Scalable and composable shared memory parallelism with tasks for multicore and manycore

Thomas Guillet, Intel

Marc Tchiboukdjian, UVSQ

TERATEC 2012 Forum, Palaiseau, 27-28 June 2012

Some application challenges on an Exascale node

1 Exascale node

- Energy dominated by data movements
- O(1000) cores / node
- Growing impact of machine jitter
- Algorithmic load balancing becomes a critical issue

> - How can we reduce data movements in applications?
> - How far can SPMD take us in terms of scalability?

Shared memory programming matters

- Expose as much parallelism as possible
- Benefit from dynamic load balancing
- Use locality-aware algorithms

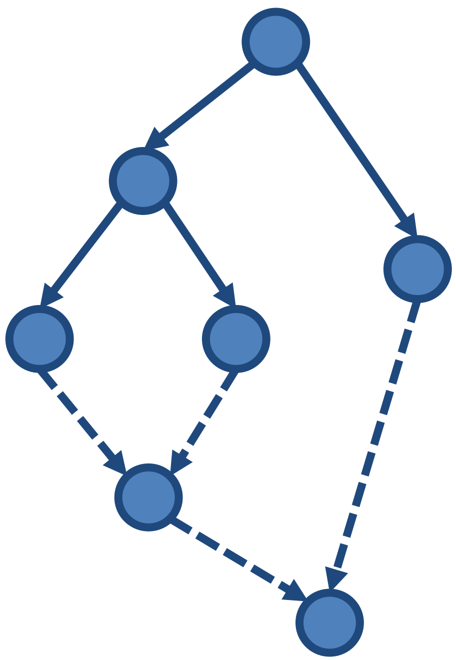
> Already benefits increasingly parallel architectures:
> - Multicore: O(10) cores
> - Intel MIC: > 50 cores

In this talk: an illustration of cache-friendly **task-based parallelism** on a kernel for unstructured FEM meshes

Tasks With Intel® Cilk™ Plus

What are tasks?

- A way of expressing opportunities for independent computations (function calls, code blocks, …)
- No explicit reference to threads



Intel® Cilk™ Plus: C/C++ language extensions for tasks in shared memory

- **spawn** to create a task
- **sync** to wait for the completion of tasks

Cilk Plus Features

- Automatic scheduling and load balancing
- Available in Intel compilers, and as open-source for GCC 4.7 (http://cilkplus.org)
- Same source code targets multicore and MIC
- Parallelism is introduced recursively: well suited for Divide and Conquer (D&C) algorithms

D&C algorithms:

- Expose fine-grain task parallelism
- Are inherently cache-friendly

Why task parallelism in applications?

With MPI and OpenMP: parallelism is explicit

- Parallelism is mandatory and relies on a fixed number of participating workers
- This breaks nested parallelism
  - Either no additional parallelism
  - Or may result in oversubscription

Only one level of coarse grain parallelism

Tasks offer *composable* parallelism

- Parallelism can be expressed anywhere in the application, libraries, …
- Runtime manages work decomposition dynamically
  - Expressing parallelism is low overhead
  - Nested parallelism is only used when needed/possible
  - Parallel slack provides load balancing

Elastic Forces Kernel in SPECFEM3D

SPECFEM3D_GLOBE [Komatitsch et al.]

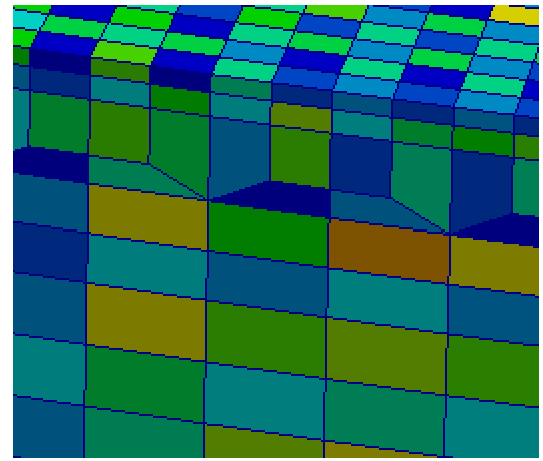
- Open source seismology package for globe-scale earthquake propagation
- Elastic wave propagation with rich physics (anisotropy, …)

Proven petascale scalability

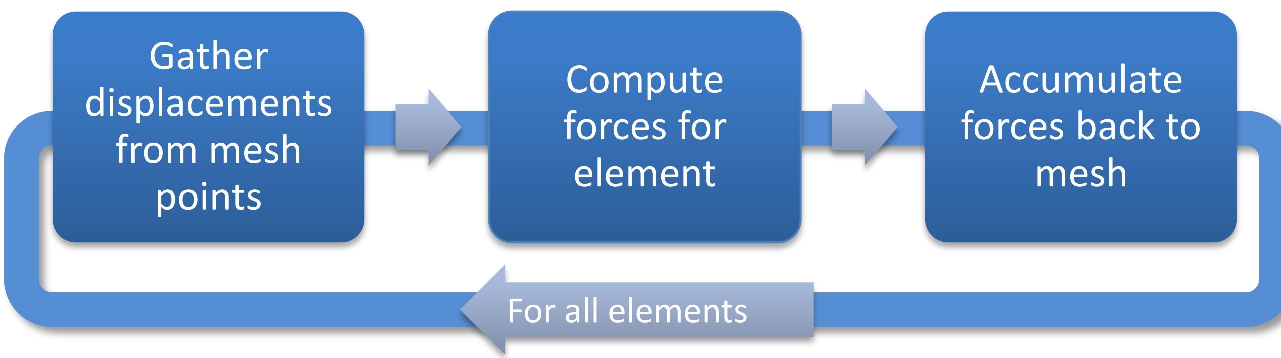
- Highly optimized application
- Very local numerical method: mass matrix is diagonal

$$\int_{\Omega} (\cdots) \cdot w \, d\Omega$$

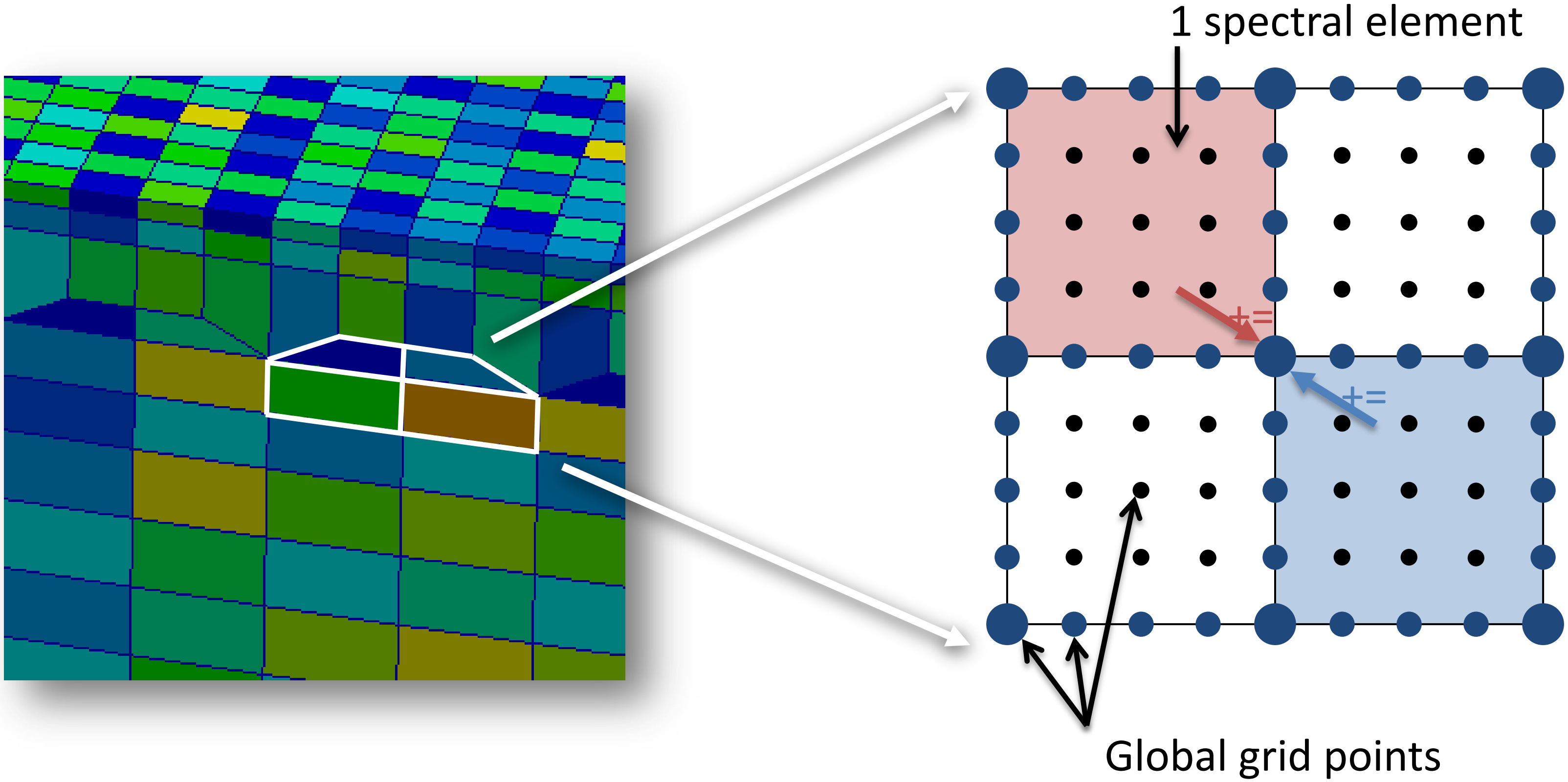
Discrete volumes:
hexahedral mesh elements



Studied here:
Stand-alone version of this kernel



Concurrency issue for shared memory parallelization

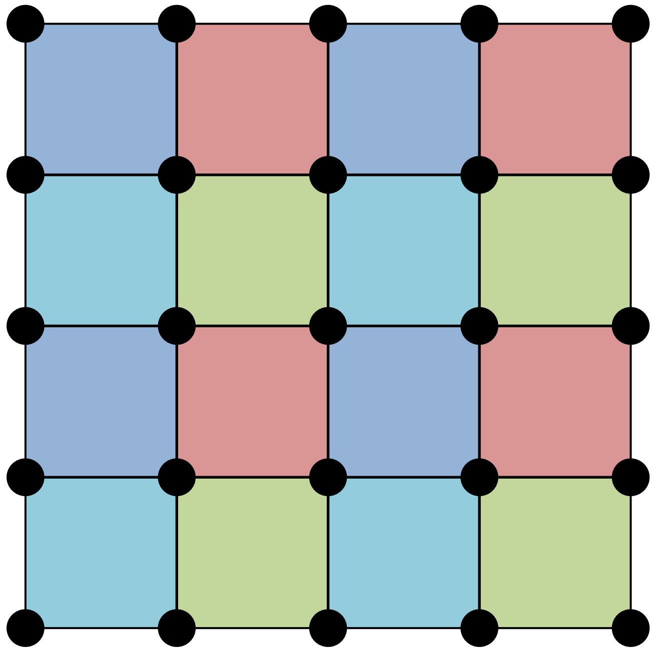


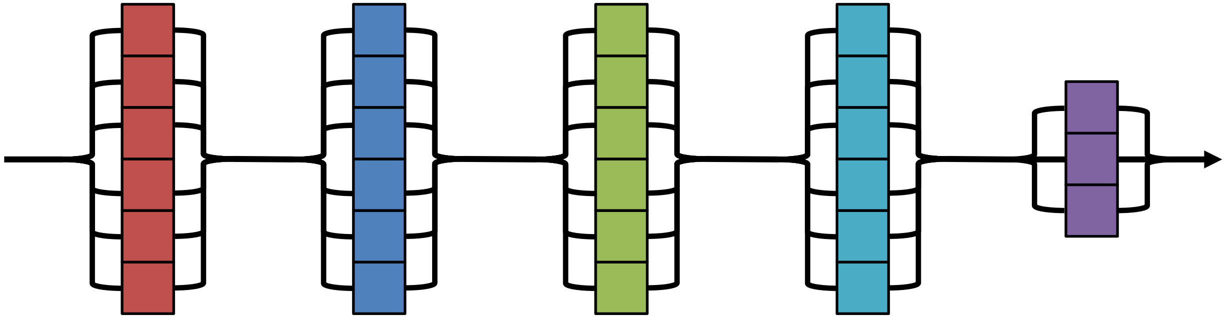
Gather displacements from mesh points → Compute forces for element → Accumulate forces back to mesh

Concurrent writes to global mesh by different spectral elements

Resolving Concurrent Writes: Mesh Coloring

- Partition the spectral elements into a number of “colors”
- No two spectral elements of a same color can contain a same global mesh point
- Elements within a same color can be processed independently

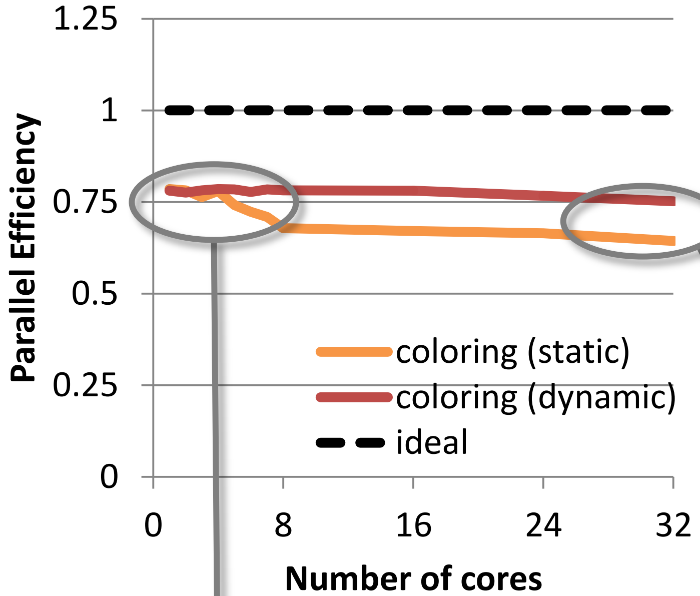


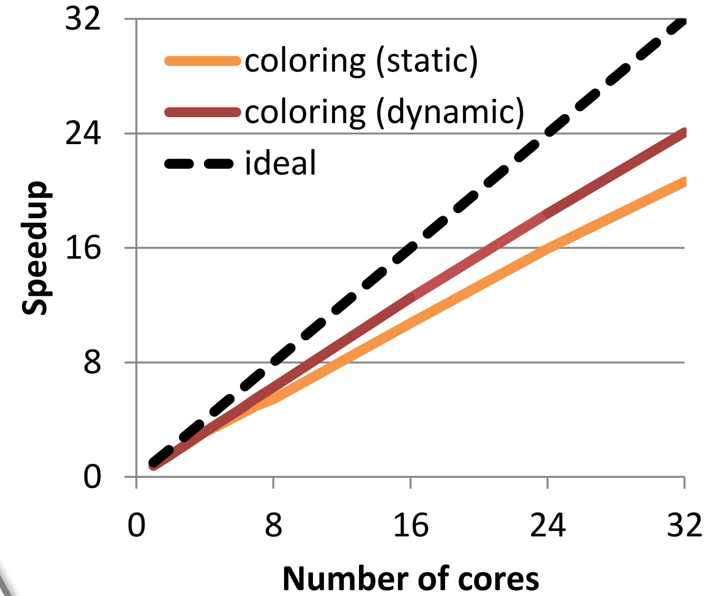


Classical algorithm, well-suited to OpenMP parallel loops

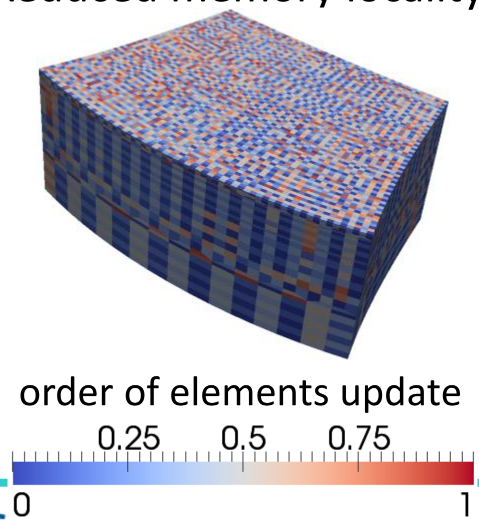
Coloring Algorithm Performance with OpenMP

Test platform: 4 sockets × 8 cores (Intel Nehalem EX)





Reduced memory locality



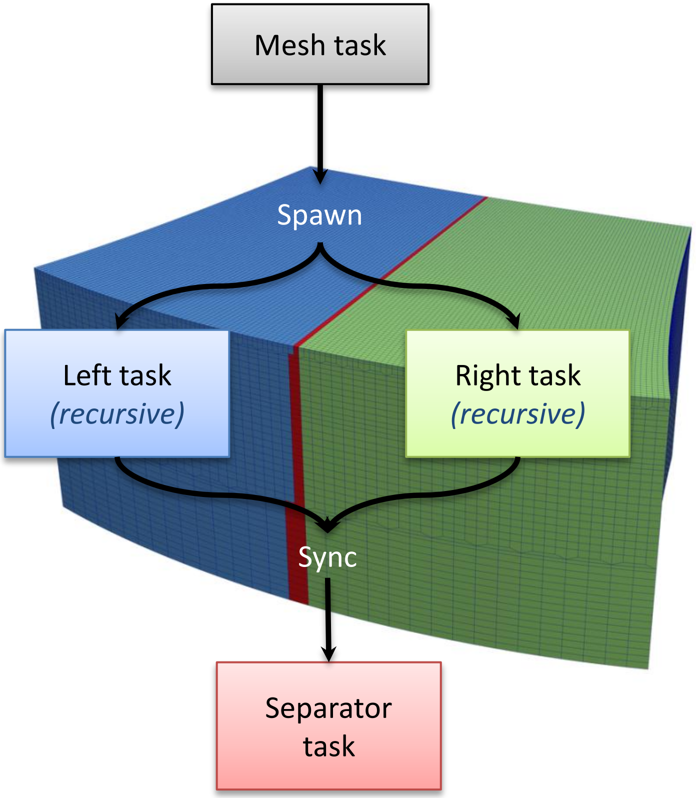
15% imbalance @ 32 cores
with OpenMP static scheduling

- OpenMP static schedule: bad load balancing
- OpenMP dynamic schedule restores scalability

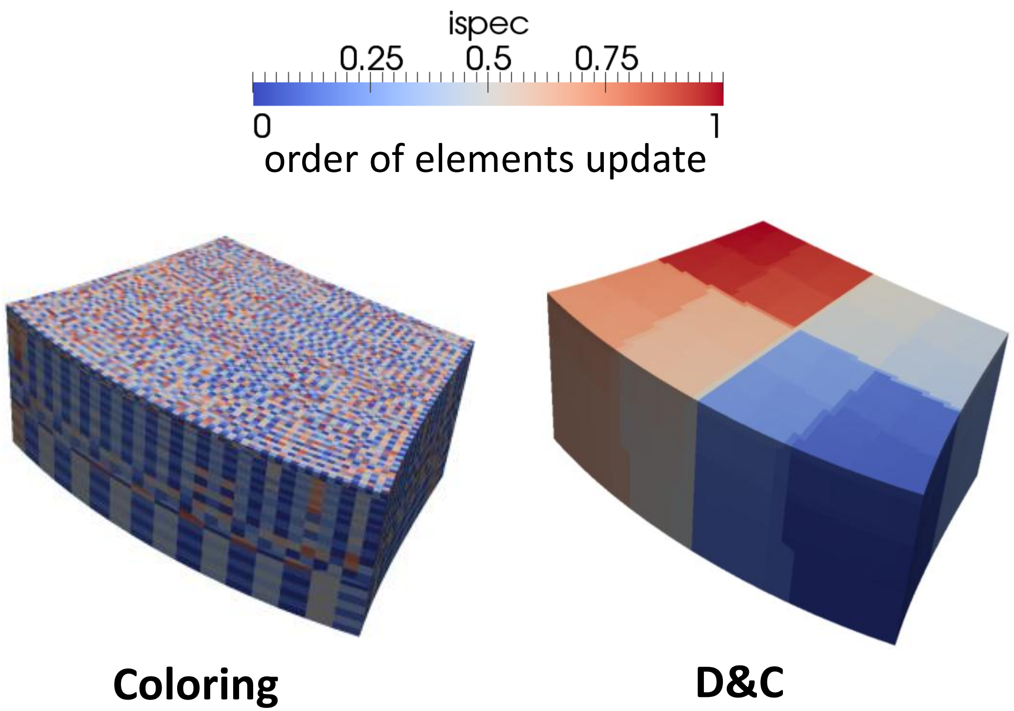
but:

- Bad data locality due to coloring algorithm
  → 25% efficiency loss

D&C Parallel Algorithm

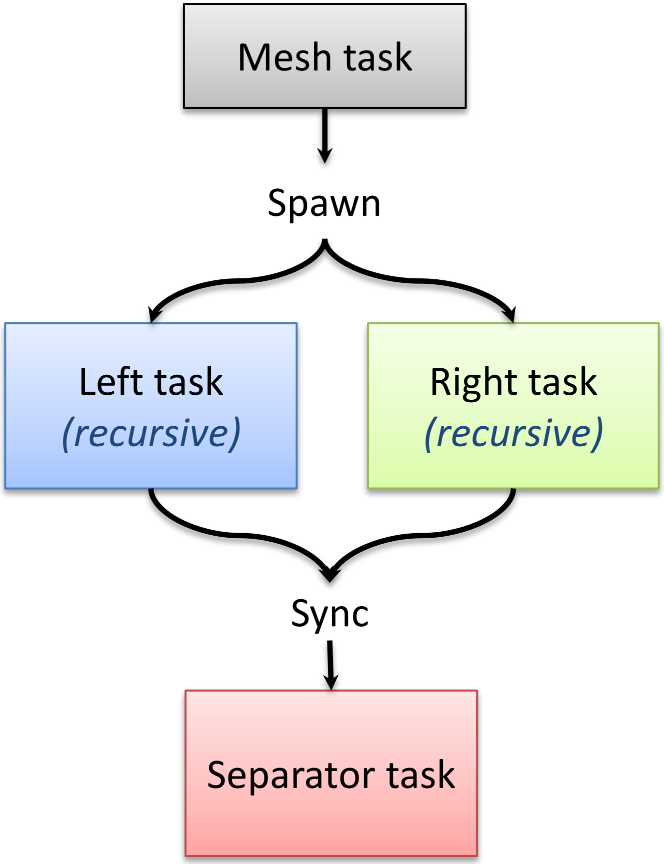


Recursively subdivide domains using a **static kd-tree**



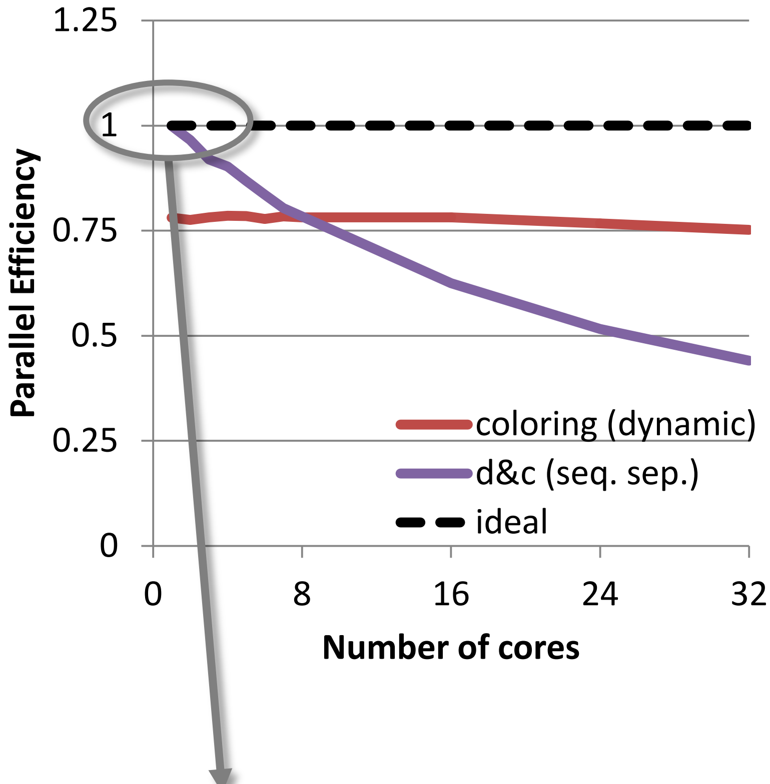
- D&C retains good dynamic load balancing with good locality
- Parallelism is introduced recursively

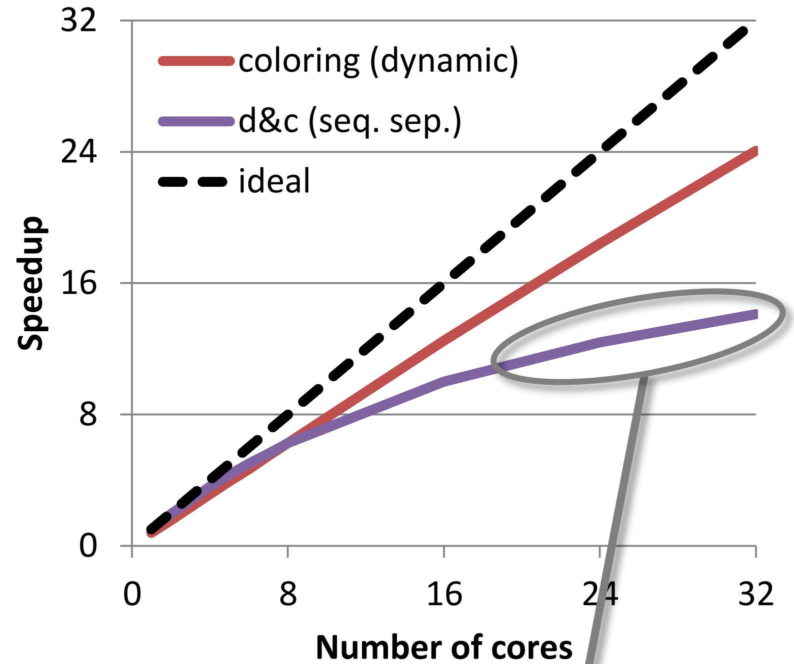
Cilk D&C Pseudocode




```
process_dc(mesh) {
    if (mesh is small enough)
    then
        process_seq(mesh)
    else
        left, right, sep = split(mesh)
        spawn process_dc(left)
        process_dc(right)
        sync
        process_seq(sep)
}
```

Cilk D&C Algorithm Performance



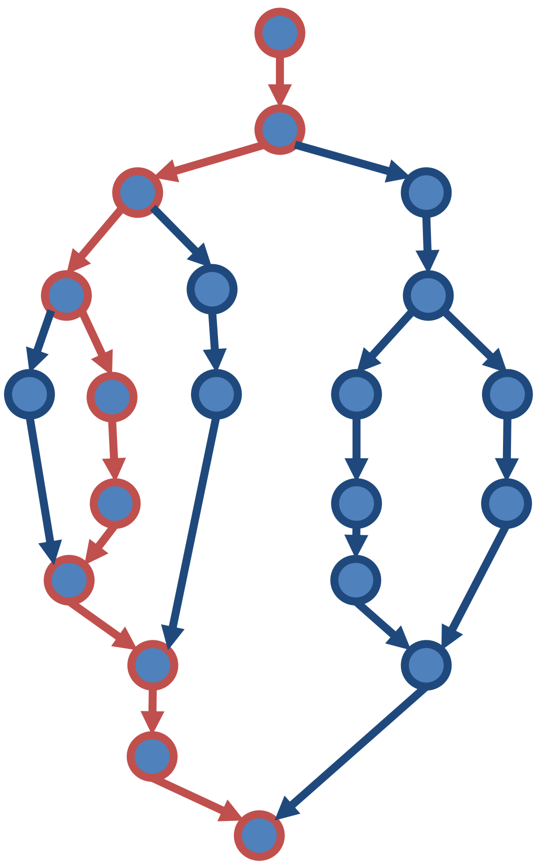


Efficiency at small core counts is good
since **locality is as good as sequential algorithm**

Sublinear scaling: why?

Work & Depth Analysis

Generalizes Amdahl's law for Cilk programs



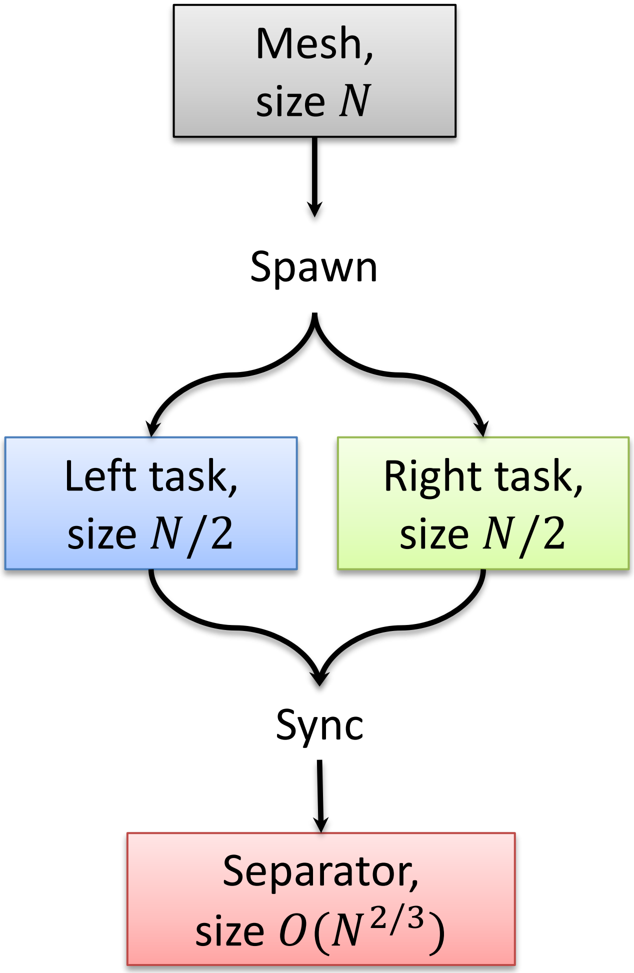
W = 21 D = 10

- W = work
  (#operations of all tasks)
- D = depth
  (#operations of tasks on the critical path)
- Cilk work-stealing scheduler guarantees

$$T_p = \frac{W}{p} + \mathcal{O}(D)$$

We need a small depth
to achieve linear speedup

What is the depth in our case?



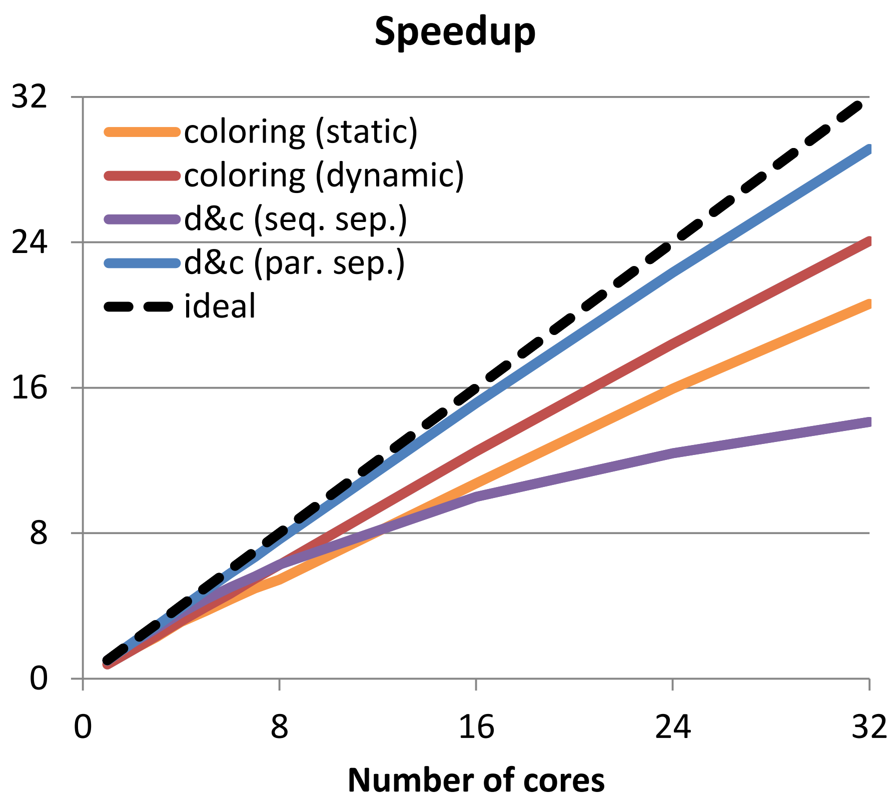
Processing of separator planes

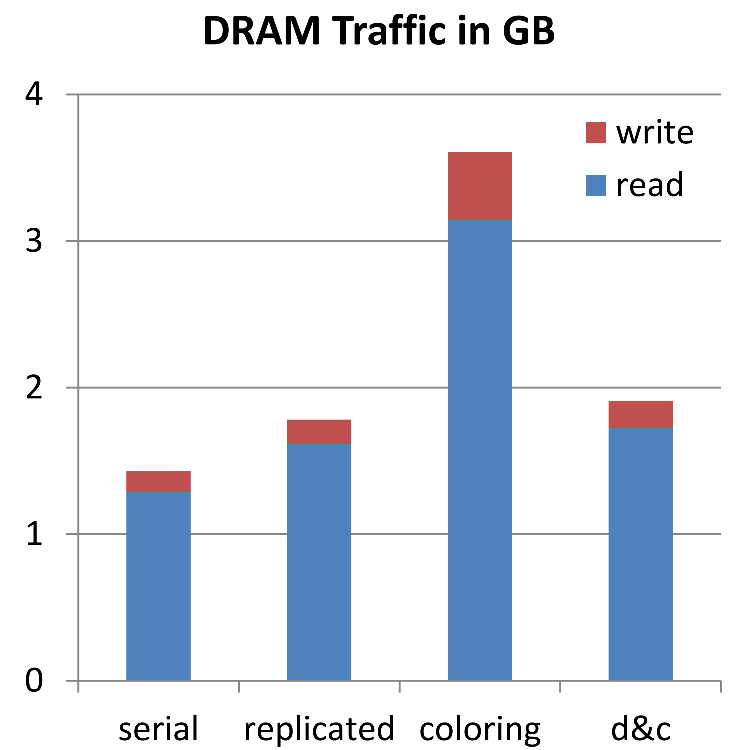
Sequential	In parallel
$D_N = D_{\frac{N}{2}} + \mathcal{O}(N^{2/3})$	$D_N = D_{\frac{N}{2}} + D_{N^{2/3}}$
$D_N = \mathcal{O}(N^{2/3})$	$D_N = \mathcal{O}(\log N)$

Parallel D&C processing of separator planes significantly reduces depth

→ **Parallelize the computation of separators using the same recursive technique**

Overall Performance





Cilk D&C parallel kernel is **1.2x faster** than the dynamic OpenMP coloring kernel, with a **1.9x DRAM traffic reduction**

Conclusion

Algorithms will matter more and more towards Exascale

- Expose lots of parallelism
- Be locality-aware

Tasks enable efficient shared-memory algorithms

- Composable parallelism: expose parallelism at all levels
- Implicit parallelism: runtime does the scheduling/load balancing
- Can leverage inherent locality of D&C algorithm

What about full applications?

- Efficient hybrid MPI+OpenMP programming is hard
- Recent efforts to combine tasks and communications in distributed memory (e.g. Asynchronous PGAS, MPI+StarSS, …)
- Will likely need to be a joint effort with numerical methods, e.g. communication-avoiding algorithms

Exascale Computing Research Contacts

- Address
  UVSQ, 45 Av. des Etats-Unis, Buffon building, 5th floor
  78 000 Versailles, France
- Web site: www.exascale-computing.eu
- Team
  William Jalby, CT , william.jalby@uvsq.fr
  Marie-Christine Sawley, Co-design, marie-christine.sawley@intel.com
  Bettina Krammer, Tools, bettina.krammer@uvsq.fr
- Collaboration partners

References

- SPECFEM3D
  Komatitsch and Tromp, Geophys. J. Int., 149 (2002) 390–412
  Komatitsch et al., J. Comp. Phys., 229 (2010) 7692–7714
- Cache-Oblivious Algorithms
  Frigo et al., FOCS 1999
- Cilk & Intel® Cilk™ Plus
  Frigo et al., PLDI 1998
  http://cilkplus.org
- MPI/SMPSs
  Marjanovic et al., ICS 2010
- Asynchronous PGAS languages:
  X10, Chapel, UPC Tasks