Algorithm Design for Multicore Processors

. . . a high-level approach

Peter Krusche Alexander Tiskin

Department of Computer Science and DIMAP
University of Warwick, Coventry, CV4 7AL, UK

29/May/2009

THE UNIVERSITY OF WARWICK

Our Motivation

We start with a simple model for designing parallel algorithms.

Question 1: Are these algorithms suitable for multicore processors?

Question 2: Can our simple algorithms be realized efficiently on hierarchical machines?

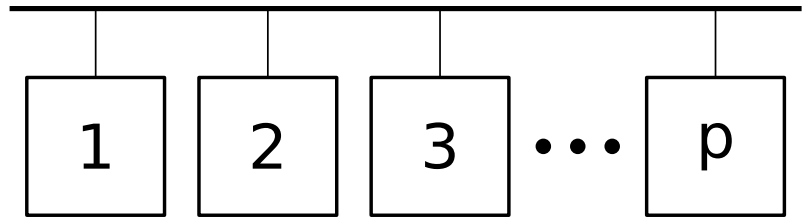
Basic Model Ingredients

A BSP computer with p processors/ cores/threads.

External and per-processor memory.

Vector instructions on each processor.

Superstep-style code execution.



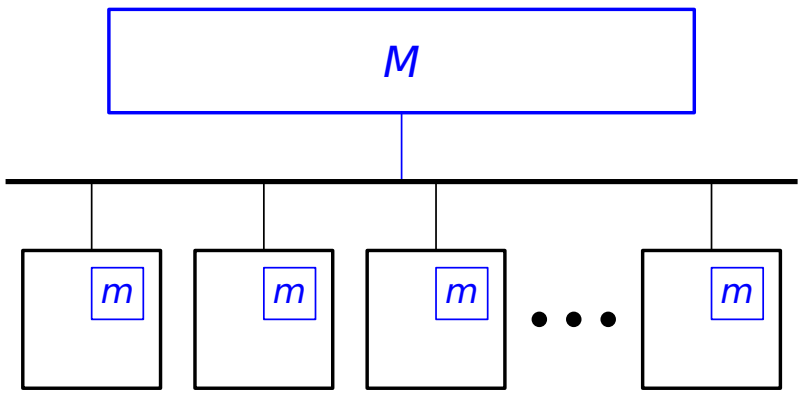
Basic Model Ingredients

A BSP computer with p processors/ cores/threads.

External and per-processor memory.

Vector instructions on each processor.

Superstep-style code execution.



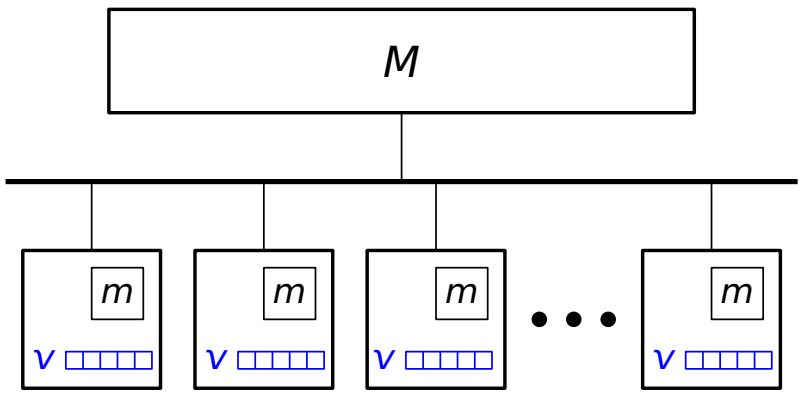
Basic Model Ingredients

A BSP computer with p processors/ cores/threads.

External and per-processor memory.

Vector instructions on each processor.

Superstep-style code execution.



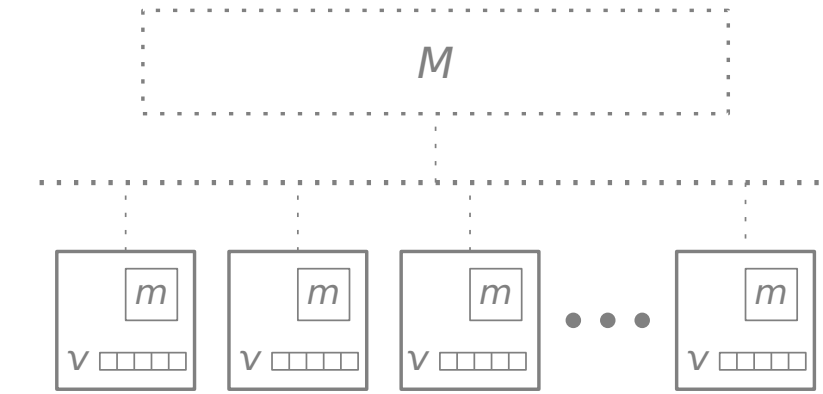
Basic Model Ingredients

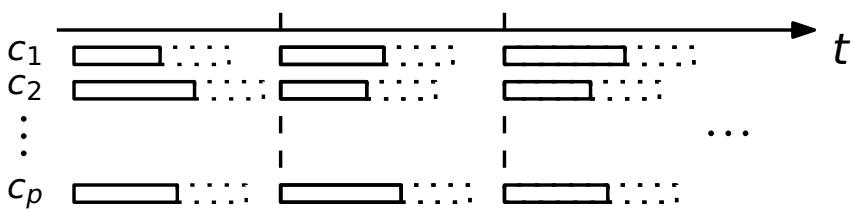
A BSP computer with p processors/ cores/threads.

External and per-processor memory.

Vector instructions on each processor.

Superstep-style code execution.





Algorithm Ingredients: Sequential Algorithms

We have a problem of size n. We study

- ... the total work $\mathcal{W}(n)$
- ... the memory requirement $\mathcal{M}(n)$
- ... the input/output size: $\mathcal{I}(n)$

We assume that the input and output are stored in the environment (e.g. external memory).

Example ($n \times n$ Matrix Multiplication)

The problem size is n.

Standard (non-Strassen) algorithm:
$\mathcal{W}(n) = O(n^3)$, $\mathcal{I}(n) = O(n^2)$, and
$\mathcal{M}(n) = O(n^2)$.

Algorithm Ingredients: Sequential Algorithms

We have a problem of size n. We study

- ... the total work $\mathcal{W}(n)$
- ... the memory requirement $\mathcal{M}(n)$
- ... the input/output size: $\mathcal{I}(n)$

We assume that the input and output are stored in the environment (e.g. external memory).

Example ($n \times n$ Matrix Multiplication)

The problem size is n.

Standard (non-Strassen) algorithm:
$\mathcal{W}(n) = O(n^3)$, $\mathcal{I}(n) = O(n^2)$, and
$\mathcal{M}(n) = O(n^2)$.

Algorithm Ingredients: Vector Parallelism

Each processor has vector instructions which work on v elements in parallel.

Example (Inner product of two n-vectors)

If v is constant, this can be implemented to give runtime $O(n/v)$.

Algorithm Ingredients: Parallel Algorithms

Across all supersteps of the algorithm, we look at

- . . . the computation time: $W(n, p)$
- . . . the communication cost: $H(n, p)$
- . . . the local memory cost: $M(n, p)$

How to do these costs relate to parallelism on multicore processors ?

Algorithm Ingredients: Parallel Algorithms

Across all supersteps of the algorithm, we look at

- . . . the computation time: $W(n, p)$
- . . . the communication cost: $H(n, p)$
- . . . the local memory cost: $M(n, p)$

How to do these costs relate to parallelism on multicore processors ?

Classical Criterion: Work Optimality

An algorithm is work-optimal (w.r.t. a sequential algorithm) if

$$W(n,p) = O(\mathcal{W}(n)/p).$$

Example (Matrix Multiplication)

Algorithms achieving $W(n,p) = O(n^3/p)$ are work-optimal w.r.t. the sequential $O(n^3)$ method.

We have absolute work-optimality if $\Omega(\mathcal{W}(n))$ is a lower bound on the total work for the given problem, and the given model.

Classical Criterion: Work Optimality

An algorithm is work-optimal (w.r.t. a sequential algorithm) if

$$W(n, p) = O(\mathcal{W}(n)/p).$$

Example (Matrix Multiplication)

Algorithms achieving $W(n, p) = O(n^3/p)$ are work-optimal w.r.t. the sequential $O(n^3)$ method.

We have absolute work-optimality if $\Omega(\mathcal{W}(n))$ is a lower bound on the total work for the given problem, and the given model.

Classical Criterion: Work Optimality

An algorithm is work-optimal (w.r.t. a sequential algorithm) if

$$W(n, p) = O(\mathcal{W}(n)/p).$$

Example (Matrix Multiplication)

Algorithms achieving $W(n, p) = O(n^3/p)$ are work-optimal w.r.t. the sequential $O(n^3)$ method.

We have absolute work-optimality if $\Omega(\mathcal{W}(n))$ is a lower bound on the total work for the given problem, and the given model.

Scalable Communication and Memory

Scalable communication:

An algorithm achieves asymptotically scalable communication if $H(n, p) = O(\mathcal{I}(n)/p^c)$

(assuming $0 < c \leq 1$).

Scalable memory:

An algorithm achieves asymptotically scalable memory if $M(n, p) = O(\mathcal{M}(n)/p^c)$.

(assuming $0 < c \leq 1$).

Scalable Communication and Memory

Scalable communication:

An algorithm achieves asymptotically scalable communication if $H(n, p) = O(\mathcal{I}(n)/p^c)$

(assuming $0 < c \leq 1$).

Scalable memory:

An algorithm achieves asymptotically scalable memory if $M(n, p) = O(\mathcal{M}(n)/p^c)$.

(assuming $0 < c \leq 1$).

Why Scalable Communication and Memory?

Scalable memory allows to increase number of virtual threads until subproblems fit into caches.

Scalable communication models algorithmic bus bandwidth sharing.

Algorithms with scalable memory and communication can be simulated efficiently on a hierarchical parallel machine.

Why Scalable Communication and Memory?

Scalable memory allows to increase number of virtual threads until subproblems fit into caches.

Scalable communication models algorithmic bus bandwidth sharing.

Algorithms with scalable memory and communication can be simulated efficiently on a hierarchical parallel machine.

Why Scalable Communication and Memory?

Scalable memory allows to increase number of virtual threads until subproblems fit into caches.

Scalable communication models algorithmic bus bandwidth sharing.

Algorithms with scalable memory and communication can be simulated efficiently on a hierarchical parallel machine.

Examples

Example (Parallel prefix on n values)

A BSP Algorithm with

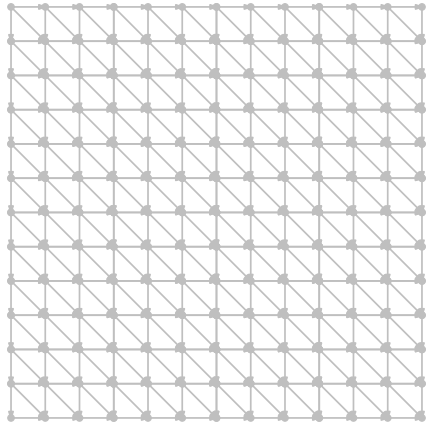
$$\begin{aligned} W(n,p) &= O(n/p), \\ H(n,p) &= O(p) \\ M(n,p) &= O(n/p), \text{ and} \\ S &= O(1). \end{aligned}$$

This is work-optimal and also achieves scalable communication and memory if $n > p^2$.

Examples

Example (Grid dag dynamic programming)

Work-optimality is no problem, neither is using vector parallelism.

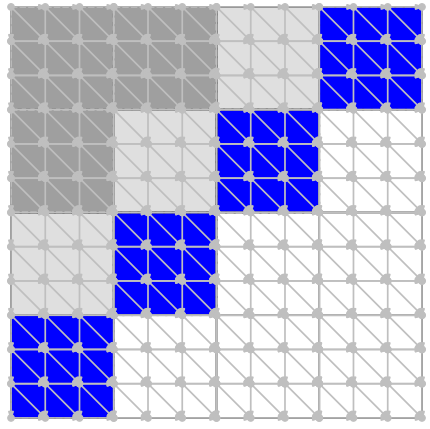


However: No algorithm can achieve work-optimality and scalable communication at the same time! [Papadimitriou/Ullman:87]

Examples

Example (Grid dag dynamic programming)

Work-optimality is no problem, neither is using vector parallelism.

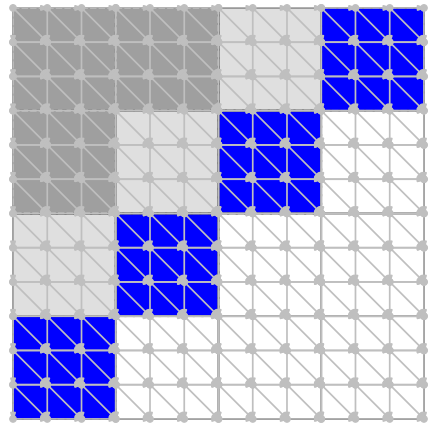


However: No algorithm can achieve work-optimality and scalable communication at the same time! [Papadimitriou/Ullman:87]

Examples

Example (Grid dag dynamic programming)

Work-optimality is no problem, neither is using vector parallelism.



However: No algorithm can achieve work-optimality and scalable communication at the same time! [Papadimitriou/Ullman:87]

Examples

Example (Longest common subsequences)

For two input sequences of length n, we can achieve [KT:2007]

$$\begin{aligned} W(n,p) &= O(n^2/p), \\ H(n,p) &= O(n/\sqrt{p} \cdot \log p) \\ M(n,p) &= O(n/\sqrt{p}), \text{ and} \\ S &= O(\log p). \end{aligned}$$

. . . allowing to use v-vector-parallelism in each thread.

This is work-optimal and achieves scalable communication/memory.

An Open Problem

Longest Increasing Subsequences

Best sequential algorithm: patience sorting in $O(n \log n)$.

Various "parallel" algorithms exist:

- $W(n, p) = O((n \log n)/p)$ if $p < \sqrt{l}$ with l length of result sequence.
- $W(n, p) = O(n \log(n/p))$.

Most scalable algorithm known:
$W(n, p) = O(n^{1.5}/p)$

$\Rightarrow$ Work-optimality can be tricky!

Summary

We can study the suitability of BSP-style algorithms for multicore systems by looking at work-optimality, scalable communication and scalable memory.

These properties in algorithms can be improved through non-trivial theoretical work based on a simple model.

Thanks! Questions?