Multithreaded algorithms

Matrix Multiplication $(n \times n)$

$C = AB \rightarrow$ divide and conquer (not Strassen)

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \overbrace{\left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right)}^{\frac{n}{2} \times \frac{n}{2}} \left(\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right) = \left(\begin{array}{c|c} A_{11}B_{11} & A_{11}B_{12} \\ \hline A_{21}B_{11} & A_{21}B_{12} \end{array}\right) + \left(\begin{array}{c|c} A_{12}B_{21} & A_{12}B_{22} \\ \hline A_{22}B_{21} & A_{22}B_{22} \end{array}\right)$$

Mult (C, A, B, n) // n is power of 2
  temp matrix T[1...n, 1...n]
  <u>if</u> n = 1
    <u>then</u> C[1,1] ← A[1,1] · B[1,1]
    <u>else</u> <partition matrices> // O(1) time
      <u>spawn</u> Mult $(C_{11}, A_{11}, B_{11}, n/2)$
      <u>spawn</u> Mult $(C_{12}, A_{11}, B_{12}, n/2)$
      <u>spawn</u> Mult $(C_{21}, A_{21}, B_{11}, n/2)$
      <u>spawn</u> Mult $(C_{22}, A_{21}, B_{12}, n/2)$

$\left(\begin{array}{c|c} A_{11}B_{11} & A_{11}B_{12} \\ \hline A_{21}B_{11} & A_{21}B_{12} \end{array}\right)$

      <u>spawn</u> Mult $(T_{11}, A_{12}, B_{21}, n/2)$
      <u>spawn</u> Mult $(T_{12}, A_{12}, B_{22}, n/2)$
      <u>spawn</u> Mult $(T_{21}, A_{22}, B_{21}, n/2)$
      <u>spawn</u> Mult $(T_{22}, A_{22}, B_{22}, n/2)$

$\left(\begin{array}{c|c} A_{12}B_{21} & A_{12}B_{22} \\ \hline A_{22}B_{21} & A_{22}B_{22} \end{array}\right)$

      <u>sync</u>
      Add (C, T, n)

Add (C, T, n) // $C \leftarrow C + T$
<base & partitioning>
spawn Add (C_{11}, T_{11}, $n/2$)
spawn Add (C_{12}, T_{12}, $n/2$)
spawn Add (C_{21}, T_{21}, $n/2$)
spawn Add (C_{22}, T_{22}, $n/2$)
sync

Analysis:

let $M_p(n)$ = P-processor execution time for Mult

$A_p(n)$ = P-processor execution time for Add

Work (want to run program on 1 processor in same time as non-parallel programs)

$$A_1(n) = 4A_1\left(\frac{n}{2}\right) + \Theta(1) = \Theta(n^2)$$ 1st case Master

good, input matrix is n^2 size, same as serial

$$M_1(n) = 8M_1\left(\frac{n}{2}\right) + \Theta(n^2)$$
$$= \Theta(n^3)$$ 1st case Master

same a serial program (not Strassen)

Critical path length

$$A_\infty(n) = A_\infty\left(\frac{n}{2}\right) + \Theta(1)$$
$$= \Theta(\lg n)$$ case 2, Master

$$M_\infty(n) = M_\infty\left(\frac{n}{2}\right) + \Theta(\lg n)$$

not max with all spawns because it depends on their execution (sync) → adding across syncs

$$= \Theta(\lg^2 n)$$ case 2, Master

when analyzing
work: + across spawns
critical path: max across spawns

6.046

Lecture 23

my comment: due to distribution of DAG dependencies and constants of threads along critical path, actual [redacted] linear speedup may be within a smaller or larger # of processors

Parallelism

$$\bar{P} = \frac{M_1(n)}{M_\infty(n)} = \Theta\left(\frac{n^3}{\lg^2 n}\right) \leftarrow$$ expect linear speedup if run with up to $\bar{P}$ processors

my comment: $P = O(\bar{P})$ linear speedup with greedy scheduler

For 1000×1000 matrices

assume constants irrelevant

$$\bar{P} \approx \frac{1000^3}{10^2} = \underline{10^7} \quad \left(10 \text{ million processors}\right) \leftarrow \text{Blue Jean} > 10K \text{ processors}$$

$\bar{P}$ is much bigger than typical P

↳ expect linear speedup in typical settings

Trade parallelism for space efficiency

(constants matter,) [redacted] a lot of parallelism

Mult-Add (C, A, B, n) // $C \leftarrow C + A \cdot B$ ← need to initialize as well in this case

< base + partition >

spawn Mult-Add (C_{11}, A_{11}, B_{11}, $n/2$)

⋮

spawn Mult-Add (C_{22}, A_{21}, B_{12}, $n/2$)

sync

spawn Mult-Add (C_{11}, A_{12}, B_{21}, $n/2$)

spawn Mult-Add (C_{22}, A_{22}, B_{22}, $n/2$)

sync

Work

$MA_1(n) = \Theta(n^3)$, same recurrence with [redacted] $\Theta(1)$ instead of $\Theta(n^2)$ for add.

Critical path length

add maxes [redacted] across syncs

$$MA_\infty(n) = 2\,MA_\infty\left(\frac{n}{2}\right) + \Theta(1)$$

$$= \Theta(n) \text{ case 1, Master}$$

Parallelism

$$\bar{P} = \frac{MA_1(n)}{MA_\infty(n)} = \theta\left(\frac{n^3}{n}\right) = \Theta(n^2)$$

For 1000×1000 matrices $\bar{P} \approx 10^6$ (enough for typical settings)

decreased $\bar{P}$ from $\approx 10^7$ to $\approx 10^6$

and

decreased space by eliminating T temp matrix

Faster in practice, since less space.

Approach: given a lot of parallelism trade it for other aspects, even constants that make a difference in practice

Sorting

Merge-Sort (A, p, r) // sort A[p..r]

  if p < r

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

      spawn Merge-Sort (A, p, q)

      spawn Merge-Sort (A, q+1, r)

      sync

      Merge (A, p, q, r) // merge A[p..q] with A[q+1..r]

  else <base case>

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

same a serial

CPL: $T_\infty(n) = T_\infty\left(\frac{n}{2}\right) + \Theta(n) = \Theta(n)$

Parallelism: $\bar{P} = \frac{T_1(n)}{T_\infty(n)} = \Theta(\lg n)$ Not much parallelism at all!

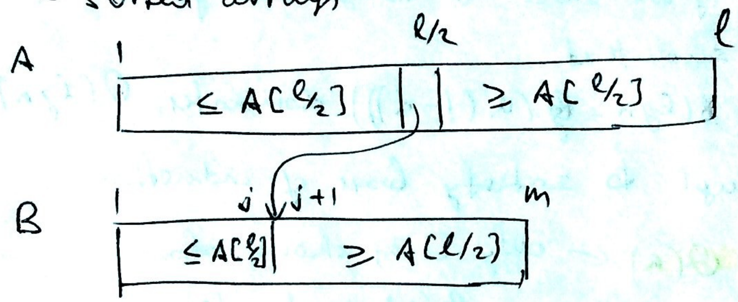
Puny!

6.046
Lecture 23

3

Must parallelize Merge,

2 sorted arrays



(a) getting $\frac{l}{2}$ from larger array guarantees that recursion is at least $\frac{n}{4}$ => guarantee of better split

P-Merge (A[1...l], B[1...m], C[1...n])

// Merge A & B into C ; $n = l + m$

<wlog $l > m$>

<base> // if small could do serial merge in practice

find j such that $B[j] \le A[l/2] \le B[j+1]$ using binary search

spawn P-Merge (A[1....l/2], B[1...j], C[1... l/2 + j])

spawn P-Merge(A[l/2+1 ... l], B[j+1 ... m], C[l/2+j+1 ... n])

sync

Critical path length : (a)

$$PM_\infty(n) \le PM_\infty\left(\frac{3n}{4}\right) + \Theta(\lg n) = \Theta(\lg^2 n)$$ case 2 Master

Work ← binary search

$$PM_1(n) = PM_1(\alpha n) + PM_1((1-\alpha)n) + \Theta(\lg n)$$

where $1/4 \le \alpha \le 3/4$

Substitution:

$PM_1(k) \le ak - b\lg k$, where $a, b > 0$ by IH since $\frac{1}{4} \le \alpha \le \frac{3}{4}$

$$PM_1(n) \le a(\alpha n) - b\lg(\alpha n) + a((1-\alpha)n) - b\lg((1-\alpha)n) + \Theta(\lg n)$$

$$= an - b(\lg(\alpha n) + \lg((1-\alpha)n)) + \Theta(\lg n)$$

$$= an - b(\lg\alpha + \lg n + \lg(1-\alpha) + \lg n) + \Theta(\lg n)$$

$= an - b(\lg\alpha + \lg n + \lg(1-\alpha) + \lg n) + \Theta(\lg n)$

$= an - b\lg n - (b(\lg n + \lg(\alpha(1-\alpha))) - \Theta(\lg n))$

$\leq an - b\lg n$ if we choose b large enough such that $b(\lg n + \lg(\alpha(1-\alpha)))$ dominates $\Theta(\lg n)$

choose a big enough to satisfy base of induction.

Thus $PM_1(n) = \Theta(n)$ ← only $O(n)$ shown above, $\Omega(n)$ is also true

Merge-Sort Analysis (parallelized Merge)

Work: $T_1(n) = \Theta(n \lg n)$ ← Merge work did not change compared to the non-parallelized Merge

CPL: $T_\infty(n) = T_\infty(\frac{n}{2}) + \Theta(\lg^2 n)$ Merge-Sort recurrence

$= \Theta(\lg^3 n)$

$\bar{P} = \Theta(\frac{n}{\lg^2 n})$ Best is $\bar{P} = \Theta(\frac{n}{\lg n})$

Problem in practice is to get constants down.