

Oblivious Algorithms for Multicores and Network of Processors

R. Chowdhury, **F. Silvestri**, B. Blakeley, V. Ramachandran



Best paper in the algorithmic track

Multicore platforms

- Multicores:
 - Default desktop configuration
 - Collection of cores on a chip communicating through a cache hierarchy under a shared memory.
- Some models in literature:
 - The simpler: one private/shared cache
 - Towards a hierarchy of caches ...
 - Each core with a private cache, sharing a main memory through a shared cache [Blelloch et al. 2008]
 - Multi-BSP: multi-level, which uses latency and gap in a BSP manner [Valiant 2008]

Multicore-obliviousness

- Issues of a multicore algorithm
 - Caching
 - **Shared-memory** parallelism
- Wide ranges of machine parameters:
 - Different core numbers: few, dozen, hundreds,...
 - Different memory hierarchies: level number, cache size, block length,...
 - Portability issues → **multicore-obliviousness!**
- Can we use previous approaches?

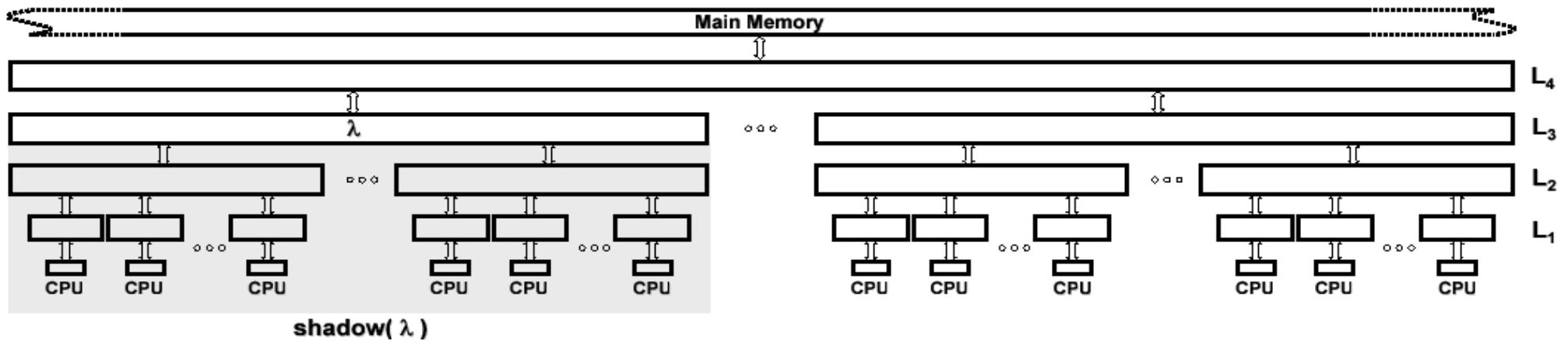
Oblivious approaches

- Cache-Oblivious (C.O.) Algorithms
 - Memory hierarchy
 - Single processor
- Network-Oblivious (N.O.) Algorithms
 - Distributed memory machines
 - Point-to-point communications
 - No memory hierarchy
 - Synchronous
- They are not suitable for multicores

Our results

- A **hierarchical multi-level caching model** (HM) for multicores
- Definition of **multicore-oblivious** (M.O.) algorithms
 - M.O. algorithms have **hints** for the online scheduler
- M.O. algorithms for:
 - Matrix transposition, FFT, Sorting
 - Gaussian Elimination Paradigm
 - List ranking
 - Connected components and other graph problems
- **Relations** between M.O. and N.O. algorithms

The HM model



- Collection of p cores
- $h-1$ cache levels and one arbitrary large main memory
- q_i caches at level i :
 - C_i cache size, B_i block length, $q_{h-1} = 1$, $q_1 = 1$
- **Shadow** of level- i cache L :
 - Cores that share L
 - All level- j ($j < i$) caches between L and cores

The HM model (2)

- A task is **anchored** to cache L
 - If it satisfies space requirements
 - The task and its subtasks are solved by cores in the shadow of L
- Parallelism is expressed by
 - parallel **for** (**pfor**); (e.g. matrix transposition)
 - **Fork/join**; (e.g. matrix multiplication)
- Algorithm performance evaluation:
 - Parallel time complexity: number of executed parallel steps
 - Cache complexity: maximum number of misses of any single cache (one for each level)

Multicore-Oblivious algorithms

- Algorithms that do **not** use multicore parameters
 - Basically, a PRAM algorithm
- Algorithms provide (oblivious) **hints** to the run-time scheduler
 - Provide help on how to schedule parallel tasks
 - Improve performances
- Three types of hints:
 - **Coarse-grained contiguous** (CGC) (used in matrix transposition)
 - **Space-bounded** (SB) (used in GEP)
 - **CGC on SBA** (CGC→SB): is a combination of previous two (used in FFT and sorting)

CGC

- Used for scheduling a sorted collection of parallel subtasks
 - e.g., **pfor**
- CGC:
 - splits the tasks into contiguous chunks of equal size ($> B_1$)
 - distributes contiguous chunks across contiguous cores
- E.g.: **M.O. matrix transposition**
 - consists of two **pfor**'s, as in the N.O. algorithm
 - $O(n^2/p + B_1)$ **optimal** parallel time complexity
 - $O(n^2/(q_i B_i) + B_i)$ **optimal** cache complexity at level i

SB

- Each task t provides an **upper bound** $S(t)$ on the space used by its sub-tasks
- When a task anchored in the level- i cache L forks a sub-task t' , t' is anchored in
 - L if $C_{i-1} < S(t') \leq C_i$
 - L' where L' is a level- k cache ($k < i$), $C_{k-1} < S(t') \leq C_k$, and L' is in the shadow of L
- **Idea**: if each task and its sub-tasks are executed by cores that share the same level- i , then only $O(S(t)/B_i)$ misses are required at level- i
- Used for forking a constant number of tasks
- The M.O. algorithm for GEP uses SB (more later)

CGC→SB

- Combination of previous two hints
- Used when a task forks a large number of sub-tasks
- Sub-tasks are evenly distributed across caches at a suitable lower level in order to fully exploit parallelism
 - Cache size sufficiently large for the task
 - Parallelism exploited
- CGC→SB is used for the FFT of n nodes (\sqrt{n} subtasks)
 - $O(n/p \log n + B_i)$ **optimal** parallel time
 - $O((n/(q_i B_i) \log_{C_i} n))$ **optimal** cache complexity for each level
 - Similar for sorting

GEP

- **Gaussian Elimination Paradigm (GEP)**: a paradigm based on three nested loops of n iterations each

Input: $n \times n$ matrix x , function $f : \mathcal{S} \times \mathcal{S} \times \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{S}$, set Σ_f of triplets $\langle i, j, k \rangle$, with $i, j, k \in [0, n)$.

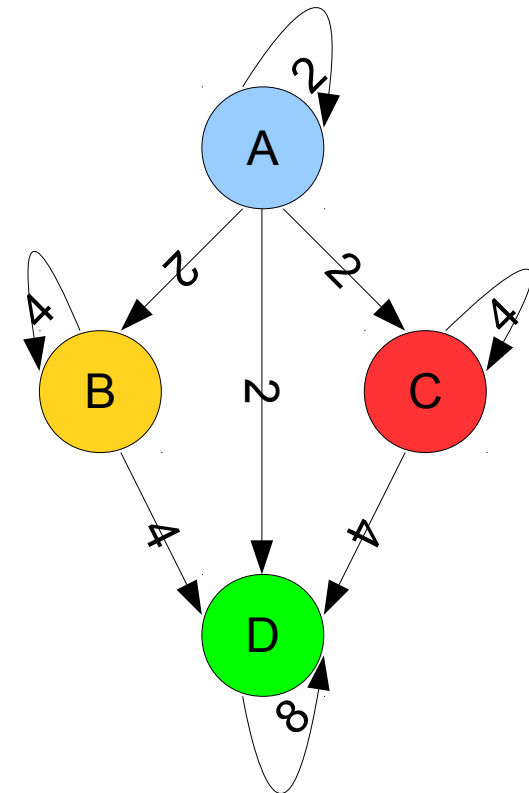
Output: transformation of x defined by f and Σ_f .

```
1: for  $k \leftarrow 0$  to  $n - 1$  do
2:   for  $i \leftarrow 0$  to  $n - 1$  do
3:     for  $j \leftarrow 0$  to  $n - 1$  do
4:       if  $\langle i, j, k \rangle \in \Sigma_f$  then
5:          $x[i, j] \leftarrow f(x[i, j], x[i, k], x[k, j], x[k, k])$ 
```

- Solves many fundamental problems:
 - Matrix multiplication
 - Floyd-Warshall's APSP
 - Gaussian Elimination & LU decomposition without pivoting

I-GEP

- Solved by the C.O. algorithm **I-GEP**
- Parallelized for a 2-level HM in an aware way
- I-GEP solves correctly and efficiently almost all GEP computations
 - C-GEP: extension of I-GEP that solves correctly any GEP computations
- I-GEP consists of 4 functions A, B, C, D that call themselves recursively



M.O. GEP

- The M.O. algorithm for GEP:
 - follows from the parallel version of I-GEP using the SB hint
- $O(n^3 / p)$ **optimal** parallel time complexity
- $O(n^3 / (q_i B_i \sqrt{C_i}))$ **optimal** cache complexity for each level
- M.O. translates into an **optimal N.O. algorithm** as well:
 - Some changes due to concurrent reads (not in the N.O. framework)

List Ranking

- **Problem:** given a list of n nodes, determining the rank of each node
- M.O. algorithm based on ideas of external memory algorithms:
 - Determining an independent set I of the nodes
 - Contract the list by removing I
 - Solve the problem on the contracted list
 - Extend the solution to the removed nodes
- Main problem: **finding the independent set**
 - Use $\log \log n$ coloring
 - $O(1)$ sorts and scans with the CGC and CGC-SB hints

List Ranking (2)

- Complexities:
 - $O(n/(q_i B_i) \log_{C_i} n + (\log \log n)^2 \log(n / B_i))$ cache complexity
 - $O(n \log n / p)$ time complexity
- Using the CGC and CGC→SB hints, we obtain M.O. algorithms for
 - Connected components
 - Euler tour
 - ...
- These algorithms translate into **N.O. algorithms** as well

M.O. vs N.O.

- The M.O. algorithms for matrix transposition and FFT are based on the N.O. ones
- The N.O. for GEP and list ranking are based on the previous M.O. algorithms
- From N.O. to M.O.
 - From message passing to shared-memory
 - Exploit locality in each cache level (not so hard!)
- From M.O. to N.O.
 - Move from shared-memory to message passing
 - **No concurrent read** (not so easy!)

Future work

- Develop other M.O. algorithms
- Do we need other hints?
- What happen if we limit the set of hints? Impossibility results?
- Improve relations between N.O. and M.O. (useful in networks of multicores)
- Missing an optimality theorem as in the C.O. and N.O approaches

QUESTIONS?



THANK YOU!

NOW:

