Microsoft Research

Calico: Rethinking the Language / Runtime-System Boundary

Ross McIlroy, Steven Smith
and Tim Harris

Ubiquitous Multi-Core

- Traditionally parallelism was confined to HPC systems
- Multi-core now pervades commodity systems
  - Consumer applications are being forced to turn to parallelism to increase performance
- Techniques employed by the HPC community do not necessarily carry over
  - Don't "own" the machine
  - Multiple competing applications
  - Different latency / throughput priorities
  - Workload typically much shorter and less predictable

Calico

- A low-level parallel runtime system
  - Support C / C++ applications
- Tackle challenges in future architectures
  - Heterogeneous Cores
  - Changing core-counts (interactive apps, energy saving, dynamic core reconfiguration)
  - Non-cache coherency
- Task-based, fork-join programming model
  - Programmer specifies what can be run in parallel
  - Runtime system decides what is run in parallel
  - Integrate with AC / THC, provide parallelism as well as asynchrony

Calico Programming Model

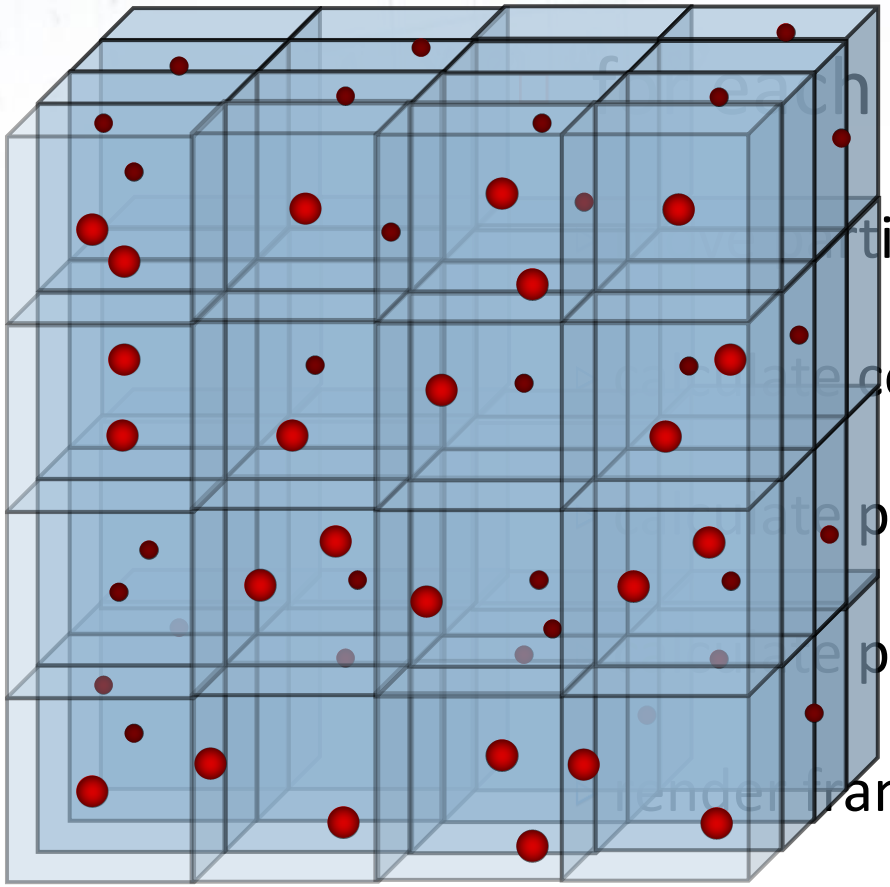
```
do {
    async msg_send(core 1, “Computing Forces”);
    par fluid                                  ls, range);
} finish;
```

Spawn a bunch of parallel tasks that can be run across multiple cores

Wait for parallel **and** async tasks to complete before continuing

Microsoft Research

FluidAnimate



for each frame

Invalidate particles to correct cell

Calculate cell density

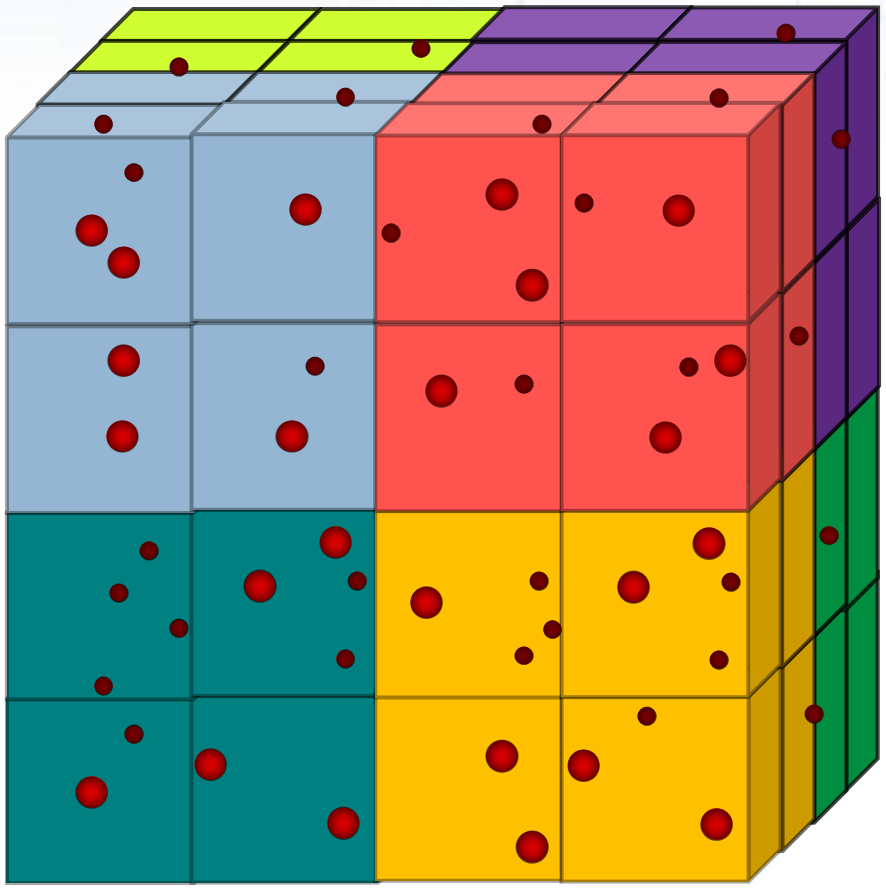
Calculate particle forces

Calculate particles position

Render frame

Microsoft Research

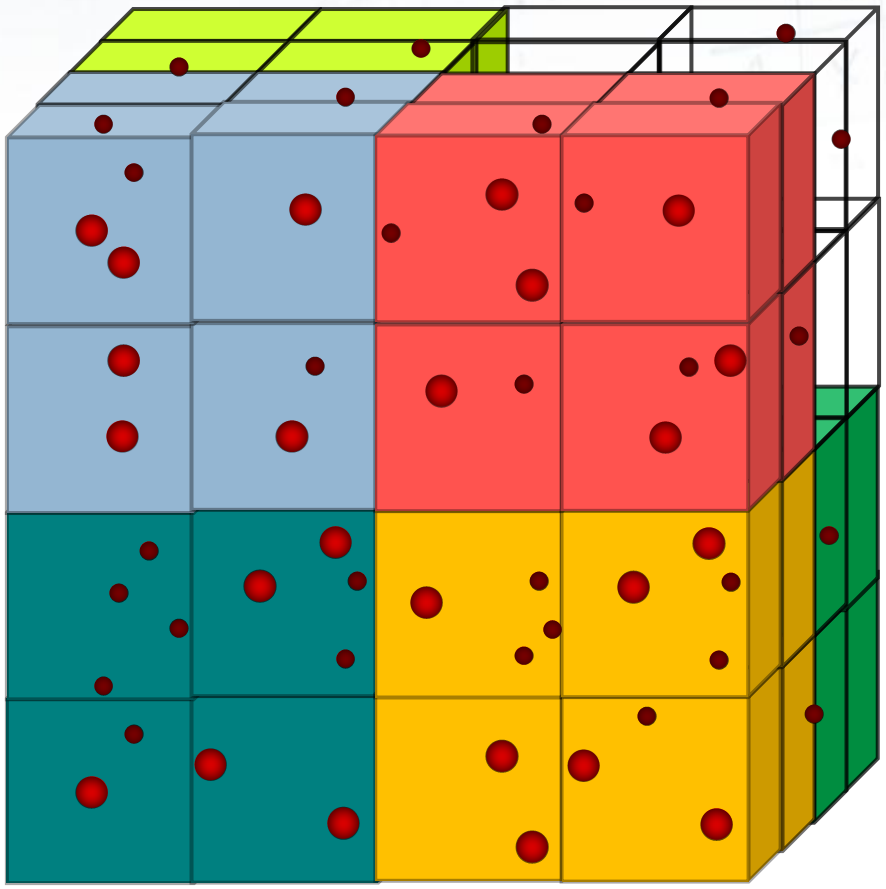
Static Partitioning



- for each frame
  - move particles to correct cell
  - calculate cell density
  - calculate particle forces
  - calculate particles position
  - render frame

Microsoft Research

Static Partitioning

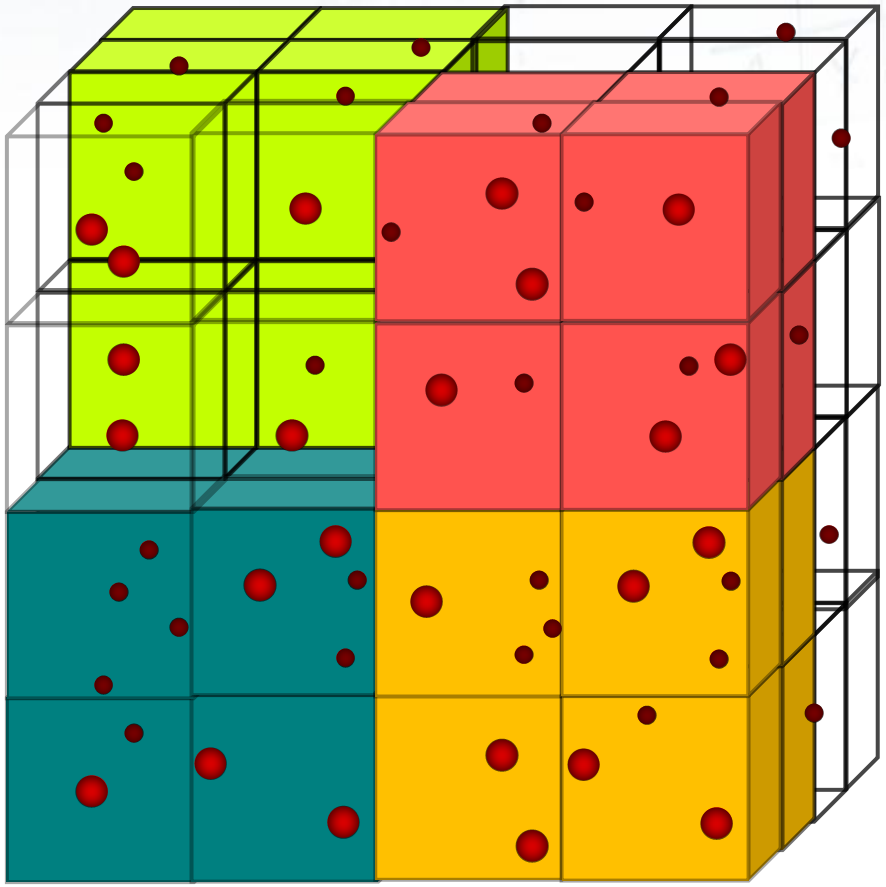


- for each frame
  - move particles to correct cell
  - calculate cell density
  - calculate particle forces
  - calculate particles position
  - render frame

Microsoft Research

Static Partitioning

- Problem: Uneven workload

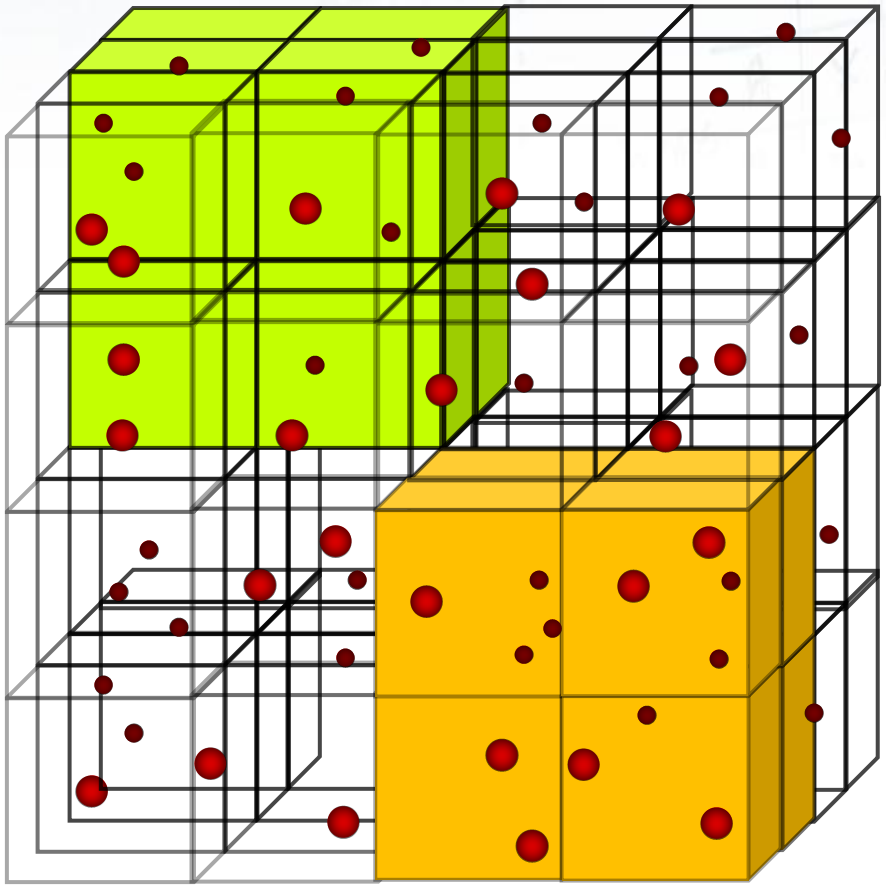


- for each frame
  - move particles to correct cell
  - calculate cell density
  - calculate particle forces
  - calculate particles position
  - render frame

Microsoft Research

Static Partitioning

- Problem: Barrier Synchronization

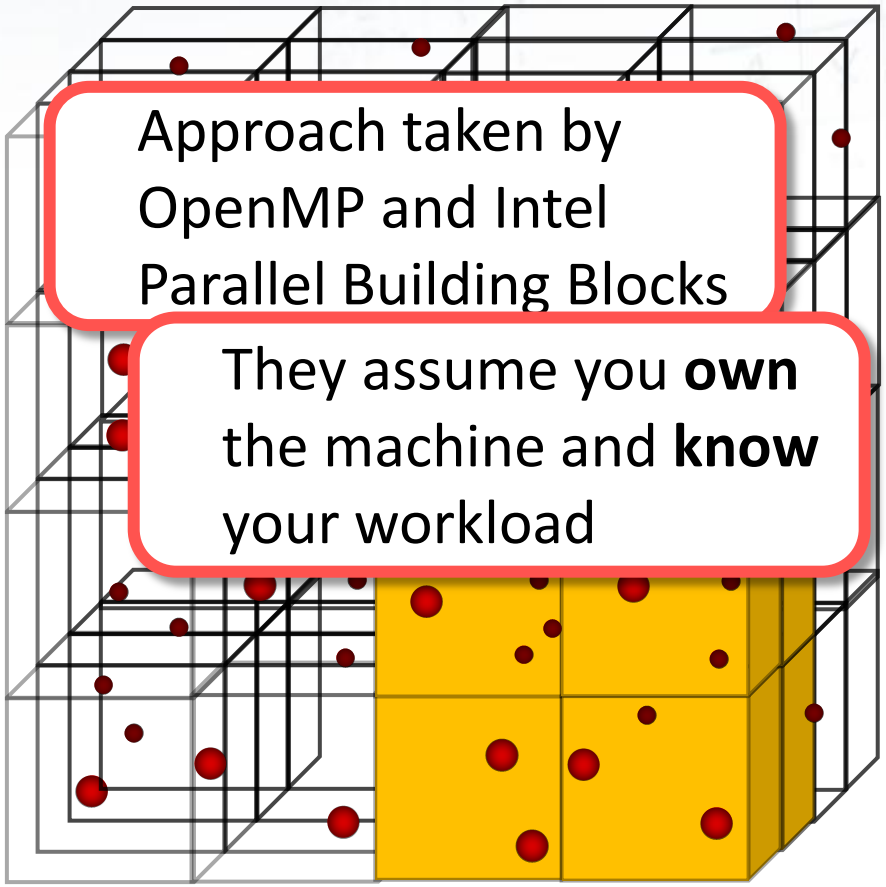


- for each frame
  - move particles to correct cell
  - calculate cell density
  - calculate particle forces
  - calculate particles position
  - render frame

Microsoft Research

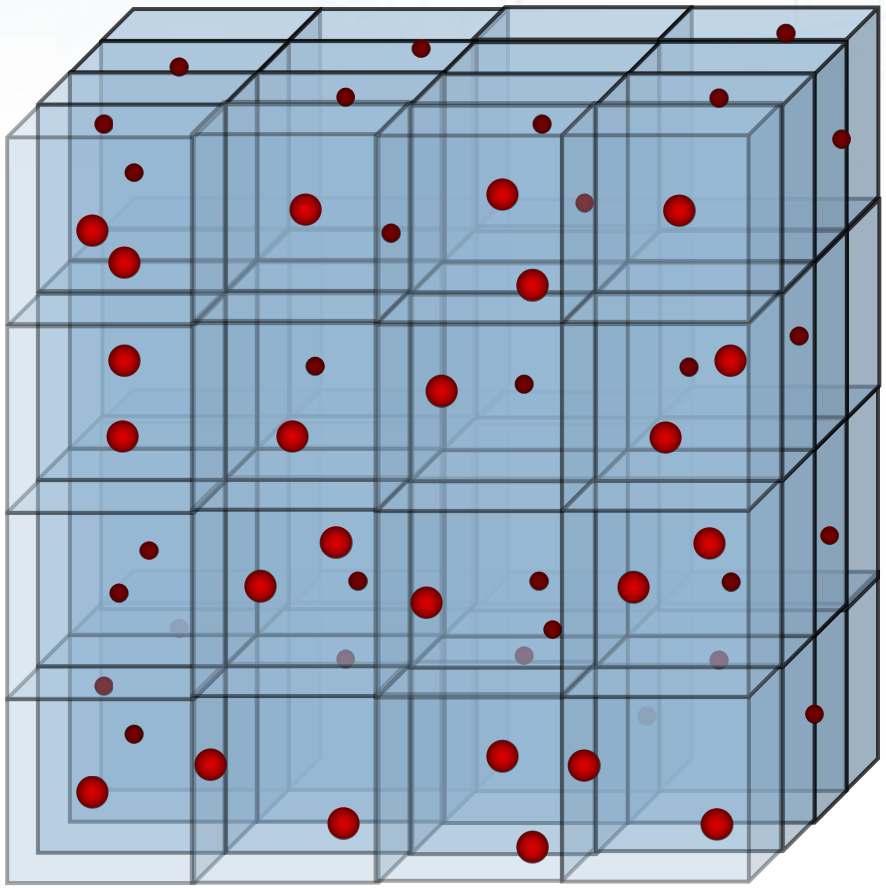
Static Partitioning

- Problem: Thread Preemption



- for each frame
  - move particles to correct cell
  - calculate cell density
  - calculate particle forces
  - calculate particles position
  - render frame

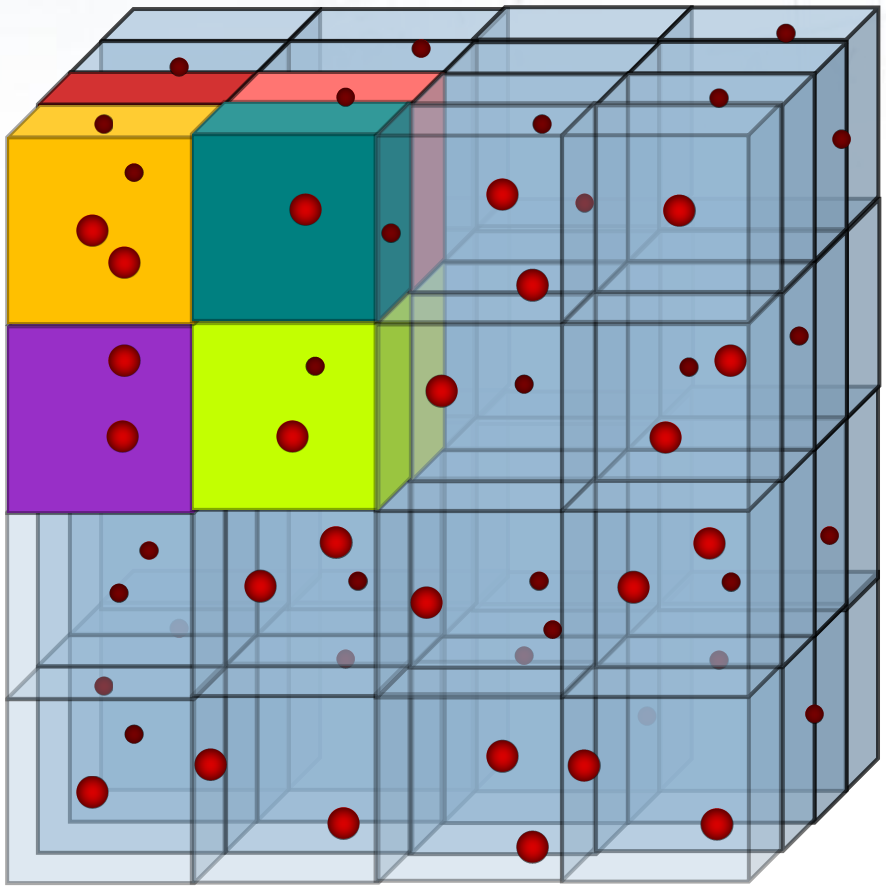
Dynamic Partitioning (Work-Stealing)



- for each frame
  - move particles to correct cell
  - calculate cell density
  - calculate particle forces
  - calculate particles position
  - render frame

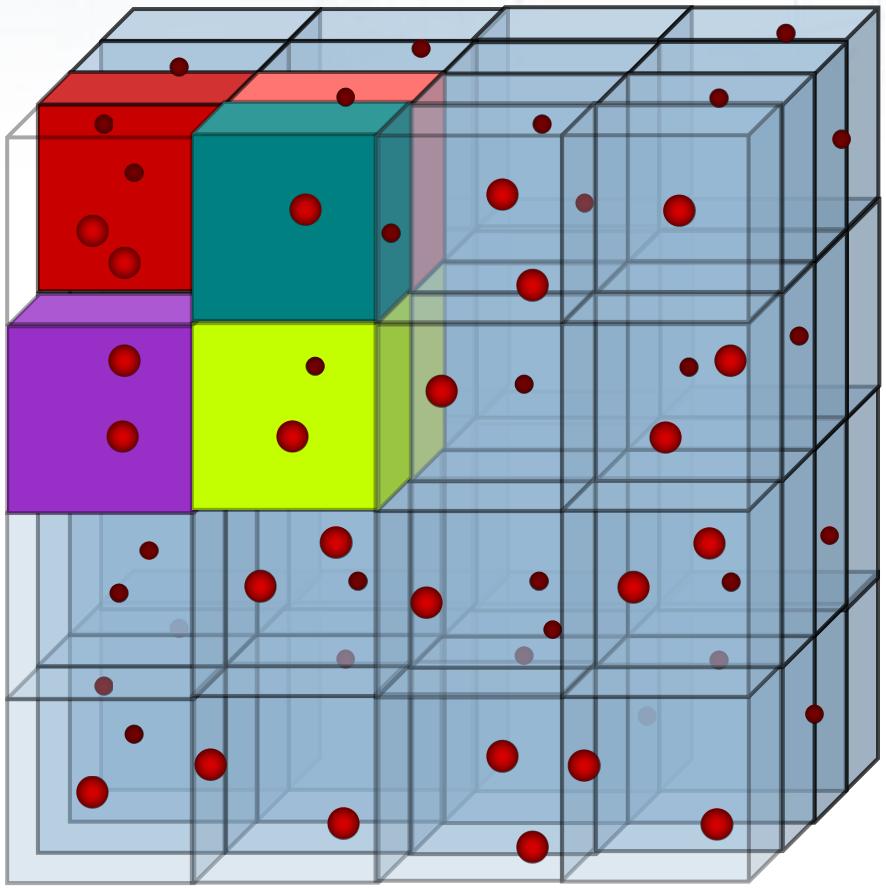
Microsoft Research

Work-Stealing



- for each frame
  - move particles to correct cell
  - calculate cell density
  - calculate particle forces
  - calculate particles position
  - render frame

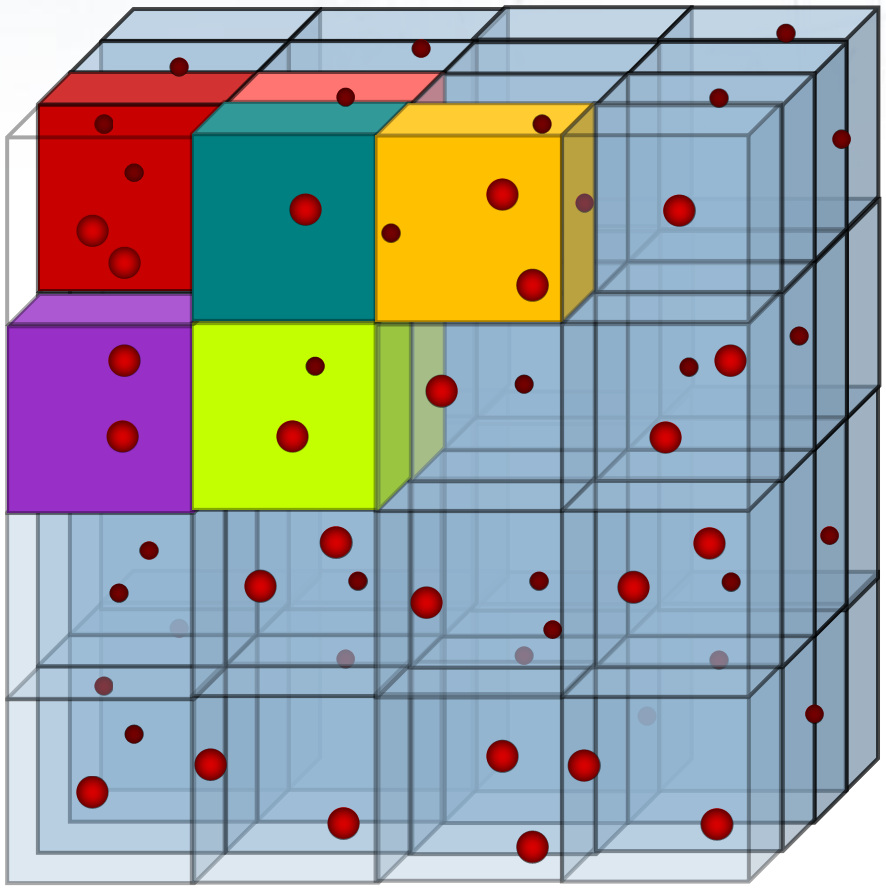
Dynamic Partitioning (Work-Stealing)



- for each frame
  - move particles to correct cell
  - calculate cell density
  - calculate particle forces
  - calculate particles position
  - render frame

Microsoft Research

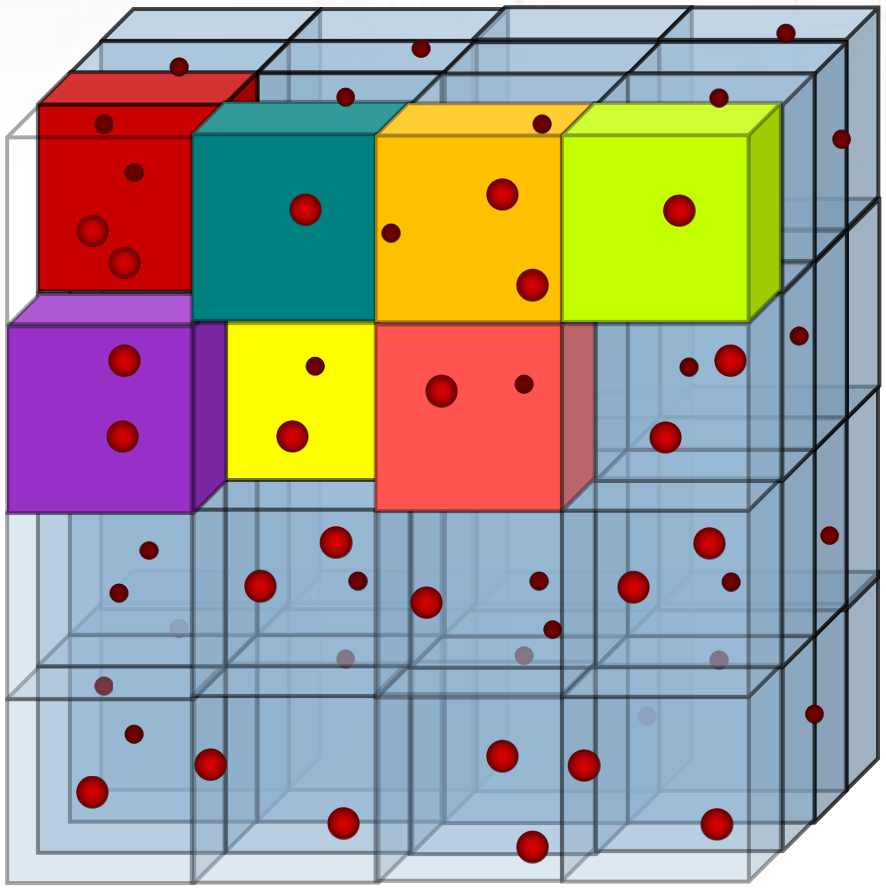
Dynamic Partitioning (Work-Stealing)



- for each frame
  - move particles to correct cell
  - calculate cell density
  - calculate particle forces
  - calculate particles position
  - render frame

Microsoft Research

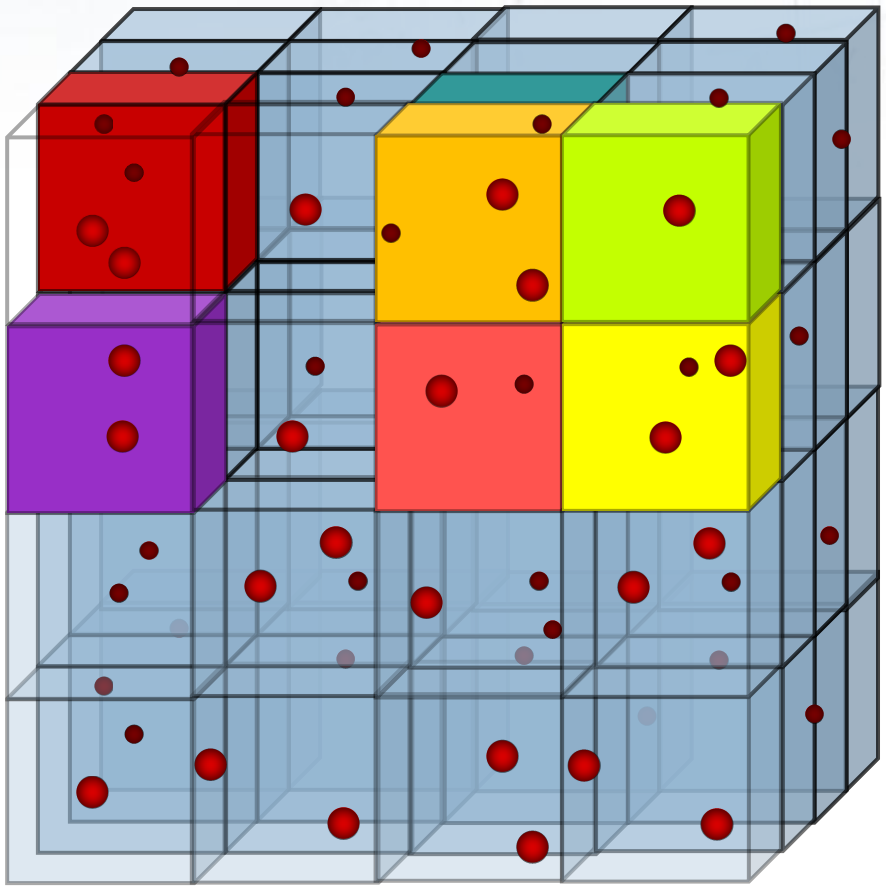
Dynamic Partitioning (Work-Stealing)



- for each frame
  - move particles to correct cell
  - calculate cell density
  - calculate particle forces
  - calculate particles position
  - render frame

Microsoft Research

Dynamic Partitioning (Work-Stealing)

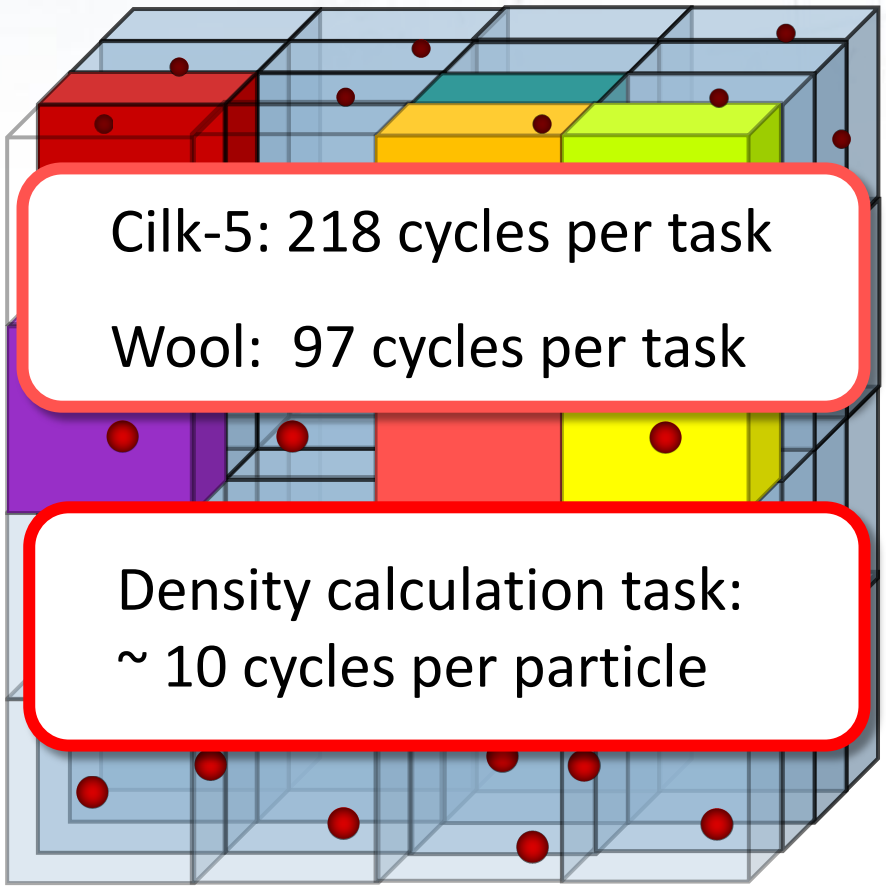


- for each frame
  - move particles to correct cell
  - calculate cell density
  - calculate particle forces
  - calculate particles position
  - render frame

Microsoft Research

Dynamic Partitioning (Work-Stealing)

- Problem: Spawn / Sync Overhead

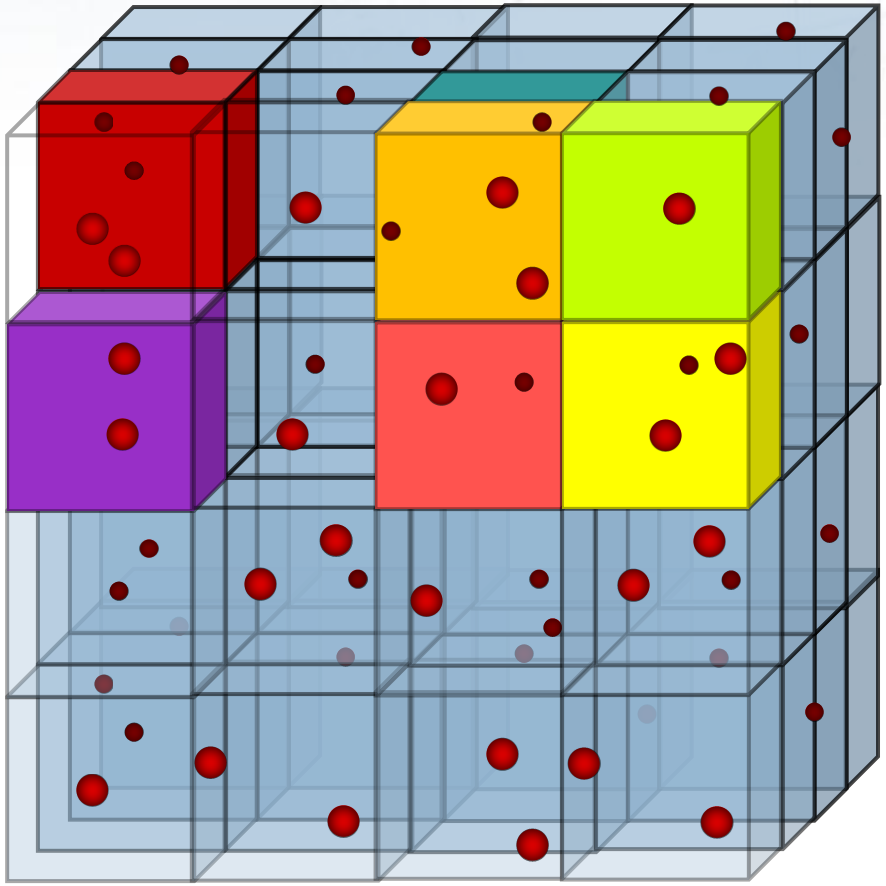


- for each frame
  - move particles to correct cell
  - calculate cell density
  - calculate particle forces
  - calculate particles position
  - render frame

Microsoft Research

Dynamic Partitioning (Work-Stealing)

- Problem: Cache Locality

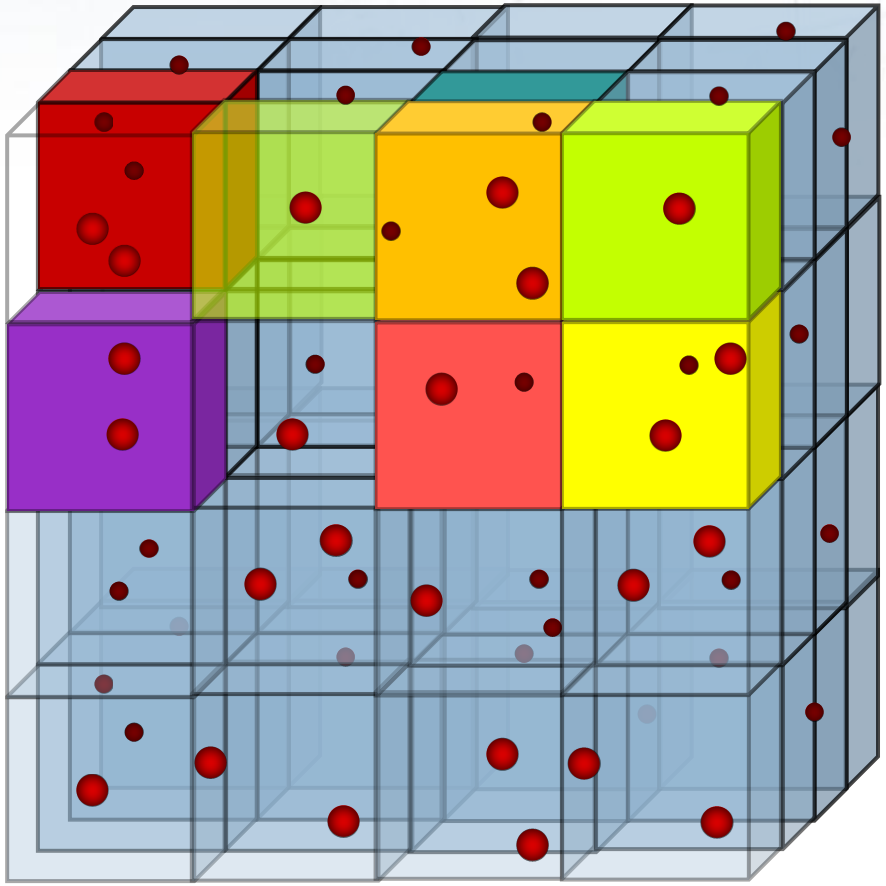


- for each frame
  - move particles to correct cell
  - calculate cell density
  - calculate particle forces
  - calculate particles position
  - render frame

Microsoft Research

Dynamic Partitioning (Work-Stealing)

- Problem: Cache Locality

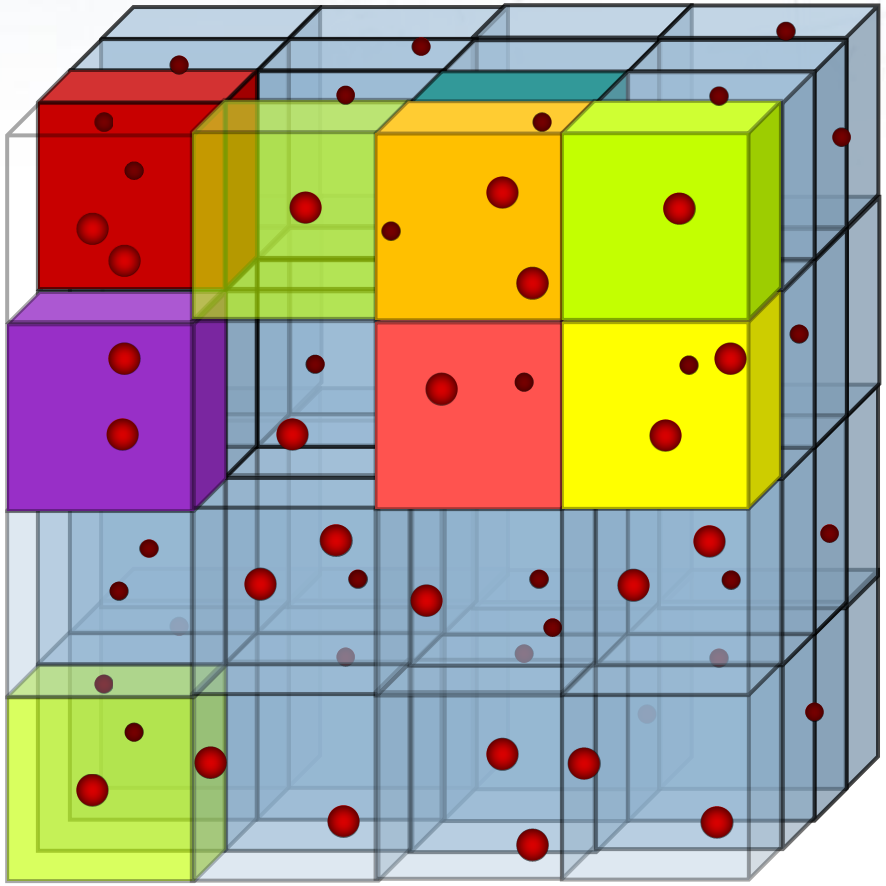


- for each frame
  - move particles to correct cell
  - calculate cell density
  - calculate particle forces
  - calculate particles position
  - render frame

Microsoft Research

Dynamic Partitioning (Work-Stealing)

- Problem: Cache Locality

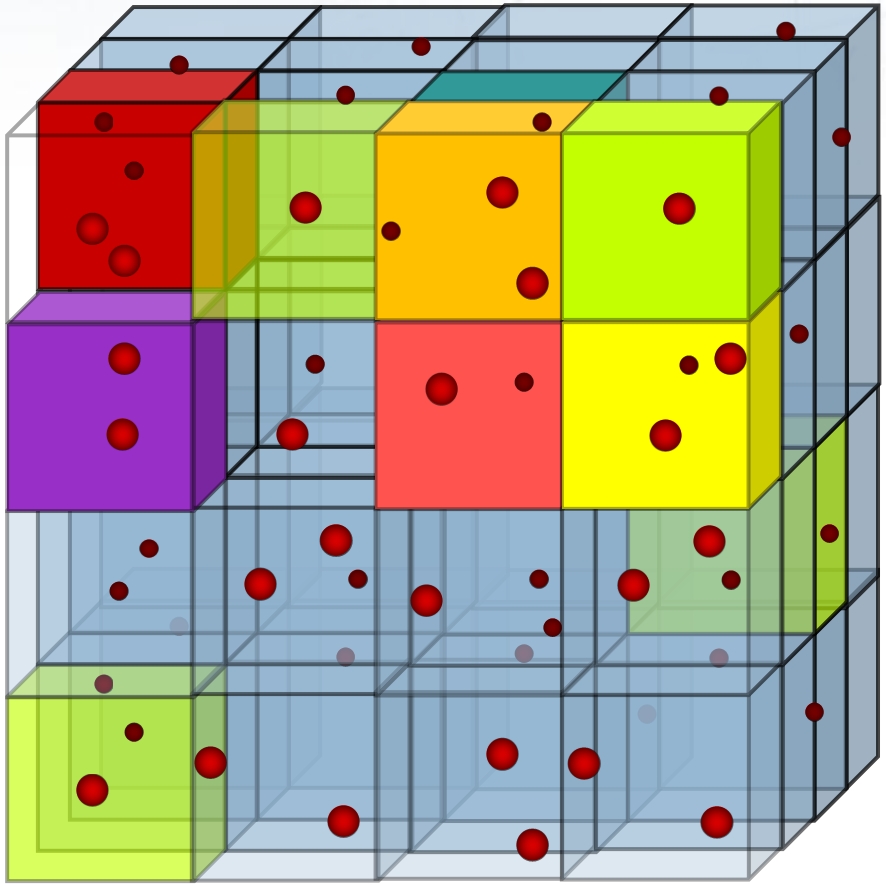


- for each frame
  - move particles to correct cell
  - calculate cell density
  - calculate particle forces
  - calculate particles position
  - render frame

Microsoft Research

Dynamic Partitioning (Work-Stealing)

- Problem: Cache Locality

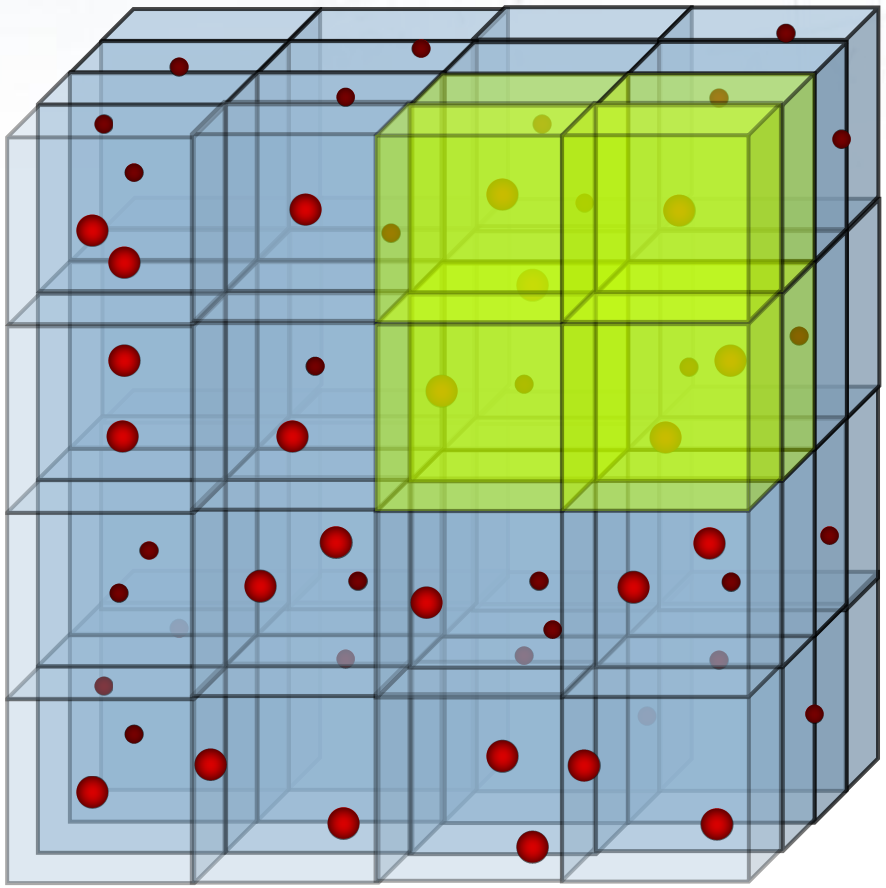


- for each frame
  - move particles to correct cell
  - calculate cell density
  - calculate particle forces
  - calculate particles position
  - render frame

Microsoft Research

Dynamic Partitioning (Work-Stealing)

- Problem: Cache Locality

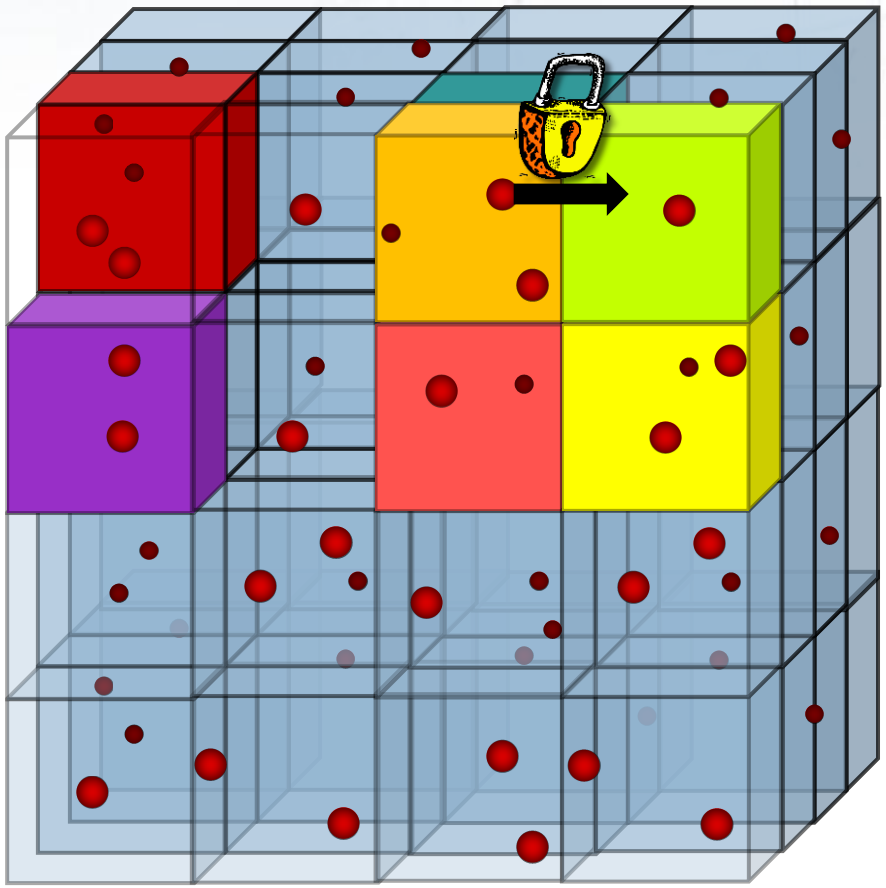


- for each frame
  - move particles to correct cell
  - calculate cell density
  - calculate particle forces
  - calculate particles position
  - render frame

Microsoft Research

Dynamic Partitioning (Work-Stealing)

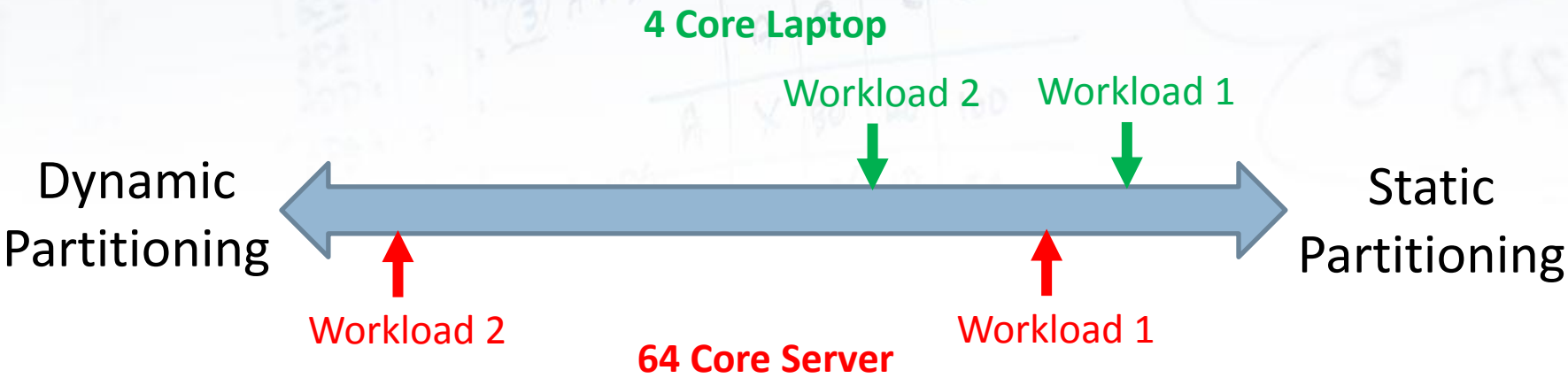
- Problem: Data Synchronization



- for each frame
  - move particles to correct cell
  - calculate cell density
  - calculate particle forces
  - calculate particles position
  - render frame

Microsoft Research

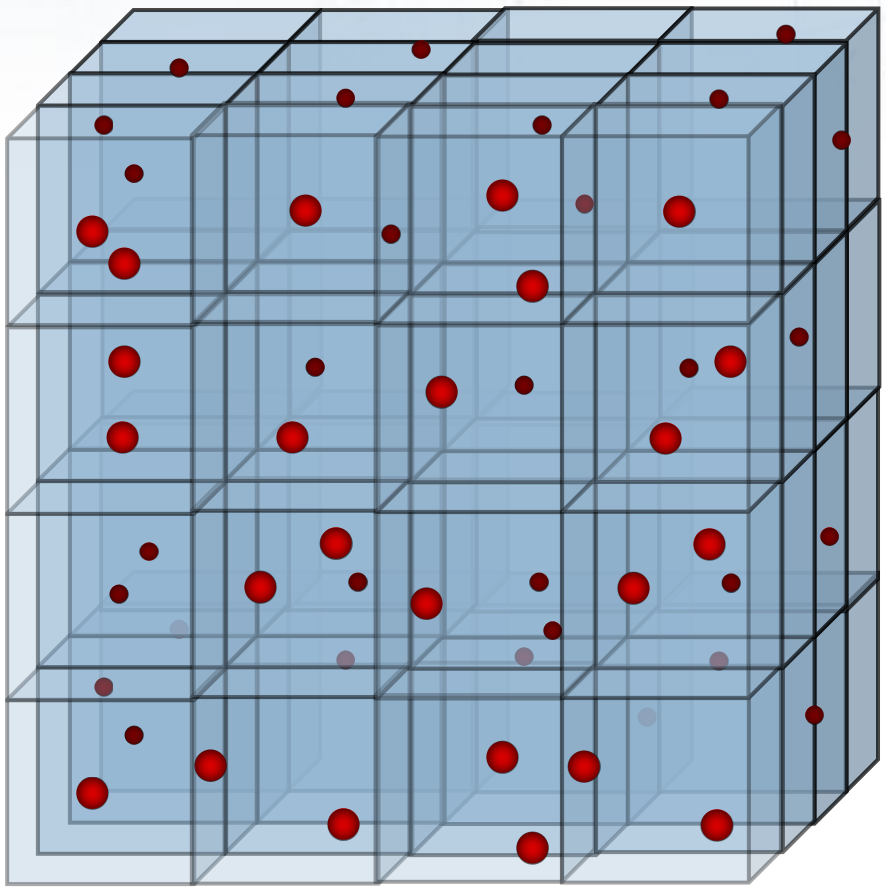
Space-Time Continuum



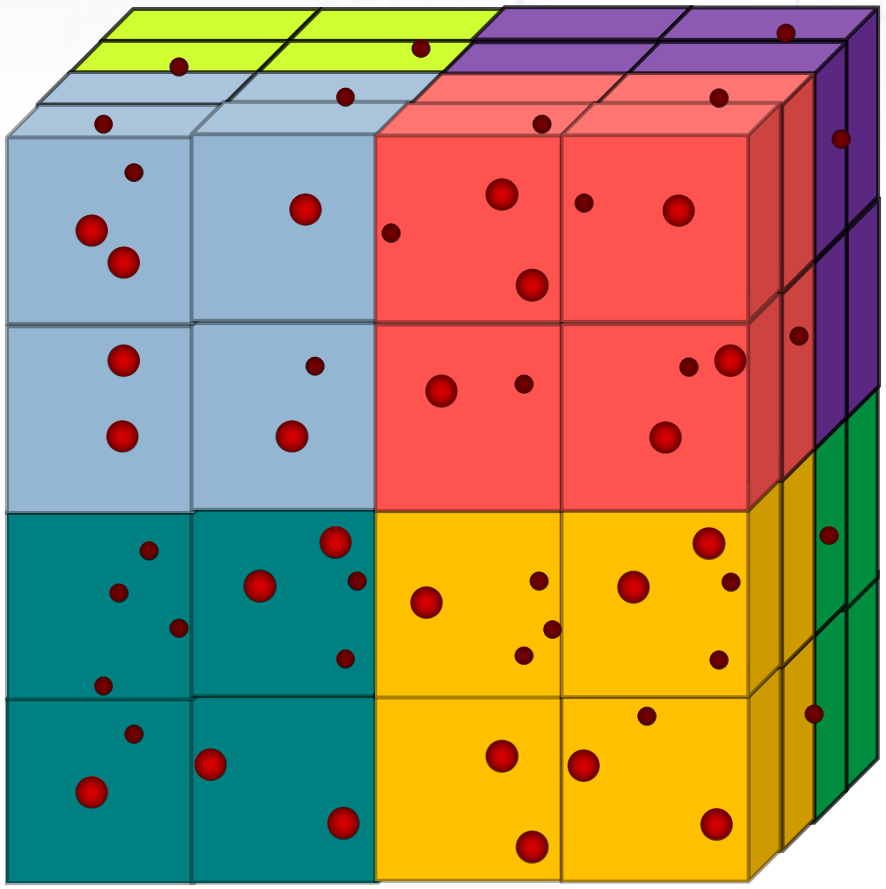
- **Disciplined partitioning** programming model
  - Flexible enough to enable movement on this continuum dynamically **at runtime**
  - Runtime system controls where
    - Is enough work for available cores? If not, repartition work
  - Application controls how
    - Parameterise how data is partitioned
    - Decide whether data-synchronisation is necessary

Microsoft Research

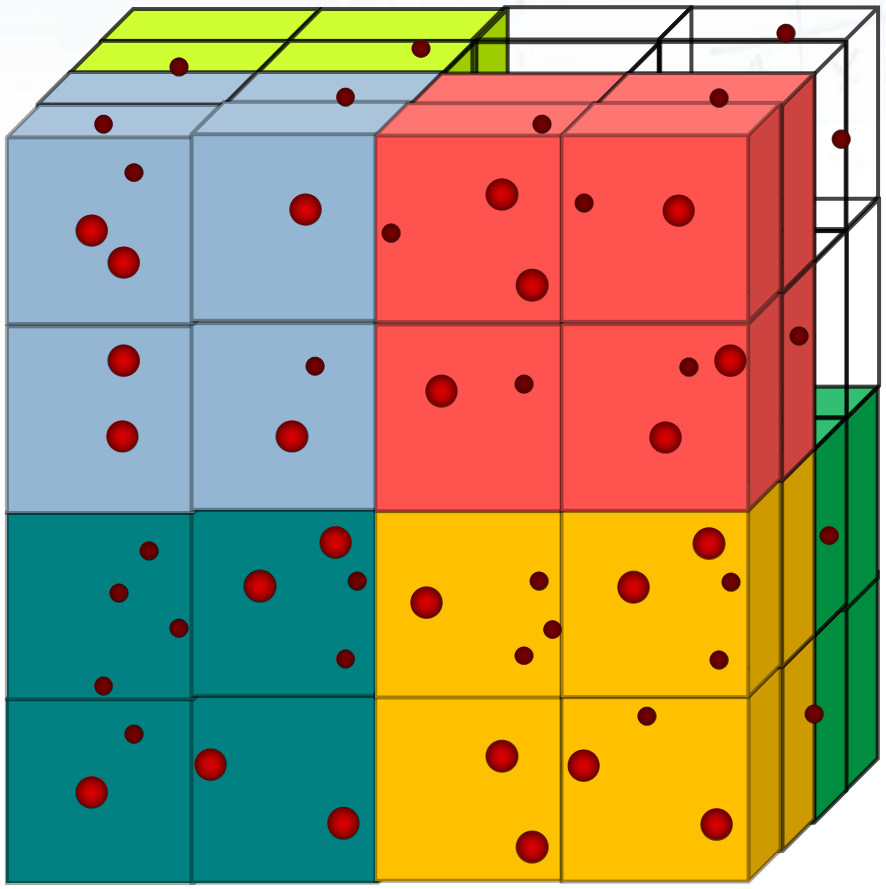
Disciplined Partitioning



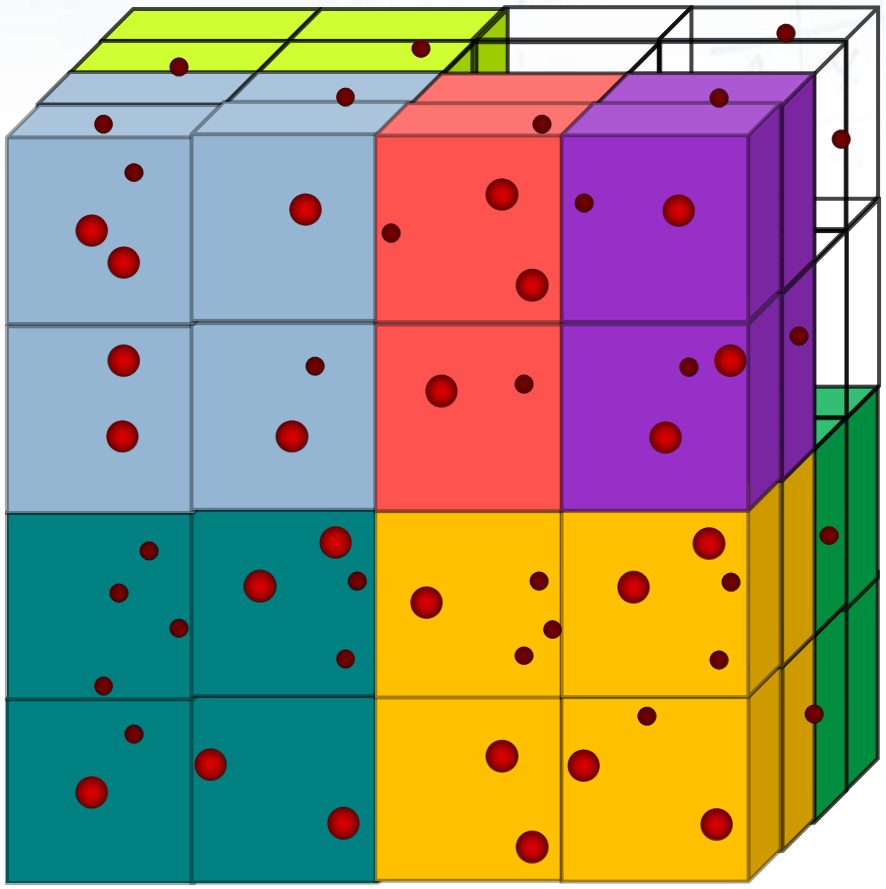
Disciplined Partitioning



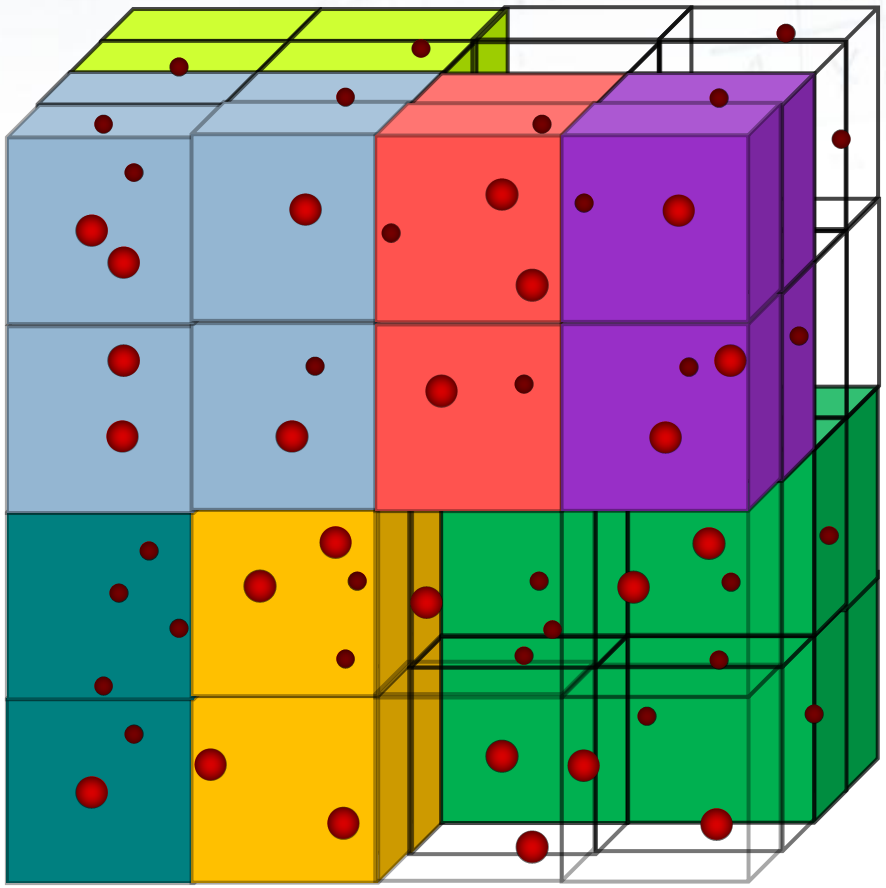
Disciplined Partitioning



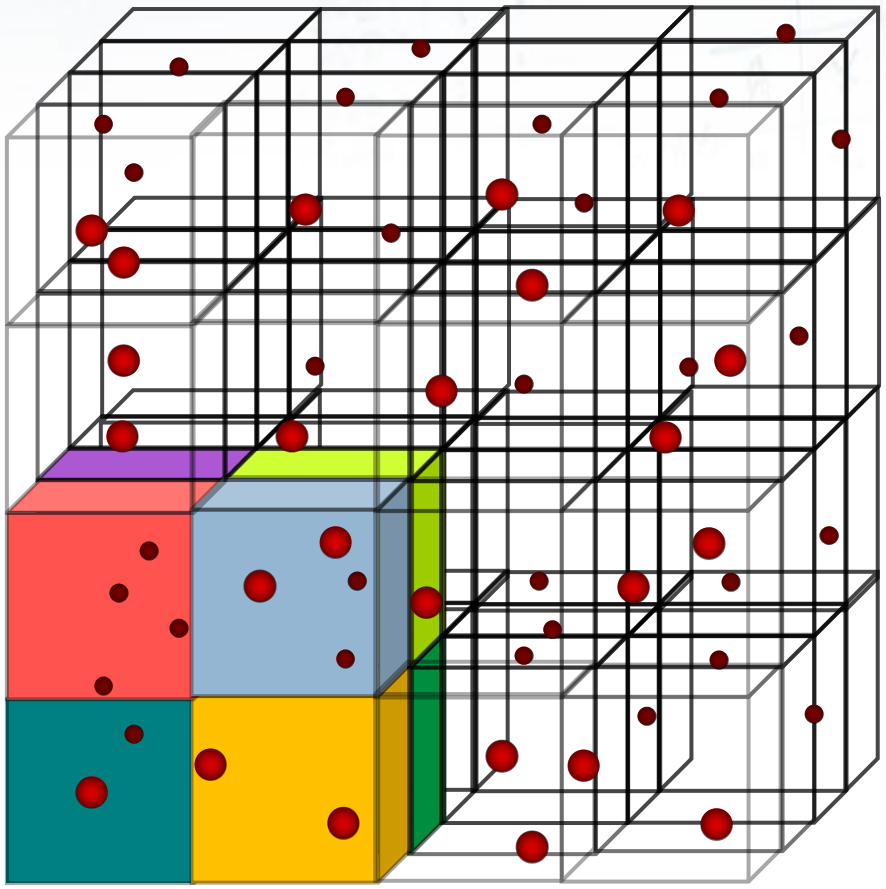
Disciplined Partitioning



Disciplined Partitioning



Disciplined Partitioning



FluidAnimate

```
void computeForces(cell_t [][][] cells, dimentions_t d) {
   range_t range= { .x_start=0, .x_curr=0, .x_end=d.x_len, ...};
   do {
      par fluidAnimate (computeForces, cells, range);
   } finish;
}
par_task fluidAnimate {
  task computeForces(cell_t cell) {
    for (particle in cell) {
        struct cell_t [] ncells = getNeighbours(cell);
        particle.force = calcForce(particle, ncells);
  }}
  range_t [] subdivide(range_t curr_cells, int num) {
     // subdivide curr into num equal cubes, and add to new
  }
  cell_t getNext(cells_t [][][] cells, range_t range) {
     // return next cell in cells, or NULL if finished
}}
```

Microsoft Research

FluidAnimate

```
par_task fluidAnimate {
  task computeForces(cell_t cell) {
    for (particle in cell) {
        struct cell_t [] ncells = getNeighbours(cell);
        particle.force = calcForce(particle, ncells);
    }
  }
  range_t [] subdivide(range_t curr_cells, int num) {
     // subdivide curr into num equal cubes, and add to new
  }
  cell_t getNext(cells_t [][][] cells, range_t range) {
     // return next cell in cells, or NULL if finished
}}
```

Microsoft Research

FluidAnimate

```
TASK(computeForces, fluidAnimate, range_t, cells_t [][][], cell_t cell, ({
    for (particle in cell) {
        struct cell_t [] ncells = getNeighbours(cell);
        particle.force = calcForce(particle, ncells);
    }
}));
SUBDIVIDE(fluidAnimate, range_t, ({
     // subdivide curr into num equal cubes, and add to new
}));
GETNEXT(fluidAnimate, cell_t [][][], cells_t, range_t, ({
     // return next cell in cells, or NULL if finished
}));
```

Microsoft Research

FluidAnimate

Par-task Type

Task Name

Task Args

```
TASK(computeForces, fluidAnimate, range_t, cells_t [][][], cell_t cell, ({
    for (particle in cell) {
        [] ncells = getNeighbours(cell);
        particle.force = calcForce(particle, ncells);
    }
}));
SUBDIVIDE(fluidAnimate, range_t, ({
    // subdivide curr into num equal cubes, and add to new
}));
GETNEXT(fluidAnimate, cells_t, range_t, cell_t [][][], ({
    // return next cell in cells, or NULL if finished
}));
```

Microsoft Research

FluidAnimate

```
TASK(computeForces, fluidAnimate, range_t, cells_t [][][], cell_t cell, ({
    for (particle in cell) {
        struct cell_t [] ncells = getNeighbours(cell);
        particle.force = calcForce(particle, ncells);
    }
}));
SUBDIVIDE(fluidAnimate, range_t, ({
     // subdivide curr into num equal cubes, and add to new
}));
GETNEXT(fluidAnimate, cells_t, range_t, cell_t [][][], ({
     // return next cell in cells, or NULL if finished
}));
```

FluidAnimate

```
TASK(computeForces, fluidAnimate, range_t, cells_t [][][], cell_t cell, ({
    for (particle in cell) {
        struct cell_t [] ncells = getNeighbours(cell);
        particle.force = calcForce(particle, ncells);
    }
}));
range_t[] __fluidAnimate_subdivide(range_t curr_cells, int num) {
     // subdivide curr into num equal cubes, and add to new
}
GETNEXT(fluidAnimate, cells_t, range_t, cell_t [][][], ({
     // return next cell in cells, or NULL if finished
}));
```

Microsoft Research

FluidAnimate

```
TASK(computeForces, fluidAnimate, range_t, cells_t [][][], cell_t cell, ({
    for (particle in cell) {
        struct cell_t [] ncells = getNeighbours(cell);
        particle.force = calcForce(particle, ncells);
    }
}));
range_t[] __fluidAnimate_subdivide(range_t curr_cells, int num) {
     // subdivide curr into num equal cubes, and add to new
}
cell_t __fluidAnimate_getNext(cells_t [][][] cells, range_t range) {
     // return next cell in cells, or NULL if finished
}
```

Microsoft Research

FluidAnimate

```
TASK(computeForces, fluidAnimate, range_t, cells_t [][][], cell_t cell, ({
    for (particle in cell) {
        struct cell_t [] ncells = getNeighbours(cell);
        particle.force = calcForce(particle, ncells);
    }
}));
range_t[] __fluidAnimate_subdivide(range_t curr_cells, int num) {
     // subdivide curr into num equal cubes, and add to new
}
cell_t __fluidAnimate_getNext(cells_t [][][] cells, range_t range) {
     // return next cell in cells, or NULL if finished
}
```

Microsoft Research

FluidAnimate

```
TASK(computeForces, fluidAnimate, range_t, cells_t [][][], cell_t cell, ({
    for (particle in cell) {
        struct cell_t [] ncells = getNeighbours(cell);
        particle.force = calcForce(particle, ncells);
    }
}));
range_t [] __fluidAnimate_subdivide(range_t curr_cells, int num) {
     // subdivide range into num equal cubes, and return new
}
cell_t __fluidAnimate_getNext(cells_t [][][] cells, range_t range) {
     // return next cell in range, or NULL if finished
}
```

```
void __computeForces_task(range_t my_range, cells_t [][][] cells) {
  cell_t cell = __fluidAnimate_getNext(cells, my_range);
  do {
    for (particle in cell) {
        struct cell_t [] ncells = getNeighbours(cell);
        particle.force = calcForce(particle, ncells);
    }
    if ((int num = calico_should_subdivide()) > 0) {
      range_t [] new_ranges = __fluidAnimate_subdivide(my_range, num);
      calico_schedule_par(__computeForces_task, new_ranges, cells);
       return;
  } while ((cell = __fluidAnimate_getNext(cells, my_range)) != NULL);
}
```

FluidAnimate

```
void __computeForces_task(range_t my_range, cells_t [][][] cells) {
  cell_t cell = __fluidAnimate_getNext(cells, my_range);
  do {
    for (particle in cell) {
        struct cell_t [] ncells = g
        particle.force = calcForce(
    }
    if ((int num = calico_should_subdi
      range_t[] new_ranges = __fluidAn
      calico_schedule_par(__computeForces_task, new_ranges, cells);
      return;
```

Automatic Repartitioning when necessary

Aggregation of multiple task iterations

```
void computeForces(cell_t [][][]cells, dimentions_t d) {
   range_t range= { .x_start=0, .x_curr=0, .x_end=d.x_len, ...};
   do {
      par_fluidAnimate (computeForces, cells, range);
   } finish;
}

}
```

Microsoft Research

FluidAnimate

```
par_task fluidAnimate {

  task moveParticles(cell_t cell)    {   ...   }
  task computeDensities(cell_t cell) {   ...   }
  task computeForces(cell_t cell)    {   ...   }
  task renderCell(cell_t cell)       {   ...   }

  range_t [] subdivide(range_t curr_cells, int num) {
     // subdivide curr into num equal cubes, and add to new
  }
  cell_t getNext(cells_t [][][] cells, range_t range) {
     // return next cell in cells, or NULL if finished
  }
  bool calcOnDifferentCore(cell_t cell, range_t range) {
     // return true if cell is not within range
}}
```

Microsoft Research

FluidAnimate

```
par_task fluidAnimate {
  task moveParticles(cell_t cell)    {
    for (particle in cell) {
      cell_t new_cell = calculateParticlesCell(particle);
      if (new_cell == cell) continue;
      if (onDifferentCore(new_cell)) {
        lockAndUpdate(new_cell, particle);
      } else {
        updateNoLock(new_cell, particle);
  }}}
  ...
  bool calcOnDifferentCore(cell_t cell, range_t range) {
     // return true if cell is not within range
  }
  ...
}}
```

Microsoft Research

FluidAnimate

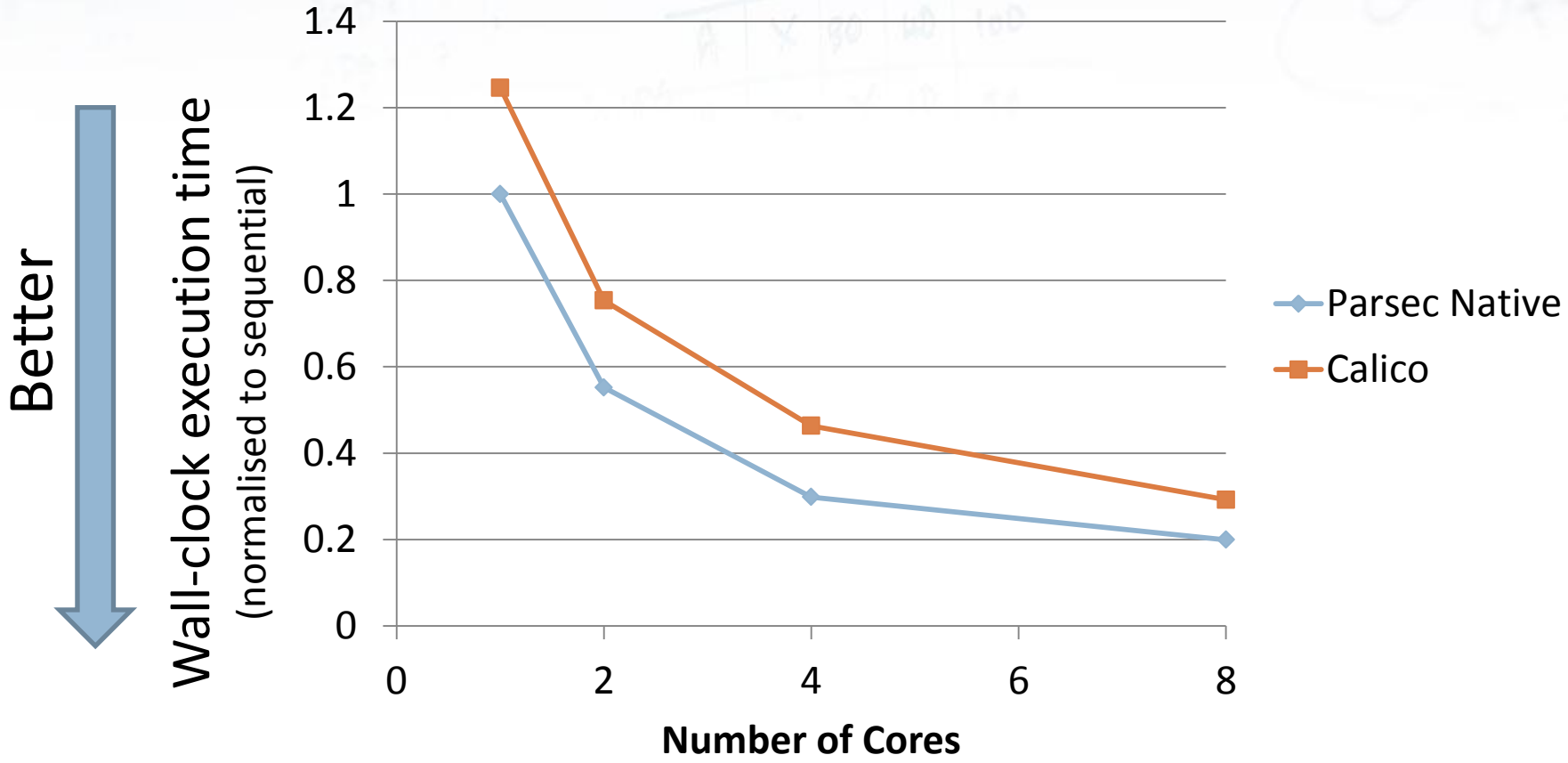
```
par_task fluidAnimate {
  task moveParticles(cell_t cell)    {
    for (particle in cell) {
      cell_t new_cell = calculateParticlesCell(particle);
      if (new_cell == cell) continue;
      if (onDifferentCore(new_cell)) {
        lockAndUpdate(new_cell, particle);
      } e
        up
  }}}
  ...
  bool calcOnDifferentCore(cell_t cell, range_t range) {
     // return true if cell is not within range
  }
  ...
}}
```

```
calcOnDifferentCore(new_cell, my_range);
```

Microsoft Research

FluidAnimate Results

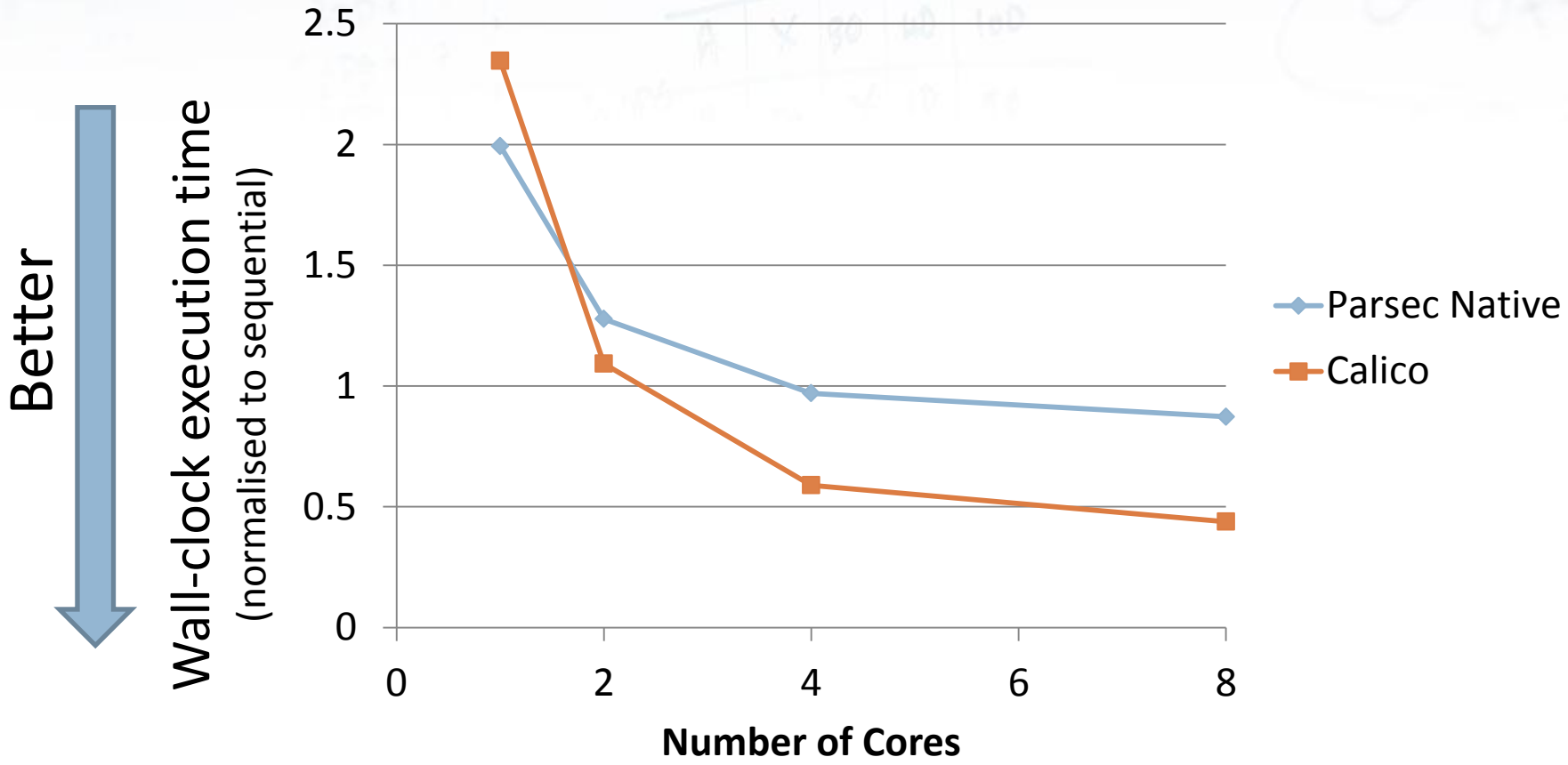
- No competition for CPU-time



Microsoft Research

FluidAnimate Results

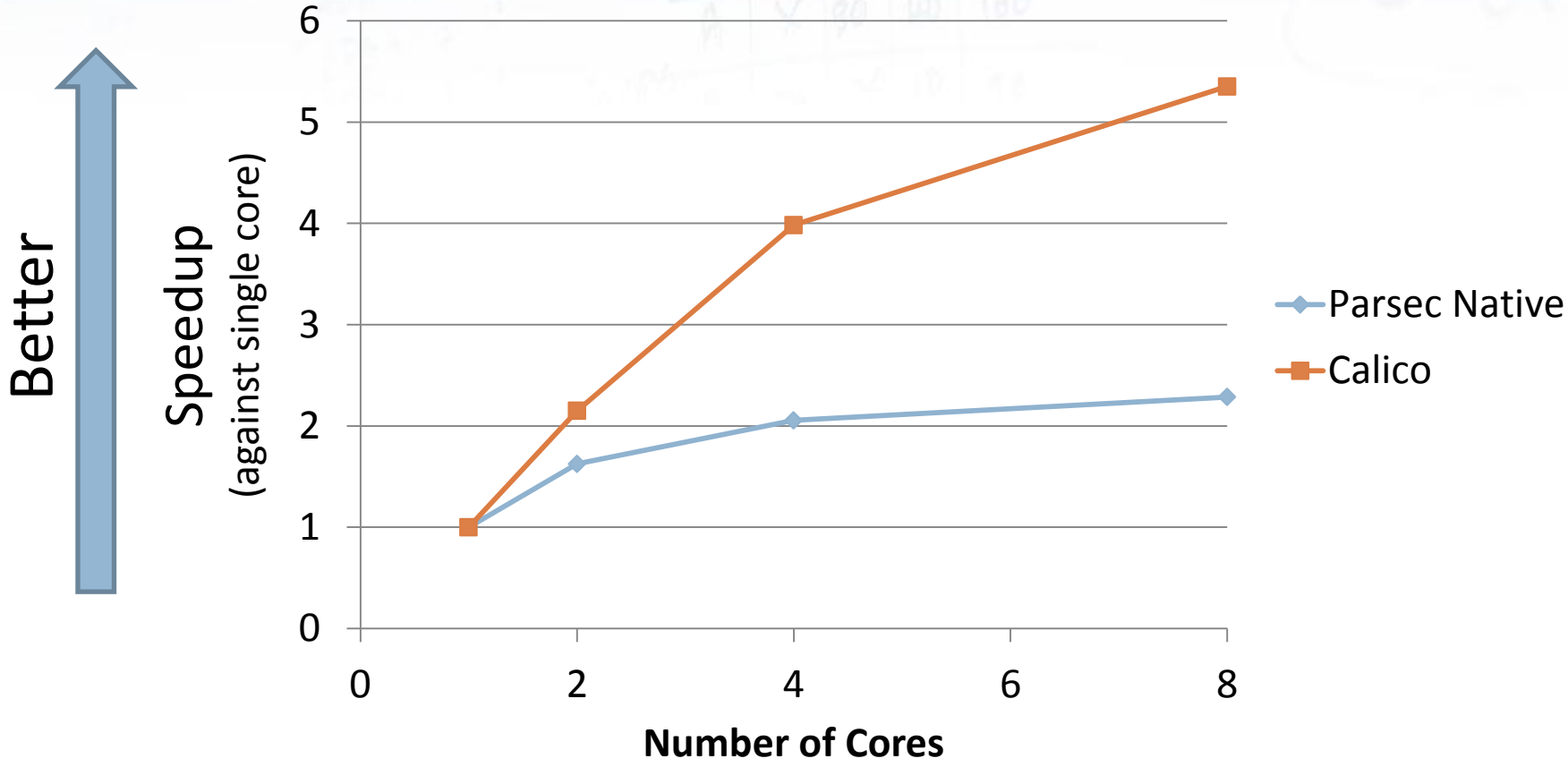
- Competition for CPU-time



Microsoft Research

FluidAnimate Results

- Competition for CPU-time



Microsoft Research

Status

- Supported Benchmarks:
  - Blacksholes
  - FluidAnimate
  - Dedup
  - Delaunay Triangulation (work ongoing)
- Supported on non-cache-coherent hardware
  - Blacksholes
  - FluidAnimate

Future Directions

- OS / Runtime System co-design
  - Blur the OS / Runtime boundary with Calico / Barrelfish
  - Inform the OS of changes in resource demands
  - Inform the runtime of changing core-counts / competing applications
  - Provide architecture details (e.g., cache layout) to runtime
  - Have runtime automatically adapt application behaviour using this information

Microsoft Research

turning ideas into reality.

©2010 Microsoft Corporation. All rights reserved.
This material is provided for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS OR IMPLIED, IN THIS SUMMARY.
Microsoft is a registered trademark or trademark of Microsoft Corporation in the United States and/or other countries.