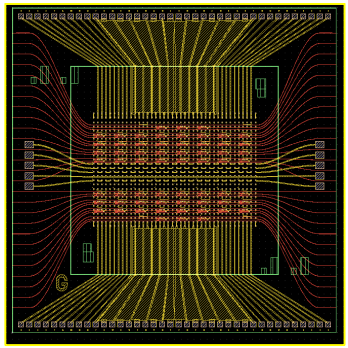
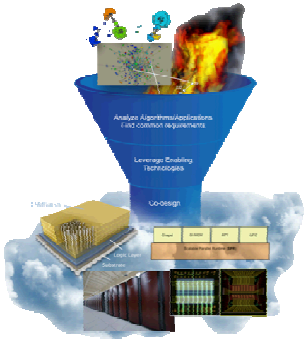
Exceptional service in the national interest

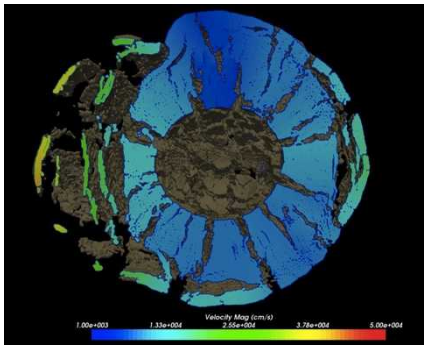
SAND2013-7139P

Sandia National Laboratories

EXTREME-SCALE COMPUTING GRAND CHALLENGE







Applications Thrust update

External advisory board review

August 28-29, 2013

U.S. DEPARTMENT OF ENERGY

NNSA

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2011-XXXXP

Team

- Richard Barrett, Jon Berry, Todd Jones, Sam Mulder, Shelley Leger, Cindy Phillips, Brian Rigdon, Marcus Smith

- External: Michael Bender (Stony Brook), Ben Moseley (TTIC), Cliff Stein (Columbia).

Activities

- Identify computational and data movement primitives that form the basis of our computational domain areas.
- Establish the link between these primitives and fundamental algorithms.
- Map primitives and algorithms to capabilities of proposed system architectural mechanisms, system software capabilities, etc. and vice versa.
- *Iterate*

Overview

- Scratch pad analysis
- GraphApp
  - Enumerate all triangles in a giant, distributed graph
- PathFinder
  - Search for a sequence of labels
- miniGhost/AMR
  - Refinement driven by object moving through domain.

Snapshot of Technologies

	MPI	Intrins.	CUDA	OpenAcc	OpenCL	Kokkos Array	OpenMP		qthreads		Cilk	ArBB/CEAN	pthreads	MKL/Math Lib.	Adv. Lang.	DSLs
							DP	TP	DP	TP						
NVIDIA (GPU)																?
AMD (CPU)																?
AMD (APU)		?														?
Intel (CPU)																?
Intel (MIC)																?
IBM (BG)																?
ARM																?

miniFE																
miniMD															Chapel	
miniGhost																
miniAMR																
PathFinder																

Many other options:

- Microbenchmarks and Kernels
- LULESH : Livermore (3d) Unstructured Lagrangian Eulerian Shock Hydrodynamics
- Cloverleaf : 2d Eulerian (AWE); CleverLeaf : SAMRAI
- CoMD : Molecular dynamics
- VPFFT++

NERSC-8 / TRINITY BENCHMARKS

These benchmark programs are for use as part of the joint NERSC / ACES NERSC-8/Trinity system procurement. There are two basic kinds of benchmarks:

MiniApplications: miniFE, miniGhost, AMG, UMT, GTC, MILC, SNAP, and miniDFT
MicroBenchmarks: Pynamic, STREAM, OMB, SMB, ZiaTest, IOR, Metabench, PSNAP, FSTest, mpimemu, and UPC_FT

The SSP is an aggregate measure based on selected runs of the MiniApplications.

- The benchmark run rules are available here (PDF, last updated August 6, 2013).
- The TrinityNERSC8BenchmarkResults Excel spreadsheet should be used to report results for all runs included as part of the Offeror's response.

SSP
Brief description, sample calculation, and Hopper baseline SSP value.

Change Log

miniFE
MiniFE mimics the finite element generation, assembly and solution for an unstructured grid problem.

MiniGhost
a Finite Difference mini-application that implements a difference stencil across a homogenous three dimensional domain.

AMG
AMG is a parallel algebraic multigrid solver for linear systems arising from problems on unstructured grids.

UMT
UMT is a 3-D, deterministic, multigroup, photon transport code for unstructured meshes.

GTC
3D Gyrokinetic Toroidal Code

SNAP
SNAP is a proxy for the performance of a modern discrete ordinates neutral particle transport application.

MILC
MILC represents part of a set of codes used to study Lattice Quantum ChromoDynamics (QCD).

MiniDFT
MiniDFT is intended to capture the performance-critical portions of a density functional theory materials science computation.

NPB UPC-FT
This is the NAS Parallel Benchmark FFT program written in the UPC language.

Pynamic
Pynamic tests dynamic loading subsystem design and the ability to handle heavy use of dynamically linked libraries from a large Python-based scientific application.

STREAM
The STREAM benchmark is designed to measure the sustainable memory bandwidth and corresponding computation rate for four simple vector kernels

OMB MPI Tests
The OSU MicroBenchmarks carry out a variety of message passing performance tests using MPI.

SMB
The "Host Processor Overhead" and "Real World Message Rate" benchmarks from the Sandia MPI Micro-Benchmark Suite (SMB).

PSNAP
P-SNAP is a micro benchmark to measure OS impact on parallel jobs.

ZiaTest
This test executes a new proposed standard benchmark method for MPI startup that is intended to provide a realistic assessment of both launch and wireup requirements.

mdtest
Used to measure file system metadata operation rates

IOR
IOR is used for testing performance of parallel file systems using various interfaces and access patterns.

mpimemu
A simple tool that helps approximate MPI library memory usage as a function of scale.

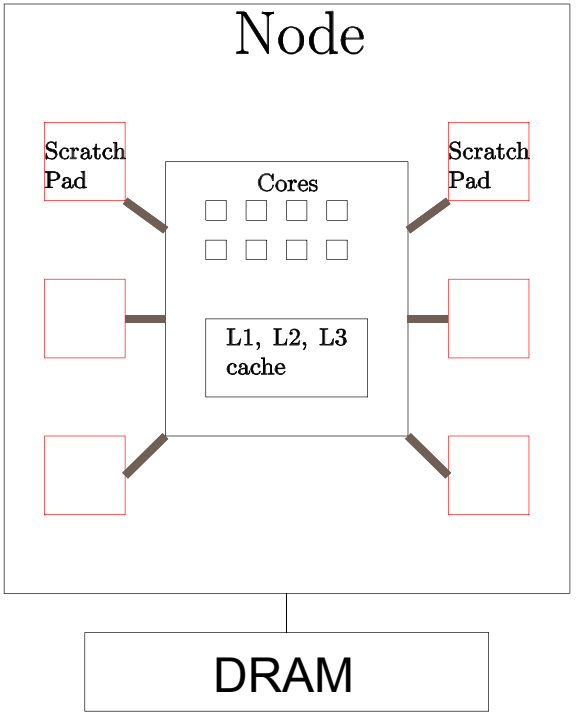
FLOP Counts for "Small" Single-Node Miniapplication Tests
Provided for reference

Theory Subteam (Berry, Phillips) Co-Design Efforts

- Design theoretical models and design and analyze algorithms to support exascale co-design
- Write and profile codes for mini-apps and experiments
- System Software interactions (attend their meetings)
  - Co-writing “GraphApp” mini-app
  - Co-designed profiling experiments for high-bandwidth sorting
- Architecture interactions
  - Learn about components
  - Influence prioritization of SST work
- University collaborations
  - Ben Moseley (TTIC), full-time during summer, 2013
  - Cliff Stein, Columbia
  - Michael Bender, SUNY Stony Brook

“Scratchpad” (High Bandwidth Mem)

- Memory bonded to chip
- Compared to DRAM
  - Smaller (<8 Gb vs 10-100Gb/node)
  - Latency same
  - Bandwidth much higher (100x?)
  - User controlled, no coherence
- Could algorithm designers exploit ***user addressability***? (e.g. smalloc())
- If so , can we influence vendors while there is time?



Co-Designed Algorithms for User-Addressable Scratchpad

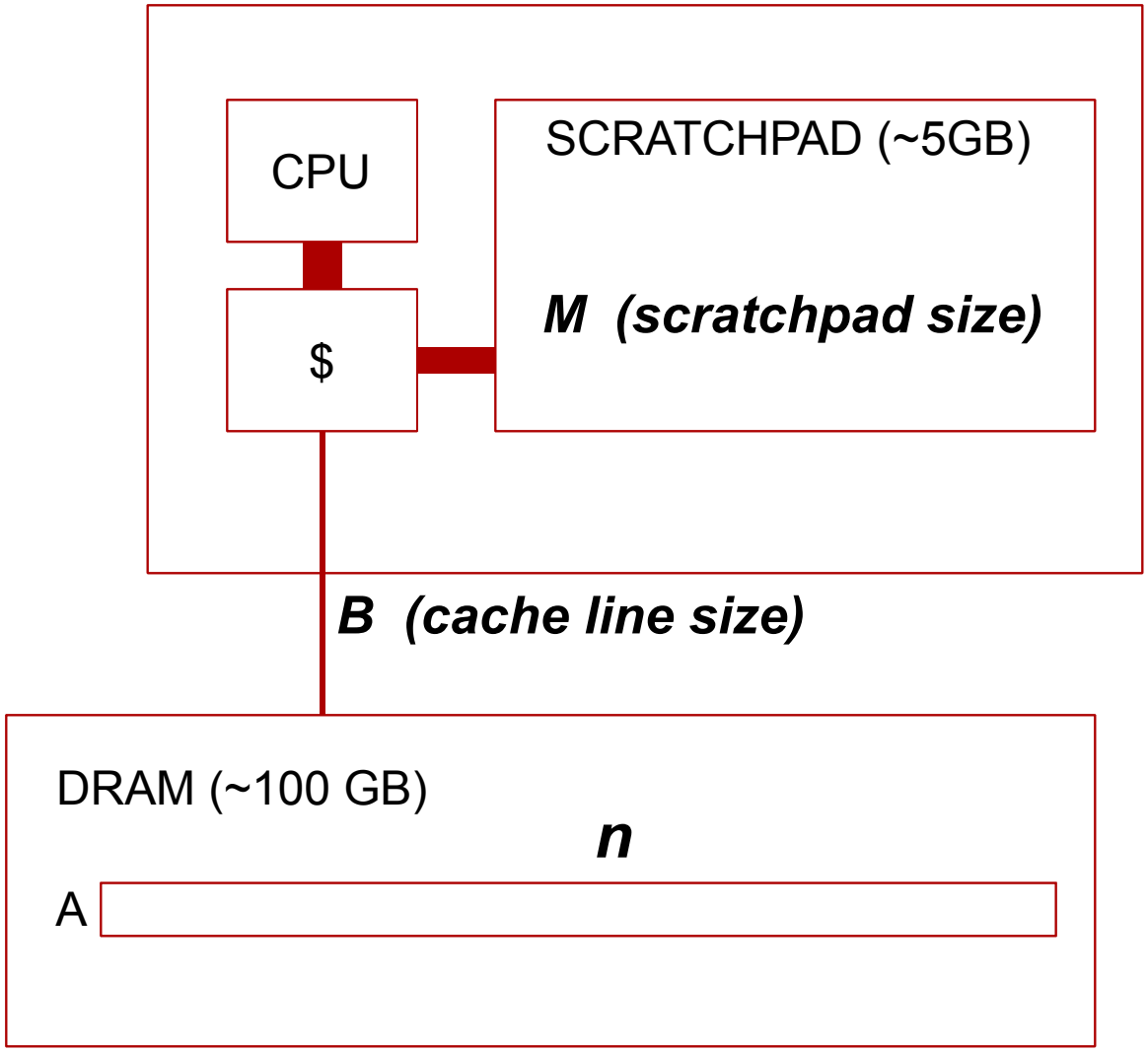
- Sorting
- Sparse matrix-matrix multiplication
  - Blocking, sorting to create sparse-formatted output
  - Mike Heroux verified importance (still need to check structure)
- K-means (clustering):
  - computer vision, geostatistics, astronomy, agriculture, targeted advertising, industrial applications (e.g. recommender systems, textile dyes, hedge funds)

Sorting Algorithm Discussion

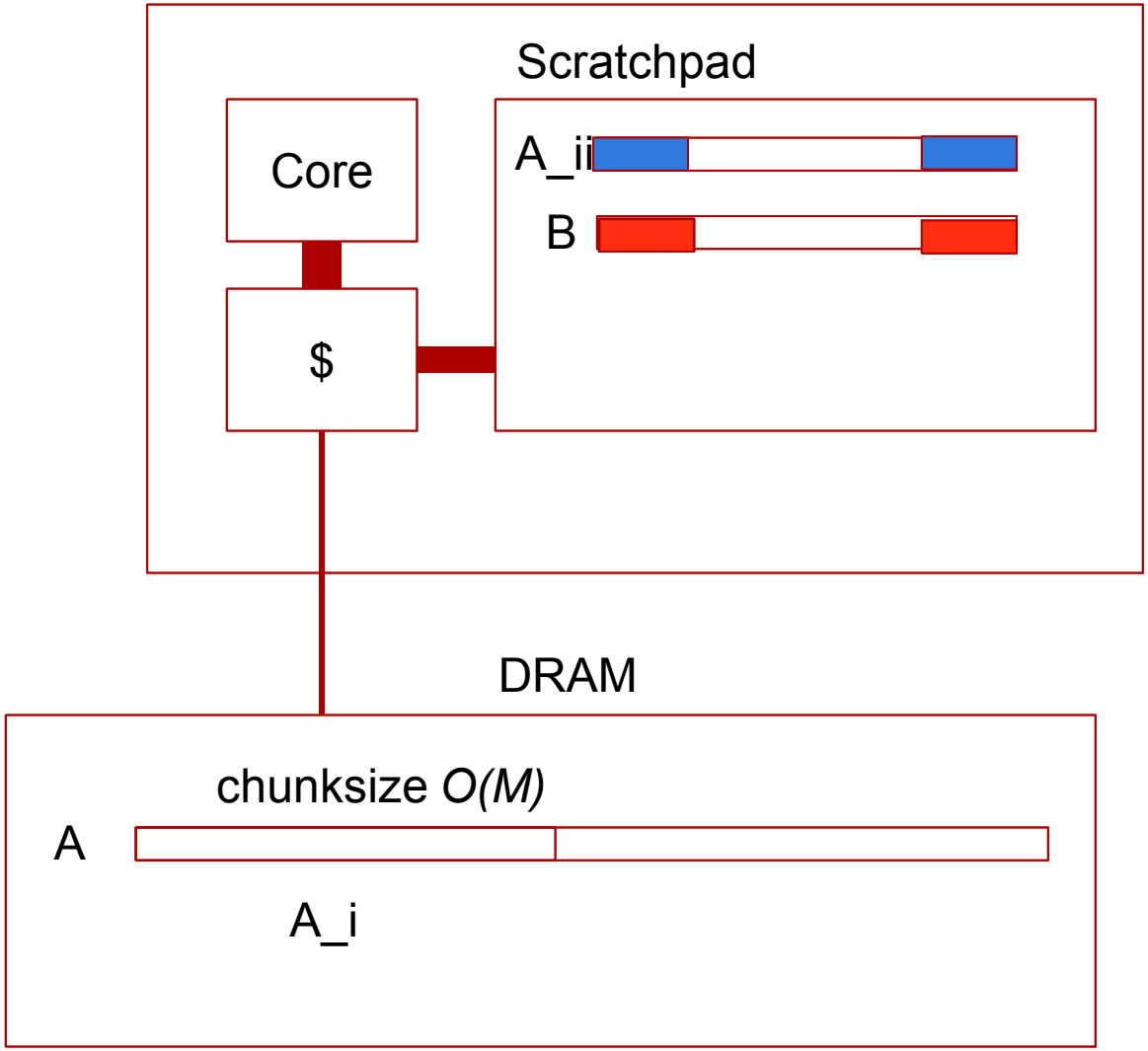
- I'll explain our algorithm (called "scratchpad-sort" for now)
  - Similar ideas have been used in external memory sorting for decades
  - But we have a non-intuitive piece directly motivated by scratchpad
- I'll give the theoretical memory complexity and plug in sample numbers comparing our algorithm to a conventional algorithm
- I'll describe a sorting experiment with real numbers from Intel performance counters

In mid-summer, SST couldn't support scratchpad. We presented our algorithm to architecture staff and they prioritized this feature; it's almost ready.

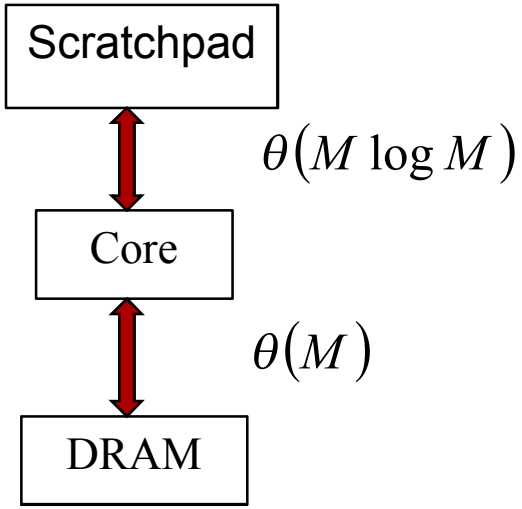
Simplified “Scratchpad” Model



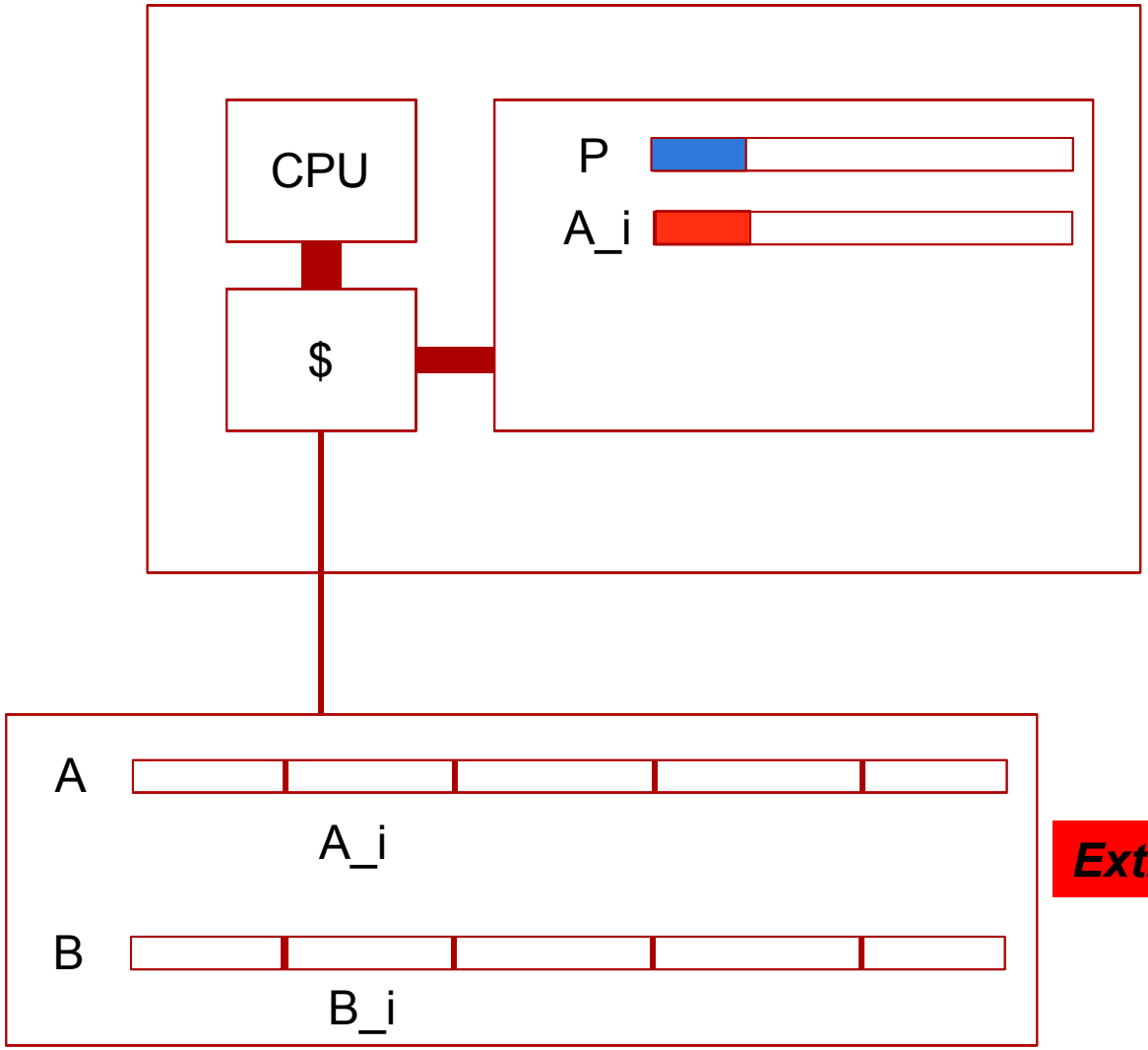
Quicksort of Scratchpad-Resident Data



Sort chunk A_i



Single Node Scratchpad Sort



scratchpad-sort

1.) Select $\Omega(M)$ pivots P

2) Load them into SP, sort

3) Load A_i into SP, sort

4) Stream through P and A_i, writing to buckets B in DRAM

5) Load each bucket to SP, sort (recurse on oversized buckets)

Extra work avoids binary search

Scratchpad-friendly

Memory Complexity

- Notation: T(n): number of DRAM cache line accesses
- Conventional quicksort expected case: $T(n) = \theta\left(\frac{n \log n}{B}\right)$
- Scratchpad Quicksort:

$$\text{DRAM accesses}: \quad T(n) = \frac{M}{B} T\left(\frac{n}{M/B}\right) + \frac{n}{B} = O\left(\frac{n}{B} \log_{M/B} \frac{n}{B}\right)$$

$$\text{Scratchpad accesses}: \; T(n) = \frac{M}{B} T\left(\frac{n}{M/B}\right) + \frac{n \log M}{\rho B}$$

where ρ is the bandwidth expansion factor of scratchpad.

If $\rho \geq \log M$, recurrences are the same, and almost linear

Numerical Thought Experiment

- Sort a trillion 8-byte elements, with O(1GB) scratchpad and 1024 byte cache line

Predict DRAM accesses:

- Conventional: T(n) ~ 44 * 2^33 cache line loads from DRAM
- Our algorithm: T(n) ~ 5/2 * 2^33 cache line loads from DRAM

This would be a savings of ~ 17X DRAM accesses

Memory Bandwidth Experiment

- Profile qthreads "qutil_qsort," a multithreaded quicksort with excellent strong scaling
  - Sort 10 billion 8-byte integers
  - Actual DRAM cache line accesses: ~25 billion
  - Percentage execution stalls: 32%
- For the same input we predict that our algorithm would make:
  - Two O(n/B) pre-fetched sweeps through DRAM, plus
  - O(n/B) writing out buckets,
  - O(n/B) writing out result
  - Predicted DRAM cache line accesses: <=5 billion accesses

We predict a factor of at least 5x fewer DRAM accesses

From Sorting to Sparse MAT-MAT and Beyond

- We have expressed sparse matrix-matrix multiplication in a way that relies heavily on sorting as a kernel
  - So we believe that we can reduce DRAM accesses for this important problem
  - But this subteam doesn't have expertise in sparse MAT-MAT, so we're doing a literature search and consulting with experts.

- Decades-old external memory algorithms of various types may be revived by the scratchpad possibility

Scratchpad Discussion Conclusion

- Two-part message to vendors: If
  1. The bandwidth expansion factor is roughly comparable to the logarithm of the scratchpad size, and
  2. The scratchpad memory is user-addressable

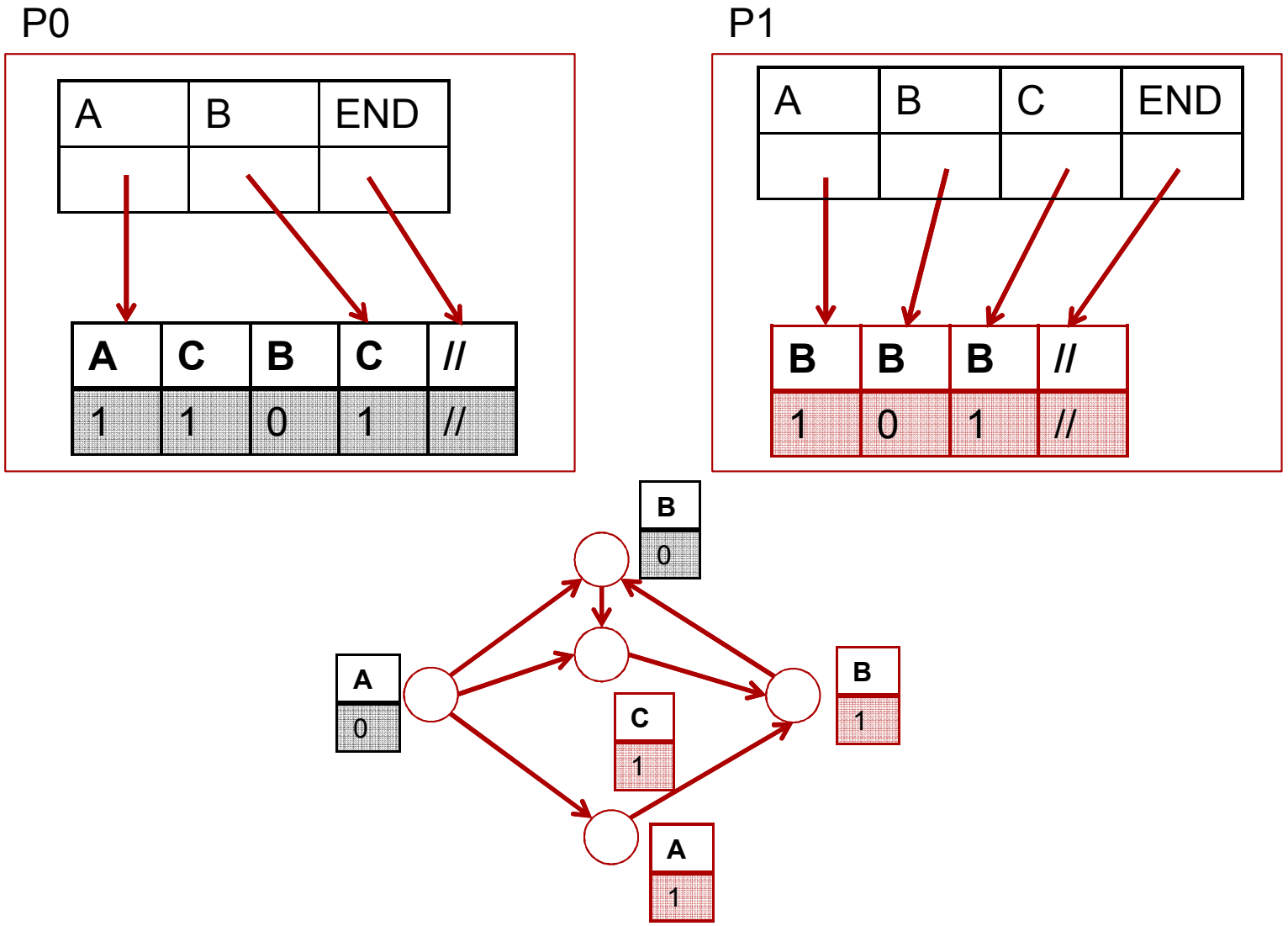
  - Then we have some evidence that fundamental, cross-cutting algorithms might exploit that resource to reduce DRAM access,

  - But we haven't yet characterized any specific speed or power advantages.

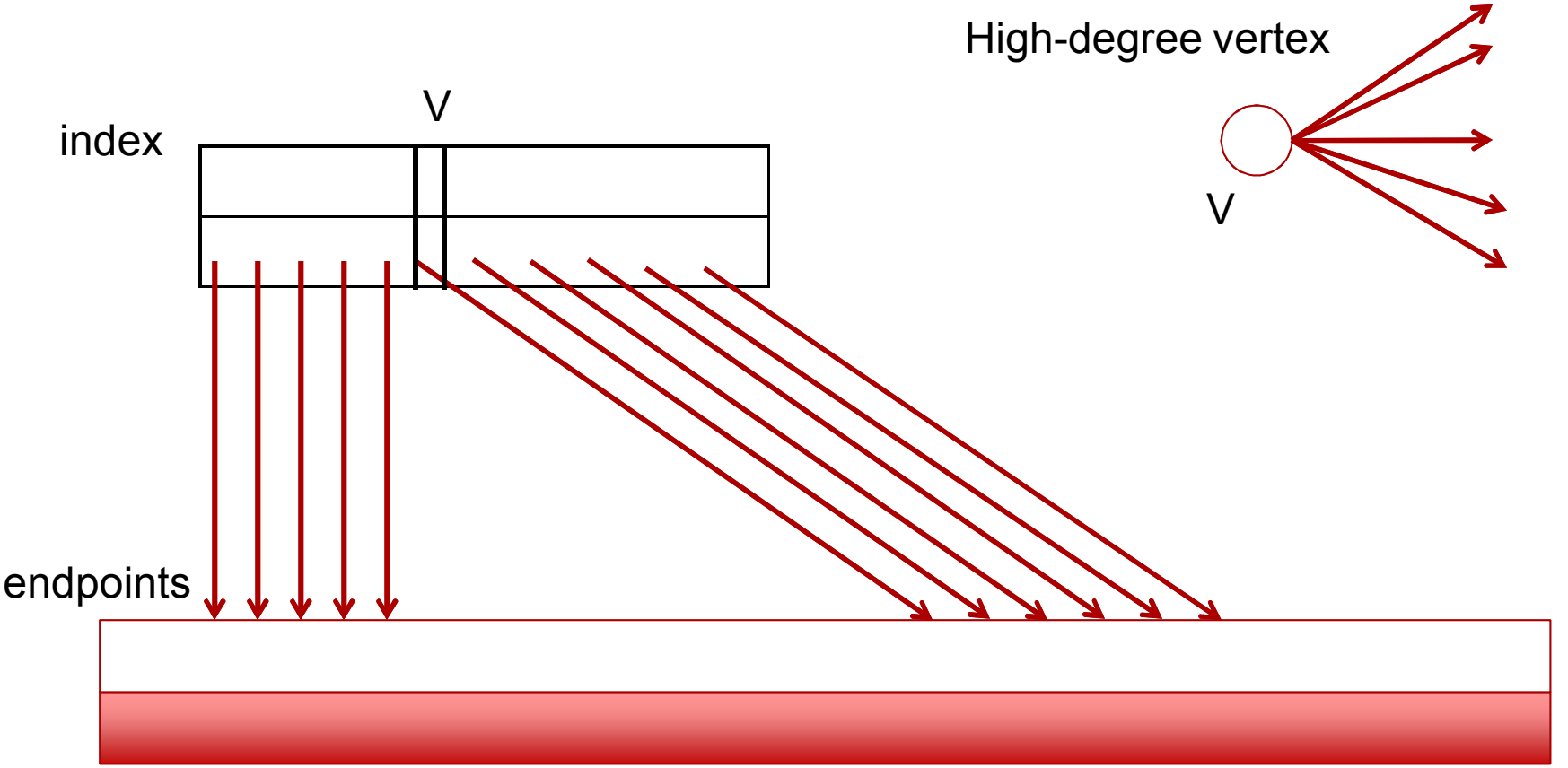
“GraphApp” Status

- Implementation Team: B. Barrett (FY13), J. Berry, D. Stark
- We have a graph traversal mini-app running both multithreaded and distributed in open-shmem
- Initial algorithmic task: compute in-degree of all vertices
  - Versions
    - 1 qthread task per update
    - Concurrency via partitioning vertices
    - Concurrency via “Manhattan Loop Collapse”
  - Status
    - Implemented
    - No experiments yet

GraphApp: Graph Representation



Real Data: “Heavy-Tailed”



Partitioning work by vertex leads to load imbalance

GraphApp: Naïve Adjlist Traversal

Processor p's code:

```
For ((v,p) in Vertices(p)) {
    For ( (w,proc) adjacent to (v,p)) {
        Traverse adjacency   (v,p) → (w,proc)
    }
}
```

How to parallelize? Outer loop? Inner loop?

Manhattan Loop Collapse

- Parallelize over the endpoints array, not the vertex array
- Binary search to find the source vertex
- Used in commercial parallelizing compilers (e.g. Cray-XMT)
- We have an implementation in GRAPHAPP

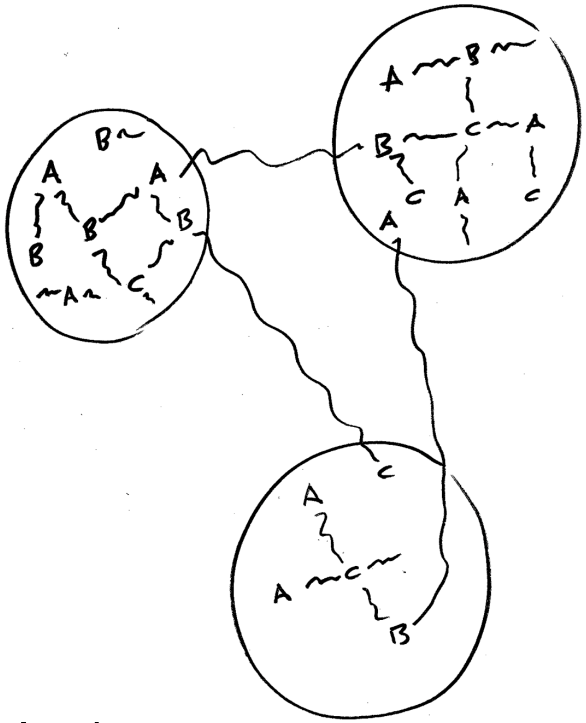
We're optimistic that this operation might exploit scratchpad

GRAPHAPP Conclusions

- In-degree code implemented
  - Tasking versions with various partitioning strategies
  - Loop-based versions
- Next steps
  - Experimentation
  - Simulation
  - Extend algorithm to enumerate triangles
  - Continue co-design
  - Talk with Lumsdaine (Indiana) about synergies

What Is PathFinder?

- Asking hard questions about relatively simple graphs
- Graph Pathway Analysis
  - Directed acyclic graphs
  - Find paths through sequentially labeled nodes
    - List of labels is termed a “signature”
  - Labeled nodes need not be adjacent
  - Paths are unconstrained
  - Simply testing for pathway existence (or not)
  - Multiple graphs, multiple signatures
- Next steps
  - Constrained solutions
    - Optimal pathways
    - Path lengths smaller than thresholds
  - Extended graph analysis (e.g. subgraph isomorphism)



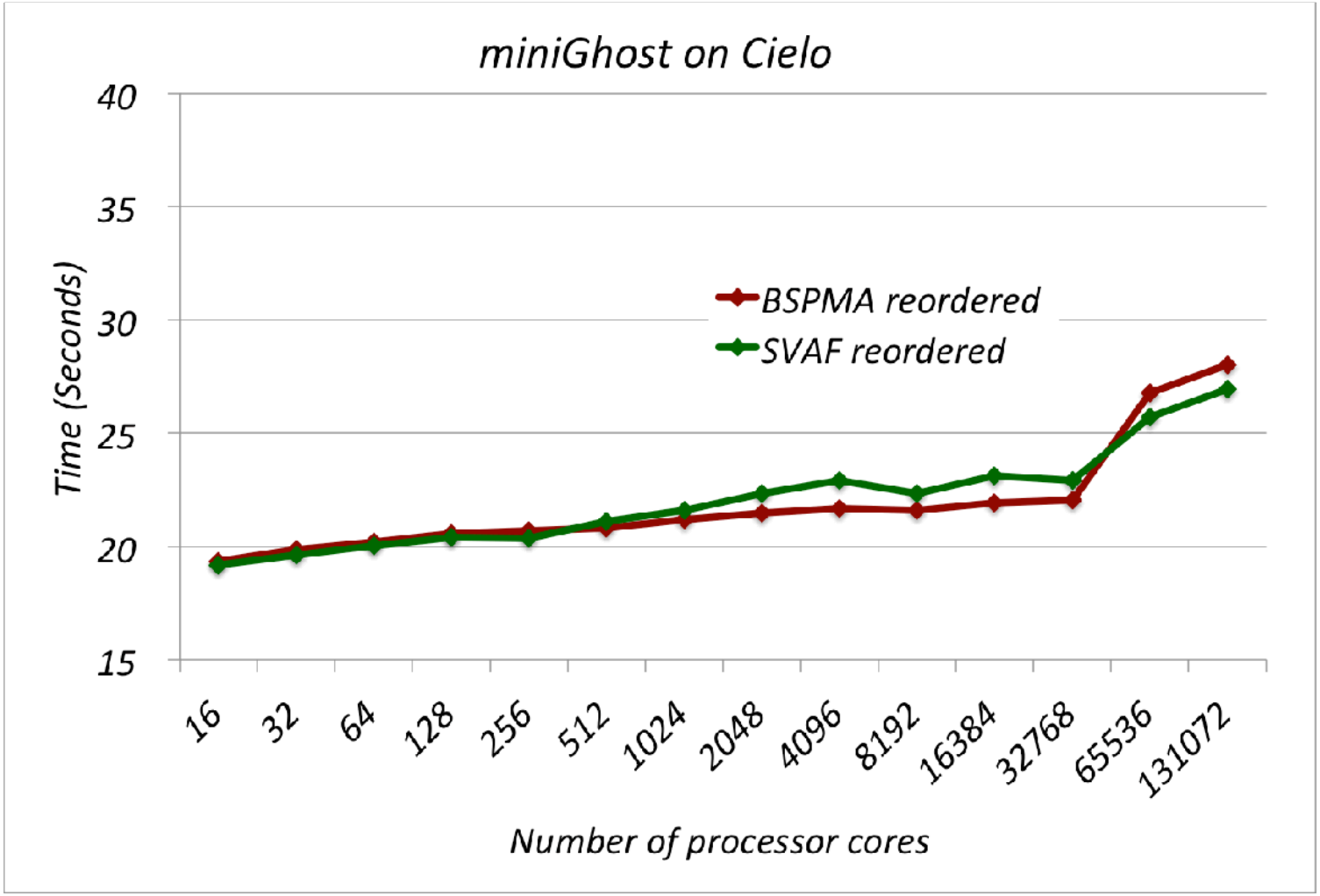
PathFinder Status

- C language (OO style)
- OpenMP task parallel
- OpenMP data parallel
- Qthreads next.
- Intel Xeon Phi specific “peer” developed

Future

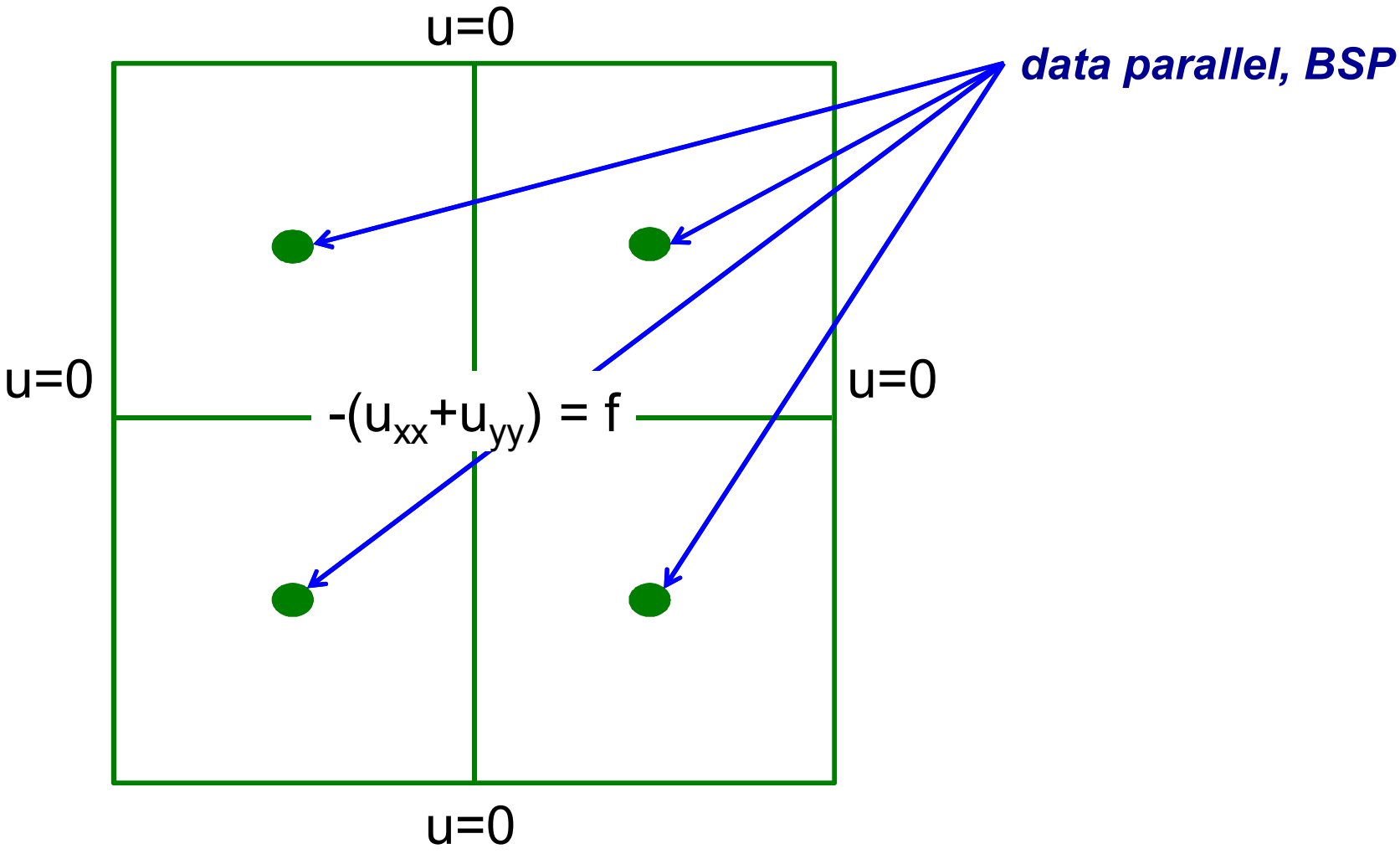
- Deeper program understanding tasks
- Finding dominators in graphs
- Value set analysis
- Noisy graphs
- Subgraph isomorphism
- Algorithm equivalency

BSP programming model



* Dedicated resource

Eulerian solver on a static mesh
miniGhost : data parallel



Task parallel over-decomposition

Benefits:

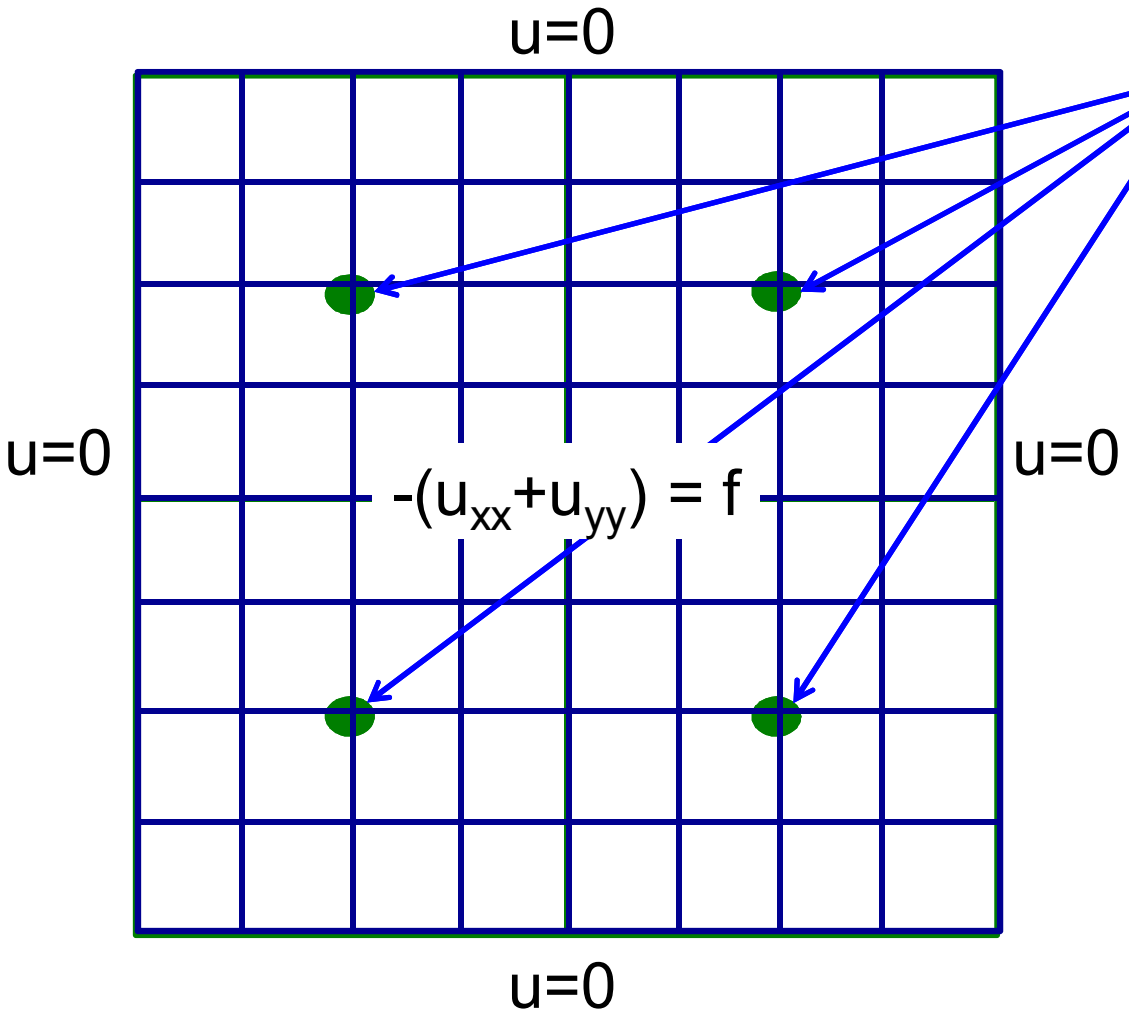
- Asynchronous behavior spreads the workload.
- Communication hiding.
- Reduced dependence on interconnect global bandwidth.
- Resilience enabling.

Requires:

- Coordination between MPI and “X” (MPI + Q)
- Dynamic adaptive scheduler.
- Increased interconnect message injection rate and bandwidth.

Shown to be viable in linear algebra (less MPI in tasks), not yet seen in finite difference/volume world. Programming model support, e.g. tbb, Charm++; qthreads+MPI. Sandia LDRDs (RAAMP, CarterEd), Sandia@CA group, Intel Phi team, ARM, others.

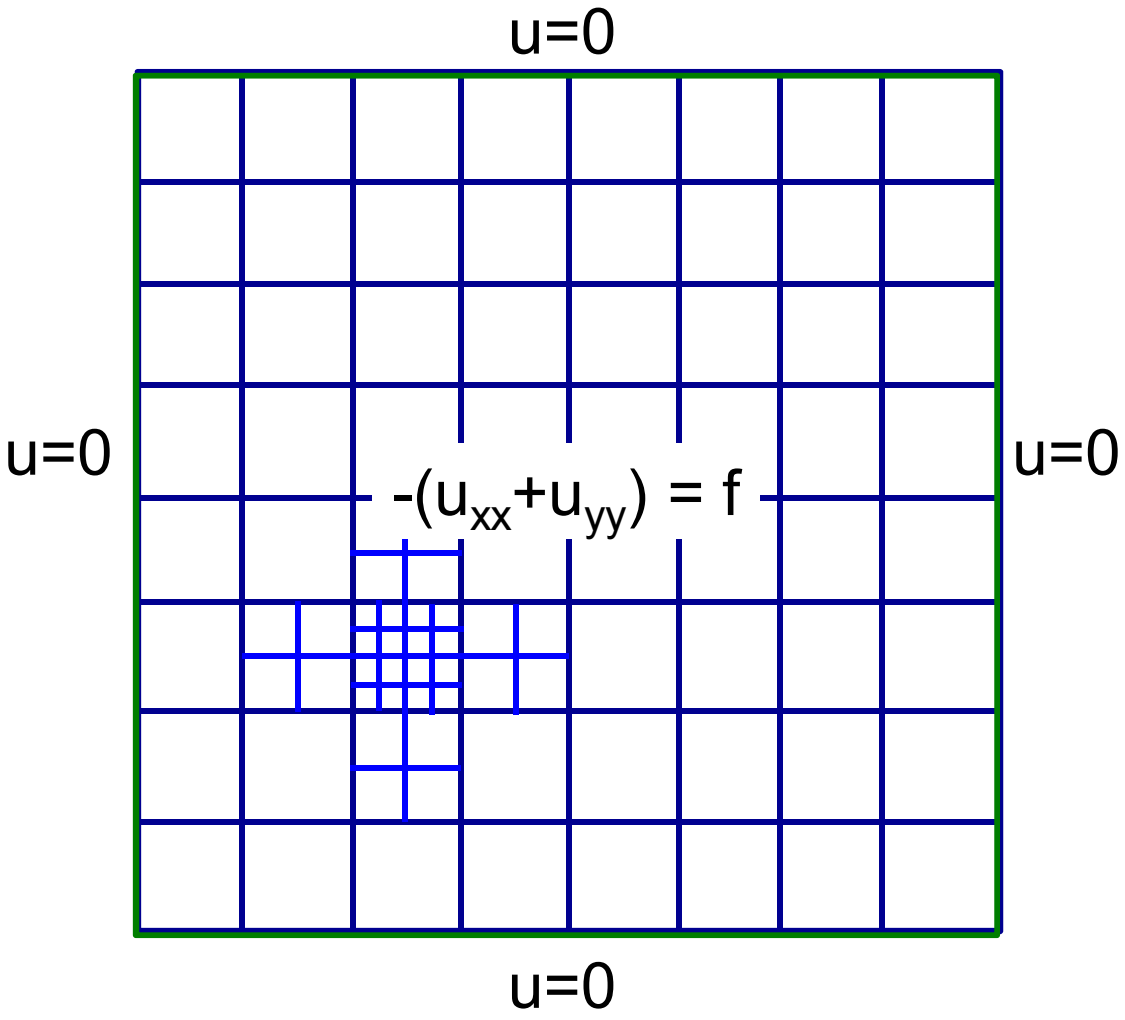
Eulerian solver on a static mesh miniGhost : task parallel



data parallel, BSP

Task parallel abstraction moves workload management decisions to the runtime system.

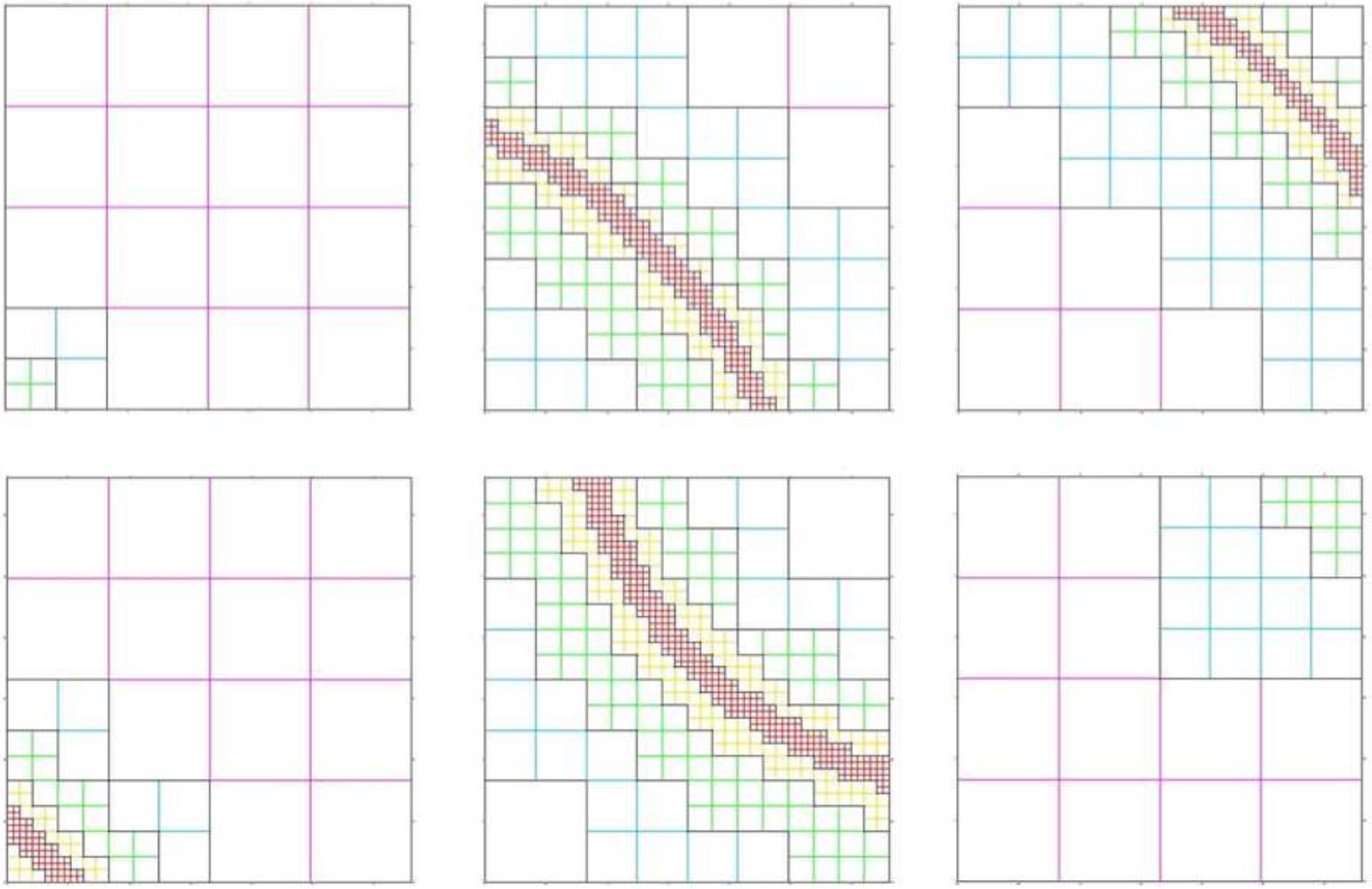
Eulerian solver on a dynamic mesh
miniAMR : currently data parallel, soon task



Task parallel abstraction moves workload management decisions to the runtime system.

Dynamic workload enables asynchronous behavior and "automatic" load balancing.

miniAMR visualization
Cray XE6, 64 cores



Performance model driven by:

- Tasks
  - Number, size, overhead
  - Workload : Computation, bc, communication, etc.
- Computation:
  - vectorization,
  - cache lines,
  - clock speed,
  - memory hierarchy
- Communication:
  - Latency and global bw
  - Number and length of messages
  - Msg inj rate and inj bw
  - Msg buffers: memory bw
  - MPI, less wait times
  - Contention

Summary of miniGhost/AMR task parallel

- Motivated by scaling issues due to BSP.
- Inspired by SPR
- OpenMP+MPI, MPI+Q implementations (so far)
- Sub-blocks
  - meta-data -> qthread binning.
  - ordering abstracted
    - Cartesian, random (Fisher–Yates-Durstenfeld shuffle)
- Plans:
  - Additional computation for data driven tasking
  - More sophisticated work queue ordering, e.g. task DAGs.
  - miniAMR task model

Shared concerns

Computations of DS&A and NW

- Programming model : task parallelism
- Processors : vectorization : 8-, 16-, 32-, 64-bit
- Intra-node : latency hiding through thread management, and coordination with
- Inter-node : interconnect support for many, small messages

- PathFinder and Solid Mechanics : shared search capabilities?