A Practical Access to the Theory of Parallel Algorithms

Christoph Kessler
Department of Computer Science
Linköping University
S-58183 Linköping, Sweden
chrke @ ida.liu.se

ACM SIGCSE-2004 Norfolk VA, USA, March 6, 2004

Outline

- PRAM model of parallel computation
- PRAM prototype realization in hardware and software
- The PRAM programming language Fork
- Example algorithm in Fork: Parallel Quicksort
- trv tool for visualization of time behavior of PRAM algorithms
- Use as lab environment for a course on parallel algorithms
- Evaluation
- Conclusions

PRAM model of parallel computation

Parallel Random Access Machine [Fortune/Wyllie'78]

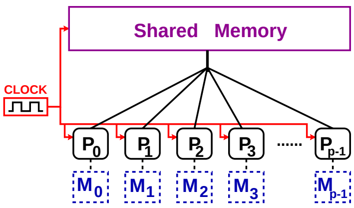
p processors

- MIMD
- common clock signal
- arithm./jump: 1 clock cycle

shared memory

- uniform memory access time
- latency: 1 clock cycle (!)
- concurrent memory accesses
- sequential consistency

private memory (optional)

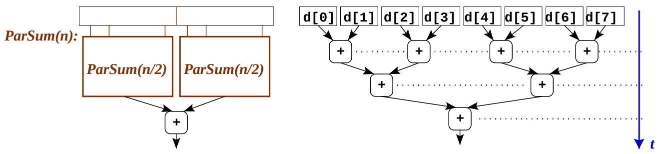


Example: Global sum computation on EREW PRAM

Given n numbers $x_0, x_1, \ldots, x_{n-1}$ stored in a shared array.

The global sum $\sum_{i=0}^{n-1} x_i$ can be computed in $\lceil \log_2 n \rceil$ time steps on an EREW PRAM with n processors.

Parallel algorithmic paradigm used: Parallel Divide-and-Conquer



- Divide phase: trivial, time $O(1)$
- Recursive calls: parallel time $T(n/2)$ with base case: memory access, time $O(1)$ $\rightarrow$ $T(n) = T(n/2) + O(1)$
- Combine phase: addition, time $O(1)$

Use induction or the master theorem [Cormen$^+$'90 Ch.4] $\rightarrow T(n) \in O(\log n)$

More on the PRAM model: see PRAM literature

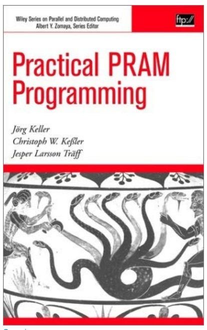
JáJá: ***An introduction to parallel algorithms.*** Addison-Wesley, 1992.

Cormen, Leiserson, Rivest: ***Introduction to Algorithms***, Chapter 30. MIT press, 1989.

Jordan, Alaghband: ***Fundamentals of Parallel Processing.*** Prentice Hall, 2003.

Keller, Kessler, Träff: ***Practical PRAM Programming.*** → Wiley Interscience, New York, 2000.

...



Cover image, copyright (c) The J. Paul Getty Museum, Malibu, CA

PRAM prototype realization in hardware and software

Saarbrücken PRAM

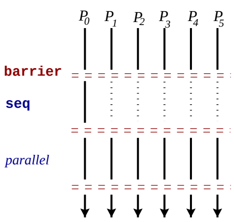
- by Wolfgang Paul and his SB-PRAM project group at Saarbrücken
- cost-efficient PRAM emulation based on "Fluent Machine" [Ranade'88]
- shared memory, hashed address space
- deterministic concurrent write (Priority CRCW PRAM)
- network: pipelined butterfly network with combining switches
- processors: RISC-like, 32-threaded, cycle-by-cycle interleaving
- constant-time multiprefix-sum / max / and / or on-the-fly
- design fixed 1991, 1 GFlops sustained performance for $p = 4096$
- research prototype $p = 2048$ built 1992–2001, $\approx$ 3 M-Euro
- software simulator
- program development tools: assembler, linker, loader, OS

The PRAM programming language Fork

MIMD parallelism
each processor has its own control (PC)

SPMD style of parallel program execution

- start fixed set of p processors
- execute `main` as one large group
- no spawning of more processors



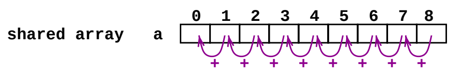
For all program variables and objects declare sharity (shared, private):

```
sh int k, *iptr, a[10];
pr float mysum;
iptr = shalloc(sizeof(int));
...
```

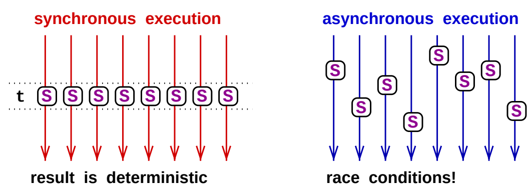

The PRAM programming language Fork (2)

Synchronous execution at the expression level

→ deterministic computation (similar to dataparallel languages à la HPF)




```
S:   a[$] = a[$] + a[$+1];
        //  $ in {0..p-1} is processor rank
```



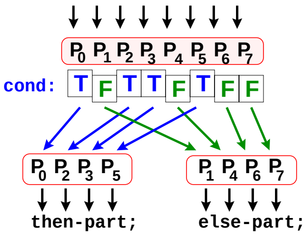
The PRAM programming language Fork (3)

Processor group concept

Programmer can relax the scope of sharing and synchronous execution:

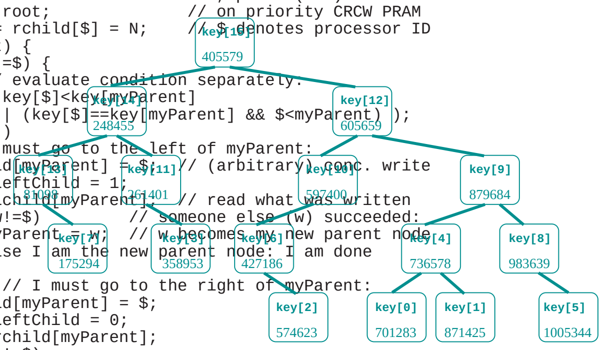
```
if ( cond )
      then-part;
else
      else-part;
```

- implicit group splitting
  `if-then-else`, `while`, ...
  → adapt to control flow
- explicit group splitting
  `fork( k, ...)`
  → parallel divide-and-conquer



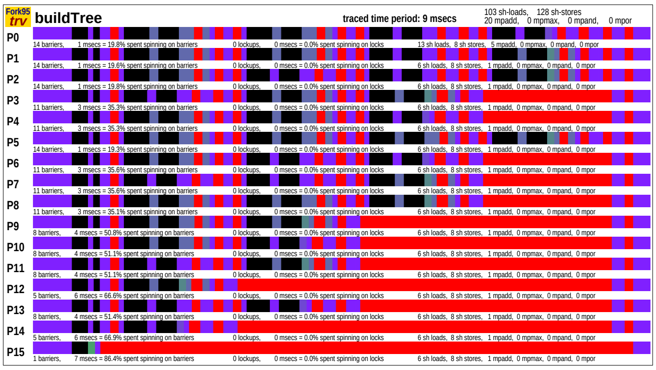
Example algorithm in Fork: Parallel Quicksort (see the paper)

```
sync void buildTree( void )  // executed by N processors
{
 pr int c, w = -1;
 root = $;     // concurrent write, proc. (N-1) succeeds
 myParent = root;              // on priority CRCW PRAM
 lchild[$] = rchild[$] = N;    // $ denotes processor ID
 if ($!=root) {
   while (w!=$) {
     farm // evaluate condition separately:
       c = (key[$]<key[myParent]
           || (key[$]==key[myParent] && $<myParent) );
     if ( c )
     { // I must go to the left of myParent:
       lchild[myParent] = $; // (arbitrary) conc. write
       IAmALeftChild = 1;
       w = lchild[myParent]; // read what was written
       if (w!=$)          // someone else (w) succeeded:
          myParent = w;   // w becomes my new parent node
       // else I am the new parent node: I am done
     }
     else { // I must go to the right of myParent:
       rchild[myParent] = $;
       IAmALeftChild = 0;
       w = rchild[myParent];
       if (w!=$)
          myParent = w;
}} } }
```



trv tool for visualization of time behavior of PRAM algorithms

Processor-time diagram



trv tool for visualization of time behavior of PRAM algorithms (2)

Tailored for tracing execution of Fork programs:

- Fork compiler instruments the program to log events during execution
  + subgroup creation / termination
  + entry, exit to barriers, locks
  + user-defined events (e.g., access to a shared data structure)
  accumulated in memory (→ low overhead), dumped to file later
- `trv` tool creates a processor-time diagram from the trace file:
  scalable format (FIG), customizable display (colors, ...), zooming possible
- phases of "useful work":
  all processors of the same group have the same color
- see idle times at barriers and locks, group splitting overheads ...
- other tools: ParaGraph, upshot, VAMPIR...
  (coarse-grained, message passing)

Use as lab environment for a course on parallel algorithms

FDA125 "Advanced Parallel Programming"

graduate-level course at Linköpings Universitet, Sweden, spring 2003 (8 st.)

- PRAM theory (time, work, cost analysis; simulation results; ...)
- Basic PRAM algorithms (list ranking, prefix sums, ...)
- Fork tutorial
- Parallel algorithmic paradigms
  data parallelism, parallel divide&conquer, pipelining, task farming, ...
- Parallel data structures
  pipelined 2-3 trees, par. hash table, par. FIFO queue, par. priority queue
- Dynamic (loop) scheduling; irregular algorithms (Barnes-Hut,...)
- Other languages: OpenMP, MPI, HPF, Cilk
  Other topics: PRAM emulation, parallel computer architecture, DSM ...

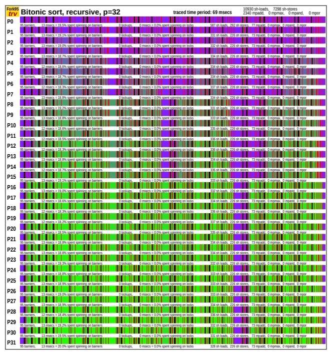
Use as lab environment for a course on parallel algorithms (2)

Programming exercise in Fork: Bitonic sort algorithm

Goals:

- understand the algorithm from textbook
  [Cormen/Leiserson/Rivest Ch. 28]
- formulate it as a Fork program
- experimentally verify $O(\log^2 N)$ complexity
- face problems in parallel programming
  (coordination, sharing, synchronicity, ...)
- apply structured parallel programming
- use trv visualization

The exercise was finished in time by 6 out of 8 students.



Evaluation

Evaluation after the parallel programming exercise

Questionnaire: see course webpage www.ida.liu.se/~chrke/courses/APP

Question	Yes	No
Was the exercise too hard / too easy / just the right degree of difficulty?	6	0
Did you look at the demo examples in Fork?	6	0
Did you find the trace file visualizer useful		
– to identify performance bottlenecks?	4	1
– to understand the structure of computation?	4	1
– to debug your program?	3	2
Did you learn something by doing the assignment		
– about the theory of parallel algorithms?	3	1
– about parallel implementation problems?	6	0
the practical exercise was a useful complement of the other parts of the course	6	0

plus free-text comments: suggestions for improvements, "Linux?", "BSP?" ...

Conclusions

PRAM model: easy to program and analyze:

- focus on pure parallelism,
- no worry about data locality or consistency;

should be taught even *before* threads and MPI.

Fork and the PRAM simulator can be used as lab equipment to complement traditional courses on parallel algorithms.

The processor-time diagram helps with understanding and verifying the structure of the parallel computation.

Download Fork and the simulator at www.ida.liu.se/~chrke/fork (system requirements: Solaris / HP-UX)

Future work: Web service for remote execution of Fork programs