Today:
- Multithreaded Algs.

COSC 581, Algorithms
March 25, 2014

Many of these slides are adapted from several online sources
Reading Assignments

• Today’s class:
  – Chapter 27.3

• Reading assignment for next class:
  – Chapter 29.1

• Announcement: Exam #2 on Tuesday, April 1
  – Will cover greedy algorithms, amortized analysis
  – HW 6-9
Remember Example from last time?

- Consider a program prototyped on 32-processor computer, but aimed to run on supercomputer with 512 processors.
- Designers incorporated an optimization to reduce run time of benchmark on 32-processor machine, from $T_{32} = 65$ to $T'_{32} = 40$.
- But, can show that this optimization made overall runtime on 512 processors slower than the original! Thus, optimization didn’t help.

### Analysis for 32 processors:

**Original:**
\[
\begin{align*}
T_1 &= 2048 \\
T_\infty &= 1 \\
T_P &= T_1/p + T_\infty \\
\Rightarrow T_{32} &= 2048/32 + 1 = 65
\end{align*}
\]

**Optimized:**
\[
\begin{align*}
T'_{1} &= 1024 \\
T'_{\infty} &= 8 \\
T'_P &= T'_{1}/p + T'_{\infty} \\
\Rightarrow T'_{32} &= 1024/32 + 8 = 40
\end{align*}
\]

### Analysis for 512 processors:

**Original:**
\[
\begin{align*}
T_1 &= 2048 \\
T_\infty &= 1 \\
T_P &= T_1/p + T_\infty \\
\Rightarrow T_{512} &= 2048/512 + 1 = 5
\end{align*}
\]

**Optimized:**
\[
\begin{align*}
T'_{1} &= 1024 \\
T'_{\infty} &= 8 \\
T'_P &= T'_{1}/p + T'_{\infty} \\
\Rightarrow T'_{512} &= 1024/512 + 8 = 10
\end{align*}
\]
Remember Example from last time?

- Consider a program prototyped on 32-processor computer, but aimed to run on supercomputer with 512 processors.
- Designers incorporated an optimization to reduce run time of benchmark on 32-processor machine, from \( T_{32} = 65 \) to \( T'_{32} = 40 \).
- But, can show that this optimization made overall runtime on 512 processors slower than the original! Thus, optimization didn’t help.

**Analysis for 32 processors:**

**Original:**
\[
T_1 = 2048 \\
T_\infty = 1 \\
T_P = T_1/P + T_\infty \\
\Rightarrow T_{32} = 2048/32 + 1 = 65
\]

**Optimized:**
\[
T'_{1} = 1024 \\
T'_{\infty} = 8 \\
T'_P = T'_1/P + T'_\infty \\
\Rightarrow T'_{32} = 1024/32 + 8 = 40
\]

**Analysis for 512 processors:**

**Original:**
\[
T_1 = 2048 \\
T_\infty = 1 \\
T_P = T_1/P + T_\infty \\
\Rightarrow T_{512} = 2048/512 + 1 = 5
\]

**Optimized:**
\[
T'_{1} = 1024 \\
T'_{\infty} = 8 \\
T'_P = T'_1/P + T'_\infty \\
\Rightarrow T'_{512} = 1024/512 + 8 = 10
\]

**Question:** For how many processors do the 2 versions of the program run equally fast?
In-Class Exercise #1

Consider the following procedures:

PROC-1(x)

\[
\begin{align*}
&n = x.length \\
&\text{let } y[1..n] \text{ be a new array} \\
&\text{PROC-SUB}(x, y, 1, n) \\
&\text{return } y
\end{align*}
\]

PROC-SUB(x, y, i, j)

\[
\begin{align*}
&\text{if } i = j \\
&\quad y[i] = x[i] \\
&\text{else } k = [(i + j)/2] \\
&\quad \text{spawn } \text{PROC-SUB}(x, y, i, k) \\
&\quad \text{PROC-SUB}(x, y, k + 1, j) \\
&\quad \text{sync} \\
&\quad \text{parallel for } l = k + 1 \text{ to } j \\
&\quad\quad y[l] = y[k] + y[l]
\end{align*}
\]

What is the work of PROC-1?

What is the span of PROC-1?

What is the parallelism of PROC-1?
In-Class Exercise #2a

Consider the following multithreaded algorithm:

\[ \text{P-FUNC}(X, Y, N) \]

\[ \begin{aligned} \text{if } N &= 1 \\ \text{then } Y[1,1] &\leftarrow X[1,1] \\ \text{Else } & \text{Partition } X \text{ into } 4 \ (N/2) \times (N/2) \text{ submatrices } X_{11}, X_{12}, X_{21}, X_{22} \\ & \text{Partition } Y \text{ into four } (N/2) \times (N/2) \text{ submatrices } Y_{11}, Y_{12}, Y_{21}, Y_{22} \\ & \text{spawn } \text{P-FUNC} \ (X_{11}, Y_{11}, N/2) \\ & \text{spawn } \text{P-FUNC} \ (X_{12}, Y_{21}, N/2) \\ & \text{spawn } \text{P-FUNC} \ (X_{21}, Y_{12}, N/2) \\ & \text{spawn } \text{P-FUNC} \ (X_{22}, Y_{22}, N/2) \\ & \text{sync} \end{aligned} \]

What is the work of P-FUNC?

What is the span of P-FUNC?

What is the parallelism of P-FUNC?
In-Class Exercise #2b

Consider the following revised version of the multithreaded algorithm:

P-FUNC-REV(X, Y, N)

    if N = 1
    then Y[1,1] ← X[1,1]
    else
        Partition X into four (N/2) × (N/2) submatrices X_{11}, X_{12}, X_{21}, and X_{22}
        Partition Y into four (N/2) × (N/2) submatrices Y_{11}, Y_{12}, Y_{21}, and Y_{22}
        spawn P-FUNC-REV (X_{11}, Y_{11}, N/2)
        sync
        spawn P-FUNC-REV (X_{12}, Y_{21}, N/2)
        sync
        spawn P-FUNC-REV (X_{21}, Y_{12}, N/2)
        P-FUNC-REV (X_{22}, Y_{22}, N/2)
        sync

What is the work of P-FUNC-REV?

What is the span of P-FUNC-REV?

What is the parallelism of P-FUNC-REV?
Multithreaded Merge Sort

\[\text{MERGE-SORT'}(A, p, r)\]

\[
\text{if } p < r
\]

\[
q = \lfloor (p + r)/2 \rfloor
\]

\[
\text{spawn } \text{MERGE-SORT'}(A, p, q)
\]

\[
\text{MERGE-SORT'}(A, q +1, r)
\]

\[
\text{sync}
\]

\[
\text{MERGE}(A, p, q, r)
\]

Same as original merge-sort, except we execute the 2 recursive calls in parallel
Multithreaded Merge Sort

\textsc{Merge-Sort}'(A, p, r)

\textbf{if} \( p < r \)

\[ q = \lfloor (p + r)/2 \rfloor \]

\textbf{spawn} \textsc{Merge-Sort}'(A, p, q)

\textsc{Merge-Sort}'(A, q +1, r)

\textbf{sync} \textsc{Merge}(A, p, q, r)

Same as original merge-sort, except we execute the 2 recursive calls in parallel
Multithreaded Merge Sort

**MERGE-SORT′(A, p, r)**

if $p < r$

$$q = \lfloor (p + r)/2 \rfloor$$

spawn **MERGE-SORT′(A, p, q)**

**MERGE-SORT′(A, q +1, r)**

sync

**MERGE(A, p, q, r)**

**Analysis**

**Work:**

$$T_1(n) = 2T_1\left(\frac{n}{2}\right) + \Theta(n)$$

$$= \Theta(n \lg n)$$

**Span:**

**Parallelization:**

Same as original merge-sort, except we execute the 2 recursive calls in parallel.
Multithreaded Merge Sort

\textbf{MERGE-SORT'}(A, p, r)

\begin{align*}
&\text{if } p < r \\
&q = \lfloor (p + r)/2 \rfloor \\
&\text{spawn } \text{MERGE-SORT'}(A, p, q) \\
&\text{MERGE-SORT'}(A, q +1, r) \\
&\text{sync} \\
&\text{MERGE}(A, p, q, r)
\end{align*}

**Analysis**

**Work:**
\begin{align*}
T_1(n) &= 2T_1 \left( \frac{n}{2} \right) + \Theta(n) \\
&= \Theta(n \lg n)
\end{align*}

**Span:**
\begin{align*}
T_\infty(n) &= T_\infty \left( \frac{n}{2} \right) + \Theta(n) \\
&= \Theta(n)
\end{align*}

**Parallelization:**

Same as original merge-sort, except we execute the 2 recursive calls in parallel.
Multithreaded Merge Sort

\textbf{MERGE-SORT'}(A, p, r)
\begin{align*}
    \text{if } p < r \\
    q &= \lfloor (p + r)/2 \rfloor \\
    \text{spawn } \text{MERGE-SORT'}(A, p, q) \\
    \text{MERGE-SORT'}(A, q + 1, r) \\
    \text{sync} \\
    \text{MERGE}(A, p, q, r)
\end{align*}

Same as original merge-sort, except we execute the 2 recursive calls in parallel.

\textbf{Analysis}

\textbf{Work:}
\begin{align*}
    T_1(n) &= 2T_1\left(\frac{n}{2}\right) + \Theta(n) \\
    &= \Theta(n \lg n)
\end{align*}

\textbf{Span:}
\begin{align*}
    T_\infty(n) &= T_\infty\left(\frac{n}{2}\right) + \Theta(n) \\
    &= \Theta(n)
\end{align*}

\textbf{Parallelization:}
\begin{align*}
    \frac{\Theta(n \lg n)}{\Theta(n)} &= \Theta(\lg n)
\end{align*}
Problem with Merge

• Serial MERGE is dominating the performance
• How can we parallelize MERGE?
Problem with Merge

• Serial MERGE is dominating the performance
• How can we parallelize MERGE?
• Divide-and-conquer:
  – Put the middle element, $z$, of the larger of the two lists in the correct position
  – Merge the subarrays containing elements smaller than $z$
  – Merge the subarrays containing elements greater than $z$
Parallel Merge Idea

Sorted subarray 1

Sorted subarray 2

Median of first subarray

Recursively merge into 2 sub-arrays)
Parallel Merge

\[ \text{P-MERGE}(T, p_1, r_1, p_2, r_2, A, p_3) \]

\[ n_1 = r_1 - p_1 + 1 \]
\[ n_2 = r_2 - p_2 + 1 \]

if \( n_1 > n_2 \)

\hspace{1em} \text{swap P's, r's and n's} \]

if \( n_1 == 0 \)

\hspace{1em} \text{return} \]

else

\hspace{1em} q_1 = [(p_1 + r_1)/2] \]
\hspace{1em} q_2 = \text{BINARY-SEARCH}(T[q_1], T, p_2, r_2) \]
\hspace{1em} q_3 = p_3 + (q_1 - p_1) + (q_2 - p_2) \hspace{1em} // \text{Where to put } T[q_1] \]
\hspace{1em} A[q_3] = T[q_1] \]

\hspace{1em} \text{spawn } \text{P-MERGE}(T, p_1, q_1 - 1, p_2, q_2 - 1, A, p_3) \]
\hspace{1em} \text{P-MERGE}(T, q_1 + 1, r_1, q_2, r_2, A, q_3 + 1) \]
\hspace{1em} \text{sync} \]
Parallel Merge Analysis

• Span:
  – Identify the maximum number of elements in the largest call to P-MERGE
  – The worst case merges \( \frac{n_1}{2} \) elements (from the larger subarray) with all \( n_2 \) elements (from the smaller subarray):

\[
\left\lfloor \frac{n_1}{2} \right\rfloor + n_2 \leq \frac{n_1}{2} + \frac{n_2}{2} + \frac{n_2}{2} = \frac{n_1 + n_2}{2} + \frac{n_2}{2} \\
\leq \frac{n}{2} + \frac{n}{4} \\
= \frac{3n}{4}
\]

\[
T_\infty(n) = T_\infty\left(\frac{3n}{4}\right) + \Theta(\lg n) = \Theta(\lg^2 n)
\]
Parallel Merge Analysis

• Work:

\[ T_1(n) = T_1(\alpha n) + T_1((1 - \alpha)n) + O(\lg n) \]
where \(\frac{1}{4} \leq \alpha \leq \frac{3}{4}\)

Can show that \(T_1(n) \leq c_1 n - c_2 \lg n\) for constants \(c_1, c_2\), and thus prove that \(T_1(n) = \Theta(n)\)
Parallel Merge Sort

\[
P\text{-MERGE}\text{SORT}(A, p, r, B, s)\\n= r - p + 1\\n\textbf{if } n == 1\\n\hspace{1em} B[s] = A[p]\\n\textbf{else}\\n\hspace{1em} \text{let } T[n] \text{ be a new array}\\n\hspace{1em} q = [(p + r)/2]\\n\hspace{1em} q' = q - p + 1\\n\hspace{1em} \textbf{spawn } P\text{-MERGE}\text{-SORT}(A, p, q, T, 1)\\n\hspace{1em} P\text{-MERGE}\text{-SORT}(A, q + 1, r, T, q' +1)\\n\hspace{1em} \textbf{sync}\\n\hspace{1em} P\text{-MERGE}(T, 1, q', q' +1, n, B, s)\]

Analysis

Work:

Span:

Parallelization:
Parallel Merge Sort

\begin{align*}
P-\text{MERGE}\text{-SORT}(A, p, r, B, s) \\
n = r - p + 1 \\
\text{if } n == 1 \\
\quad & B[s] = A[p] \\
\text{else} \\
\quad & \text{let } T[n] \text{ be a new array} \\
\quad & q = \lfloor (p + r)/2 \rfloor \\
\quad & q' = q - p + 1 \\
\quad & \text{spawn } P-\text{MERGE}\text{-SORT}(A, p, q, T, 1) \\
\quad & P-\text{MERGE}\text{-SORT}(A, q + 1, r, T, q' +1) \\
\quad & \text{sync} \\
\quad & P-\text{MERGE}(T, 1, q', q' +1, n, B, s)
\end{align*}

Analysis

Work:
\[
T_1(n) = 2T_1 \left( \frac{n}{2} \right) + \Theta(n) = \Theta(n \lg n)
\]

Span:

Parallelization:
Parallel Merge Sort

P-MERGESORT(A, p, r, B, s)
n = r – p + 1
if n == 1
    B[s] = A[p]
else
    let T[n] be a new array
    q = ⌊(p + r)/2⌋
    q′ = q – p + 1
    spawn P-MERGE-SORT(A, p, q, T, 1)
P-MERGE-SORT(A, q + 1, r, T, q′ +1)
sync
    P-MERGE(T, 1, q′,q′ +1, n, B, s)

Analysis

Work:

\[ T_1(n) = 2T_1 \left( \frac{n}{2} \right) + \Theta(n) \]
\[ = \Theta(n \lg n) \]

Span:

\[ T_\infty (n) = T_\infty \left( \frac{n}{2} \right) + \Theta(\lg^2 n) \]
\[ = \Theta(\lg^3 n) \]

Parallelization:
Parallel Merge Sort

\[
P\text{-MERGESORT}(A, p, r, B, s)\\n= r - p + 1\\n\text{if } n == 1\\n\quad B[s] = A[p]\\n\text{else}\\n\quad \text{let } T[n] \text{ be a new array}\\n\quad q = \lfloor (p + r)/2 \rfloor\\n\quad q' = q - p + 1\\n\quad \text{spawn } P\text{-MERGE-SORT}(A, p, q, T, 1)\\n\quad P\text{-MERGE-SORT}(A, q + 1, r, T, q' + 1)\\n\quad \text{sync}\\n\quad P\text{-MERGE}(T, 1, q', q' + 1, n, B, s)\\n\]

Analysis

**Work:**

\[
T_1(n) = 2T_1\left(\frac{n}{2}\right) + \Theta(n)\\n= \Theta(n \lg n)
\]

**Span:**

\[
T_\infty(n) = T_\infty\left(\frac{n}{2}\right) + \Theta(\lg^2 n)\\n= \Theta(\lg^3 n)
\]

**Parallelization:**

\[
\frac{\Theta(n \lg n)}{\Theta(\lg^3 n)} = \Theta\left(\frac{n}{\lg^2 n}\right)
\]
Summary of Multithreading

We’ve looked at the following:

• How to create a computation dag, and analyze it in terms of work and span.
• How to write parallel code using parallel, spawn, and sync.
• How to analyze parallel code in terms of work, span, and parallelism.
• How to determine whether code has a race condition.
• Parallel algorithms for:
  – multithreaded matrix multiplication
  – multithreaded merge sort
Reading Assignments

• Reading assignment for next class:
  – Chapter 29.1

• Announcement: Exam #2 on Tuesday, April 1
  – Will cover greedy algorithms, amortized analysis
  – HW 6-9