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

\[
x(0) \ldots x(n/m-1) \quad x(n/m) \ldots x(2n/m-1) \quad x((m-1)n/m) \ldots x(n-1)
\]

![Diagram showing the summation process](image-url)
Master/Slave Send/Recv Approach

Master:

\[ s = \frac{n}{m} \]

\[ \text{for} \ (i = 0, \ x = 0; \ i < m; \ i++, \ x = x + s) \]
\[ \text{send}(&\text{numbers}[x], \ s, \ i+1 /*slave id*/); \]

\[ \text{sum} = 0; \]
\[ \text{for} \ (i = 0; \ i < m; \ i++) \{ \]
\[ \text{recv}(&\text{part_sum}, \ \text{any_proc}); \]
\[ \text{sum} = \text{sum} + \text{part_sum}; \]
\[ \} \]

Slave:

\[ \text{recv}(\text{numbers}, \ s, \ \text{master}); \]
\[ \text{part_sum} = 0; \]
\[ \text{for} \ (i = 0; \ i < s; \ i++) \]
\[ \text{part_sum} = \text{part_sum} + \text{numbers}[i]; \]
\[ \text{send}(&\text{part_sum}, \ \text{master}); \]
Using MPI_Scatter and MPI_Reduce

See `man MPI_Scatter` and `man MPI_Reduce`

```c
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
Analysis

Sequential:

- \( n - 1 \) additions thus \( O(n) \)

Parallel \((p = m)\):

- communication \#1: \( t_{\text{scatter}} = p(t_s + \frac{n}{p}t_w) \)
- computation \#1: \( t_{\text{partialsum}} = (\frac{n}{p})t_f \)
- communication \#2: \( t_{\text{reduce}} = p(t_s + t_w) \)
- computation \#2: \( t_{\text{finalsum}} = (p - 1)t_f \)
- overall: \( t_p = 2pt_s + (n + p)t_w + (n/p + p - 1)t_f = O(n + p) \)

- 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/
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);
    }
}
```
- **divide-and-conquer** with binary partitioning
- note number of working processors decreases going up the tree
Simple Binary Tree Code

/* 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
*/

lo = 1;
while (lo < nproc) {
    if (me < lo) {
        id = me + lo;
        if (id < nproc)
            send(buf, lenbuf, id);
    }
    else if (me < 2*lo) {
        id = lo;
        recv(buf, lenbuf, id);
    }
    lo *= 2;
}

This is used to scatter the vector; the reverse algorithm combines the partial sums.
**Analysis**

- Assume $n$ is a power of 2 and ignoring $t_s$

- Communication #1: divide
  \[
  t_{\text{divide}} = \frac{n}{2}t_W + \frac{n}{4}t_W + \frac{n}{8}t_W + \cdots + \frac{n}{p}t_W = \frac{n(p-1)}{p}t_W
  \]

- Communication #2: combine
  \[
  t_{\text{combine}} = \lg p \cdot t_W
  \]

- Computation:
  \[
  t_{\text{comp}} = \left(\frac{n}{p} + \lg p\right)t_f
  \]

- Total:
  \[
  t_P = \left(\frac{n(p-1)}{p} + \lg p\right)t_W + \left(\frac{n}{p} + \lg p\right)t_f
  \]

- Slightly better than before - as $p \to n$, cost $\to O(n)$
Higher Order Trees

possible to divide data into higher order trees, e.g. a quad tree
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, 0, 1, 3, 6, 10, 15, 21]
- inclusive sequential scan code (clear dependency!)
  ```plaintext
  \[
  \text{scan} X[0] = X[0];
  \text{for} (i=1; i<N; i++)
  \quad \text{scan} X[i] = \text{scan} X[i-1] + X[i];
  \]
- parallel PRAM code (why is PRAM important?)
  ```plaintext
  ```plaintext
  \text{for} (d=1; d < N; d *= 2)
  \quad \text{FORALL} (k=0; k < N; k++) \text{ IN PARALLEL}
  \quad \text{if} (k >= d)
  \quad \quad X[k] = X[k-d] + X[k];
  ```
- key paper: Scans as Primitive Parallel Operations, Guy Blelloch, 1989
Scan Example: 8 Values on 8 Nodes

\[
\begin{align*}
\text{for } (d=1; \ d < N; \ d \times= 2) \\
\quad \text{FORALL } (k=0; \ k < N; \ k++) \text{ IN PARALLEL} \\
\quad \quad \text{if } (k \geq d) \\
\quad \quad \quad X[k] = X[k-d] + X[k]; \\
\end{align*}
\]

<table>
<thead>
<tr>
<th>step ((j))</th>
<th>offset ((d = 2^{j-1}))</th>
<th>node and value of data on that node</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>000 001 010 011 100 101 110 111</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0   1   2   3   4   5   6   7</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1   0+1=1 1+2=3 2+3=5 3+4=7 4+5=9 5+6=11 6+7=13</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2   0   1   0+3=3 1+5=6 3+7=10 5+9=14 7+11=18</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>4   0   1   3   6   0+10=10 1+14=15 3+18=21</td>
<td></td>
</tr>
<tr>
<td>final values</td>
<td>0   1   3   6   10  15  21  28</td>
<td></td>
</tr>
</tbody>
</table>

See also the serial C implementation of the above algorithm.
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 one bucket to each region
- 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

\[
t_S = n + m((n/m) \lg(n/m)) = n + n \lg(n/m) = O(n \lg(n/m))
\]
Sequential Bucket Sort
Parallel Bucket Sort#1

- assign one bucket to each process:

![Diagram showing Parallel Bucket Sort]

- Processes distribute and collect elements.
- Buckets hold unsorted numbers.
- Sorted numbers are concatenated.
Parallel Bucket Sort#2

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

\[
\text{MPI\_Alltoall(} \text{void}^* \text{sendbuf, int sendcount, MPI\_Datatype sendtype,}
\text{void}^* \text{recvbuf, int recvcount, MPI\_Datatype recvtype,}
\text{MPI\_Comm comm)}
\]
Analysis

- initial partitioning and distribution
  \[ t_{\text{comm}1} = pt_s + twn \]

- sort into small buckets
  \[ t_{\text{comp}2} = n/p \]

- send to large buckets: (overlapping communications)
  \[ t_{\text{comm}3} = (p - 1)(ts + (n/p^2)tw) \]

- sort of large buckets
  \[ t_{\text{comp}4} = (n/p)\lg(n/p) \]

- total
  \[ t_p = pt_s + nt_w + n/p + (p - 1)(ts + (n/p^2)tw) + (n/p)\lg(n/p) \]

- at best \( O(n) \)

- what would be the worse case scenario?
Example#3: Integration

Consider the evaluation of an integral using the trapezoidal rule

\[ I = \int_{a}^{b} f(x) \, dx \]
if (process_id == master) {
    printf("Enter number of regions\n");
    scanf("%d", &n);
}
broadcast(&n, master, p_group)
region = (b−a)/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);
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
Example#4: N-Body Problems

- summing long-range pairwise interactions, e.g. gravitation

\[ F = \frac{Gm_am_b}{r^2} \]

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:

\[ F_x = \frac{Gm_am_b}{r^2} \left( \frac{x_b-x_a}{r} \right) \]
\[ F_y = \frac{Gm_am_b}{r^2} \left( \frac{y_b-y_a}{r} \right) \]
\[ F_z = \frac{Gm_am_b}{r^2} \left( \frac{z_b-z_a}{r} \right) \]

- 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
Simple Sequential Force Code

```c
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++) {
        if (i != j) {
            r_{ij}^2 = (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]/r_{ij}^2 * (x[i] - x[j]) / sqrt(r_{ij}^2);
            Fy[i] = Fy[i] + G*m[i]*m[j]/r_{ij}^2 * (y[i] - y[j]) / sqrt(r_{ij}^2);
            Fz[i] = Fz[i] + G*m[i]*m[j]/r_{ij}^2 * (z[i] - z[j]) / sqrt(r_{ij}^2);
        }
    }
```

- aside: how could you improve this sequential code?
- $O(n^2)$ - this will get very expensive for large $n$
- is there a better way?
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
Barnes-Hut Algorithm: 2D Illustration