Outline: Parallelisation via Data Partitioning

- partitioning strategies
- vector summation via partitioning, via divide-and-conquer
- binary trees (divide-and-conquer)
- reduce and scan algorithms
- bucket sort
- numerical integration - adaptive techniques
- N-body problems

Ref: Lin and Snyder Ch 5, Wilkinson and Allen Ch 4.
Challenge from L8: can you write a well balanced parallel Mandelbrot set program using static task assignment?

Partitioning Strategies

- replicated data approach (no partitioning)
  - each process has entire copy of data but does subset of computation
- partition program data to different processes
  - most common
  - strategies: domain decomposition, divide-and-conquer
- partitioning of program functionality
  - much less common
  - functional decomposition
- consider the addition of numbers

$$s = \sum_{i=0}^{n-1} x_i$$

Example#1: Simple Summation of Vector

- divide numbers into $m$ equal parts

Master/Slave Send/Recv Approach

Master:

```c
s = n / m;
for (i = 0, x = 0; i < m; i++, x = x + s)
  send(&numbers[x], s, i+1 /*slave id*/);
sum = 0;
for (i = 0; i < m; i++)
  recv(&part_sum, any_proc);
  sum = sum + part_sum;
```

Slave:

```c
recv(numbers, s, master);
part_sum = 0;
for (i = 0; i < s; i++)
  part_sum = part_sum + numbers[i];
send(&part_sum, master);
```
Using MPI_Scatter and MPI_Reduce

See man MPI_Scatter and man MPI_Reduce

- sendcount = n/m;
- MPI_Scatter(numbers, s /*sendcount*/, MPI_FLOAT, /*send data*/
  numbers, s /*recvcount*/, MPI_FLOAT, /*recv data*/
  0 /*root*/, MPI_COMM_WORLD);

for (i = 0; i < s; i++)
  part_sum = part_sum + numbers[i];

MPI_Reduce(&part_sum, &sum, 1 /*count*/, MPI_FLOAT,
  MPI_SUM, 0 /*root*/, MPI_COMM_WORLD);

- NOT master/slave
- the root sends data to all processes (including itself)

- note related MPI calls:
  - MPI_ScatterV(): scatters variable lengths
  - MPI_Allreduce(): returns result to all processors

Domain Decomposition via Divide-and-Conquer

- problems that can be recursively divided into smaller problems of the same type
- recursive implementation of the summation problem:

  ```c
  int add(int *s) {
    if (numbers(s) == 1)
      return (s[0]);
    else {
      divide(s, s1, s2);
      part_sum1 = add(s1);
      part_sum2 = add(s2);
      return (part_sum1 + part_sum2);
    }
  }
  ```

Analysis

Sequential:

- n − 1 additions thus O(n)

Parallel (p = m):

- communication #1: \(t_{\text{scatter}} = p(l_s + \frac{n}{p}l_w)\)
- computation #1: \(t_{\text{partialsum}} = \frac{n}{p}l_f\)
- communication #2: \(t_{\text{reduce}} = p(l_s + l_w)\)
- computation #2: \(t_{\text{finalsum}} = (p - 1)l_f\)
- overall: \(t_p = 2p l_s + (n + p)l_w + (n/p + p - 1)l_f = O(n + p)\)
- worse than sequential code!!

Discussion point: in this example, we are assuming the associative property of addition (+)? Is this strictly true for floating point numbers? What impact does this have for such parallel algorithms?

Binary Tree

- divide-and-conquer with binary partitioning
- note number of working processors decreases going up the tree

![Binary Tree Diagram](image-url)
### Simple Binary Tree Code

```c
/* Binary Tree broadcast */
a) 0->1
b) 0->2, 1->3
c) 0->4, 1->5, 2->6, 3->7
d) 0->8, 1->9, 2->10, 3->11, 4->12, 5->13, 6->14, 7->15 */

// This is used to scatter the vector; the reverse algorithm combines the partial sums.
```

### Higher Order Trees

- possible to divide data into higher order trees, e.g. a quad tree

### Analysis

- assume $n$ is a power of 2 and ignoring $t_s$
- communication#1: divide
  $$t_{\text{divide}} = \frac{2}{p} t_w + \frac{4}{p} t_w + \frac{8}{p} t_w + \ldots + \frac{n}{p} t_w = \frac{n(p-1)}{2} t_w$$
- communication#2: combine
  $$t_{\text{combine}} = \lg p \cdot t_w$$
- computation:
  $$t_{\text{comp}} = (\frac{n}{p} + \lg p) t_f$$
- total:
  $$t_p = (\frac{n(p-1)}{p} + \lg p) t_w + (\frac{n}{p} + \lg p) t_f$$
- slightly better than before - as $p \to n$, cost $\to O(n)$

### The Reduce and Scan Abstractions

- the summation of a vector already partitioned between processes is an example of the reduce and scan abstraction
- reduce: combines a set of values to produce a single value
- scan: performs a sequential operation in parts and carries along the intermediate results
- reduce usually involves mapping a binary (or higher level) tree communication pattern between processes
- examples (to discuss) include
  - finding second smallest array element
  - computing a k-way histogram
  - length of longest run of 1s
  - index of first occurrence of x
(see Lin and Snyder for further details)
### Scans

- Consider a vector \( X = [0, 1, 2, 3, 4, 5, 6, 7] \)
- The scan operation replaces each element with the cumulative sum of all preceding elements (either inclusive or exclusive of current element)
- Inclusive scan \( X \) = \([0, 1, 3, 6, 10, 15, 21, 28] \)
- Exclusive scan \( X \) = \([0, 1, 3, 6, 10, 15, 21] \)
- Inclusive sequential scan code (clear dependency!)
  
  ```
  for (i=1; i<n; i++)
  X[i] = X[i-1] + X[i];
  ```
- Parallel PRAM code (why is PRAM important?)
  
  ```
  for (d=1; d < N; d *= 2)
  FORALL (k=0; k < N; k++) IN PARALLEL
  if (k >= d)
  X[k] = X[k-d] + X[k];
  ```
- Key paper: Scans as Primitive Parallel Operations, Guy Blelloch, 1989

### Example#2: Bucket Sort

- Divide number range \((a)\) into \(m\) equal regions
  
  \[
  \left(0 \rightarrow \frac{a}{m} - 1 \right), \left(\frac{a}{m} \rightarrow 2\frac{a}{m} - 1 \right), \left(2\frac{a}{m} \rightarrow 3\frac{a}{m} - 1 \right), \ldots
  \]
- Assign each number to a bucket
- Stage 1: Numbers are placed into appropriate buckets
- Stage 2: Each bucket is sorted using a traditional sorting algorithm
- Works best if numbers are evenly distributed over the range \(a\)
- Sequential time
  
  \[
  t_s = n + m(n/m \log(n/m)) = n + n \log(n/m) = O(n \log(n/m))
  \]

### Scan Example: 8 Values on 8 Nodes

- For \((d=1; d < N; d *= 2)\)
  
  ```
  FORALL (k=0; k < N; k++) IN PARALLEL
  if (k >= d)
  X[k] = X[k-d] + X[k];
  ```
- Node and value of data on that node

<table>
<thead>
<tr>
<th>step ((j)) ((d = 2^{j-1}))</th>
<th>offset</th>
<th>000</th>
<th>001</th>
<th>010</th>
<th>011</th>
<th>100</th>
<th>101</th>
<th>110</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0+1</td>
<td>1+2</td>
<td>3</td>
<td>4+5</td>
<td>5+6</td>
<td>6+7</td>
<td>6+7</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0+3</td>
<td>1+5</td>
<td>3+7</td>
<td>5+9</td>
<td>7+11</td>
<td>9+13</td>
<td>22</td>
</tr>
</tbody>
</table>

### Sequential Bucket Sort

- See also the serial C implementation of the above algorithm.
Parallel Bucket Sort#1

- assign one bucket to each process:

Parallel Bucket Sort#2

- assign \( p \) small buckets to each process
- note possible use of `MPI_Alltoall()`

### Analysis

- initial partitioning and distribution
  \[ t_{\text{comm1}} = pt_s + t_w n \]
- sort into small buckets
  \[ t_{\text{comp2}} = n/p \]
- send to large buckets: (overlapping communications)
  \[ t_{\text{comm3}} = (p - 1)(ts + [n/p^2]t_w) \]
- sort of large buckets
  \[ t_{\text{comp4}} = (n/p)\log(n/p) \]
- total
  \[ t_p = pt_s + nt_w + n/p + (p - 1)(ts + [n/p^2]t_w) + (n/p)\log(n/p) \]
- at best \( O(n) \)
- what would be the worse case scenario?

### Example#3: Integration

- consider the evaluation of an integral using the trapezoidal rule
  \[ I = \int_a^b f(x)dx \]
Static Distribution: SPMD Model

```c
if (process_id == master) {
    printf("Enter number of regions\n");
    scanf("%d", &n);
}
broadcast(&n, master, p_group);
region = (a - b) / p;
start = a + region * process_id;
end = start + region;
d = (b - a) / n;
area = 0.0;
for (x = start; x < end; x = x + d)
    area = area + 0.5 * (f(x) + f(x + d)) * d;
reduce_add(&area, master, p_group);
```

Example#4: N-Body Problems

- summing long-range pairwise interactions, e.g. gravitation
  
  \[ F = \frac{G m_a m_b}{r^2} \]

  where \( G \) is the gravitational constant, \( m_a \) and \( m_b \) are the mass of two bodies, and \( r \) is the distance between them

- in Cartesian space:
  
  \[
  F_x = \frac{G m_a m_b}{r^2} \left( \frac{x_b - x_a}{r} \right)
  
  F_y = \frac{G m_a m_b}{r^2} \left( \frac{y_b - y_a}{r} \right)
  
  F_z = \frac{G m_a m_b}{r^2} \left( \frac{z_b - z_a}{r} \right)
  
- what is the total force on the sun due to all other stars in the milky way?
- given the force on each star we can calculate their motions
- molecular dynamics is very similar but the long forces are electrostatic

Adaptive Quadrature

- not all areas require the same number of points
- when to terminate division into smaller areas is an issue
- the parallel code will have uneven workload

```
f(x)
```

Simple Sequential Force Code

```c
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        if (i != j)
            rij2 = (x[i] - x[j]) * (x[i] - x[j])
                 + (y[i] - y[j]) * (y[i] - y[j])
                 + (z[i] - z[j]) * (z[i] - z[j]);
            Fx[i] = Fx[i] + G * m[i] * m[j] / rij2 * (x[i] - x[j]) / sqrt(rij2);
            Fy[i] = Fy[i] + G * m[i] * m[j] / rij2 * (y[i] - y[j]) / sqrt(rij2);
            Fz[i] = Fz[i] + G * m[i] * m[j] / rij2 * (z[i] - z[j]) / sqrt(rij2);
```
Clustering

• idea: the interaction with several bodies that are clustered together but are located at large $r$ for another body can be replaced by the interaction with the center of mass of the cluster

Barnes-Hut Algorithm

• start with whole space in one cube
  ■ divide the cube into 8 sub-cubes
  ■ delete sub-cubes if they have no particles in them
  ■ sub-cubes with more than 1 particle are divided into 8 again
  ■ continue until each cube has only one particle (or none)

• this process creates an oct-tree
• total mass and centre of mass of children sub-cubes is stored at each node
• force is evaluated by starting at the root and traversing the tree, BUT stopping at a node if the clustering algorithm can be used
• scaling is $O(n \log n)$
• load balancing likely to be an issue for parallel code