A Data Cache with Dynamic Mapping

P. D'Alberto, A. Nicolau and A. Veidenbaum

ICS-UCI

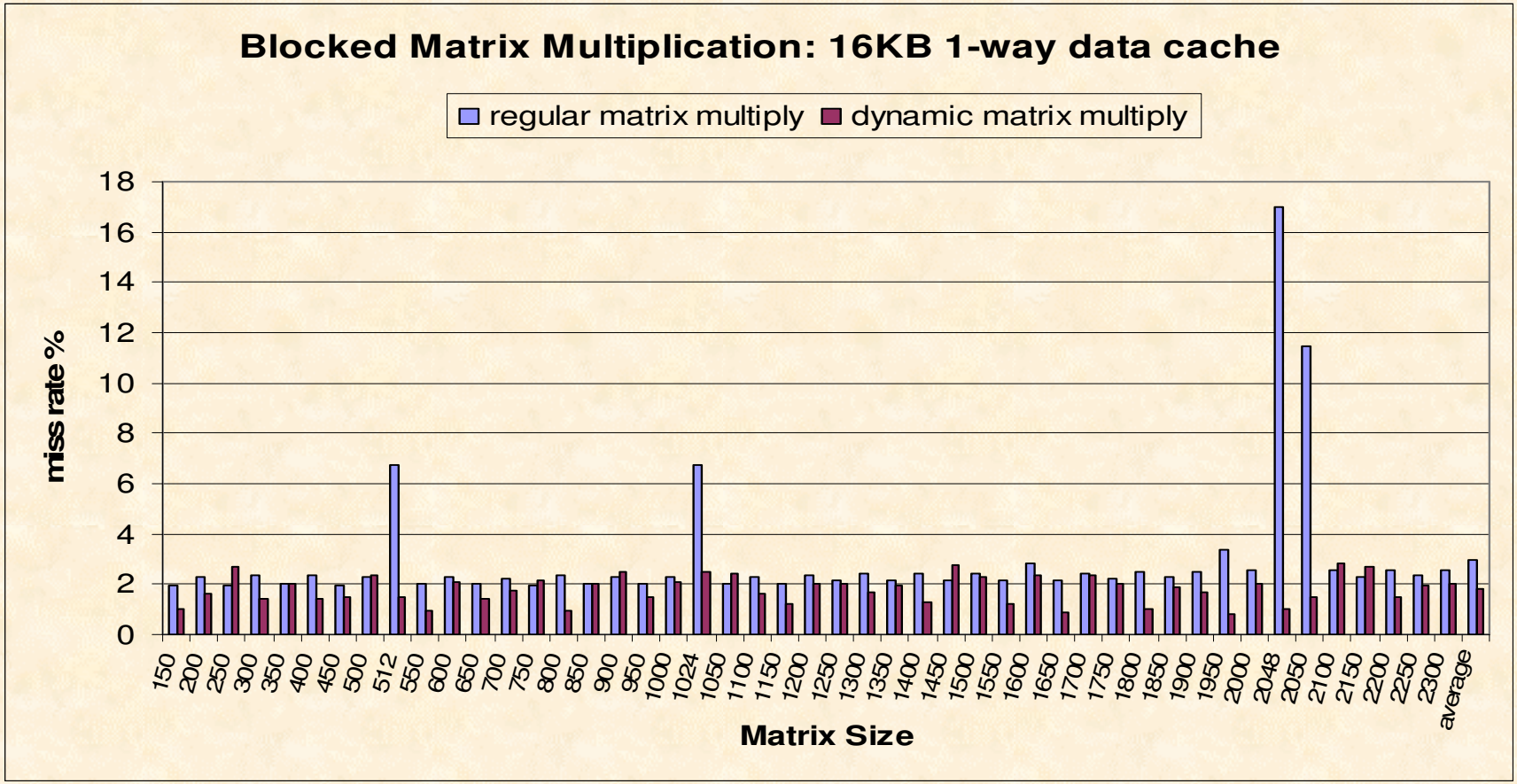
Speaker

Paolo D'Alberto

Problem Introduction

- Blocked algorithms have good performance on average
  - Because exploiting data temporal locality
- For some input sets data cache interference nullifies cache locality

Problem Introduction, cont.



Problem Introduction, cont.

- What if we remove the spikes
  - The average performance improves
  - Execution time is predictable
- We can achieve our goal by:
  - Only Software
  - Only Hardware
  - Both HW-SW

Related Work (Software)

- Data layout reorganization [Flajolet et al. 91]
- Padding [Panda et al. 99]
  - Data are reorganized before & after computation
- Data copy [Granston et al. 93]
  - Data are moved in memory during computation
- Computation reorganization [Pingali et al.02]
  - e.g., Tiling [Wolfe]

Related Work (Hardware)

- Changing cache mapping
  - Using different cache mapping functions [Gonzalez 97]
  - Increasing cache associativity [IA64]
  - Changing cache Size
- Bypassing caches
  - No interference: data are not stored in cache. [MIPS R5K]
  - HW-driven Pre-fetching

Related Work (HW-SW)

- Profiling
  - Hardware adaptation [Veidenbaum et al. 99]
  - Software adaptation [Gatlin et al. 99]
- Pre-fetching [Jouppi et al.]
  - Latency hiding mostly, used also for cache interference reduction
- Static Analysis [Ghosh et al. 99 - CME]
  - e.g., compiler driven data cache line adaptation [D'Alberto et al. 01]

Dynamic Mapping, (Software)

- We consider applications where memory references are affine functions only
- We associate a memory reference with a **twin affine function**
- We use the twin function as input address for the target data cache
- We use the original affine function to access memory

Example of twin function

- We consider the references A[i][j] and B[i][j]
  - The affine functions are:
    - A0+(iN+j)*4
    - B0+(iN+j)*4
- When there is interference (i.e.,|A0-B0| mod C < L where C and L are cache size and cache line size):
  - We use the twin functions
    - A0+(iN+j)*4
    - B0+(iN+j)*4+L

Dynamic Mapping, (Hardware)

- We introduce a new 3-address load instruction
  - A register destination
  - Two register operands: the results of twin function and of original affine function
- Note:
  - the twin function result may correspond to no real address
  - the original function is a real address
    - (and goes though TLB – ACU)

Pseudo Assembly Code

ORIGINAL CODE

Set $R0, A_0

Set $R1, B_0

…

Load $F0, $R0

Load $F1, $R1

Add $R0,$R0,4

Add $R1,$R1,4

MODIFIED CODE

Set $R0, A_0

Set $R1, B_0

Add $R2, $R1, 32

…

Load $F0, $R0

LoadTwin $F1, $R1, $R2

Add $R2, $R2, 4

Add $R0, $R0, 4

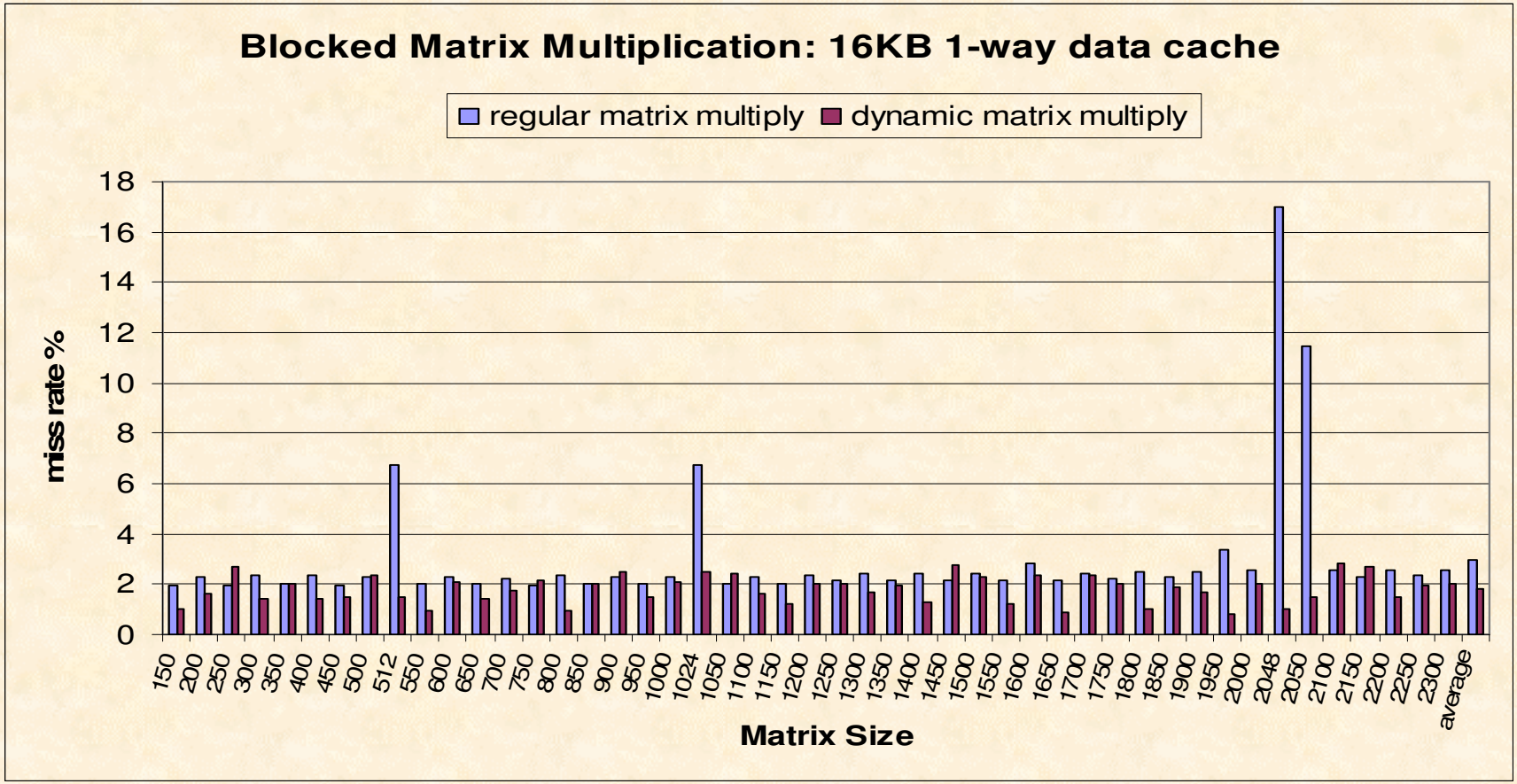
Add $R1, $R1, 4

…

Experimental Results

- We present experimental results obtained by using combination of approaches:
  - Padding
  - Data Copy
  - Without using any cycle-accurate simulator
- Matrix multiplication:
  - Simulation of cache performance a data cache size 16KB 1-way for optimally blocked algorithm

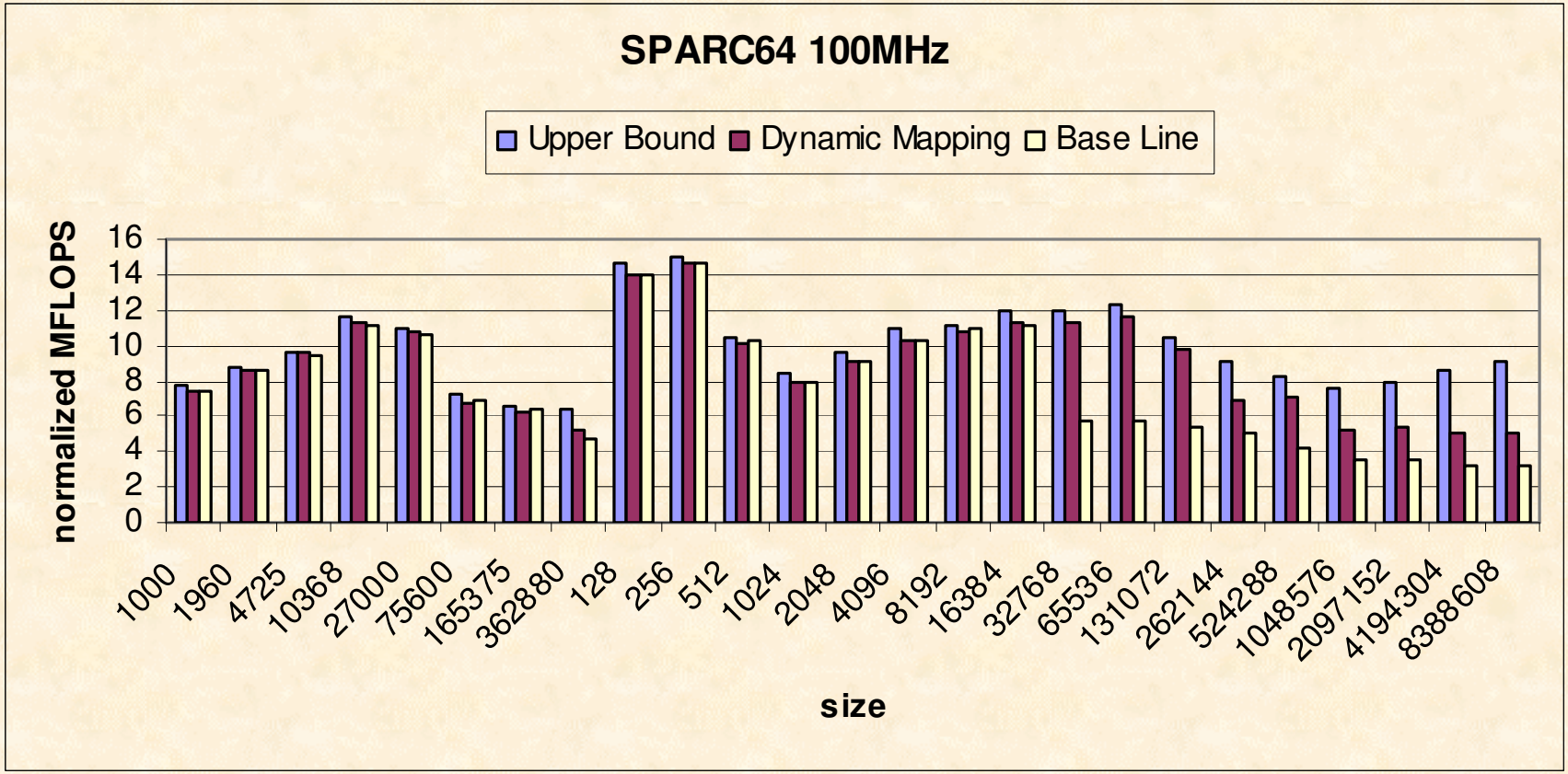
Matrix Multiply (simulation)



Experimental Results, cont.

- n-FFT, Cooley-Tookey algorithm using balanced decomposition in factors
  - The algorithm has been proposed first by Vitter et al
    - This is our baseline
  - Complexity
    - Best case O(n log log n) - Worst case $O(n^2)$
  - Normalized performance (MFLOPS)
    - nlog n/(time of one FFT)
    - We use the codelets from FFTW
    - For 128KB 4-way data cache
  - Performance comparison with FFTW is in the paper

FFT
128KB 4-way data cache



Future work

- Dynamic Mapping is not fully automated
  - The code is hand made
- A clock-accurate processor simulator is missing
  - To estimate the effects of twin function computations on performance and energy
- Application on a set of benchmarks

Conclusions

- The hardware is relatively simple
  - Because it is the compiler (or user) that activates the twin computation
    - and change the data cache mapping dynamically
- The approach aims to achieve a data cache mapping with:
  - zero interference,
  - no increase of cache hit latency
  - minimum extra hardware