Scaling FMM with Data-Driven OpenMP Tasks on Multicore Architectures

Abdelhalim Amer (Halim)[1], Satoshi Matsuoka[2], Miquel Pericàs[3], Naoya Maruyama[4], Kenjiro Taura[5], Rio Yokota[2], and Pavan Balaji[1]

[1] Argonne National Laboratory, USA

[2] Tokyo Institute of Technology, Japan

[3] Chalmers University of Technology, Sweden

[4] RIKEN Advanced Institute of Computational Science, Japan

[5] University of Tokyo, Japan

IWOMP 2016, 12th International Workshop on OpenMP, Oct 05-07, 2016

Dynamic Irregular Methods

- Characteristics
  - Irregular data access patterns
  - Complex flow control
  - Imbalanced workloads
  - Patterns, flows, and workloads known at run-time
- Challenges
  - Inherent parallelism, but difficult to exploit with regular parallel methods (e.g. parallel loops)
- Scalability requirements on modern multicore systems
  - Fine-grained concurrency
  - Avoid/reduce bulk-synchronization
  - Data locality incentives

Parallelization Methods Landscape

- **High parallel slackness** (informally): Parallel work >> P (number of processors)
  - Over decomposition
- **Runtime costs**: work unit management, dependency tracking, …
- **Locality opportunity**: amenable to locality optimizations
  - Loop tiling; NUMA-awareness

Overly simplified classification (naïve)

Bulk-Synchronous Coarse-Grained → **Bulk-Synchronous** Fine-Grained → **Async Tasks** Coarse-Grained → **Async Tasks** Fine-Grained

- Data-Locality Opportunities

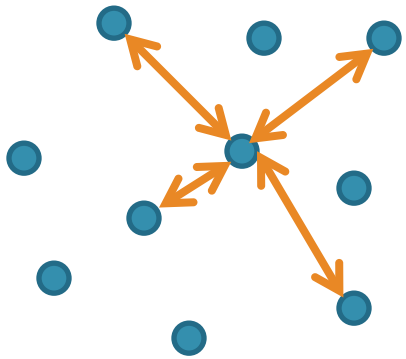
- Parallel Slackness
- Runtime Costs

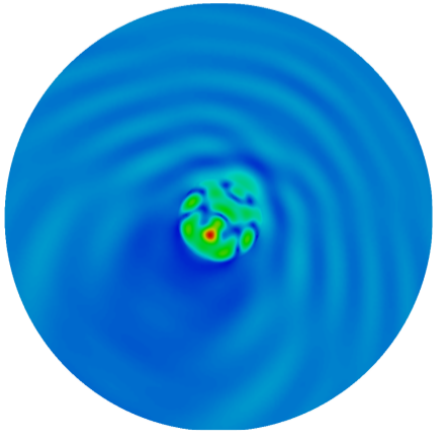
Goal: Characterize parallelization methods for dynamic irregular methods using **FMM** as use-case

Fast Multipole Methods (FMM)

- Solve N-Body problems
- O(N) complexity
- Dynamic irregular method
- Applications in several domains

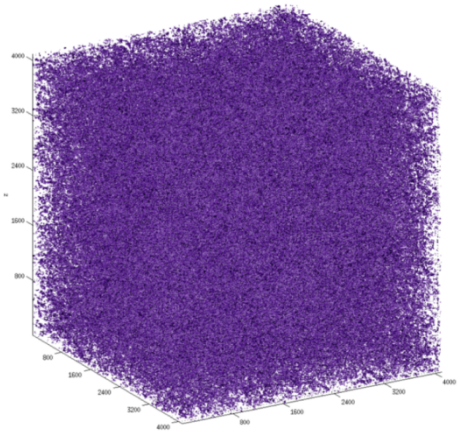
N-Body Problem
$O(N^2)$ interactions



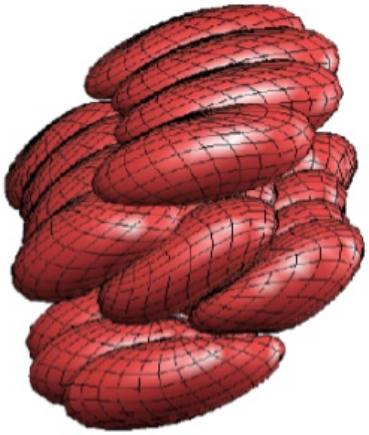


Electro-Dynamics: Multi-Level Fast Multipole BEM for 3-D Elastodynamics, Computer Methods in Applied Mechanics and Engineering 197 (2008)

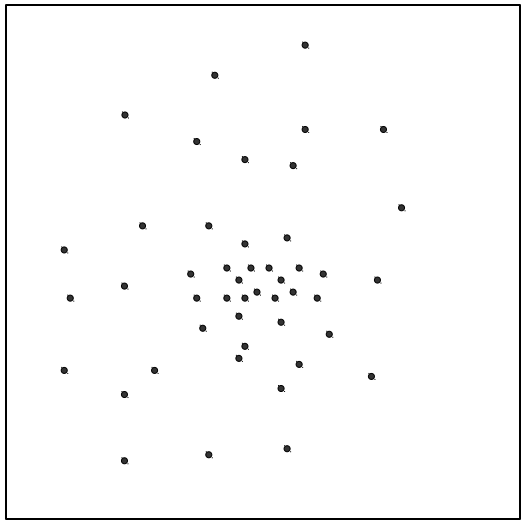
Fluid Dynamics:
Petascale turbulence simulation using a highly parallel fast multipole method on GPUs (2012)



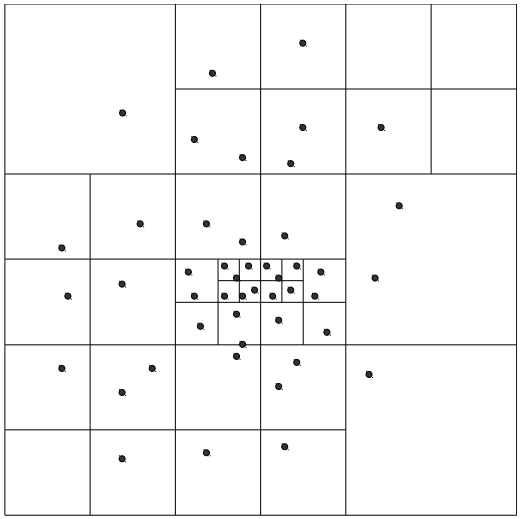
Blood-Flow:
Petascale direct numerical simulation of blood flow on 200k cores and heterogeneous architectures, SC 2010



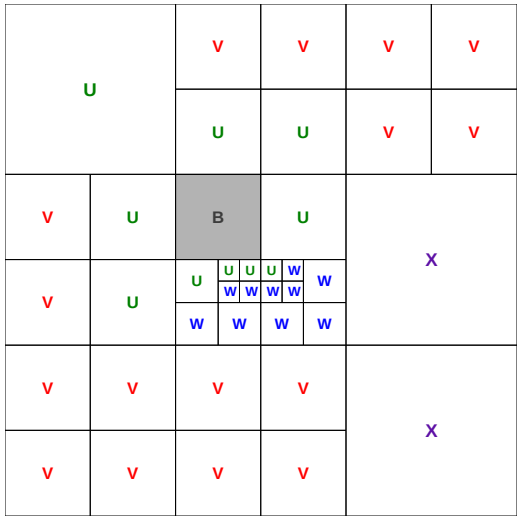
Kernel-Independent FMM: Domain Decomposition

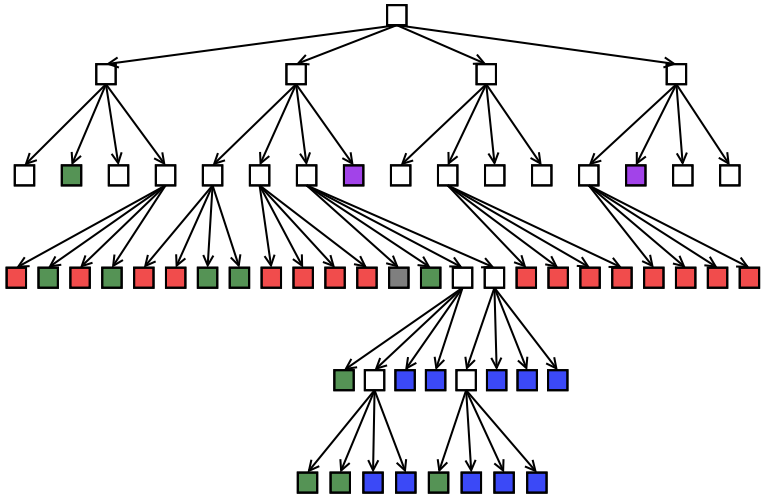


Refined mesh
<= *q* bodies per box

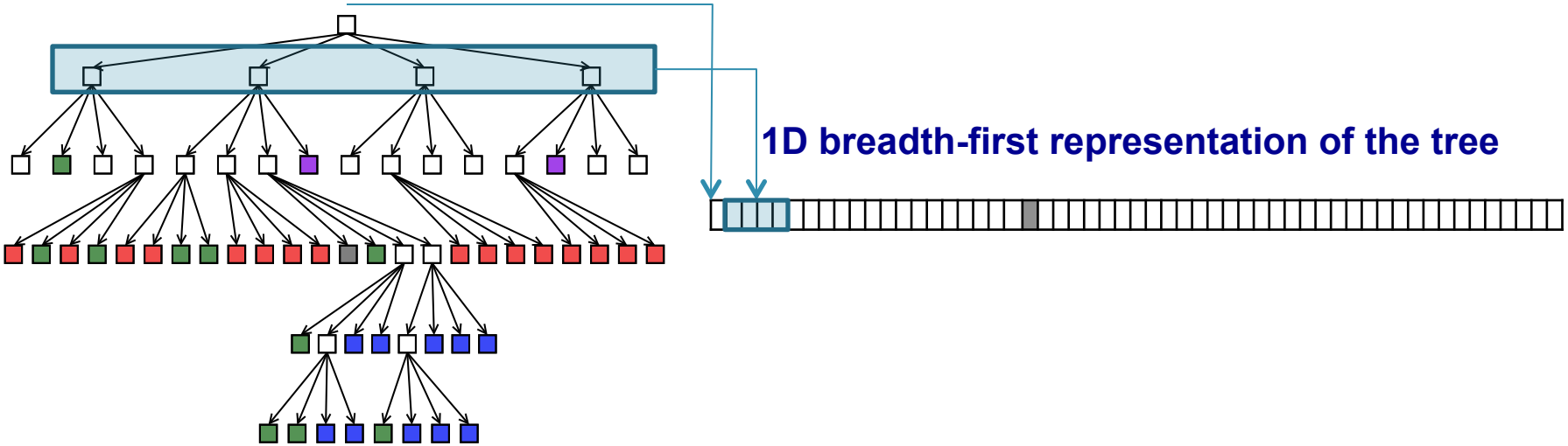


Building interaction lists, U, V, X, and W



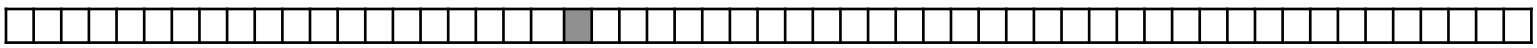


Kernel-Independent FMM: Data Structures



1) Meta Data

Interaction lists



- U: 6,22,24,…
- V: 21,23,…
- X: 12,18,…
- W: 43,44,…

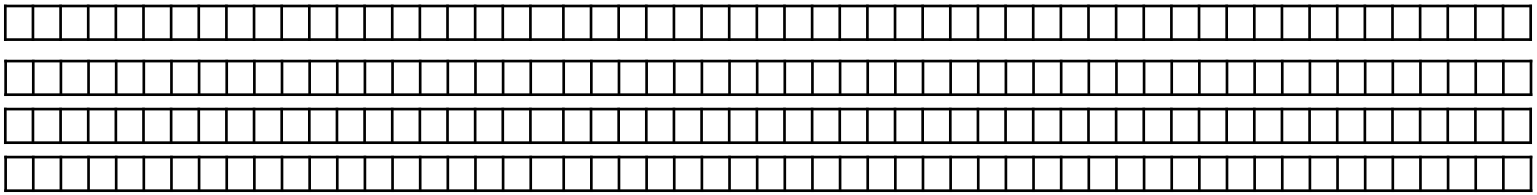
2) Computational Data

Source bodies

Target bodies

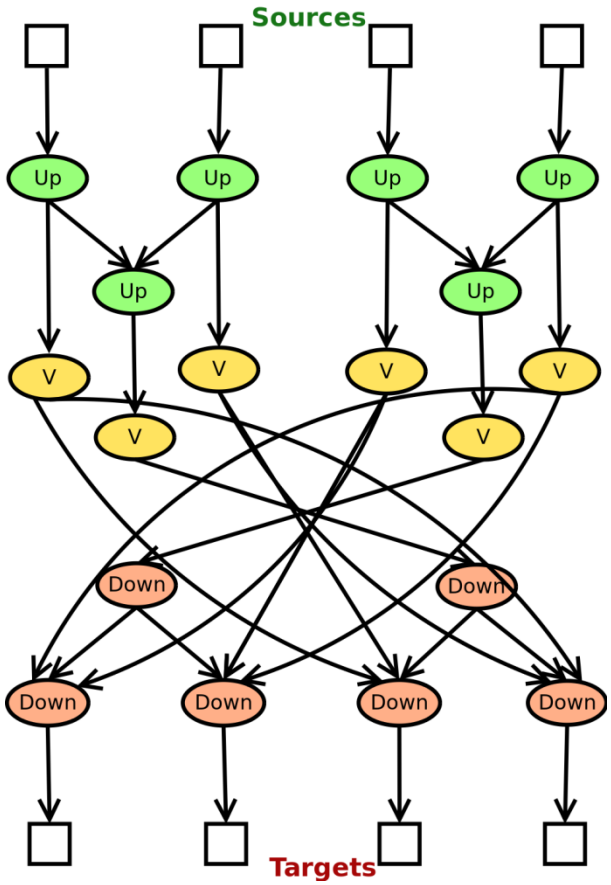
Equivalent Density

Check surface

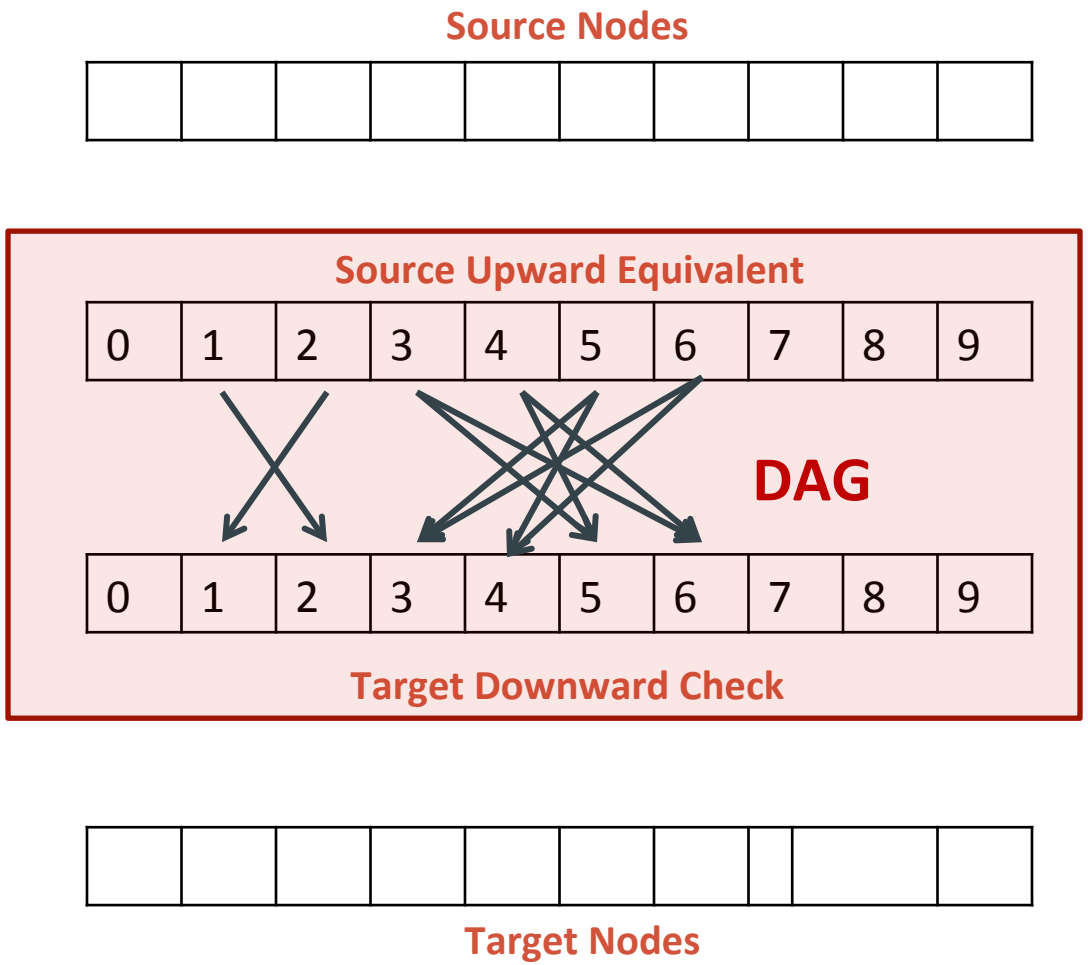


…….

KI-FMM Computational Patterns (1/2)



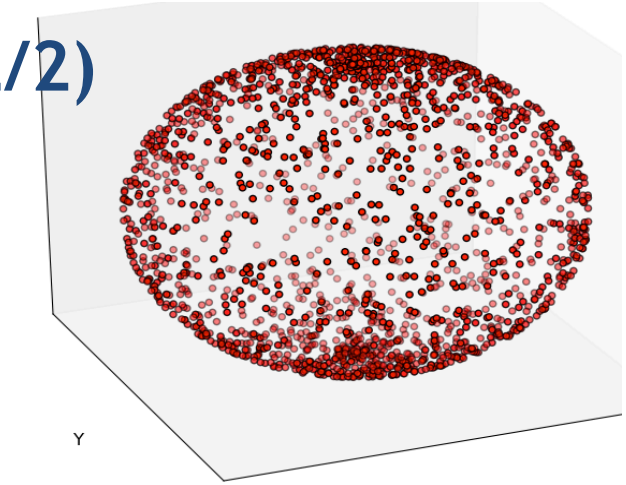
Far-field computation fine-grained task graph



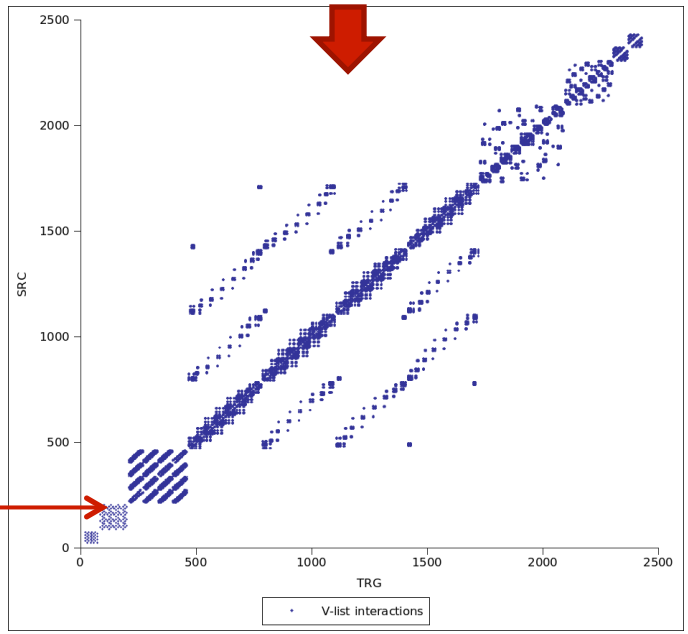
Relative data structures

KI-FMM Computational Patterns (2/2)

- Most fine-grained tasks operate on single (src, trg) pair of boxes
- Various types of computation
  - Pairwise interactions (U-list, upward, …)
  - DGEMMs (upward, downward)
  - FFTs (V-list)
  - Pointwise products (V-list)



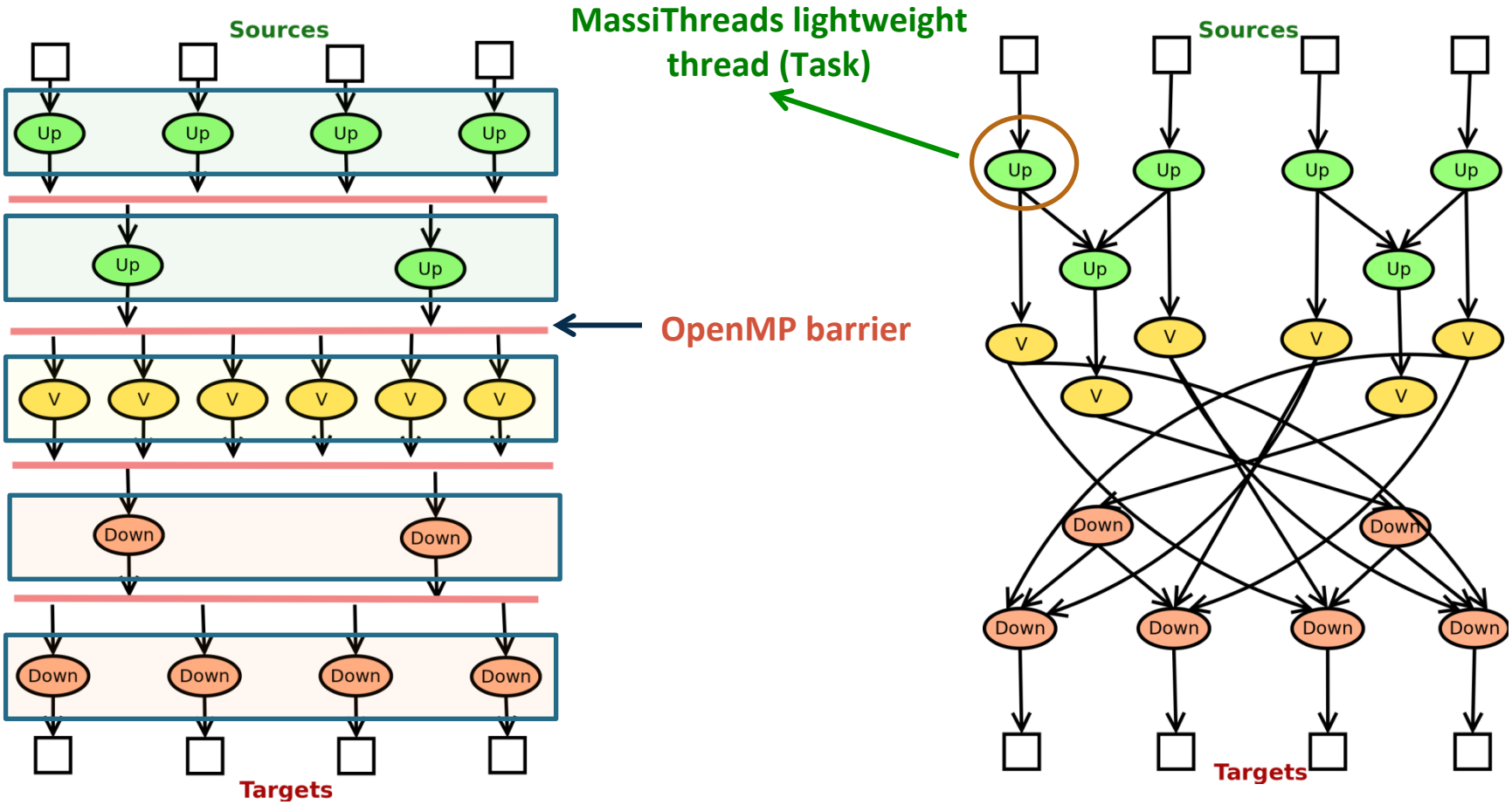
Elliptical body distribution



V-List computational pattern adjacency matrix

- Each point is a pointwise product
  - Loads 96KB
  - Does 16K DP floating point ops

Kernel Independent FMM Implementations



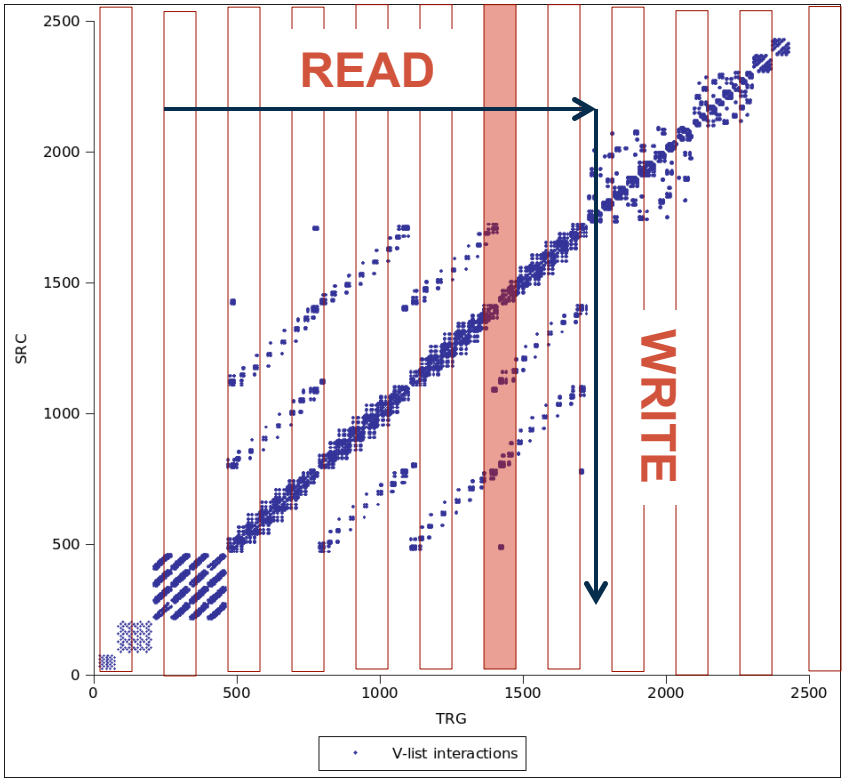
Bulk-Synchronous [1]

Fine-Grained Data-Driven [2]

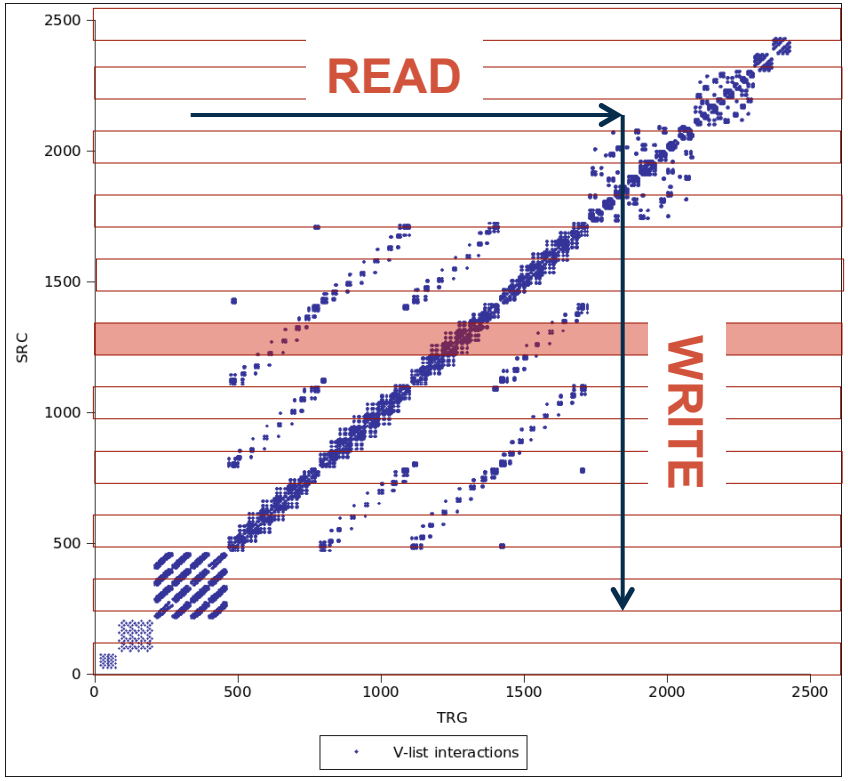
[1] Chandramowlishwaran, Aparna, et al. "Optimizing and tuning the fast multipole method for state-of-the-art multicore architectures." Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on. IEEE, 2010., IPDPS 2010

[2] Amer, Abdelhalim, et al. "Fork-Join and Data-Driven Execution Models on Multi-core Architectures: Case Study of the FMM." Supercomputing. Springer Berlin Heidelberg, 2013.

1D Source- vs. Target-Centric Patterns

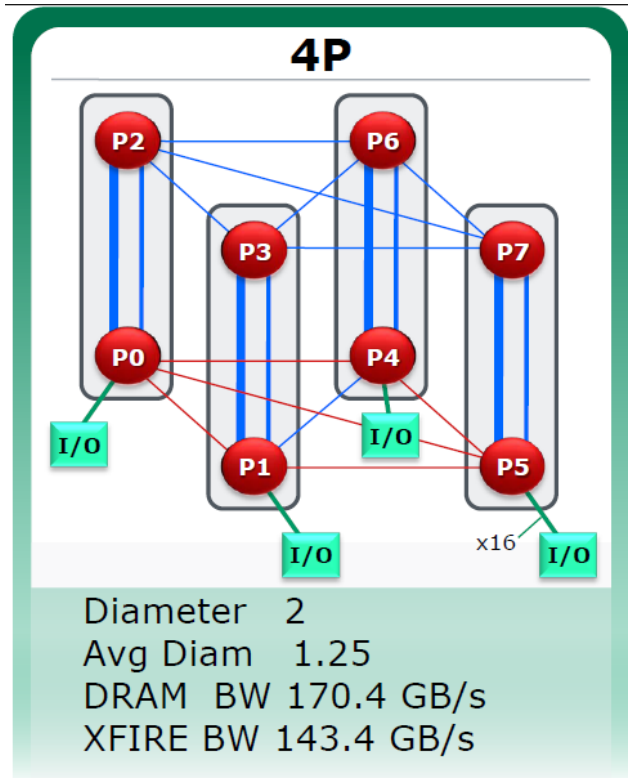


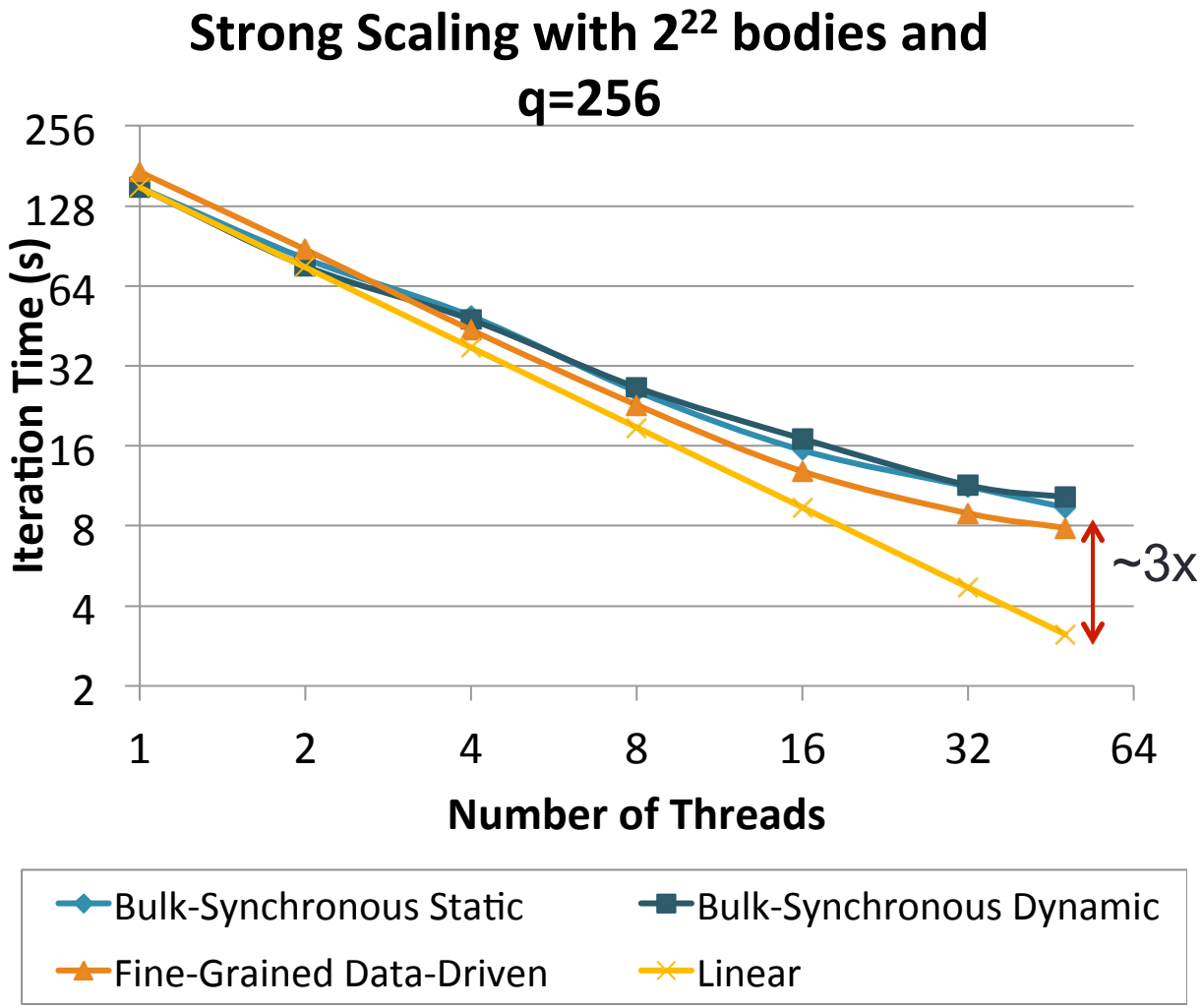
Bulk-Synchronous = target-centric



Fine-Grained Data-Driven = source-centric

Scalability Collapse



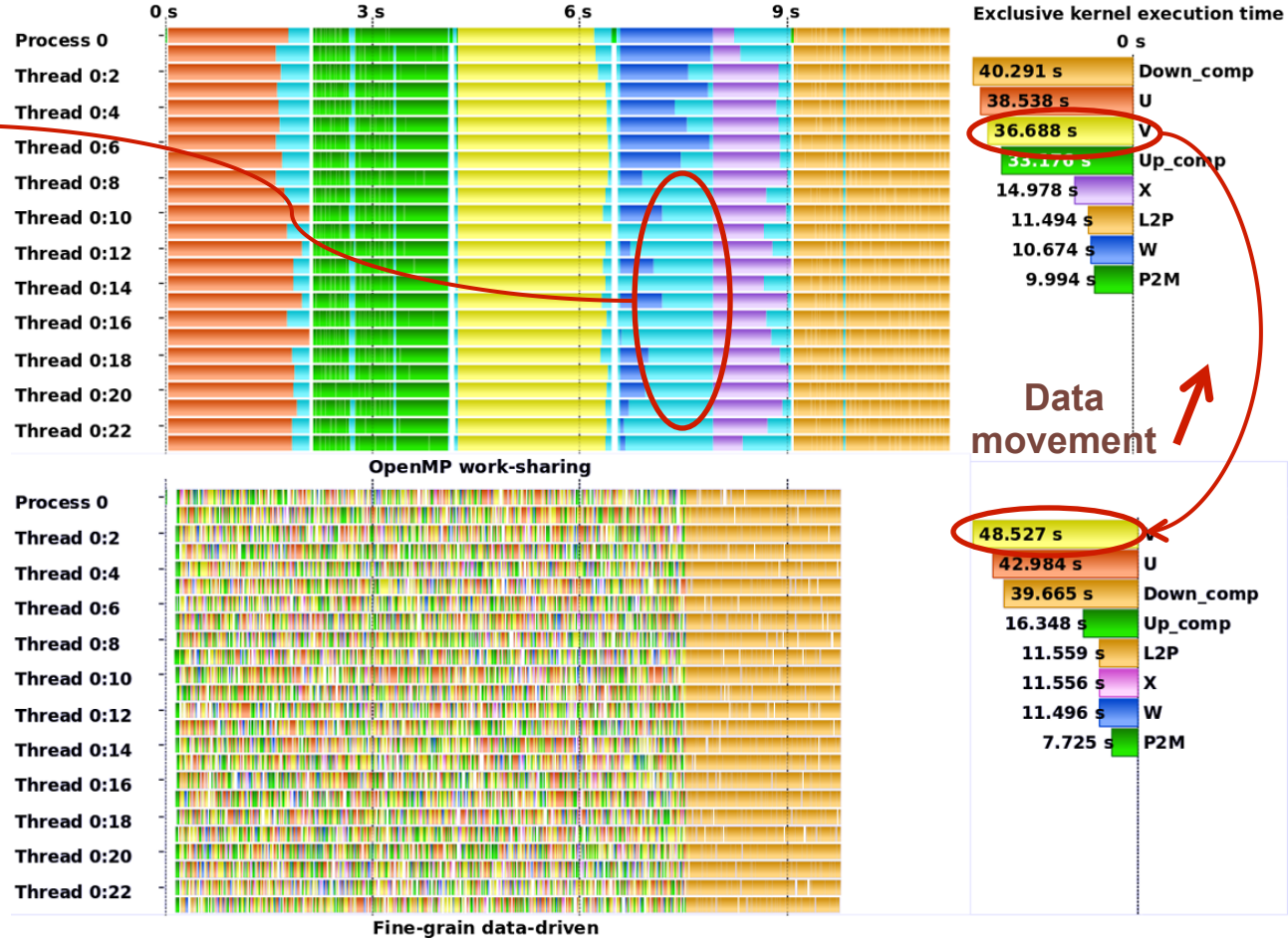


AMD Magny-Cours
(Hotchips 2009 presentation)

+ GCC 4.9.2

Parallel Slackness & Data Locality Tradeoffs

Significant idleness from bulk-synchronous steps and static scheduling



- Data-Driven effectively eliminates idleness
- Improves tree-producer-consumer data reuse
- Makes things worse for communication intensive kernel (V)

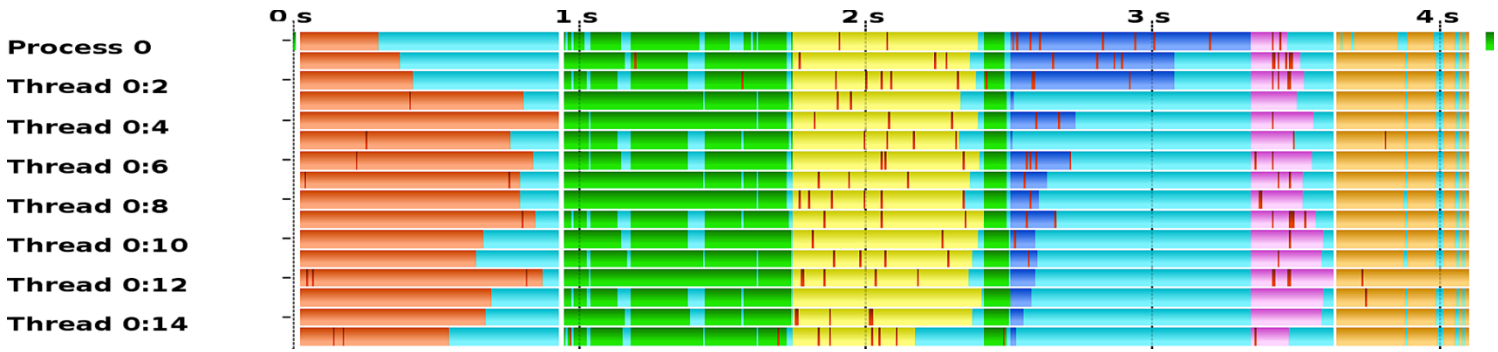
Static vs. Dynamic Scheduling in Bulk-Synchronous

```
void * V-list-phase (){
   // Traverse all target nodes in parallel
   # pragma omp parallel for schedule ( OMP_SCHED )
   for ( trg =0; trg < trgNodeMax ; trg ++){
      // Accumulate the contribution of all
      // source nodes into the target
      for ( src in Vlist ( trg ))
        compute_V ( trg , src );
   }
} // Implicit Barrier Synchronization
```

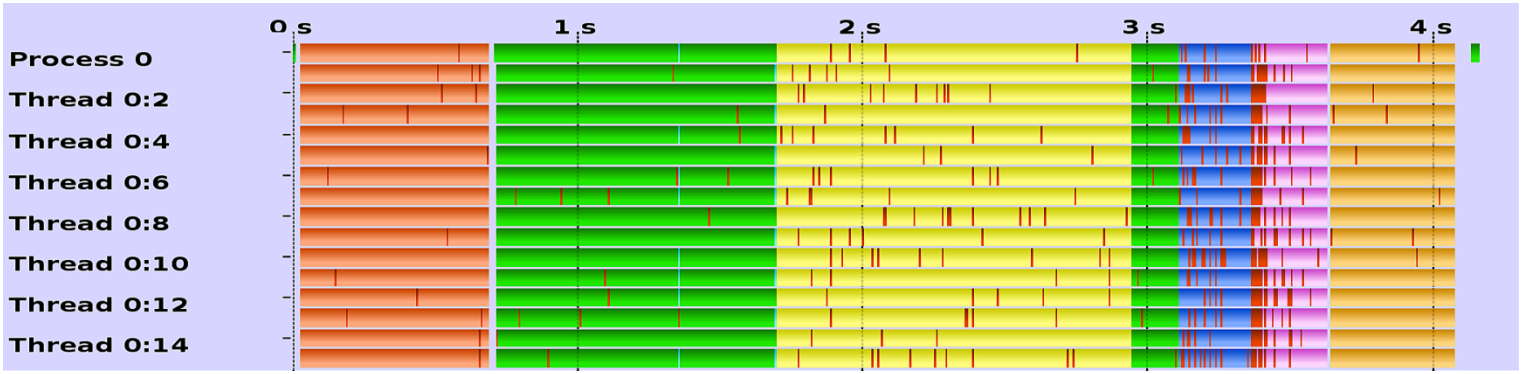
Static

Dynamic

Static



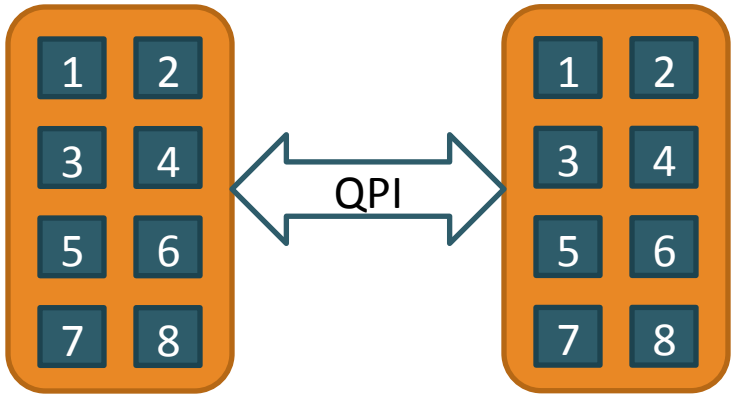
Dynamic

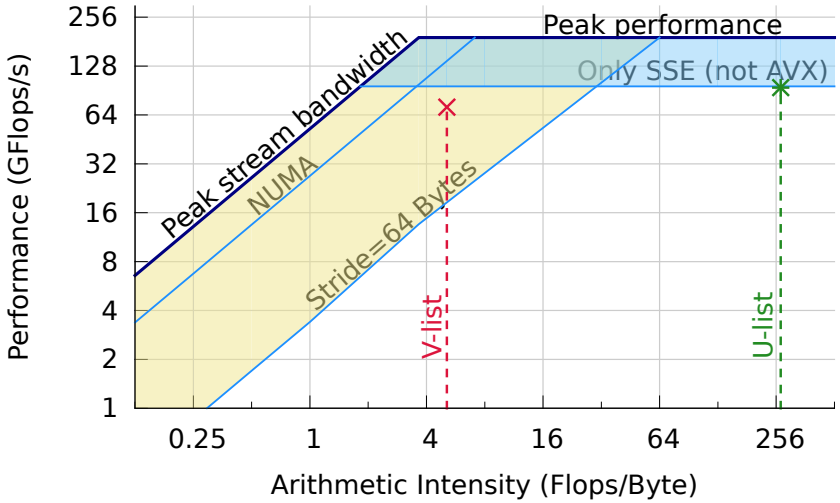


Optimizing for Cache First

- Facts on a 2-socket Sandy Bridge EP
  - 64B strided memory accesses drop BW by 4x
  - Remote memory accesses drop BW by 2x
- Rule of thumb: optimize for cache before optimizing for NUMA
- NUMA-awareness in KIFMM is challenging
  - Difficulty to keep data local across stages
- Current implementation: some data is first-touched by a single thread, other touched by multiple threads

2-Socket Sandy Bridge EP

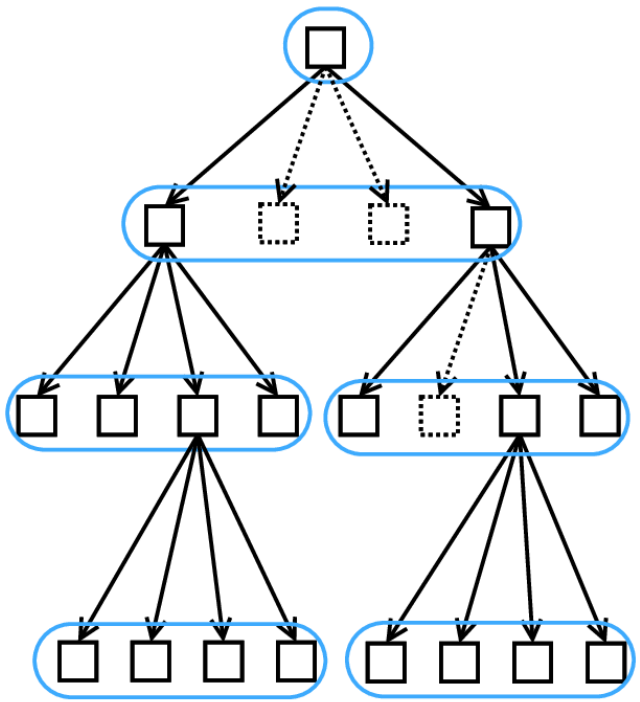




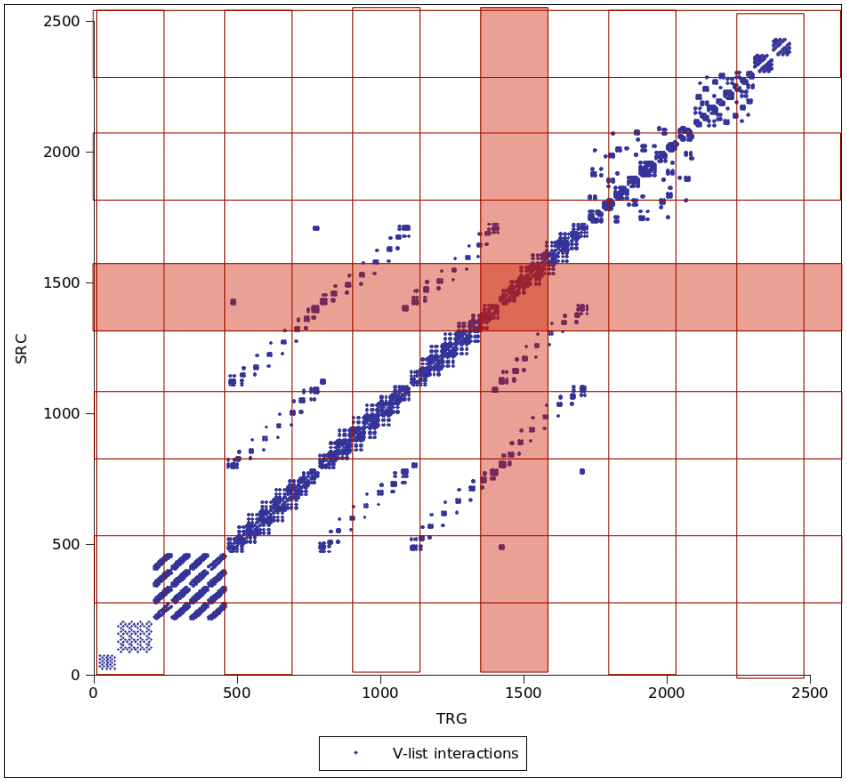
V-list and U-list arithmetic intensity (with respect to DRAM accesses) on the roofline model of Sandy Bridge EP

2D Tiling of Computational Patterns

- Two main goals
  1. Spawn work units with high data locality potential
  2. Control granularity (unlike the lightweight thread implementation)



Data Partitioning has to take into account the tree data structure



2D V-list Partitioning

Tiled Dynamic Bulk-Synchronous Implementation

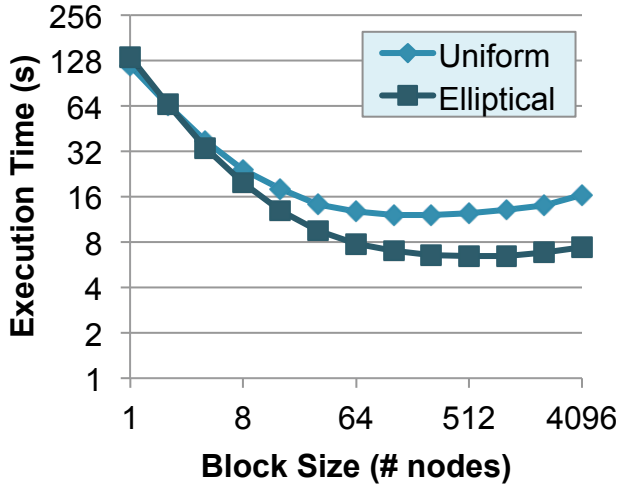
- 2-dimensional tiling of each parallel loop
- Same tile (block) size across all loops

```
void * V-list-phase (){
   // Traverse all target blocks in parallel
   # pragma omp parallel for schedule ( dynamic )
   for ( i =0; i < trgNodeMax ; i += BS ){
      // Traverse all source blocks
      for ( j =0; j < srcNodeMax ; j += BS ){
         // Traverse the targets in the block
         for ( trg = i ; trg < BS ; trg ++){
           // Traverse the sources in the block
           for ( src = j ; src < BS ; src ++){
            // Accumulate the contribution of all
            // source nodes into the target
             if ( src in Vlist ( trg ))
               compute_V ( trg , src );
           }
         }
      }
   } // Implicit Barrier Synchronization
}
```

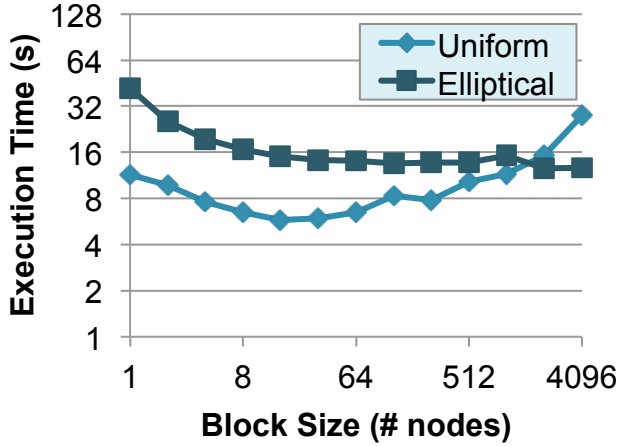
Block Size

Tuning and Analysis of the Tiled Bulk-Synchronous Method

Sandy Bridge EP



Magny-Cours

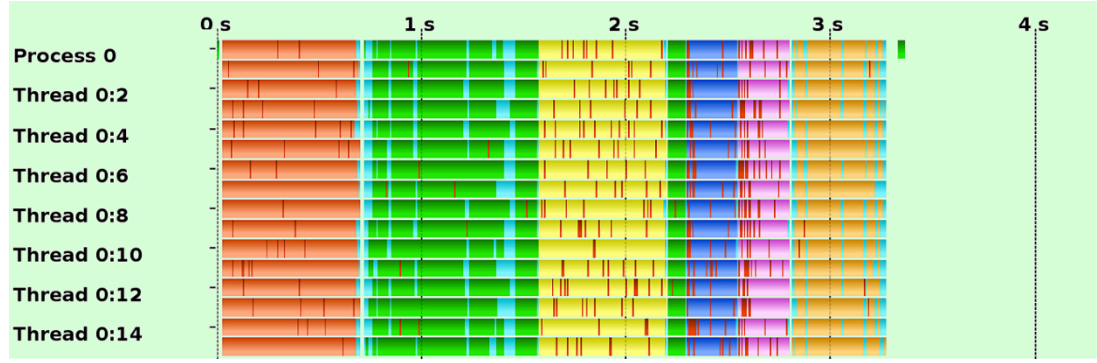


Tuning the Block Size (BS)

Static

Dynamic

Dynamic + Tiling



Tiled Implementation with OpenMP Tasks

```
# define DATA_OUT eff_val [beg_eval : trg_stride]
# define DATA_IN eff_den [beg_eden : src_stride]
void * V - list - phase ( trg ){
  // Traverse all target blocks
  for ( i =0; i < trgNodeMax ; i += BS ) {
    int trg_stride = eff_trg_size * BS ;
    int beg_eval = trg_stride * i ;
    // Traverse all source blocks
    for ( j =0; j < srcNodeMax ; j += BS ) {
      int src_stride = eff_src_size * BS ;
      int beg_eden = src_stride * j ;
      # pragma omp task depend (out:DATA_OUT) depend(in: DATA_IN )
        // Traverse the target V-list blocks
        for (n=i; n < i+BS; n+=VBS)
          // Traverse the source V-list blocks
          for (m=j; m < j+BS; m+=VBS)
            // Traverse targets in a V-list block
            for (trg=n; trg < n+VBS; trg ++)
              // Traverse sources in a V-list block
              for (src=m; src < m+VBS; src ++)
                // Accumulate the contribution of all
                // source nodes into the target
                  if (src in Vlist(trg))
                    compute_V(trg ,src);
  }}}
```

Executed within
#pragma omp parallel
{
#pragma omp single
}

IN/OUT dependencies on array sections

Async Task

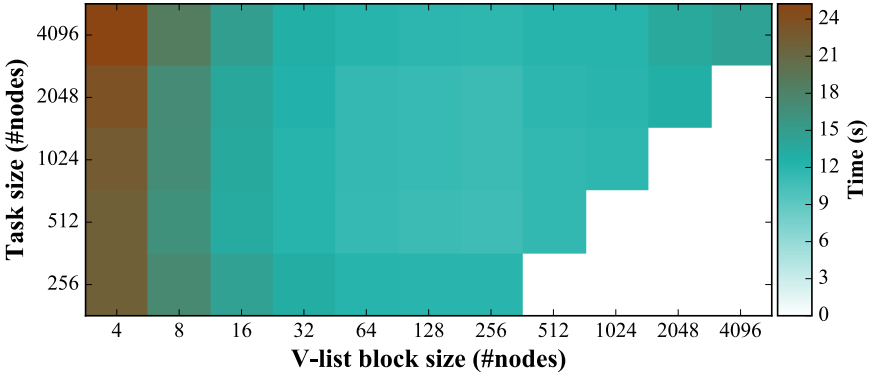
Second tiling factor for V-list to reduce cache-thrashing

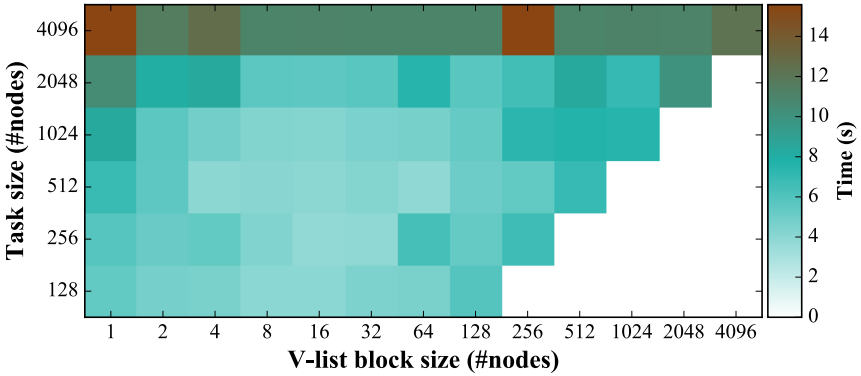
2D Tuning of the Data-Driven OpenMP Tasking Method

- 2^{22} bodies; q = 256
- Full concurrency
- Compilers: **ICC 15** (Sandy Bridge) **and GCC 4.9.2** (Magny-Cours)

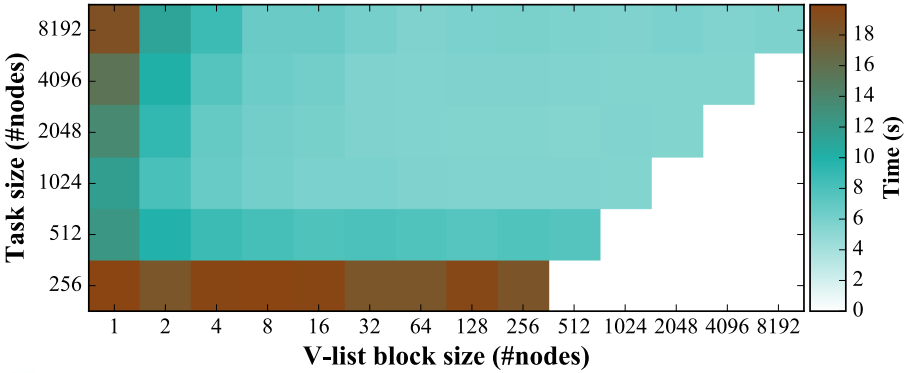
Sandy Bridge EP (24 threads) | **Magny-Cours (48 threads)**

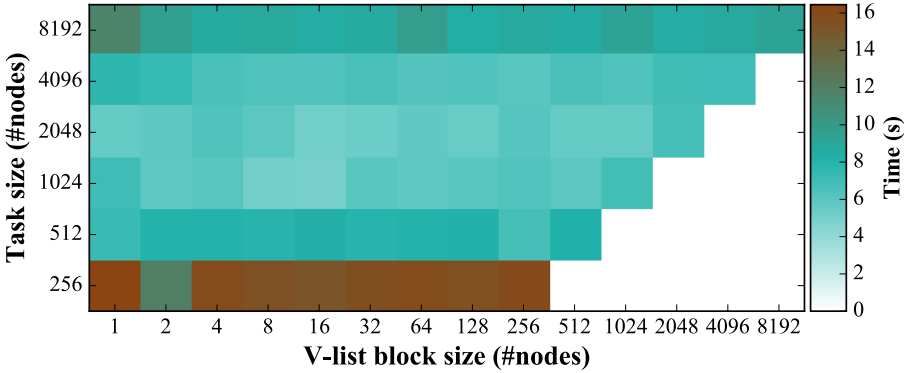
Uniform





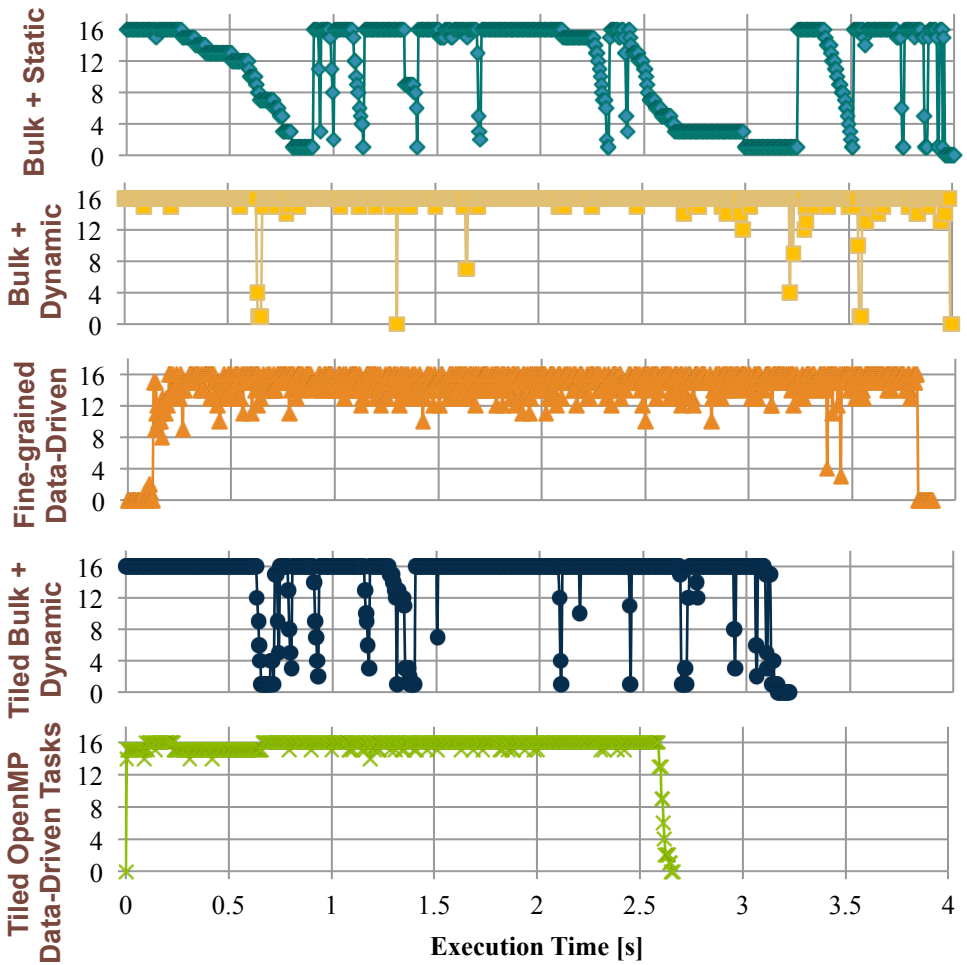
Elliptical



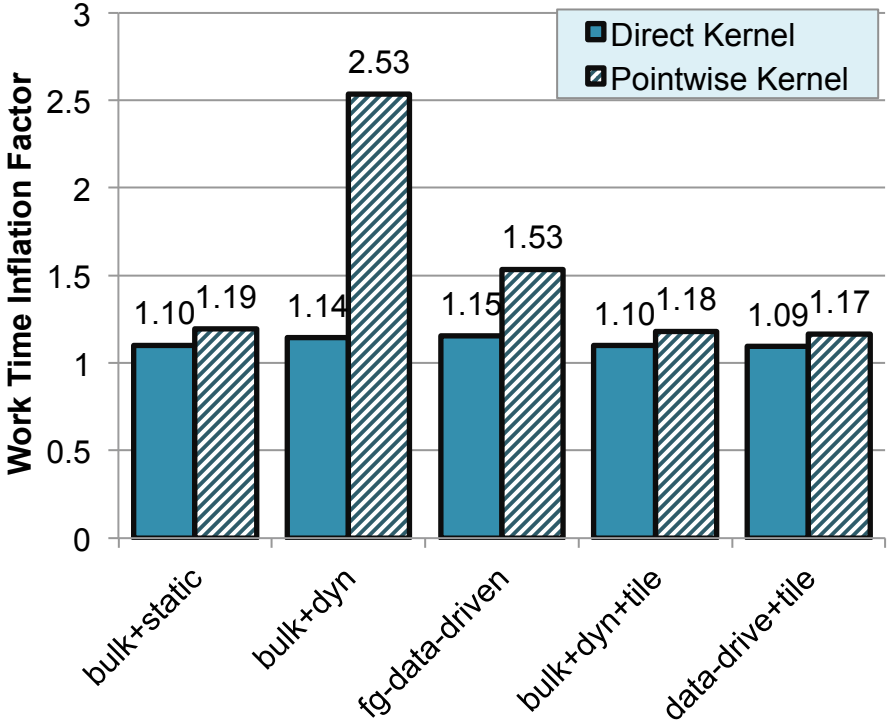


Profiling Idleness and Work-Time Inflation on Sandy Bridge EP

- 2^{22} bodies; q = 256
- Full concurrency
- Compilers: **ICC 15** (Sandy Bridge) **and GCC 4.9.2** (Magny-Cours)



Profiling Parallel Idleness
(metric = concurrently running tasks)



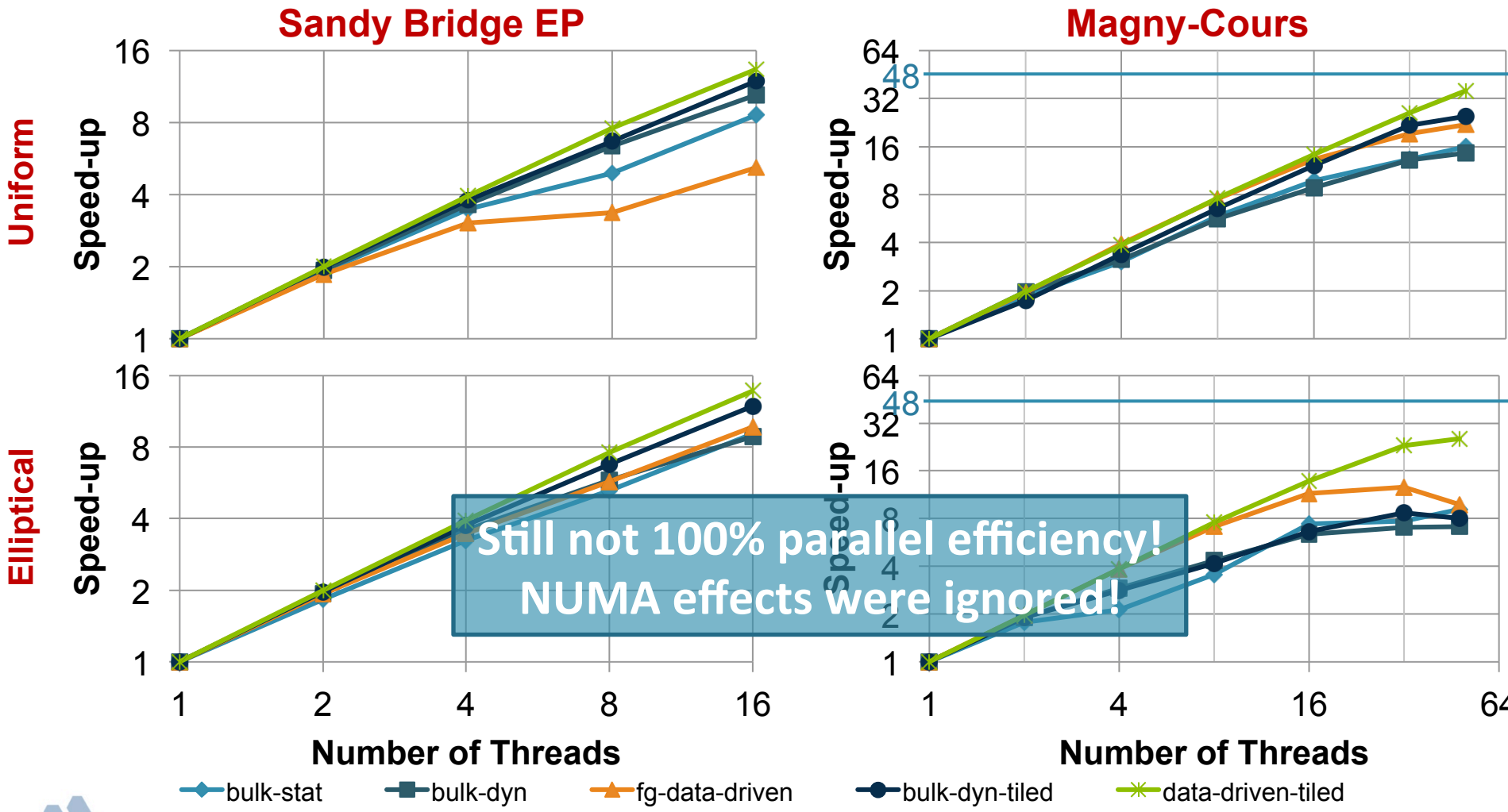
Profiling Work-Time Inflation (WTI) [1]

$$r = \frac{\sum_{th} Work_{th}}{Work_{seq}}$$

[1] Stephen L. Olivier, Bronis R. De Supinski, Martin Schulz, and Jan F. Prins. Characterizing and mitigating work time inflation in task parallel programs. Scientific Programming, 21(3):123–136, 2013.

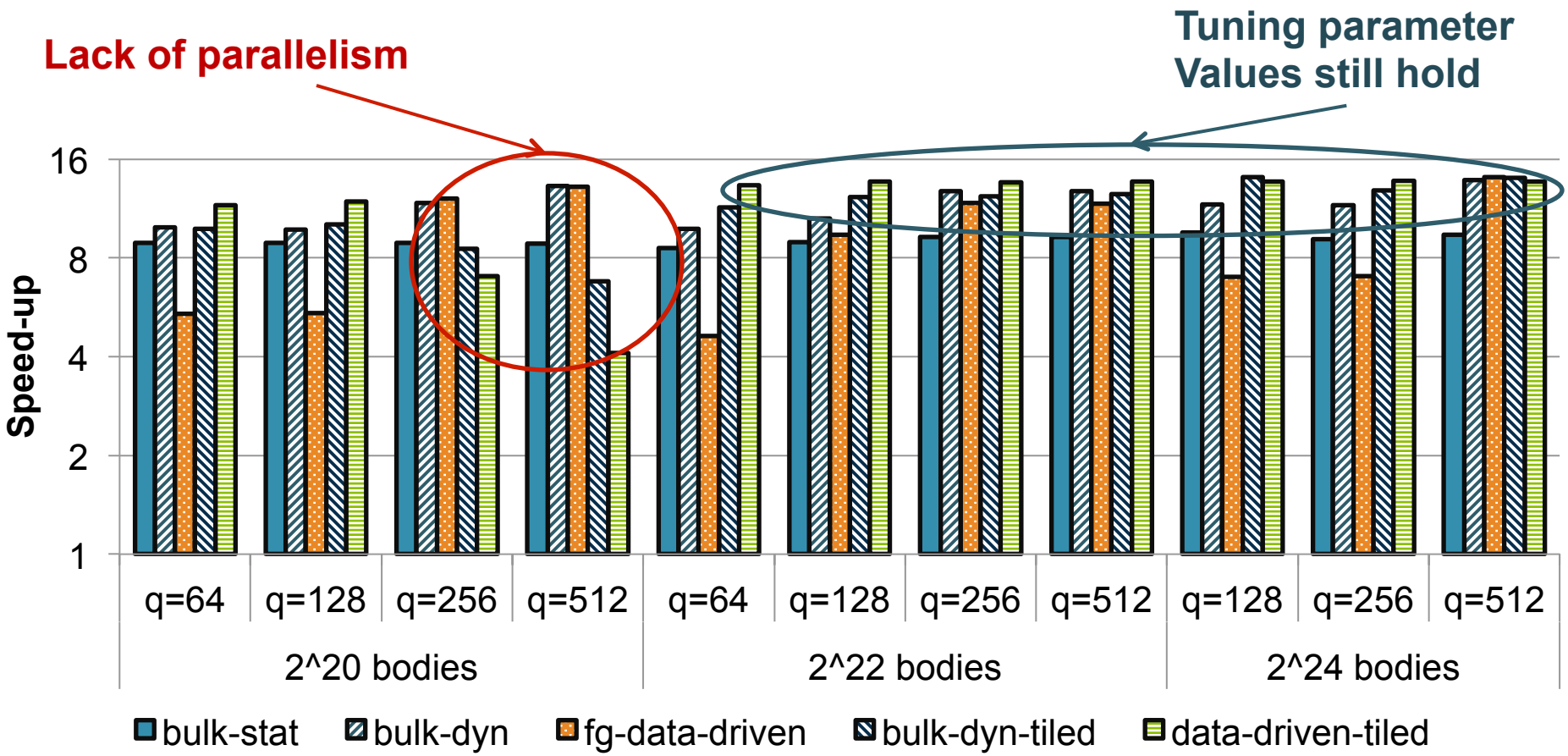
Strong Scaling Performance Comparison

- 2^{22} bodies; q = 256
- Full concurrency
- Compilers: **ICC 15** (Sandy Bridge) **and GCC 4.9.2** (Magny-Cours)



Portability of the Tuning Parameter Values

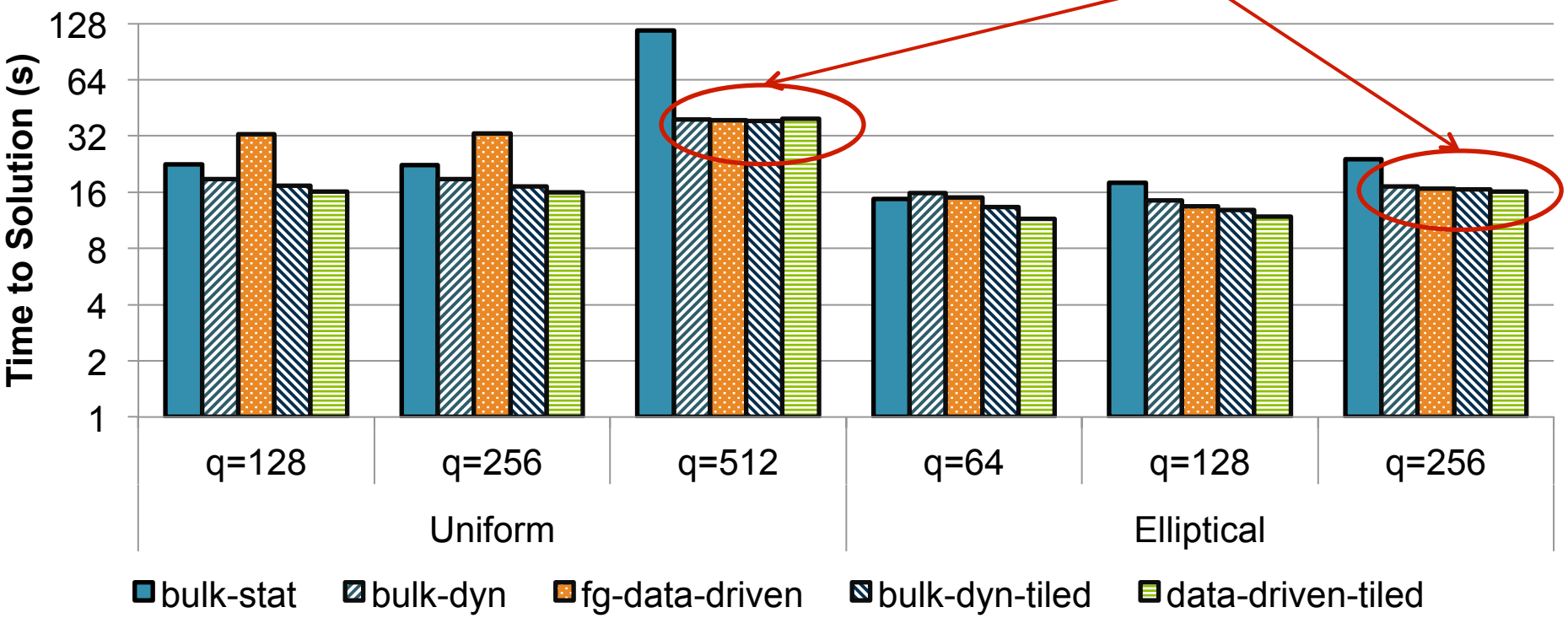
- Sandy Bridge EP; 16 Threads
- Tiled implementations tuned for 2^{22} bodies and q = 256



Scalability != Efficiency

- Sandy Bridge EP; 16 Threads
- Large problem: 2^{24} bodies

Fine-grained data-driven and dynamic bulk-synchronous scale well but overall execution is slower



Lessons Learnt

- **Scalability**
  - A dynamic approach requires data-locality incentives to promote data reuse and reduce cache thrashing
  - Tiling computational patterns proved to be successful with proper tuning
  - Tuning parameter values are fairly portable across input problem sizes as long as parallel slackness is not hindered significantly
- **Programmability**
  - Manual tiling is gross and error prone!
  - Sparse data dependencies are difficult to express
  - Array sections express unnecessary denpendencies
- **Looking ahead**
  - Ideally, the runtime should cluster fine-grained tasks to optimize for locality
  - Fine grained tasks imply high scheduling overhead which makes it very difficult