Case Studies in Asynchronous, Message-Driven Shared Memory Programming

Pritish Jetley
Parallel Programming Laboratory
pjetley2@illinois.edu

Outline

- Shared memory programming today
- Charm++ on multicore systems
- Shared memory (SM) programming in Charm++
- Case studies
  - Barnes-Hut (SPLASH)
  - SAH-based *k*d-tree construction

SM programming today

- Fork-join
  - Amorphous, thread-based (pthreads)
  - Data parallelism-centric (OpenMP)
  - Tasks (TBB, Cilk)

- Message-driven execution (Charm++)

Fork-join model

+	-
Simple to program (?)	Forced synchrony
Global view of control	Low-level Mutex
Natural fit for certain problems	Grainsize control

Charm++ on multicore systems

- Decompose algorithm into objects encapsulating its natural elements
- Objects present reactive interfaces
- Control flows through asynch. entry method invocations
- Data flows through pointer exchange

SM programming with Charm++ and MDE

+

Natural decomposition

Dependencies = messages

Asynchrony

Dynamic
load balancing

Task prioritization

-

No global view of control

Charm++ has no
faults whatsoever

MDE is low-level

Performance and productivity studies

- How easy (or hard) is it to write SM programs in Charm++?
- Can we expect improvements in performance?
- Are there abstractions that would improve programmability in Charm++?

Comparison points

- SPLASH2 Barnes-Hut benchmark
  - Study evolution of self-gravitating systems
  - Tree-based code
  - Uses pthreads

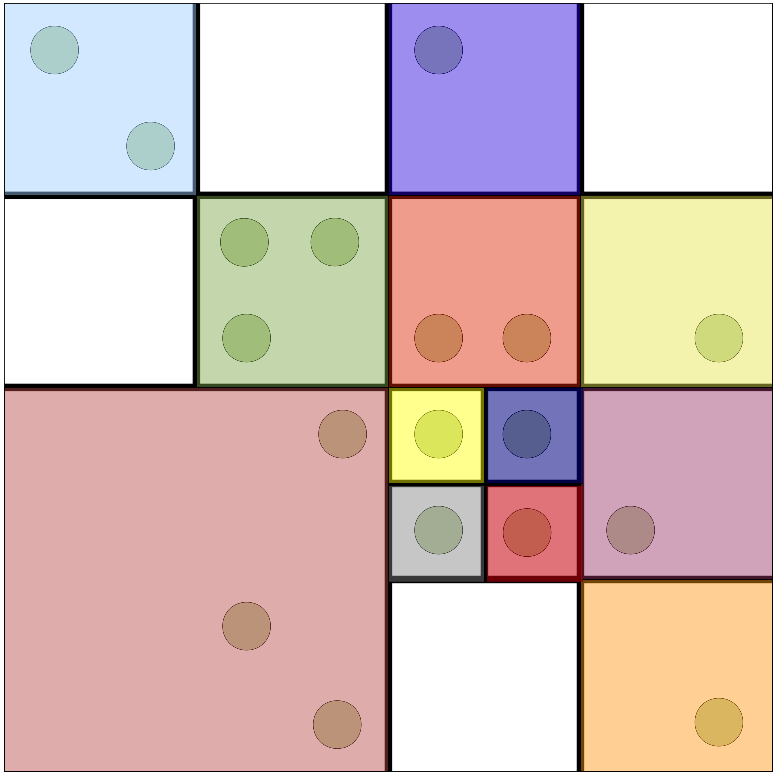
- SAH-based *k*d-tree construction
  - High-performance ray tracing
  - Nested parallelism
  - Uses TBB

SPLASH Barnes-Hut

- Domain decomposition and tree building
  - Partition space into compact, disjoint regions containing approximately equal numbers of particles
  - Regions arranged in an octree
  - Independent subtrees: **task parallel**
  - Shuffle particles into child bins: **data parallel**
- Force calculation
  - Objects own non-intersecting sets of particles, and calculate forces on them

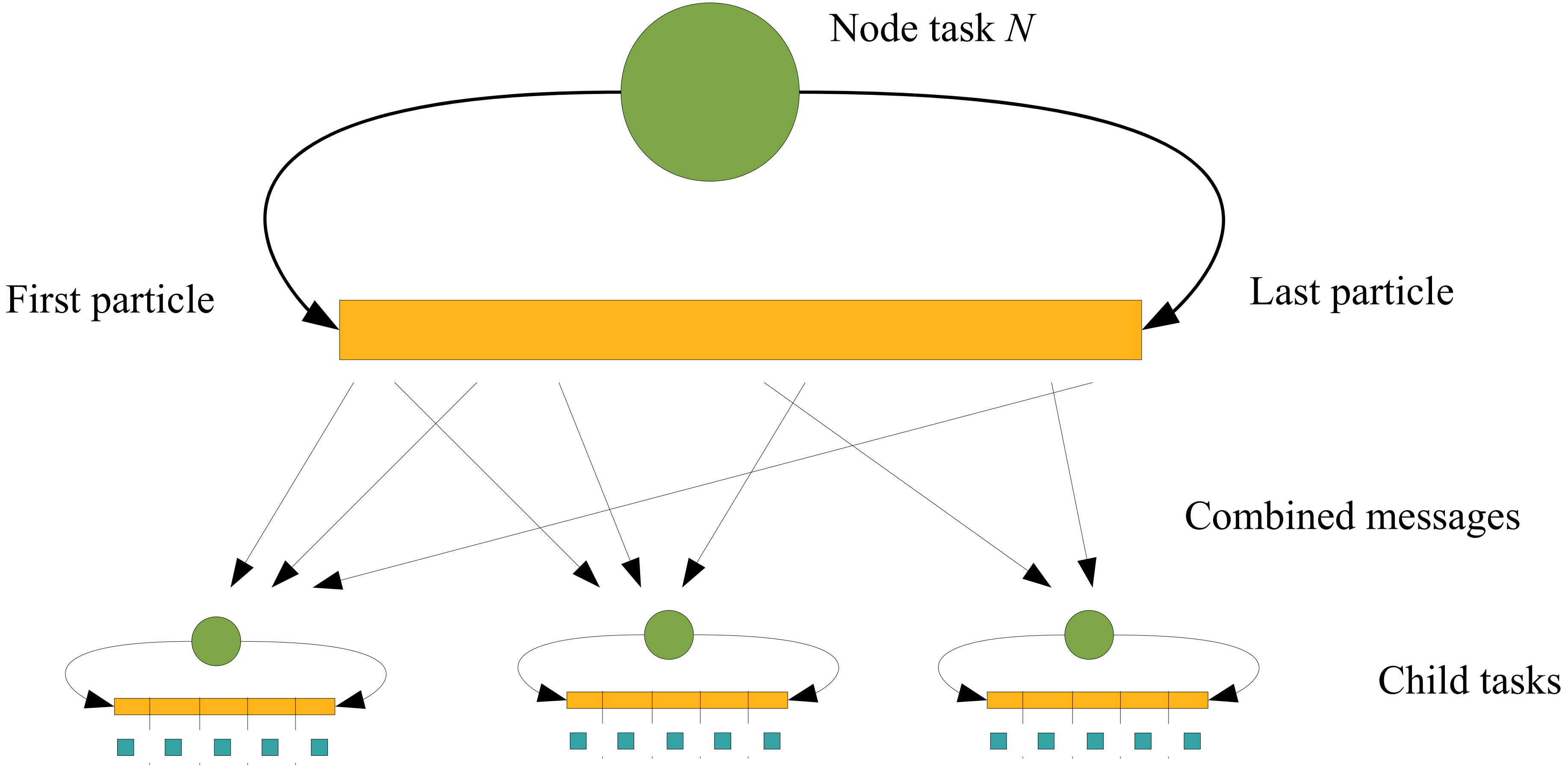
Decomposition

- Recursively divide partition into quadrants if more than *τ* particles within it



τ = 3

Domain decomposition



Decomposition with pthreads

```
void decompose(){
    for(int I = 0; I < myNP; I++){
        Particle *p = myParticles[I];
        Cell *cell = g_root;
        while(1){
            cell->LOCK();
            if(!cell->isLeaf()){
                save = cell;
                int which = cell->which(p->key);
                cell = cell->child(which);
                save->UNLOCK();
            }
            else{
                cell->particles.add(p);
                cell->split();
                cell->UNLOCK();
                break;
            }
        }
    }
}
```

Decomposition with Charm++

```
TreePiece::recvParticles(Particle *ptr, int np){
 if(myRoot->isLeaf()){
  myRoot->addParticles(ptr,np);
  if(myRoot->split()){
   forwardParticlesToChildren(myRoot->particles);
  }
 }
 else{
  forwardParticlesToChildren(ptr,np);
 }
}
```

```
void T
 for(i
  tre
                                                  childParticles[I],
                                                  childPartilces[I].size());
 }
}
```

```
void TreePiece::flushParticles(int I){
 treePieceProxy[I].recvParticles(buffered[I],
                                                    buffered[I].size());
}
```

Tree traversal

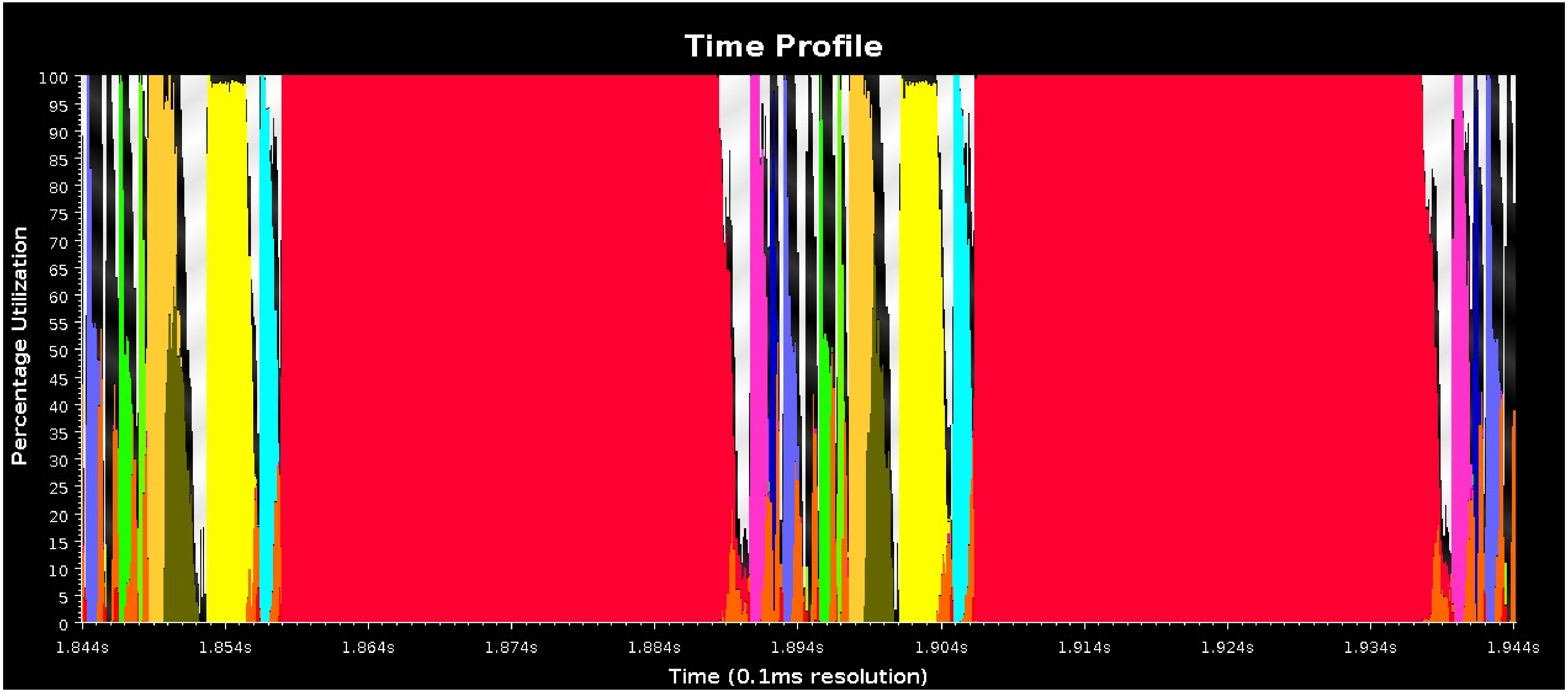
```
Traverse(Leaf b, Node n){
  if(IsLeaf(n)){
    LeafForces(b,n);
  }
  else if(Side(n)/|r(n)-r(b)|
< Theta_T){
    CellForces(b,n);
  }
```

Fewer barriers

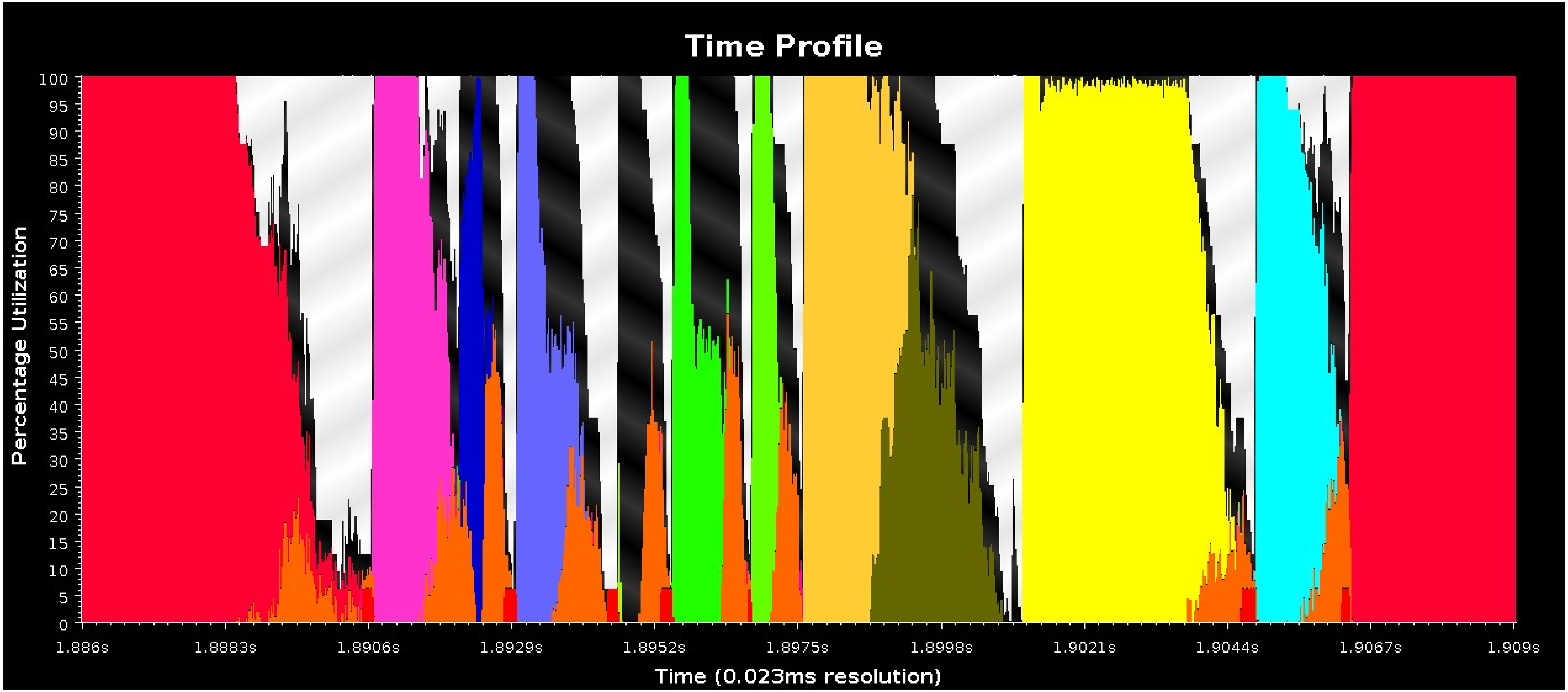
Title:100k.1.comparison.eps
Creator:gnuplot 4.2 patchlevel 6
CreationDate:Tue Apr 19 01:05:26 2011

Title:10k.1.comparison.eps
Creator:gnuplot 4.2 patchlevel 6
CreationDate:Tue Apr 19 01:03:33 2011

Performance profile



Performance profile



More results

Title:10k.2.comparison.eps
Creator:gnuplot 4.2 patchlevel 6
CreationDate:Tue Apr 19 01:08:11 2011

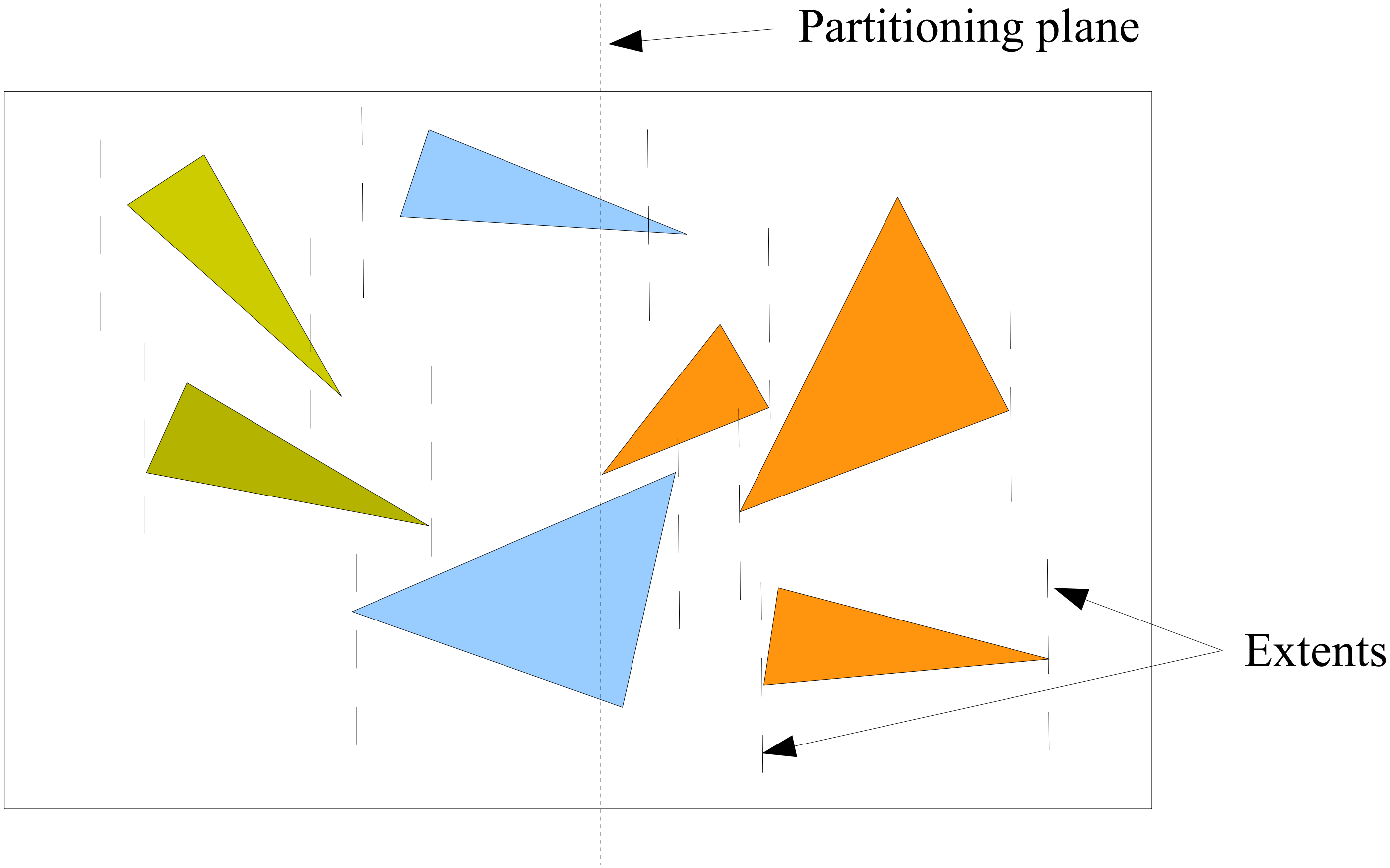
Title:100k.2.comparison.eps
Creator:gnuplot 4.2 patchlevel 6
CreationDate:Tue Apr 19 01:08:05 2011

SAH-based *k*d-trees

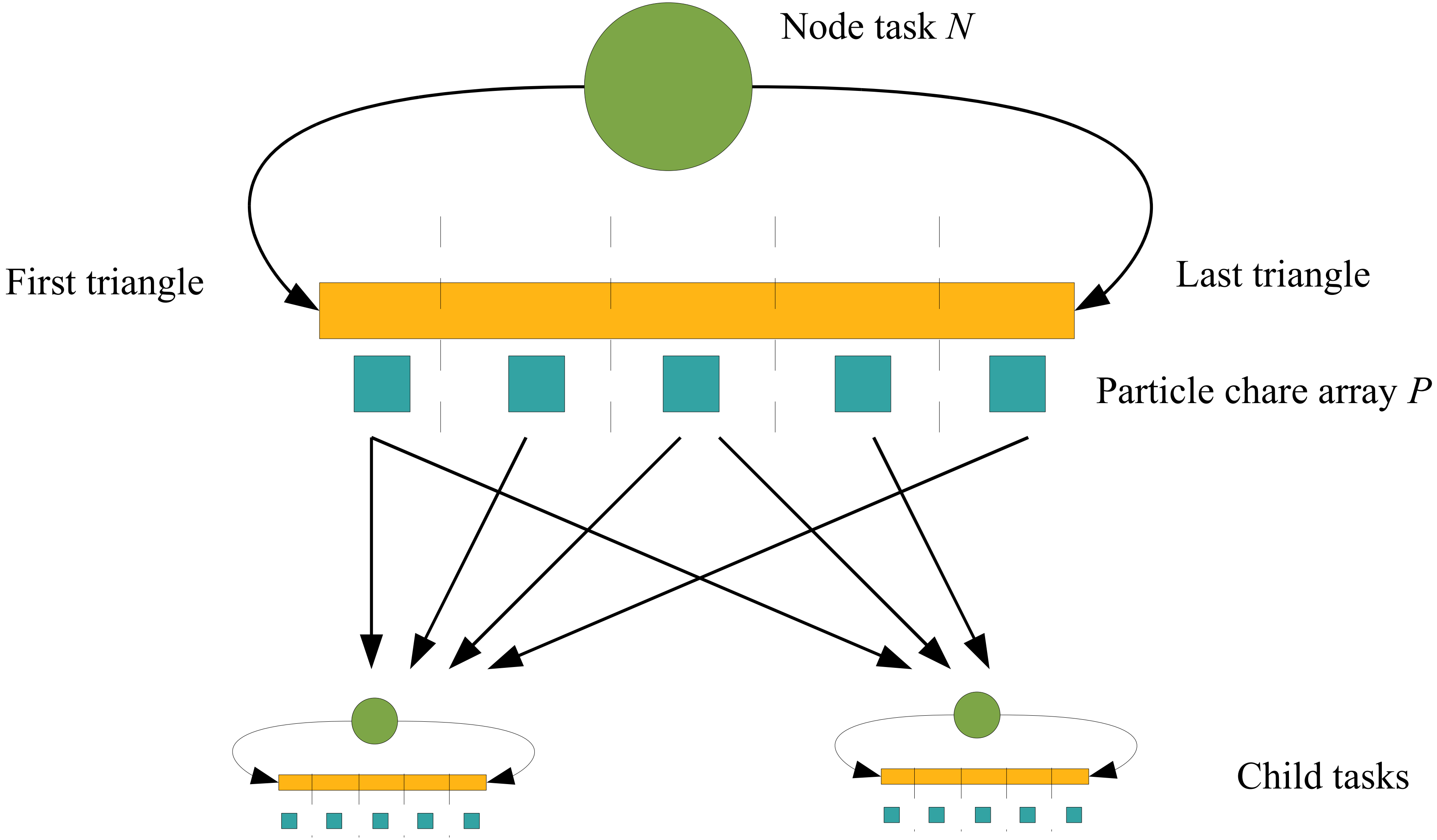
- Used to efficiently render complex graphical scenes
- **Task parallel** construction of independent subtrees (dynamically created *chares*)
- **Data parallel** calculation of node split point (*chare arrays*)

Binary Space Partitioning

- SAH decides position of partition based on triangle distribution and partition surface area



*k*d-tree construction



Node task *N*

First triangle

Last triangle

Particle chare array *P*

Child tasks

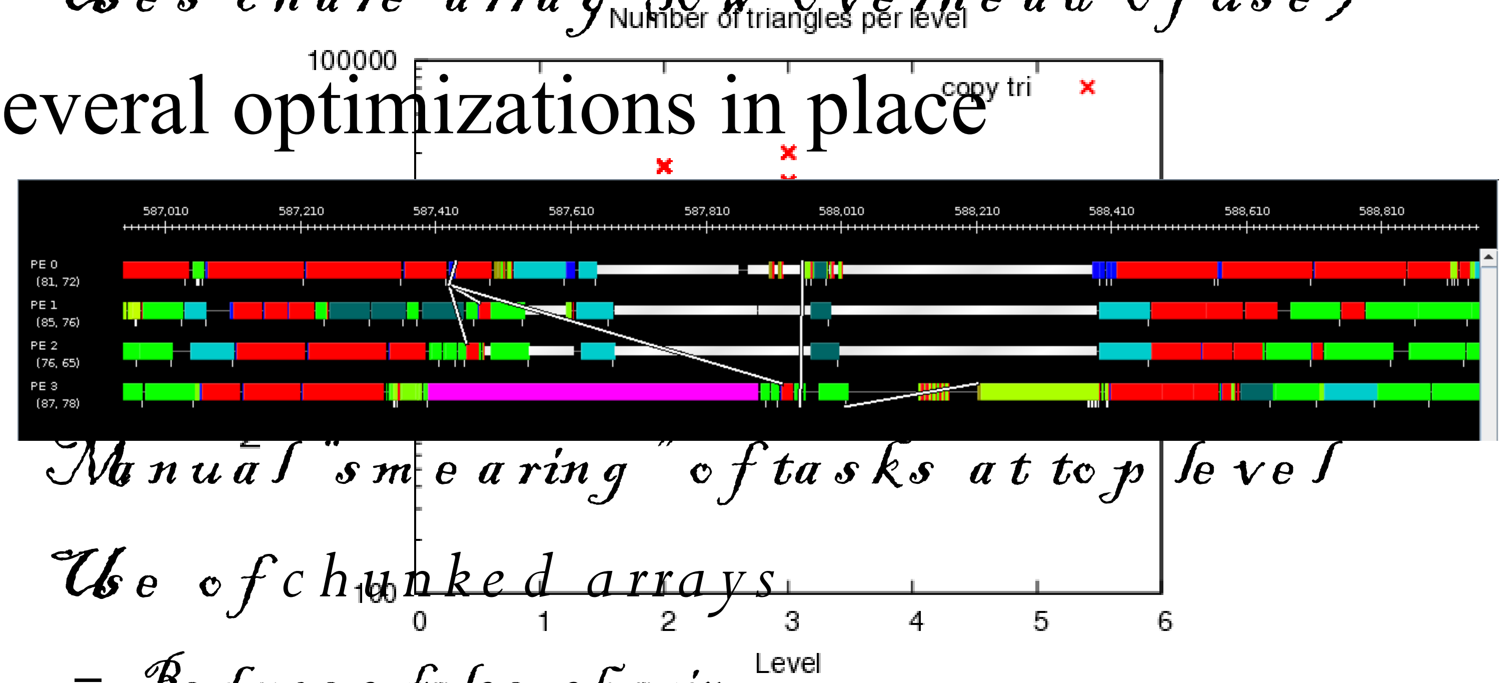
Charm++ pseudocode

- Use SDAG to sequence events in parallel scan

```
entry void Worker::scanTriangleCounts(ActivationRec ar,
                                      NodeTaskID N){
dist = W >> 1;
while (dist > 0){
 if(thisIdx < dist){
  ScanMsg m;
  m.NL = myNL; m.NR = ar.nTris-myNR;
  RefNum (m) = dist;
  workers[thisIdx+dist].recvNeighborCounts (m);
 }
 when recvNeighborCounts[dist](ScanMsg m1){
  myNL += m.NL; myNR -= m.NR;
  dist >>= 1;
 }
}
Plane bestPlane = calculateSAH();
reduce(bestPlane,N,NodeTask::getBestPlanes);
}
```

Charm++ implementation

- One chare for each node of kd-tree (orchestrator)
- For data-parallel operations, orchestrator either
  - Fires new chares (dynamic load balance)
  - Uses chare array (low overhead of use)
- Several optimizations in place
  -
  -
  - Manual "smearing" of tasks at top level
  - Use of chunked arrays
    - Reduces false sharing
    - Reduces amount of coordination communication



Results

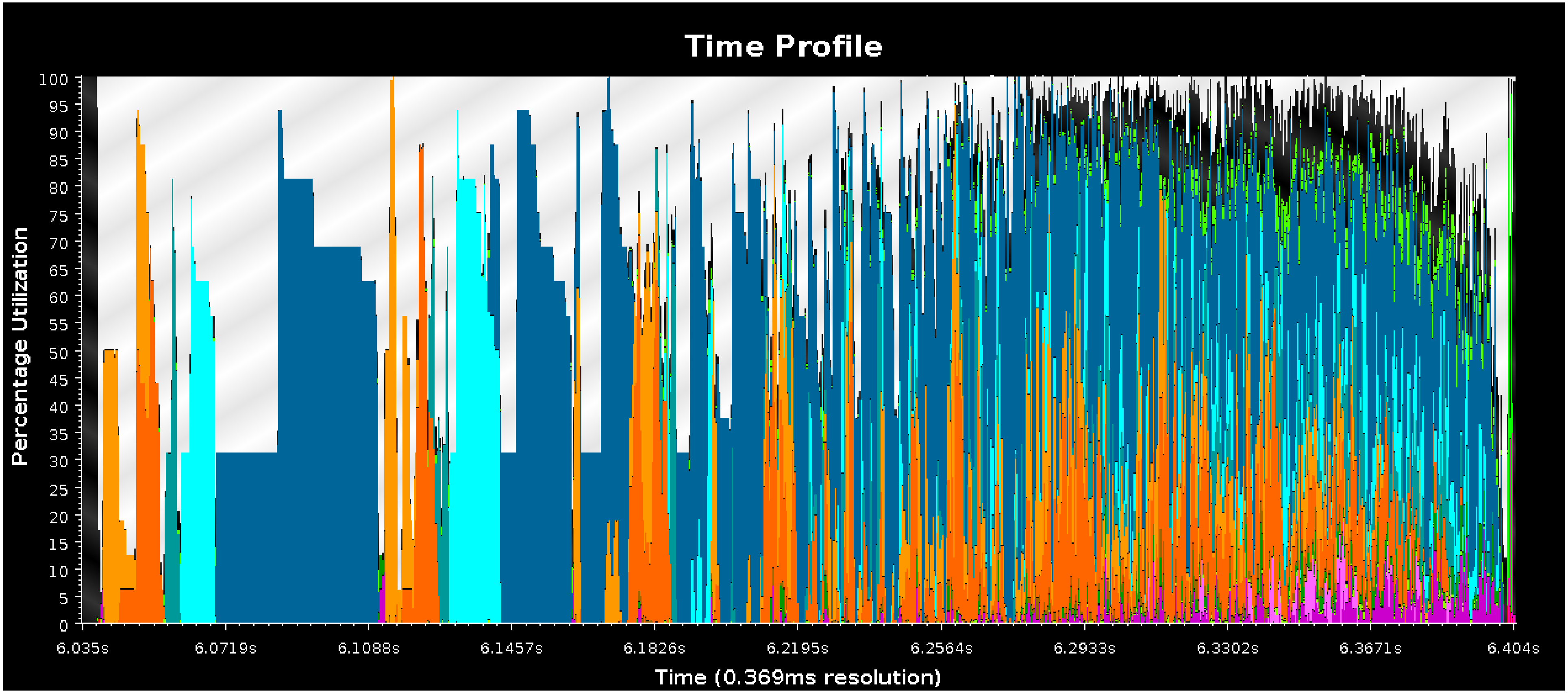
Title:bunny.eps
Creator:gnuplot 4.2 patchlevel 6
CreationDate:Tue Apr 19 01:18:08 2011

Title:fairy.eps
Creator:gnuplot 4.2 patchlevel 6
CreationDate:Tue Apr 19 01:18:08 2011

Title:angel.eps
Creator:gnuplot 4.2 patchlevel 6
CreationDate:Tue Apr 19 01:18:08 2011

Title:happy.eps
Creator:gnuplot 4.2 patchlevel 6
CreationDate:Tue Apr 19 01:18:08 2011

Performance profile


Time Profile
Percentage Utilization
6.035s
6.0719s
6.1088s
6.1457s
6.1826s
6.2195s
6.2564s
6.2933s
6.3302s
6.3671s
6.404s
Time (0.369ms resolution)