Parallel partial ordering for exact and approximate median splitting

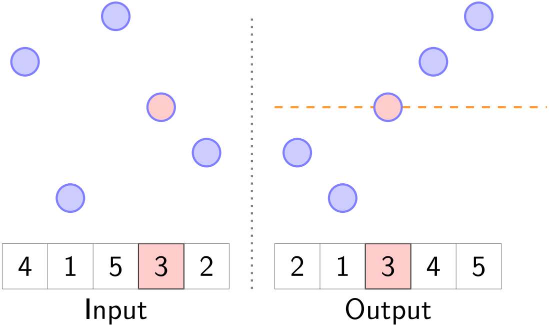
Matthieu Garrigues <matthieu.garrigues@ensta.fr>

February 1, 2011

Outline

The problem

- Given an array A of N values, find the median value.
- Rearrange A such that
  $\max_{i \in [0, N/2[} A[i] \leq A[N/2] \leq \min_{i \in]N/2, N/2[} A[i]$

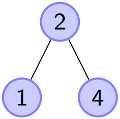
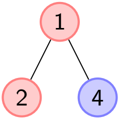
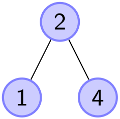
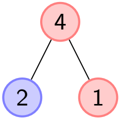
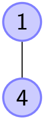
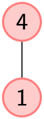


Efficient sequential algorithms exist

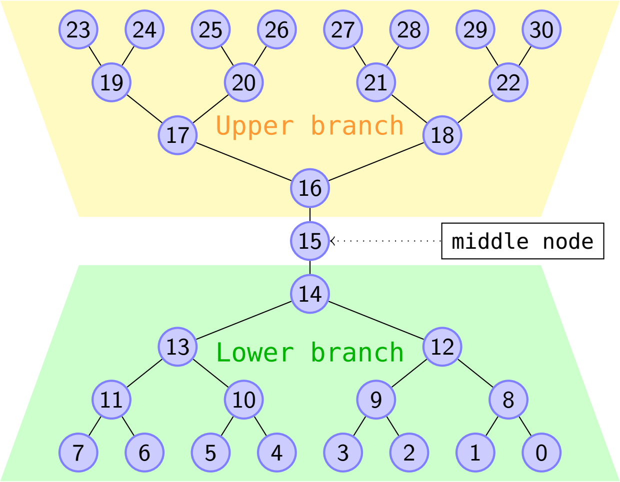
- QuickSelect [BFP+73]
- QuickSelect + Median of median [BFP+73]

What about parallel implementations?

- A lot of research has been conducted for coarse-grained parallel computers ([AfAGR97]).
- However, today, on graphic processor units (GPU), the fastest way to search the median is to sort the whole input array then pick the middle value (1 Giga-key/s on Nvidia GTX480 using [MG10]).
- Can we build a faster algorithm that only gives the median using GPU?

Primitive	Input	Output
$select_{min}$		
$select_{max}$		
reorder		

Input: $A[0]...A[N-1]$ an array of N elements drawn from some totally ordered set.



A *primitive* version of our algorithm

Sand clock algorithm

For each node n with children c_1 and c_2:

- If n is in the lower branch, call $select_{max}(n, c_1, c_2)$
- If n is in the upper branch, call $select_{min}(n, c_1, c_2)$
- If n is the middle node, $reorder(n-1,\ n,\ n+1)$

Iterate until convergence

Convergence

- Find a solution in N/2 iteration in the worst case
- Inefficient because of the bottleneck at the middle node

Reducing the bottleneck

Swap inter branches algorithm

For each node n in the lower branch, m its counterpart in the upper branch:

- $reorder(n, m)$

For each node n with children c_1 and c_2:

- If n is in the lower branch, call $select_{max}(n, c_1, c_2)$
- If n is in the upper branch, call $select_{min}(n, c_1, c_2)$
- If n is the middle node, $reorder(n-1,\ n,\ n+1)$

Convergence

- Find a solution in N/4 iteration in the worst case
- Inefficient because of the remaining bottleneck at the middle node

Reducing the bottleneck

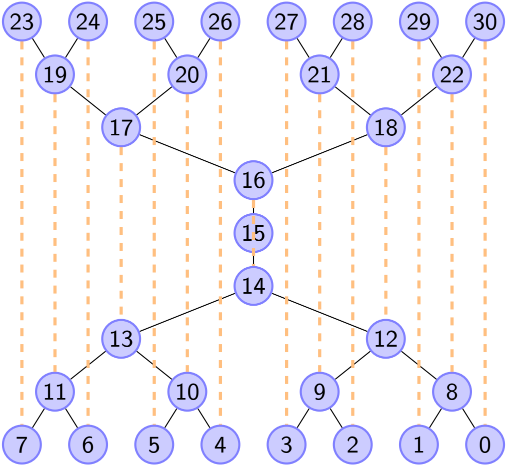


Figure: Structure of swap inter branches algorithm. The dashed lines show the additional *reorder* scheme.

Final algorithm

Remove the bottleneck

- Children of a index on level L randomly change over the iterations, but remain on level $L+1$
- Counterpart of a node on level L also randomly changes over the iterations, but remains on level L on the other branch.

Convergence

- In average, find a solution in $log_2(N)$ iterations.

Convergence

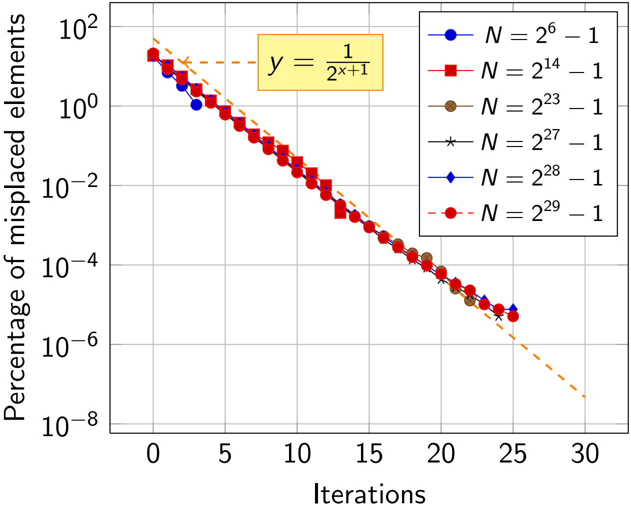
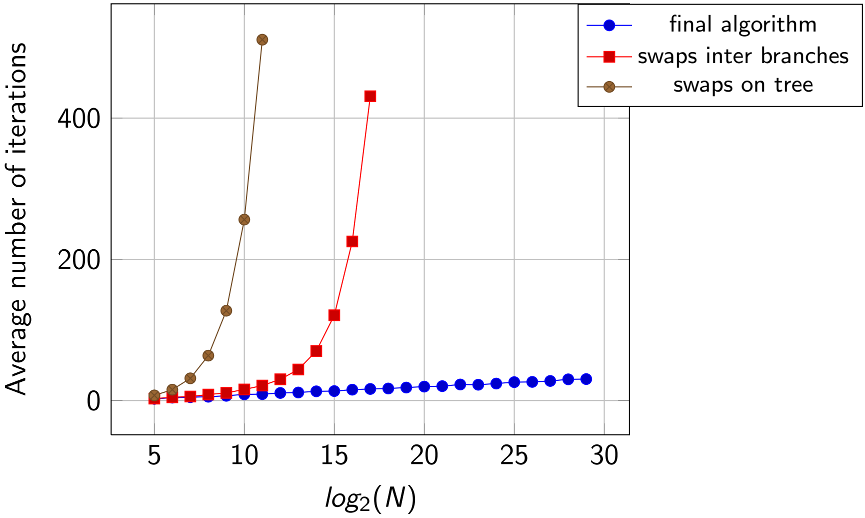


Figure: Convergence of random shifts.

Benchmark the three algorithms



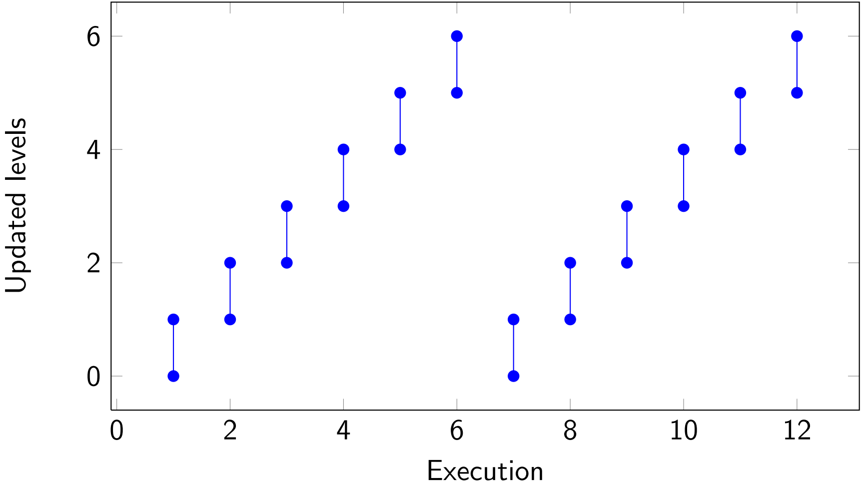
How much is this algorithm parallel?

Parallelism of one iteration

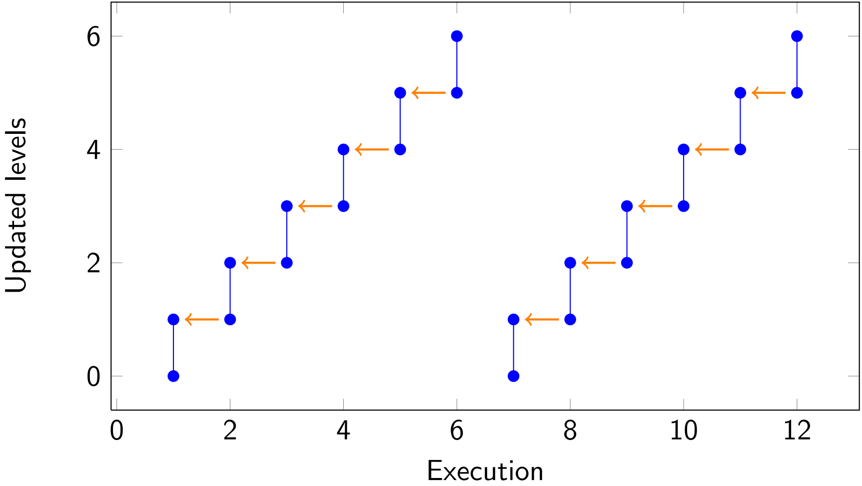
- All nodes of a level can be processed in parallel
- But, it needs to sequentially browse the levels from the leafs to the middle node.

What about running several iterations in parallel?

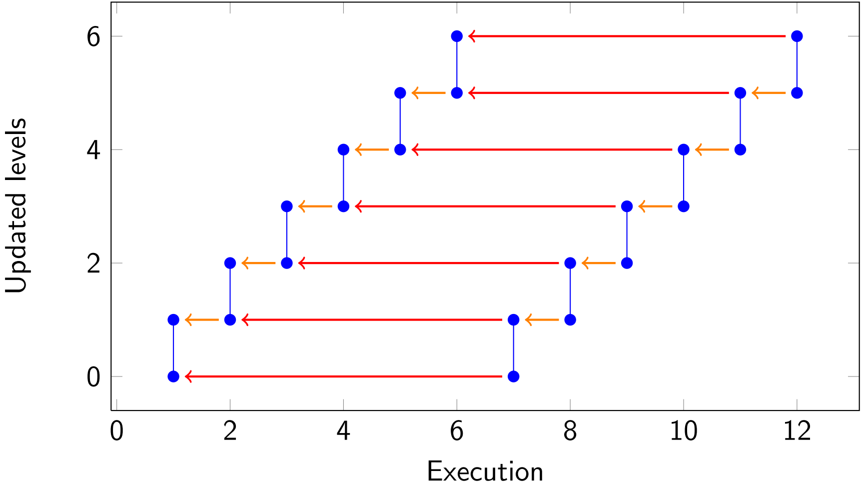
Dependencies of level processings



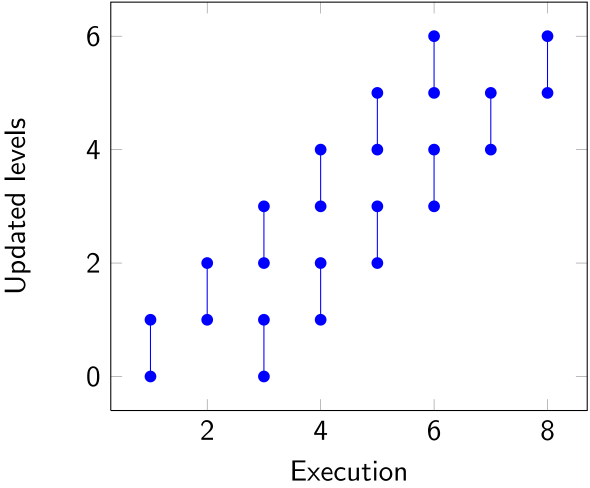
Dependencies of level processings



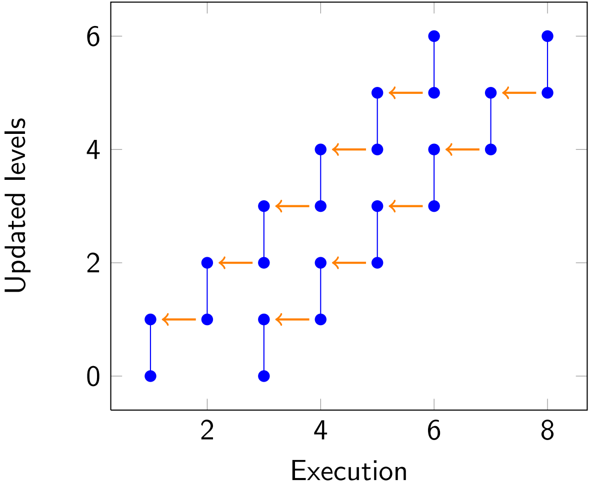
Dependencies of level processings



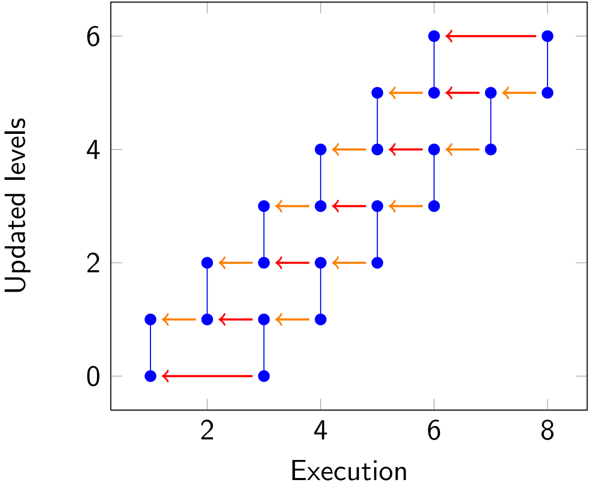
Pipelining the iterations



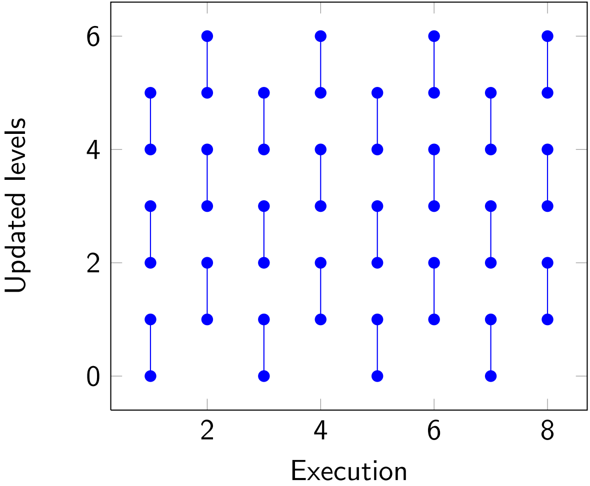
Pipelining the iterations



Pipelining the iterations

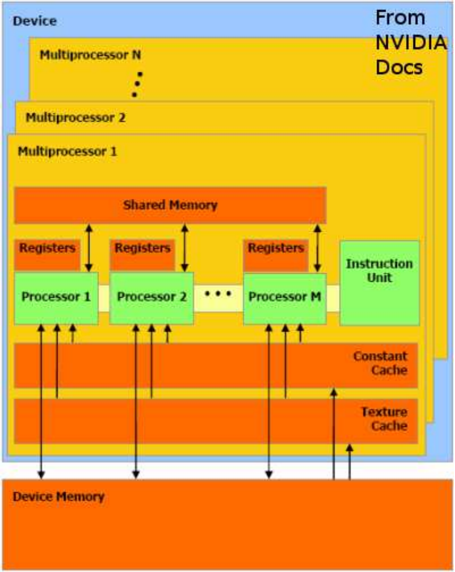


Feeding the whole pipeline



GPU architecture

- On chip shared memory is 100x faster than global memory
- Two levels of parallelism
- How can we map our algorithm on such multi-level architectures?



coarse-grained parallelism

Given the following order...

- B and C two sub-arrays of A
- $B \leq C \iff \max_{i<size(B)} B[i] \leq \min_{i<size(C)} C[i]$
- $B \geq C \iff \min_{i<size(B)} B[i] \geq \max_{i<size(C)} C[i]$

... we redefine the primitives

- $reorder(B, C)$ rearranges B and C to get $B \leq C$
- $select_{min}(B, C, D)$ rearranges B, C and D to get $B \leq C$ and $B \leq D$
- $select_{max}(B, C, D)$ rearranges B, C and D to get $B \geq C$ and $B \geq D$

Multi-level generalization

Multi-level parallelism

- Theses new *coarse-grained* primitives are themselves solvable using a sequential or parallel median-split algorithm.
- Using our algorithm to run the primitives, we re-parallelize at lower scale

Integration on GPU

- A *coarse-grained* version distributes segments to multi-processors.
- A *fine-grained* version locally runs the primitives on each multi-processors using shared memory.

Conclusion

Pros

- Run in $log_2(N)$ iterations in average
- Anytime
- Convergence is easy and fast to detect
- Granularity can be controlled
- Multi-level parallelism

Cons

- We do not know yet how it behaves on real world parallel computers.

Thank you for your attention.
Questions?

References I

- Ibraheem Al-furiah, Srinivas Aluru, Sanjay Goil, and Sanjay Ranka, *Practical algorithms for selection on coarse-grained parallel computers*, IEEE Transactions on Parallel and Distributed Systems **8** (1997), 813–824.
- Manuel Blum, Robert W. Floyd, Vaughan Pratt, Ronald L. Rivest, and Robert E. Tarjan, *Time bounds for selection*, J. Comput. Syst. Sci. **7** (1973), 448–461.
- Duane Merrill and Andrew Grimshaw, *Revisiting sorting for gpgpu stream architectures*, Tech. Report CS2010-03, University of Virginia, Department of Computer Science, Charlottesville, VA, USA, 2010.