CS 267

Tricks with Trees

James Demmel

www.cs.berkeley.edu/~demmel/cs267_Spr14

Outline

° A log n lower bound to compute any function in parallel
° Reduction and broadcast in O(log n) time
° Parallel prefix (scan) in O(log n) time
° Adding two n-bit integers in O(log n) time
° Multiplying n-by-n matrices in O(log n) time
° Inverting n-by-n triangular matrices in O(log^2 n) time
° Inverting n-by-n dense matrices in O(log^2 n) time
° Evaluating arbitrary expressions in O(log n) time
° Evaluating recurrences in O(log n) time

° "2D parallel prefix", for image segmentation (Bryan Catanzaro, Kurt Keutzer)
° Sparse-Matrix-Vector-Multiply (SpMV) using Segmented Scan
° Parallel page layout in a browser (Leo Meyerovich, Ras Bodik)
° Solving n-by-n tridiagonal matrices in O(log n) time
° Traversing linked lists
° Computing minimal spanning trees
° Computing convex hulls of point sets...
A log n lower bound to compute any function of n variables

- Assume we can only use binary operations, one per time unit.
- After 1 time unit, an output can only depend on two inputs.
- Use induction to show that after k time units, an output can only depend on $2^k$ inputs.
  - After $\log_2 n$ time units, output depends on at most n inputs.
- A binary tree performs such a computation.

Broadcasts and Reductions on Trees

Parallel Prefix, or Scan

- If "+" is an associative operator, and x[0], ..., x[p-1] are input data then parallel prefix operation computes:
  $$y[j] = x[0] + x[1] + \ldots + x[j]$$ for $j=0,1,\ldots,p-1$
- Notation: $j:k$ means $x[j]+x[j+1]+\ldots+x[k]$, blue is final value.

Mapping Parallel Prefix onto a Tree - Details

- Up-the-tree phase (from leaves to root):
  1) Get values L and R from left and right children.
  2) Save L in a local register Lsave.
  3) Pass sum L+R to parent.
- By induction, Lsave = sum of all leaves in left subtree.
- Down the tree phase (from root to leaves):
  1) Get value S from parent (the root gets 0).
  2) Send S to the left child.
  3) Send S + Lsave to the right child.
- By induction, S = sum of all leaves to left of subtree rooted at the parent.
E.g., Fibonacci via Matrix Multiply Prefix

\[ F_{n+1} = F_n + F_{n-1} \]

\[
\begin{pmatrix}
    F_{n+1} \\
    F_n
\end{pmatrix} =
\begin{pmatrix}
    1 & 1 \\
    1 & 0
\end{pmatrix}
\begin{pmatrix}
    F_n \\
    F_{n-1}
\end{pmatrix}
\]

Can compute all \( F_n \) by matmul_prefix on \[ \left[ \begin{array}{cccccc}
0 & 1 & 1 & 1 & 1 & 1 \\
1 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 0 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 1 \\
1 & 1 & 1 & 1 & 1 & 0
\end{array} \right] \]
then select the upper left entry

---

Adding two n-bit integers in \( O(\log n) \) time

Let \( a = a[n-1]a[n-2]\ldots a[0] \) and \( b = b[n-1]b[n-2]\ldots b[0] \) be two n-bit binary numbers

- We want their sum \( s = a + b = s[n]s[n-1]\ldots s[0] \)
  - \( c[0] = 0 \) ... rightmost carry bit
  - \( c[i] = (a[i] \text{ xor } b[i]) \text{ or } (a[i] \text{ and } b[i]) \ldots \text{ next carry bit} \)
  - \( a[i] = (a[i] \text{ xor } b[i]) \text{ xor } c[i-1] \)

Challenge: compute all \( c[i] \) in \( O(\log n) \) time via parallel prefix

for all (0 <= i <= n-1) \( p[i] = a[i] \text{ xor } b[i] \) ... propagate bit
for all (0 <= i <= n-1) \( g[i] = a[i] \text{ and } b[i] \) ... generate bit

\[
\begin{pmatrix}
    c[0] \\
    c[1]
\end{pmatrix} =
\begin{pmatrix}
    p[0] & g[0] \\
    0 & 1
\end{pmatrix}
\begin{pmatrix}
    c[0] \\
    c[1]
\end{pmatrix} =
\begin{pmatrix}
    c[0] \text{ xor } c[1] & c[0] \text{ and } c[1] \\
    0 & 1
\end{pmatrix}
\]

... 2-by-2 Boolean matrix multiplication (associative)

\[
\begin{pmatrix}
    C[0] \\
    C[1]
\end{pmatrix} =
\begin{pmatrix}
    C[0] \text{ xor } C[1] & C[0] \text{ and } C[1] \\
    0 & 1
\end{pmatrix}
\]

... evaluate each \( P[i] = C[0] \text{ xor } C[1] \text{ xor } \ldots \text{ xor } C[0] \) by parallel prefix

- Used in all computers to implement addition - Carry look-ahead

Other applications of scan = parallel prefix

- There are many applications of scans, some more obvious than others
  - add multi-precision numbers (represented as array of numbers)
  - evaluate recurrences, expressions
  - solve tridiagonal systems (but numerically unstable!)
  - implement bucket sort and radix sort
  - to dynamically allocate processors
  - to search for regular expression (e.g., grep)
- Names: +\ (APL), cumsum (Matlab), MPI_SCAN
- Note: 2n operations used when only n-1 needed

Multiplying n-by-n matrices in \( O(\log n) \) time

For all (1 <= i,j,k <= n) \( P(i,j,k) = A(i,k) \times B(k,j) \)

- cost = 1 time unit, using \( n^2 \) processors

For all (1 <= i,j <= n) \( C(i,j) = \sum_{k=1}^{n} P(i,j,k) \)

- cost = \( O(\log n) \) time, using \( n^2 \) trees with \( n^2 / 2 \) processors
Inverting triangular n-by-n matrices in $O(\log^2 n)$ time

- Fact: \[
\begin{bmatrix} A & 0 \\ C & B \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} & 0 \\ -B^{-1}CA^{-1} & B^{-1} \end{bmatrix}
\]

- Function $\text{Tri}_\text{Inv}(T)$
  - assume $n = \dim(T) = 2^m$ for simplicity
  - time($\text{Tri}_\text{Inv}(n)$) = time($\text{Tri}_\text{Inv}(n/2)$) + $O(\log(n))$

  - Change variable to $m = \log n$ to get time($\text{Tri}_\text{Inv}(n)$) = $O(\log^2 n)$

Inverting Dense n-by-n matrices in $O(\log^2 n)$ time

- Lemma 1: Cayley-Hamilton Theorem
  - expression for $A^{-1}$ via characteristic polynomial in $A$

- Lemma 2: Newton's Identities
  - Triangular system of equations for coefficients of characteristic polynomial, where matrix entries = $s_k$

- Lemma 3: $s_k = \text{trace}(A^k) = \sum A^k_{ii}$

  - Csanky's Algorithm (1976)
    - 1) Compute the powers $A^2, A^3, \ldots, A^{n-1}$ by parallel prefix
      - cost = $O(\log^2 n)$
    - 2) Compute the traces $s_k = \text{trace}(A^k)$
      - cost = $O(\log n)$
    - 3) Solve Newton identities for coefficients of characteristic polynomial
      - cost = $O(\log^2 n)$
    - 4) Evaluate $A^{-1}$ using Cayley-Hamilton Theorem
      - cost = $O(\log n)$

  - Completely numerically unstable

Evaluating arbitrary expressions

- Let $E$ be an arbitrary expression formed from $+, -, *, /, ()$, and $n$ variables, where each appearance of each variable is counted separately

- Can think of $E$ as arbitrary expression tree with $n$ leaves (the variables) and internal nodes labeled by $+, -, \ast, /$ and $0$

  - Theorem (Brent): $E$ can be evaluated in $O(\log n)$ time, if we reorganize it using laws of commutativity, associativity and distributivity

  - Sketch of (modern) proof: evaluate expression tree $E$ greedily by repeatedly
    - collapsing all leaves into their parents at each time step
    - evaluating all "chains" in $E$ with parallel prefix

Evaluating recurrences

- Let $x_i = f(x_{i-1})$, $f_i$ a rational function, $x_0$ given

  - How fast can we compute $x_n$?

  - Theorem (Kung): Suppose degree($f_i$) = $d$ for all $i$
    - If $d=1$, $x_n$ can be evaluated in $O(\log n)$ using parallel prefix
    - If $d>1$, evaluating $x_n$ takes $\Omega(n)$ time, i.e. no speedup is possible

  - Sketch of proof when $d=1$
    - $x_n = \{a_i x_{n-1} + b_i \} / \{c_i x_{n-1} + d_i \}$ can be written as
      \[
      \begin{bmatrix} \text{num} \\ \text{den} \end{bmatrix} = \begin{bmatrix} a_i & b_i \\ c_i & d_i \end{bmatrix} \begin{bmatrix} \text{num}_{n-1} \\ \text{den}_{n-1} \end{bmatrix}
      \]
    - Can use parallel prefix with 2-by-2 matrix multiplication

  - Sketch of proof when $d>1$
    - degree($x_i$) as a function of $x_n$ is $d^i$
    - After $i$ parallel steps, degree(anything) $\geq 2^i$
    - Computing $x_i$ takes $\Omega(2^i)$ steps
Image Segmentation (1/4)
- Contours are subjective – they depend on perspective
  - Surprise: Humans agree (somewhat)
- Goal: generate contours automatically
  - Use them to break images into separate segments (subimages)
  - J. Malik’s group has leading algorithm
  - Enable automatic image search and retrieval ("Find all the pictures with Fred")

Image Segmentation (2/4)
- Think of image as matrix \( A(i,j) \) of pixels
  - Each pixel has separate R(ed), G(reen), B(lue) intensities
- Bottleneck (so far) of Malik’s algorithm is to compute other matrices indicating whether pixel \((i,j)\) likely to be on contour
  - Ex: \( C(i,j) = \) average *R intensity* of pixels in rectangle above \((i,j)\) – average *R intensity* of pixels in rectangle below \((i,j)\)
  - \( C(i,j) \) large for pixel \((i,j)\) marked with \( \bullet \), so \((i,j)\) likely to be on contour

- Algorithm eventually computes eigenvectors of sparse matrix with entries computed from matrices like \( C \)
  - Analogous to graph partitioning in later lecture

Image Segmentation (3/4)
- New Bottleneck: Given \( A(i,j) \), compute \( S(i,j) \) where
  - \( S(i,j) = \) sum of \( A(i,j) \) for \( r \leq i \) and \( s \leq j \)
  - \( S(r,s) = \) sum of \( A(r,s) \) for \( r \leq i \) and \( s \leq j \)
  - \( S(i,j) = \sum S(r,s) \) for \( r \leq i \) and \( s \leq j \)

- "2 dimensional parallel prefix"
  - Do parallel prefix independently on each row of \( A(i,j) \):
    - \( S_{row}(i,j) = \sum A(r,s) \) for \( s \leq j \)
  - Do parallel prefix independently on each column of \( S_{row} \):
    - \( S(i,j) = \sum S_{row}(r,j) \) for \( r \leq i \)

Image Segmentation (4/4)
- Approach (Bryan Catanzaro)
  - Compute \( S(i,j) = \sum A(r,s) \) for \( r \leq i \) and \( s \leq j \)
  - Then sum of \( A(i,j) \) over any rectangle (\( l_{row} \leq i \leq r_{row} \) and \( l_{col} \leq j \leq r_{col} \)) is
    \[
    S(l_{row}, r_{row}) - S(l_{row}, l_{col} - 1) - S(r_{row}, r_{col} - 1) + S(l_{row}, l_{col} - 1)
    \]
Sparse-Matrix-Vector-Multiply (SpMV) \( y = A \cdot x \)

Using Segmented Scan (SegScan)

* Segscan computes prefix sums of arbitrary segments
  
  Segscan \([3, 1, 4, 5, 6, 1, 2, 3]\),

  \[ \begin{bmatrix} T, F, F, T, F, T \end{bmatrix} \]

  \[ = \begin{bmatrix} 3, 4, 5, 6, 7, 9 \end{bmatrix} \]

* Use CSR format of Sparse Matrix \( A \), store \( x \) densely

  \[
  \begin{array}{c|c}
  1 & 2 \\
  0 & 4 \\
  2 & 0 \\
  3 & 0 \\
  \end{array}
  \]

  \[
  \begin{array}{c|c}
  \text{Val} & 1 \ 2 \ 3 \ 4 \ 5 \ 3 \\
  \end{array}
  \]

  \[
  \begin{array}{c|c}
  \text{A} & 0 \ 0 \ 0 \ 5 \\
  \text{Col}_\text{Ind} & 1 \ 3 \ 4 \ 5 \ 1 \ 5 \\
  \end{array}
  \]

  \[
  \begin{array}{c|c}
  \text{Row}_\text{Ptr} & 1 \ 4 \ 7 \ 9 \\
  \end{array}
  \]

* Create array \( P \) of all nonzero \( A(\text{i,j}) \cdot x(j) = \text{Val(k)} \cdot x(\text{Col}_\text{Ind}(k)) \)

* Create array \( S \) showing where segments (rows) start

  \[
  \begin{array}{c|c}
  \text{P} & 7 \ 4 \ 3 \ 14 \ 32 \ 15 \ 3 \\
  \end{array}
  \]

  \[
  \begin{array}{c|c}
  \text{S} & T \ F \ F \ T \ F \ T \ F \ F \\
  \end{array}
  \]

* Extract \( A \cdot x \) as:

  \[
  \begin{bmatrix}
  14 & 61 & 24 \\
  \end{bmatrix}
  \]

* Ras Bodik, Leo Meyerovich

---

Summary of tree algorithms

* Lots of problems can be done quickly - in theory - using trees

* Some algorithms are widely used
  - broadcasts, reductions, parallel prefix
  - carry look ahead addition

* Some are of theoretical interest only
  - Csanky's method for matrix inversion
  - Solving tridiagonal linear systems (without pivoting)
  - Both numerically unstable
  - Csanky needs too many processors

* Embedded in various systems
  - MPI, Split-C, Titanium, NESL, other languages
  - CM-5 hardware control network

---

Page layout in a browser

* Applying layout rules to html description of a webpage is a bottleneck, scan can help

* Simplest example
  - Given widths \([x_1, x_2, \ldots, x_n]\) of items to display on page, where should each item go?
  - Item \( j \) starts at \( x_1 + x_2 + \ldots + x_{j-1} \)

* Real examples have complicated constraints
  - Defined by general trees, since in html each object to display can be composed of other objects
  - To get location of each object, need to do preorder traversal of tree, "adding up" constraints of previous objects
  - Scan can do preorder traversal of any tree in parallel
    - Not just binary trees

* Ras Bodik, Leo Meyerovich