UCRL-PRES-222585

Parallelizing the Communication-Intensive Point-Clustering Algorithm

Brian T. N. Gunney and David A. Hysom

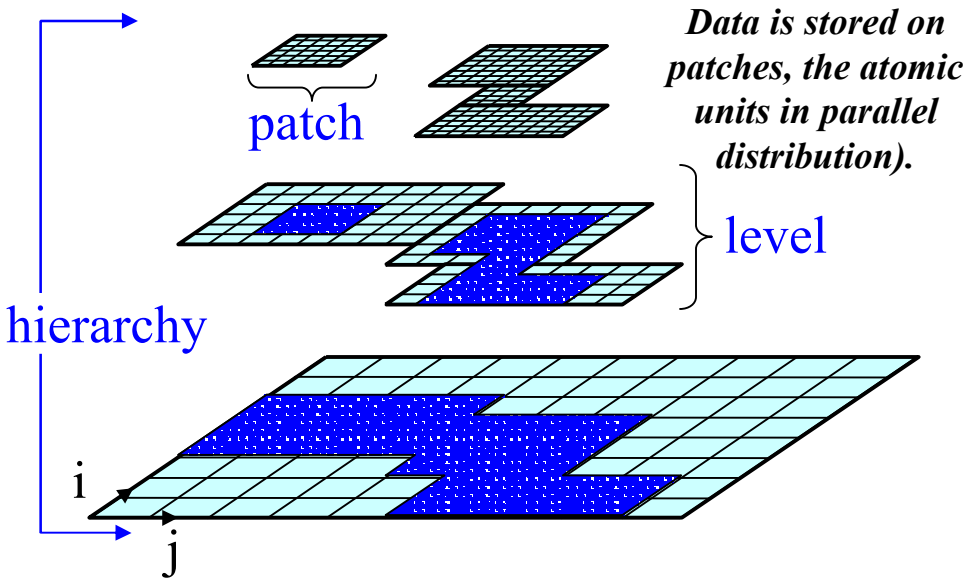
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory

ScicomP 12
20 July 2006

This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

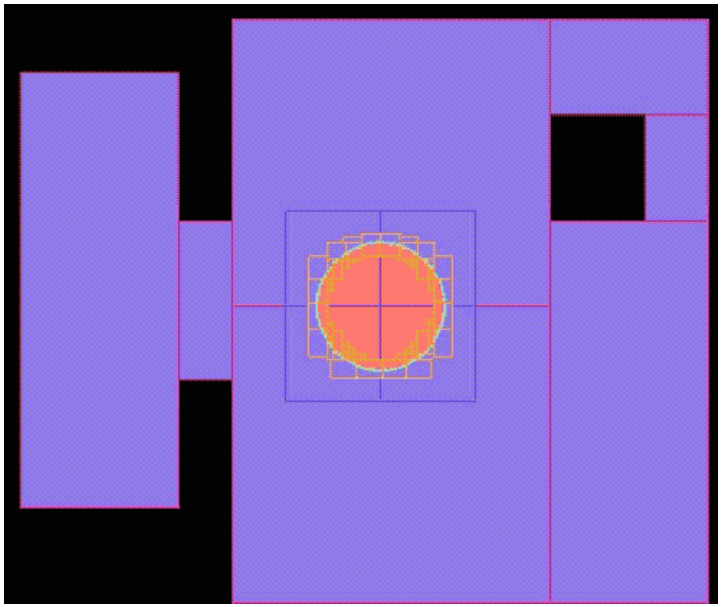
Global nature of structured AMR mesh adaption presents scalability challenges

Hierarchical mesh



Data is stored on patches, the atomic units in parallel distribution).

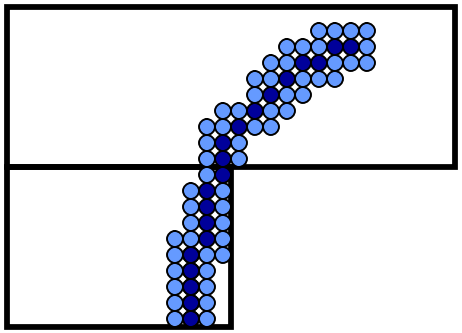
Simulation



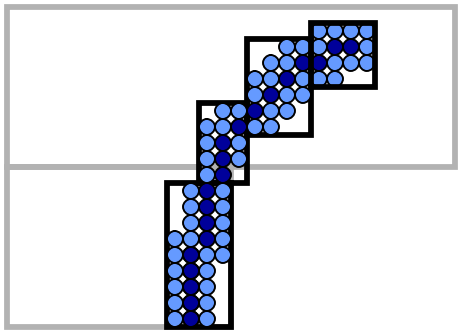
Scalability challenges

1. Dynamic meshes are adapted throughout simulation.
2. Computation generally uses nearest neighbor data. Grid adaption involves the entire grid.
3. LLNL's BG/L machine has 125K processors

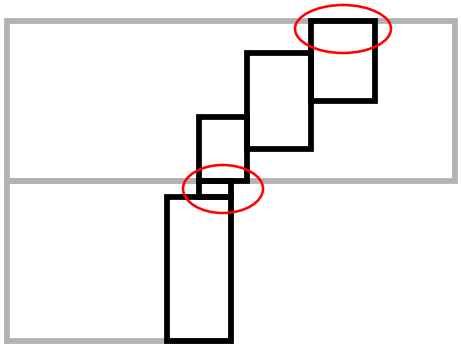
Mesh adaption consists of operations on the whole mesh



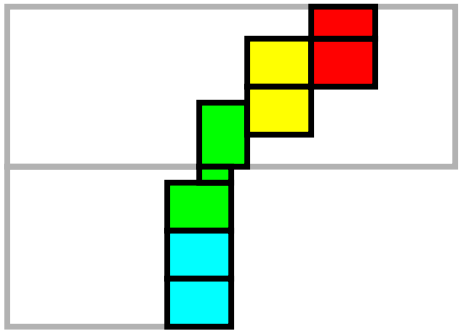
1. Tag cells



2. Cluster



3. Apply restrictions

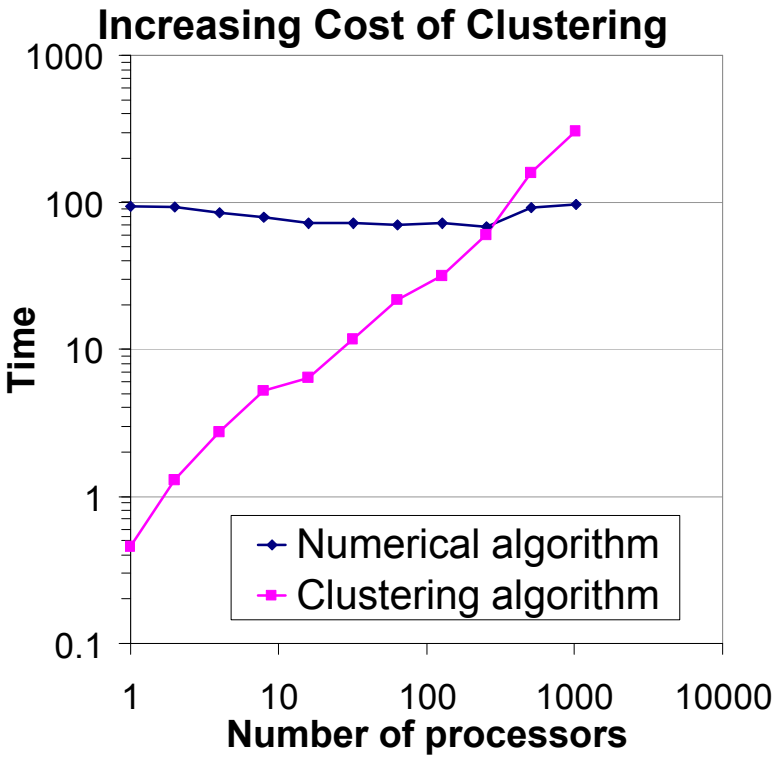


4. Load balance

5. Create new level

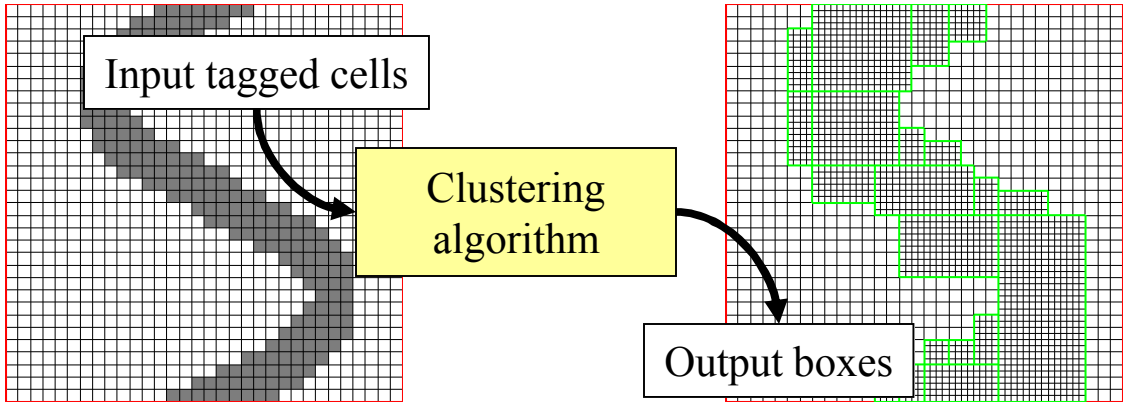
6. Initialize data on new level.
   a. If refining, interpolate data from coarser level.
   b. If remeshing, also copy data from old level. Replace old level with new.

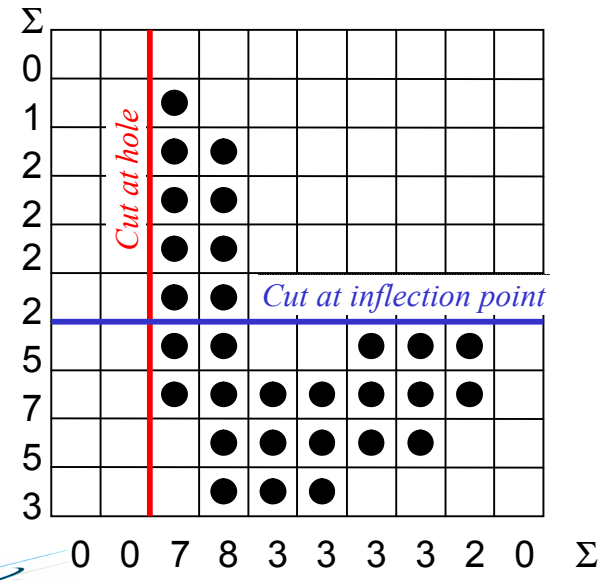
Clustering should be a small overhead cost but does not scale well

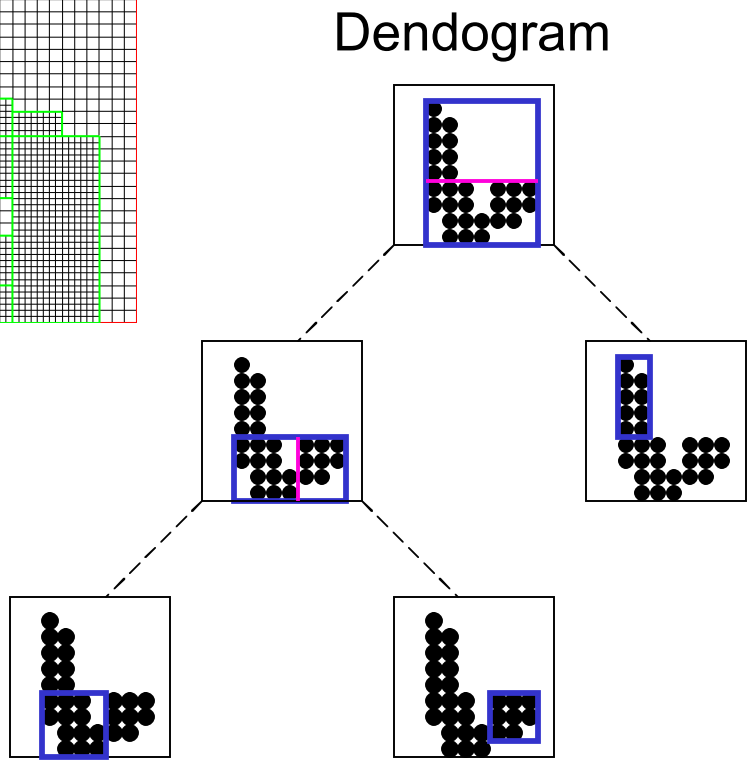


- Overhead cost are often small but may not scale well.
  - Requires whole-mesh operations.
  - Requires distant data.
- Unscalable overhead algorithms prevent application scaling.

Clustering is a recursive bisection algorithm requiring distant data







Berger and Rigoutsos, IEEE Transactions on Systems, Man and Cybernetics, Vol. 21, No. 5, 1991.

Recent parallel implementations of clustering algorithm in SAMRAI use SPMD model

Global communication:

```
fcn(box) {
  h = signature(box)
  if (h is acceptable)
    accept(box)
  else
    split(h,left,right)
    fcn(left)
    fcn(right)
}
```

1. `signature()` includes global sum-reduce.
2. `accept()` and `split()` are local operations.
3. All processors run algorithm.
4. All processors have outputs.

Grouped communication:

```
fcn(box) {
  h = signature(box)
  if (h is acceptable)
    accept(box)
  else
    split(h,left,right)
    if (overlap(left )) fcn(left)
    if (overlap(right)) fcn(right)
}
```

Functions in red require communication (synchronization)

1. `signature()` includes sum-reduce to processor 0.
2. `accept()` and `split()` include broadcast from processor 0.
3. Only processor 0 runs full algorithm.
4. *Only processor 0 has outputs, which must be globalized using a broadcast.*

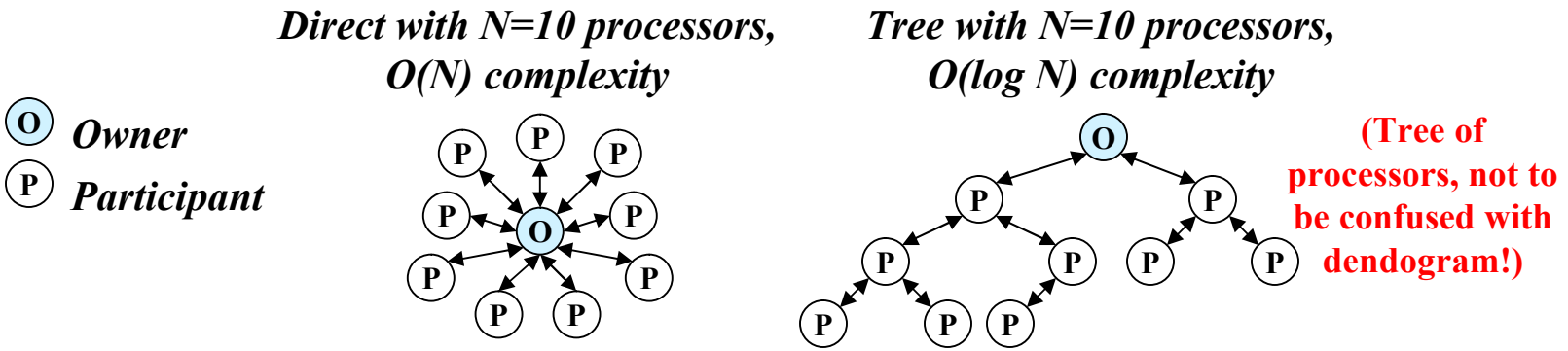
Wissink, et. al. "Enhancing Scalability of Parallel Structured AMR Calculations", 17th ACM International Conference on Supercomputing, June 2003.

Most function calls require very few processors

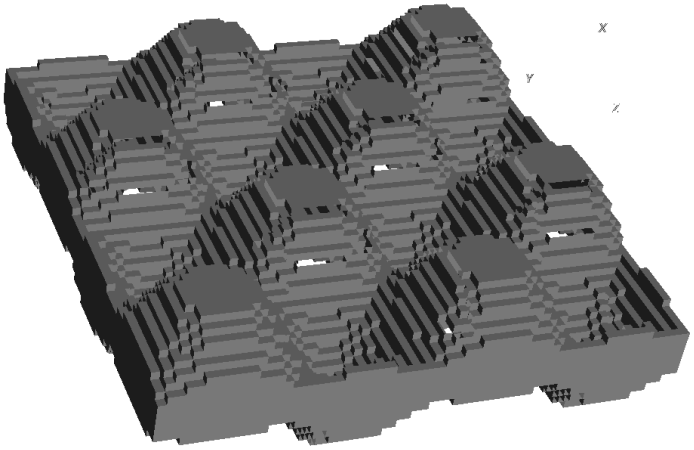
Recursion level	Max # of participating processors	Percent of function calls at or below level
0	128	100
1	100	98
2	80	95.6
3	80	91.5
4	32	84.7
5	18	74.5
6	8	61.9
7	8	46.3
8	8	30.6
9	8	17
10	4	7.1
11	2	1.4

Hand-coded collective communications is faster than communicators and non-blocking

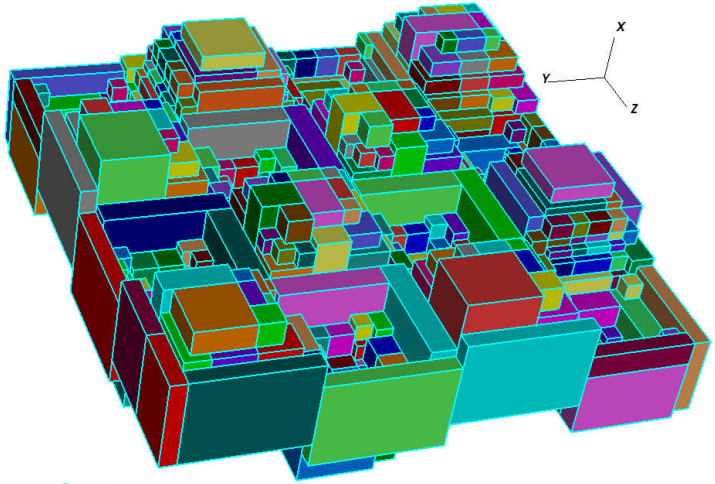
- Avoid expensive formation of MPI communicators (experience from Linux clusters and IBM SP)
  - Communication is required to create new MPI communicator.
- Hand-coded communication along edges of a tree instead of spokes of a wheel.
- Supports non-blocking collective communications by using non-blocking point-to-point calls.



Sinusoidal-front test problem simulates complex surface for adaption



Tagged cells near sinusoidal front

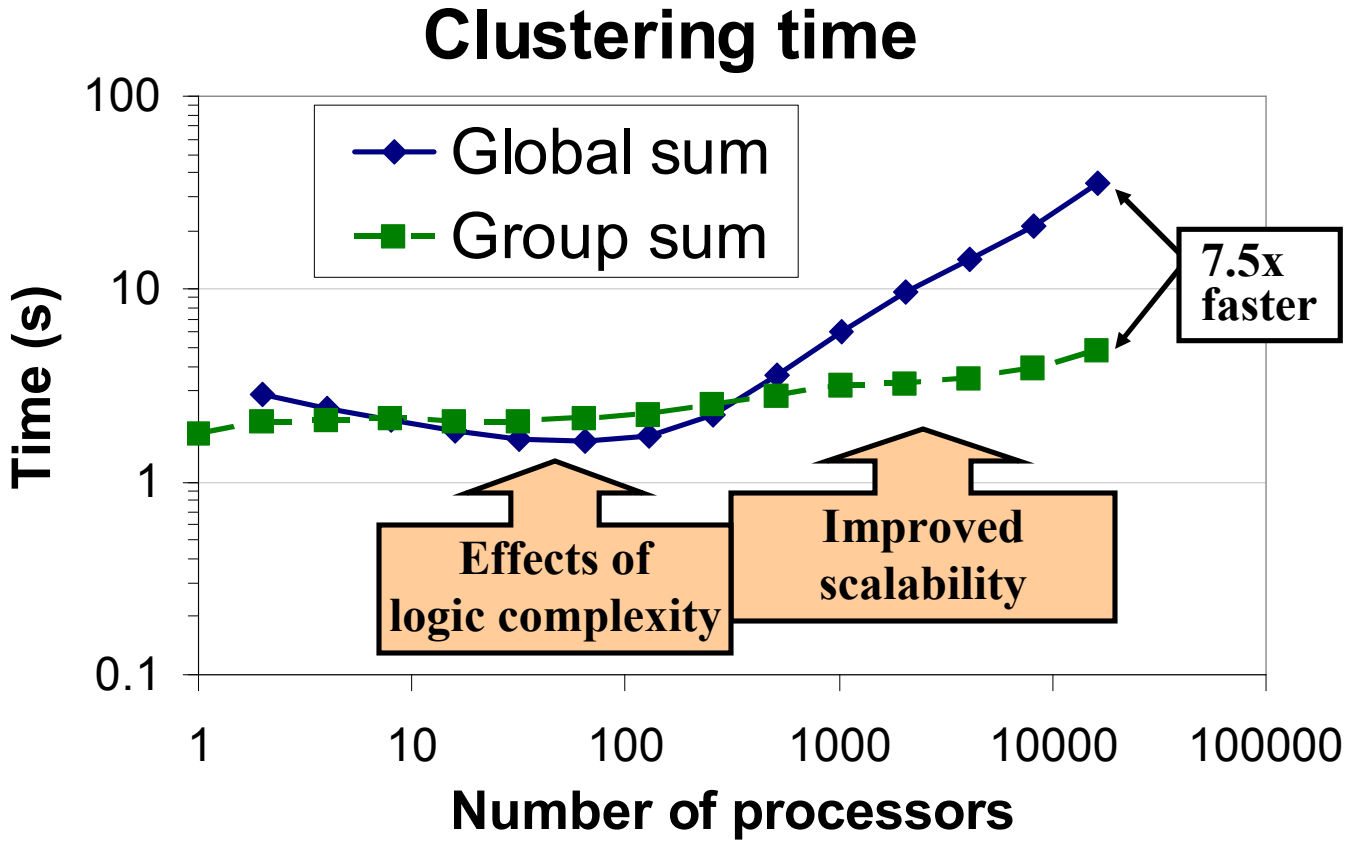


Boxes generated

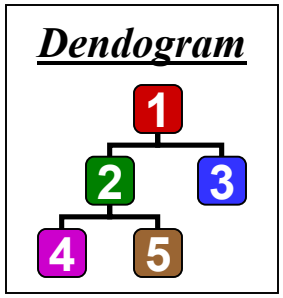
- Timings for initial mesh generation and 5 complete remeshes
- Small problem:
  - 4 levels of refinement
  - 782 candiate boxes evaluated
  - 327 boxes accepted
- Large problem
  - 6 levels of refinement
  - 29715 candiate boxes evaluated
  - 12402 boxes accepted
  - 38 times size of small problem

New group sum is significantly faster than global sum

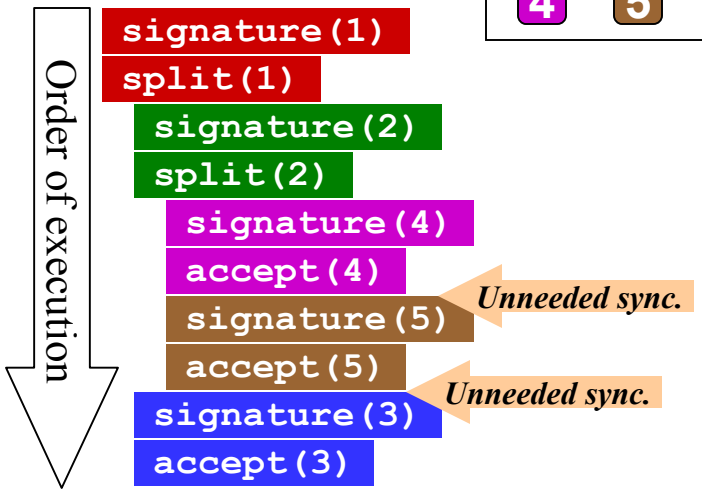
Strong scaling results



Clustering algorithm contains independent paths appropriate for task-parallel model

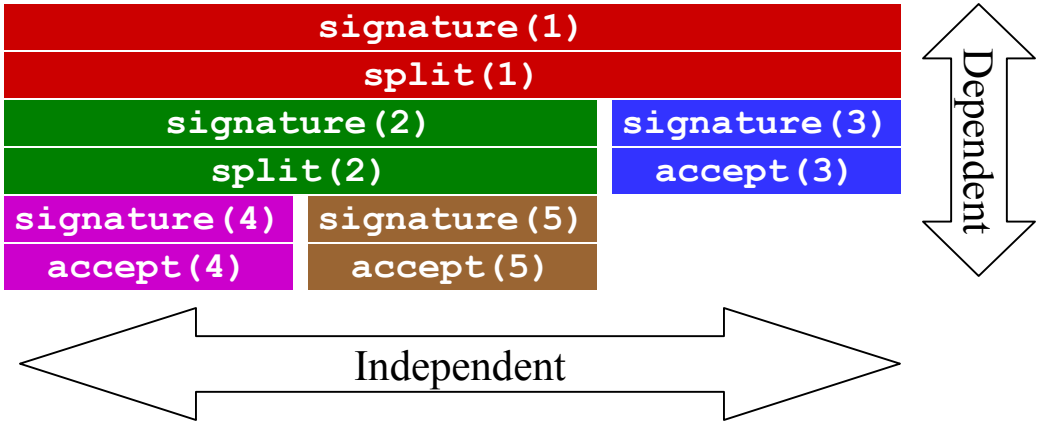


Procedural



1. Impose specific ordering.
2. Create unneeded synchronizations.

Task-oriented



1. Each dendogram node is a task.
2. Do independent tasks concurrently.
3. Overlap communications and computations in multiple tasks.

Task-parallel implementation follows natural tasks in clustering algorithm

Sequential, old

```
fcn(box) {
    …
    split(h,left,right)
    if (overlap(left))
      fcn(left)
    if (overlap(right))
      fcn(right)
}
```

Sequentialized!

Replace with

Task-oriented, new

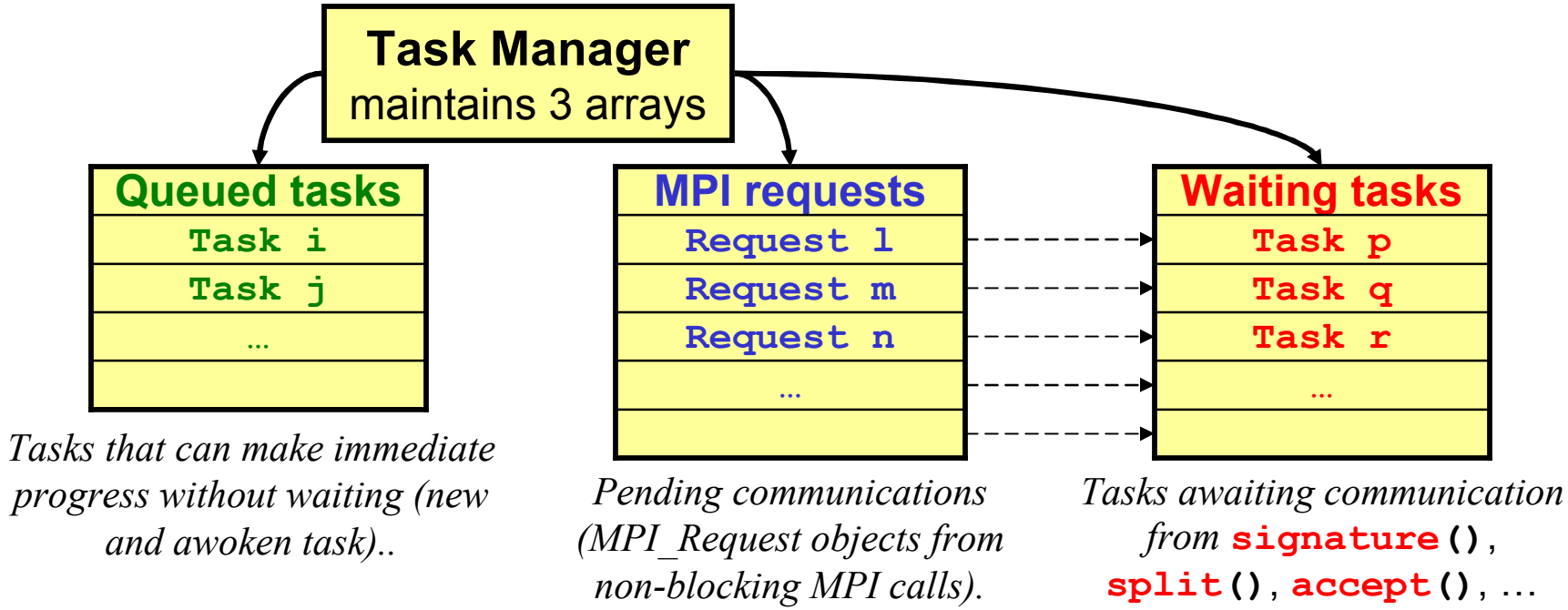
```
task(node) {
    …
    split(h,left,right)
    add_to_task_mgr(left,right)
    sleep_until_children_finish()
    add_to_task_mgr(parent_node)
}
```

Interruptible communication wait

Interruptible non-communication wait

- Replace sequential operations on left and right children with insertion into task manager (next slide).
- Each instance of `task()` is associated with one node of the dendogram. It is *not* recursive.
- Communication and sleeping steps are "interruptible" so waiting tasks can be set aside to work on tasks that can make immediate progress.
- Tasks are initiated by insertion into task manager (not directly called).
- Task-parallel algorithm driven by task manager (next slide).

Task manager selects active tasks to minimize processor wait times



Task Manager Algorithm (user-space thread controller):

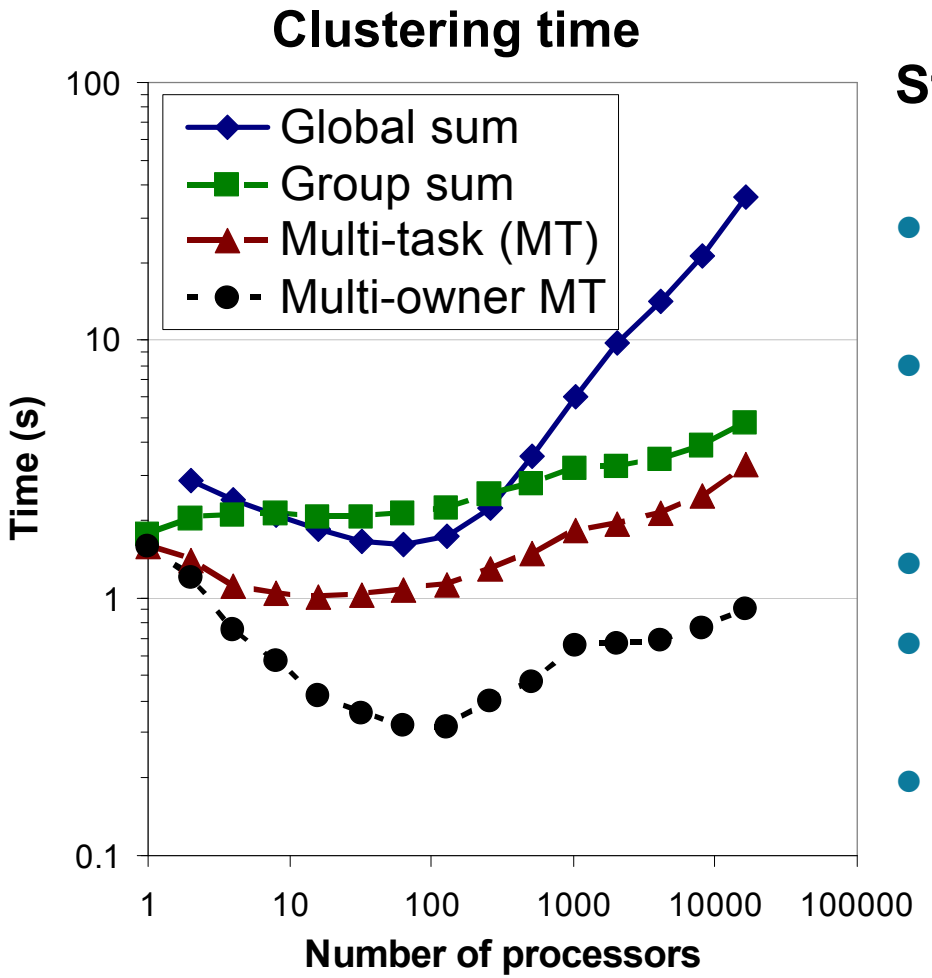
1. Start/continue all tasks in **task queue** (and empty the queue)
2. Wait for some **pending communication requests** to complete
3. Continue **waiting tasks** for completed communications
4. Repeat until no more task or pending request exists.

Use multiple owners to reduce workload and communication traffic for processor 0

Multi-owner option (select owner from participating processor group):

- **Give coordination responsibility to processor with the most overlap with candidate box.**
  - **Incurs cost of computing overlap.**
  - **Other criteria for ownership makes little difference.**
- **Reduces traffic congestion around single-owner.**
- **Improves load balance.**
- ***Requires all-gather instead of broadcast for globalizing output (drawback addressed by related research).***

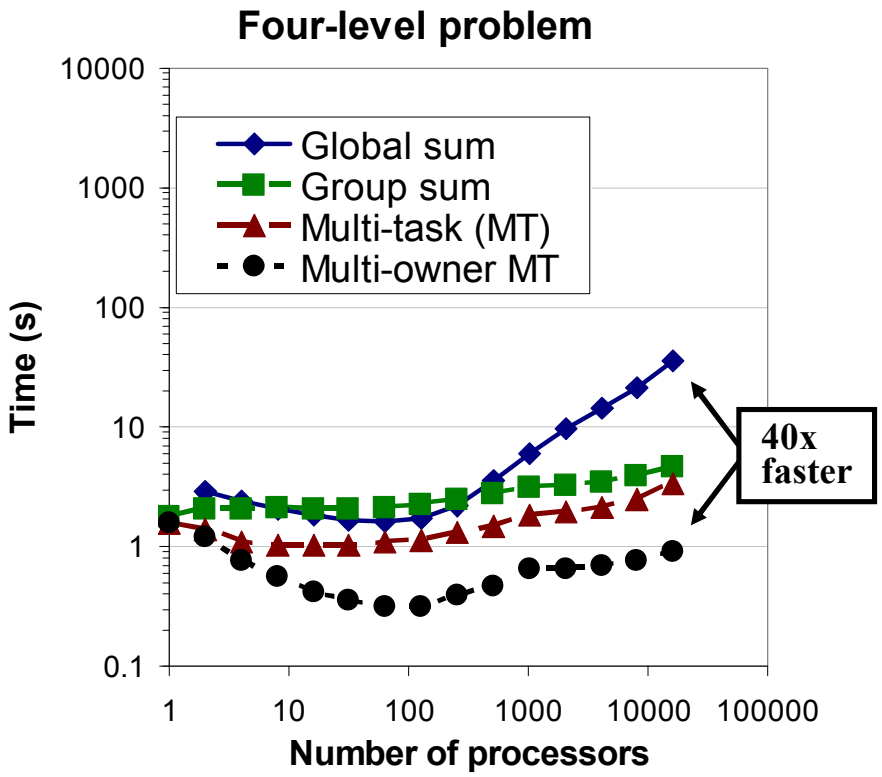
Multi-task approach significantly improves speed and scalability

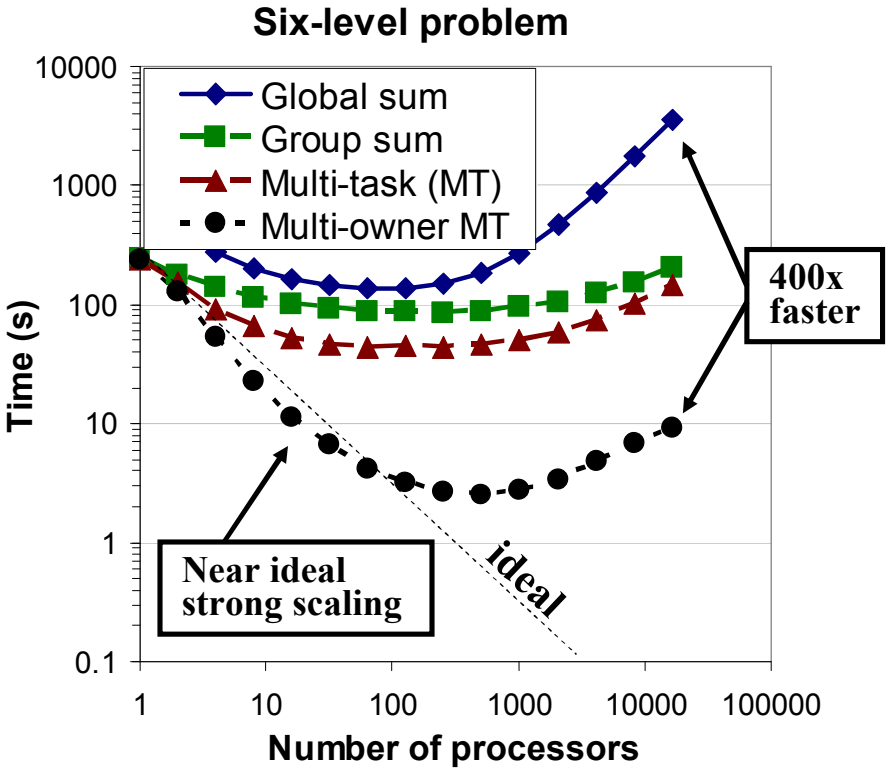


Strong scaling results for small problem:

- **Improved performance for all processor counts.**
- **Improved scalability (less rise in the curves) for all processor counts.**
- **40 times faster at 16K proc.**
- **Multi-owner amplifies the positive effects of multi-task.**
- **Largely similar behavior observed on Linux clusters up to 2K procs.**

Performance gain is more pronounced for larger problem

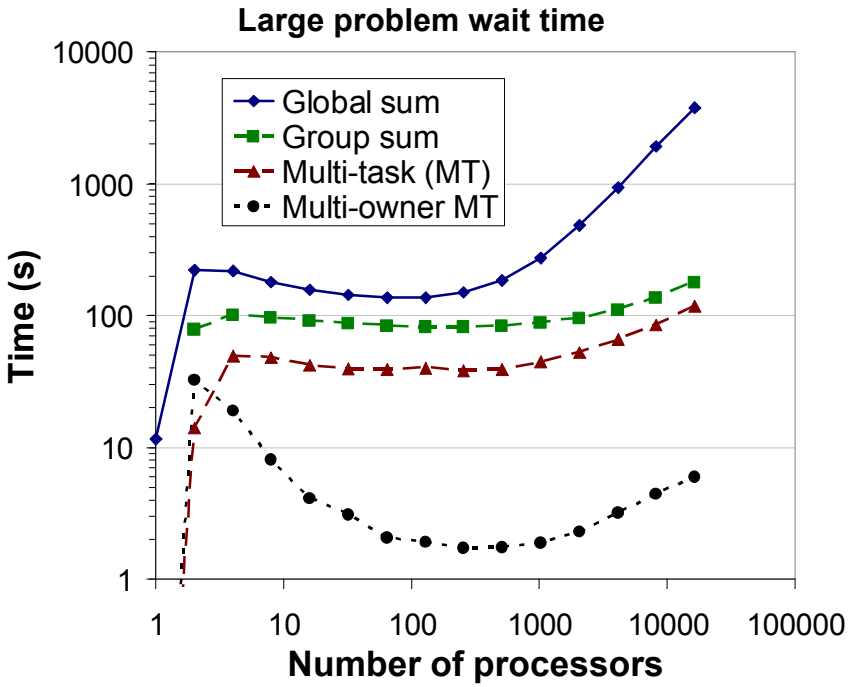


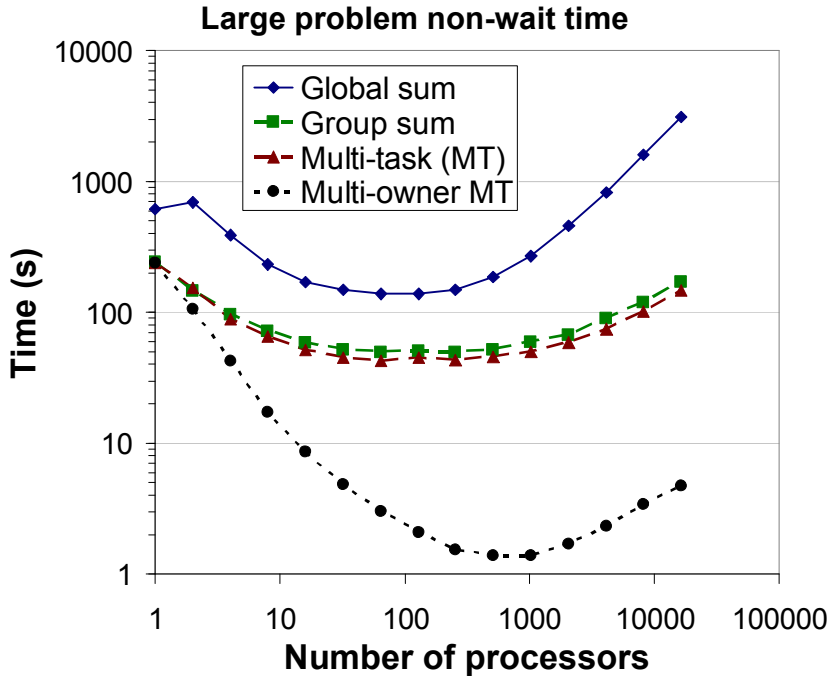


Multi-owner multitask algorithm on the 38x larger problem:

- 100 times as long at 1 proc.
- only 10 times as long at 16K proc.

Wait and non-wait times for all algorithms





Summary and Conclusion

- The clustering is a communication-intensive overhead mesh operation in SAMR.
- SPMD (global sum) parallelization scales poorly.
- We used groups to reduce amount of communicated data.
- Independent tasks within algorithm maps to task-parallel approach.
  - Task switching minimizes communication wait.
  - Each task is still SPMD.
- Distribute group ownership to reduce load imbalance and communication bottlenecks.
- New algorithm outperforms old for all processor counts in study.
  - Scales better than original.
  - Is 400 times faster at 16K processors.
- Gunney, Wissink and Hysom, “Parallel Clustering Algorithms for Structured AMR”, Journal of Parallel and Distributed Computing, to appear.