The PRAM Model & Optimality

Alexandre David
1.2.05
adavid@cs.aau.dk

Outline

- Introduction to Parallel Algorithms (Sven Skyum)
  - PRAM model
  - Optimality
  - Examples

PRAM Model

- A PRAM consists of
  - a *global* access *memory* (i.e. shared)
  - a set of *processors* running the same program (though not always), with a private *stack*.
- A PRAM is **synchronous.**
  - One global clock.
- Unlimited resources.

Classes of PRAM

- How to resolve *contention*?
  - EREW PRAM – exclusive read, exclusive write
  - CREW PRAM – concurrent read, exclusive write
  - ERCW PRAM – exclusive read, concurrent write
  - CRCW PRAM – concurrent read, concurrent write

Most realistic?
Most convenient?

Example: Sequential Max

```
Function smax(A,n)
    m := -∞
    for i := 1 to n do
        m := max{m,A[i]}
    od
    smax := m
end
```

Time $O(n)$

Sequential dependency, difficult to parallelize.

Example: Sequential Max (bis)

```
Function smax2(A,n)
    for i := 1 to n/2 do
        B[i] := max{A[2i-1],A[2i]}
    od
    if n = 2 then
        smax2 := B[1]
    else
        smax2 := smax2(B,n/2)
    fi
end
```

Time $O(n)$

Dependency only between every call.

Example: Parallel Max

Time $O(\log n)$

```
Function smax2(A,n) [p1,p2,…,pn/2]
    for i := 1 to n/2 pardo
        pi: B[i] := max{A[2i-1],A[2i]}
    od
    if n = 2 then
        p1: smax2 := B[1]
    else
        smax2 := smax2(B,n/2) [p1,p2,…,pn/4]
    fi
end
```

Analysis of the Parallel Max

- Time: $O(\log n)$ for $n/2$ processors.
- *Work done?*
  - $p(n)=n/2$ number of processors.
  - $t(n)$ time to run the algorithm.
  - $w(n)=p(n)*t(n)$ work done.
    Here $w(n)=O(n \log n)$.

    ? *Is it optimal?*

Optimality

Definition

If $w(n)$ is of the **same order** as the time for the best known sequential algorithm, then the parallel algorithm is said to be **optimal**.

Analysis of the Parallel Max

- Time: $O(\log n)$ for $n/2$ processors.
- *Work done?*
  - $p(n)=n/2$ number of processors.
  - $t(n)$ time to run the algorithm.
  - $w(n)=p(n)*t(n)$ work done.
    Here $w(n)=O(n \log n)$.
    Is it optimal? ***NO, O(n) to be optimal.***
    Why?

?

But…

Can a parallel algorithm solve a problem with **less** work than the best known sequential solution?

Design Principle

Construct optimal algorithms to run **as fast as possible**.

=

Construct optimal algorithms using **as many processors as possible**!

Because optimal with p $\rightarrow$ optimal with fewer than p.
Opposite false.
Simulation does not add work.

Brent's Scheduling Principle

Theorem

If a parallel computation consists of
- ***k*** **phases**
- taking time $t_1, t_2, \ldots, t_k$
- using $a_1, a_2, \ldots, a_k$ processors
- in phases $1, 2, \ldots, k$

then the computation can be done in time ***O(a/p+t)*** using ***p*** **processors** where $t = \mathrm{sum}(t_i)$, $a = \mathrm{sum}(a_i t_i)$.

Brent's Scheduling Principle

- i'th phase:
  - p processors simulate a_i processors.
  - Each of them simulate at most ceil(a_i/p)≤$a_i/p+1$, which consumes time t_i at a constant factor for each of them.
  - Total ≤ sum(t_i*(a_i/p+1)) = a/p+t

Previous Example

- k phases = log$n$.
- t_i = constant time.
- $a_i = n/2, n/4, \dots, 1$ processors.
- With p processors we can use time $O(n/p + \log n)$.
- Choose $p = O(n/\log n) \rightarrow$ time $O(\log n)$ and this is **optimal**!

There is a "but": You need to know n in advance to schedule the computation.

Prefix Computations

Input: array A[1..n] of numbers.
Output: array B[1..n] such that B[k] = sum(i:1..k) A[i]
Sequential algorithm:

```
function prefix+(A,n)
    B[1] := A[1]
    for i = 2 to n do
        B[i] := B[i-1]+A[i]
    od
end
```

Time *O(n)*

Prefix Computation

```
function prefix+(A,n)
        B[1] := A[1]
    if n > 1 then
        for i = 1 to n/2 pardo
                C[i]:=A[2i-1]+A[2i]
        od
        D:=prefix+(C,n/2)
        for i = 1 to n/2 pardo
                B[2i]:=D[i]
        od
        for i = 2 to n/2 pardo
                B[2i-1]:=D[i-1]+A[2i-1]
        od
    fi
    prefix+:=B
end
```

Parallel Prefix Computation

```
function prefix⁺(A,n)[p1,…,pn]
        p1: B[1] := A[1]
        if n > 1 then
                for i = 1 to n/2 pardo
                        pi: C[i]:=A[2i-1]+A[2i]
                od
                D:=prefix⁺(C,n/2)[p1,…,pn/2]
                for i = 1 to n/2 pardo
                        pi: B[2i]:=D[i]
                od
                for i = 2 to n/2 pardo
                        pi: B[2i-1]:=D[i-1]+A[2i-1]
                od
        fi
        prefix⁺:=B
end
```

Prefix Computations

- The point of this algorithm:
  - It works because + is associative (i.e. the compression works).
  - It will work for *any* other associative operations.
  - Brent's scheduling principle:

For any associative operator computable in *O(1)*, its prefix is computable in *O(* log*n)* using *O(n/* log*n)* processors, which is optimal!

Merging (of Sorted Arrays)

- Rank function:
  - rank(x,A,n) = 0 if x < A[1]
  - rank(x,A,n) = max{i | A[i] ≤ x}
  - Computable in time $O(\log n)$ by binary search.
- Merge A[1..n] and B[1..m] into C[1..n+m].
- Sequential algorithm in time $O(n+m)$.

Parallel Merge

```
function merge1(A,B,n,m)[p1,…,pn+m]
    for i = 1 to n pardo pi:
        IA[i] := rank(A[i]-1,B,m)
        C[i+IA[i]] := A[i]
    od
    for i = 1 to m pardo pi:
        IB[i] := rank(B[i],A,n)
        C[i+IB[i]] := B[i]
    od
    merge1 := C
end
```

?

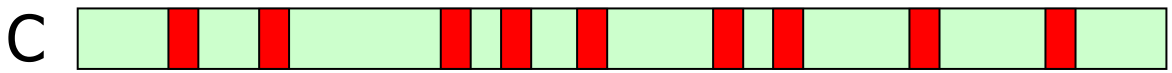
CREW
Not optimal.

Optimal Merge - Idea

n/log(n) sub-arrays of log(n) elements

m/log(m) sub-arrays of log(m) elements

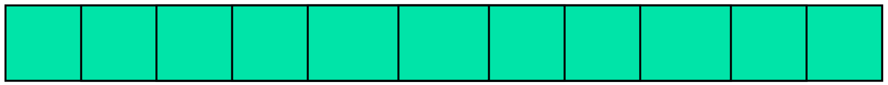
previous merge: n/log(n) + m/log(m) elements
position of the ends in C
costs O(log(n+m)),
(optimal) on (m+n)/log(n+m) processors!



Merge n/log(n)+m/log(m) lists with sequential merge in parallel.
Max length of sub-list is O(log(n+m)).

Example: Max in O(1)

- Max of an array in constant time!

A  *n* elements

1. Use *n* processors to initialize B.
2. Use n^2 processors to compare all A[i] & A[j].
3. Use *n* processors to find the max.

$$B[i]_{1 \le i \le n} = 0$$

$$A[i] > A[j] \Rightarrow B[j] = 1$$

$$B[i] = 0 \Rightarrow A[i]$$

Lessons

- PRAM not realistic
  - no communication
- Reasoning on algorithms still interesting
  - notion of optimality applies
  - scheduling principle applies