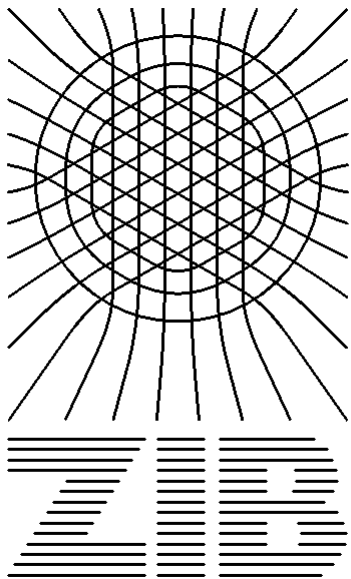


A Cache Simulator for Shared Memory Systems



Florian Schintke, Jens Simon, Alexander Reinefeld

{schintke,simon,ar}@zib.de

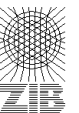
Konrad-Zuse-Zentrum für Informationstechnik Berlin

- Computer Science Research -

Takustr. 7

14195 Berlin, Germany

May 2001



Objectives

- A new cache simulator 'LDAsim':
 - based on the Latency-of-Data-Access model
 - to optimize the usage of a memory hierarchy for an application
 - to check the performance of new memory hierarchies
- With the features:
 - highly configurable
 - multiprocessor support with shared main memory (SMPs)
 - execution driven
 - written in C++ with Standard Template Library

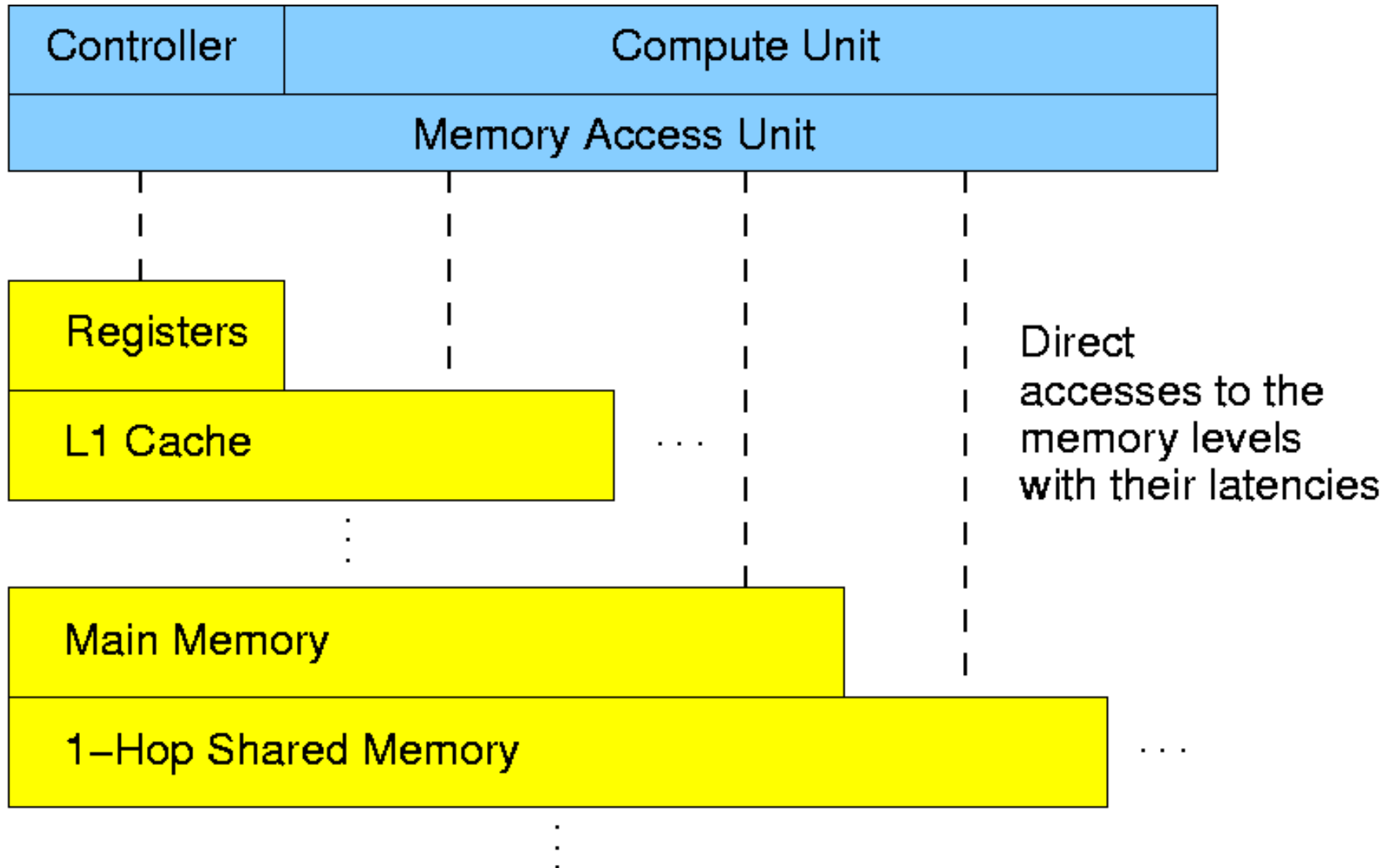
Presentation Overview

- Latency-of-Data-Access (LDA) Model
- LDA Simulator
- Example (Matrix Multiply)

Latency-of-Data-Access (LDA) Model

- Computation performance is dominated by the speed of the memory system
 - CPU speed grows faster than memory speed
 - the gap between both grows at a rate of $\sim 40\%$ per year
- LDA model takes this into account
 - estimated execution time is
 - time for fp operations
 - latencies of the accessed hierarchy level for memory operations
- To use the LDA model you need
 - number of fp operations
 - number of accesses to the different memory levels with their latencies

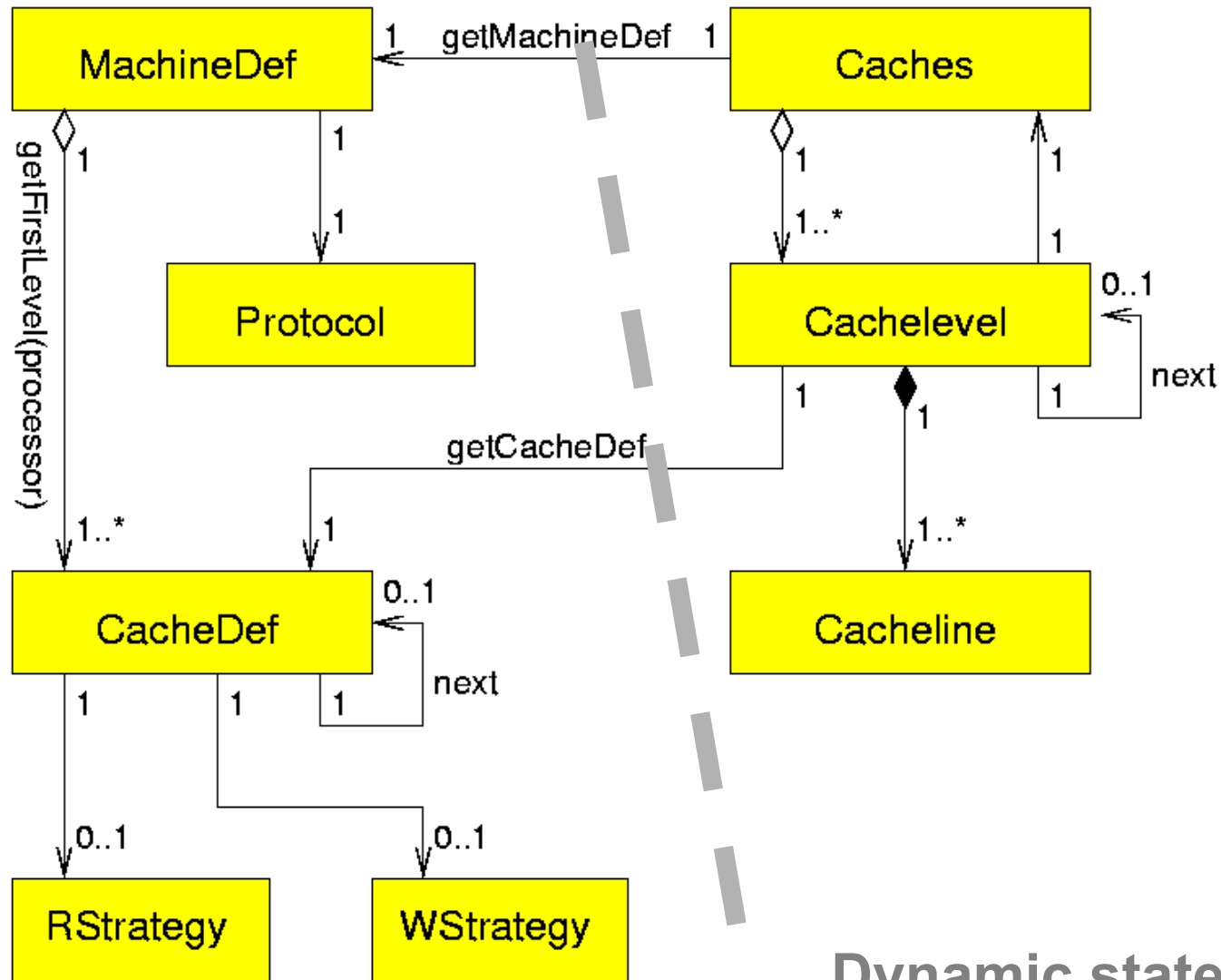
Latency-of-Data-Access (LDA) Model



LDAsim - Features

- calculates program execution time accordingly to the LDA model
- execution driven, C++
- simulated target is independent of the host system
- Instrumentation is done by hand
 - simulation overhead only for instrumented code segments
 - analysis of kernels not influenced by surroundings
- arbitrary depth of memory hierarchies supported
- configurable for each cache level:
number of blocks, blocksize, associativity, replacement strategy, write strategy, read and write latency, consistency protocol (SMPs)
- checkpointing

LDAsim – Software Architecture



Static architecture description

Dynamic state of the caches

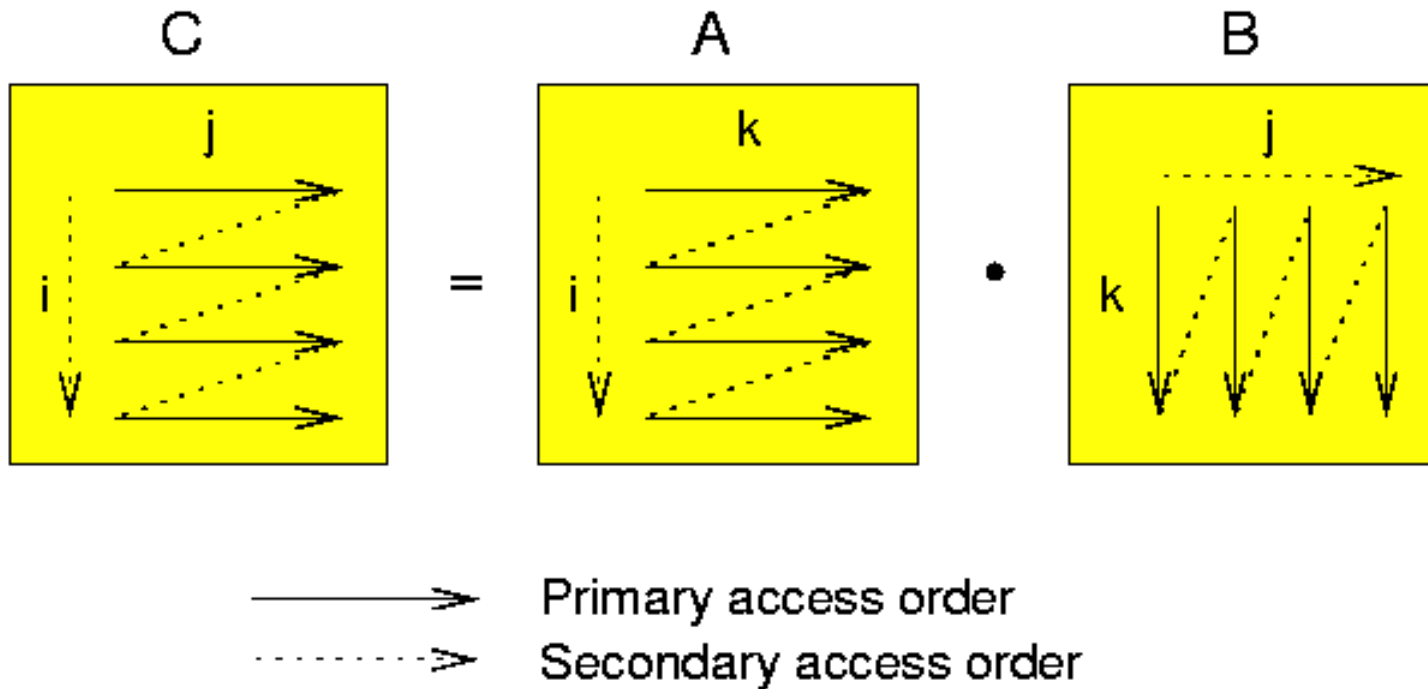
LDAsim - Usage

- call a simulator method for every fp operation
- call a simulator method for every load and store operation
- instrumented matrix multiply:

```
for (i = 0; i < matrixSize; i++)  
    for (j = 0; j < matrixSize; j++) {  
        for (k = 0; k < matrixSize; k++) {  
            caches.load(&a[i][k] - offset, 4);  
            caches.load(&b[k][j] - offset, 4);  
            caches.doFloatOps(2); /* muladd */  
        }  
        caches.store(&c[i][j] - offset, 4);  
    }
```

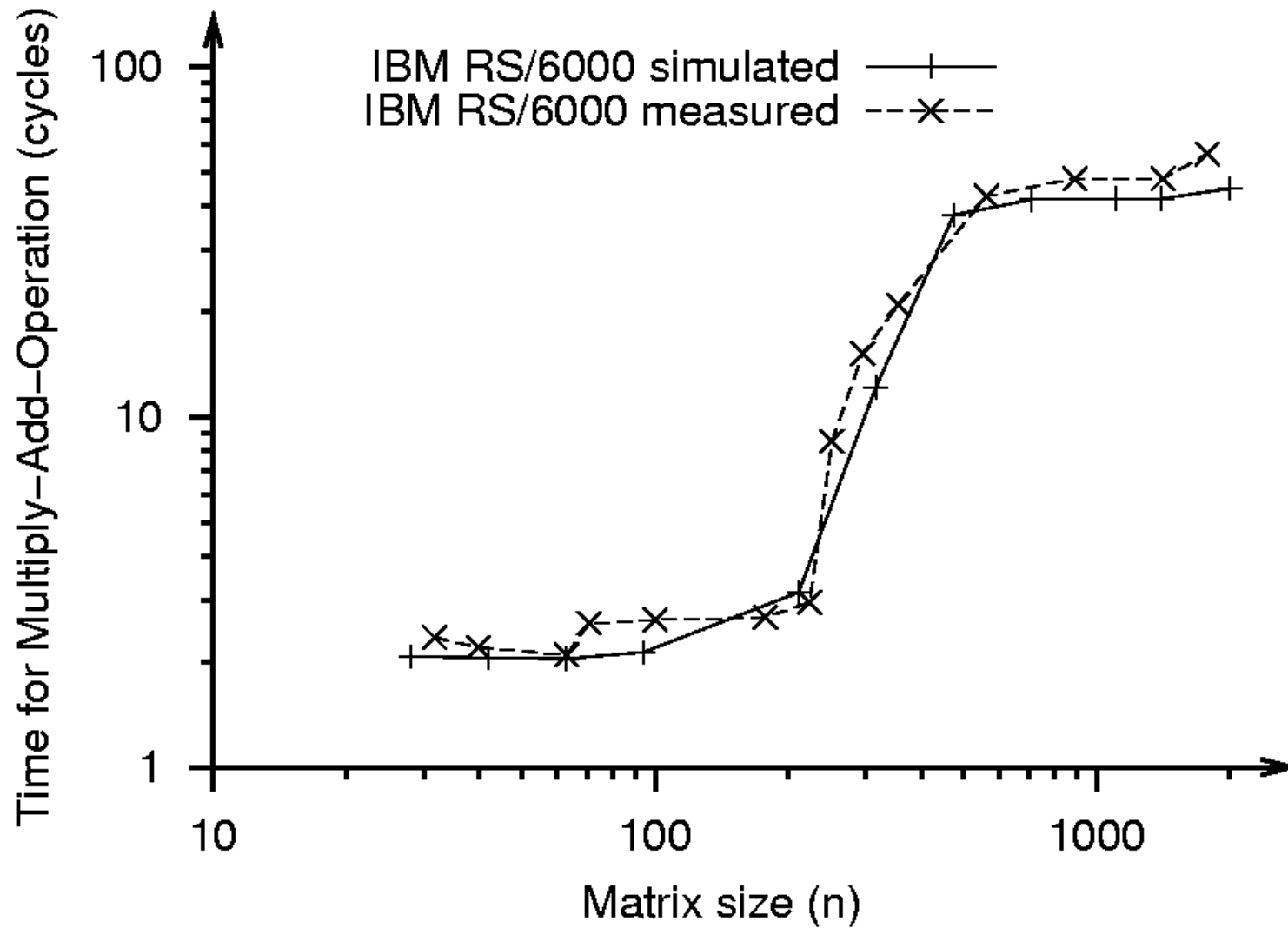
Example – Matrix Multiply – Data Layout

Access patterns:

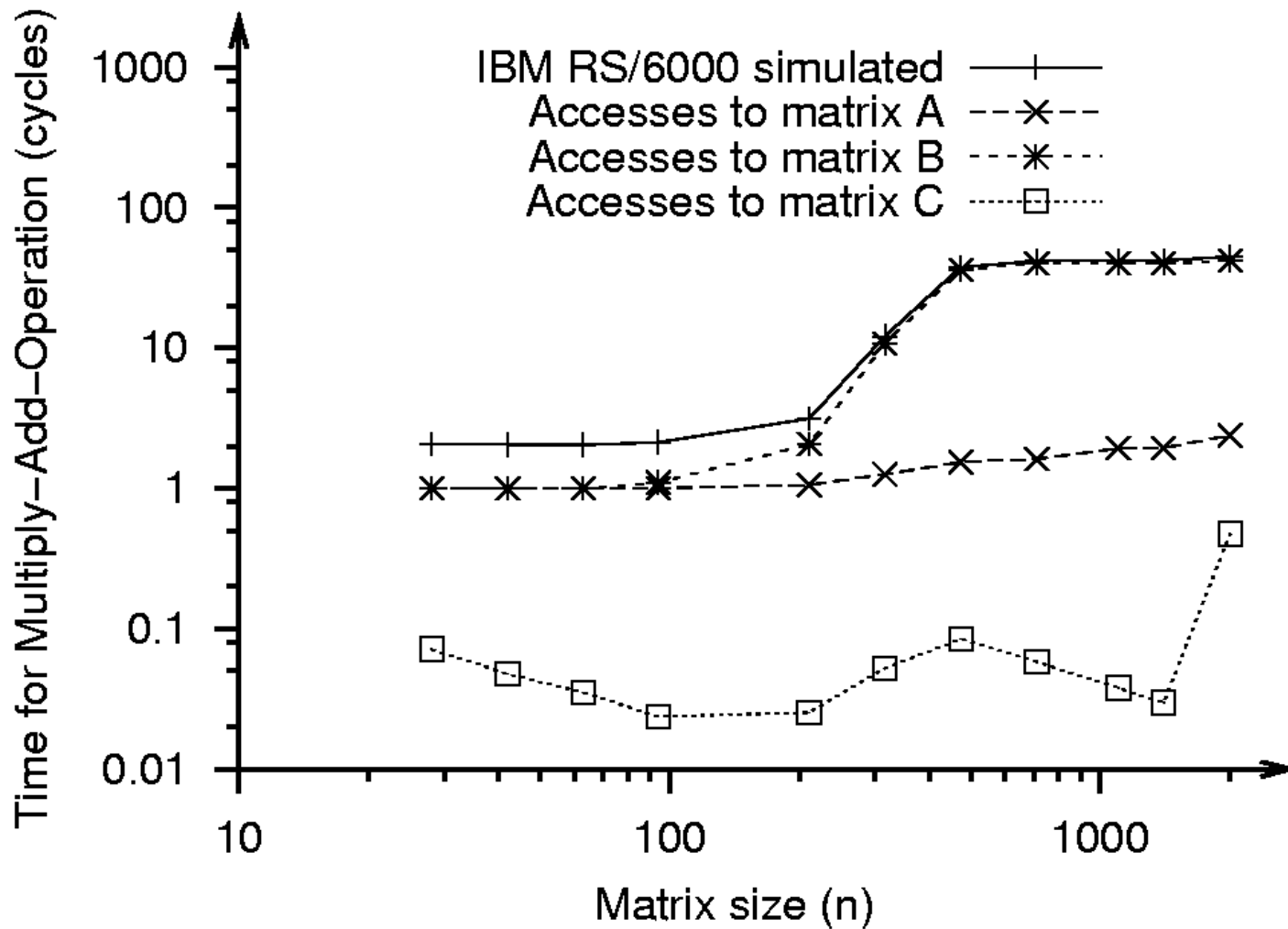


Matrix A and C are accessed linearly.

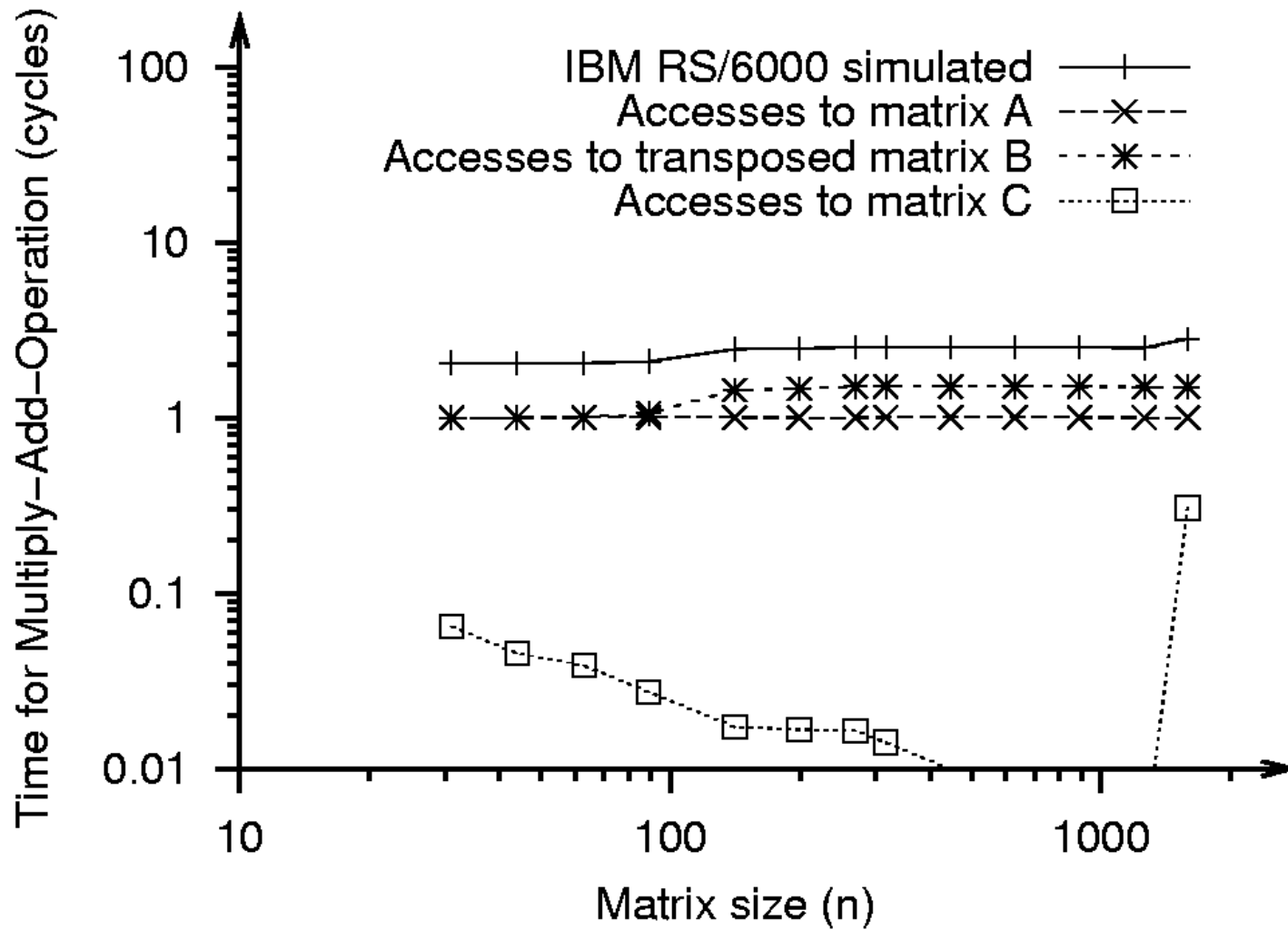
Comparison: Measured vs. Simulated



Costs for Accesses to Matrix A, B, and C



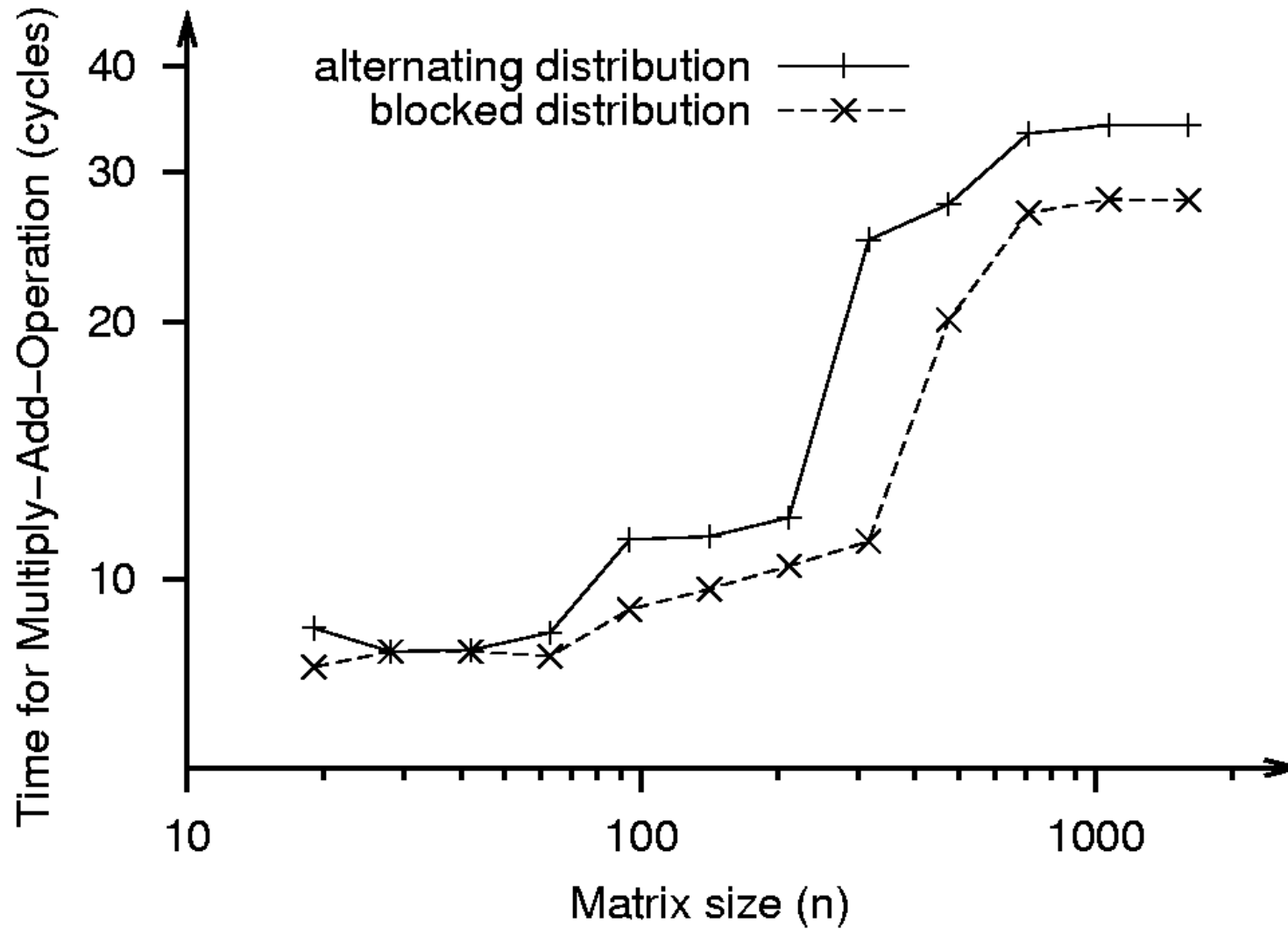
Optimized Data Layout (Transposed Matrix B)



Example – Matrix Multiply (Two Processors)

- High memory contention (alternating distribution):
 - 1st processor calculates all even cells of matrix C
 - 2nd processor calculates all odd cells of matrix C
 - ⇒ matrix C **cannot** be cached (memory contention).
- Low memory contention (blocked distribution):
 - 1st processor calculates the first half of matrix C
 - 2nd processor calculates the second half of matrix C
 - ⇒ matrix C **can** be cached.

Example – Matrix Multiply (Two Processors)



Related Work

Many other cache simulators exist

- examples: RSIM, limes, WWT II, mlcache, etc...
- different level of detail
- limited number of memory levels
- target architecture depends on host system
- ...

Conclusion

- In real applications it is hard to get information about the actual cache usage. LDAsim can help.
- LDA model is a proper cost model
- LDAsim can help in detecting bad data layouts
- LDAsim properly reflects the impact of memory contention
- Open Source:
<http://www.zib.de/schintke/ldasim/>
- Could also be used to simulate distributed shared memory systems with their higher latencies and more complicated consistency protocols.