UiT

THE ARCTIC
UNIVERSITY
OF NORWAY

Lecture 5: Partitioning & Divide-and-Conquer Strategies

Parallell Programming (INF-3201)

University of Tromsø

John Markus Bjørndalen

Outline

- Partitioning
- Divide-and-conquer
- Partitioning and Divide-and-conquer examples
  - Bucket sort
  - Numerical integration
  - N-body problem

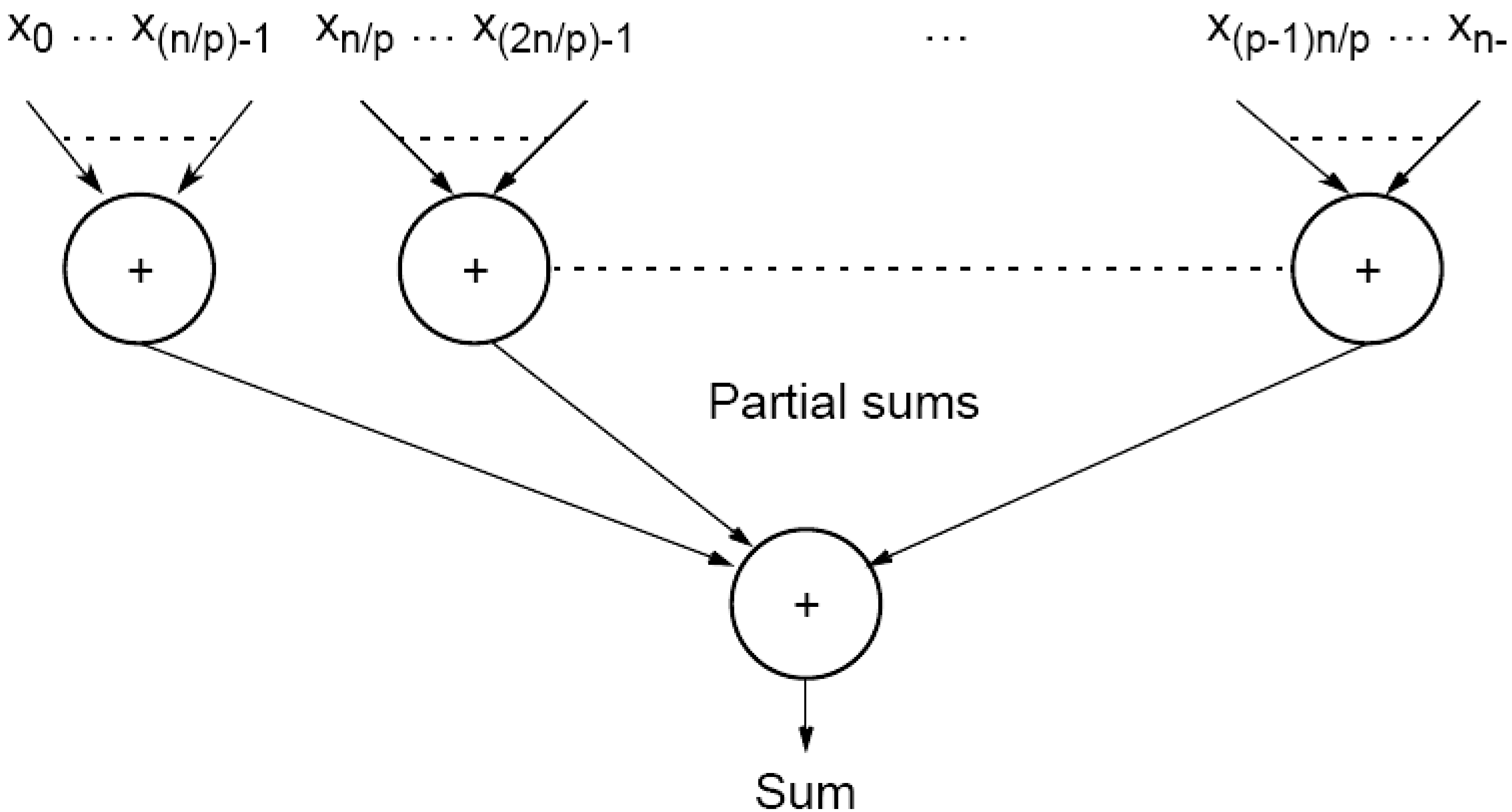
Partitioning

Partitioning simply divides the problem into parts.

Divide and Conquer

Characterized by dividing problem into sub-problems of same form as larger problem. Further divisions into still smaller sub-problems, usually done by recursion.

Partitioning a sequence of numbers into parts and adding the parts



Using point-to-point communication

- **Master**

```
s = n/p;                              // number of numbers for slaves
for (i = 0, x = 0; i < p; i++, x = x + s)
    send(&numbers[x], s, P_i);  // send s numbers to slave
sum = 0;
for (i = 0; i < p; i++) {       // wait for results from slaves
    recv(&part_sum, P_ANY);
    sum += part_sum;            // accumulate partial sums
}
```

- **Slave**

```
recv(&numbers, s, P_master);    // receive s numbers from master
part_sum = 0;
for (i = 0; i < s; i++)         // add numbers
   part_sum = part_sum + numbers[i];
send(&part_sum, P_master);      // send sum to master
```

Using collective communication

- Master

```
s = n/p;                                        // number of numbers
scatter(&numbers, s, P_group, root=master); // send numbers to slaves
sum = 0;
reduce_add(&sum, s, P_group, root=master);  // results from slaves
```

- Slave

```
scatter(&numbers, s, P_group, root=master);     // receive s numbers
part_sum = 0;
for (i = 0; i < s; i++)                          // add numbers
   part_sum = part_sum + numbers[i];
reduce_add(&part_sum, s, P_group, root=master); // send sum to master
```

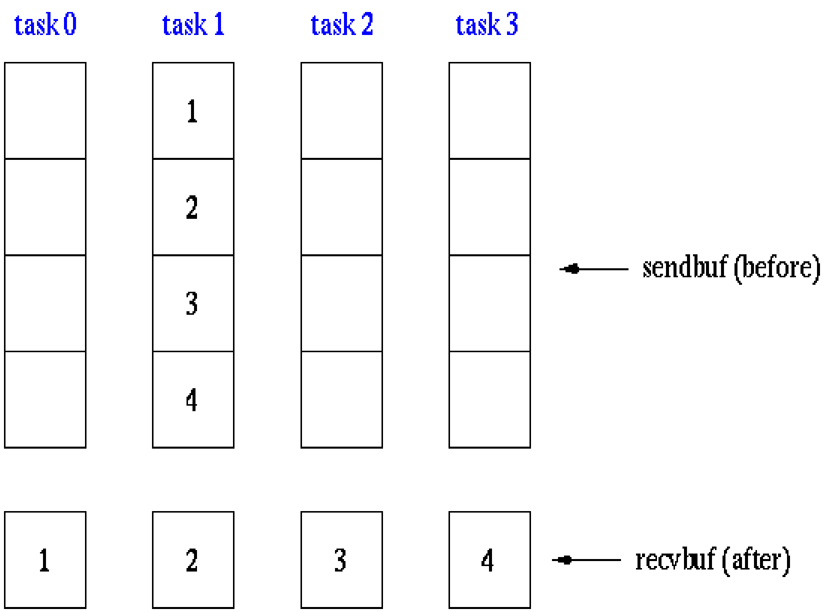
Recall: scatter(), reduce()

MPI_Scatter

Sends data from one task to all other tasks in a group

sendcnt = 1;
recvcnt = 1;
src = 1; task 1 contains the message to be scattered

```
MPI_Scatter(sendbuf, sendcnt, MPI_INT,
            recvbuf, recvcnt, MPI_INT,
            src, MPI_COMM_WORLD);
```



MPI_Reduce

Perform and associate reduction operation across all tasks in the group and place the result in one task

count = 1;
dest = 1; result will be placed in task 1

```
MPI_Reduce(sendbuf, recvbuf, count, MPI_INT, MPI_SUM,
           dest, MPI_COMM_WORLD);
```

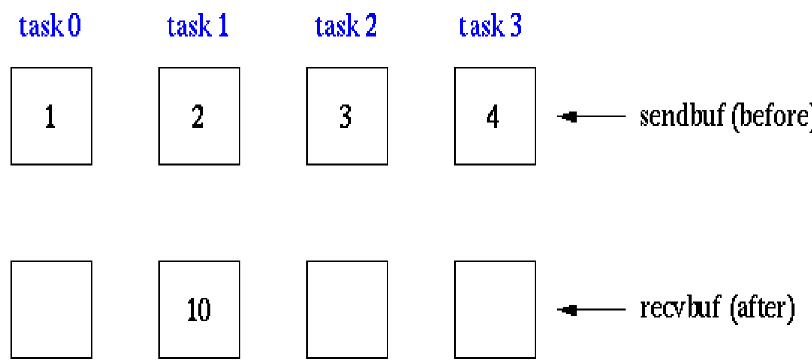


Figure from Blaise Barney, “Message Passing Interface (MPI)”, Livermore Computing

Outline

- Partitioning
- Divide-and-conquer
- Partitioning and Divide-and-conquer examples
  - Bucket sort
  - Numerical integration
  - N-body problem

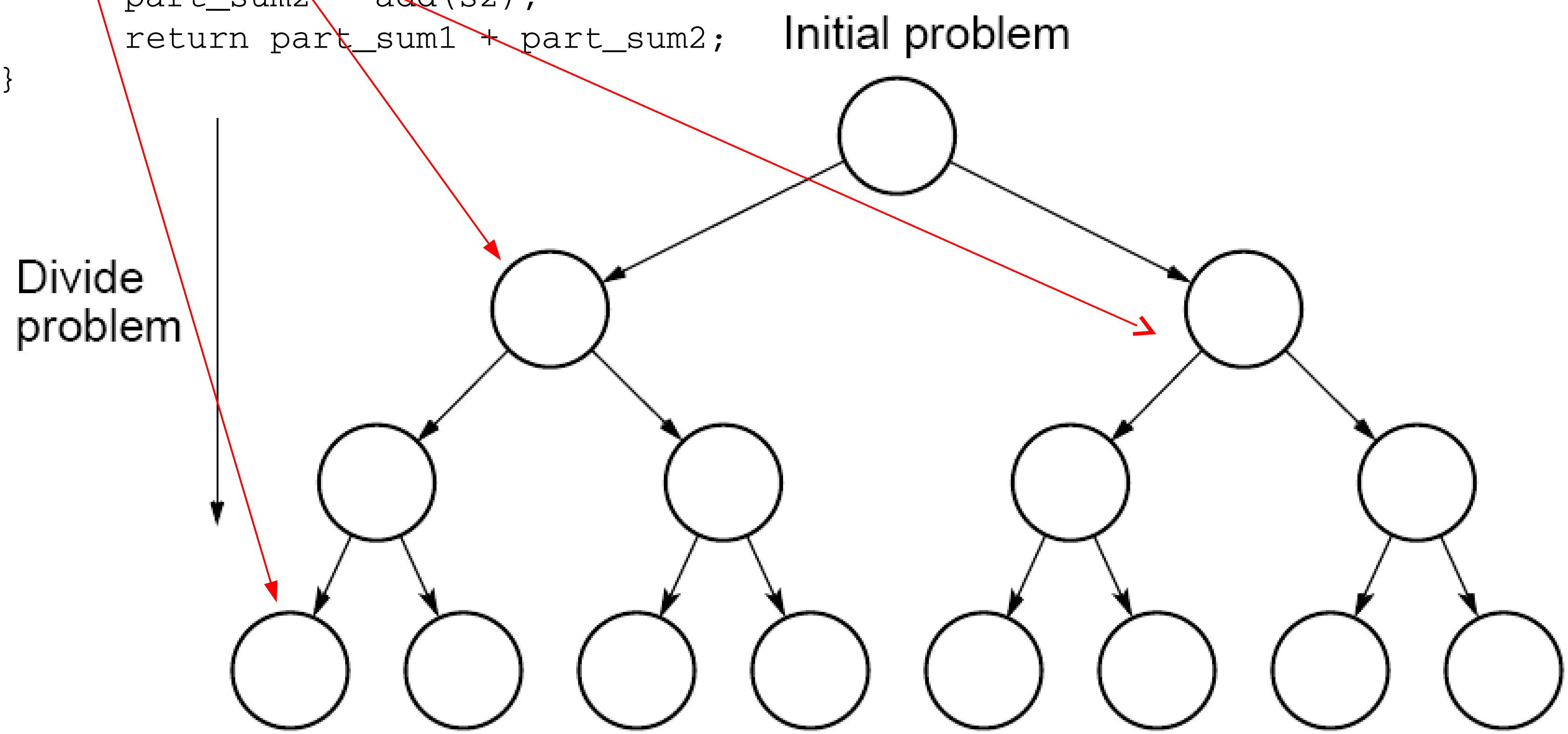
Divide-and-conquer

- Features
  - Recursively divide a problem into sub-problems that are of the same form as the larger problem.
  - Simple sub-problems are performed and results combined.
- Example: sequential recursive addition

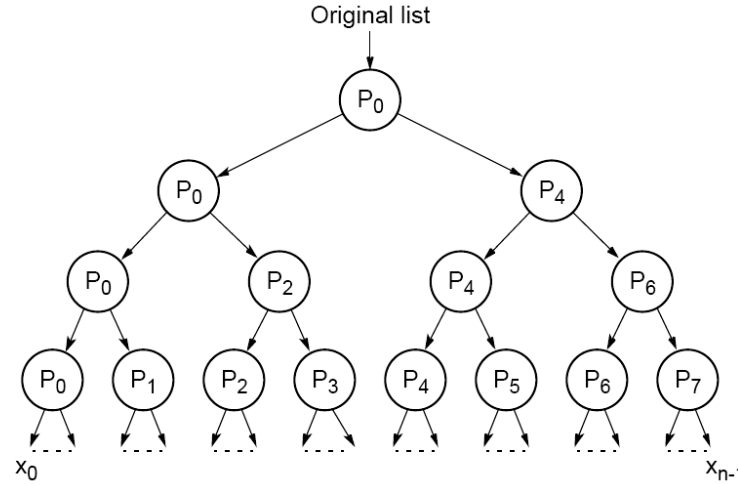
```
int add(int *s)              /* add list of numbers, s */
{
   if (number(s) <= 2)
       return n1 + n2;       /* terminate recursion */
   else {
       divide(s, s1, s2);    /* divide s into two parts, s1 and s2 */
       part_sum1 = add(s1); /*recursive calls to add sub lists */
       part_sum2 = add(s2);
       return part_sum1 + part_sum2;
   }
}
```

Tree construction

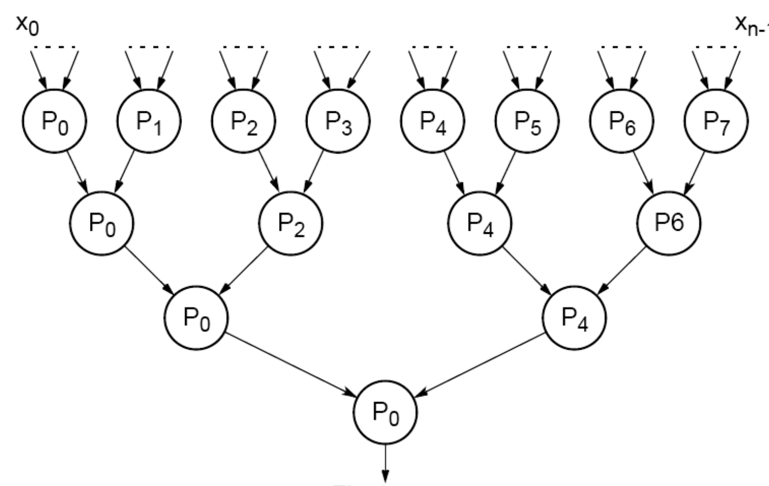
```
int add(int *s)
{
      if (number(s) <= 2)
            return n1 + n2;
      else {
            divide(s, s1, s2);
            part_sum1 = add(s1);
            part_sum2 = add(s2);
            return part_sum1 + part_sum2;
      }
}
```



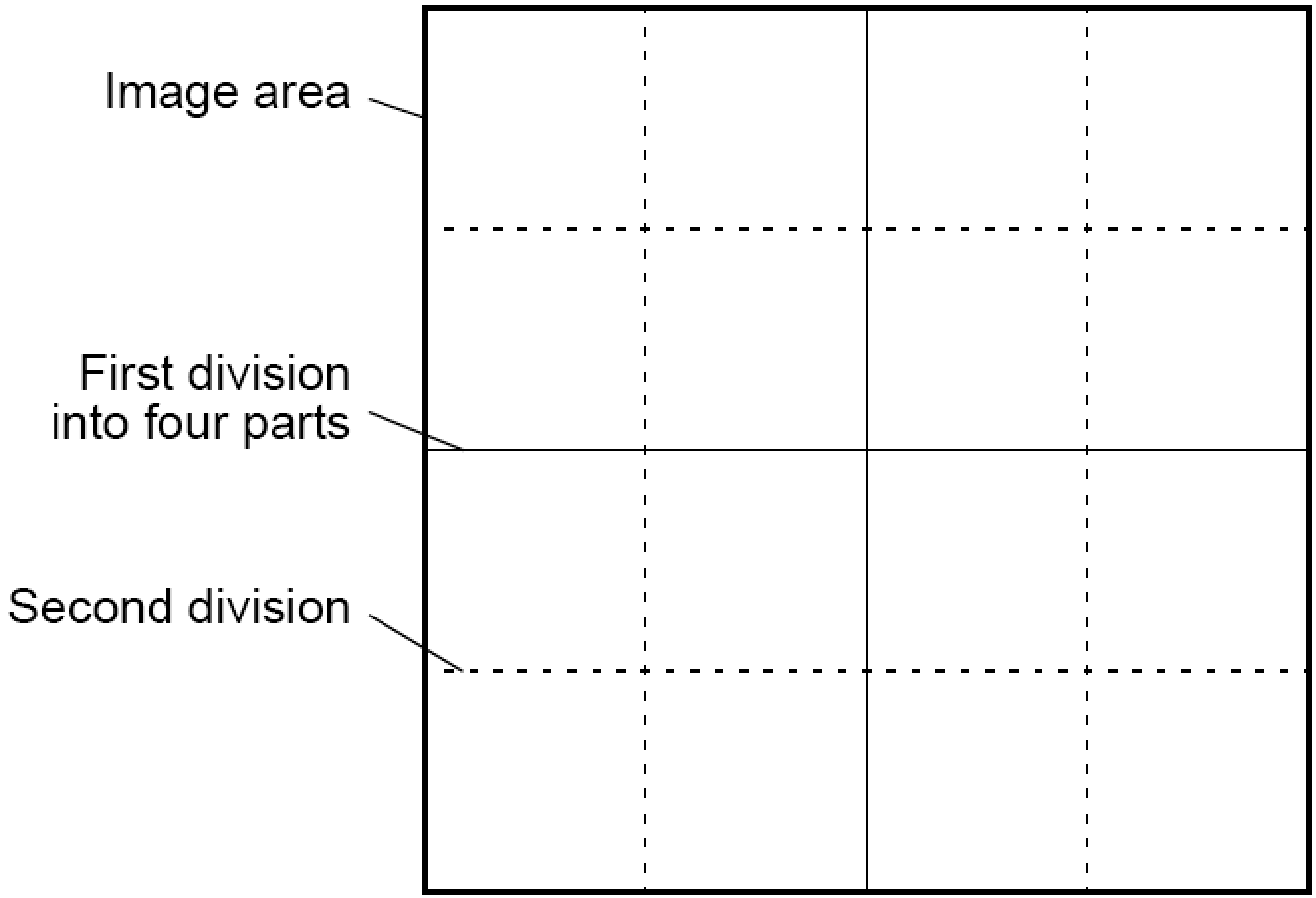
Dividing a list into parts in parallel



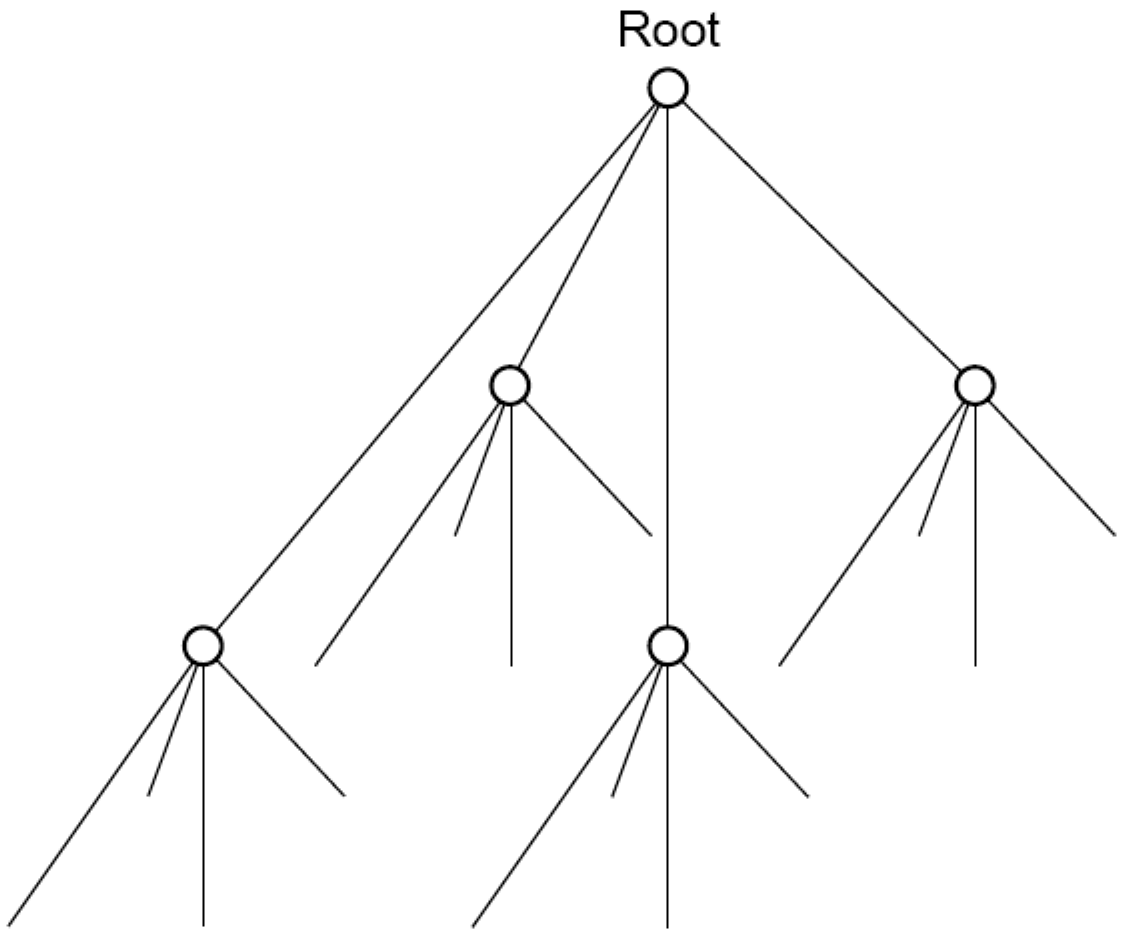
Combining partial sums



Dividing an image



Quadtree

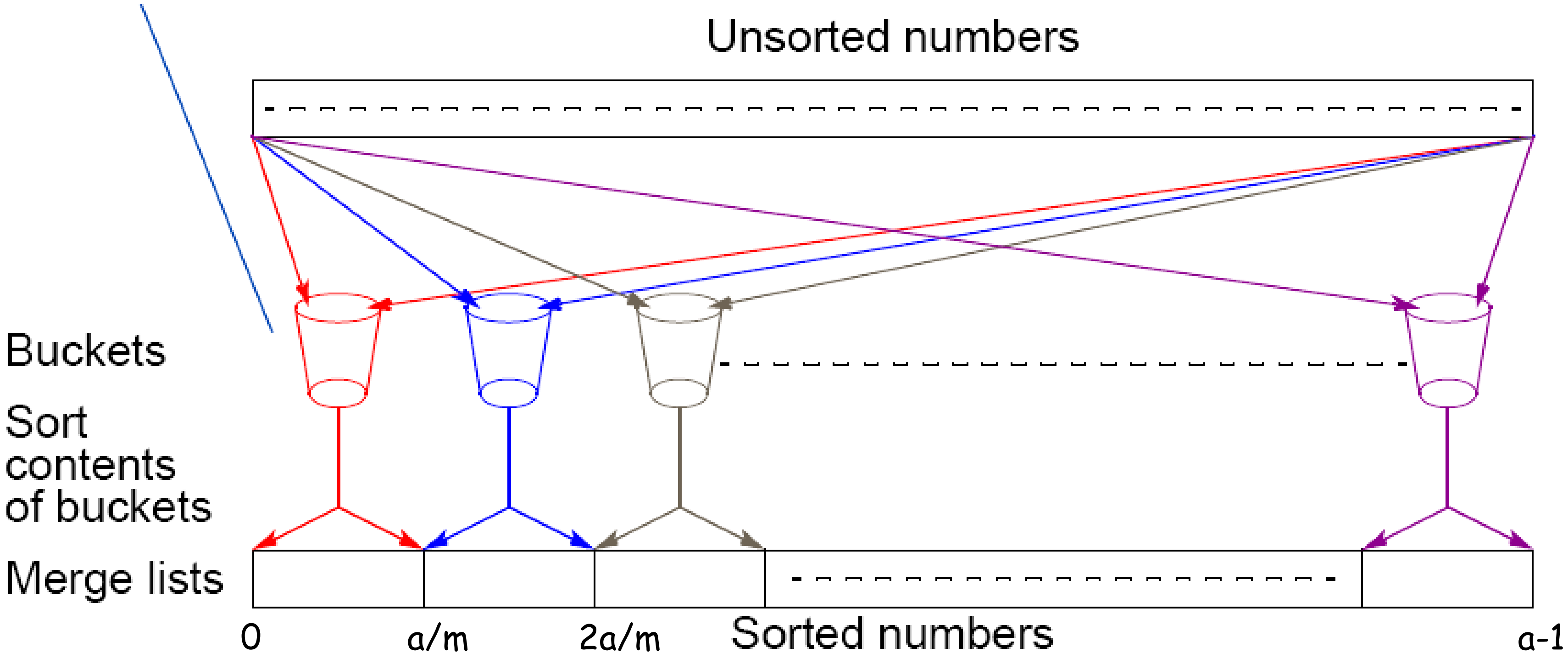


Outline

- Partitioning
- Divide-and-conquer
- Partitioning and Divide-and-conquer examples
  - Bucket sort
  - Numerical integration
  - N-body problem

Bucket sort

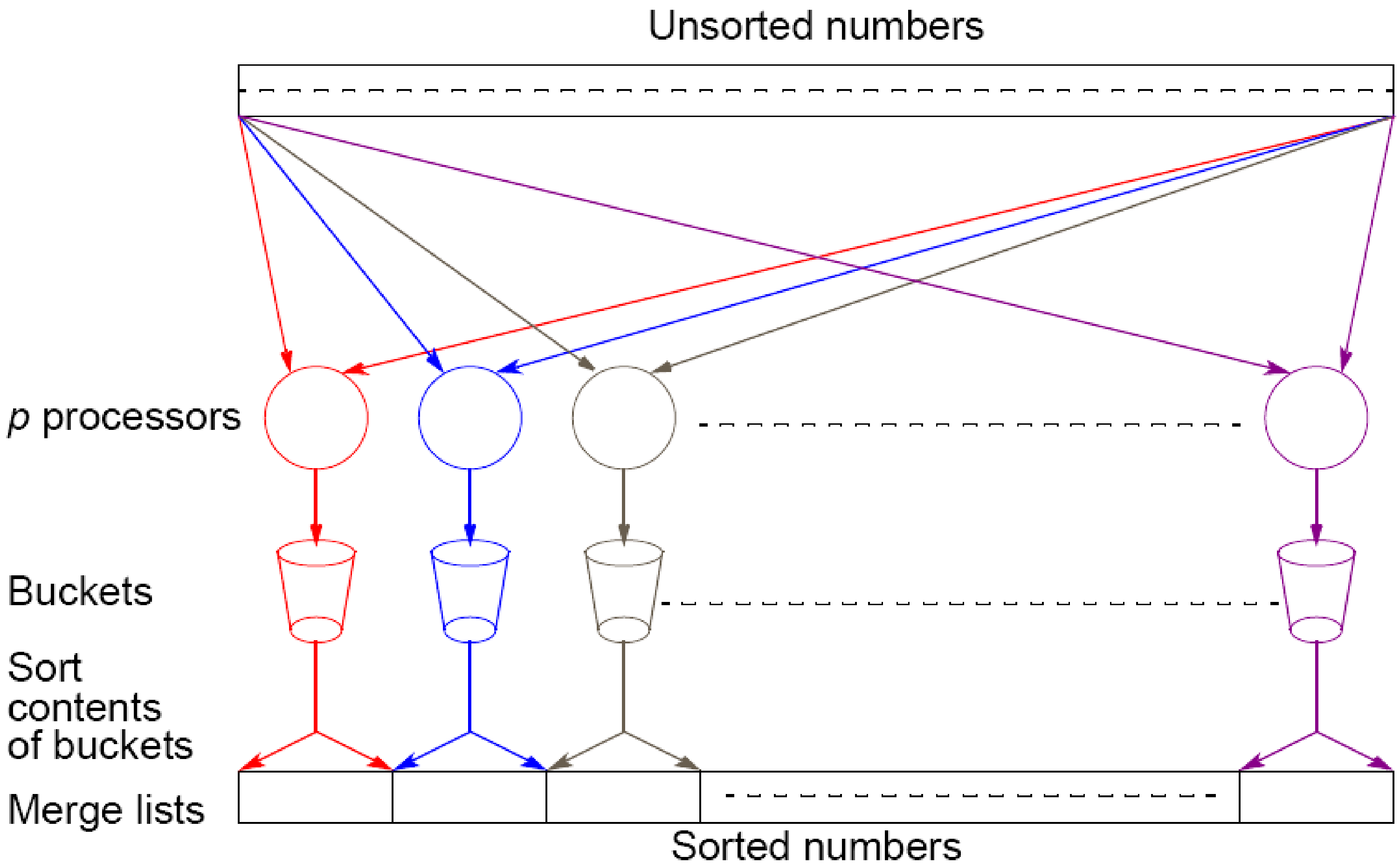
One "bucket" assigned to hold numbers that fall within each of m equal regions. Numbers in each bucket sorted using a sequential sorting algorithm.



Parallel version of bucket sort

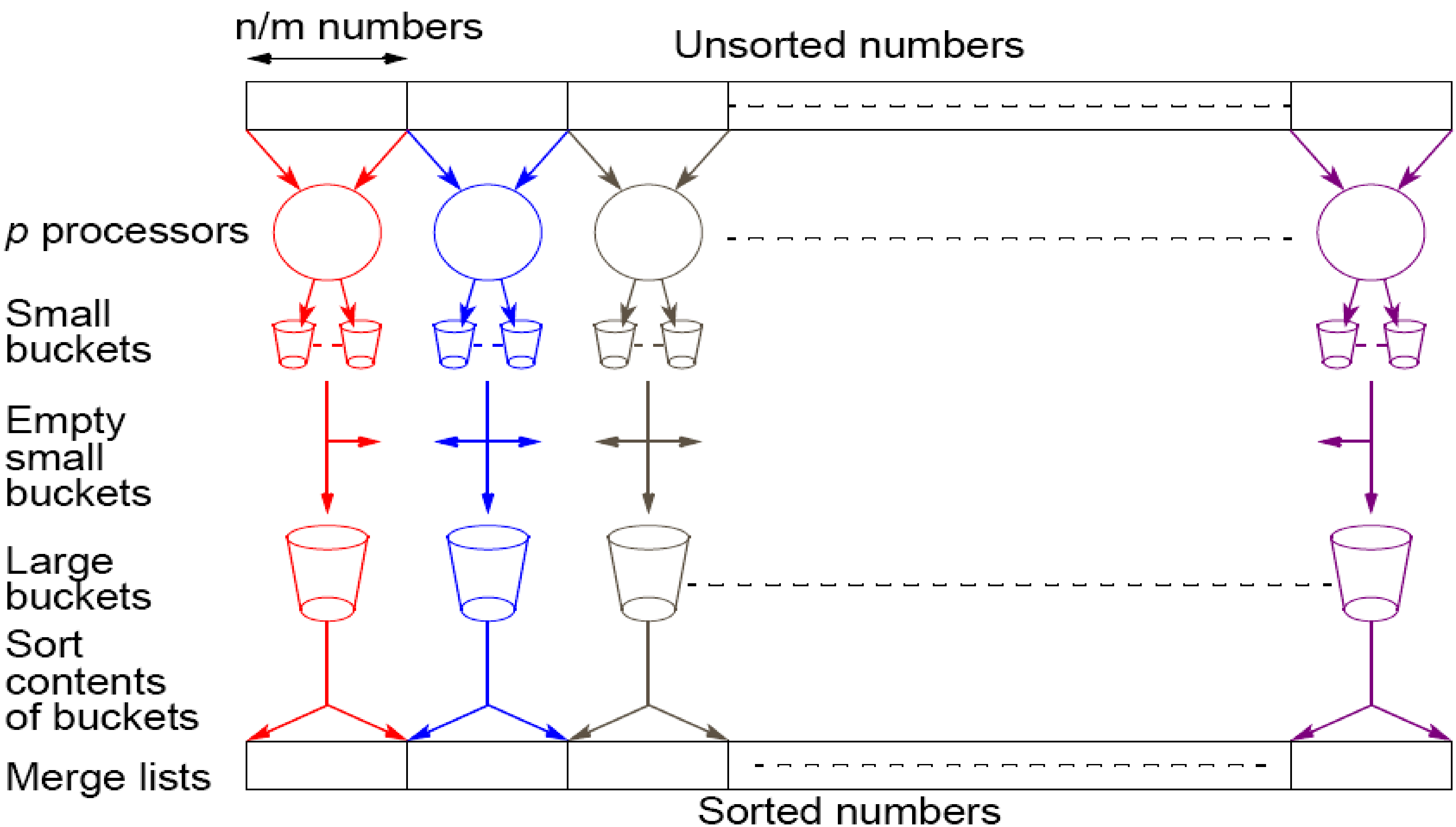
Simple approach

Assign one processor for each bucket.



Another parallel version of bucket sort

Parallel sorting time complexity: $t_p = n/p + (n/p)\log(n/p)$ where $m=p$



Introduces new message-passing operation - all-to-all broadcast.

Another version, page 2

Partition sequence into *m* regions, one region for each processor.

Each processor maintains *p* “small” buckets and separates numbers in its region into its own small buckets.

Small buckets then emptied into *p* final buckets for sorting, which requires each processor to send one small bucket to each of the other processors (bucket *i* to processor *i*).

Recall: MPI_Alltoall()

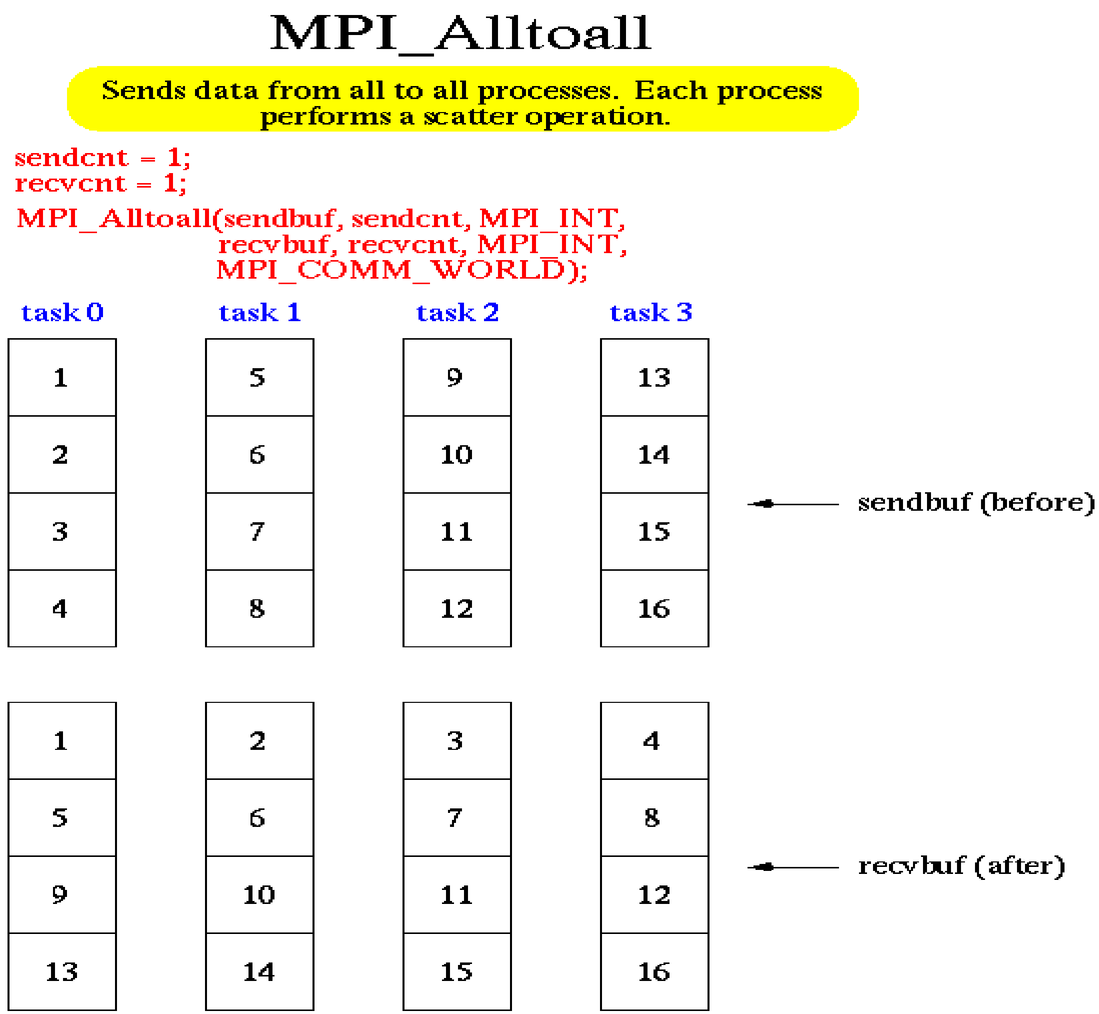
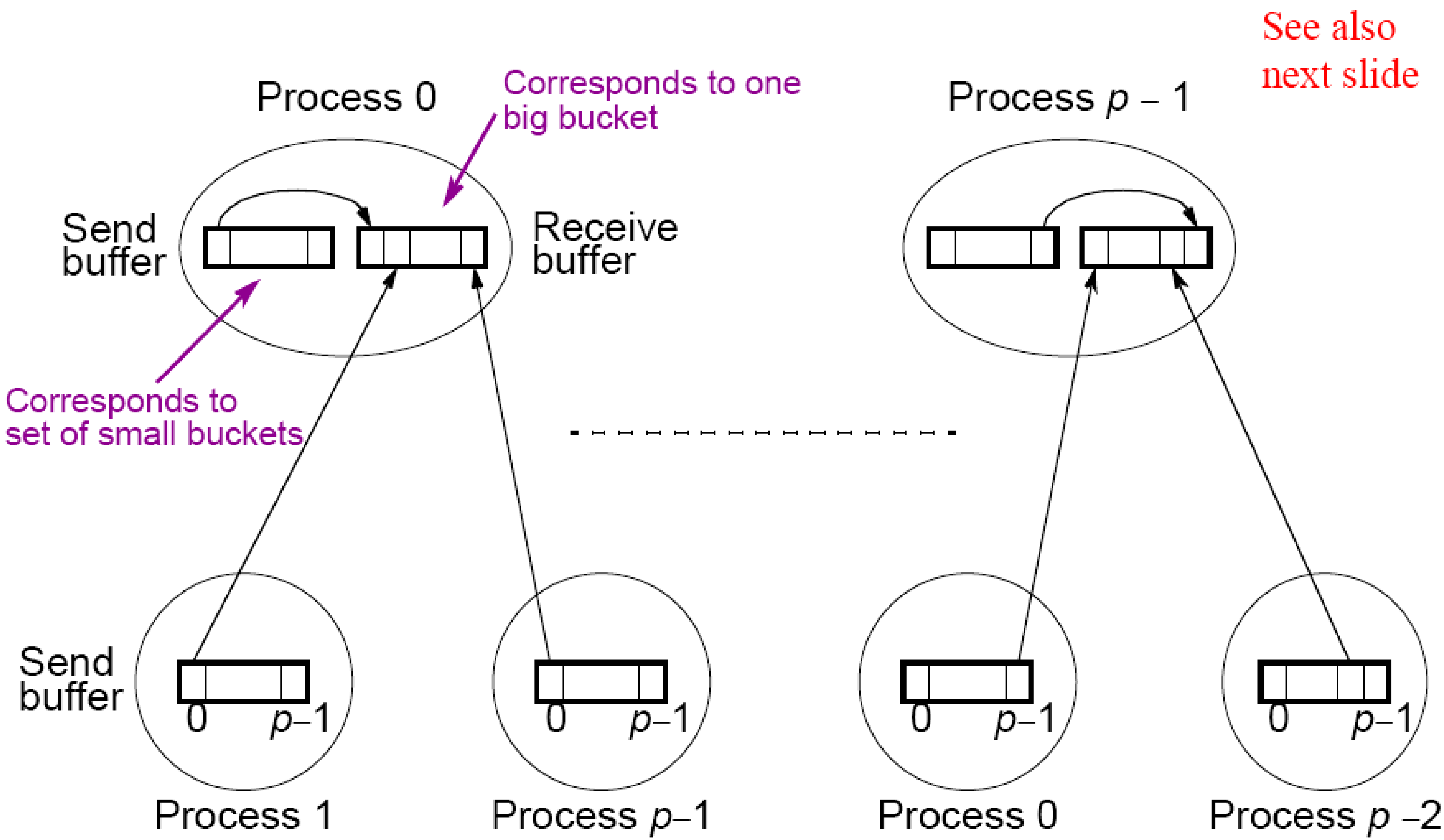


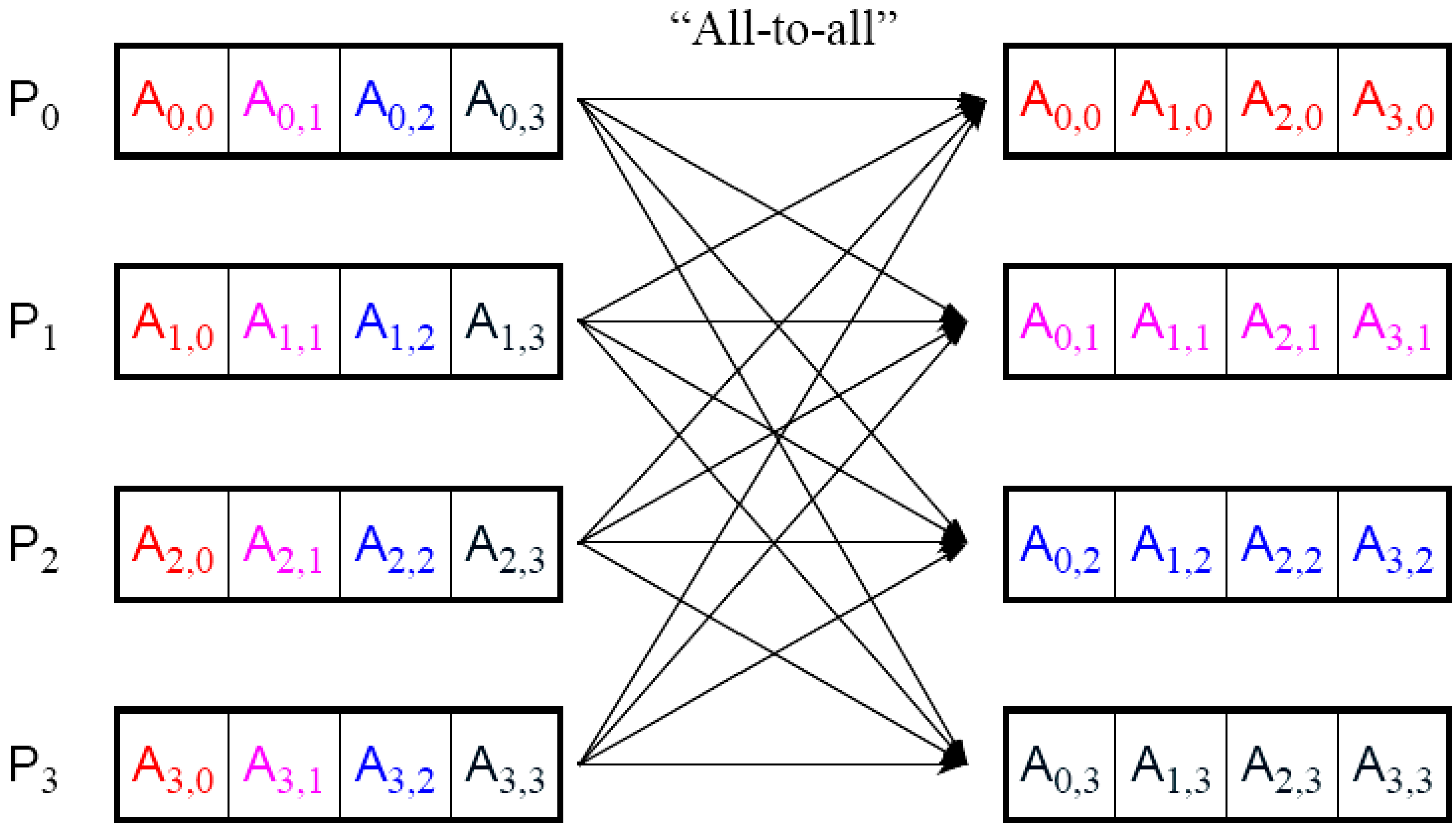
Figure from Blaise Barney, “Message Passing Interface (MPI)”, Livermore Computing

"all-to-all" broadcast routine

Sends data from each process to every other process



“all-to-all” routine actually transfers rows of an array to columns: Transposes a matrix.

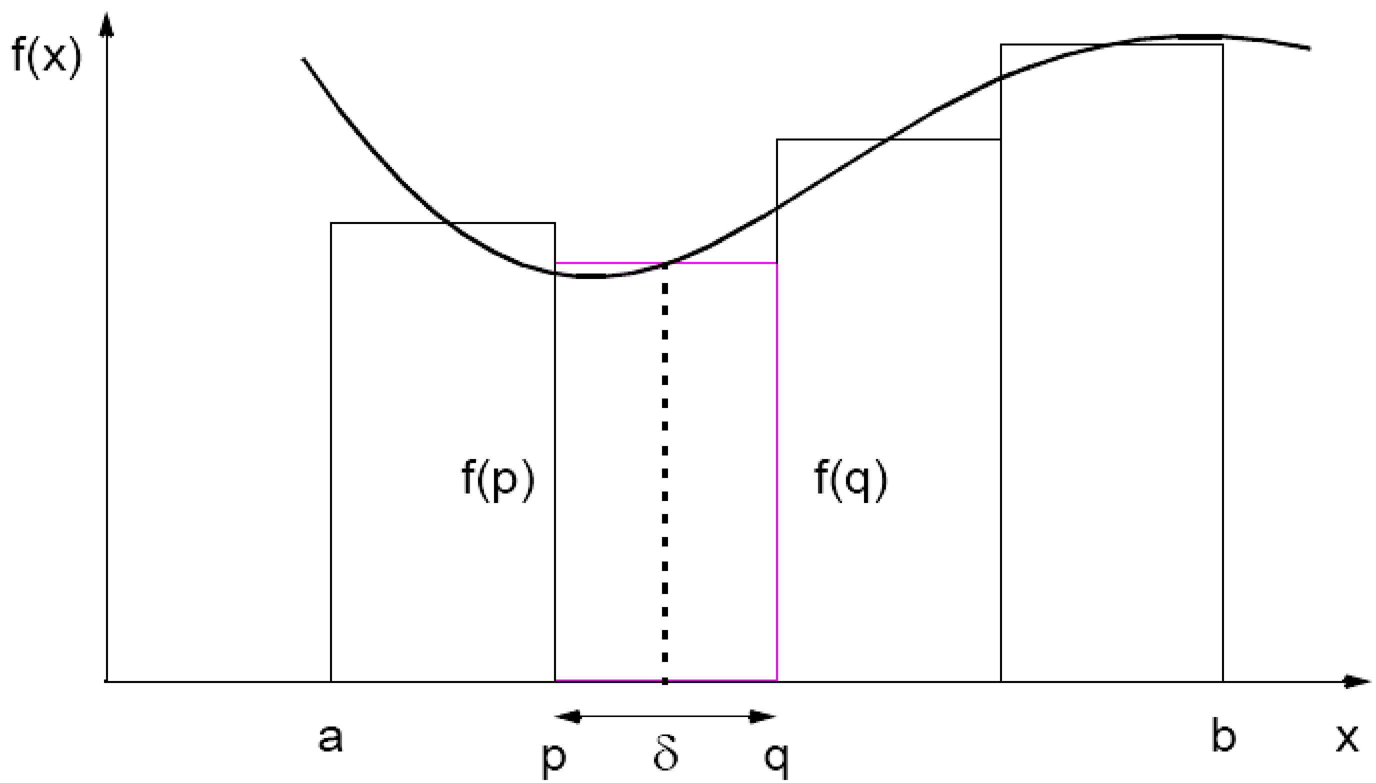


Outline

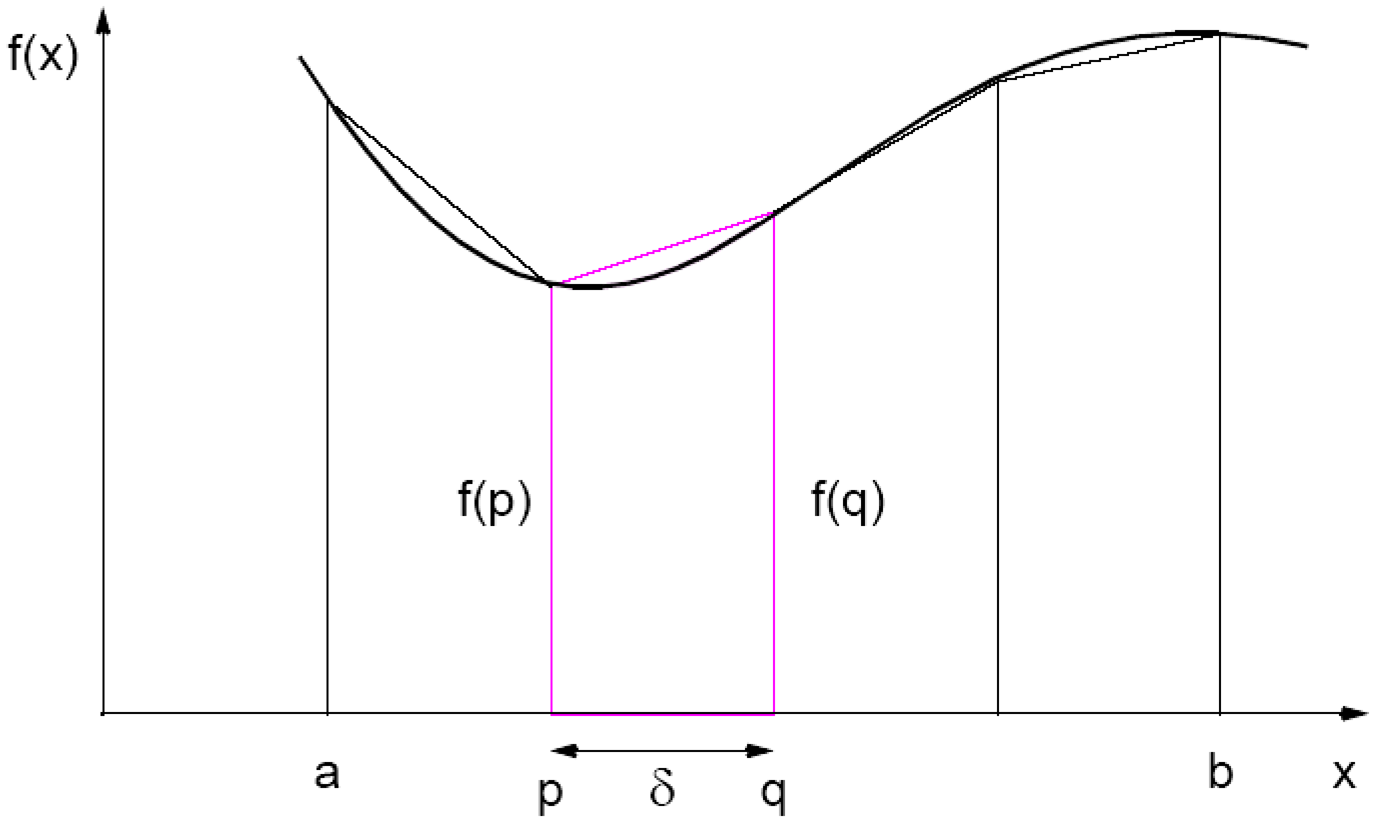
- Partitioning
- Divide-and-conquer
- Partitioning and Divide-and-conquer examples
  - Bucket sort
  - Numerical integration
  - N-body problem

Numerical integration using rectangles

Each region calculated using an approximation given by rectangles:
Aligning the rectangles:



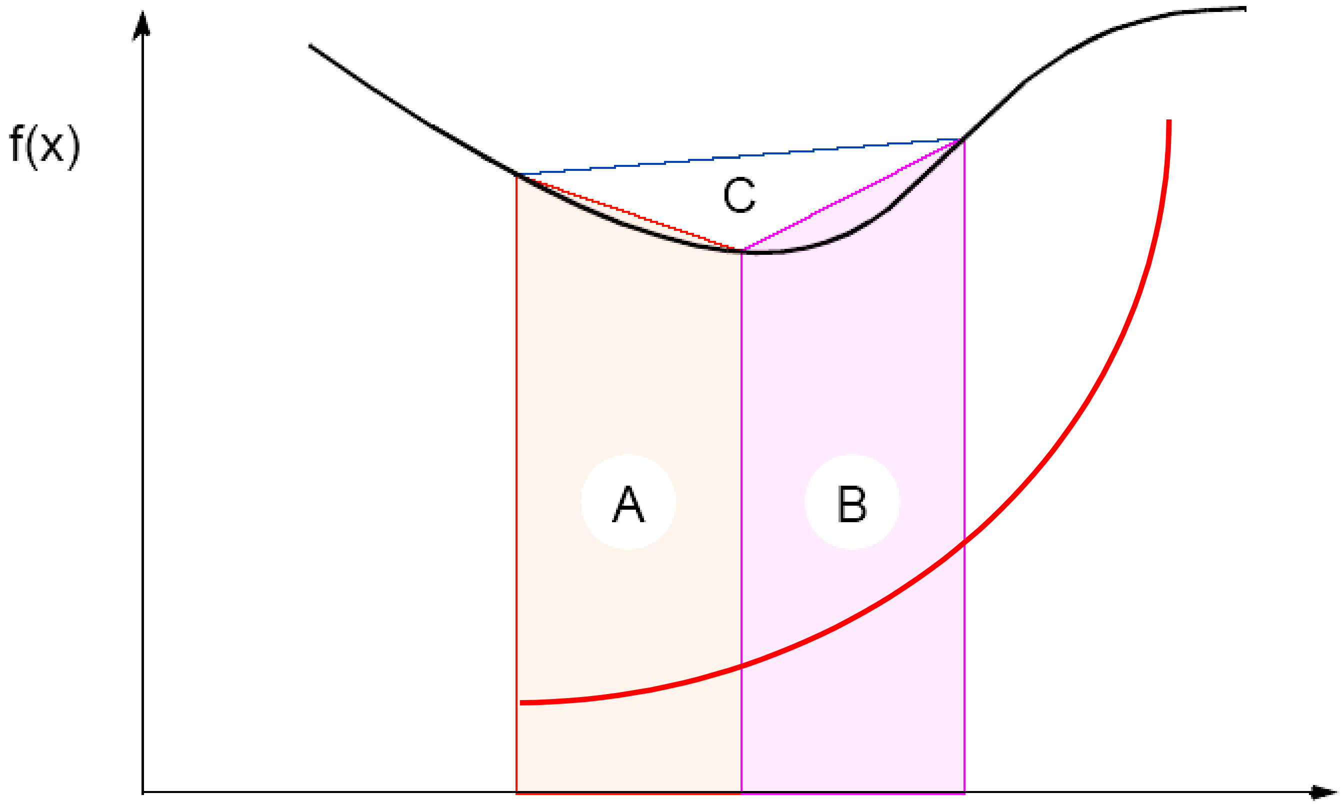
Numerical integration using trapezoidal method



May not be better!

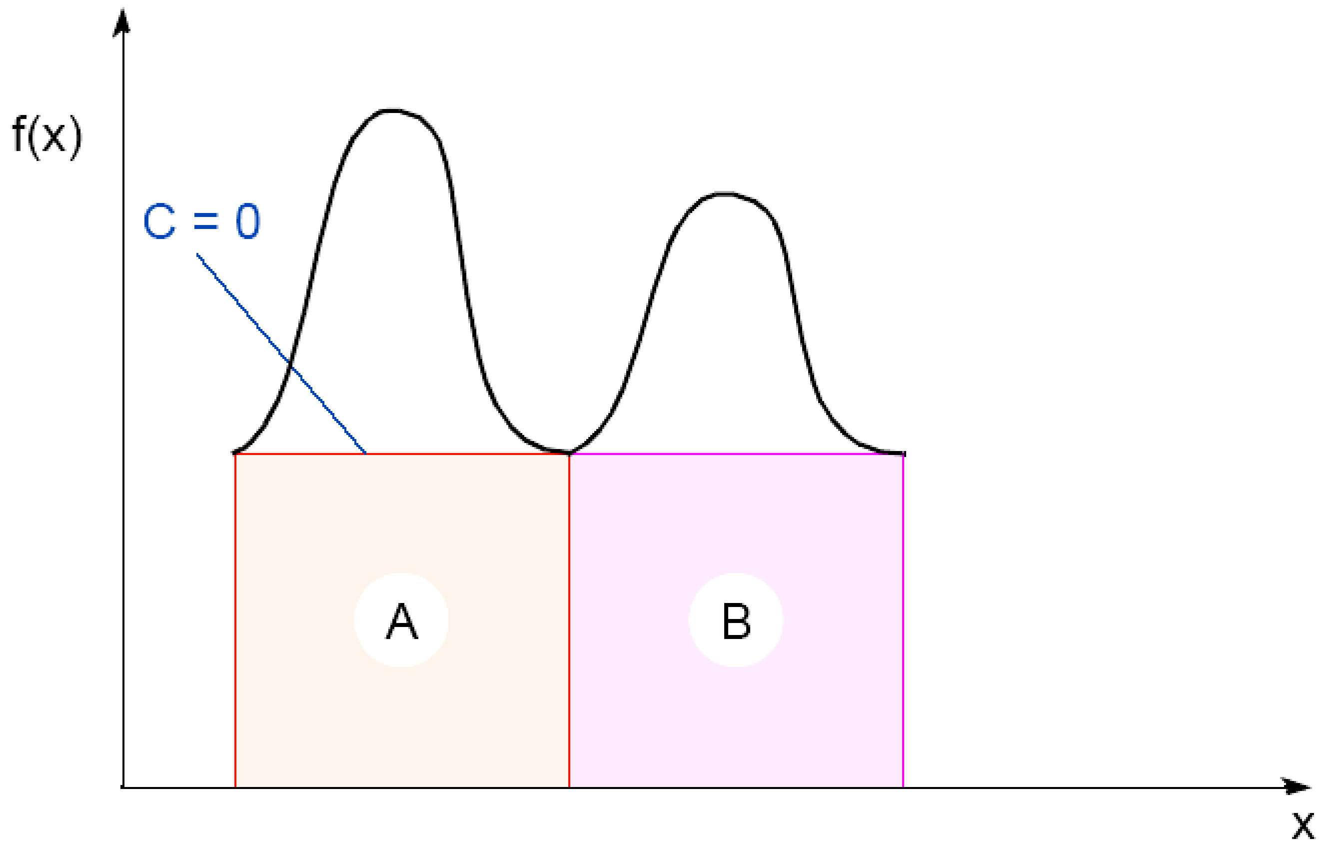
Adaptive Quadrature

Solution adapts to shape of curve. Use three areas, *A*, *B*, and *C*. Computation terminated when larger of *A* and *B* *(e.g. B)* sufficiently close to sum of other two areas *(e.g. A+C)*.



Adaptive quadrature with false termination.

Some care might be needed in choosing when to terminate.



Might cause us to terminate early, as two large regions are the same (i.e., $C = 0$).

Outline

- Partitioning
- Divide-and-conquer
- Partitioning and Divide-and-conquer examples
  - Bucket sort
  - Numerical integration
  - N-body problem

Gravitational *N*-Body Problem

Finding positions and movements of bodies in space subject to gravitational forces from other bodies, using Newtonian laws of physics.

Gravitational *N*-Body Problem Equations

Gravitational force between two bodies of masses m_a and m_b is:

$$F = \frac{Gm_a m_b}{r^2}$$

G is the gravitational constant and *r* the distance between the bodies. Subject to forces, body accelerates according to Newton's 2nd law:

$$F = ma$$

m is mass of the body, *F* is force it experiences, and *a* the resultant acceleration.

N-body simulation

- Two galaxies interact with each other and collide using Barnes-Hut algorithm
- http://www.youtube.com/watch?v=ua7YlN4eL_w

Details

Let the time interval be Δt. For a body of mass m, the force is:

$$F = \frac{m(v^{t+1} - v^{t})}{\Delta t}$$

New velocity is:

$$v^{t+1} = v^{t} + \frac{F\Delta t}{m}$$

where v^{t+1} is the velocity at time $t + 1$ and v^{t} is the velocity at time t.

Over time interval Δt, position changes by

$$x^{t+1} - x^{t} = v\Delta t$$

where x^{t} is its position at time t.

Once bodies move to new positions, forces change.

Computation has to be repeated.

Sequential Code

Overall gravitational *N*-body computation can be described by:

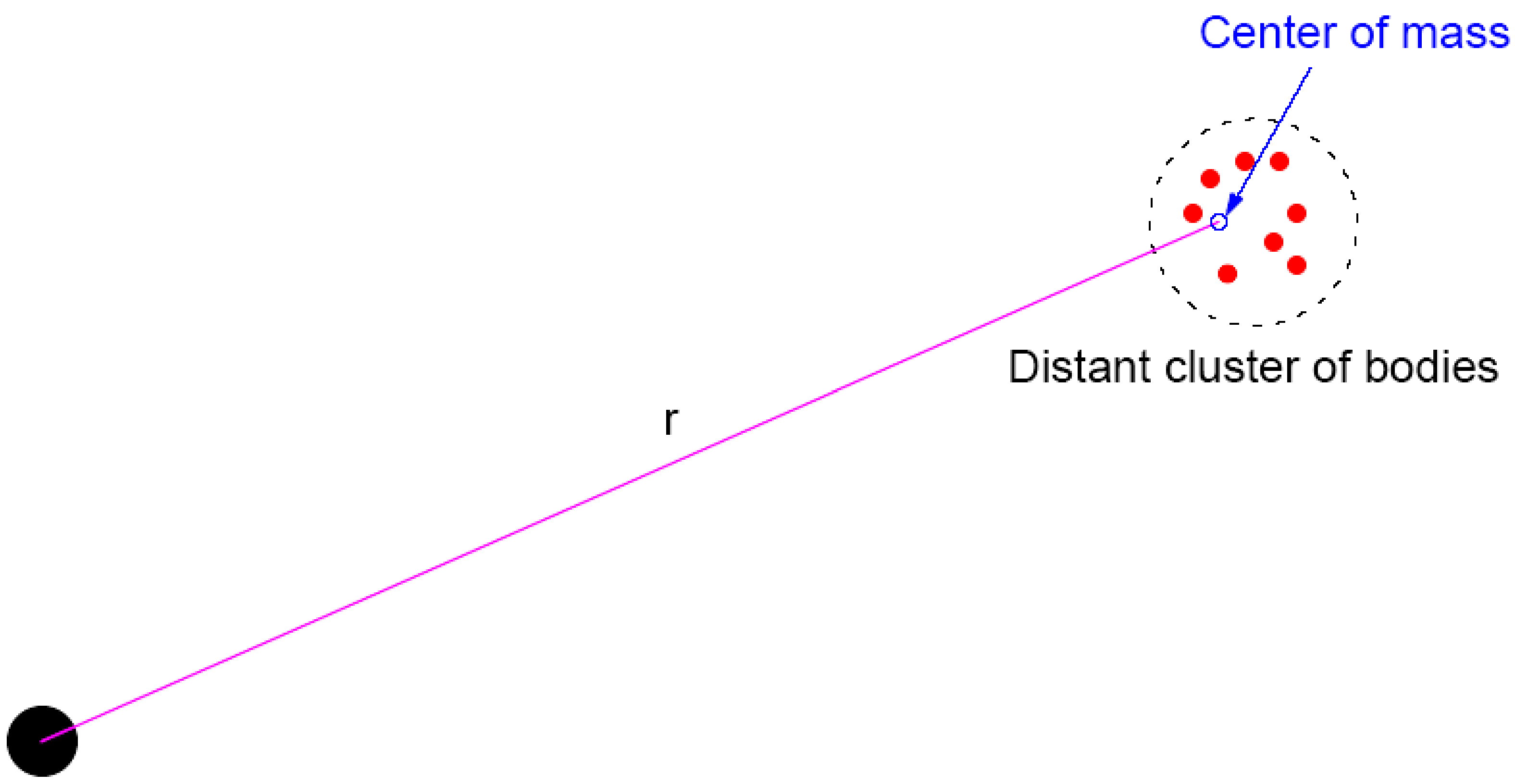
```
for (t = 0; t < tmax; t++) {          // for each time period
  for (i = 0; i < N; i++) {           // for each body
    F = Force_routine(i);             // compute force on/from ith body
    v[i]new = v[i] + F * dt / m;      // compute new velocity
    x[i]new = x[i] + v[i]new * dt;    // and new position
  }
  for (i = 0; i < N; i++) {           // for each body
    x[i] = x[i]new;                   // update velocity & position
    v[i] = v[i]new;
  }
}
```

Parallel Code

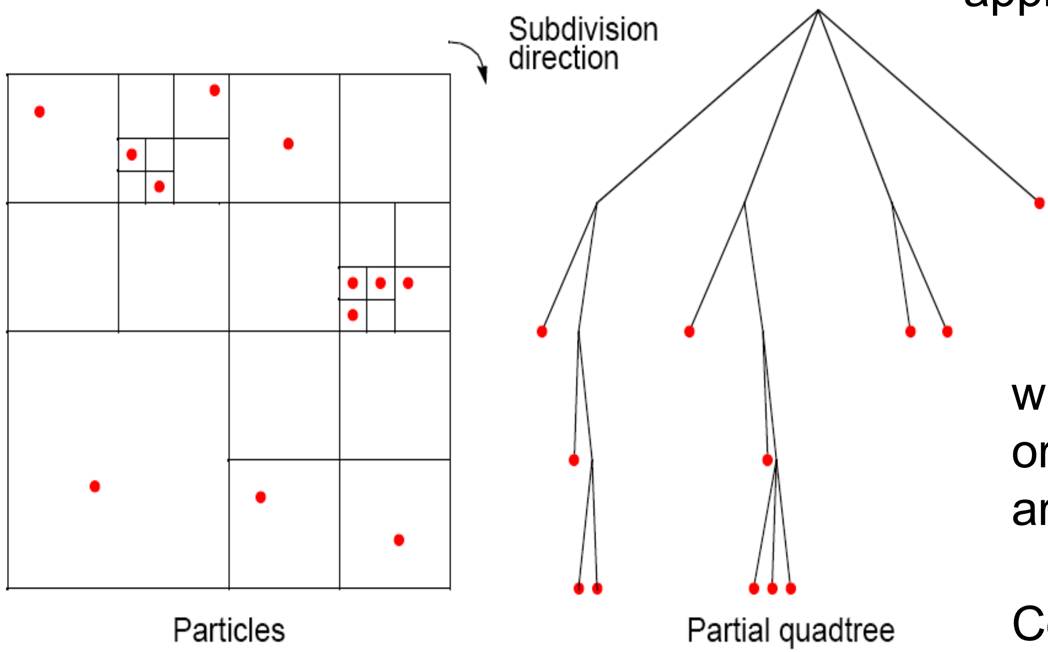
The sequential algorithm is an $O(N^2)$ algorithm (for one iteration) as each of the N bodies is influenced by each of the other N - 1 bodies.

Not feasible to use this direct algorithm for most interesting N-body problems where N is very large.

Time complexity can be reduced approximating a cluster of distant bodies as a single distant body with mass sited at the center of mass of the cluster:



Recursive division of 2-dimensional space



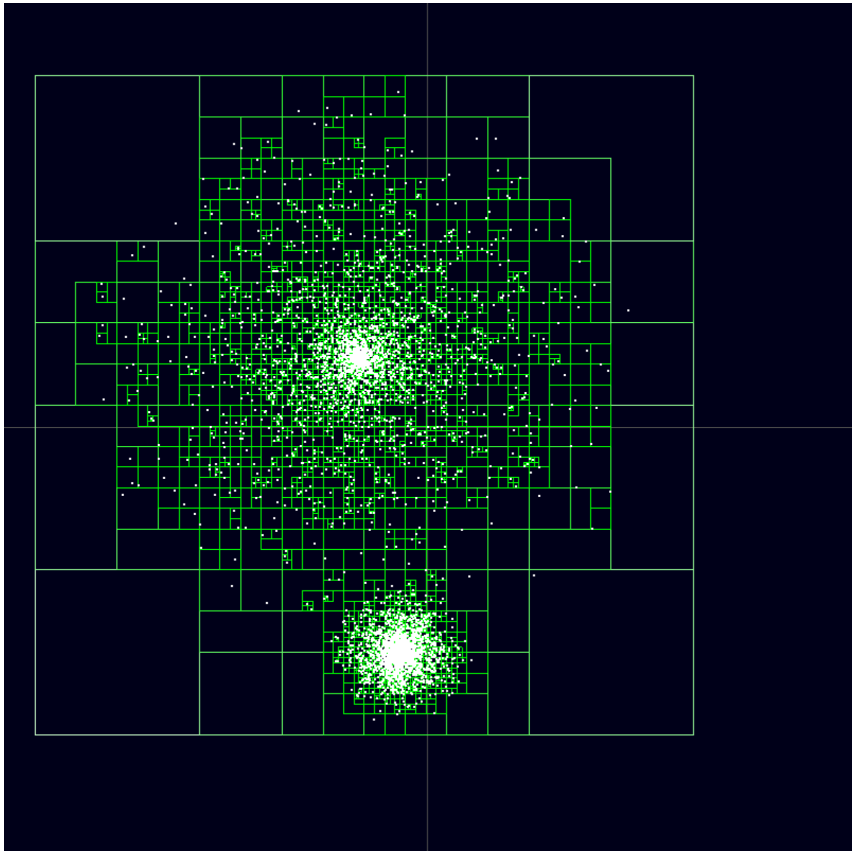
Force on each body obtained by traversing tree starting at root, stopping at a node when the clustering approximation can be used, e.g. when:

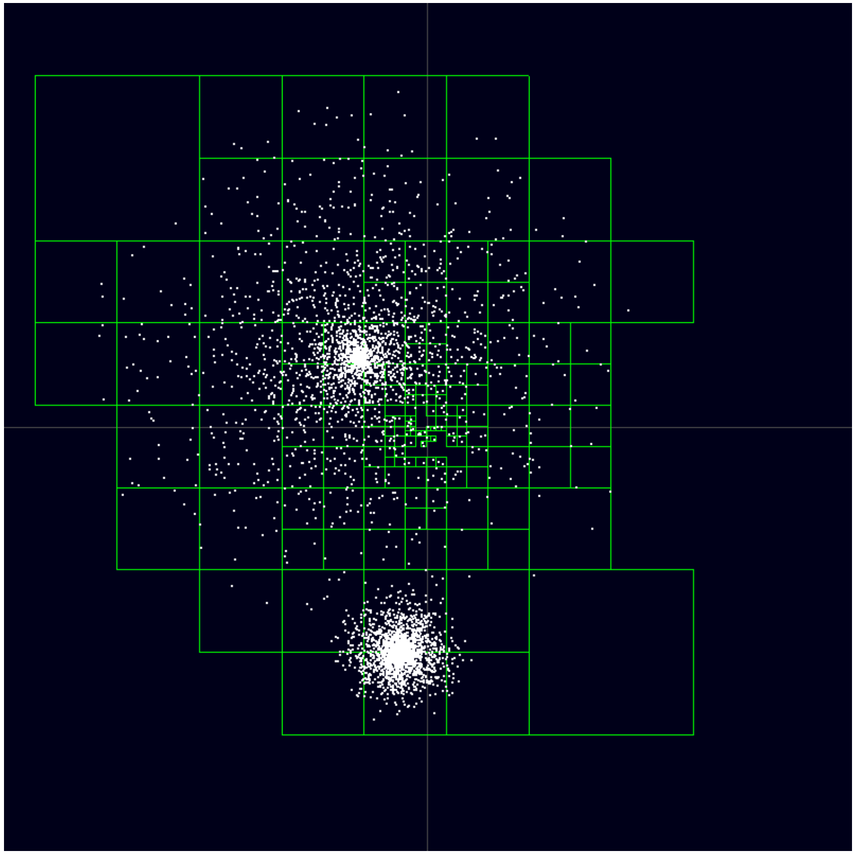
$$r \geq \frac{d}{\theta}$$

where θ is a constant typically 1.0 or less; d is a dimension of the area;

Constructing tree requires a time of O($n$log$n$), and so does computing all the forces, so that overall time complexity of method is O($n$log$n$).

Complete tree vs. used tree

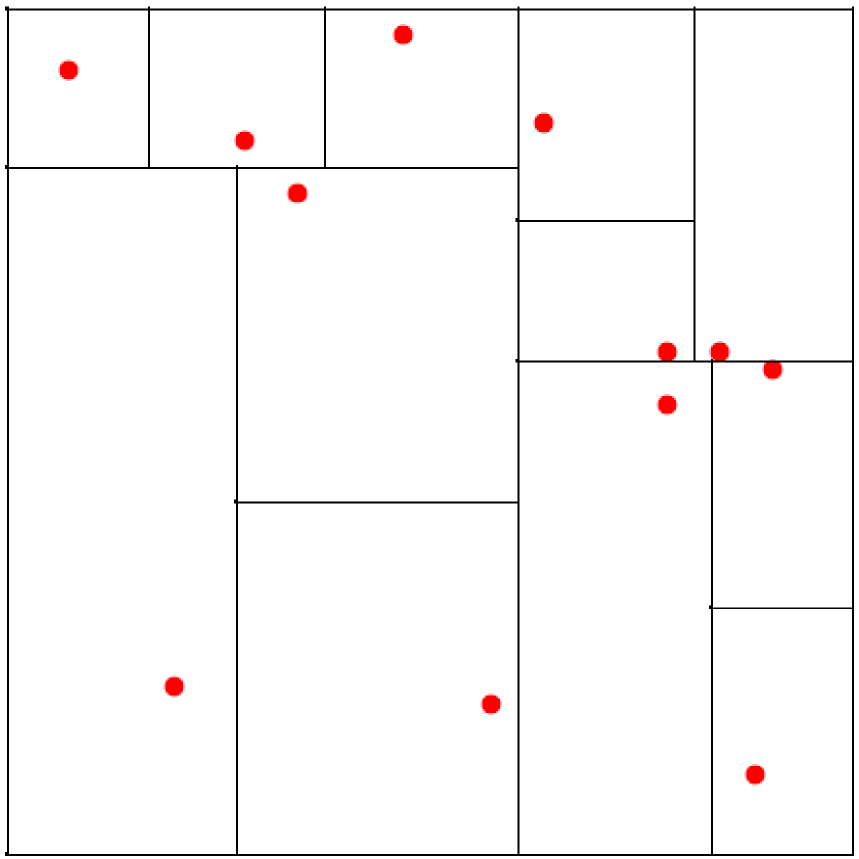




Source: wikipedia

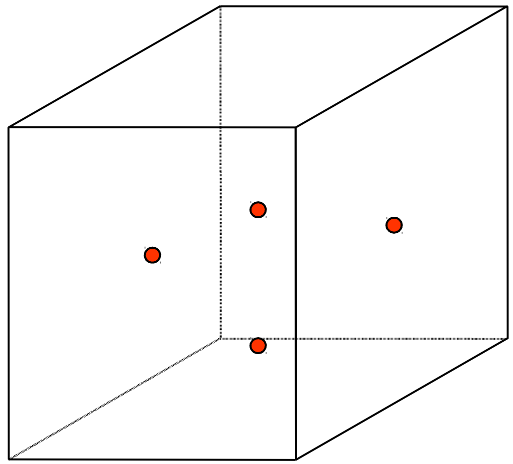
Orthogonal Recursive Bisection

(For 2-dimensional area) First, a vertical line found that divides area into two areas each with equal number of bodies. For each area, a horizontal line found that divides it into two areas each with equal number of bodies. Repeated as required.



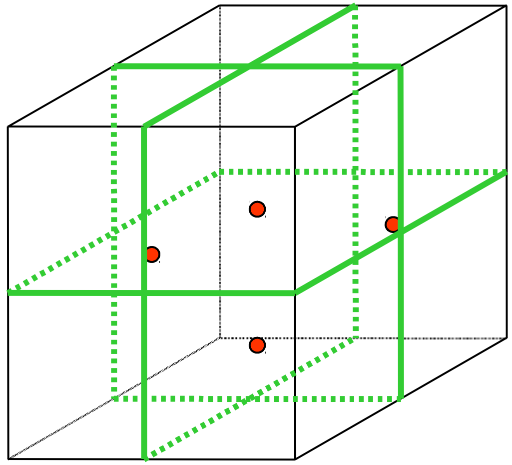
Barnes-Hut Algorithm

Start with whole space in which one cube contains the bodies (or particles).



- First, this cube is divided into eight subcubes.

- If a subcube contains no particles, subcube deleted from further consideration.



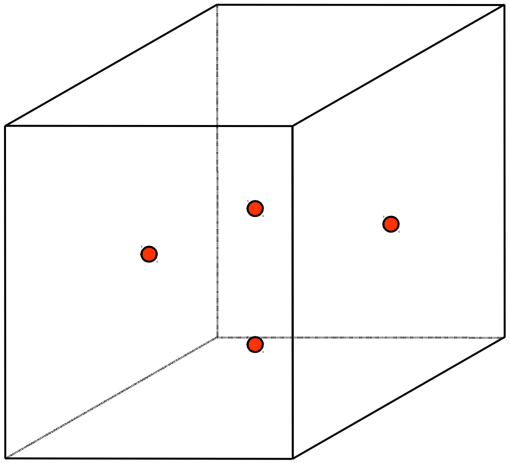
- If a subcube contains one body, subcube retained.

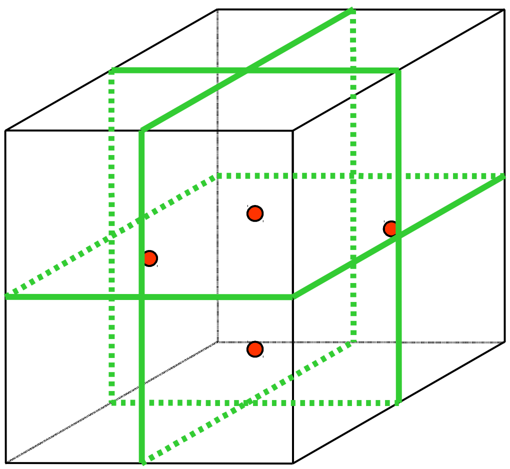
- If a subcube contains more than one body, it is recursively divided until every subcube contains one body.

Creates an *octtree* - a tree with up to eight edges from each node.

The leaves represent cells each containing one body.

After the tree has been constructed, the total mass and center of mass of the subcube is stored at each node.





References

- Barry Wilkinson & Michael Allen. Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers.
- Blaise Barney, “Message Passing Interface (MPI)”, Livermore Computing.