Optimizing the Barnes-Hut Algorithm for Multicore Clusters

Junchao Zhang, Babak Behzad, Marc Snir

University of Illinois at Urbana-Champaign, {jczhang, bbehza2, snir}@illinois.edu

SC'13, November 17-22, 2013, Denver, Colorado, USA

Motivation

- Important trends of supercomputer evolution:
  - Number of cores per node keeps increasing;
  - Amount of memory per core is decreasing;
  - One-sided communication (rDMA) is increasingly well supported on the interconnection networks.
- It is important to explore new programming models to cope with or leverage these trends.
- Through the Barnes-Hut (BH) algorithm, we study a promising programing paradigm, i.e., partitioned global address space (PGAS) + X, which
  - Uses a global address space for internode communication;
  - Hides latency through intranode multitasking;
  - Achieves data sharing and message reduction through multithreading.
- We show many advantages gained from this design.

What is Barnes-Hut?

- An $O(n\log n)$ algorithm for the n-body problem, which needs to compute forces on every body (particle).
- Use an octree as the main data structure.
- Has ***massive*** parallelisms:
  - For all bodies, can compute their forces in parallel.
  - For one body, can interact with different tree nodes (aka *cells*) in parallel, provided the results are summed together finally.
- ***Irregular***, whether to traverse children of a cell (i.e., *open* a cell) is computation dependent.

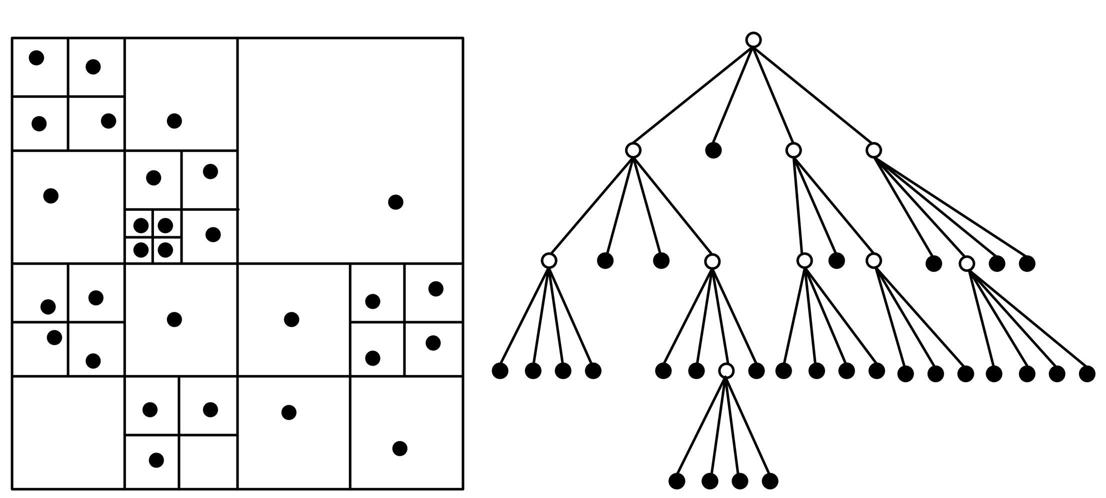

Fig. 1 A 2D body distribution and the corresponding quadtree. To compute forces on a body, we need to traverse the tree from the root. If a cell to be accessed is on remote node, we'd better cache it locally after fetching it, since we may reuse it for subsequent bodies.

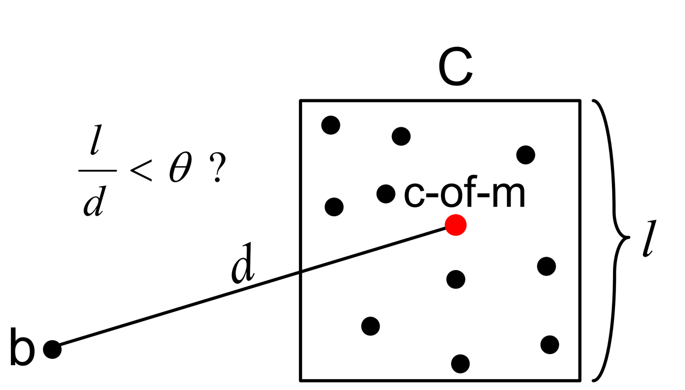

Fig. 2 Test if body b is far-away enough from cell C. If not, we need to "open" C. c-of-m = center-of-mass. θ is a user-defined constant. This test result is known at runtime, which makes it hard to reason communication statically.

Hybrid BH Design

- Implemented in PPL, a C++ template library having a PGAS memory model.
- Octree is distributed among compute nodes and cells are linked by *global* pointers provided by PPL.
- Each cell has a flag `localized` to indicate whether its children have been cached locally.

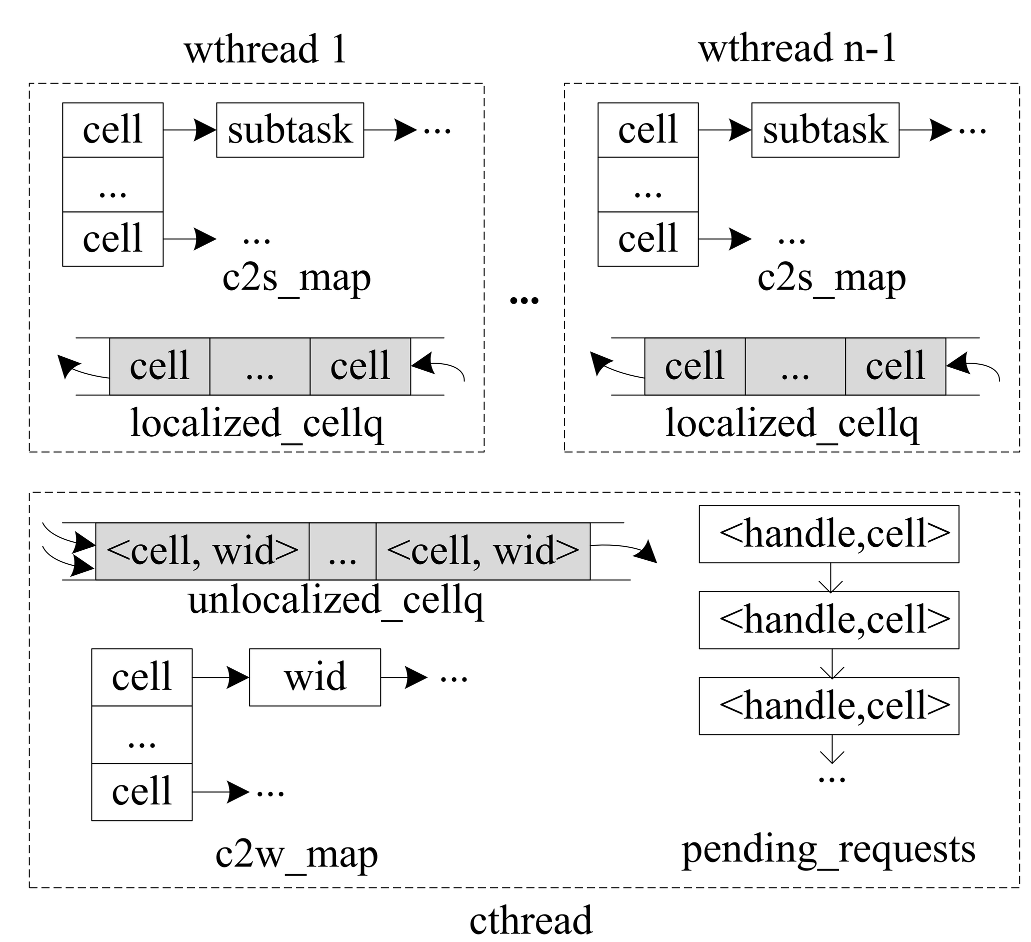

Fig. 3 Data structures used by threads, shaded are shared

- ***Multithreading***: One communication thread (cthread) + multiple worker threads (wthreads) per process.
- ***Multitasking*** for fine-grain parallelism.
  - A *task* consists of tree traversal and force computation for one or more (if use tiling[4]) bodies;
  - A *subtask* encapsulates interactions between a body and a subtree rooted at an *unlocalized* cell.
- ***Wthreads*** are responsible for computation.
  - Grab bodies from a body array assigned to the owner process and generate tasks.
  - Execute tasks (i.e., traverse the octree), whenever need to open a unlocalized cell, generate a subtask and continue to execute ready-to-run subtasks if any, or just generate a new task.
  - Manage subtasks locally through a cell-to-subtask map `c2s_map`.
  - Notify cthread which cells to localize by pushing <cell, worker id> to a queue `unlocalized_cellq`
- ***Cthread*** is responsible for communication.
  - Pops cells from `unlocalized_cellq` and fetches their children by *dereferencing child pointers* (in a non-blocking fashion).
  - Caches child cells through *pointer-swizzling.*
  - Tests pending requests, if completed, sets corresponding cells as localized and pushes them back to `localized_cellq` on relevant wthreads
- ***Locality***: Between nodes, sort bodies in a space filling curve to reduce communications; Within a node, use tiling [4] to improve L1 cache reuse.

Experimental Setup

- Tested on a Linux cluster with each node having two hex-core Xeon® X5650 CPU@2.66GHz, with up to 64 nodes.
- Generated input bodies using the Plummer model.
- Compared the current code (named PPL BH) with UPC BH [1] (a BH code without multithreading), SPLASH-2 BH [5] (on single node) and other BH codes.

Results

- ***Less memory*** by sharing the cached octree through multithreading in stead of having each core build its own cached octree as in UPC BH.
  - With n=1M, θ=0.5 on 64 nodes, saved 59% of memory.
- ***Less communication***, since data sharing also helps to filter duplicated off-node data requests.
  - With the same setting, saved 57% internode message.
- ***Better load balancing*** by applying OpenMP *guided*-like scheduling, to adapt to runtime fluctuations.
  - Execution time variation of the 12 threads is < 2.5%, which was up to 71% in UPC BH.
- ***Better performance and scalability***

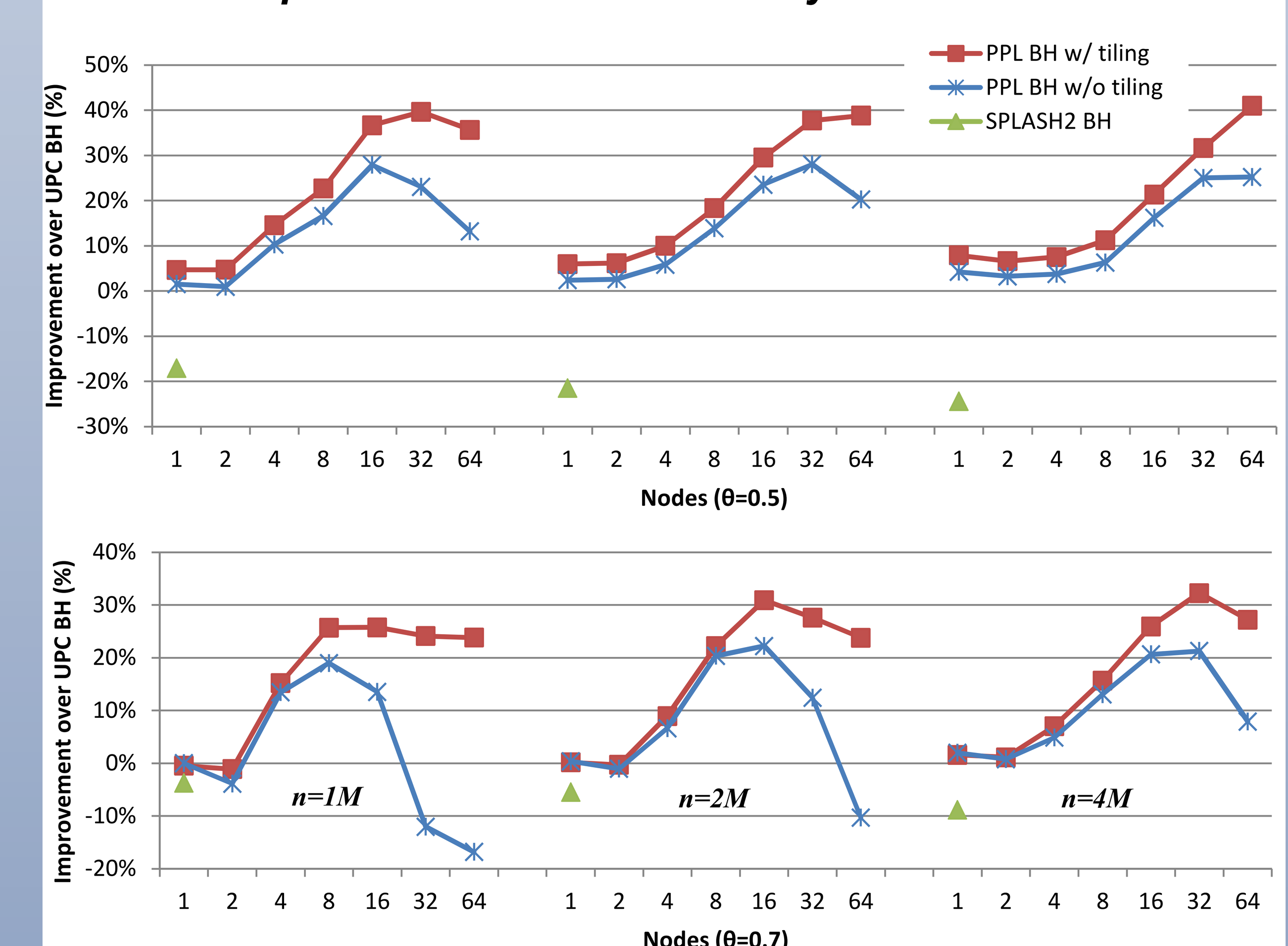

Fig. 4 Performance improvement over UPC BH. Note that the single node performance is even better than SPLASH-2 BH. With tiling, there're 128 bodies per task. Without tiling, 1 body per task.

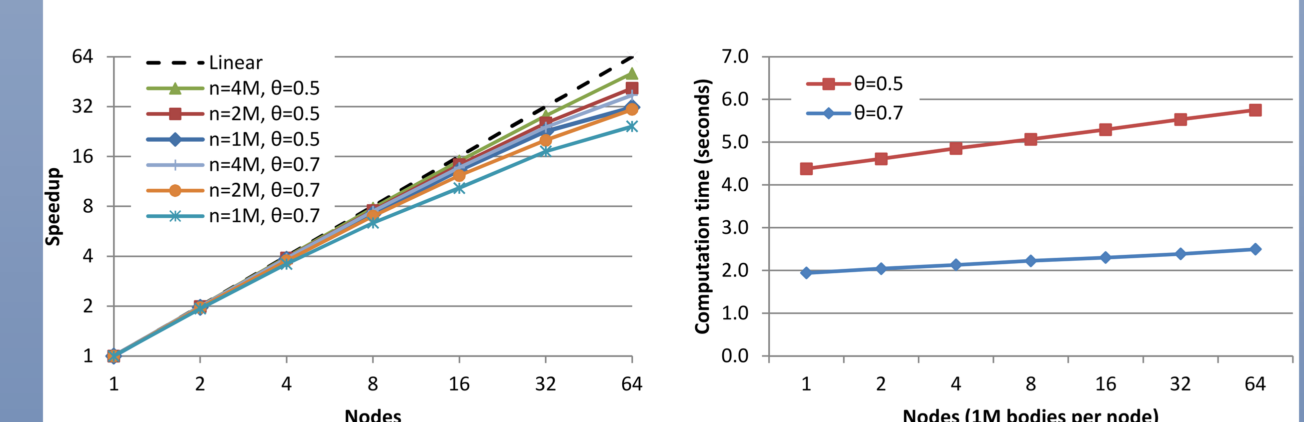

Fig. 5 Strong scaling (left) and weak scaling (right) curves.

(continue)

- Compared with two other state-of-the-art BH codes, i.e., Charm++ BH [3] and PEPC [2]
  - With n=1M, θ=0.5, on 1~64 nodes, PPL BH is 1.4x ~ 4.5x faster, while its number of lines of code is 2x ~ 8x fewer.

Conclusions and Future Work

- We optimized the Barnes-Hut algorithm in a PGAS + X programming style, which brings many benefits as shown and thus is well suited to multicore clusters.
- We integrated multithreading and one-sided communication, with multitasking as the primary mechanism for latency hiding.
- In future, we like to abstract common runtime services from PPL BH and try to benefit other irregular applications.

References

1. J. Zhang, B. Behzad, and M. Snir. Optimizing the Barnes-Hut algorithm in UPC. In ACM/IEEE SC'11.
2. M. Winkel, R. Speck, H. Hubner, et al. A massively parallel, multi-disciplinary barnes-hut tree code for extreme-scale n-body simulations. Computer Physics Communications, 183:880-889, 2012.
3. L. Kale, A. Arya, A. Bhatele, et al. Charm++ for productivity and performance: A submission to the 2011 HPC class II challenge. Technical Report 11-49, UIUC, 2011.
4. Y. Jo and M. Kulkarni. Enhancing locality for recursive traversals of recursive structures. In ACM OOPSLA'11.
5. S.C. Woo, M. Ohara, E. Torrie et al., Methodological considerations and characterization of the SPLASH-2 parallel application suite. In ACM ISCA'95.

Acknowledgments

This work was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357, and by the U.S. Department of Energy Sandia National Lab grant 1205852.

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN