

Algorithms for Multicore Computing

Vijaya Ramachandran

Department of Computer Sciences

University of Texas at Austin

`vlr@cs.utexas.edu`

`www.cs.utexas.edu/~vlr`

ALGORITHM DESIGN

A large part of computer science deals with developing correct and efficient methods for various computational problems.

Research in **algorithm design and analysis** deals with

- developing efficient algorithms for important problems
- developing new techniques for algorithm design
- developing lower bounds that complement algorithmic upper bounds to derive optimality results
- developing abstract high-level models of computation, and efficient algorithms, algorithmic techniques and lower bounds for these models

RESEARCH THEMES

- Graph algorithms and data structures
- Parallel algorithms
- Models for parallel machines
- cache-efficient algorithms
- Deterministic and randomized algorithms

RESEARCH THEMES

- Graph algorithms and data structures
- Parallel algorithms
- Models for parallel machines
- cache-efficient algorithms
- Deterministic and randomized algorithms

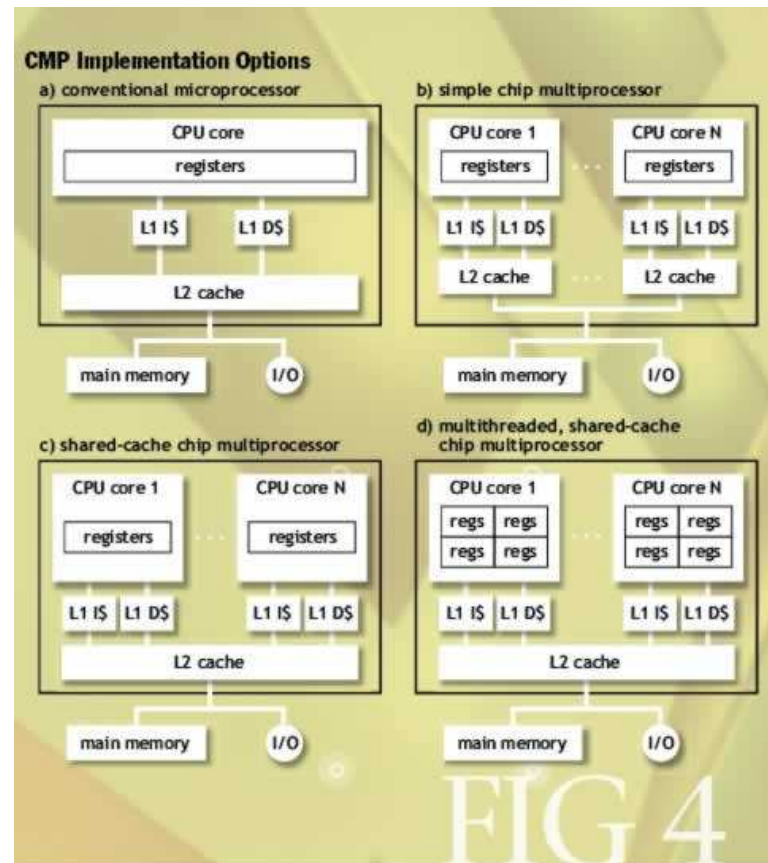
Current Focus: Multicore computing

THE MULTICORE ERA

- Due to power consumption and other reasons, microprocessors are being built with multiple processors on chip.
- Dual-cores are already on most desktops, and number of cores is expected to increase (dramatically) for the foreseeable future
- Computer science research needs to address the multitude of challenges that come with this shift to the multicore era.
- **Our focus:** Developing algorithms, schedulers, lower bounds, models, and a general theory for multicore computing.

MULTICORES / CHIP MULTIPROCESSORS (CMP)

Olukotun & Hammond 2005



CHIP MULTIPROCESSOR (CMP) / MULTICORE

CMP: A collection of processors /cores on a chip communicating through a cache hierarchy

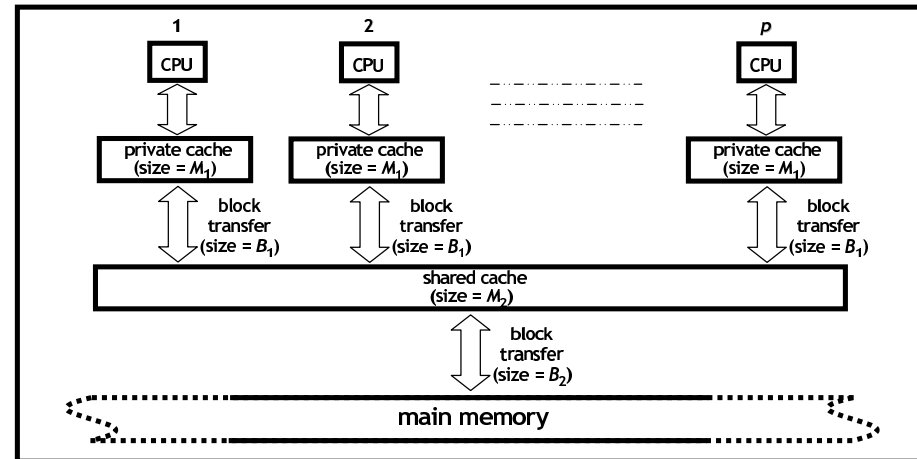
CHIP MULTIPROCESSOR (CMP) / MULTICORE

CMP: A collection of processors /cores on a chip communicating through a cache hierarchy

- Cost of memory access is based on whether the data item is in the cache (i.e., no-cost *cache hit*) or not (i.e., *cache miss* incurring the I/O cost of a block transfer).
- Each cache in the hierarchy has finite size, smaller than the space needed for typical computations.

Efficient algorithms for CMPs must address both parallelism and caching issues.

MULTICORE (3-level)



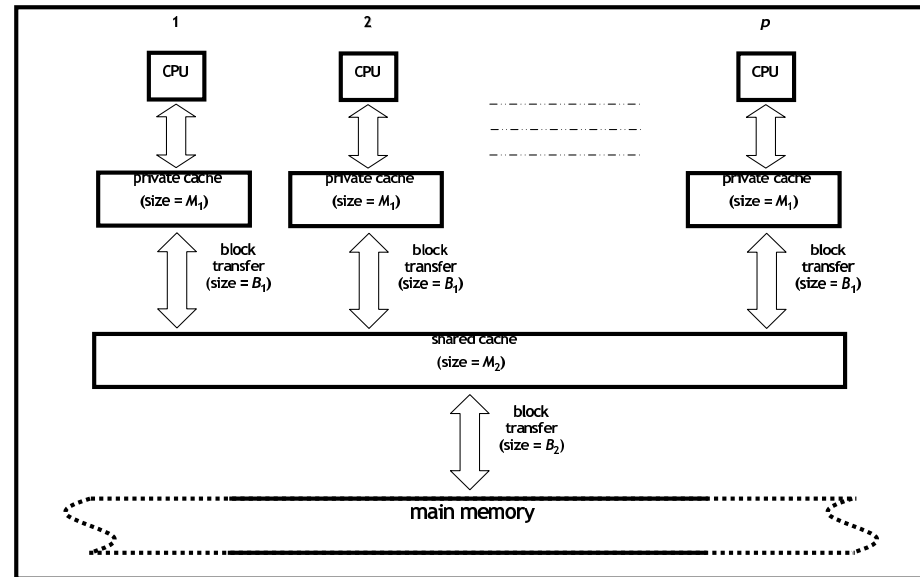
- p processors, each with private L_1 cache of size C_1
- A shared cache L_2 of size C_2 shared by all processor- L_1 units
- An arbitrarily large global shared memory
- Blocksize B_1 between L_1 and L_2 and B_2 between L_2 and memory

MULTICORE PERFORMANCE MEASURES

Need to minimize:

- Number of parallel steps executed
- L_1 cache complexity: # of block transfers between L_1 and L_2
- L_2 cache complexity: # of block transfers between L_2 and main memory

COMPETING DEMANDS OF PRIVATE AND SHARED CACHES

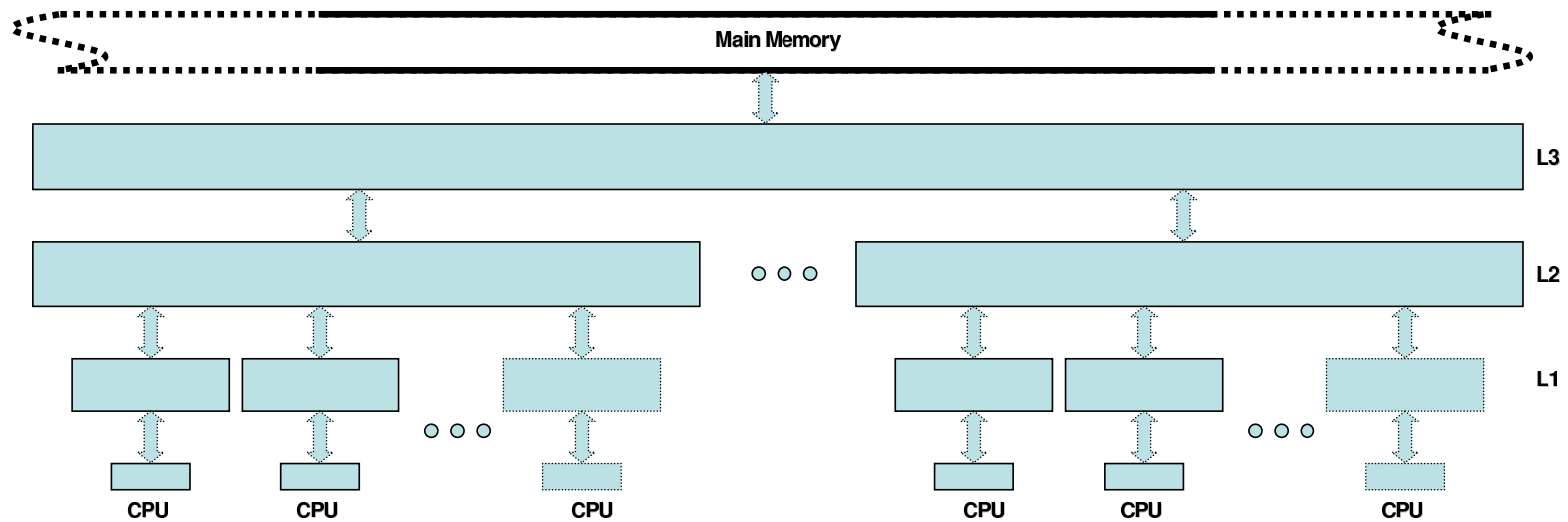


- For good *shared cache* performance, processors should work on the same set of cache blocks at the same time
- For good *private cache* performance, processors should work on disjoint sets of cache blocks at the same time

An efficient multicore algorithm must balance these two competing demands.

MULTILEVEL HIERARCHY FOR MULTICORE

A hierarchy of caches that are shared by increasing number of processors at higher levels of the hierarchy.



RESEARCH AGENDA FOR MULTICORES

- Algorithm design and analysis for multicores
- Effective use of parallelism in cache-efficient sequential algorithms
- Effective algorithms for run-time schedulers for multicores
- ‘Multicore-oblivious’ algorithms
- Multicore data structures
- Lower bounds
- Implementation / experimental evaluation

LONGEST COMMON SUBSEQUENCE (LCS)

A *subsequence* of a sequence X is obtained by deleting zero or more symbols from X .

Example. $Z = abca$ is a subsequence of $X = abcba$

LONGEST COMMON SUBSEQUENCE (LCS)

A *subsequence* of a sequence X is obtained by deleting zero or more symbols from X .

Example. $Z = abca$ is a subsequence of $X = abcba$

Given sequences X and Y , a *longest common subsequence (LCS)* of X and Y is a sequence Z that is a subsequence of both X and Y , and is longest among such subsequences.

Example.

$X = abcba, Y = abcabc \rightarrow Z = abca$ is an LCS of X and Y .

THE LCS DP (REVIEW)

All algorithms for LCS are based on the well-known 2-part dynamic programming (DP) method:

- *Forward Phase*. Compute the cost (i.e., length) of the LCS
- *Main Phase*. Use forward phase to compute an actual LCS sequence by generating the *traceback path*

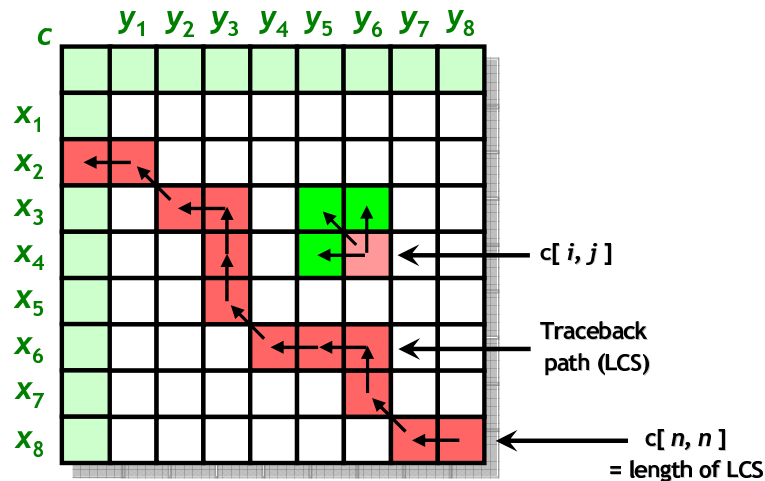
DP with Local Dependencies

The LCS Recurrence

Given: $X = x_1 x_2 \dots x_n$ and $Y = y_1 y_2 \dots y_n$

Fills up an array $c[0 \dots n, 0 \dots n]$ using the following recurrence.

$$c[i, j] = \begin{cases} 0 & \text{if } i = 0 \vee j = 0, \\ c[i - 1, j - 1] + 1 & \text{if } i, j > 0 \wedge x_i = y_j, \\ \max\{c[i, j - 1], c[i - 1, j]\} & \text{otherwise.} \end{cases}$$



Local Dependency:

value of each cell depends only on values of adjacent cells.

COMPLEXITIES OF EARLIER LCS ALGORITHMS

Given $|X| = |Y| = n$, cache size M , block-size B

- **Sequential time and I/O.** The classic LCS DP runs in $\Theta(n^2)$ time, uses $\Theta(n^2)$ space and incurs $\Theta\left(\frac{n^2}{B}\right)$ I/Os. Space can be reduced to $\Theta(n)$ (Hirschberg 1975).
 - No sub-quadratic time algorithm known
 - DP computation requires $\Omega\left(\frac{n^2}{MB}\right)$ I/Os.
- **Parallel time.** The *critical path length* of the DP computation is $2n - 1$, hence any p -processor parallel algorithm based on the DP formulation must incur parallel time $\Omega\left(\frac{n^2}{p} + n\right)$.

EFFICIENT MULTICORE ALGORITHM FOR LCS

Two-phase approach to obtaining efficient multicore algorithm:

- Design a more cache-efficient sequential algorithm
- Build on this algorithm to obtain multicore algorithm

(Research results obtained with recently graduated Ph.D. student Rezaul Alam Chowdhury)

SEQUENTIAL CACHE-EFFICIENT ALGORITHM: FORWARD PHASE

Input is 'virtual' cost table Q with values on input boundary.

- Decompose Q into four equal-sized quadrants
- Generate output boundary of Q by generating the right and bottom boundaries of the four quadrants recursively.
- Return $c[n, n]$ as length of LCS.

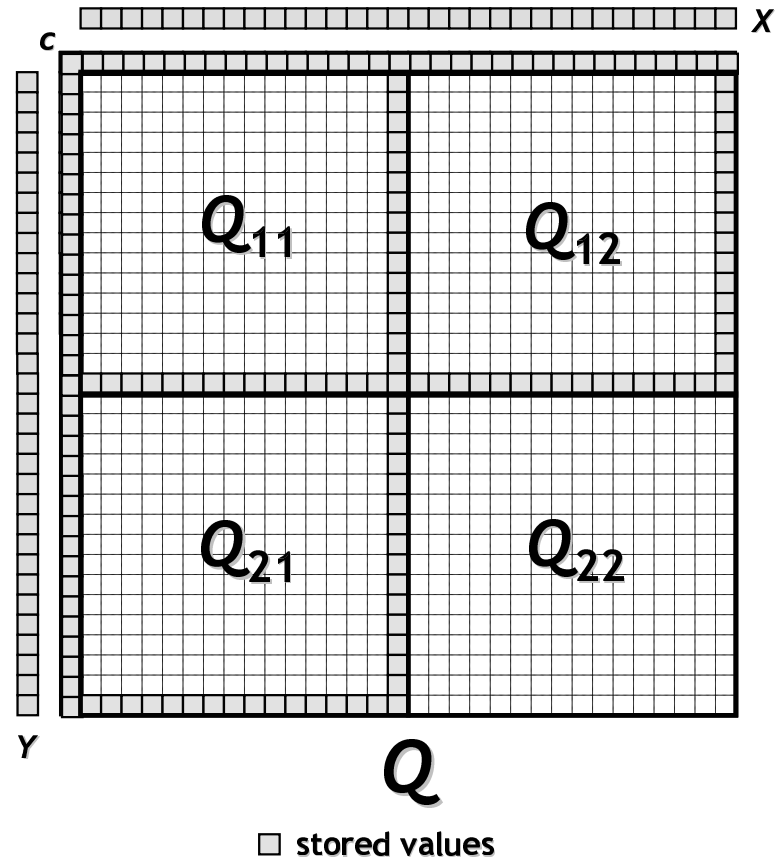
SEQUENTIAL CACHE-EFFICIENT ALGORITHM: FORWARD PHASE

Input is 'virtual' cost table Q with values on input boundary.

- Decompose Q into four equal-sized quadrants
- Generate output boundary of Q by generating the right and bottom boundaries of the four quadrants recursively.
- Return $c[n, n]$ as length of LCS.

'Cache-oblivious'

FORWARD PHASE FOR LCS LENGTH



COMPLEXITY OF FORWARD PASS

Space complexity $S_f(n)$:

$$S_f(n) = \begin{cases} \mathcal{O}(1) & \text{if } n = 1, \\ S_f\left(\frac{n}{2}\right) + \mathcal{O}(n) & \text{otherwise;} \end{cases}$$

Hence, $S_f(n) = \mathcal{O}(n)$

COMPLEXITY OF FORWARD PASS

Space complexity $S_f(n)$:

$$S_f(n) = \begin{cases} \mathcal{O}(1) & \text{if } n = 1, \\ S_f\left(\frac{n}{2}\right) + \mathcal{O}(n) & \text{otherwise;} \end{cases}$$

Hence, $S_f(n) = \mathcal{O}(n)$

I/O Complexity $I_f(n)$: Let α be a suitable constant such that the computation fits in cache if $n \leq \alpha \cdot M$. Then,

$$I_f(n) = \begin{cases} \mathcal{O}\left(1 + \frac{n}{B}\right) & \text{if } n \leq \alpha M, \\ 4I_f\left(\frac{n}{2}\right) + \mathcal{O}\left(1 + \frac{n}{B}\right) & \text{otherwise;} \end{cases}$$

The solution is $I_f(n) = \mathcal{O}\left(1 + \frac{n}{B} + \frac{n^2}{BM}\right)$

LDDP FOR STRING PROBLEMS IN BIOINFORMATICS

The basic LDDP method for LCS generalizes to apply to several string problems in bioinformatics.

We have experimental results for the following problems:

- Pair-wise sequence alignment with affine gap costs
- Median: 3-way sequence alignment with affine gap costs
- RNA secondary structure prediction with simple pseudo-knots

MULTICORE ALGORITHM WITH $\Theta(n)$ CRITICAL PATHLENGTH

Given a parallel algorithm:

- Critical path length T_∞ is the number of parallel steps in the algorithm given unbounded number of processors.

We define

- cache-efficient critical path length I_∞ as the minimum number of parallel steps in a parallel algorithm that also *matches the sequential work and I/O bound*.

MULTICORE METHODOLOGY

- We present a recursive *tiling* version of the DP algorithm parameterized by a tunable *tiling parameter* τ at each level of recursion.

- For each type of caching system (3-level or multi-level), we specify a *tiling sequence* $t[d]$, $d \geq 0$.

Each $t[d]$ is a non-negative integer and specifies the value of the tiling parameter to be used at the d th level of recursion of the algorithm.

TILED-LCS

As with sequential LCS, Tiled-LCS is a two phase algorithm:

- *Forward Phase:* Tiled-Boundary-LCS
- *Main Phase* is the overall algorithm Tiled-LCS, which uses the forward phase

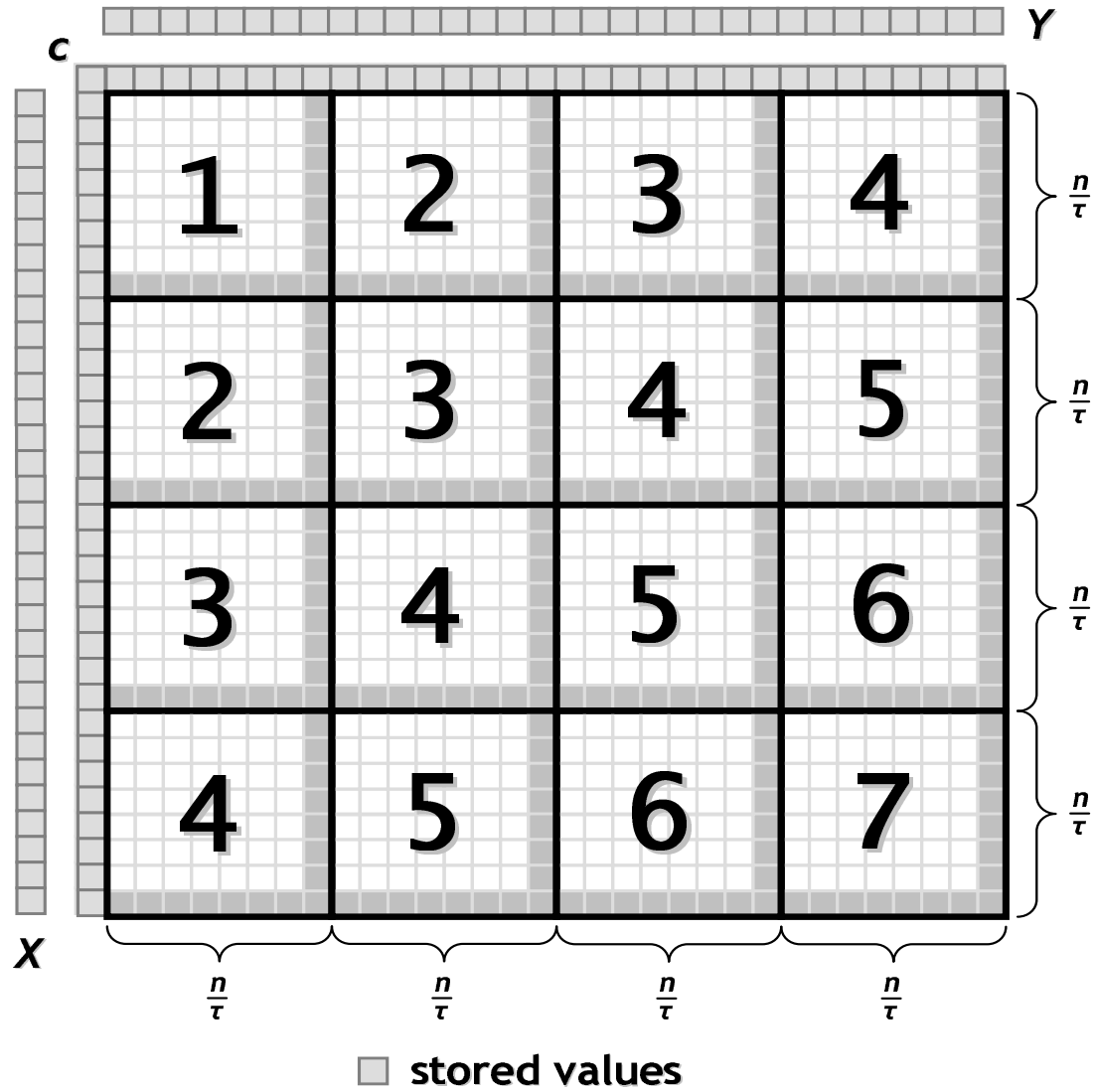
TILED-BOUNDARY-LCS

Input: Virtual cost table Q with cost values available on its input boundaries, and tiling parameter τ .

Output: Cost values on the right and bottom boundaries of Q , and, for each such output position, the entry point of its traceback path on input boundary of Q .

- Divide the table Q into τ^2 equal-sized sub-squares.
- Recursively compute on these sub-squares in $2\tau - 1$ parallel ‘steps’, where parallel step i recursively computes all sub-squares along the i th diagonal in parallel.

TILED-BOUNDARY-LCS



TILING PARAMETERS FOR TILED-BOUNDARY-LCS

Problem	<u>Tiling Parameters</u>	
	Sequential	Multicore
Tiled- Boundary- LCS	$t[d] = 2,$ $\forall d$	$t[r] = p,$ $r = \log(n/C_2),$ $t[d] = 2, d \neq r$

- Algorithm executes sequentially if $t[d] = 2$
- Algorithm executes with parallelism p when $t[d] > 2$.

TILING PARAMETERS FOR TILED-BOUNDARY-LCS

Problem	<u>Tiling Parameters</u>	
	Sequential	Multicore
Tiled- Boundary- LCS	$t[d] = 2,$ $\forall d$	$t[r] = p,$ $r = \log(n/C_2),$ $t[d] = 2, d \neq r$

- Algorithm executes sequentially if $t[d] = 2$
- Algorithm executes with parallelism p when $t[d] > 2$.
- Parallel time is $O\left(\frac{n^2}{p} + n\right)$ with p processors
- I/O complexity is $O\left(\frac{n^2}{CB}\right)$ for both L_1 and L_2 cache.

PARALLEL RUNNING TIME ON MULTICORE

Multicore: At recursion level $r = \log n / C_2$, size of each subproblem is reduced from n to $\frac{n}{2^{\log n / C_2}} = C_2$.

Each such subproblem of size C_2 is executed with $2p - 1$ parallel steps using $p \leq n$ processors, with each step executing a sequential computation of $\left(\frac{C_2}{p}\right)^2$ steps. Hence, parallel time is

$$T(n, p) = \mathcal{O} \left(\frac{n^2}{C_2^2} \cdot (2p - 1) \cdot \frac{C_2^2}{p^2} \right) = \frac{n^2}{p}$$

I/O COMPLEXITY

Multicore L_2 cache: The cache here is similar to the sequential case. If C is cache size, the sequential I/O complexity will hold if a subproblem of size C is always fully completed before another subproblem is started. This is the case in both tiling schedules.

Multicore L_1 cache (C is cache size): By the parallel time complexity, each processor performs $\Theta(n^2/p)$ steps of computation.

The number of I/Os at each cache will be $O(\frac{n^2}{p \cdot C \cdot B})$ if every subproblem passing through the cache has size at least C .

For 3-level multicore, size of each L_1 subproblem is $\frac{C_2}{p} \geq C_1$.

MULTI-LEVEL MEMORY HIERARCHY

- For $h \geq 3$, a hierarchy of caches L_i , $1 \leq i \leq h$, that are successively shared by larger groups of processors.
- At level i , $i \geq 1$
 - the size of each level- i cache is C_i
 - number of level- $(i - 1)$ caches that share a given level- i cache is p_i
- For all i , $C_i \geq p_i \cdot C_{i-1}$ for all i
- Top two levels represent sequential memory hierarchy:
 - Level h : a single infinite shared memory (so $C_h = \infty$)
 - Level $h - 1$: a single shared cache of finite size C_{h-1} (so $p_h = 1$).
- $p_1 = 1$, i.e., each individual processor has a private cache of size C_1 .

MULTI-LEVEL MEMORY HIERARCHY

- For $h \geq 3$, a hierarchy of caches L_i , $1 \leq i \leq h$, that are successively shared by larger groups of processors.
- At level i , $i \geq 1$
 - the size of each level- i cache is C_i
 - number of level- $(i - 1)$ caches that share a given level- i cache is p_i
- For all i , $C_i \geq p_i \cdot C_{i-1}$ for all i
- Top two levels represent sequential memory hierarchy:
 - Level h : a single infinite shared memory (so $C_h = \infty$)
 - Level $h - 1$: a single shared cache of finite size C_{h-1} (so $p_h = 1$).
- $p_1 = 1$, i.e., each individual processor has a private cache of size C_1 .

When $h = 3$, this gives multicore model:

$p_1 = 1$, $p_2 = p$, p caches of size C_1 , one cache of size C_2 , and an infinite shared memory (i.e., $C_3 = \infty$).

Problem	Tiling Parameters for Multilevel Algorithms
LCS & PA	$t[r_i] = p_i, h - 1 \geq i \geq 2,$ $r_i = \text{recursion level where subproblem size is } C_i$ $\& t[d] = 2 \text{ with no parallelism if } d \neq \text{any } r_i$
Median	$t[r_i] = \sqrt{p_i}, h - 1 \geq i \geq 2,$
Parenthesis	$r_i = \text{recursion level where subproblem size is } C_i$
GEP	$\& t[d] = 2 \text{ with no parallelism if } d \neq \text{any } r_i$

CFTR DNA Sequences on AMD Opteron 850 (runtime w.r.t. 8-core PA-CO) gap open cost = 2, gap extension cost = 1, mismatch cost = 1						
Seqs	Seq Lgths ($\times 10^6$)	FASTA	PA-CO (Simple parallelization)			
			1 core	2 cores	4 cores	8 cores
human	1.80	21h 43m	17h 41m	8h 58m	5h 26m	3h 42m
baboon	1.51	(5.87)	(4.79)	(2.43)	(1.47)	(1.00)
human	1.80	20h 15m	14h 28m	7h 20m	4h 28m	3h 12m
chimp	1.32	(6.34)	(4.53)	(2.30)	(1.40)	(1.00)
baboon	1.51	17h 57m	13h 0m	6h 36m	4h 3m	2h 52m
chimp	1.32	(6.22)	(4.51)	(2.29)	(1.40)	(1.00)
human	1.80	27h 55m	21h 47m	11h 2m	6h 37m	4h 32m
rat	1.50	(6.15)	(4.80)	(2.43)	(1.46)	(1.00)
rat	1.50	18h 39m	15h 42m	7h 58m	4h 55m	3h 31m
mouse	1.49	(5.31)	(4.47)	(2.27)	(1.40)	(1.00)

16S Bacterial rDNA Sequences from <i>Pseudanabaena</i> Group							
gap open cost = 2, gap extension cost = 1, mismatch cost = 1							
Lngths	Cost	Knudsen	ukk.alloc	ukk.checkp	<u>MED-CO</u>		
					1 core	4 cores	8 cores
367	299	1,061	396	548	431	181	153
387		(6.93)	(2.59)	(3.58)	(2.82)	(1.18)	(1.00)
388							
378	324	1,136	—	707	460	190	162
388		(7.01)	(—)	(4.36)	(2.84)	(1.17)	(1.00)
403							
342	339	936	—	795	388	166	147
367		(6.37)	(—)	(5.41)	(2.64)	(1.13)	(1.00)
389							
342	432	1,154	—	1,595	464	195	163
370		(7.08)	(—)	(9.79)	(2.85)	(1.20)	(1.00)
474							

OTHER RELATED RESULTS FOR LCS ON MULTICORE

- Overall algorithm to obtain an LCS sequence (not just its length)
- Multicore-oblivious algorithm up to critical path length $\Theta(n \log n)$
- Competitive run-time schedulers for multicores

RESEARCH AGENDA FOR MULTICORES

- Algorithm design and analysis for multicores
- Effective use of parallelism in cache-efficient sequential algorithms
- Effective algorithms for run-time schedulers for multicores
- multicore data structures
- 'Multicore-oblivious' algorithms
- Lower bounds
- Implementation / experimental evaluation