Parallel Algorithms for the Multicore Era

Vijaya Ramachandran
Department of Computer Science
University of Texas at Austin

The Multicore Era

- *Multicores* have arrived and the multicore era represents a *paradigm shift* in general-purpose computing.
- Algorithms research needs to address the multitude of challenges that come with this shift to the multicore era.

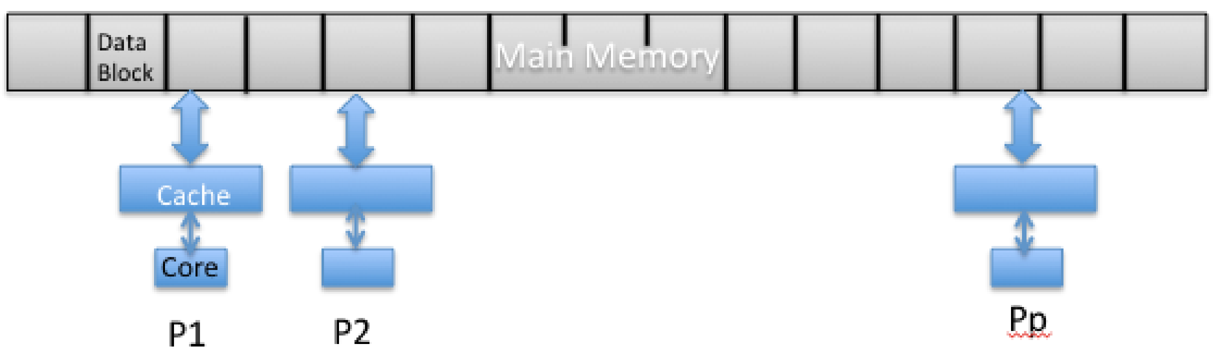
THE PAST: THE VON NEUMANN ERA

An algorithm in the *von Neumann model* assumes a single processor that executes unit-cost steps with unit-cost access to data in memory.

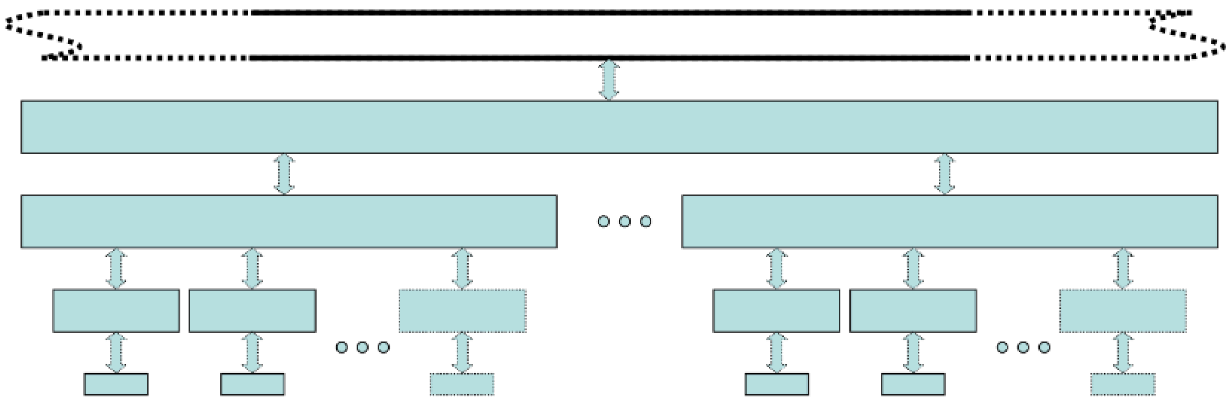
- A very simple abstract model
- Has been very successful for the past several decades
- Has facilitated development of good portable code whose performance by and large matched the theoretical analysis:
  - Sorting: *Quick-sort, Merge-sort, Heap-sort*
  - Graph algorithms: *minimum spanning tree, shortest paths, maximum flow*

THE PRESENT INTO THE FUTURE: MULTICORE ERA

- p cores, each with private cache of size M
- An arbitrarily large global shared memory
- Data organized in blocks of size B.



MULTICORE WITH MULTI-LEVEL CACHE HIERARCHY



Parallel Models and Multicore Modeling

- **Theoretical model:** PRAM
- **Realistic Theoretical Models (communication costs included)**
  - *Fixed interconnection networks*
  - *Bridging Models:*
    BSP, LogP (distributed memory), QSM (shared memory)
- **Modeling Multicores:**
  - *Bulk-synchronous with caching:*
    Multi-BSP
  - HBP Multithreaded algorithms [Cole-Ramachandran 2010, 2012]

Balanced Parallel (BP) Multithreaded Computations

M-Sum$(A[1..n], s)$ % Returns $s = \sum_{i=1}^{n} A[i]$
if $n = 1$ **then return** $s := A[1]$ **end if**
fork(M-Sum$(A[1..n/2], s_1)$; M-Sum$(A[\frac{n}{2}+1..n], s_2)$)
join: **return** $s = s_1 + s_2$

- *Sequential execution* computes recursively in a dfs traversal of this computation tree.
- Forked tasks can run in parallel.
- Runs on $p \geq 1$ cores in $O(n/p + \log p)$ parallel steps by forking $\log p$ times to generate p parallel tasks.

M-Sum is an example of a *Balanced Parallel (BP)* computation.

HIERARCHICAL BALANCED PARALLEL (HBP) COMPUTATIONS

Depth-n-MM(X, Y, Z, n) % Returns $n \times n$ matrix $Z = A \cdot B$

if $n = 1$ **then return** $Z \leftarrow Z + X \cdot Y$ **end if**
fork(
  DEPTH-N-MM$(X_{11},\ Y_{11},\ Z_{11},\ n/2)$;
  DEPTH-N-MM$(X_{11},\ Y_{12},\ Z_{12},\ n/2)$;
  DEPTH-N-MM$(X_{21},\ Y_{11},\ Z_{21},\ n/2)$;
  DEPTH-N-MM$(X_{21},\ Y_{12},\ Z_{22},\ n/2)$)
join
fork(
  DEPTH-N-MM$(X_{12},\ Y_{21},\ Z_{11},\ n/2)$
  DEPTH-N-MM$(X_{12},\ Y_{22},\ Z_{12},\ n/2)$
  DEPTH-N-MM$(X_{22},\ Y_{21},\ Z_{21},\ n/2)$
  DEPTH-N-MM$(X_{22},\ Y_{22},\ Z_{22},\ n/2)$)
join

Depth-n-MM is an example of *Hierarchical Balanced Parallel (HBP)* computation.

Multithreaded Computations

- Many programming languages support multithreading.

- Current run-time environments have run-time schedulers that schedule available parallel tasks on idle cores.

  Typically, a core is not left idle if there is an available parallel task.

  - Multithreaded computations can be scheduled by most run-time schedulers since a thread generates a parallel task in its task queue at each fork in the computation.

  - Bulk-synchronous computations impose a specific scheduler for the algorithms; cores may often idle at the global synchronization point, waiting for all other cores to complete the synchronization.

Communication Costs: Cache Misses and False Sharing

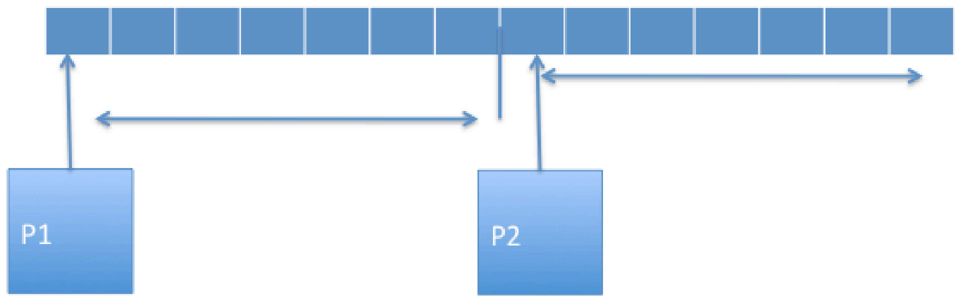
Cache Miss. A cache miss occurs in a computation if the data item being read is not in cache.

This results in a delay while the block that contains the data item is read into cache (by evicting a data item present in cache – we assume an optimal cache replacement policy).

Cache misses can occur in both sequential and parallel executions.

False Sharing . False sharing occurs if the same block of data is accessed by two or more processors in a parallel environment, and at least one of these processors writes into a location in the block.

Block of size B shared by P1 and P2



False Sharing: B/2 cache misses incurred by P1 and by P2

Each of P_1 and P_2 could incur the cost of $B/2$ cache misses as the block *ping-pongs* between their caches in order to serve their write requests.

- *False-sharing* is an inherent consequence of shared-memory architecture, where data is pre-packaged in blocks.

HBP and Block-Resilient HBP

[Cole-Ramachandran 2010, 2012]

- *Hierarchical Balanced Parallel (HBP)* computations use balanced fork-join trees and build richer computations through sequencing and recursion.
- Design HBP with good sequential cache complexity, and good parallelism.
- Incorporate *block resilience* in the algorithm to guarantee low overhead due to false sharing.
- Design *resource-oblivious* algorithms (i.e., with no machine parameters in the algorithms) that are analyzed to perform well (across different schedulers) as a function of the number of parallel tasks generated by the scheduler.

Block Resilient HBP Algorithm	$f(r)$	$L(r)$	T_∞	$Q(n, M, B)$
KNOWN				
Scans (MA, PS)	1	1	$O(\log n)$	$O(n/B)$
Matrix Transposition (in BI)	1	1	$O(\log n)$	$O(n/B)$
Strassen's MM (in BI)	1	1	$O(\log^2 n)$	$O(n^\lambda/(B \cdot M^{\frac{\lambda}{2}-1}))$
RM to BI	$\sqrt{r}$	1	$O(\log n)$	$O(n^2/B)$
Direct BI to RM	$\sqrt{r}$	$\sqrt{r}$	$O(\log n)$	$O(n^2/B)$
MODIFIED				
BI-RM (gap RM)	$\sqrt{r}$	gap	$O(\log n)$	$O(n^2/B)$
FFT	$\sqrt{r}$	1	$O(\log n \cdot \log\log n)$	$O(\frac{n}{B}\log_M n)$
List Ranking	$\sqrt{r}$	1	$O(\log^2 n \cdot \log\log n)$	$O(\frac{n}{B}\log_M n)$
Connected Comp.*	$\sqrt{r}$	1	$O(\log^3 n \cdot \log\log n)$	$O(\frac{n}{B}\log_M n \cdot \log n)$
Depth-n-MM	1	1	$O(n)$	$O(n^3/(B\sqrt{M}))$
NEW				
BI-RM for FFT*	$\sqrt{r}$	1	$O(\log n)$	$O(\frac{n^2}{B}\log_M n)$
Sort (SPMS)	$\sqrt{r}$	1	$O(\log n \cdot \log\log n)$	$O(\frac{n}{B}\log_M n)$

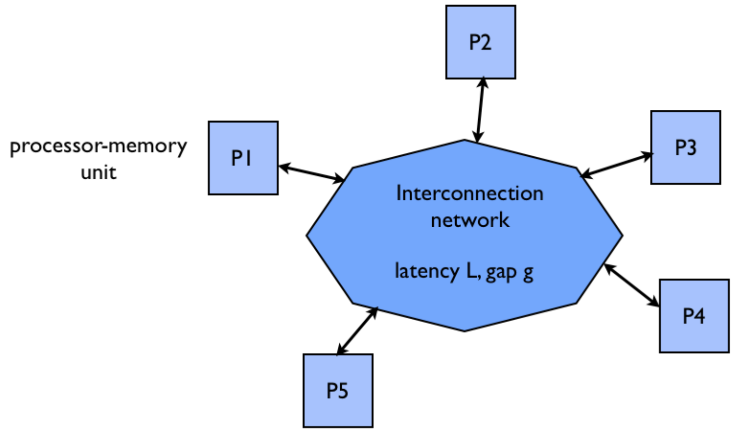
Bounds for Randomized Work Stealing (RWS)

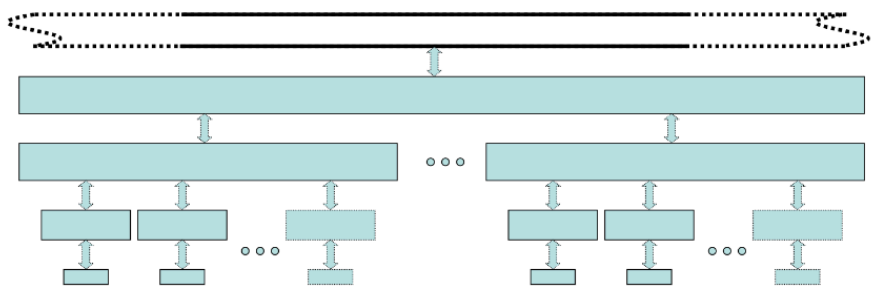
Block Resilient HBP Algorithm	RWS Expected # Steals, S with FS Misses [Cole-R12c]	Cache Misses with S Steals [Cole-R12a]	FS Misses [Cole-R12b]
Scans, MT	$p \cdot (\log n + \frac{b}{s} B)$	$Q + S$ [FS06,CR12a]	$S \cdot B$
RM to BI	$p \cdot (\log n + \frac{b}{s} B)$	$Q + S \cdot B$	$S \cdot B$
MM, Strassen	$p \cdot (\log^2 n + \frac{b}{s} B \log n)$	$Q + S^{\frac{1}{3}} \frac{n^2}{B} + S$	$S \cdot B$
Depth-n-MM	$p \cdot (n + \frac{b}{s} n\sqrt{B})$	$Q + S^{\frac{1}{3}} \frac{n^2}{B} + S$ [FS06,CR12a]	$S \cdot B$
I-GEP	$p \cdot (n \cdot \log^2 n + \frac{b}{s} n\sqrt{B})$	$Q + S^{\frac{1}{3}} \frac{n^2}{B} + S$ [FS06,CR12a]	$S \cdot B$
BI to RM for MM and FFT	$p \cdot (\log n + \frac{b}{s} B)$	$Q + S \cdot B + \frac{n^2}{B} \log \log_B n$	$S \cdot B$
LCS	$p(1 + \frac{b}{s}) \cdot n^{\log_2 3}$	$Q + n\sqrt{S}/B + S$ [FS06,CR12a]	$S \cdot B$
FFT, sort	$p \cdot (\log n \cdot \log \log n$ $+ \frac{b}{s} B \log_B n)$	$C_{\text{sort}} = O(Q + S \cdot B$ $+ \frac{n}{B} \frac{\log n}{\log[(n \log n)/S]})$	$S \cdot B$
List Ranking	$p \cdot \log n \cdot \log \log n$ $\cdot (\log n + \frac{b}{s} B)$	$Q + C_{\text{sort}} \cdot \log n$	$S \cdot B$

Bounds for a Simple Centralized Scheduler $\mathcal{S}_C$

Block Resilient HBP Algorithm	$L(r)$	**Fs Misses with S Parallel Tasks**	**Value of S for Scheduler $\mathcal{S}_C$**	**Cache Misses w/ S Parallel Tasks**
Scans (PS, MT)	1	$B \cdot S$	p	$Q + S$
Depth-n-MM	1	$B \cdot S$	$p^{3/2}$	$Q + S^{\frac{1}{3}} \frac{n^2}{B} + S$
MM, Strassen	1	$B \cdot S$	$p \log p$	$Q + S^{\frac{1}{3}} \frac{n^2}{B} + S$
RM to BI	1	$B \cdot S$	p	$Q + S \cdot B$
Direct BI to RM	$\sqrt{r}$	$\frac{n}{\sqrt{p}} B \cdot S$	p	$Q + S \cdot B$
BI-RM (gap RM)	gap	$\min\{\frac{n}{\sqrt{p}}, B \log^2 B\} BS$	p	$Q + S \cdot B$
BI-RM for FFT	1	$B \cdot S$	$p \cdot \log \frac{\log n}{\log(n^2/p)}$	$Q + SB + \frac{n^2}{B} \log \log_B n$
FFT, SPMS Sort	1	$B \cdot S$	$p \cdot \frac{\log n}{\log(n/p)}$	$Q + SB + \frac{n}{B} \frac{\log n}{\log \left[\frac{(n \log n)}{S} \right]}$

BSP and Multi-BSP





BULK-SYNCHRONOUS VERSUS MULTITHREADED ALGORITHMS

- Bulk-synchronous parallel and cache-efficient algorithms.
  - Bulk synchronous programming style does not exploit the available support for run-time schedulers that can be used to minimize idling processors. *(Use multithreaded algorithms instead.)*
- Multi-BSP model [Valiant 2008]: Bulk-synchronous with caches but uses L and g instead of cache misses.
  - *Bulk-synchronous programming style.*
  - *Communication Cost.* l (L) and g versus cache and false sharing misses.

- *False-sharing* is an inherent consequence of shared-memory architecture, where data is pre-packaged in blocks.
  - Block-resilient algorithms [CR12] address this feature, and use block-resiliency in algorithms to reduce the cost of false-sharing.

Heterogeneous Computing Environments

Most parallel computing environments are not homogeneous:

- Supercomputers are often heterogeneous, e.g., a network of multicores.
  - Many HBP algorithms have complementary 'network-oblivious' algorithms [CSBR10], and so can port across distributed and shared memory environments.
- A multicore typically runs multiple tasks (e.g., o.s. tasks run concurrently with the application parallel algorithm).
  - Multithreaded algorithms offer flexibility to distribute tasks to processors according to their availability for this computation (in contrast to synchronous or bulk synchronous parallel algorithms).

- The *block resilient* techniques for minimizing false sharing reduce the data boundaries at which parallel tasks interact, and this offers the promise of efficient data accesses across heterogeneous platforms.

CHALLENGES AHEAD

Our Initial Contributions:

- A suitable framework for efficient multicore algorithms: *HBP multithreaded algorithms*.
- Suitable cost measures: *good parallelism with work- and cache-efficiency (including false sharing)*.
- Portable algorithms independent of machine parameters: *resource-oblivious algorithms*

The Work Ahead:

- Design a collection of algorithmic techniques that give rise to efficient multicore algorithms for important computational problems.
- Analyze and develop efficient run-time schedulers that schedule the available parallel tasks efficiently, in a distributed environment.
- Develop framework, techniques and analyses to address heterogeneity in parallel computing environments, fault-tolerance, energy efficiency, . . .