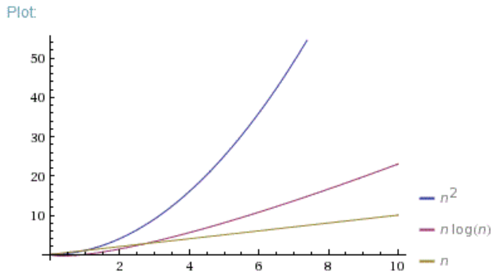
BITS Pilani
Pilani Campus

Cost Optimal Parallel Algorithms

K Hari Babu (Slides adapted from Prof. Shan’s)
Department of Computer Science & Information Systems

Performance: Cost vs. Work (Parallel Summation)

- Summation using parallel reduction:
  - Time Complexity of the algorithm: log n
  - Processors used: n/2
  - Cost: **n/2 * (log n)**
- Sequential version
  - Time complexity of sequential algorithm: n – 1
  - Processors used=1
  - Cost of sequential algorithm: **n-1**
- What is the total work done (number of additions)?
  - Parallel algorithm: n/2 + n/4 + ... + 1 = n-1
  - Sequential algorithm: n-1
- Though work done is same, but there is gap in the cost



Performance: Cost vs. Work

- Resource Efficiency
  - Ratio of costs
  - =(n-1) / (n/2 * (log n)) = 2/(log n)
- Work Efficiency
  - Ratio of work
  - = n-1/n-1 = 1
- How do you explain the difference?
  - Ineffective Parallelization:
    - Idle Processors

Cost optimal parallel algorithm

- A cost optimal parallel algorithm is an algorithm for which the cost is in the same complexity class as an optimal sequential algorithm
  - If the number of operations performed (work) by the parallel algorithm is of the same complexity class as an optimal sequential algorithm, then a cost-optimal parallel algorithm exists
- Is there a cost-optimal parallel reduction algorithm that also has time complexity $\Theta(\log n)$?
  - The cost of parallel algorithm is : n/2 * (log n)
  - The cost of sequential algorithm is: n-1
  - To make them equal, the minimum number of processors should be
    - $p = \frac{n-1}{\log(n)} \Rightarrow p = \Theta(\frac{n}{\log(n)})$

Cost Optimal Parallel Algorithm

BITS Pilani

- Brent’s Theorem:
  - Given A, *a* parallel algorithm with computation time *t*, if parallel algorithm A performs *m* computational operations, one can construct an algorithm A’ to perform the same work with *p* processors in time $t + \frac{m-t}{p}$
    - Proof: Let s_i denote the number of computational operations performed by parallel algorithm at step *i* where $1 \le i \le t$. By definition $\sum_{i=1}^{t} s_i = m$. Using p processors we can simulate step *i* in time $\left\lceil \frac{s_i}{p} \right\rceil$. The entire computation A can be performed with p processors in time
    - $\sum_{i=1}^{t} \left\lceil \frac{si}{p} \right\rceil \le \sum_{i=1}^{t} \frac{s_i+p-1}{p}$ //by adding a fraction (p-1)/p
    - $\le \sum_{i=1}^{t} \frac{p}{p} + \sum_{i=1}^{t} \frac{s_i-1}{p}$
    - $\le t + (m-t)/p$

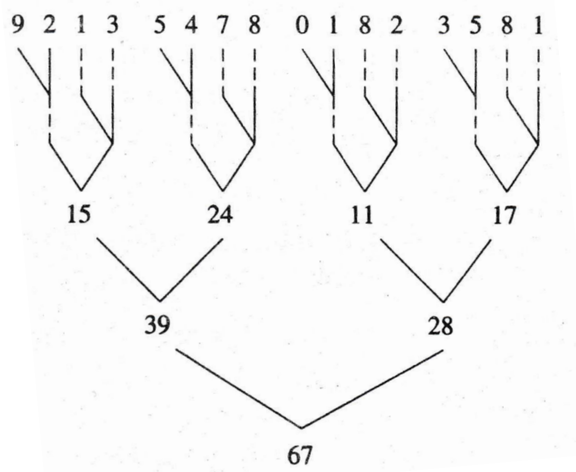
Note this reduction is work-preserving, meaning that the total work does not change. Also, note p is lesser than the initial number of processors, which is manifested by the increase in the time required

Applying Brent's Theorem

BITS Pilani

- Applying Brent's theorem to check whether by reducing processors, time complexity remains the same
  - Execution time with $\left\lfloor \frac{n}{\log(n)} \right\rfloor$ processors is
    - $\lceil \log n \rceil + \frac{n-1-\ \lceil \log n \rceil}{\left\lfloor \frac{n}{\log n} \right\rfloor} = \Theta(\log n + \log n - \frac{\log n}{n} - \frac{log^2 n}{n})$
    - $= \Theta(\log n)$
    - In this case reducing the number of processors from $n\ to\ \left\lfloor \frac{n}{\log n} \right\rfloor$ does not change the complexity of the parallel algorithm

Can add n values in $\Theta(logn)$ using $\left\lfloor \frac{n}{\log n} \right\rfloor$ processors
During first few iterations, each processor emulates a set of processors, adding to the execution time but not increasing the complexity
During later iterations, when no more than $\left\lfloor \frac{n}{\log n} \right\rfloor$ processors are needed, algorithm is identical to the original algorithm



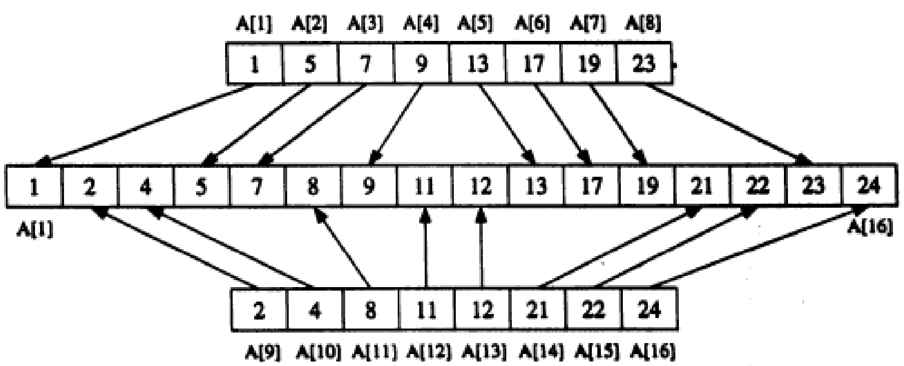
Cost vs. Work: A different example

BITS Pilani

- Problem: Merging two sorted lists A and B (each of size N) and store the sorted list in C // Assume elements are unique
- Parallel solution:
  - for all Pi, where i = 1 to n
    Pi computes the notional position of A[i] in B, say j;
    (Hint: run Binary Search sequentially)
    C[j+i] = A[i];
  - for all Pi, where i = 1 to n
    Pi computes the notional position of B[i] in A, say j;
    C[j+i] = B[i];

Merging two sorted lists

- Given on the side is CREW PRAM algorithm for merging two lists
  - Assumption: union of two lists has disjoint values
- 1 processor per element
- Each processor searches in another half to determine the final location
  - Finalindex=high+i-n/2



MERGE.LISTS (CREW PRAM):

```
Given: Two sorted lists of n/2 elements each, stored in
        A[1] ··· A[n/2] and A[(n/2)+1] ··· A[n]
        The two lists and their unions have disjoint values
Final condition: Merged list in locations A[1] ··· A[n]
Global  A[1 ··· n]
Local   x, low, high, index
begin
  spawn (P1, P2, ..., Pn)
  for all Pi where 1 ≤ i ≤ n do
    { Each processor sets bounds for binary search }
    if i ≤ n/2 then
      low ← (n/2) + 1
      high ← n
    else
      low ← 1
      high ← n/2
    endif
    { Each processor performs binary search }
    x ← A[i]
    repeat
      index ← ⌊(low + high)/2⌋
      if x < A[index] then
        high ← index − 1
      else
        low ← index + 1
      endif
    until low > high
    { Put value in correct position on merged list }
    A[high + i − n/2] ← x
  endfor
end
```

Cost vs Work

- Time Complexity of the Parallel Merging Algorithm;
  - Binary Search: $\Theta(\log n)$
  - Placing element = O(1)
  - Total time taken =$\Theta(\log n)$
  - No of processors=n
  - Cost = $\Theta(n\ log\ n)$
- Sequential version
  - Time complexity of sequential algorithm: $\Theta(n)$
  - Processors used=1
  - Cost of sequential algorithm:$\Theta(n)$
- Work:
  - $\Theta(n\ log\ n)$ comparisons required
  - N-1 comparisons required in sequential

Cost vs Work

- Efficiency factors:
  - Work Efficiency:
    - Sequential Work / Parallel Work = 1/log(N)
  - Resource Efficiency:
    - Sequential cost / Parallel Cost = 1/log(N)

Is MergeLists Parallel Algorithm Cost Optimal?

BITS Pilani

- A parallel system is cost-optimal if the cost of parallel system is proportional to the execution time of the best serial algorithm on a single processor.
  - Merge lists algorithm is not cost-optimal because
  - $\Theta(n \, log \, n) > \Theta(n)$
- Will there exist a cost-optimal algorithm?
  - Work done by parallel and sequential algorithm is not same
  - $\Theta(n \, log \, n) > n - 1$
  - Therefore cost-optimal algorithm is not possible by reducing number of processors
  - May be possible if number of operations (work) also reduced

References

- Chapter 2 from M.J. Quinn, *Parallel Computing : Theory & Practice*, McGraw Hill Inc. 2nd Edition 2002 innovate achieve lead

BITS Pilani
Pilani Campus

Thank You