New developments in the Charm++ load balancing framework

Abhinav Bhatele

Parallel Programming Laboratory
University of Illinois

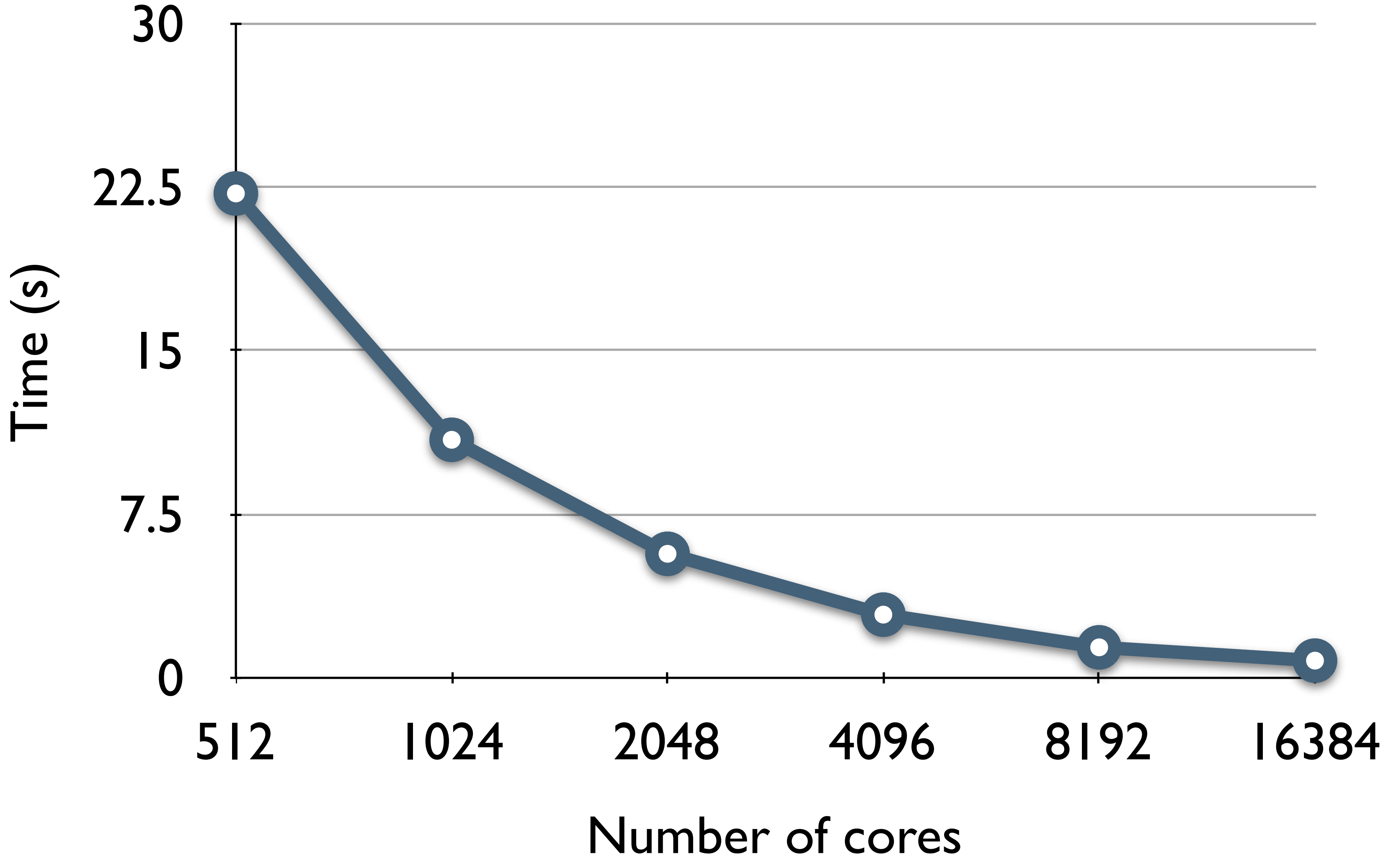
Load balancing in Charm++

- Seed load balancing: chares created as the program execution happens
- Measurement-based load balancing: based on the principle of persistence
    - Centralized load balancing
    - Hierarchical load balancing
    - Neighborhood load balancing

Seed load balancing

- Useful in the context of state space search problems where chares are fired during execution
  - Involves the movement of object creation messages (seed)
- Entry methods are called only once (no persistence)
- Fully distributed load balancing strategies:
  - Random seed assignment: close to optimal but can lead to high communication
  - Work stealing: Good for applications with lots of chares and leads to less communication
  - Neighborhood load balancing: Good for applications with few chares per processor, more proactive

Recent results



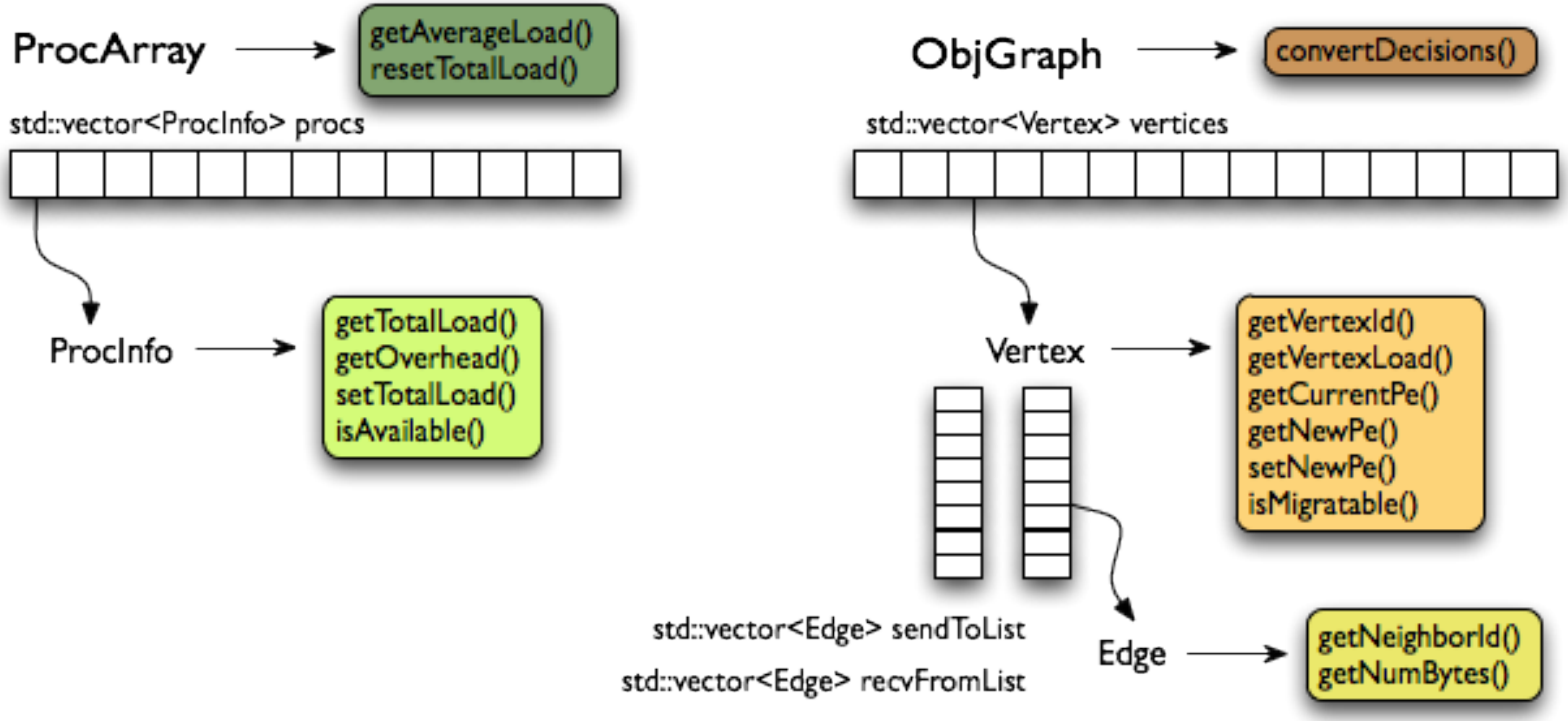
Unbalanced Tree Search speedup using seed load balancing low overhead (efficiency 88% on 16K cores) on Blue Gene/P (Yanhua Sun, Gengbin Zheng)

Measurement-based load balancing

- Based on the principle of persistence:
  "Computational loads and communication patterns tend to persist over time"
- Various centralized schemes in Charm
  - greedy, refinement-based
  - communication-aware, topology-aware
  - NUMA-aware, power-aware
  - library-based: METIS, Scotch

Interface to load balancing data

- Useful for communication-aware strategies



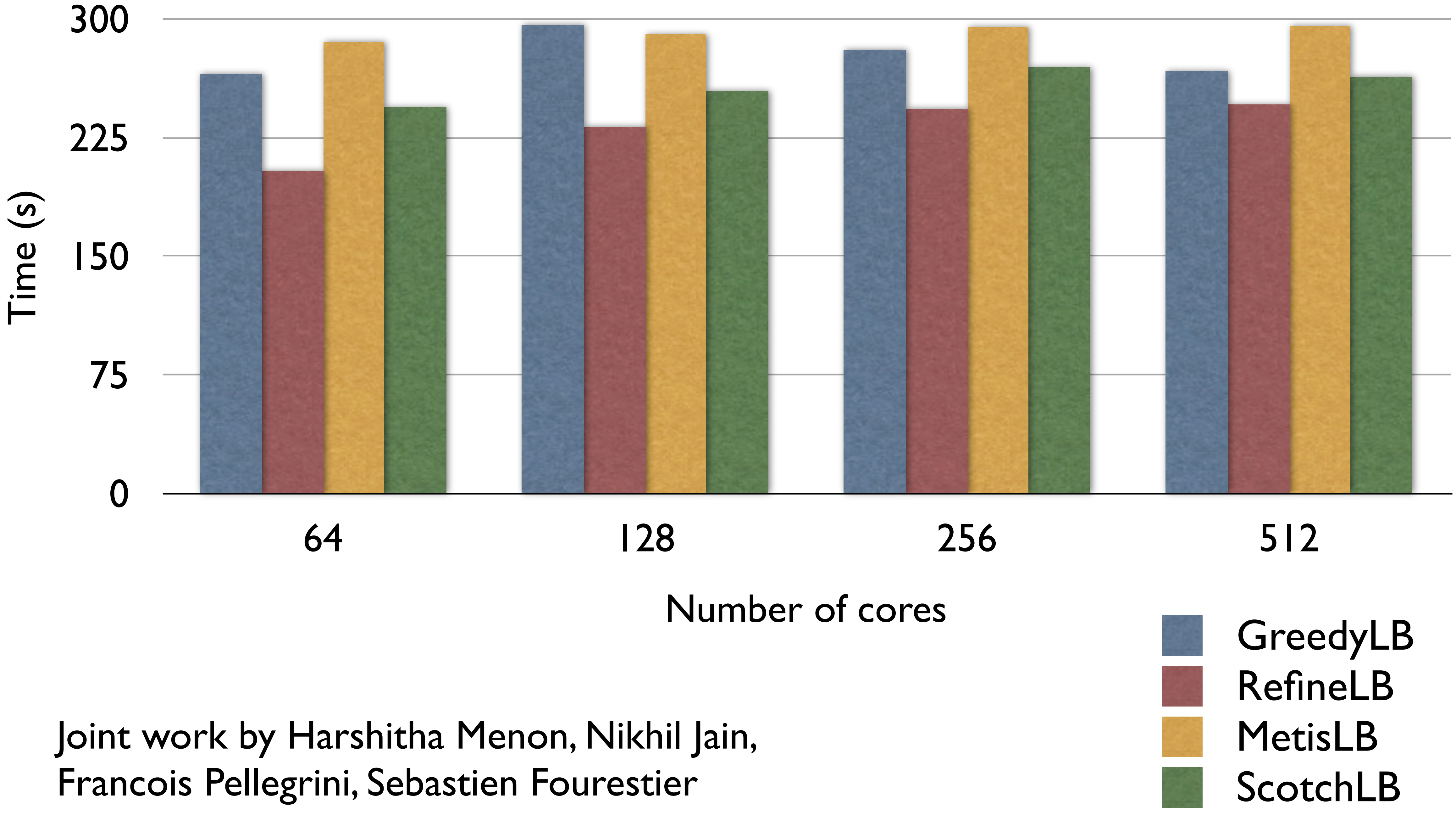
Writing a load balancer

```
void FooLB::work(LDStats *stats) {
  /** ========================== INITIALIZATION ============================= */
  ProcArray *parr = new ProcArray(stats);
  ObjGraph *ogr = new ObjGraph(stats);

  /** ============================= STRATEGY ================================ */

  /// The strategy goes here
  /// The strategy goes here
  /// The strategy goes here
  /// The strategy goes here
  /// The strategy goes here

  /** ============================== CLEANUP ================================ */
  ogr->convertDecisions(stats);
}
```

Example strategy

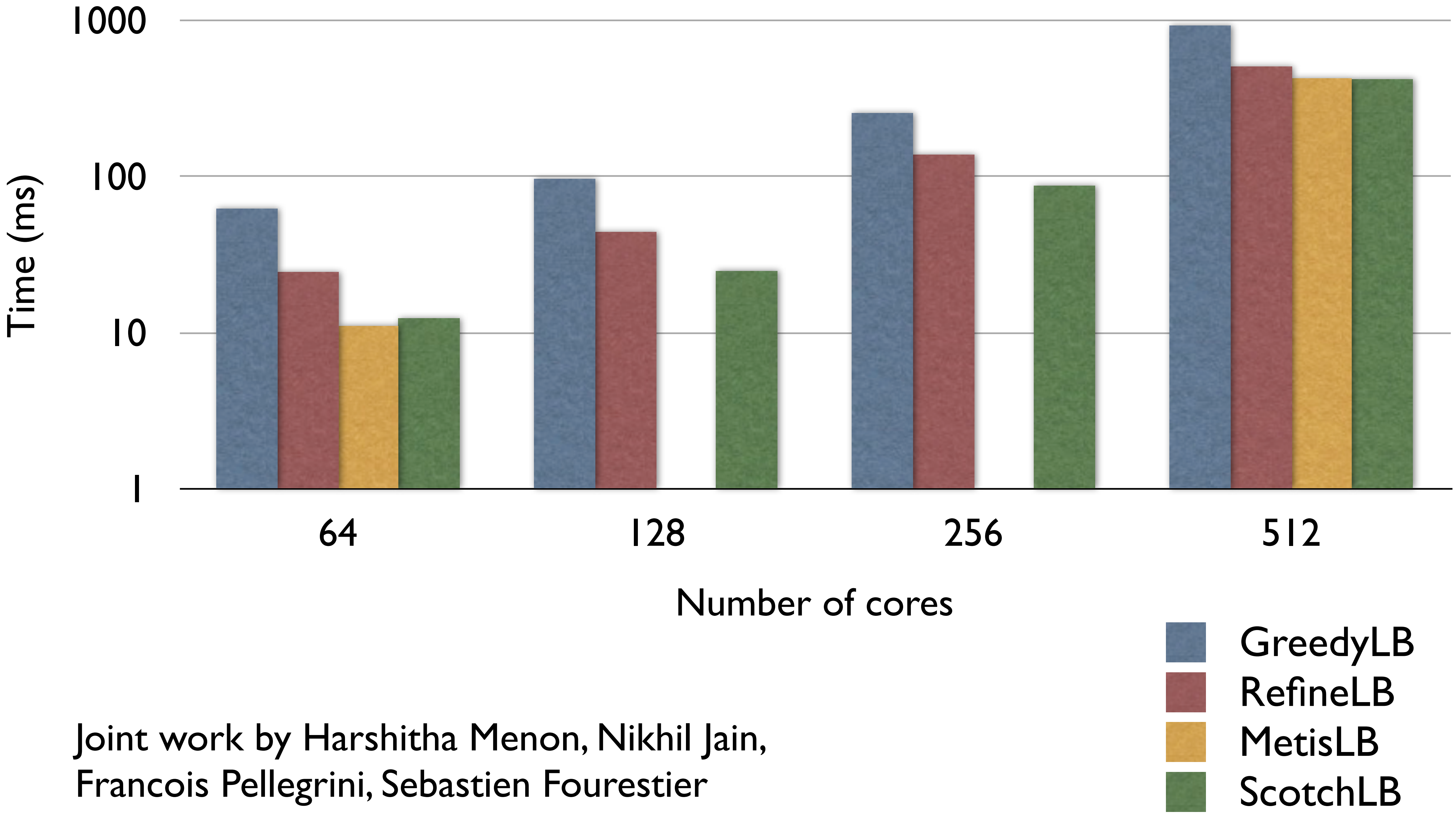
```
  // breadth first traversal
  while(!vertexq.empty()) {
    start = vertexq.front();
    vertexq.pop();

    for(i = 0; i < ogr->vertices[start].sendToList.size(); i++) {
      // look at all neighbors of a node in the queue and map them while
      // inserting them in the queue (so we can look at their neighbors next)
      nbr = ogr->vertices[start].sendToList[i].getNeighborId();
      if(ogr->vertices[nbr].getNewPe() == -1) {
        vertexq.push(nbr);

        if(parr->procs[nextPe].getTotalLoad() + ogr->vertices[nbr].getVertexLoad() >
avgLoad) {
          nextPe++;
          avgLoad += (avgLoad - parr->procs[nextPe].getTotalLoad())/(numPes-nextPe);
        }
        ogr->vertices[nbr].setNewPe(nextPe);
        parr->procs[nextPe].setTotalLoad(parr->procs[nextPe].getTotalLoad() + ogr-
>vertices[nbr].getVertexLoad());
      }
    } // end of for loop
  } // end of while loop
```

3D imbalanced stencil



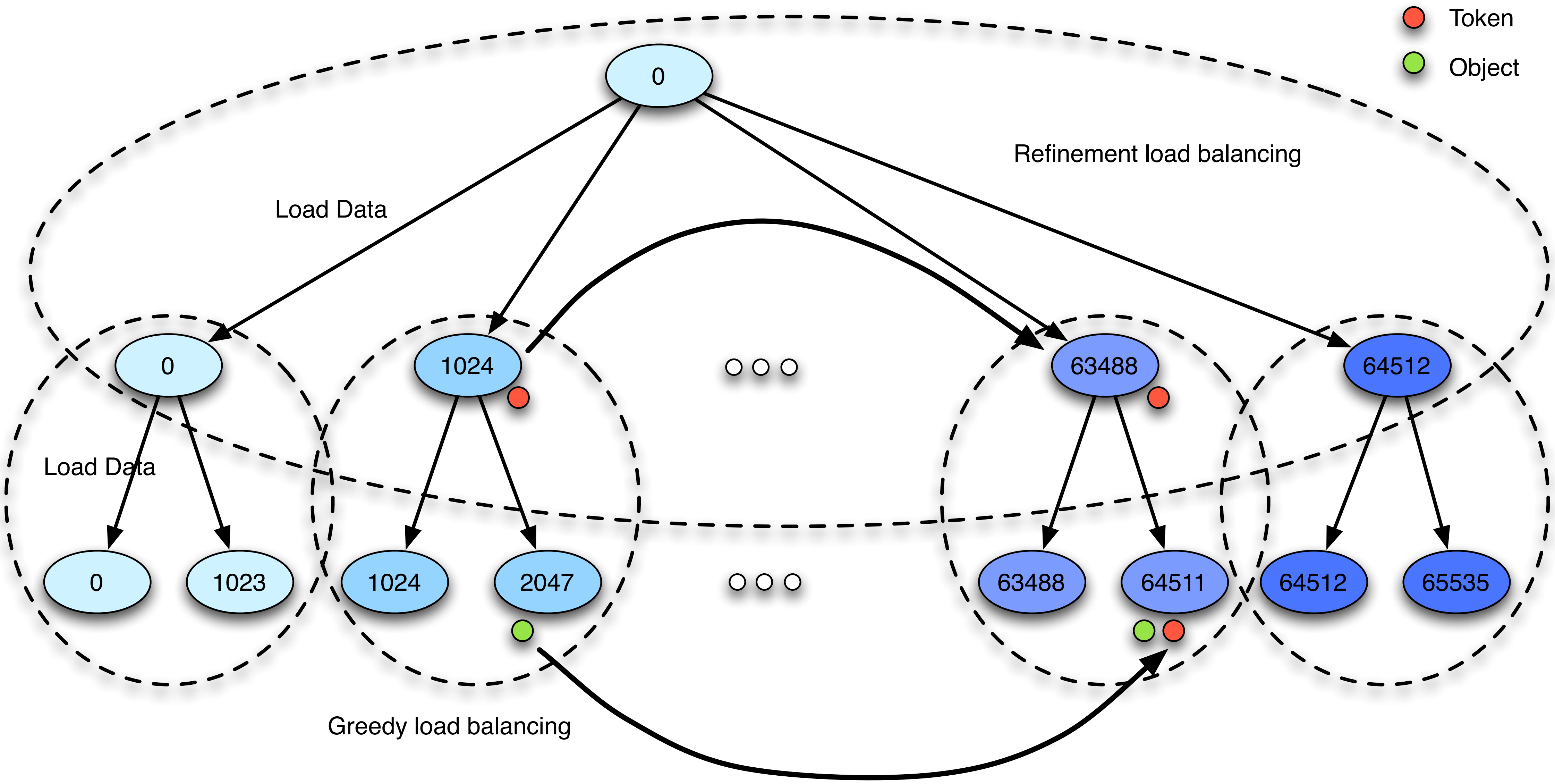
Joint work by Harshitha Menon, Nikhil Jain, Francois Pellegrini, Sebastien Fourestier

kNeighbor



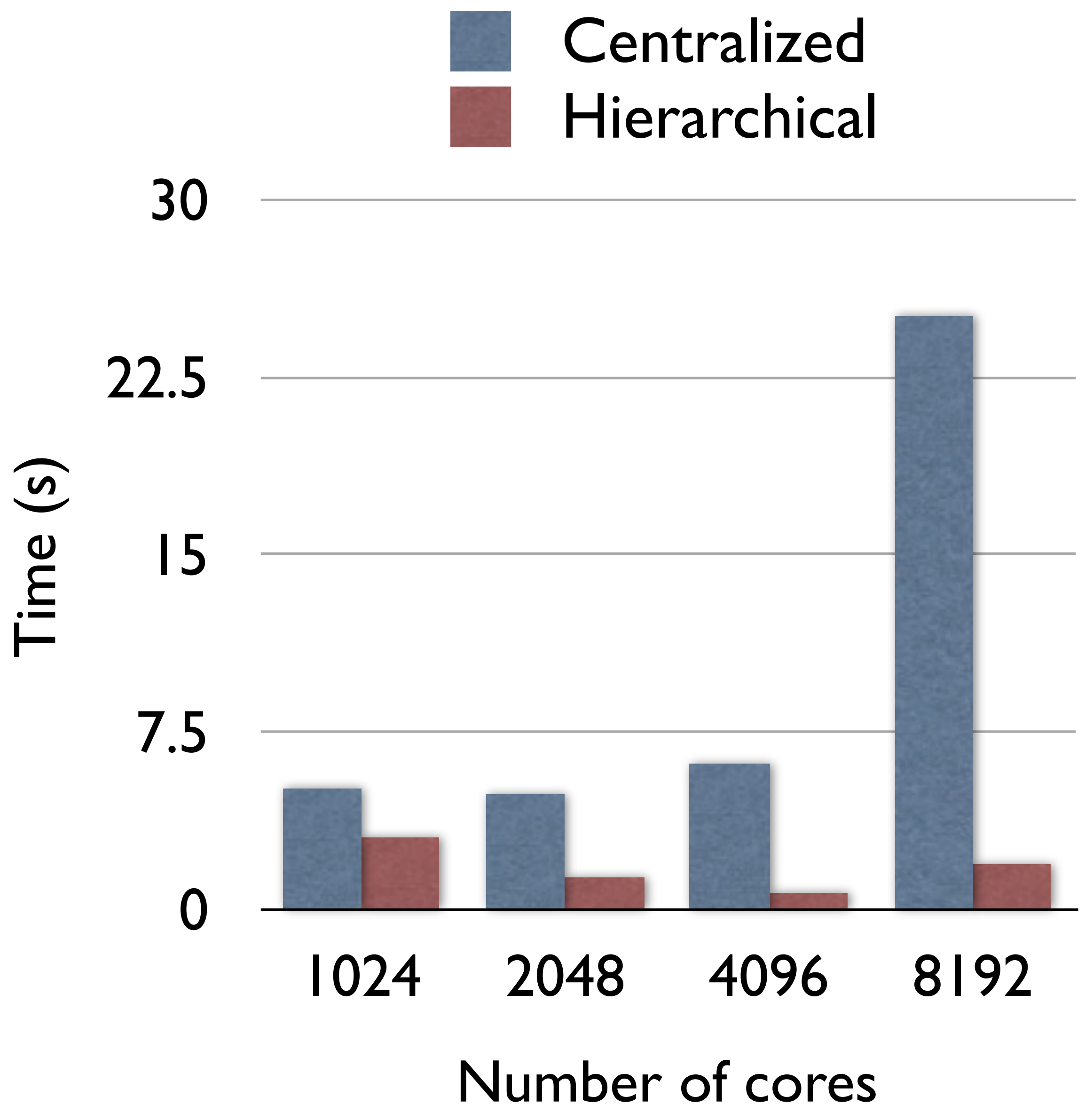
Joint work by Harshitha Menon, Nikhil Jain,
Francois Pellegrini, Sebastien Fourestier

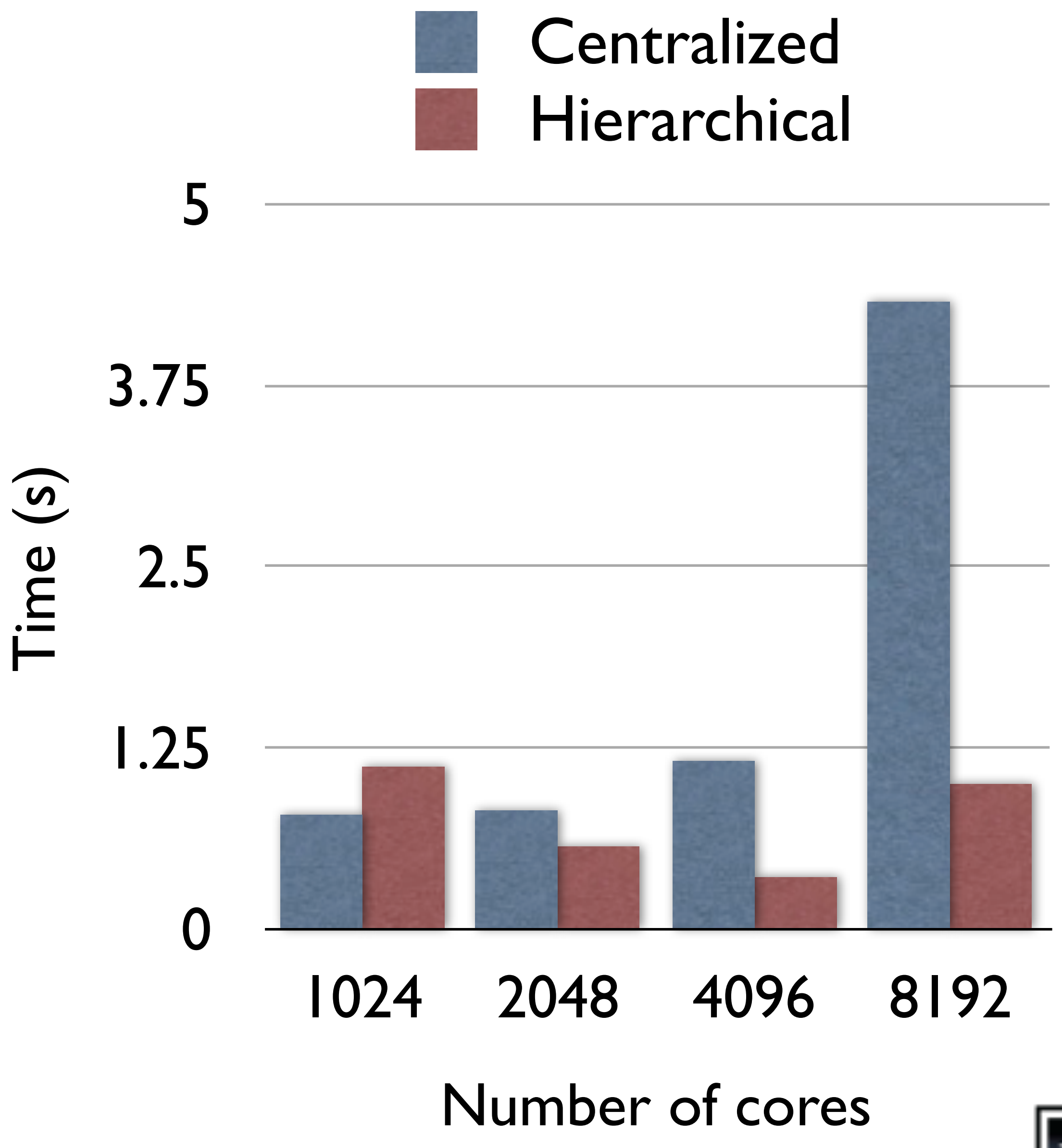
Load balancing in NAMD



Joint work by Gengbin Zheng, Esteban Meneses, Abhinav Bhatele

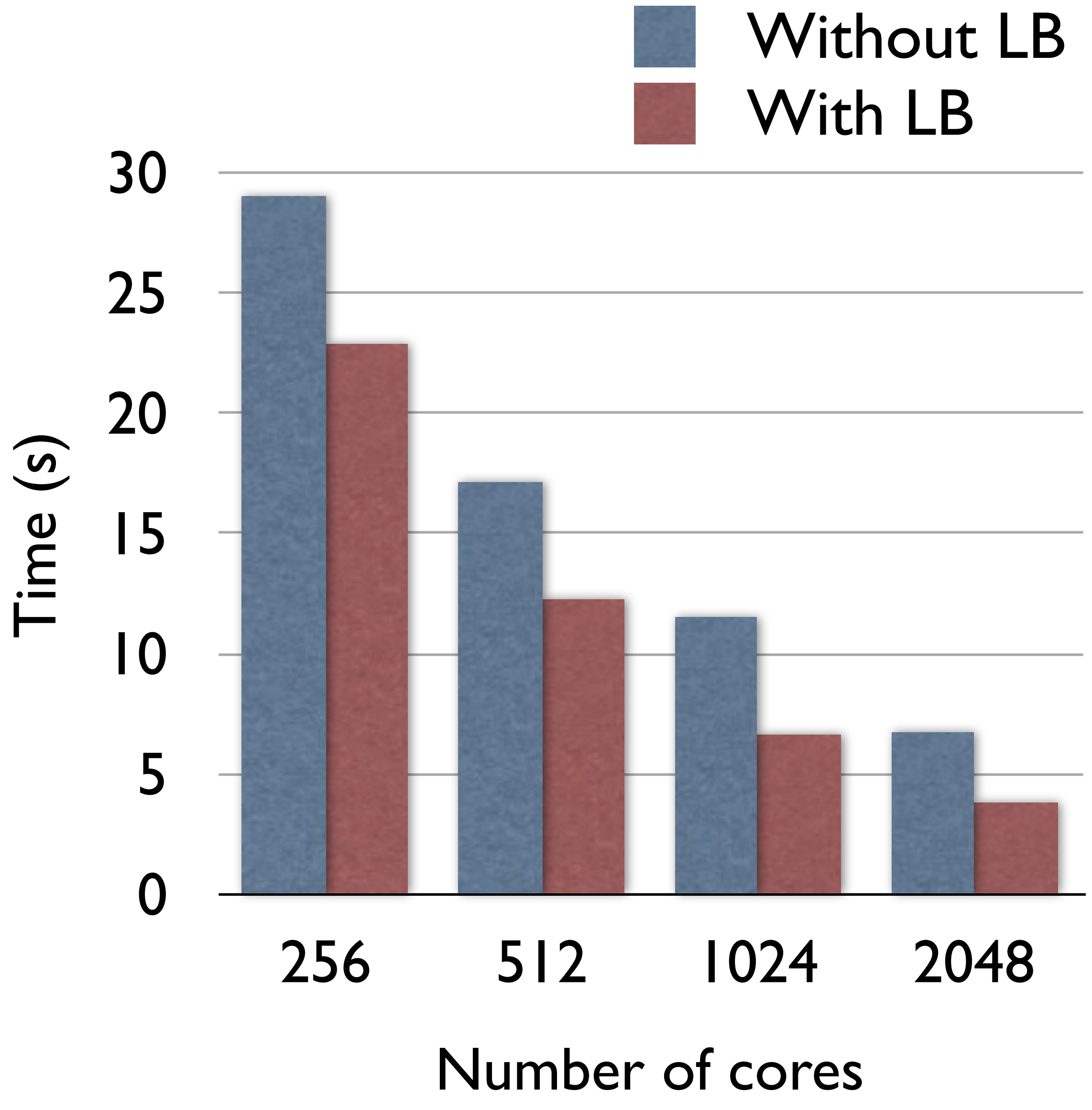
1 million atoms on BG/P





Load balancing in ChaNGa

- Based on approximating chares by their centroid
- Orthogonal recursive bisection in three dimensions



Dwarf (5 million particles) running on Blue Gene/P

Joint work by Pritish Jetley and other members of the ChaNGa group

Load Balancing Contest