Department of Information Engineering
University of Padova

Exploiting Fine Grained Parallelism in SPE

E. Milani N. Zago

ICTCS, Varese, September 21st, 2012

Table of contents

Introduction

Fundamental Problem

- RAM does not capture memory access complexity
- Computational complexity is not enough on actual machines

Strategy

- Machine models (memory and processor)
- Algorithmic techniques

Questions

- What can be imported from other settings/contexts?
- Is it possible/convenient to exploit parallelism in a scalar setting?

Models and Algorithms

Two major strategies to cope with latency:

- Temporal/Spatial Locality
- Concurrency

Models and Algorithms

Two major strategies to cope with latency:

- Temporal/Spatial Locality

  $\Rightarrow$ Hierarchical Memories, Block Transfer

- Concurrency

  $\Rightarrow$ Pipelined Memories

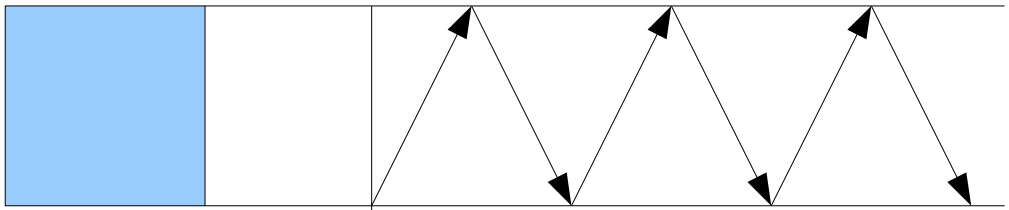
Models and Algorithms

Two major strategies to cope with latency:

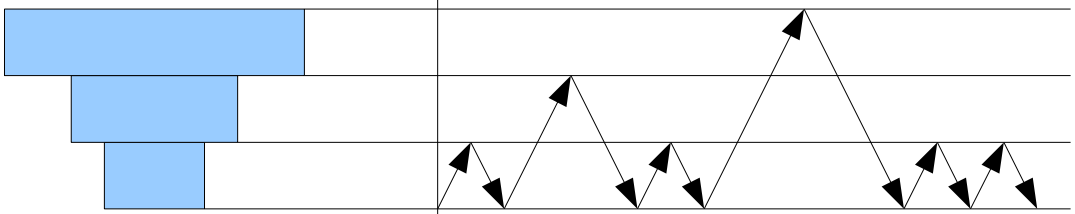
- Temporal/Spatial Locality
  $\Rightarrow$ Hierarchical Memories, Block Transfer
  $\Rightarrow$ Memory access function $a(x)$
- Concurrency
  $\Rightarrow$ Pipelined Memories
  $\Rightarrow$ Constant access request rate

Memory Models

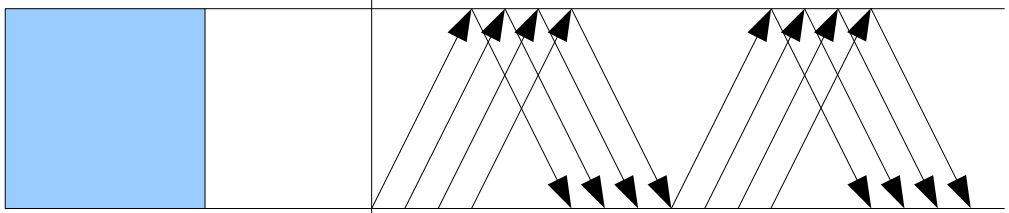
RAM with latency



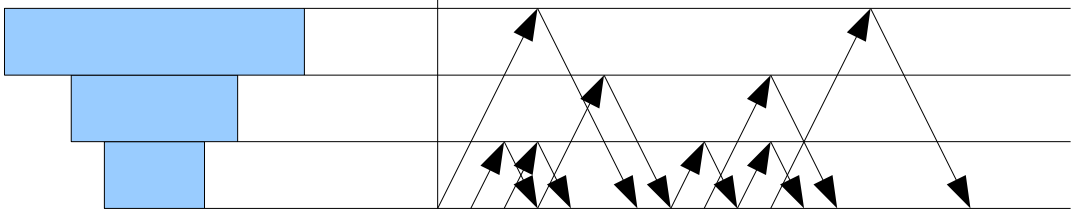
Hierarchical Memory



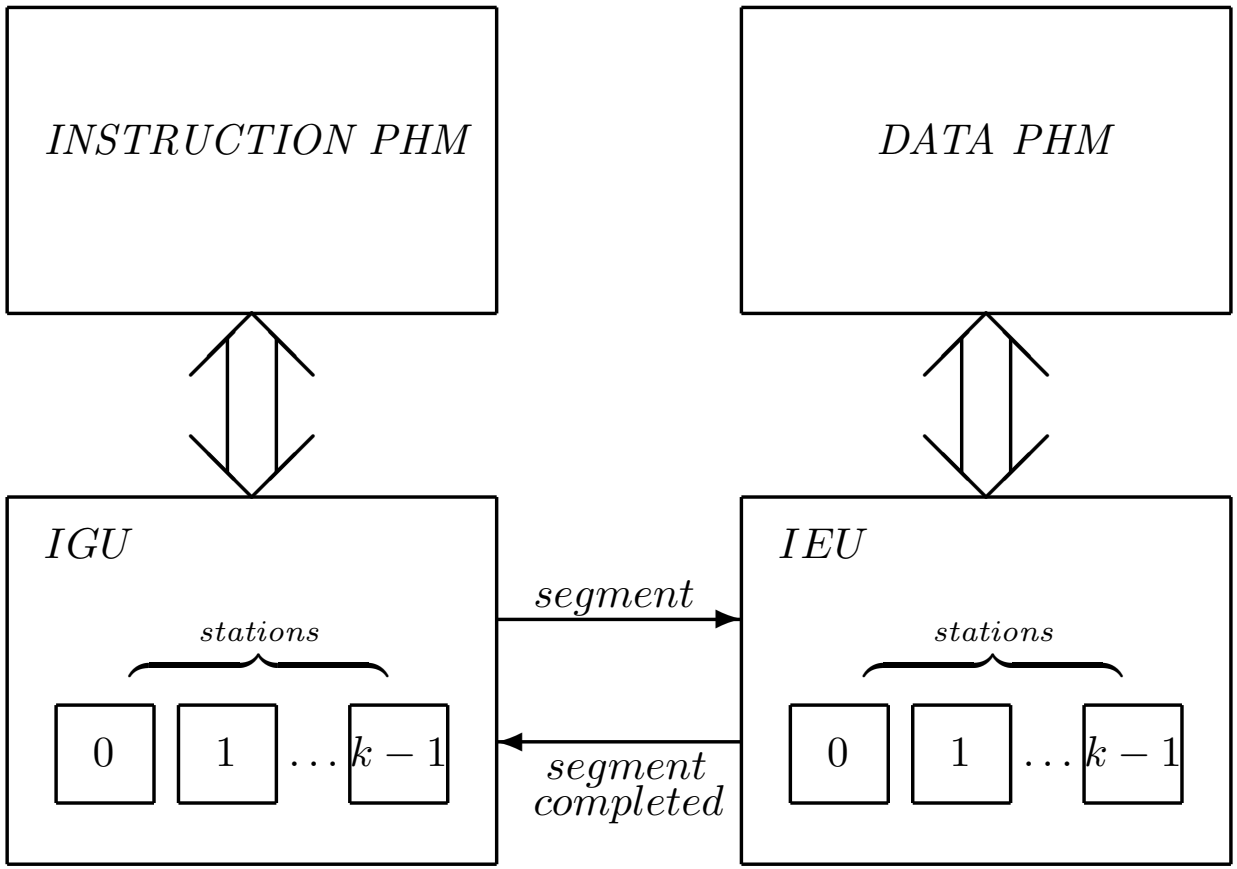
Pipelined Memory



Pipelined Hierarchical Memory



Speculative Prefetcher and Evaluator



Speculative Prefetcher and Evaluator

- Instructions are executed in segments of variable size (`segmentsize()`)

Speculative Prefetcher and Evaluator

- Instructions are executed in segments of variable size (`segmentsize()`)
- No slow down because of data dependencies...

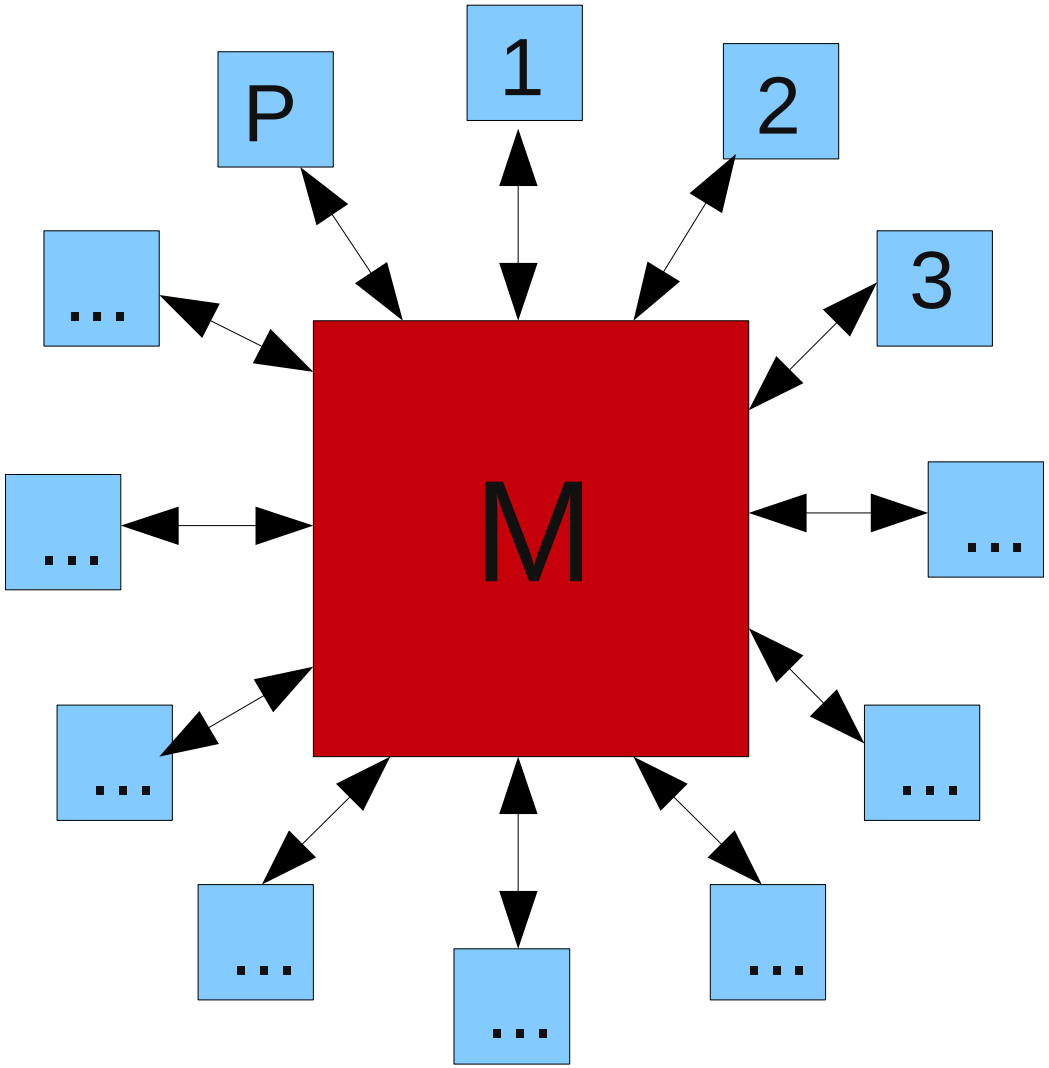
Speculative Prefetcher and Evaluator

- Instructions are executed in segments of variable size (`segmentsize()`)
- No slow down because of data dependencies...
- ...or even $O(1)$ address dependence depth

Speculative Prefetcher and Evaluator

- Instructions are executed in segments of variable size (`segmentsize()`)
- No slow down because of data dependencies...
- ...or even $O(1)$ address dependence depth
- Enhancements such as dynamic loop unrolling and branch prediction are allowed

Parallel Random Access Machine



Parallel Random Access Machine

Many equivalent flavours: SIMD, MIMD...

The actual difference is in how shared memory is managed:

- EREW: exclusive read, exclusive write
- CREW: concurrent read, exclusive write
- CRCW: concurrent read, concurrent write (contention policy)

Work-Time Framework

- *Parallel Programming Model* which targets PRAMs

Work-Time Framework

- *Parallel Programming Model* which targets PRAMs
- `pardo` statement defines *parallel steps*

Work-Time Framework

- *Parallel Programming Model* which targets PRAMs
- `pardo` statement defines *parallel steps*
- sets $s_1, \ldots, s_T$ of instructions on M cells of memory

Work-Time Framework

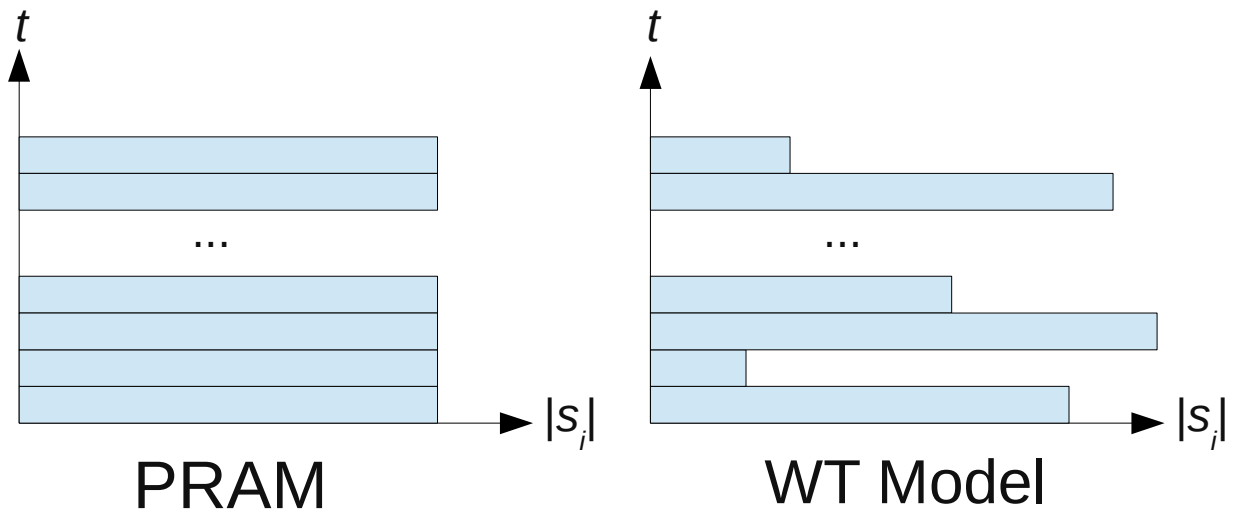
- *Parallel Programming Model* which targets PRAMs
- `pardo` statement defines *parallel steps*
- sets $s_1, \ldots, s_T$ of instructions on M cells of memory
- in general, each s_i has a different size p_i

Work-Time Framework

- *Parallel Programming Model* which targets PRAMs
- `pardo` statement defines *parallel steps*
- sets $s_1, \ldots, s_T$ of instructions on M cells of memory
- in general, each s_i has a different size p_i
- Time T; Work $W = \sum_{i=1}^{T} p_i$

Work-Time Framework

- *Parallel Programming Model* which targets PRAMs
- `pardo` statement defines *parallel steps*
- sets $s_1, \ldots, s_T$ of instructions on M cells of memory
- in general, each s_i has a different size p_i
- Time T; Work $W = \sum_{i=1}^{T} p_i$
- Easily schedulable on PRAM: $O(W/P + T)$



Exploiting Parallelism

- PRAM to Disk Model

  Chiang, Y., Goodrich, M. T., Grove, E. F., Tamassia, R., Vengroff, D. E., Vitter, J. S.: External-Memory Graph Algorithms. SODA '95
- D-BSP to Hierarchical Memory

  Fantozzi, C., Pietracaprina, A. A., Pucci, G.: Translating Submachine Locality into Locality of Reference. *Journal of Parallel and Distributed Computing 66*

Exploiting Parallelism

- PRAM to Disk Model

  Chiang, Y., Goodrich, M. T., Grove, E. F., Tamassia, R., Vengroff, D. E., Vitter, J. S.: External-Memory Graph Algorithms. SODA '95

- D-BSP to Hierarchical Memory

  Fantozzi, C., Pietracaprina, A. A., Pucci, G.: Translating Submachine Locality into Locality of Reference. *Journal of Parallel and Distributed Computing 66*

- PRAM to Pipelined Memory

  Luccio, F., and Pagli, L.: A model of sequential computation with pipelined access to memory. *Math. Syst. Theory 26*

Our Work

Our Work

- Parallel model: Work-Time Framework
- Sequential model: SPE
- A general technique for implementing WT Algorithms on SPE
- Large classes of optimal SPE programs

Novelty

- Explicitly refer to a feature of problems: available parallelism
- Target physically implementable machines, not bound to a particular memory access function

WT Simulation - Single Step

Simulation of parallel step i (exclusive write)

WT

for $j, 1 \leq j \leq p$ pardo
    operation$_j$

SPE

segmentsize(min(k, p))
for $j, 1 \leq j \leq p$ do
    instructions$_j$

WT Simulation - Single Step

Simulation of parallel step i (exclusive write)

WT

for $j, 1 \leq j \leq p$ pardo
    operation$_j$

SPE

segmentsize(min(k, p))
for $j, 1 \leq j \leq p$ do
    instructions$_j$

- Program size: $O(1)$
- Δ Space: $O(p_i)$
- Time: $O(p + a(M^i_{SPE}))$

WT Simulation - Single Step

- Program size does not depend on input size

WT Simulation - Single Step

- Program size does not depend on input size

  $\Rightarrow$ negligible instruction load latency

WT Simulation - Single Step

- Program size does not depend on input size

  $\Rightarrow$ negligible instruction load latency
- Data memory use may increase by up to p_i

WT Simulation - Single Step

- Program size does not depend on input size

  $\Rightarrow$ negligible instruction load latency
- Data memory use may increase by up to p_i

  $\Rightarrow$ heavily depends on the WT algorithm

WT Simulation - Single Step

- Program size does not depend on input size

  $\Rightarrow$ negligible instruction load latency
- Data memory use may increase by up to p_i

  $\Rightarrow$ heavily depends on the WT algorithm

  $\Rightarrow$ degree of memory reuse

WT Simulation - Single Step

- Program size does not depend on input size

  $\Rightarrow$ negligible instruction load latency
- Data memory use may increase by up to p_i

  $\Rightarrow$ heavily depends on the WT algorithm

  $\Rightarrow$ degree of memory reuse

  $\Rightarrow$ amount of output produced

WT Simulation - Single Step

- Program size does not depend on input size

  $\Rightarrow$ negligible instruction load latency
- Data memory use may increase by up to p_i

  $\Rightarrow$ heavily depends on the WT algorithm

  $\Rightarrow$ degree of memory reuse

  $\Rightarrow$ amount of output produced
- Memory accesses fully overlap

WT Simulation - Single Step

- Program size does not depend on input size

  $\Rightarrow$ negligible instruction load latency
- Data memory use may increase by up to p_i

  $\Rightarrow$ heavily depends on the WT algorithm

  $\Rightarrow$ degree of memory reuse

  $\Rightarrow$ amount of output produced
- Memory accesses fully overlap

  $\Rightarrow$ proportional to p_i and $a(M^i_{SPE})$

WT Simulation - Concurrent Write

Different solutions, depending on the concurrent write policy:

- *priority* policy $\rightarrow$ predicated instructions
- associative op. policy $\rightarrow$ accumulation

`segmentsize(min(`k`, `p`))`
`for` $j, 1 \leq j \leq p$ `do`
`instructions`$_j$
$acc \leftarrow \max\{acc; output_j\}$

- Also an `if(test)` statement can be used...
  ...when the test is simple enough!

WT Simulation - Whole Algorithm

WT Simulation - Whole Algorithm

- Total space complexity: $M_{PH} = O(n + W)$

WT Simulation - Whole Algorithm

- Total space complexity: $M_{PH} = O(n + W)$
- Total time complexity: $O(W + T \cdot a(M_{PH}))$

WT Simulation - Whole Algorithm

- Total space complexity: $M_{PH} = O(n + W)$
- Total time complexity: $O(W + T \cdot a(M_{PH}))$

Optimal SPE programs if

- Work–optimal WT algorithms
- The *average parallelism* is larger than worst case latency

Merge

Problem: merging 2 sorted lists of n elements.

- Kruskal algorithm: $T = O(\log n)$, $W = O(n)$

Merge

Problem: merging 2 sorted lists of n elements.

- Kruskal algorithm: $T = O(\log n)$, $W = O(n)$
- $\Rightarrow T_{SPE} = O(W + T \cdot a(M)) = O(n + \log n \cdot a(n))$

Merge

Problem: merging 2 sorted lists of n elements.

- Kruskal algorithm: $T = O(\log n)$, $W = O(n)$
- $\Rightarrow T_{SPE} = O(W + T \cdot a(M)) = O(n + \log n \cdot a(n))$
- linear for $a(x) = x^{\alpha}, 0 < \alpha < 1$, $a(x) = \log x$.

On other hierarchical models:

- $O(n \log n)$ if $a(x) = x^{\alpha}, 0 < \alpha < 1$
- $O(n \log^* n)$ if $a(x) = \log x$

MergeSort

Problem: sorting a list of n elements.

Warning

Merge is linear *only if* the input is in the fastest $O(n)$ locations.

Solution

When merge istances are too small wrt latency, execute them in an interleaved fashion.

Non–local Matrix Multiplication

Problem: multiplying 2 $n \times n$ matrices.

Non–local Matrix Multiplication

Problem: multiplying 2 $n \times n$ matrices.

- The notorious WT algorithm has $T = O(\log n)$, $W = O(n^3)$

Non–local Matrix Multiplication

Problem: multiplying 2 $n \times n$ matrices.

- The notorious WT algorithm has $T = O(\log n)$, $W = O(n^3)$
- $\Rightarrow T_{SPE} = O(W + T \cdot a(M)) = O(n^3 + \log n \cdot a(n^3))$

Non–local Matrix Multiplication

Problem: multiplying 2 $n \times n$ matrices.

- The notorious WT algorithm has $T = O(\log n)$, $W = O(n^3)$
- $\Rightarrow T_{SPE} = O(W + T \cdot a(M)) = O(n^3 + \log n \cdot a(n^3))$
- Optimal even if no locality is exploited

Non–local Matrix Multiplication

Problem: multiplying 2 $n \times n$ matrices.

- The notorious WT algorithm has $T = O(\log n)$, $W = O(n^3)$
- $\Rightarrow T_{SPE} = O(W + T \cdot a(M)) = O(n^3 + \log n \cdot a(n^3))$
- Optimal even if no locality is exploited

Space complexity

$O(n^3)$ memory is required!

Conclusions and Future Work

Conclusions

- Parallelism is a viable strategy for overlapping accesses
- Large memory footprint, if too much parallelism

Conclusions and Future Work

Conclusions

- Parallelism is a viable strategy for overlapping accesses
- Large memory footprint, if too much parallelism

Future Work

- Integration with locality exploitation
- Exploitation of *coarse grained* parallelism (D-BSP)

Thank you!

Thank you for your attention!

. . . questions?