Vector Machines
Model for Parallel Computation

Bryan Petzinger
Adv. Theory of Computing
What is Parallel Computation?

- Utilizing multiple processors simultaneously over the course of a computation.
Example: Sequential vs Parallel Mergesort
Sequential Mergesort

• Given the sequence: 56 45 21 34 78 33 48

• Combine into pairs:
  • 45 56  21 34  33 78  2 48 (four comparisons)

• Combine the pairs into quadruples
  • 21 34 45 56  2 33 48 78 (five comparisons)

• Combine the quadruples into an eight-tuple
  • 2 21 33 34 45 48 56 78
Sequential Mergesort (cont)
Sequential Mergesort (cont)

- For a size $n$ sequence
- Each merge involves fewer than $n$ comparisons $O(n)$
- Require $\log_2 n$ merges
- $O(n \log_2 n)$
Parallel Mergesort

- For the same sequence (size 8) parallel mergesort requires 15 processors
- Essentially one processor for each node of the tree in the previous diagram
Parallel Mergesort (cont)

- Each number is transmitted to one of the eight processors at the top of the tree
- Each of these 8 processors passes the number to a processor below it
- For each of the remaining seven processors
  - Receives one number from each of the two processors above it.
  - Smaller number is passed down and another is request from its parents
- Ultimately the processor at the root communicates the sorted sequence as output
Parallel Mergesort (cont)

- The circle representing each processor contains two numbers.
  - The first is the number of numbers processed.
  - The second is the number of comparisons (worst case).

- It takes 2 comparisons before the root node receives any numbers.
- Afterwards all comparisons are dominated by the root.
- $2 + 7$ comparisons total
- $O(n)$
Vectors and Vector Operations
Vectors

• Binary digit strings
  • ...00001010
  • ...00000111
  • ...11110100
  • ...11110101

• Properties
  • Infinite to the left
  • Constant (repeating 0s or 1s)
Binary Integers

• Vectors have two parts: constant and non-constant

• ...000001000110

• Constant: sign
  • 0's → positive
  • 1's → negative
Binary Integers (cont.)

- Non-constant: magnitude
- ...000001000110 = +70
- ...111111000110 = -6
Unary Integers

- Non-negative integers
- $+1n$
- $U(4) = +14 = 1111$
- $U(0) = +$
Boolean Operations

- Similar to usual definitions of propositional logic
- and, or, not, implication, etc
- $v_1 \rightarrow v_2 = \neg v_1 \lor v_2$
Boolean Operations (cont.)

• $v_1 = \ldots0000110001110$
• $v_2 = \ldots0000010101011$
• $v_1 \& v_2 = \ldots0000010001010$

<table>
<thead>
<tr>
<th>b1</th>
<th>b2</th>
<th>b1 &amp; b2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Shift Operations

- **Left shift**
  - Digits shifted left one position
  - 0 or 1 fills the newly vacant position
  - “v << 0 1” shift left one position, fill with 0s

- **Right shift**
  - Digits shifted right one position
  - Rightmost bit disappears
  - “v >> 3” shift right three positions
Shift Operations (cont)

- Vector $v = \ldots000111001 = +111001$
- $V \ll 0\ 1 = +1110010$
- $V \ll 1\ 1 = +1110011$
- $V \ll 1\ 3 = +111001111$
- $V \gg 4 = +11$
Vector Machines

- Set of registers, each containing a vector
- Instructions to perform on those registers

\[
\begin{align*}
\text{Start} \\
v_3 &= v_1 \oplus v_2 \\
\text{Halt}
\end{align*}
\]
Example

```cpp
if( v1 != +0) {
    v2 = v1 << 1 4
}
```
Example

Input: v1 and v2
   6
Output: v3 = v1 <<1 v2
   +010 = 2

begin
   x = -
   x = x <<0 v2
   x = !x
   v3 = v1 <<0 v2
   v3 = v3 || x
end
Vector Machines and Function Computation
Remarks

- Binary left shift-0 $\rightarrow$ multiplication (powers of 2)
- Unary left shift-1 $\rightarrow$ addition
Example

Input: \( v_1 = +1 \)
Output: \( v_2 = +1 \log_2(n+1) = U(\log_2(n+1)) \)
beg
\( v_3 = v_1 \ll 1 \)
v_2 = +
v_4 = +1
while \( v_3 \gg v_4 \neq + \)
v_2 = v_2 \ll 1 \)
v_4 = v_4 \ll 0 \)
end
end

- Left shift-1
  - Increment uval
  - \( v_2 \) counts number of iterations of while loop

- Left shift-0
  - Double bval
Time Complexity

begin

\[ v3 = v1 \ll 1 \]
\[ v2 = + \]
\[ v4 = +1 \]
\[ \text{while } (v3 >> v4 \neq +) \]

\[ v2 = v2 \ll 1 \]
\[ v4 = v4 \ll 0 \]

end

begin

3 parallel steps

\[ v2 = v2 \ll 1 \]
\[ v4 = v4 \ll 0 \]

2 parallel steps

log(n+1) iterations

• Computes in O(1) plus log2(n+1) * 2 steps

• time(n) is O(log2n)
Formal Definition

\( M = \langle V, S \rangle \) be a vector machine, and suppose that \( f \) is a \( k \)-ary partial number-theoretic function, for some fixed \( k \geq 0 \). Then we shall say that \( M \) computes \( f \) provided that:

- If \( M \)'s registers \( v_1, v_2, \ldots, v_k \) initially contain vectors \( U(n_1) = +1n_1, U(n_2) = +1n_2, \ldots, U(n_k) = +1n_k \), all other registers contain vector +; and if \( f(n_1, n_2, \ldots, n_k) \) is defined, then, upon termination, registers \( v_1, v_2, \ldots, v_k \) contain vectors \( U(n_1) = +1n_1, U(n_2) = +1n_2, \ldots, U(n_k) = +1n_k \) as before and register \( v_{k+1} \) contains \( U(f(n_1, n_2, \ldots, n_k)) = +1f(n_1, n_2, \ldots, n_k) \).

- If \( M \)'s registers \( v_1, v_2, \ldots, v_k \) initially contain vectors \( U(n_1) = +1n_1, U(n_2) = +1n_2, \ldots, U(n_k) = +1n_k \), all other registers contain vector +; and if \( f(n_1, n_2, \ldots, n_k) \) is undefined, then either \( M \) never terminates or, if \( M \) does terminate, then the final contents of \( v_{k+1} \) are not of the form \( +1m \).
Formal Definition (cont)

- First case: $f$ is defined. $M$ terminates with original registers unchanged and the result of $f$ is stored in an additional register.
- Second case: $f$ is undefined. $M$ never terminates, or $M$ terminates but doesn't store $+1m$ into $v_k+1$. 
Unary to Binary

- Formal definition defined in terms of unary vectors
- We want to include operations on binary vectors
- Need a method to convert unary to binary, and vice versa.
Unary to Binary converter

• Function \( B(n) \)
• \( B(8) = +1000 \)
• \( B(27) = +11011 \)
• \( |B(8)| = 4 = \log_2(8+1) \)
• \( |B(27)| = 5 = \log_2(27+1) \)
• \( |B(n)| = \log_2(n+1) \)
Input: $v_1 = +1n = U(n)$ for $n \geq 0$
Output $v_2 = +amam-1...a2a1a0 = B(n)$, where $m = |B(n)| - 1$

begin

$v_3 = +1\log_2(n+1) = +1|B(n)|$  
/// vector machine of previous example

$v_4 = +1$

while ($v_3 \neq +$)
    $v_4 = v_4 << 0 \ 1$
    $v_3 = v_3 >> 1$
end

$v_5 = v_1$
$v_4 = v_4 >> 1$  
/// $v_4 = B(2m)$
$v_5 = v_1 << 1 \ 1$  
/// $v_5 = +1n+1$

while ($v_4 \neq +$)
    if ($v_5 >> v_4 \neq +$)  
        /// is diff greater than power of 2 under consideration
        $v_2 = v_2 << 1 \ 1$  
        /// left-shift-1
        $v_5 = v_5 >> v_4$
    end
else
    $v_2 = v_2 << 0 \ 1$  
    /// left-shift-0
end
while (v4 != +)
    if (v5 >> v4 != +) //is diff greater than power of 2 under consideration
        v2 = v2 << 1 1 //left-shift-1
    end
    v5 = v5 >> v4
end
else v2 = v2 << 0 1 //left-shift-0
    v4 = v4 >> 1 //v4 becomes next smaller power of 2
end
...

<table>
<thead>
<tr>
<th>Input vector U(27) = +127</th>
<th>Output Register v2</th>
<th>Auxiliary Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>initialization</td>
<td>+</td>
<td>+10000</td>
</tr>
<tr>
<td>While loop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Iteration 1</td>
<td>+1</td>
<td>+1000</td>
</tr>
<tr>
<td>Iteration 2</td>
<td>+11</td>
<td>+100</td>
</tr>
<tr>
<td>Iteration 3</td>
<td>+110</td>
<td>+10</td>
</tr>
<tr>
<td>Iteration 4</td>
<td>+1101</td>
<td>+1</td>
</tr>
<tr>
<td>Iteration 5</td>
<td>+11011 = B(27)</td>
<td>+a</td>
</tr>
</tbody>
</table>
Masks

- Mask of index \( i \)
- Positive vector of length \( 2i \)
- For each \( i \geq 0 \), there will be \( i \) distinct masks
- Ex. \( i = 4 \) results in four masks of index 4, each of length 16
  - \( M_{40} = +1010101010101010 \)
  - \( M_{41} = +1100110011001100 \)
  - \( M_{42} = +1111000011110000 \)
  - \( M_{43} = +1111111100000000 \)
Example: Reverse Word
Considerations

• ASCII-style encoding to represent alphabet
  • Our alphabet is \{a,b\}
  • Cardinality 2, requires only 1 bit per character
  • 0 = a
  • 1 = b
Considerations (cont)

- Problematic scenario:
  - Input word = abb
  - Input vector = +011 = +11 = bb
- Require 2 input vectors to avoid this issue
  - 1st vector will be the encoded word
  - 2nd vector will be the words length in unary
Using Masks

- Suppose 8 bit word
  - $2^k = 8; k = 3$
- Generate $k$ 8-bit masks
  - $m_{30} = +10101010$
  - $m_{31} = +11001100$
  - $m_{32} = +11110000$
Algorithm

- \( w = a1a2a3a4a5a6a7a8 \)
- Apply masks to word
  - \( w = (w \& m3k) \gg 2k || (w \& !m3k) \ll 0 \)
- Coarsest mask first
  - \( m32 \rightarrow a5a6a7a8a1a2a3a4 \)
  - \( m31 \rightarrow a7a8a5a6a3a4a1a2 \)
  - \( m30 \rightarrow a8a7a6a5a4a3a2a1 \)
Pseudocode

Input: \( v_1 = +\text{Trans}(w) = +a_1a_2...a_n \), where \( n = 2^k \)
\[ v_2 = U(|\text{Trans}(w)|) = +1^n \]

Output: \( v_3 = +\text{Trans}(wR) = a_n...a_1 \)

begin
  \[ y = \text{convert}(v_2) >> 1 = +10^{k-1} \]
  \[ x = m_{k,k-1} \]
  \[ v_3 = v_1 \]
  \[ v_3 = (v_3 & x) >> y || (v_3 & !x) << 0 \] \( Y \)
  \[ x = x \oplus (x >> y) \]
end

end
Time & Space

- Linear space - $O(n)$
- Logarithmic time - $O(\log_2 n)$ parallel steps
- Single-tape Turing machine is $O(n^2)$
- Two-tape Turing machine is $O(n)$
- Difference is *parallel steps*
- Same amount of work, but faster
Vector Machines and Parallel Computation
Parallelism

- Single processor responsible for each bit in non-constant portion of vector
- E.g. processor p1 would be responsible for the rightmost bit, p2 for the next bit, and so on.
Example

- \( v_k = v_i \oplus v_j \)
  - \( v_i = +1010 \)
  - \( v_j = +1101 \)
- Involves processors \( p_1, p_2, p_3, p_4 \)
- Each processor has its own local memory & dedicated channels for interacting with that memory
Remarks

- Requires an infinite number of processors to be available
- Finitely many active at once
Vector Machines and Formal Languages
Language Acceptance

Ex. Palindromes:
Input: +Trans(w), U(|Trans(w)|)
Output: v3 { +1 if w = wR, +0 otherwise}
begin
  v4 = reverse_word(v1, v2)
  if (v1 ⊕ v4 == +)
    v3 = +1
  else v3 = +0
end
Ex. $L = \{a^{2^i} \mid i \geq 0\}$

Input:
$v_1 = +\text{Trans}(w) = +1n$
$v_2 = U(|w|) = +1n$

Output:
$V_3 = \{ +1 \text{ if } w = a^{2^i} \text{ for } i \geq 0, +0 \text{ otherwise} \}$
begin
  v4 = convert(v1)  
  converter
  //unary to binary

v5 = +1

if (v4 == +)  
  //empty word not in L
  v3 = +0
else if (v4 == v5)  
  //word a is in L
  v3 = +1
else
  v6 = v4
  while (v6 != + && v5 != v4)
    v5 = v5 << 0 1  
    //double v5
    v6 = v6 >> 1
  end
  if (v5 == v4)
    v3 = +1
  else
    v3 = +0
  end