Programming Parallel Algorithms

Guy E. Blelloch
Carnegie Mellon University

Why are PRAM algorithms a failure outside of the theory community?

1.
2.
3.
4.

Why are serial RAM algorithm successful outside the theory community?

1. **RAM model matches commercial machines?**
   Not really true, but a good-enough approximation
2. **RAM model matches standard languages.**
   Algol, Pascal, C
   Algorithms are easy to program

How do you program a PRAM?

What languages are there?

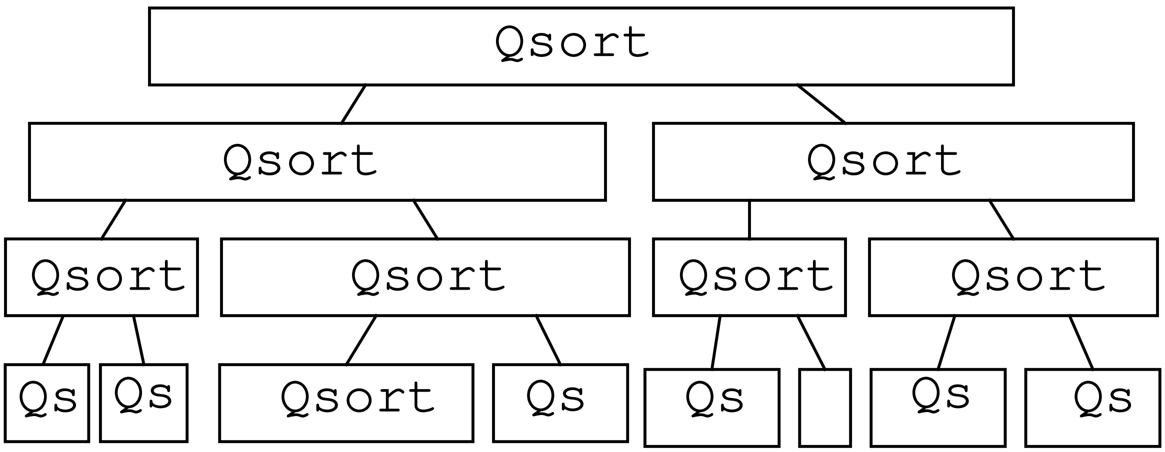
Programming a V   anilla PRAM

Inconveniences:

- Looping.
- Keeping track of inactive processors.
- Keeping processors synchronous.
- Allocating tasks to processors

Example: Quicksort

A simpleparallel algorithm



PRAM Complexity:

$$T(n) = O\left(\frac{(n \log n)}{P} + (\log n)^2\right)$$

With high probability assuming randomly selected pivots.

> 10 pages of code on a vanilla PRAM

Outline

Claim: The vanilla PRAM is not a good programming model. We need to supply higher-level model.

Data-Parallel Languages

Fortran 90 (HPF)

- Parallel Prefix
- List-Ranking
- Limitations

NESL

- Complexity
- Quicksort
- Connected Components

Data Parallel Languages

Flat Data-Parallel Languages

- Fortran 90 and High Performance Fortran
- C*
- *Lisp

Use the Work-Time paradigm and Brent's lemma:

$$T = O\left(\frac{W}{P} + S\right)$$

Advantages:

- Virtualization
  No need to assume a fixed number of processors
- Synchronous.
  Well suited for PRAM algorithms
  Easy to debug code

Fortran 90 / HPF

```
A = \( 2 3 4 1 5 0 \)

I = \( 1 5 2 4 \)
```

Elementwise operations:

```
A + 3 -> ( 5 6 7 4 8 3 )

A[I]  -> ( 2 5 3 1 )
```

Intrinsics:

```
SUM(A)           -> 15

PREFIX(A)        -> ( 2 5 9 10 15 15 )

PACK(A, A < 3) -> ( 3 4 5 )
```

The elements of A starting at 1, up to N with Stride 2.

```
A[1:N:2]
```

For example:

```
A[1:N:2] + A[2:N:2]  ->  ( 5 5 5 )
```

Pointer Doubling

(Wyllie' s Algorithm)

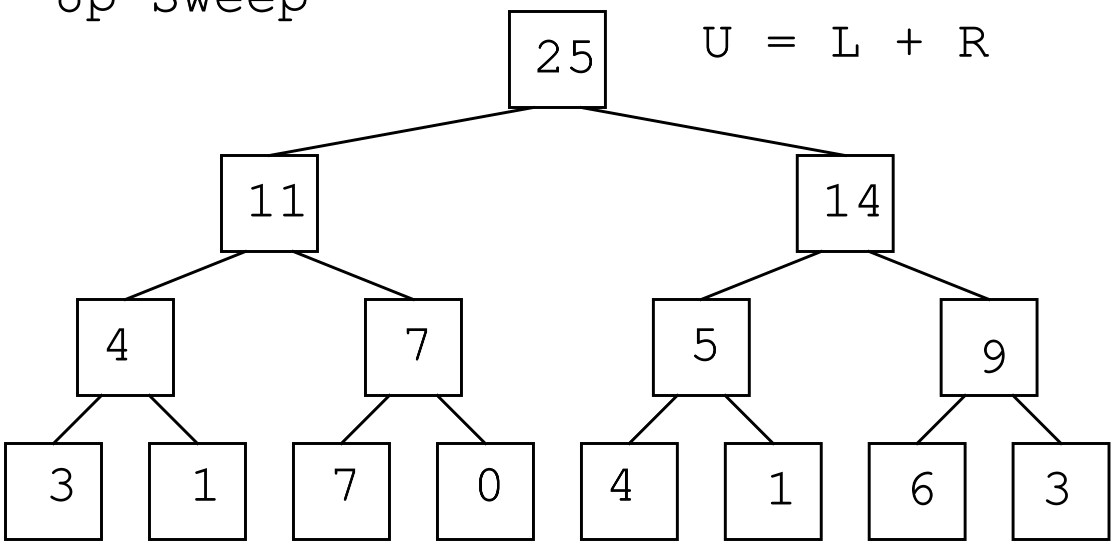
`A` = Array of values

`P` = Array of pointers

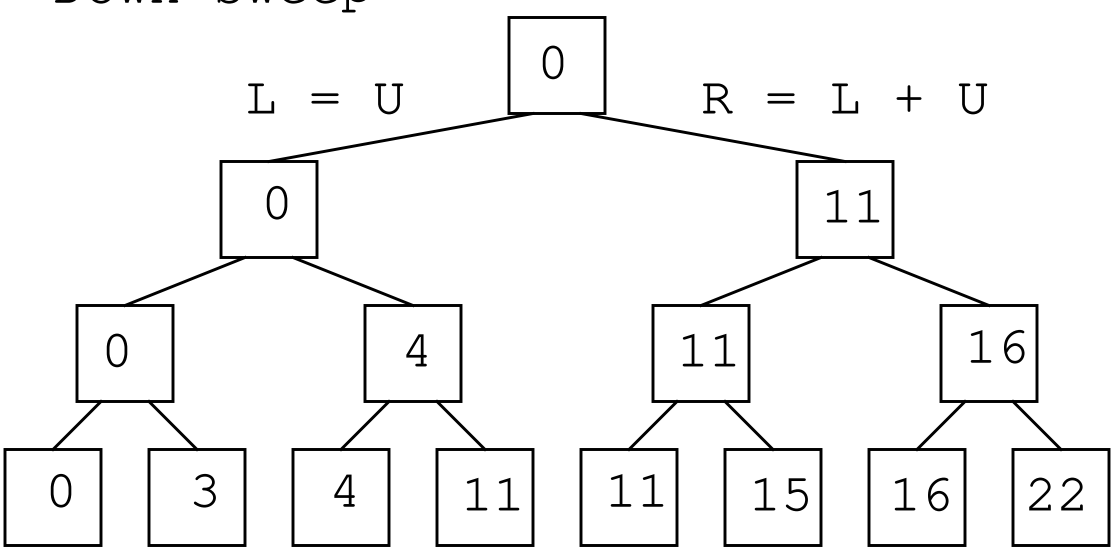
```
 Do 10 i = 1, log(n)
     A = A + A[P]
     P = P[P]
10 Continue
```

Plus-Scan (Parallel Prefix)

Up Sweep



Down Sweep



Up Sweep

```
S = 1
Do While (S .LE. N/2)
  A[2*S:L:2*S]=A[S:L:2*S]+A[2*S:L:2*S]
  S = 2*S
End Do
```

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	
[	3	1	7	0	4	1	6	3	]
[	3	4	7	7	4	5	6	9	]
[	3	4	7	11	4	5	6	14	]
[	3	4	7	11	4	5	6	25	]

$$Work(n) = O(n)$$

$$Time(n) = O(\log n)$$

Down Sweep

```
A[2*S] = 0
Do While (S .GE. 0)
  TEMP[2:L:2*S] = A[2*S:L:2*S]
  A[2*S:L:2*S]=A[S:L:2*S]+A[2*S:L:2*S]
  A[2:L:2*S] = TEMP[2:L:2*S]
  S = S/2
End Do
```

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	
[	3	4	7	11	4	5	6	0	]
[	3	4	7	0	4	5	6	11	]
[	3	0	7	4	4	11	6	16	]
[	0	3	4	11	11	15	16	22	]

Implementation

Limitations

Quicksort

```
Subroutine Quicksort(A,N)
Integer N,NLess,Less(N),Greater(N),A(N)

IF (N < 1) Return

Pivot = A(1)
Nless = Count(A < Pivot)
Less = Pack(A, A < Pivot)
Greater = Pack(A, A >= Pivot)

Call Quicksort(Less, Nless)
A[1:Nless] = Less

Call Quicksort(Greater, N - Nless)
A[Nless+1:N] = Greater

Return
End
```

$$Work(n) = O(n \log n)$$

$$Time(n) = O(n)$$

NESL

A language for teaching parallel algorithms.

1. Framework
   Interactive environment
2. Nested Data-Parallel
   Sequences can be nested
   Parallel functions can be called in parallel
3. Simple rules for composing complexity
   Work : Serial complexity
   Step : Parallel complexity

Concise and simple code

Currently runs on:

- Connection Machine CM-2, Cray Y -MP, Workstations

Parallel Operations

Sequences:

```
{7, 9, 2, 11}              int sequence
"this is a sequence"       char sequence
```

Data is distributed across the processors

Apply-to-each:

```
    {e + 3: e in [7, -2, 5, 4]}
->  {10, 1, 8, 7}

    function factorial(i) =
      if i = 1 then 1
      else i * factorial(i-1);

    {factorial(i): i in [3, 1, 7, 5, 2]}
->  [6, 1, 5040, 120, 2, 24]
```

Functions on Sequences

```
    # “how long?”
->  9

    permute(“nesl”, [2, 1, 3, 0])
->  “lens”

    {e in [7, -2, 5, 9] | e > 0}
->  [7, 5, 9]

    plus_scan([2, 1, -3, 11, 5])
->  [0, 2, 3, 0, 11]

    “foo ” ++ “bar”
->  “foo bar”

    subseq(“long string”, 5, 11)
->  “string”
```

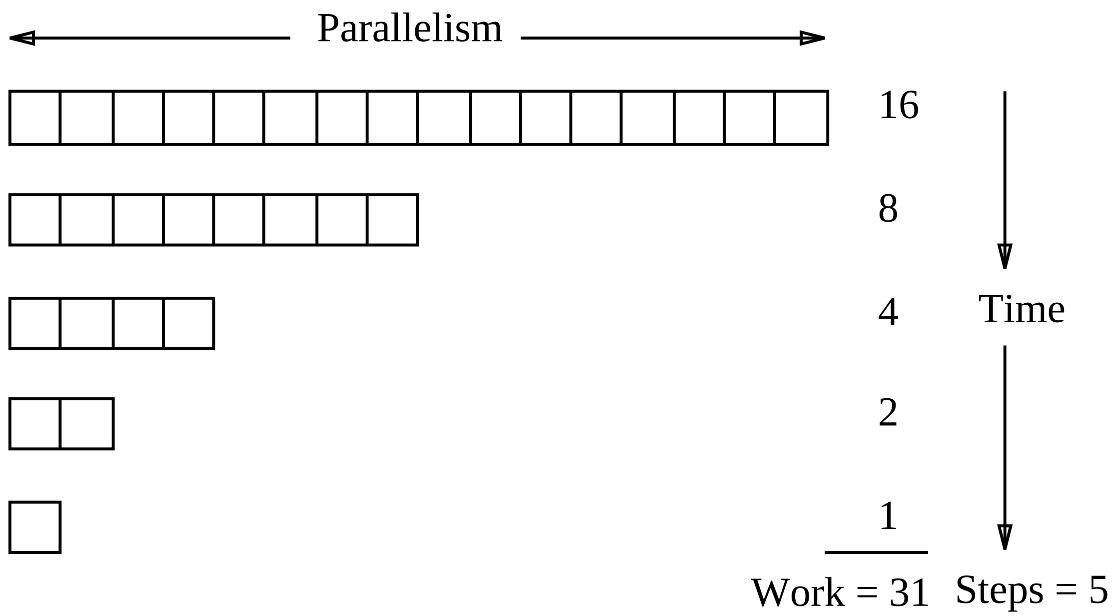
Scan

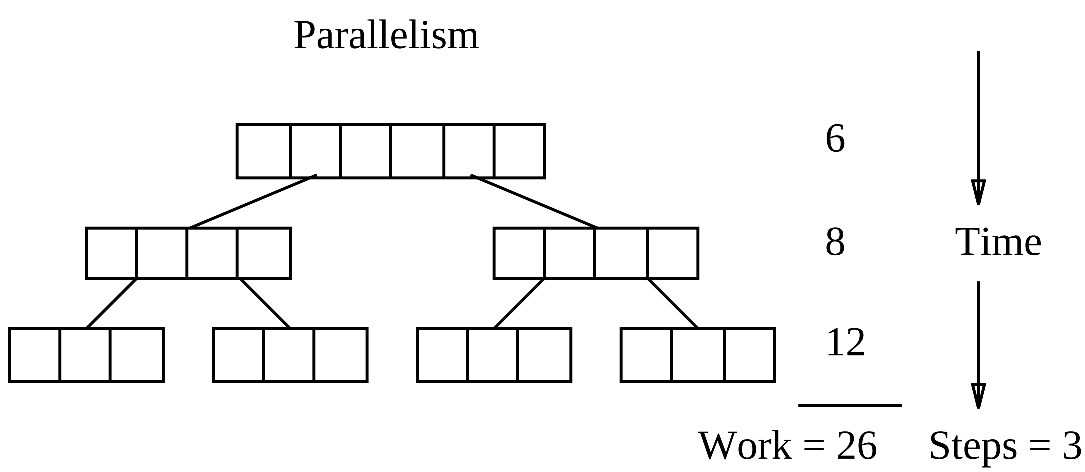
```
function scan(A) =
if #A = 1 then [0]
else
  let e = even_elts(A);
      o = odd_elts(A);
      r = scan({e + o: e; o})
  in interleave(r,{e + r: e; r});
```

$$W(n) = W\left(\frac{n}{2}\right) + k_1 n = O(n)$$

$$S(n) = S\left(\frac{n}{2}\right) + k_2 = O(\log n)$$

Complexity





Work complexity (W) = number of blocks

Step complexity (S) = depth

Complexity

Apply-to-each:

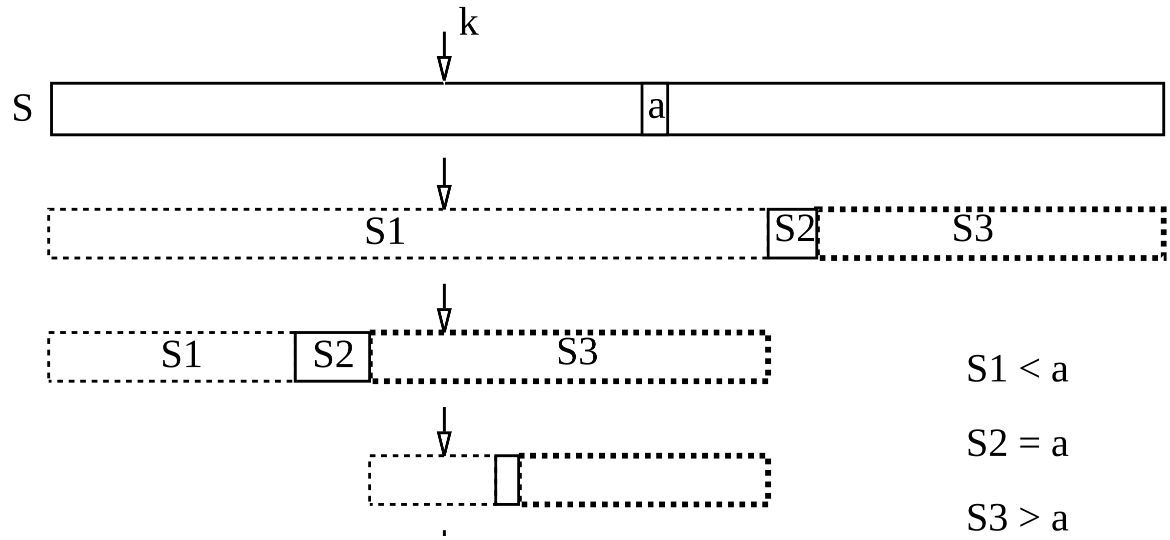
$$A = \{A_0, A_1, A_2, \ldots, A_{n-1}\}$$

```
pexp = {exp(e): e in A}
```

$$W_{pexp}(A) = \sum_{0}^{n-1} W_{exp}(A_i)$$

$$S_{pexp}(A) = max_{i=0}^{n-1} S_{exp}(A_i)$$

Median Finding



procedure SELECT(k, S):
if $|S| <= 1$ **then return** the single element S
else begin
  choose an element a randomly from S;
  let $S1, S2, S3$ be the sequences of elements
    in S less than, equal to and greater than a
  if $|S1| >= k$ **then return** SELECT $(k, S1)$
  else
    if $|S1| + |S2| >= k$ **then return** a
    else return SELECT($k - |S1| - |S2|, S3$)
end

Nested Parallelism

Nested vectors:

```
A = [[2, 1], [7, 3, 0], [4]]

B = [("fire", 2), ("consortium", 4),
     ("merchant", 1)]
```

Nested parallel calls:

```
{sum(e): e in A}     -> [3, 10, 4]

{{i - 2: i in e}: e in A}
   -> [[0, -1], [5, 1, -1], [2]]
```

Manipulating nested vectors:

```
B[1]  -> ("consortium",4)

permute(["big", "brown", "cow"],
        [1, 2, 0])

  -> ["cow", "big", "brown"]
```

Quicksort

```
function quicksort(S) =
if (#S < 2) then S
else
  let pivot = S[rand(#S)];
      S1 = {e in S| e < pivot};
      S2 = {e in S| e == pivot};
      S3 = {e in S| e > pivot};
      R  = {quicksort(v): v in [S1,S3]}
  in R[0] ++ S2 ++ R[1]
```

Nesl Code

```
function contract(e,v)
if #e == 0 then v
else
  let
    c = {rand(1): v};
    ae = {(e1,e2) in e| c[e1] < c[e2]};
    v = v <- ae;
    e = {(e1,e2) in e| v[e1] /= v[e2]}
  in contract(e,v)
```

Final Projects

From a course on programming parallel algorithms

- Maximum Flow
  (Processor efficient version)
- Cole Mergesort
- Plane Sweep T   ree
- Comparison of string matching methods
- Biconnected Components
- Deterministic list ranking
- 3D Volume Rendering
- LCA and other tree operations
- Comparison of connected components algorithms
- Parallel Expression Evaluation

Status

- Available via anonymous FTP
  (Send email to blelloch@cs.cmu.edu)
- About 50 users
- 1week course at Dartmouth
  June 23 -- June 29

Conclusions