Algorithms PART II: Partitioning and Divide & Conquer

HPC Fall 2010
Prof. Robert van Engelen
Overview

- Partitioning strategies
- Divide and conquer strategies
- Further reading
Partitioning Strategies

- **Data partitioning**
  - Perform *domain decomposition* to run parallel tasks on subdomains
  - “Scatter-compute-gather” where local computation may require communication and scatter/gather may involve computations

- **Task partitioning**
  - Decompose functions into independent subfunctions and execute the subfunctions in parallel

### Code Example

```
function f(x,y)
    u=g(x)
    v=h(y)
    return u+v
end

Thread 1
u=g(x)

Thread 2
v=h(y)

return u+v
```

*Block partitioning of a 2D domain*
Partitioning Strategies

Partitioning strategy (data partitioning):
1. Break up a given problem into $P$ subproblems
2. Solve the $P$ subproblems concurrently
3. Collect and combine the $P$ solutions

Embarrassingly parallel

- Is a simple form of data partitioning into independent subproblems without initial work and no communication between tasks (workers)
Partitioning Example 1: Summation

- Summation of $n$ values $X = [x_1, \ldots, x_n]$

1. Divide $X$ into $P$ equally-sized sublists $X_p$, $p = 0, \ldots, P-1$ and distribute the $X_p$ sublists to the $P$ processors
2. The processors sum the local parts $s_p = \sum X_p$
3. Combine the local sums $s = \sum s_p$

- Algorithms:
  1. Scatter list $X$ using a scatter-tree
  2. Serial summation of parts
  3. Reduce local sums
Partitioning Example 1: Summation

Local summations: $n/P$ steps

Total amount of data transferred: $n/2 \log_2(P)$

Log$_2(P)$ steps

Total amount of data transferred: $P-1$
Partitioning Example 1: Summation

- **Communication time**
  - Scatter: \( t_{\text{comm}1} = \sum_{k=1}^{\log_2(P)} (t_{\text{startup}} + 2^{-k}n \ t_{\text{data}}) = \log_2(P) t_{\text{start}} + n(P-1)/P \ t_{\text{data}} \)
  - Reduce: \( t_{\text{comm}2} = \log_2(P) (t_{\text{start}} + t_{\text{data}}) \)
  - Total: \( t_{\text{comm}} = 2 \log_2(P) t_{\text{start}} + (n(P-1)/P + \log_2(P)) \ t_{\text{data}} \)

- **Computation time**
  - Local sum: \( t_{\text{comp}1} = n/P \)
  - Global sum: \( t_{\text{comp}2} = \log_2(P) \)
  - Total: \( t_{\text{comp}} = n/P + \log_2(P) \)

- **Speedup**, assuming \( t_{\text{startup}} = 0 \)
  - Sequential time: \( t_s = n-1 \)
  - Parallel time: \( t_P = (n(P-1)/P + \log_2(P)) \ t_{\text{data}} + n/P + \log_2(P) \)
  - Speedup: \( S_P = t_s/t_P = O(n / (n + \log(P))) \)
  - Best speedup w/o communication: \( S_P = O(P/\log(P)) \)
General M-Ary Partitioning

Example: partitioning an image, e.g. to compute histogram in parallel

3-level 4-ary partitioning for $4^3 = 64$ processors
Partitioning Example 2: Parallel Bucket Sort

- Bucket sort of a list of values bounded within a range \([lo...hi]\)

1. Partition \(n\) values in \(n/P\) segments
2a. Sort each segment into \(P\) small buckets (local computation)
2b. Send content of small buckets to \(P\) large buckets
3. Sort \(P\) large buckets and merge lists

![Diagram of Parallel Bucket Sort Process]
Partitioning Example 2: Parallel Bucket Sort

Input: list X of length n with minimum value L and maximum U
Output: sorted list X

def function bucket(x) = P*(x-L)/(U-L);

scatter list X to local X_p lists each of size n/P
forall processors p = 0,...,P-1
  for i = 0,...,n/P-1
    x = X_p[i]
    put x into small bucket b_p[bucket(x)]
all-to-all of small buckets b_p into large buckets B_p
sort values in B_p[0,...,P-1] using a sequential sort algorithm
gather X from B_p into a merged sorted list
Partitioning Example 2: Parallel Bucket Sort

Communication time (assuming uniform distribution in X)
- Scatter: \( t_{comm1} = \log_2(P)t_{startup} + n(P-1)/P \ t_{data} \)
- All-to-all: \( t_{comm2} = (P-1)(t_{startup} + n/P^2 \ t_{data}) \)
- Gather: \( t_{comm3} = \log_2(P)t_{startup} + n(P-1)/P \ t_{data} \)

Computation time (assuming uniform distribution in X)
- Small bucket sort: \( t_{comp1} = n/P \)
- Large bucket sort: \( t_{comp2} = n/P \log_2(n/P) \)

Speedup
- Sequential time: \( t_s = n \log_2(n/P) \) (with \( P \) buckets)
- Parallel time: \( t_P = 2 \log_2(P)t_{startup} + 2 n(P-1)/P \ t_{data} \)
  + \( (P-1)(t_{startup} + n/P^2 \ t_{data}) \)
  + \( n/P (1 + \log_2(n/P)) \)
- Speedup w/o communication: \( S_P = O(P) \)
Partitioning Example 3: 
Barnes Hut Algorithm

\[ F = \frac{Gm_1m_2}{r^2} \]

Direction of the force between two bodies at points \( p \) and \( q \)

\[ \vec{F} = \frac{Gm_pm_q}{r^2} \left( \frac{\vec{p} - \vec{q}}{r} \right) \]

\[ F = ma \]

\[ \vec{F}^t = \frac{m(\vec{v}^{t+\frac{1}{2}} - \vec{v}^{t-\frac{1}{2}})}{\Delta t} \]

\[ \vec{v}^{t+\frac{1}{2}} = \vec{v}^{t-\frac{1}{2}} + \frac{\vec{F}^t \Delta t}{m} \]

\[ \vec{x}^{t+1} = \vec{x}^t + \vec{v}^{t+\frac{1}{2}} \Delta t \]
Partitioning Example 3: Barnes Hut Algorithm

Particles in 2D space

Quadtree

Parent computes $M$ and $C$

Particle at $(x,y)$ and mass $m$

A square w/o particle is deleted

Mass of parent is sum of masses of children

$$M = \sum_{i=0}^{3} m_i$$

Center of mass

$$C = \frac{1}{M} \sum_{i=0}^{3} m_i c_i$$
Partitioning Example 3: Barnes Hut Algorithm

for (t = 0; t < tmax; t++)
{
    Build_tree();
    Compute_Total_Mass_Center();
    Compute_Force();
    Update_Positions();
}

Computational time is $O(n \log n)$

Assuming $P = n$ then $t_P = O(\log P)$

$$C = \frac{1}{M} \sum_{i=0}^{3} m_i c_i \quad (*)$$

$$\vec{F} = \frac{G m_p m_q}{r^2} \left( \frac{\vec{p} - \vec{q}}{r} \right) \quad (**)$$

Comput_Force()
{
    for (i = 0; i < n; i++)
        Compute_Tree_Force(i, root);
}

Compute_Tree_Force(i, node)
{
    if (box at node contains one particle)
        $F =$ force using eq (**) 
    else
    {
        $r =$ distance from $i$ to $C$ (*) of box 
        $D =$ size of box
        if $(D/r < \theta)$
            $F =$ force using eq (**) with total $M$
        else
            for (all children $c$ of box)
                $F =$ $F +$ Compute_Tree_Force($i$, $c$);
    }
    return $F$;
Divide and Conquer

- Divide and conquer strategy (definition by JáJá 1992)
  1. Break up a given problem into independent subproblems
  2. Solve the subproblems recursively and concurrently
  3. Collect and combine the solutions into the overall solution

- In contrast to the partitioning strategy, divide and conquer uses recursive partitioning with concurrent execution to divide the problem down into independent subproblems
- In deeper levels of recursion the number of active processors may increase or decrease
Divide & Conquer Example 1: Parallel Recursive Matmul

- Block matrix multiplication in recursion by decomposing matrix in 2×2 submatrices and computing the submatrices recursively

```cpp
Mat matmul(Mat A, Mat B, int s)
{
    if (s == 1)
        C = A * B;
    else
    {
        s = s/2;
        P0 = matmul(A_p,p, B_p,p, s);
        P1 = matmul(A_p,q, B_q,p, s);
        P2 = matmul(A_p,p, B_p,q, s);
        P3 = matmul(A_p,q, B_q,q, s);
        P4 = matmul(A_q,p, B_p,p, s);
        P5 = matmul(A_q,q, B_q,p, s);
        P6 = matmul(A_q,p, B_p,q, s);
        P7 = matmul(A_q,q, B_q,q, s);
        C_p,p = P0 + P1;
        C_p,q = P2 + P3;
        C_q,p = P4 + P5;
        C_q,q = P6 + P7;
    }
    return C;
}
```

- Level of parallelism increases with deepening recursion
- Suitable for shared memory systems
Divide and Conquer Example 2: Parallel Convex Hull Algorithm

- The planar convex hull of a set of points $S=\{p_1, p_2, \ldots, p_n\}$ of $p_i=(x, y)$ coordinates is the smallest convex polygon that encompasses all points $S$ on the $x$-$y$ plane.
Divide and Conquer Example 2: Parallel Convex Hull Algorithm

- The *upper convex hull* spans points \( \{q_1, \ldots, q_s\} \subseteq S \) from point \( q_1 \) with minimum \( x \) to \( q_s \) with maximum \( x \).
- The *convex hull* = *upper convex hull* + lower convex hull
- Problem:
  - Given points \( S = \{p_1, \ldots, p_n\} \) such that \( x(p_1) < x(p_2) < \ldots < x(p_n) \), construct the upper convex hull in parallel.

![Upper convex hull diagram](image-url)
Divide and Conquer Example 2: Parallel Convex Hull Algorithm

- Parallel convex hull:
  1. Divide the $x$-sorted points $S$ into sets $S_1$ and $S_2$ of equal size
  2. Compute upper convex hull recursively on $S_1$ and $S_2$
  3. Combine $UCH(S_1)$ and $UCH(S_2)$ by computing the upper common tangent to form $UCH(S)$

![Diagram of upper common tangent]
Divide and Conquer Example 2: Parallel Convex Hull Algorithm

- Base case of recursion: two points, which are returned as $UCH(S)$
- The line segment $(a,b)$ can be computed sequentially in $O(\log n)$ time with $n = |UCH(S_1) + UCH(S_2)|$ using a binary search method
- Line segments can be implemented as linked list of points, thus $UCH(S_1)$ and $UCH(S_2)$ can be connected using one pointer change of $a$ to point to $b$ in $O(1)$ time

- Parallel convex hull time complexity recurrence relation:
  \[ T(n) \leq T(n/2) + a \log n \]
  with solution:
  \[ T(n) = O(\log^2 n) \]
- Parallel convex hull operations recurrence relation:
  \[ W(n) \leq 2W(n/2) + b n \]
  with solution:
  \[ W(n) = O(n \log n) \]
  which is cost optimal, since sequential algorithm is $O(n \log n)$
Divide and Conquer Example 3: First-Order Linear Recurrences

- First-order linear recurrence
  
  \[ y_1 = b_1 \]
  
  \[ y_i = a_i y_{i-1} + b_i \quad 2 \leq i \leq n \]

- Example applications:
  
  - Prefix sum \( y_i = \sum_{j=1..i} b_j \) is a special case of a first-order linear recurrence with \( a_i = 1 \) (the multiplicative unit element)
  
  - \( n \)-th order polynomial evaluation using Horner's rule
    
    \[ p(x) = (((b_1 x + b_2) x + b_3) x + \ldots + b_{n-1}) x + b_n \]

    is a special case of a first-order linear recurrence with \( a_i = x \)

  - Solving a bi-diagonal system \( By = c \), let
    
    \[ a_i = - \frac{l_i}{d_i} \]
    
    \[ b_i = \frac{c_i}{d_i} \]

    then solve linear recurrence to obtain solution \( y \)
Divide and Conquer Example 3: First-Order Linear Recurrences

- Rewrite $y_i = a_i y_{i-1} + b_i$ into $y_i = a_i (a_{i-1} y_{i-2} + b_{i-1}) + b_i$
- This equation defines a linear recurrence of size $n/2$ for even index $i$

$$
\begin{align*}
z_1 &= b_1' \\
z_i &= a_i' z_{i-1} + b_i' & 2 \leq i \leq n/2
\end{align*}
$$

1. Let
   
   $a_i' = a_{2i} a_{2i-1}$

   $b_i' = a_{2i} b_{2i-1} + b_{2i}$

2. Solve $z_i$ recursively

3. For $1 \leq i \leq n$ set
   
   $y_i = z_{i/2}$ if $i$ is even
   
   $y_i = a_i z_{(i-1)/2} + b_i$ if $i$ is odd $> 1$
   
   $y_i = b_1$ if $i = 1$
Divide and Conquer Example 3: First-Order Linear Recurrences

Parallel algorithm:

```
linrecsolve(a[], b[], y[], n)
{
    if (n==1)
        { y[1] = b[1];
            return;
        }
    forall (i = 1 to n/2)
        { a\new[i] = a[2*i]*a[2*i-1];
            b\new[i] = a[2*i]*b[2*i-1]+b[2*i];
        }
    linrecsolve(a\new, b\new, z, n/2);
   forall (i = 1 to n)
        { if (i == 1)
            y[1] = b[1];
        else if (even(i))
            y[i] = z[i/2];
        else
            y[i] = a[i]*z[(i-1)/2]+b[i];
        }
}
```

\[
b_1'
= a_2 b_1 + b_2
\]
\[
b_1''
= a_2' b_1' + b_2'
= ((a_2 b_1 + b_2) a_3 + b_3) a_4 + b_4
\]
\[
b_1'''
= a_2'' b_1'' + b_2''
= (((a_2' b_1' + b_2') a_3' + b_3') a_4' + b_4')
= (((((a_2 b_1 + b_2) a_3 + b_3) a_4 + b_4) a_5 + b_5) a_6 + b_6) a_7 + b_7) a_8 + b_8
\]
Divide and Conquer Example 4: Triangular Matrix Inversion

- Consider \( Ax = b \) with triangular matrix \( A \)

\[
\begin{pmatrix}
a_{11} & \cdots & a_{1n} \\
a_{21} & \cdots & a_{2n} \\
a_{31} & a_{32} & \cdots & a_{3n} \\
\vdots & \vdots & \ddots & \vdots \\
a_{n1} & a_{n2} & \cdots & a_{nn}
\end{pmatrix}
\]

- Partition \( A \) into \((n/2) \times (n/2)\) blocks

\[
\begin{pmatrix}
A_1 \\
A_2 \\
A_3
\end{pmatrix}
\]

- Then \( A^{-1} \) is given by

\[
\begin{pmatrix}
A_1^{-1} & 0 \\
-A_3^{-1}A_2A_1^{-1} & A_3^{-1}
\end{pmatrix}
\]
Divide and Conquer Example 4: Triangular Matrix Inversion

- Parallel algorithm:
  1. Divide $A$ into $A_1$, $A_2$, $A_3$
  2. Recursively compute inverses of $A_1$ and $A_3$ in parallel
  3. Multiply $-A_3^{-1}A_2A_1^{-1}$ and combine with $A_1^{-1}$ and $A_3^{-1}$ to get $A^{-1}$

- Time complexity is given by the recurrence relation
  \[ T(n) = T(n/2) + cn \]
  with $P=n^2$ processors to compute $-A_3^{-1}A_2A_1^{-1}$ in $O(n)$ operations in parallel, thus $T(n) = O(n)$ time
Divide and Conquer Example 5: Banded Triangular Systems

- Consider $Ax = b$ with banded matrix $A$ with $m=3$

$$
\begin{pmatrix}
  a_{11} & a_{21} & a_{31} & a_{42} & a_{53} & a_{64} & a_{75} & a_{86} & a_{97} \\
  a_{21} & a_{22} & a_{32} & a_{43} & a_{54} & a_{65} & a_{76} & a_{87} & a_{98} \\
  a_{31} & a_{32} & a_{33} & a_{44} & a_{55} & a_{66} & a_{77} & a_{88} & a_{99} \\
  a_{42} & a_{43} & a_{44} & a_{53} & a_{64} & a_{75} & a_{86} & a_{97} & a_{98} \\
  a_{53} & a_{54} & a_{55} & a_{65} & a_{76} & a_{87} & a_{98} & a_{99} & a_{99} \\
  a_{64} & a_{65} & a_{66} & a_{75} & a_{86} & a_{97} & a_{98} & a_{99} & a_{99} \\
  a_{75} & a_{76} & a_{77} & a_{86} & a_{97} & a_{98} & a_{99} & a_{99} & a_{99} \\
  a_{86} & a_{87} & a_{88} & a_{98} & a_{99} & a_{99} & a_{99} & a_{99} & a_{99} \\
  a_{97} & a_{98} & a_{99} & a_{99} & a_{99} & a_{99} & a_{99} & a_{99} & a_{99}
\end{pmatrix}
$$

- Define block diagonal $D$ and inverse $D^{-1}$

$$
D = \begin{pmatrix}
  A_{11} & A_{22} & \cdots \\
  \cdots & \cdots & \cdots \\
  \cdots & \cdots & A_{n/m,n/m}
\end{pmatrix}
$$

$$
D^{-1} = \begin{pmatrix}
  A_{11}^{-1} & A_{22}^{-1} & \cdots \\
  \cdots & \cdots & \cdots \\
  \cdots & \cdots & A_{n/m,n/m}^{-1}
\end{pmatrix}
$$
Divide and Conquer Example 5: Banded Triangular Systems

- Compute $d = D^{-1}b$ and $B = D^{-1}A$ where $B_{i,i-1} = A_{ii}^{-1}A_{i,i-1}$

$$d = D^{-1}b = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ \vdots \\ d_{n/m} \end{pmatrix}, \quad \text{B} = D^{-1}A = \begin{pmatrix} I_m & I_m & I_m \\ B_{21} & I_m & \vdots \\ \vdots & \ddots & \vdots \\ & & B_{n/m,n/m-1} & I_m \end{pmatrix}$$

- Solve first-order linear recurrence on $m \times m$ matrices $B_{i,i-1}$

$$x_1 = d_1$$
$$x_i = -B_{i,i-1}x_{i-1} + d_1 \quad 2 \leq i \leq n/m$$

- Parallel time $O(m + m \log(n/m))$ with $P=nm$ processors
  - Compute all $A_{ii}^{-1}$ (each requiring $O(m)$ operations) in parallel with parallel matrix inversion algorithm
  - Compute all $B_{i,i-1} = A_{ii}^{-1}A_{i,i-1}$ in $O(m)$ operations in parallel
  - Recurrence depth is $\log_2(n/m)$, each step has $O(m)$ operations
Divide and Conquer Example 6: LU of Tridiagonal Matrix

Consider tridiagonal matrix LU decomposition

\[
\begin{pmatrix}
a_1 & c_1 \\
b_2 & a_2 & c_2 \\
& b_3 & a_3 & c_3 \\
& & \ddots & \ddots & \ddots \\
& & & b_n & a_n
\end{pmatrix}
= \begin{pmatrix} 1 \\ l_2 & 1 \\ & l_3 & 1 \\ & & \ddots & \ddots & 1 \\ & & & & l_n & 1 \\ & & & & & d_n
\end{pmatrix}
\begin{pmatrix}
d_1 & u_1 \\
d_2 & d_2 & u_2 \\
& d_3 & d_3 & u_3 \\
& \ddots & \ddots & \ddots & \ddots \\
& \ddots & \ddots & \ddots & \ddots & \ddots \\
& \ddots & \ddots & \ddots & \ddots & \ddots \\
& \ddots & \ddots & \ddots & \ddots & \ddots \\
& \ddots & \ddots & \ddots & \ddots & \ddots \\
& \ddots & \ddots & \ddots & \ddots & \ddots \\
& & & & & d_n
\end{pmatrix}
\]

The LU decomposition \( A = L \, U \) satisfies

\[a_1 = d_1\]
\[c_i = u_i\]
\[a_i = d_i + l_i u_{i-1}\]
\[b_i = l_i d_{i-1}\]

thus

\[d_1 = a_1\]
\[d_i = a_i - l_i u_{i-1} = a_i - u_{i-1} b_i / d_{i-1} = \left[ a_i \, d_{i-1} - b_i c_{i-1} \right] / d_{i-1}\]
Divide and Conquer Example 6: LU of Tridiagonal Matrix

Let

\[
\begin{align*}
R_1 &= \begin{bmatrix} a_1 & 0 \\ 1 & 0 \end{bmatrix} & R_i &= \begin{bmatrix} a_i & -b_i c_{i-1} \\ 1 & 0 \end{bmatrix} & T_i &= R_i R_{i-1} \ldots R_1
\end{align*}
\]

From the Möbius transformation we have

\[
d_i = \frac{\begin{bmatrix} 1 \\ 0 \end{bmatrix}^T T_i \begin{bmatrix} 1 \\ 1 \end{bmatrix}}{\begin{bmatrix} 0 \\ 1 \end{bmatrix}^T T_i \begin{bmatrix} 1 \\ 1 \end{bmatrix}}
\]

Algorithm:

- Set up matrices \( R \)
- Solve first-order linear recurrence (prefix sum) of \( T \)
- Compute \( d_i \)
- From the solution of \( d_i \) compute \( l_i = b_i / d_{i-1} \)
Further Reading

- [PP2] pages 106-131
- [PSC] pages 321-337