaled back to reduce runtime and number of nodes)
- Keep proxy app up-to-date with main app development
  - In some cases (e.g., SW4lite, ProfugusMC), proxy app leads GPU development
- Provide feedback on Proxy App team assessment results (maybe we missed something)
- Optimize proxy app GPU implementation for new GPU architectures and features
  - Most time-consuming kernels are the most important.

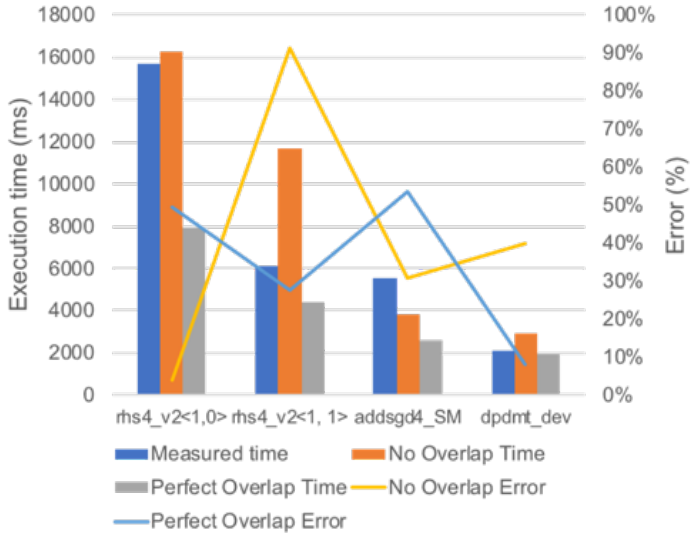
Why GPU Assessment Matters

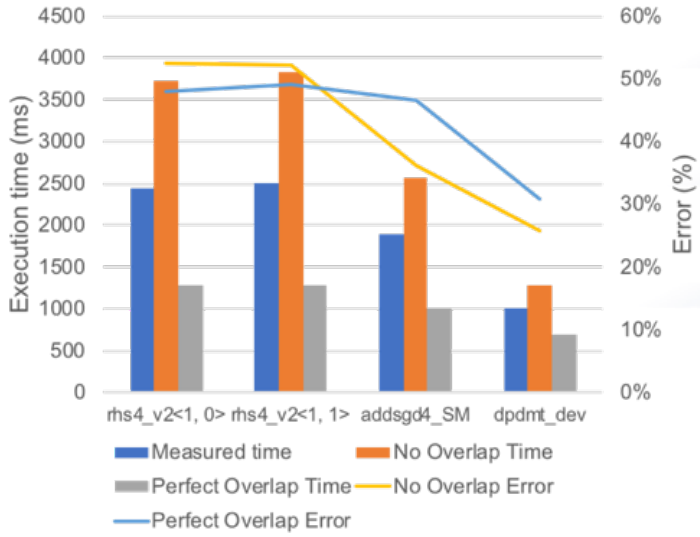
- Ask yourselves
  - If a vendor uses my proxy app to help with design decisions for GPU architecture features, will the result benefit performance of my application?
  - If my proxy app is used for procurement decisions for a machine with accelerators, will the resulting machine architecture and configuration match my application requirements?
- If you don't have a representative GPU implementation that is agile enough to be easily built and run (including on simulators and prototypes), your application will not influence these decisions!

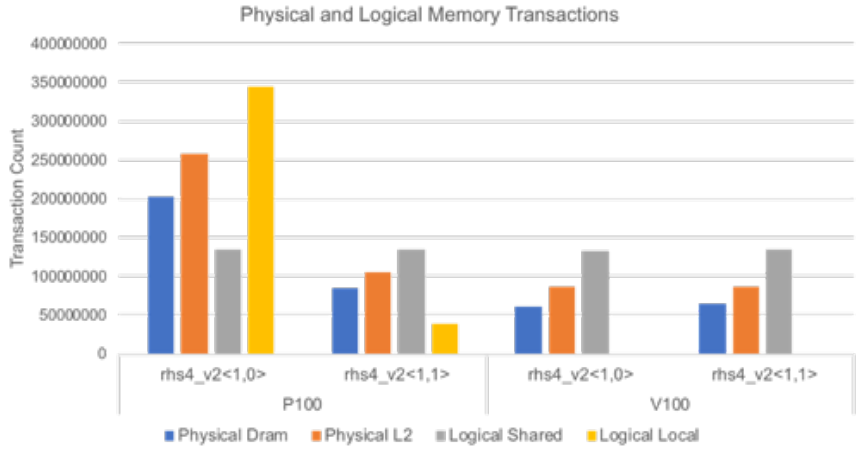
GPU Assessment Procedure

- Similar to CPU assessment that has already started
- Steps
  - Obtain code and inputs for CPU and GPU versions
  - Build and run CPU and GPU versions and verify results
  - Profile execution
  - Collect characterization data
    - Instruction mix
    - Achieved cache and memory bandwidths
    - Communication characteristics
  - Analysis
    - Compare full and proxy app characteristics
    - Roofline models
    - Identify performance and scaling bottlenecks

SW4lite Performance on P100 vs. V100







LOH.1-h100.in input running on one GPU

EXASCALE COMPUTING PROJECT

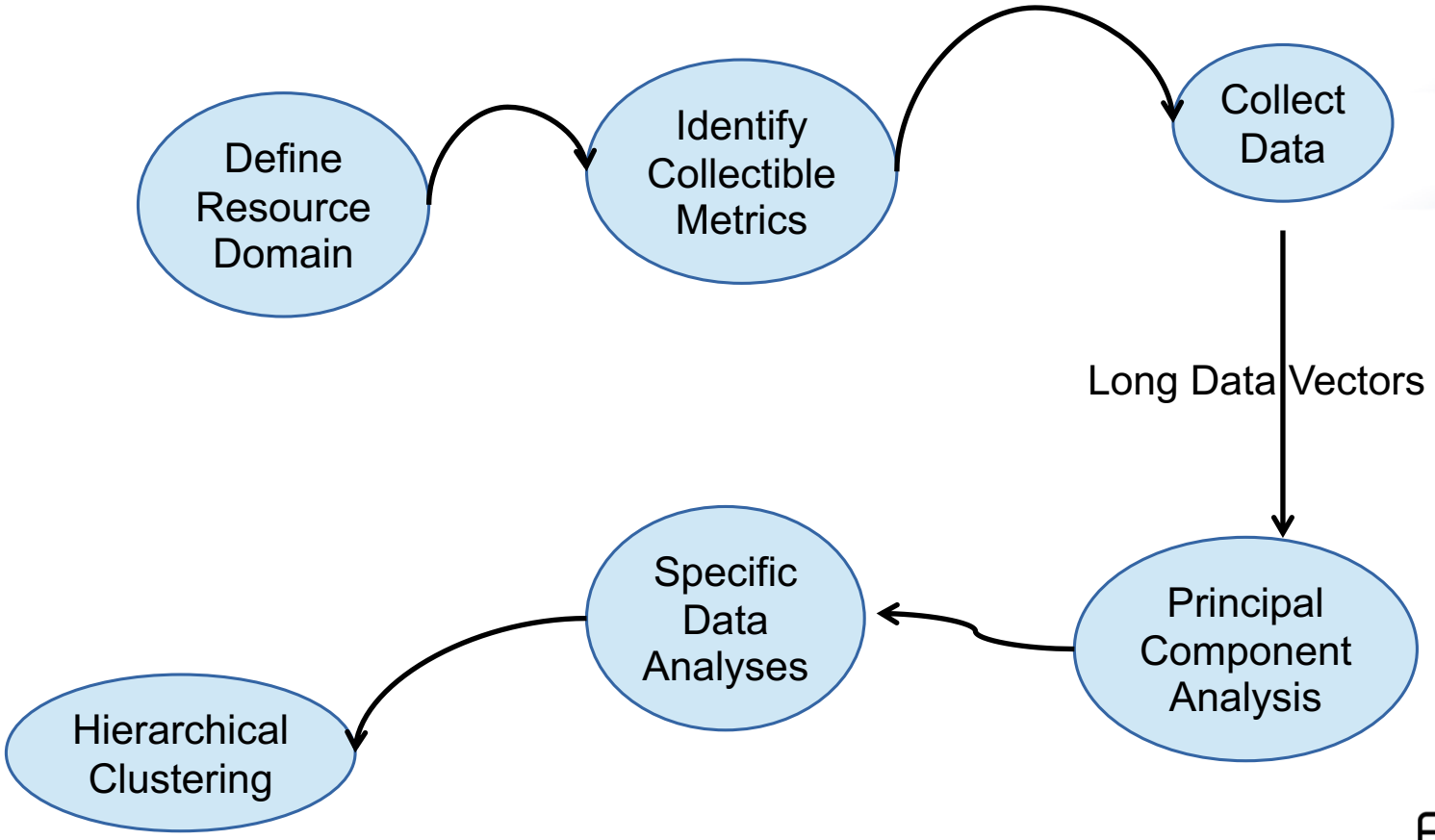
SW4lite Results on Summit

Test case	Ngp	Nts	Nodes	GPUs	Runtime (sec)
LOH.1-h100	1.55e7	1073	1	1	19.4
LOH.1-h100	1.55e7	1073	1	6	5.16
LOH.1-h50	1.23e8	1073	1	3	55.4
LOH.1-h50	1.23e8	1073	1	6	31.8
LOH.1-h50	1.23e8	1073	2	8	25.4
LOH.1-h25	9.82e8	1073	2	12	115
LOH.1-h25	9.82e8	1073	4	24	69.7

Remarks:

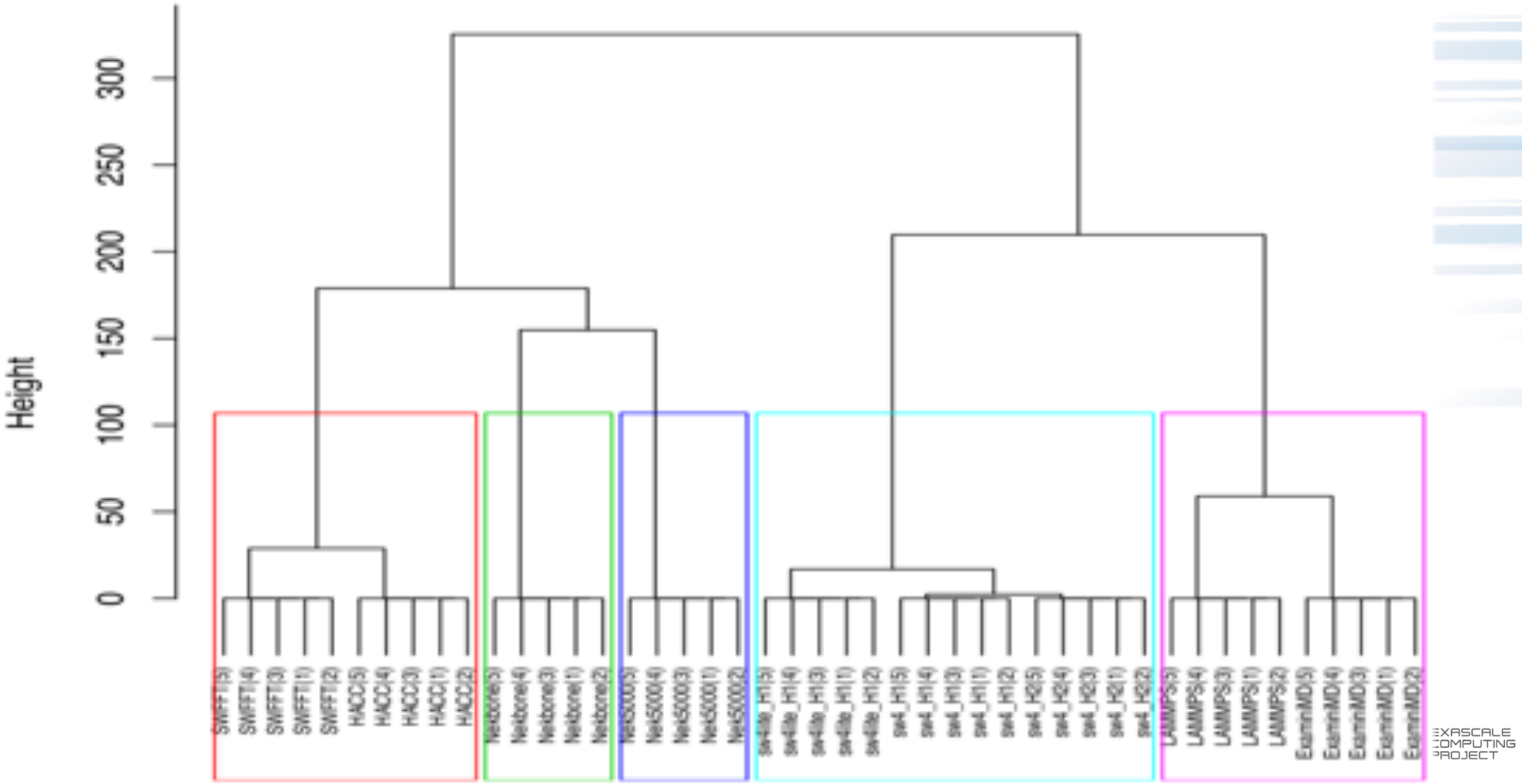
- Reasonable strong scaling
- Weak scaling is not good (much worse that SW4 on Cori 2 KNL, where runtime increased 6% going from h100 to h50 and increased 30% going from h50 to h25)

Clustering Analysis of Proxy Representativenes

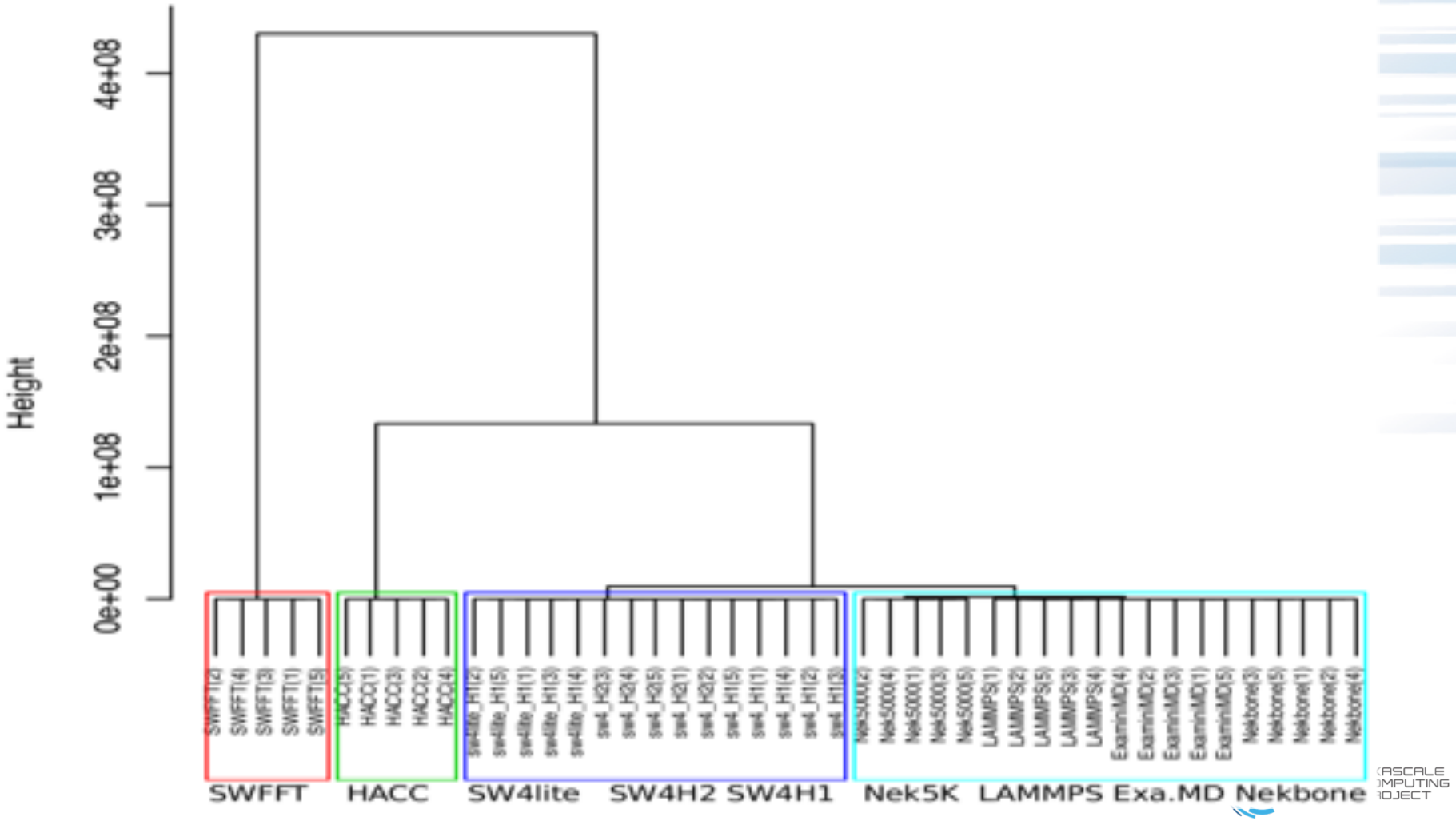


IEEE Cluster 2018, ""A Methodology for Characterizing the Correspondence Between Real and Proxy Applicati

Broadwell Basic Node Hardware Metric Clustering



Broadwell Communication Clustering

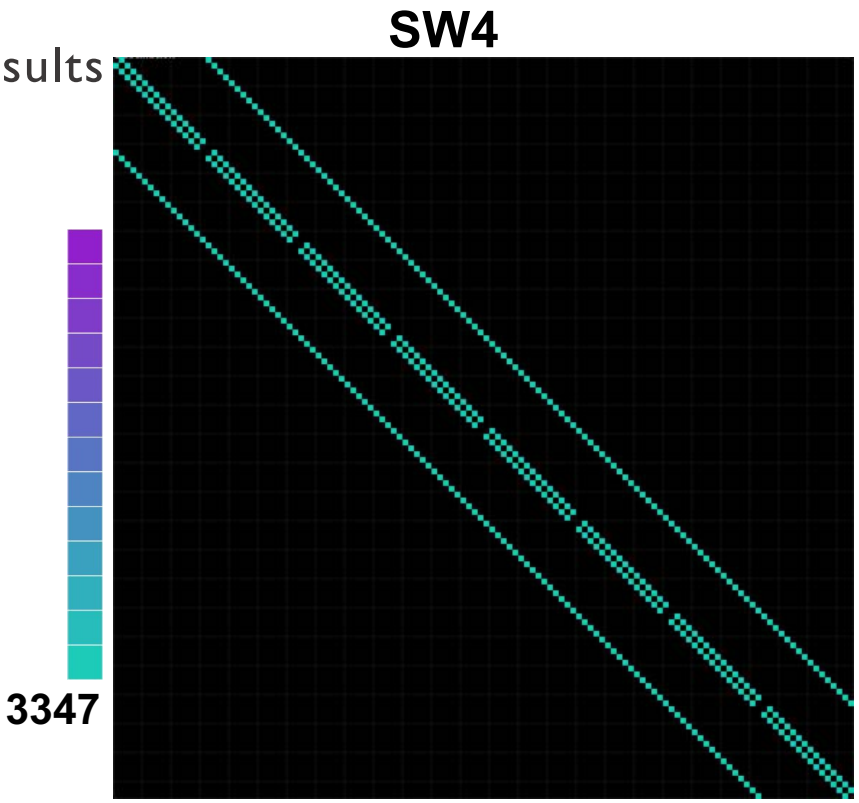


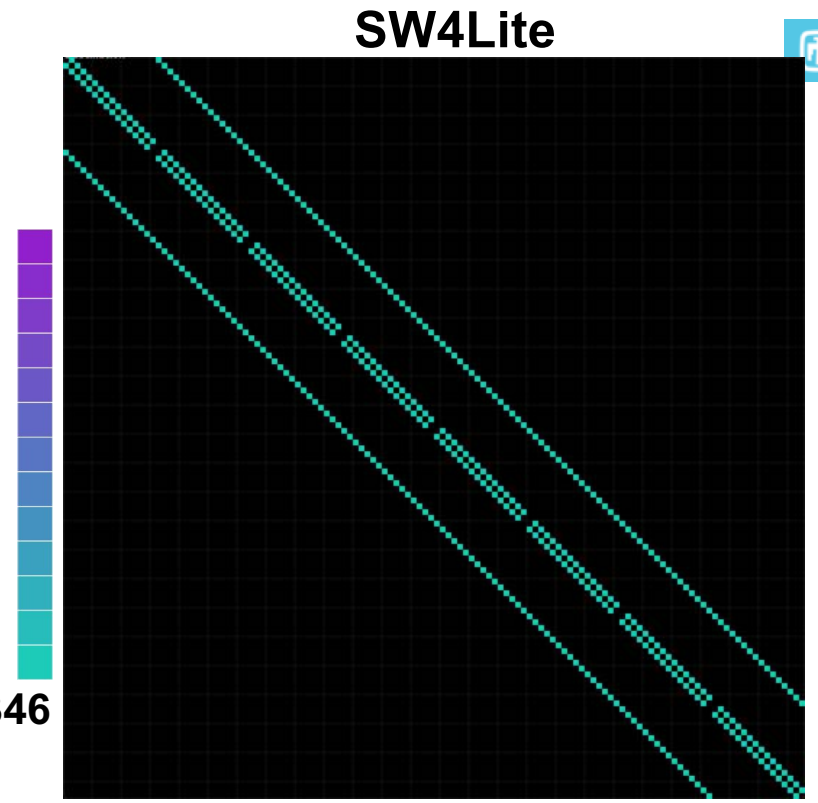
Proxy/Parent Communication Similarity

- Communication metrics based on behavior (not MPI primitives used)
  - Pairwise communication data analysis
    - Point to point communication patterns (source, destination)
    - Total number of messages sent for each pair
    - Using CrayPat tool
  - Communication vector data clustering
    - KB/sec - Total size of data transferred (KB) / total execution time (sec)
    - MPI KB/sec - Total size of data transferred (KB) / total time spent in MPI (sec)
    - Message size histogram data
    - Using mpiP tool

PMBS2018, “Exploring and Quantifying How Communication Behaviors in Proxies Relate to Real Applications”

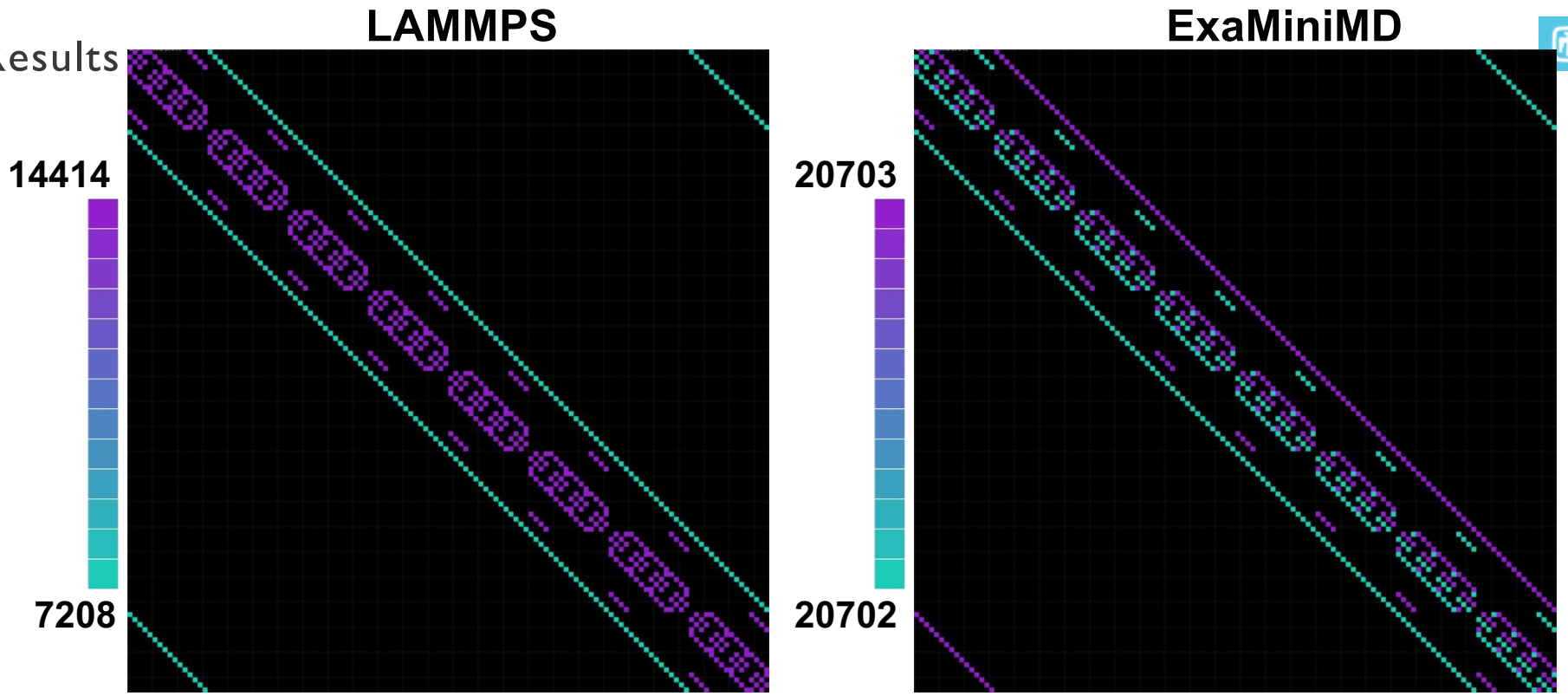
Results





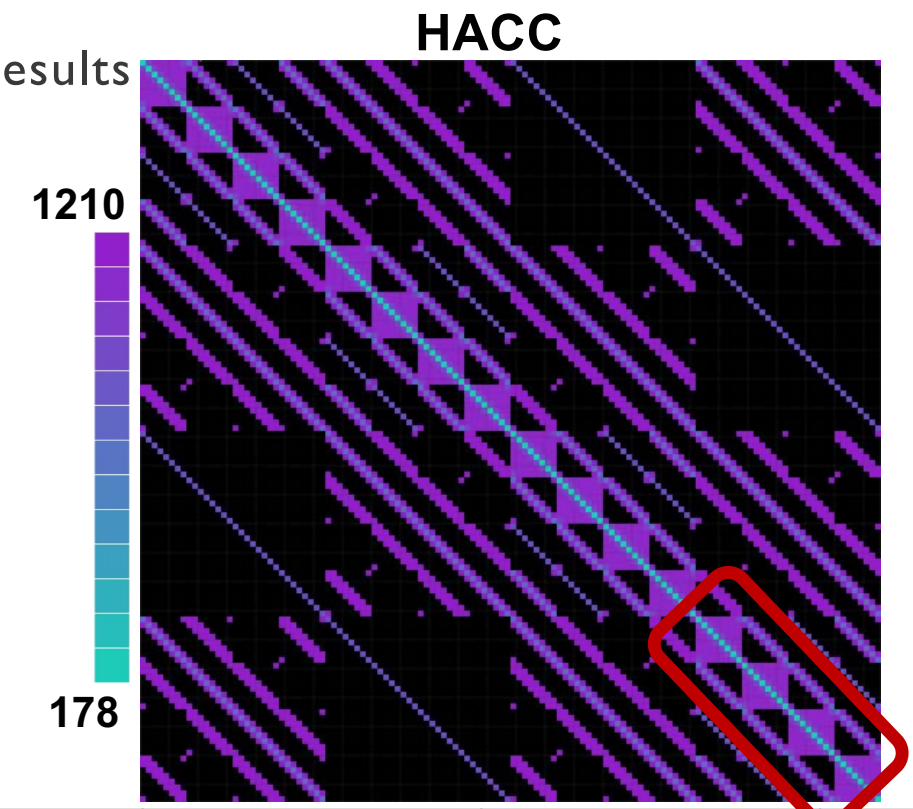
Parent/Proxy	Parent in Proxy		Proxy in Parent		Full Set		Parent in Proxy		Proxy in Parent	
	#msg	#pair	#msg	#pair	PCorr	SCorr	PCorr	SCorr	PCorr	SCorr
SW4/ SW4lite	100	100	100	100	1	1	1	1	1	1

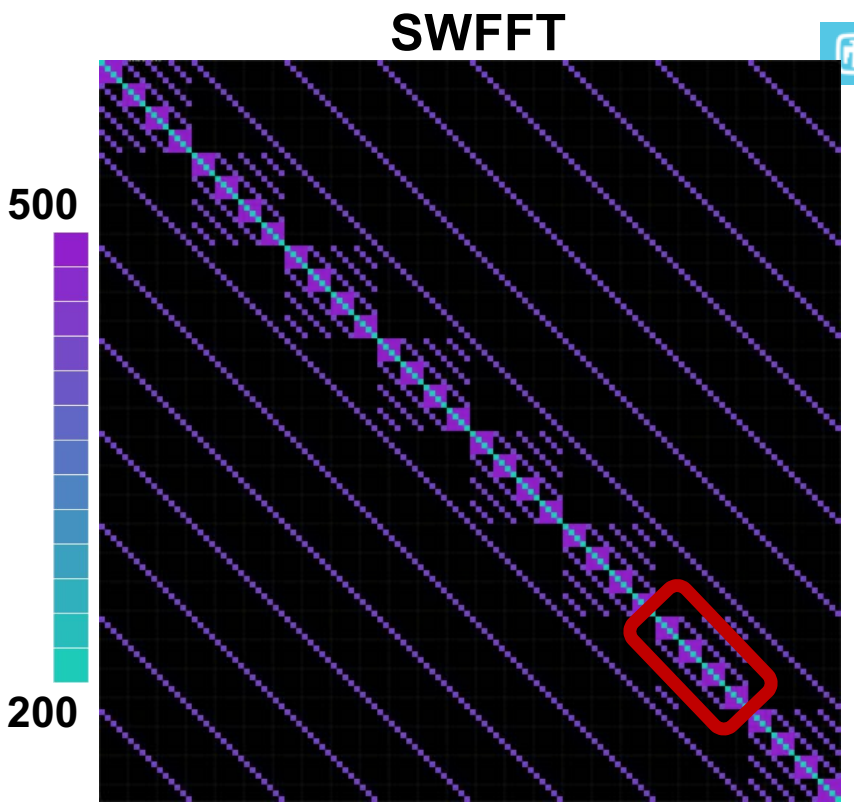
Results



Parent/Proxy	Parent in Proxy		Proxy in Parent		Full Set		Parent in Proxy		Proxy in Parent	
	#msg	#pair	#msg	#pair	PCorr	SCorr	PCorr	SCorr	PCorr	SCorr
LAMMPS/ ExaMMD	100	100	100	100	0	0	0	0	0	0

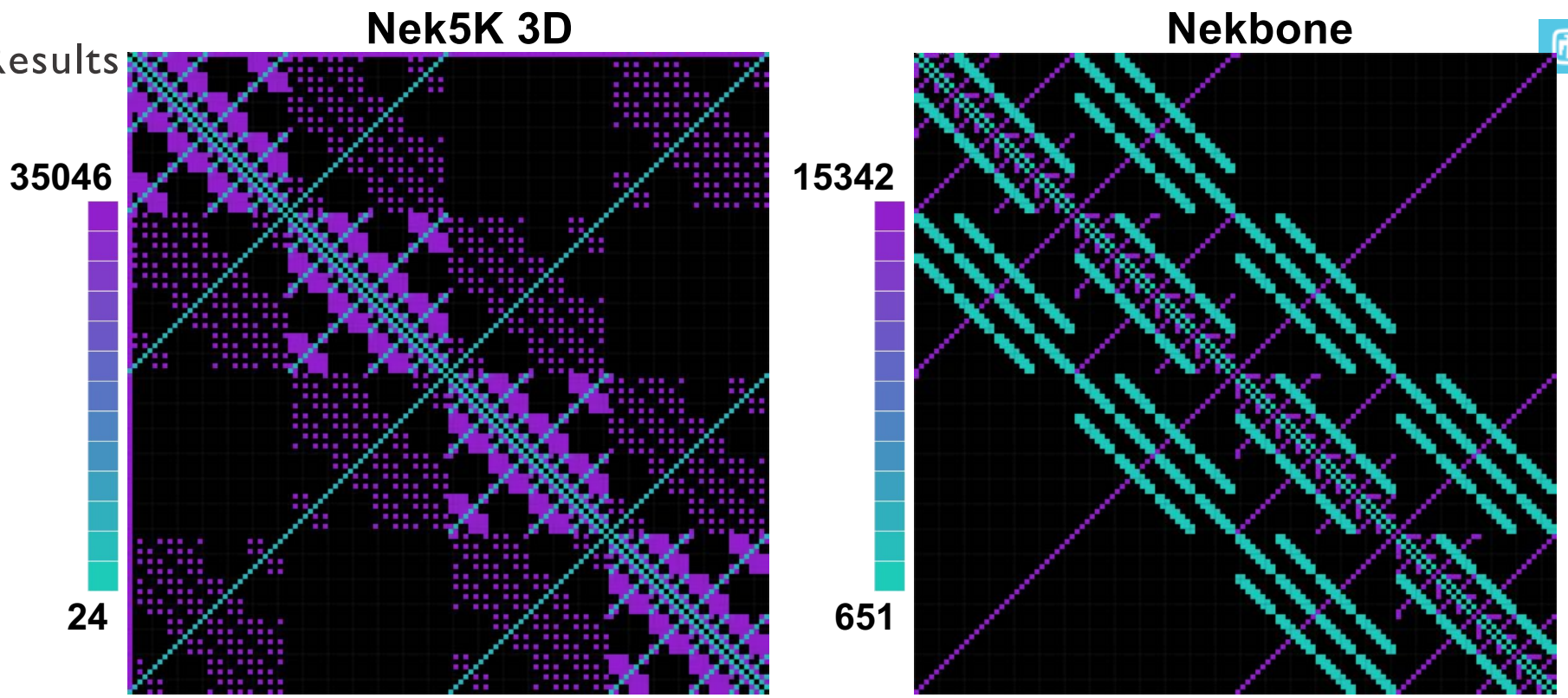
Results





Parent/Proxy	Parent in Proxy		Proxy in Parent		Full Set		Parent in Proxy		Proxy in Parent	
	#msg	#pair	#msg	#pair	PCorr	SCorr	PCorr	SCorr	PCorr	SCorr
HACC/ SWFFT	51.7	29.4	71.4	71.4	0.58	0.31	0.61	0.28	0.87	0.81

Results



Parent/Proxy	Parent in Proxy		Proxy in Parent		Full Set		Parent in Proxy		Proxy in Parent	
	#msg	#pair	#msg	#pair	PCorr	SCorr	PCorr	SCorr	PCorr	SCorr
Nek5K 3D/ Nekbone 3D	99.9	51.4	58.0	68.4	-0.1	-0.05	-0.65	-0.23	0.04	0.49

Acknowledgment

This research was supported by the **Exascale Computing Project (ECP)**, Project Number 17-SC-20-SC, a collaborative effort of two DOE organizations, the Office of Science and the National Nuclear Security Administration, responsible for the planning and preparation of a capable exascale ecosystem including software, applications, hardware, advanced system engineering, and early testbed platforms, to support the nation's exascale computing imperative.

Questions???


EXASCALE COMPUTING PROJECT

Thank you!


U.S. DEPARTMENT OF
ENERGY
Office of
Science


National Nuclear Security Administration