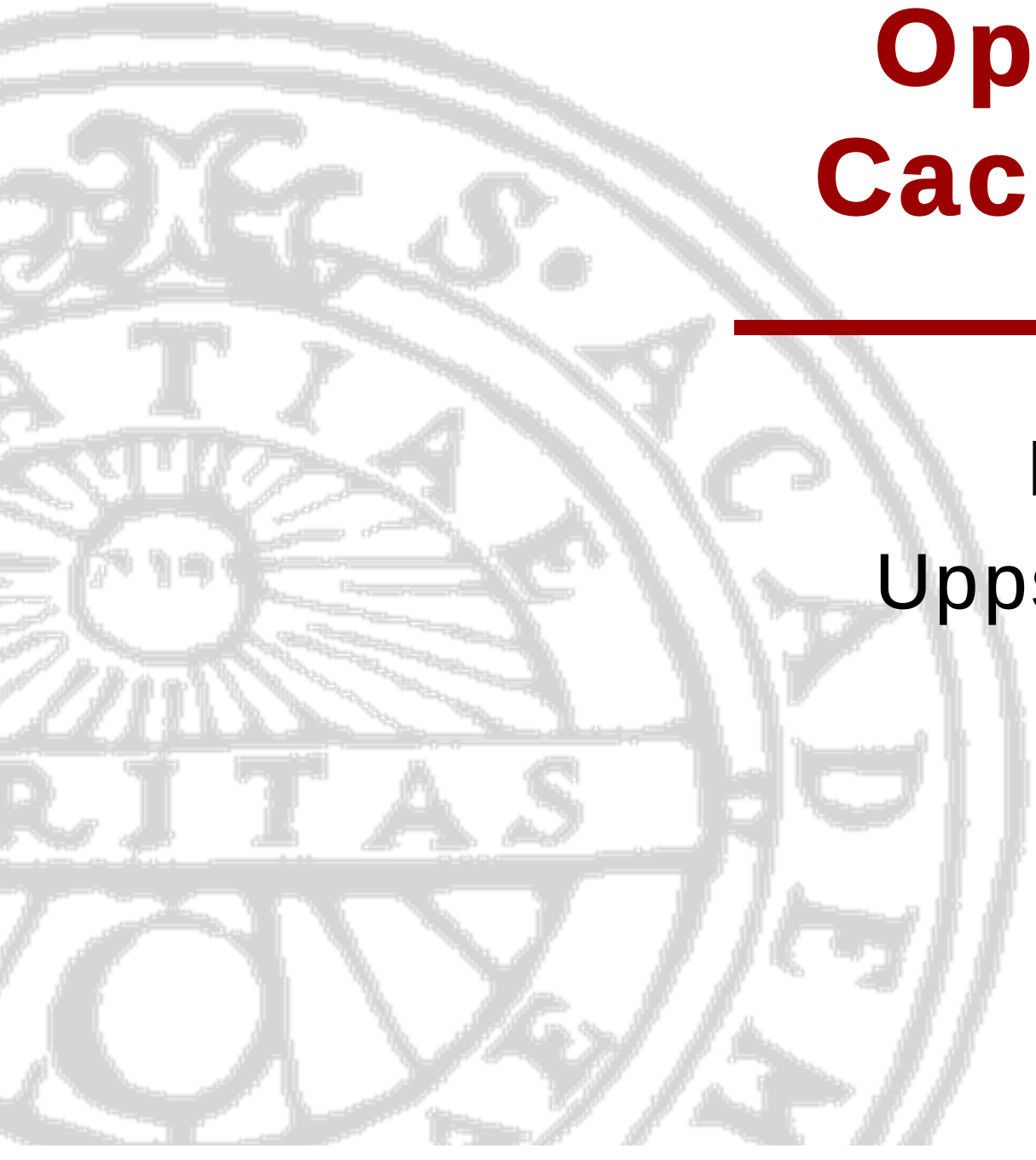


Optimizing for Cache / Memory

Erik Hagersten
Uppsala University

UPPSALA UNIVERSITET

Optimizing for the memory system: What is the potential gain?

- Latency difference between L1$ and mem: ~50x
- Bandwidth difference L1$ and mem on a MC: ~100x

- At least a factor 2-10x is within reach when optimizing for caches

- Optimizations tactics: Make applications access their data from [L1] cache instead of from memory

AVDARK 2013

UPPSALA UNIVERSITET

Four tricks for better performance

- Keep the active footprint small
- Use the entire cache line once it has been brought into the cache
- Fetch a cache line prior to its usage
- Avoid putting ”ill-suited” data structures into the cache

AVDARK 2013

UPPSALA UNIVERSITET

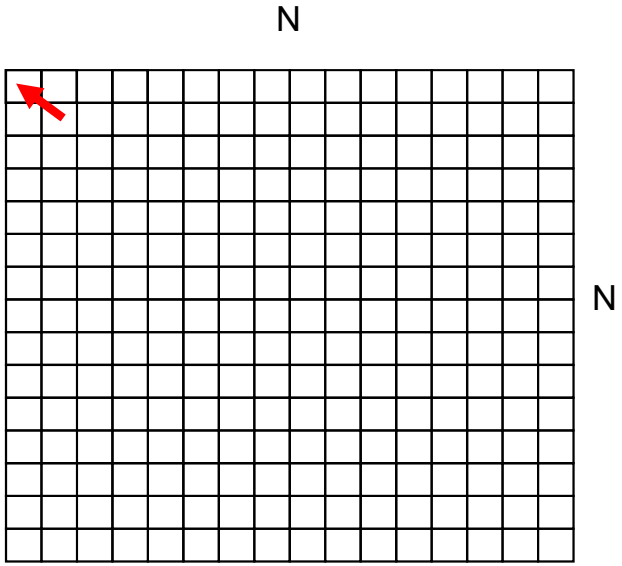
Final cache lingo slide

- **Miss ratio:** What is the likelihood that a memory access will miss in a cache [%]?
- **Miss rate:** D:o per time unit, e.g. per-second, per-1000-instructions
- **Fetch ratio/rate*):** What is the likelihood that a memory access will cause a fetch to the cache [including HW/SW prefetching]
- **Fetch utilization*):** What fraction of a cacheline was used before it got evicted
- **Writeback utilization*):** What fraction of a cacheline written back to memory contains dirty data

AVDARK 2013

What can go Wrong? A Simple Example...

Perform a diagonal copy 10 times

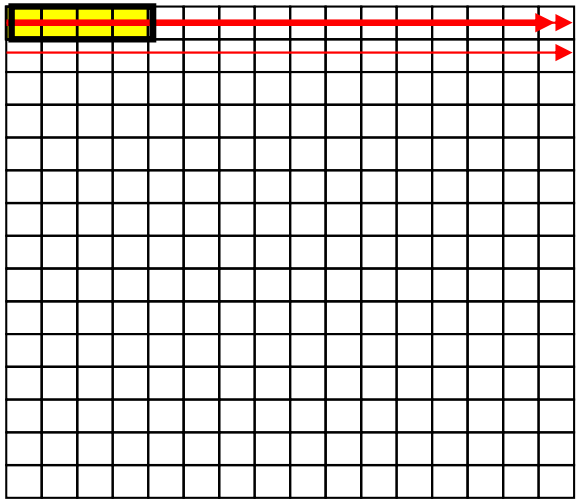


AVDARK 2013

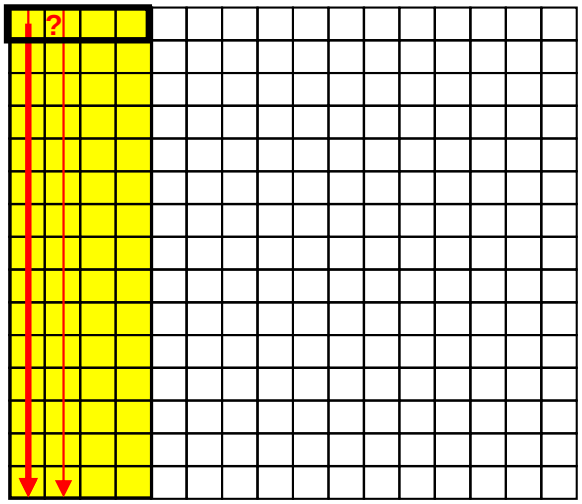
Note that these examples also assume an outer loop that will repeat the code several times…

Example: Loop order

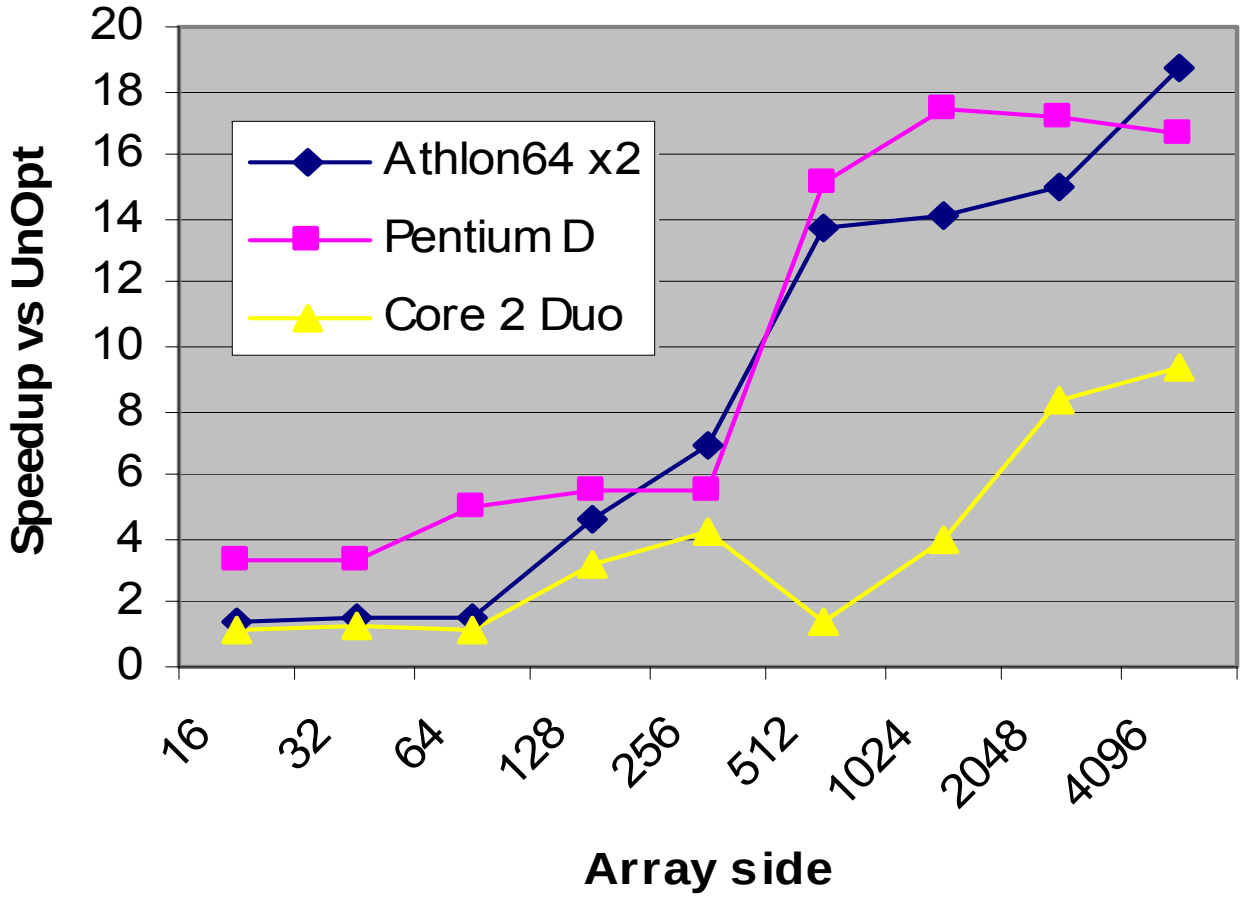
```
//Optimized Example A
for (i=1; i<N; i++) {
  for (j=1; j<N; j++) {
    A[i][j]= A[i+1][j+1];
  }
}
```



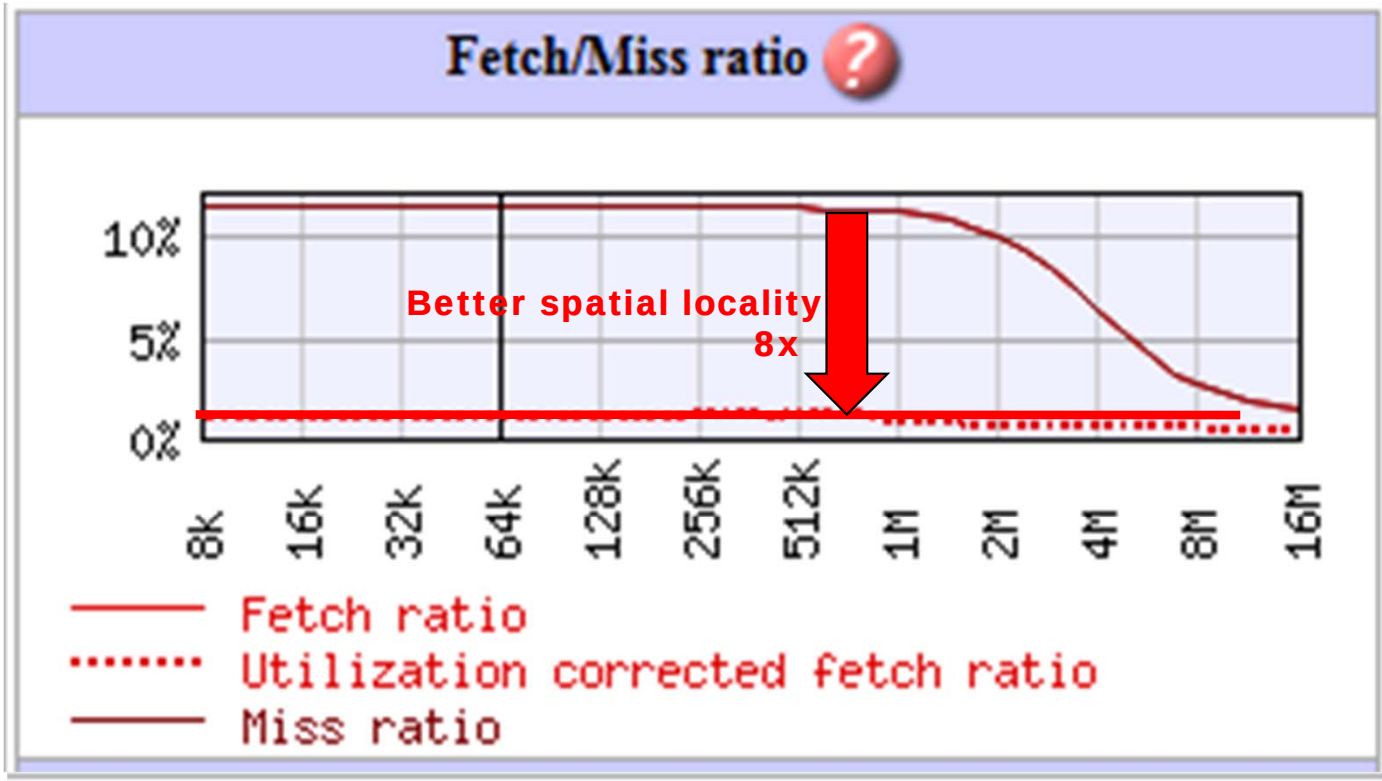
```
//Unoptimized Example A
for (j=1; j<N; j++) {
  for (i=1; i<N; i++) {
    A[i][j] = A[i+1][j+1];
  }
}
```



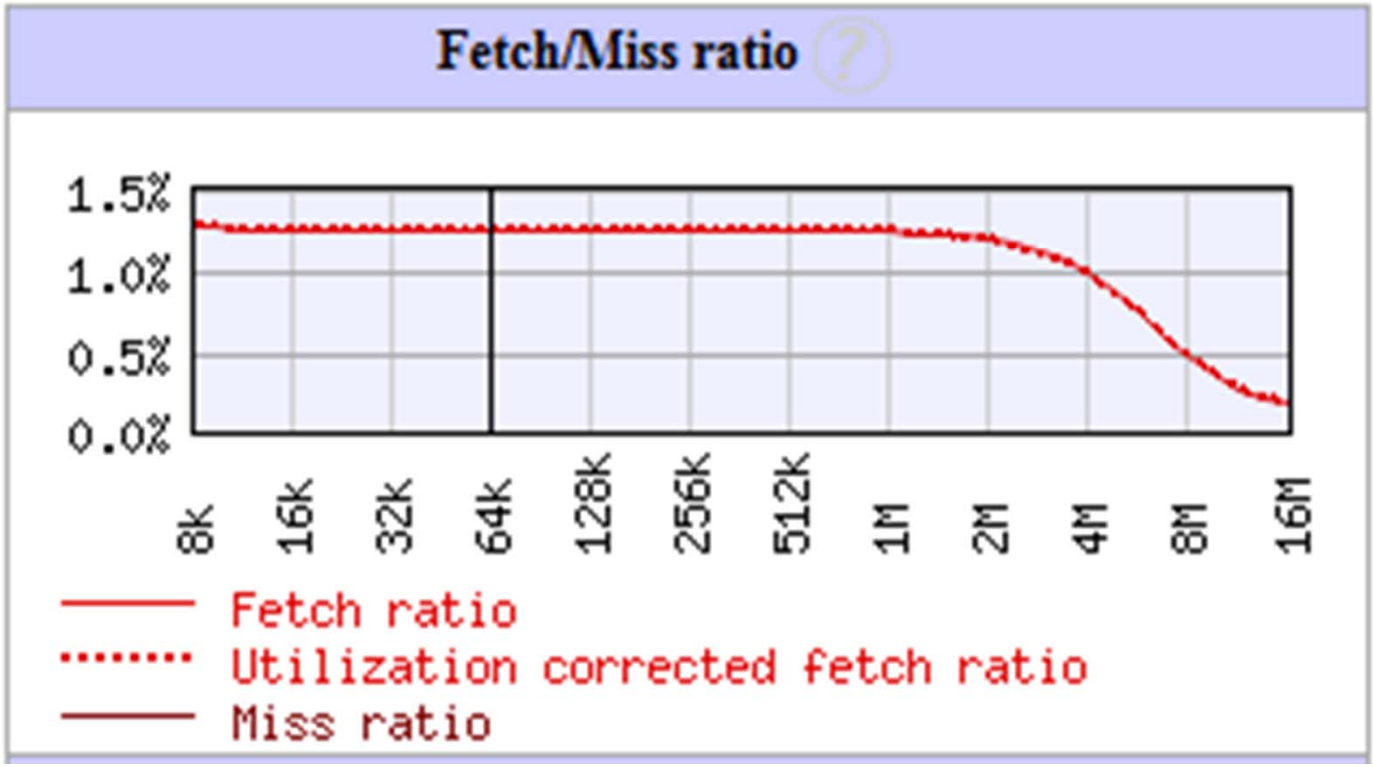
Performance Difference: Loop order



UNOPT:

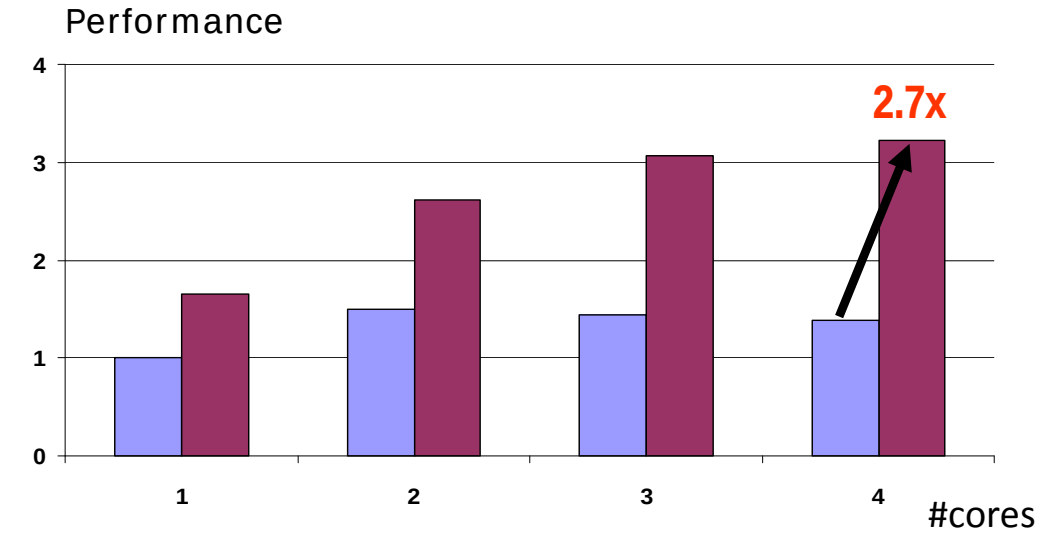

Fetch/Miss ratio
10%
5%
0%
Better spatial locality
8x
8k
16k
32k
64k
128k
256k
512k
1M
2M
4M
8M
16M
Fetch ratio
Utilization corrected fetch ratio
Miss ratio

OPT:


Fetch/Miss ratio
1.5%
1.0%
0.5%
0.0%
8k
16k
32k
64k
128k
256k
512k
1M
2M
4M
8M
16M
Fetch ratio
Utilization corrected fetch ratio
Miss ratio

UPPSALA UNIVERSITET

Example: LBM

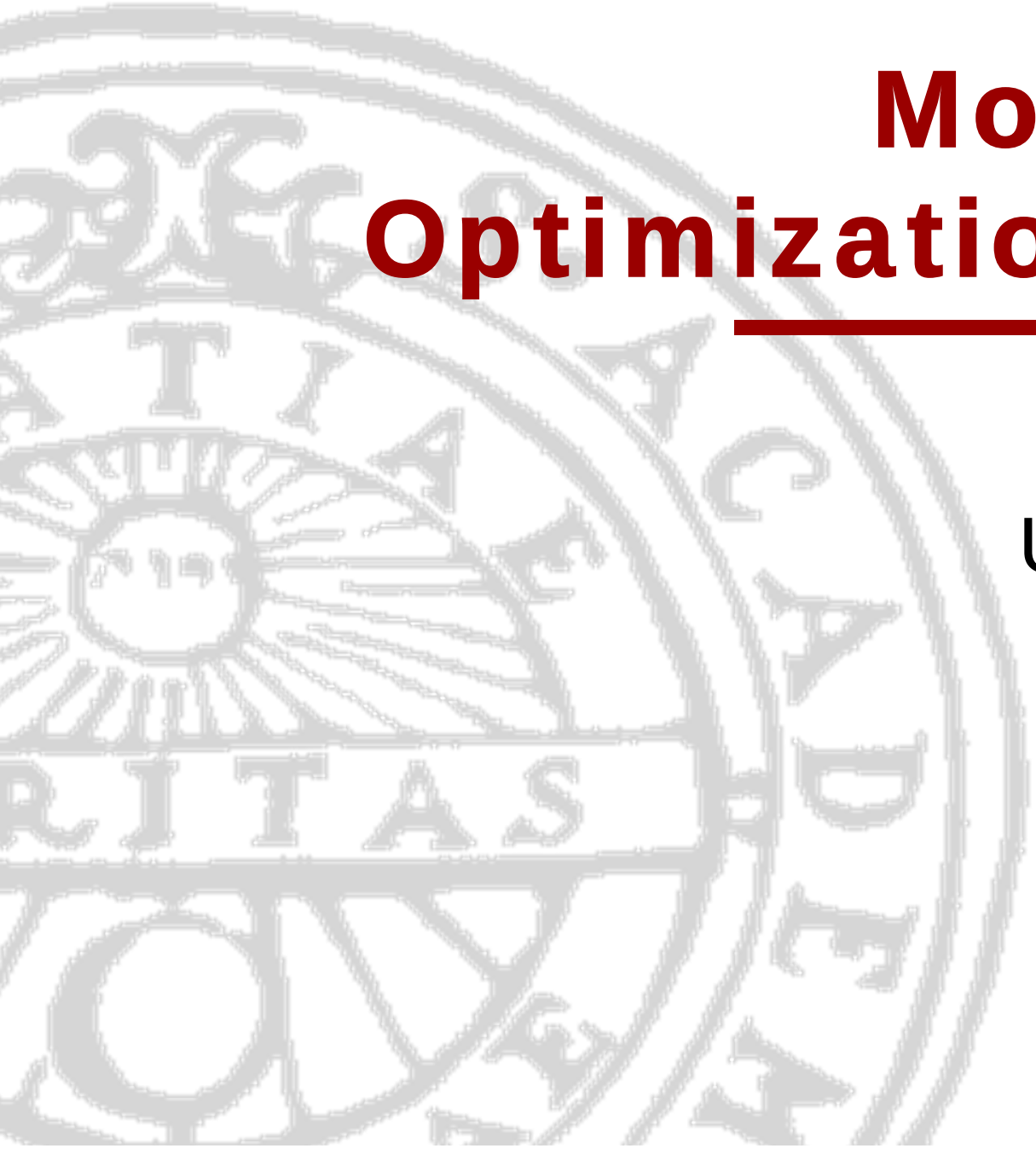


App: LBM

Optimization can be rewarding, but costly…

- Require expert knowledge about MC and architecture
- Weeks of wading through performance data

➔This fix required one line of code to change



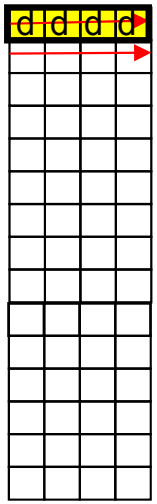
More Software Optimization Examples

Erik Hagersten
Uppsala University

Example: Sparse data utilization

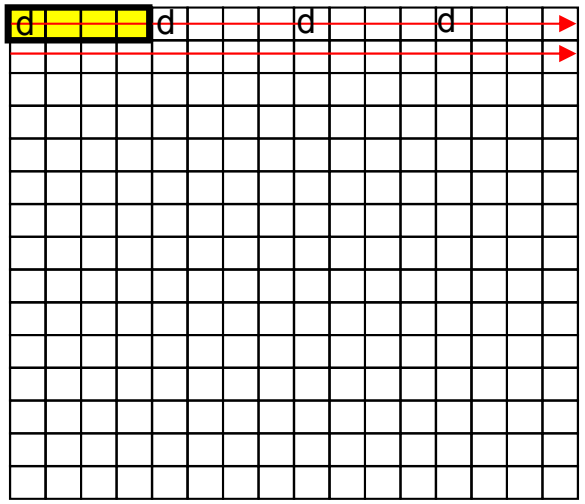
```
//Optimized Example A
for (i=1; i<N; i++) {
  for (j=1; j<N; j++) {
    A_d[i][j]= A_d[i-1][j-1];
  }
}
```

Better spatial locality



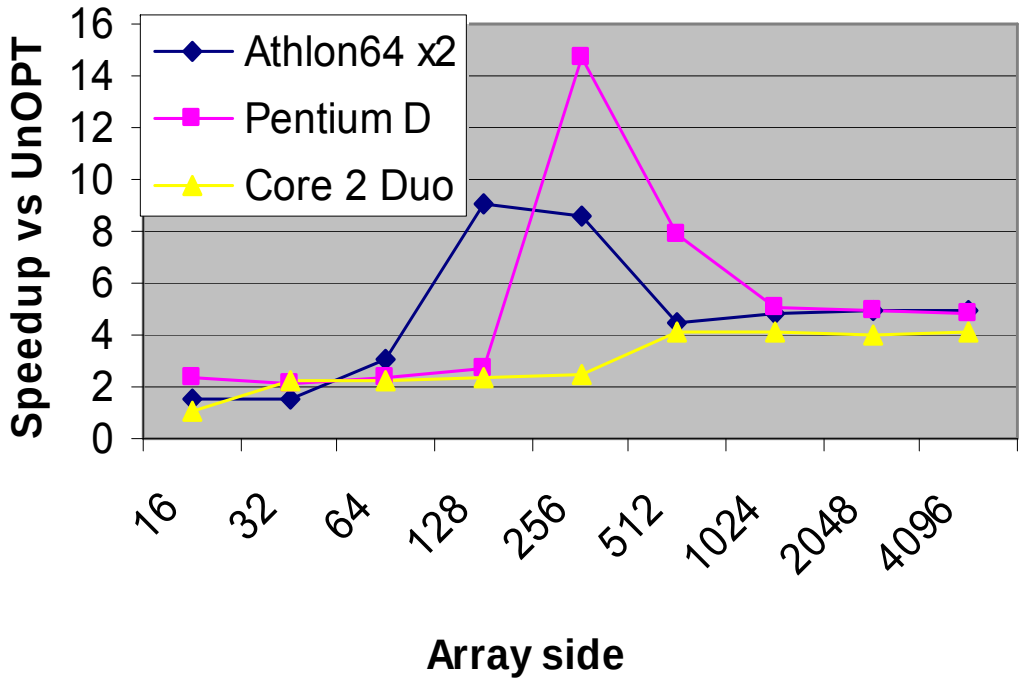
Smaller array

```
//Unoptimized Example A
for (i=1; i<N; i++) {
  for (j=1; j<N; j++) {
    A[i][j].d = A[i-1][j-1].d;
  }
}
```

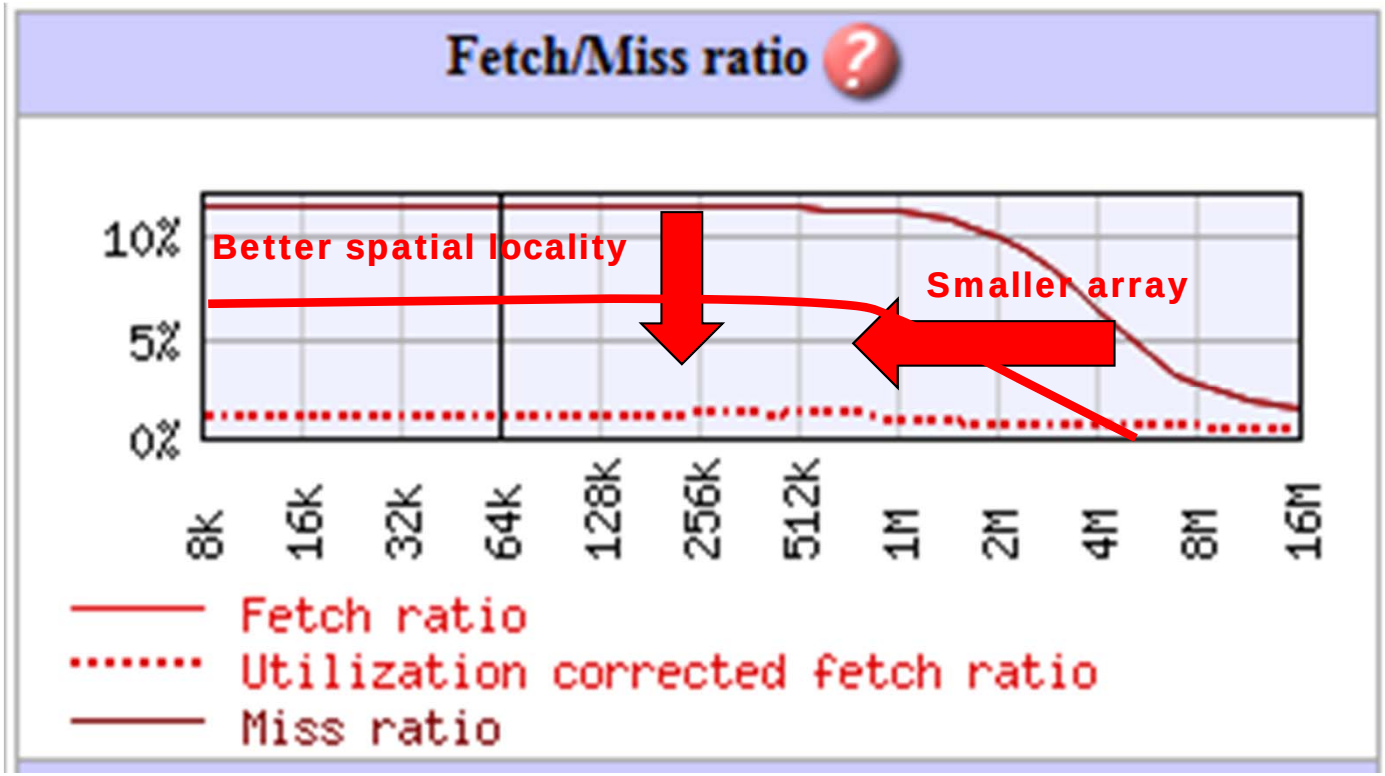



```
struct vec_type
{
        int a;
        int b;
        int c;
        int d;
};
```

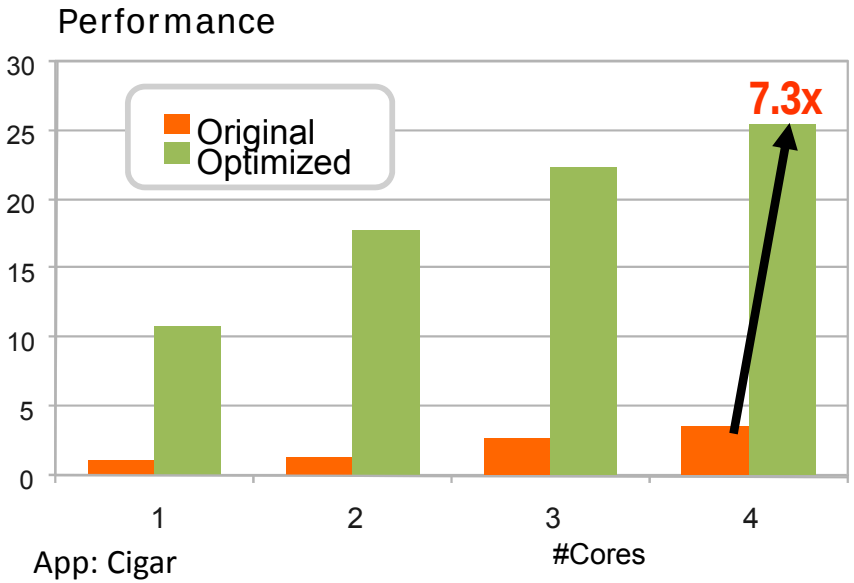
Performance Difference: Sparse Data



Example: Sparse data utilization



Example: Cigar



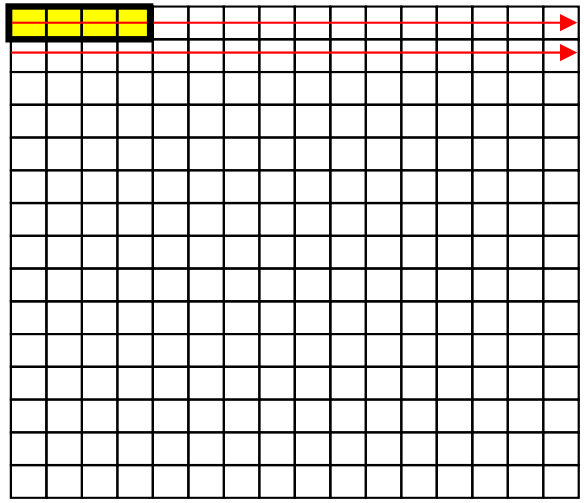
Looks like a perfect scalable application!

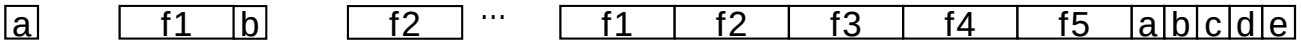
Are we done?

➔ Duplicate one data structure

Example: Sparse data allocation

```
sparse_rec sparse [HUGE];

for (int j = 0; j < HUGE; j++)
{
    sparse[j].a = 'a'; sparse[j].b = 'b'; sparse[j].c = 'c'; sparse[j].d = 'd'; sparse[j].e = 'e';
    sparse[j].f1 = 1.0; sparse[j].f2 = 1.0; sparse[j].f3 = 1.0; sparse[j].f4 = 1.0; sparse[j].f5 = 1.0;
}
```

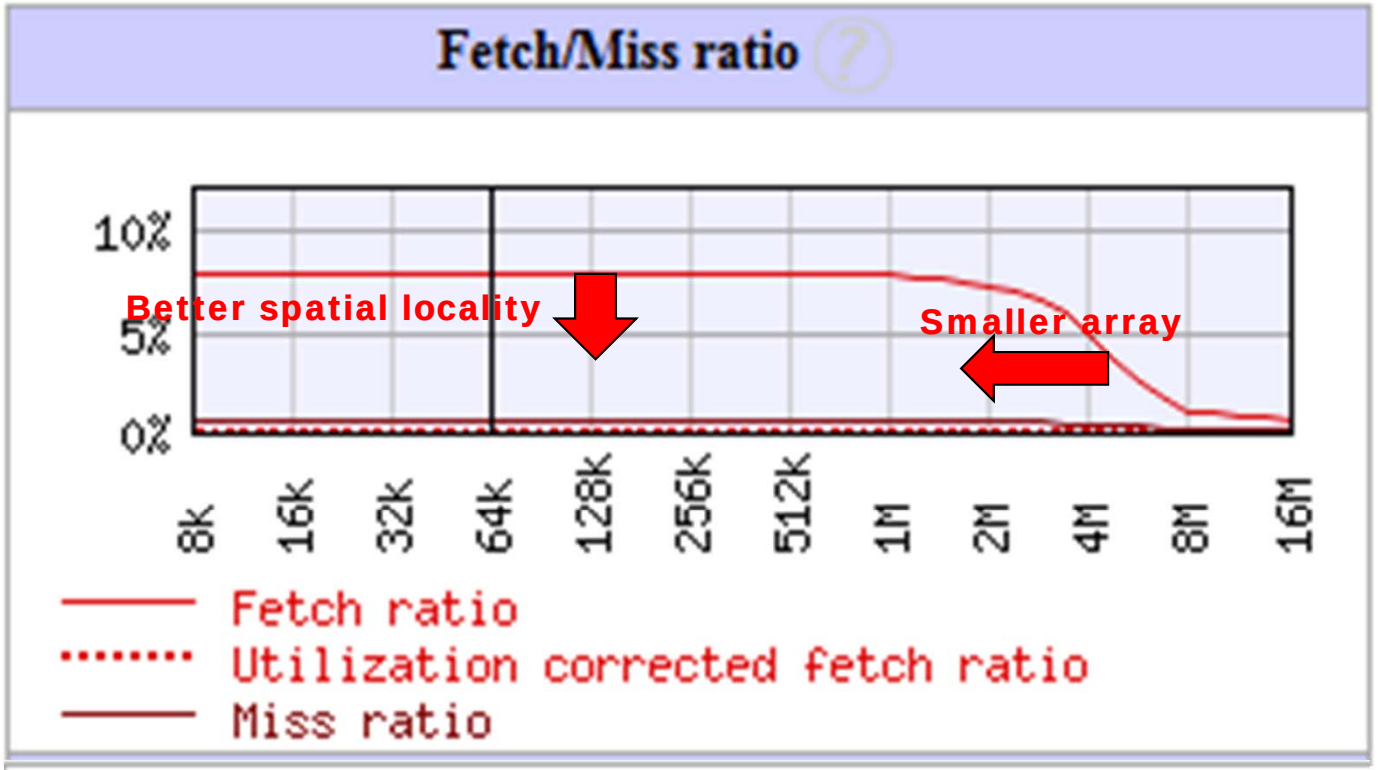




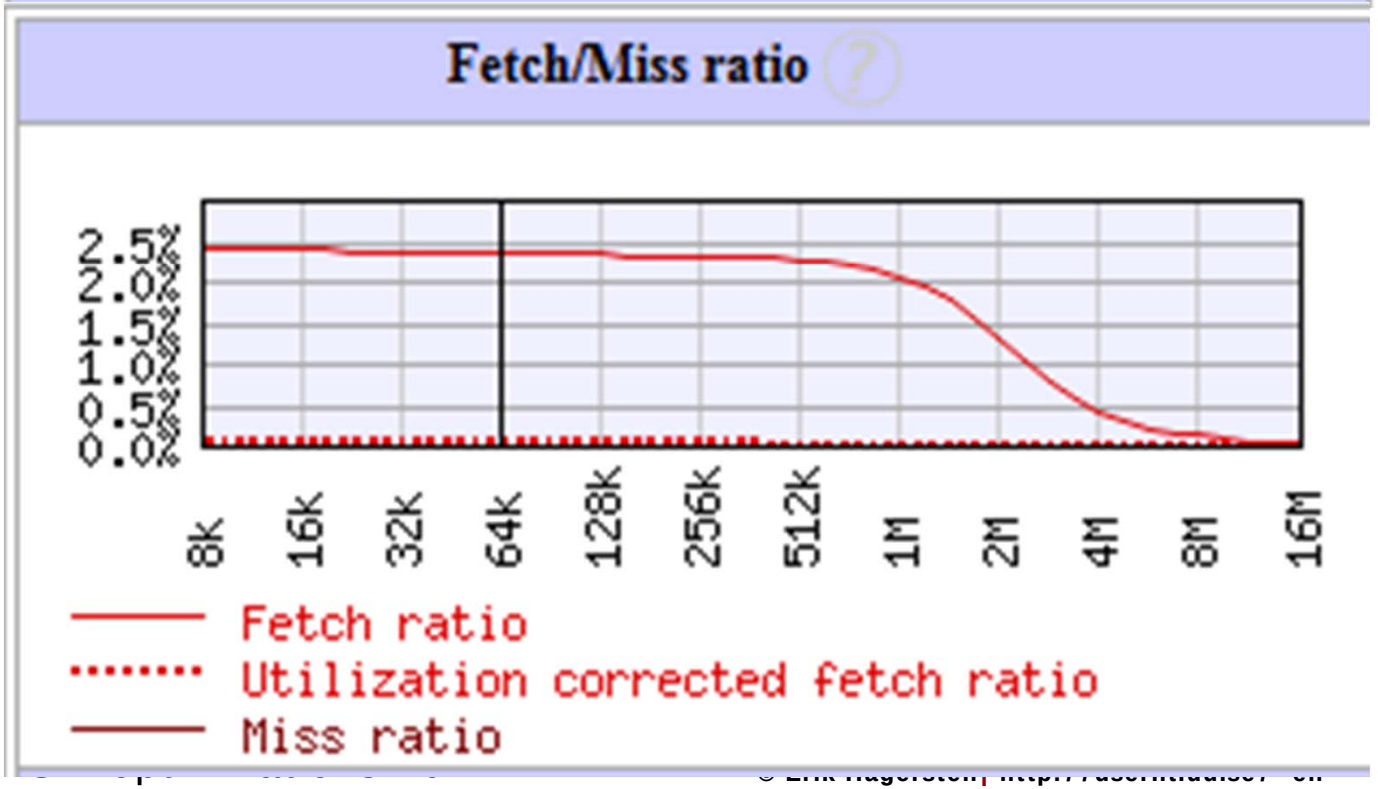

```
struct sparse_rec
{
            // size 80B
            char a;
            double f1;
            char b;
            double f2;
            char c;
            double f3;
            char d;
            double f4;
            char e;
            double f5;
};
```

```
struct dense_rec
{
            //size 48B
            double f1;
            double f2;
            double f3;
            double f4;
            double f5;
            char a;
            char b;
            char c;
            char d;
            char e;
};
```

UNOPT:


Fetch/Miss ratio
10%
5%
0%
Better spatial locality
Smaller array
8k
16k
32k
64k
128k
256k
512k
1M
2M
4M
8M
16M
Fetch ratio
Utilization corrected fetch ratio
Miss ratio

OPT:


Fetch/Miss ratio
2.5%
2.0%
1.5%
1.0%
0.5%
0.0%
8k
16k
32k
64k
128k
256k
512k
1M
2M
4M
8M
16M
Fetch ratio
Utilization corrected fetch ratio
Miss ratio

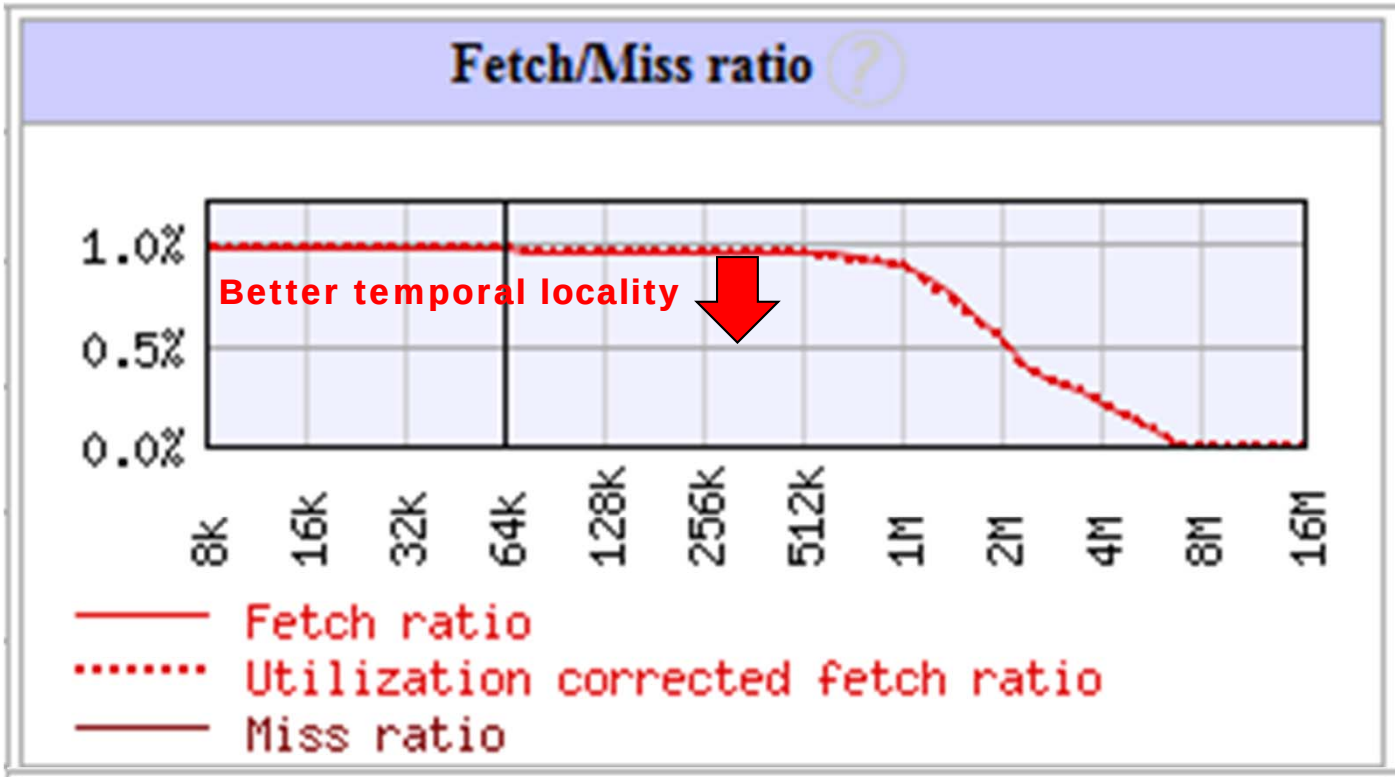
UPPSALA UNIVERSITET

Loop Fusion

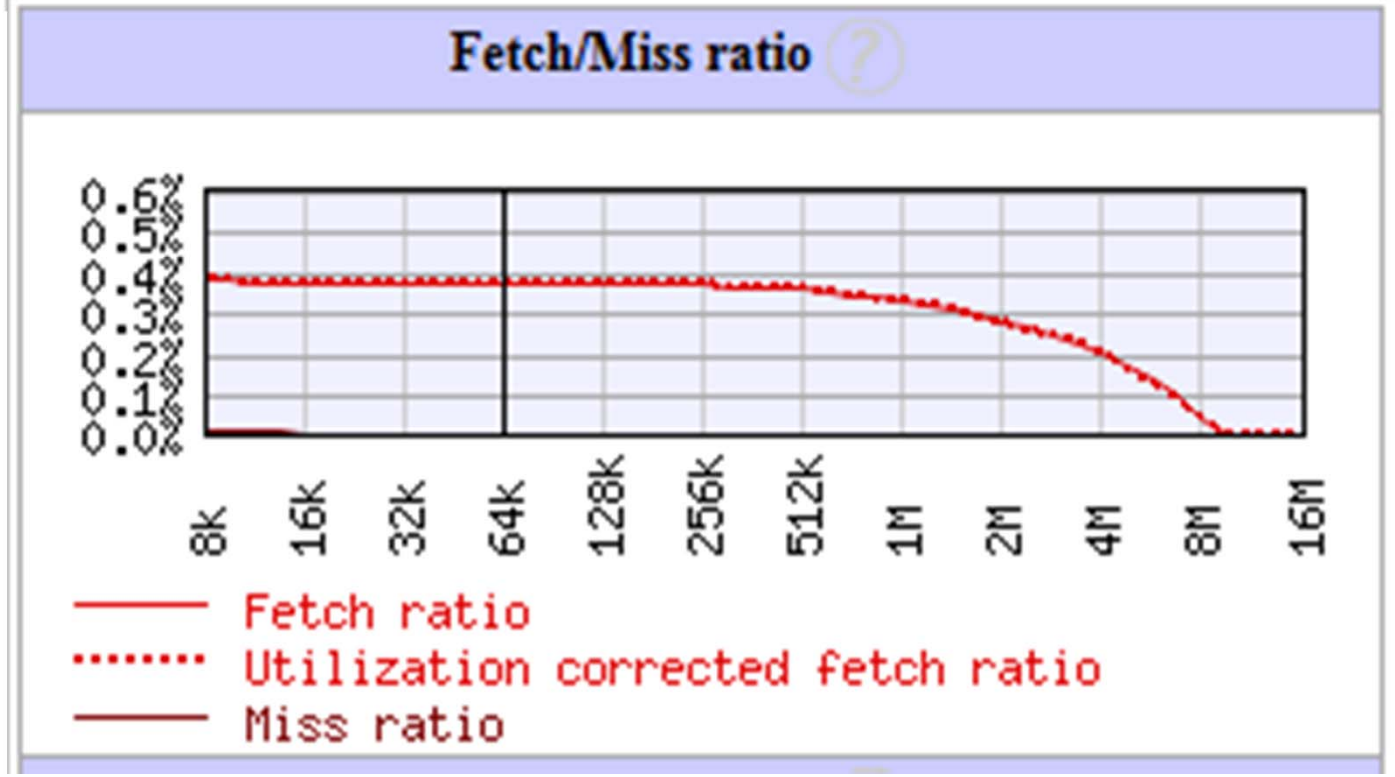
```
/* Unoptimized */
for (i = 0; i < N; i = i + 1)
   for (j = 0; j < N; j = j + 1)
        a[i][j] = 2 * b[i][j];
for (i = 0; i < N; i = i + 1)
   for (j = 0; j < N; j = j + 1)
        c[i][j] = K * b[i][j] + d[i][j]/2

/* Optimized */
for (i = 0; i < N; i = i + 1)
   for (j = 0; j < N; j = j + 1){
        a[i][j] = 2 * b[i][j];
        c[i][j] = K * b[i][j] + d[i][j]/2;
   }
```

AVDARK 2013

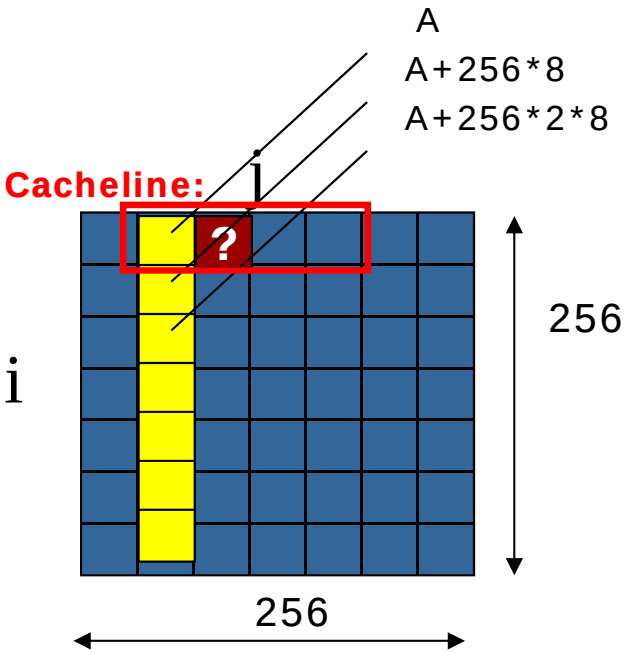
UNOPT:


Fetch/Miss ratio
1.0%
0.5%
0.0%
Better temporal locality
8k
16k
32k
64k
128k
256k
512k
1M
2M
4M
8M
16M
Fetch ratio
Utilization corrected fetch ratio
Miss ratio

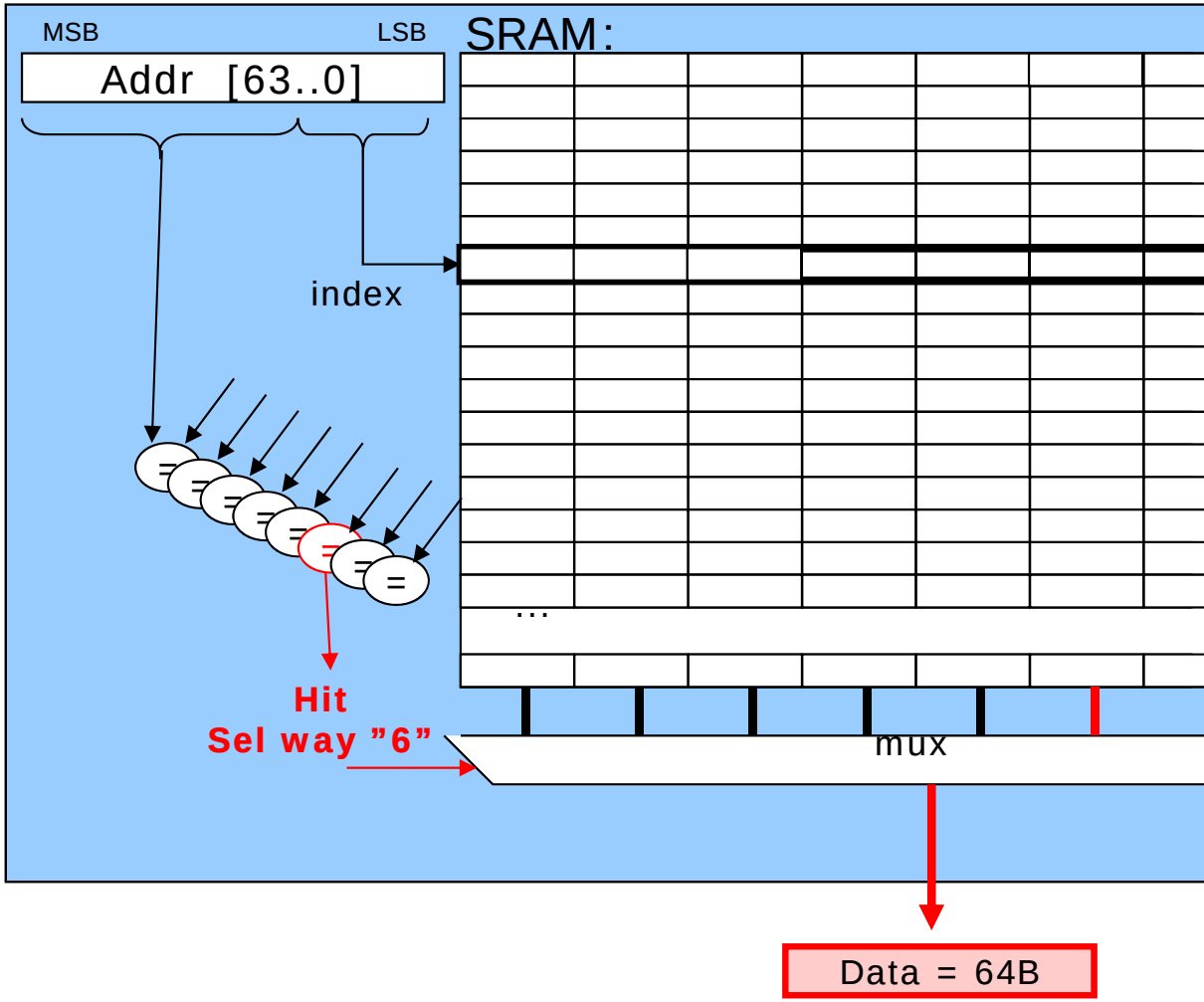
OPT:


Fetch/Miss ratio
0.6%
0.5%
0.4%
0.3%
0.2%
0.1%
0.0%
8k
16k
32k
64k
128k
256k
512k
1M
2M
4M
8M
16M
Fetch ratio
Utilization corrected fetch ratio
Miss ratio

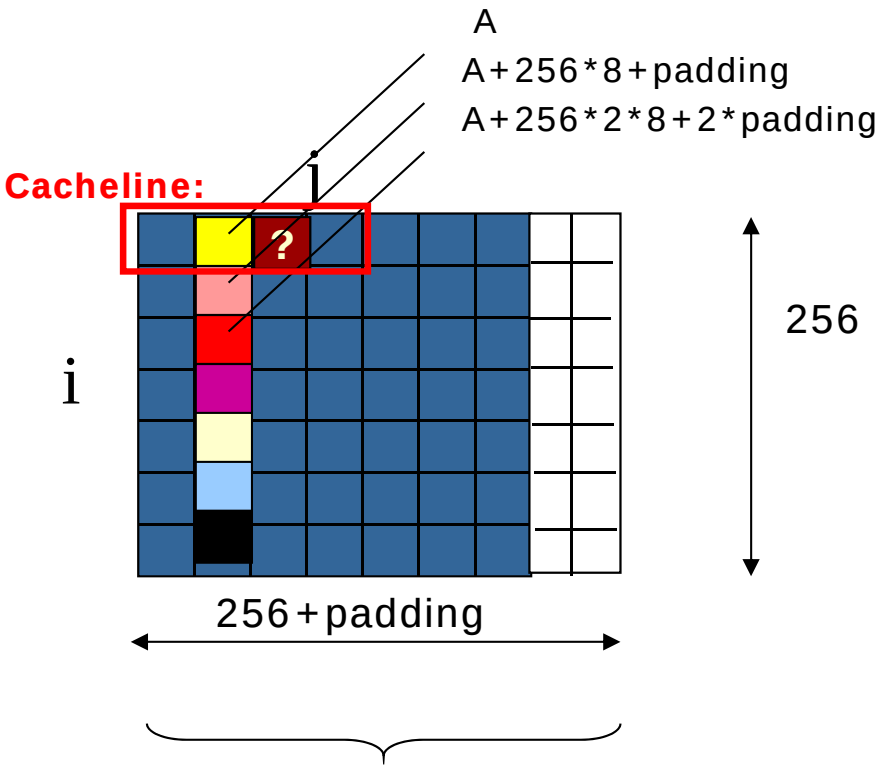
Padding of data structures



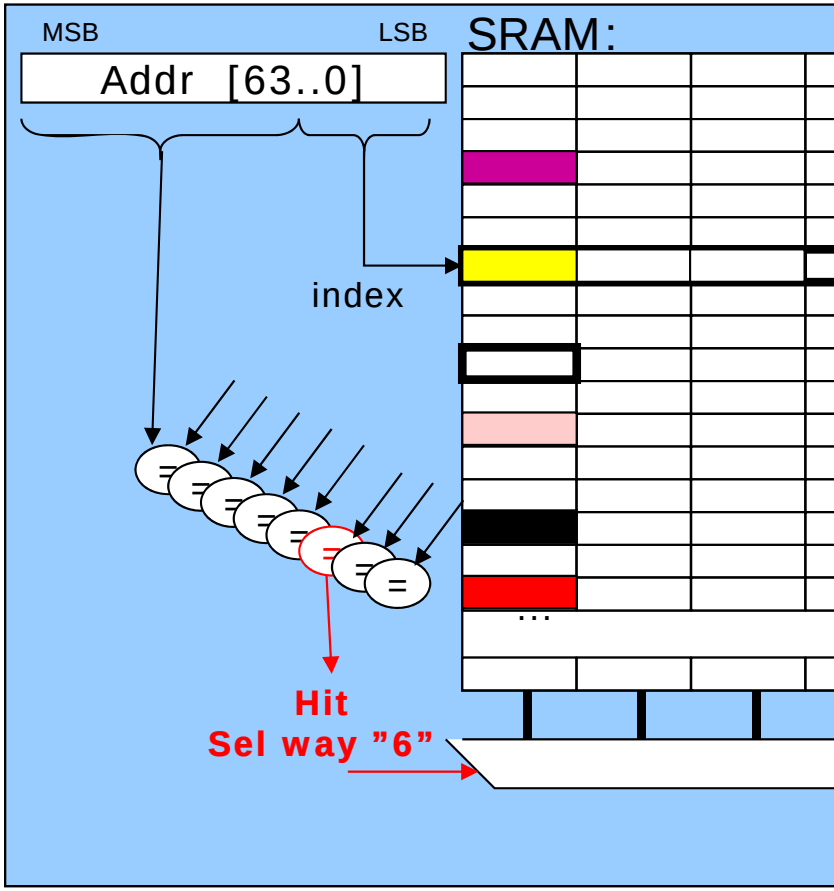
Generic Cache:



Padding of data structures

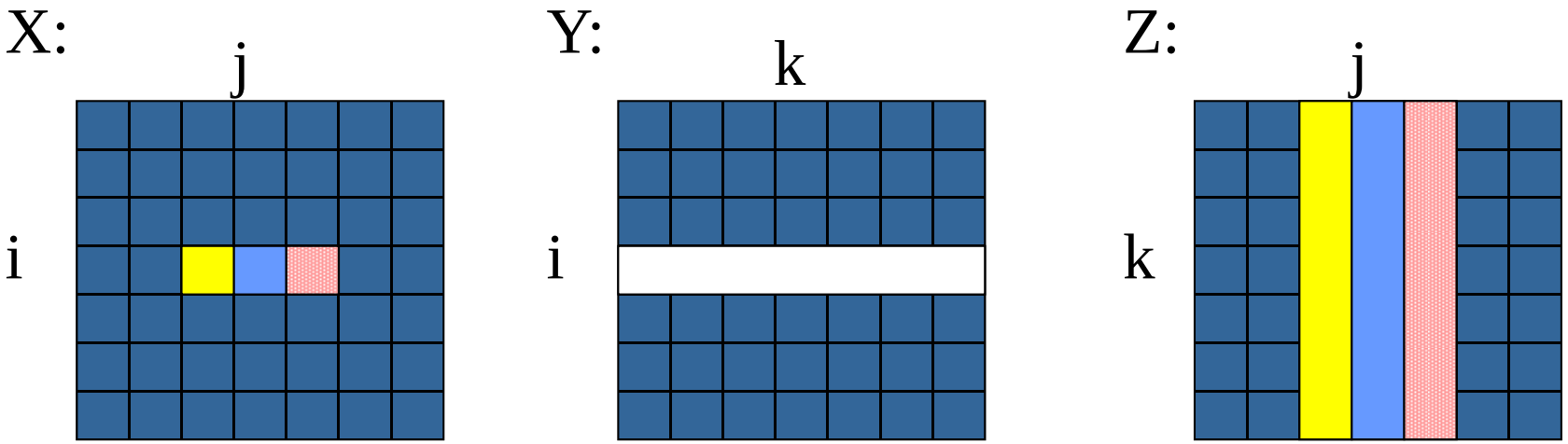


Generic Cache:



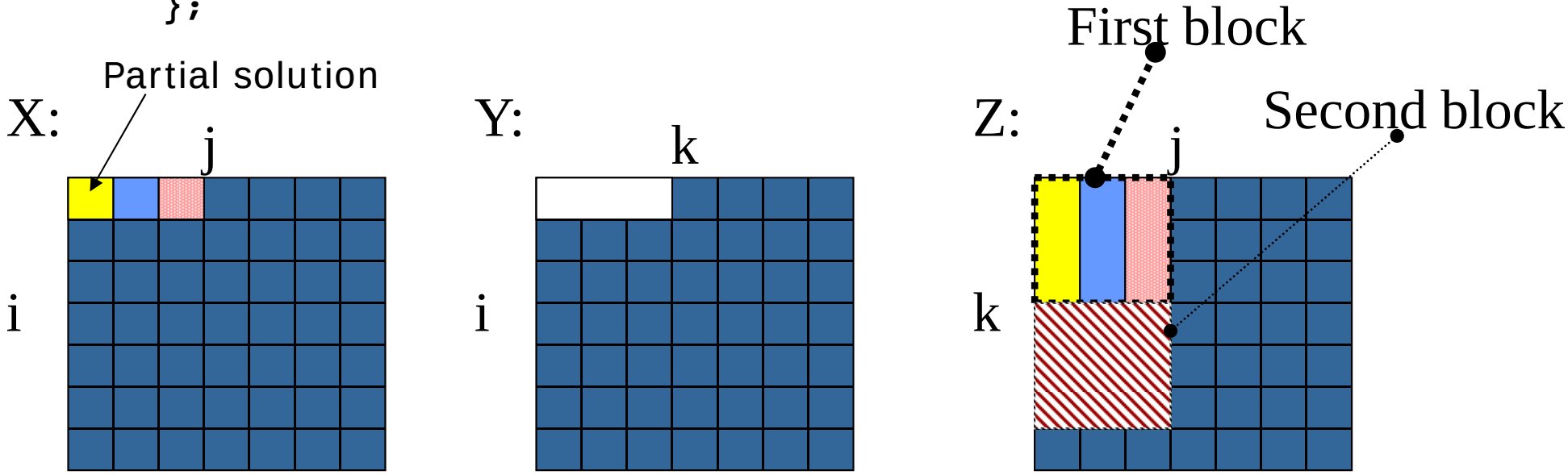
Blocking

```
/* Unoptimized  ARRAY: X =Y * Z */
for (i = 0; i < N; i = i + 1)
   for (j = 0; j < N; j = j + 1)
        {r = 0;
         for (k = 0; k < N; k = k + 1)
                r = r + y[i][k] * z[k][j];
         x[i][j] = r;
        };
```



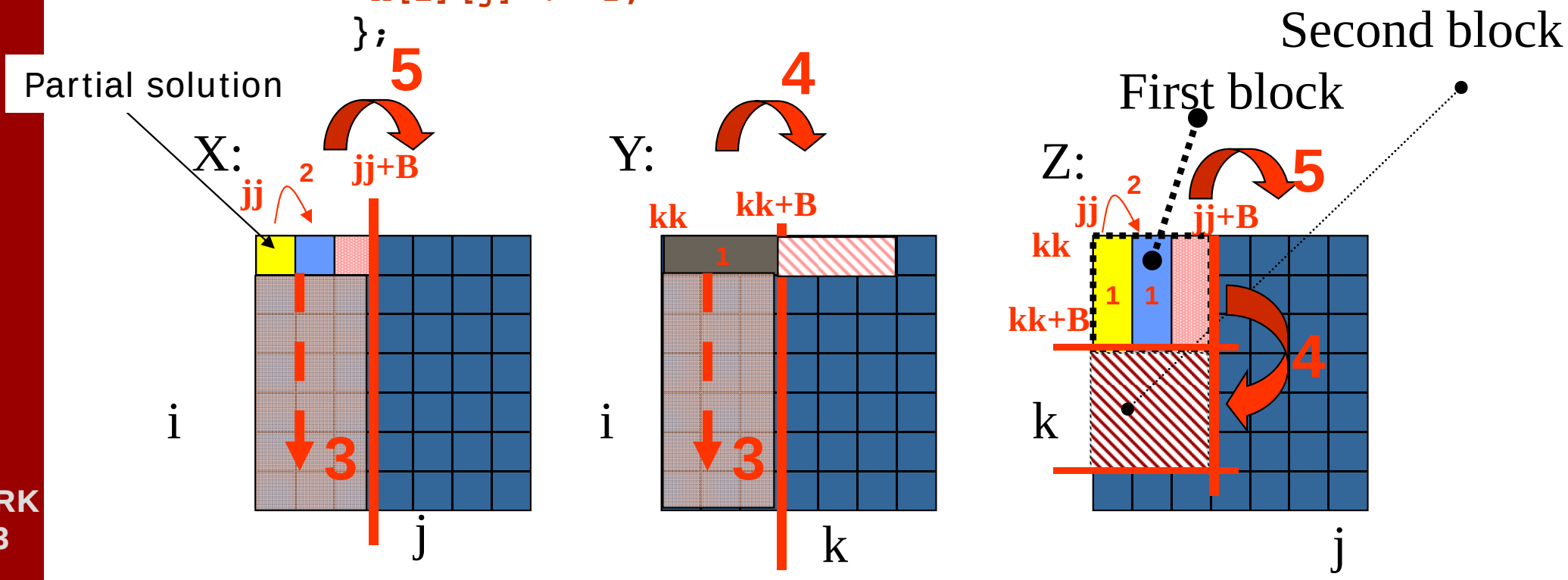
Blocking

```
/* Optimized  ARRAY: X = Y * Z */
for (jj = 0; jj < N; jj = jj + B)
for (kk = 0; kk < N; kk = kk + B)
for (i = 0; i < N; i = i + 1)
   for (j = jj; j < min(jj+B,N); j = j + 1)
        {r = 0;
         for (k = kk; k < min(kk+B,N); k = k + 1)
                r = r + y[i][k] * z[k][j];
         x[i][j] += r;
        };
```



Blocking: the Movie!

```
/* Optimized  ARRAY: X = Y * Z */
for (jj = 0; jj < N; jj = jj + B)                    /* Loop 5 */
for (kk = 0; kk < N; kk = kk + B)                    /* Loop 4 */
for (i = 0; i < N; i = i + 1)                        /* Loop 3 */
   for (j = jj; j < min(jj+B,N); j = j + 1)          /* Loop 2 */
        {r = 0;
         for (k = kk; k < min(kk+B,N); k = k + 1)    /* Loop 1 */
                r = r + y[i][k] * z[k][j];
         x[i][j] += r;
        };
```



Software Prefetch Optimizations

Erik Hagersten

Uppsala University, Sweden

eh@it.uu.se

UPPSALA UNIVERSITET

SW Prefetching

```
/* Unoptimized */
for (j = 0; j < N; j++)
   for (i = 0; i < N; i++)
        x[j][i] = 2 * x[j][i];

/* Optimized */
for (j = 0; j < N; j++)
   for (i = 0; i < N; i++){
        PREFETCH x[j+1][i]
        x[j][i] = 2 * x[j][i];
   }
```

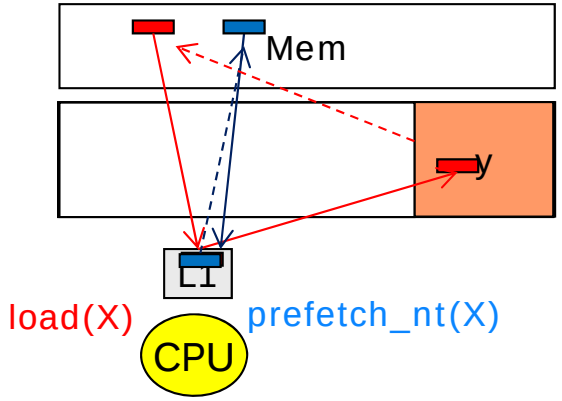
Typically, the HW prefetcher will successfully prefetch sequential streams

Are there examples where SW prefetching can do any better?

AVDARK 2013

UPPSALA UNIVERSITET

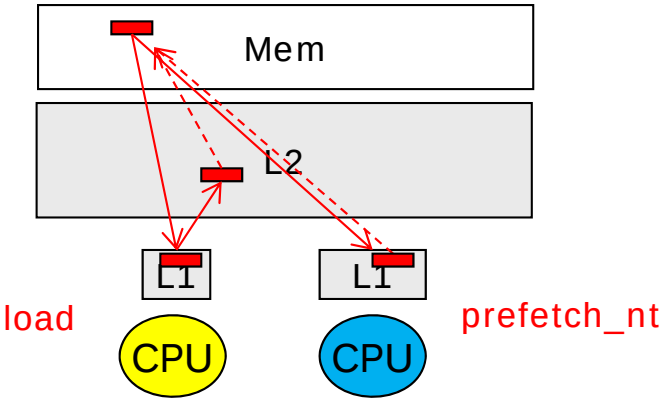
Cache Waste

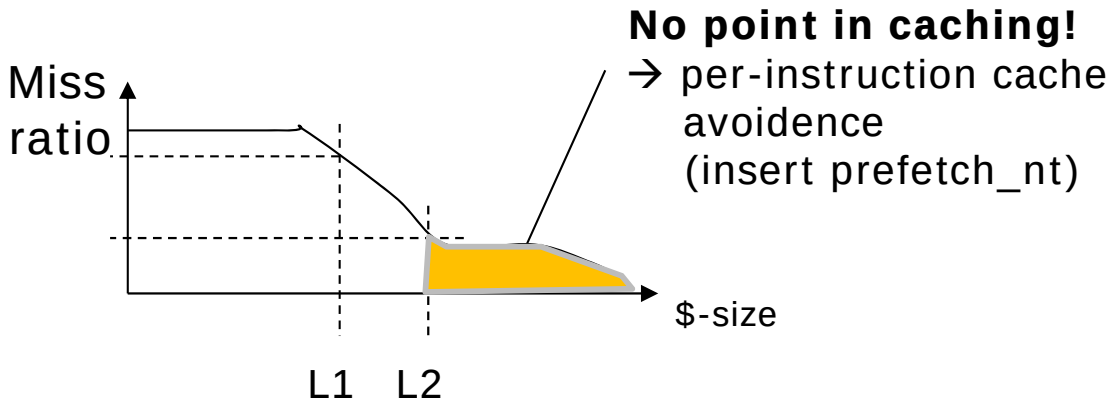


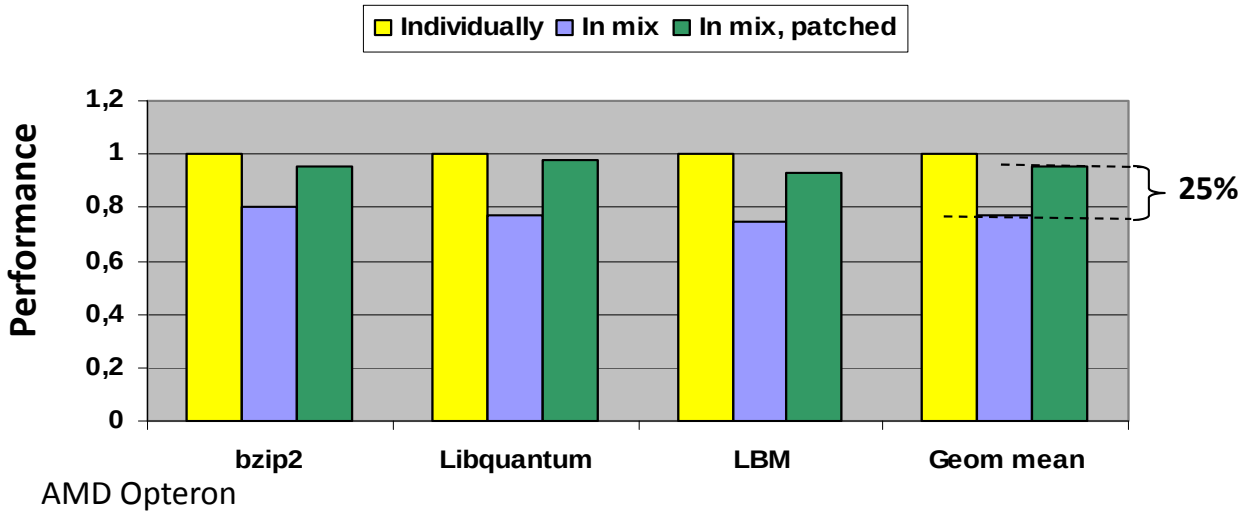

```
/* Unoptimized */
for (s = 0; s < ITERATIONS; s++){
   for (j = 0; j < HUGE; j++)
        x[j] = x[j+1];          /* will hog the cache but not benefit*/
   for (i = 0; i < SMALLER_THAN_L2_CACHE; i++)
        y[i] = y[i+1];          /* will be evicted between usages /*
}
/* Optimized */
for (s = 0; s < ITERATIONS; s++){
   for (j = 0; j < HUGE; j++) {
        PREFETCH_NT x[j+1] /* will be installed in L1, bypass L2 */
         x[j] = x[j+1];
   for (i = 0; I < SMALLER_THAN_L2_CACHE; i++)
        y[i] = y[i+1];      /* will always hit in the cache*/
}
```

➔ Can also be beneficial if applications are co-scheduled on MC and share cache with other applications.

UART: Avoiding cache waste (A. Sandberg)







AMD Opteron

UPPSALA UNIVERSITET

UART: Resource-lean SW prefetching (Muneeb)

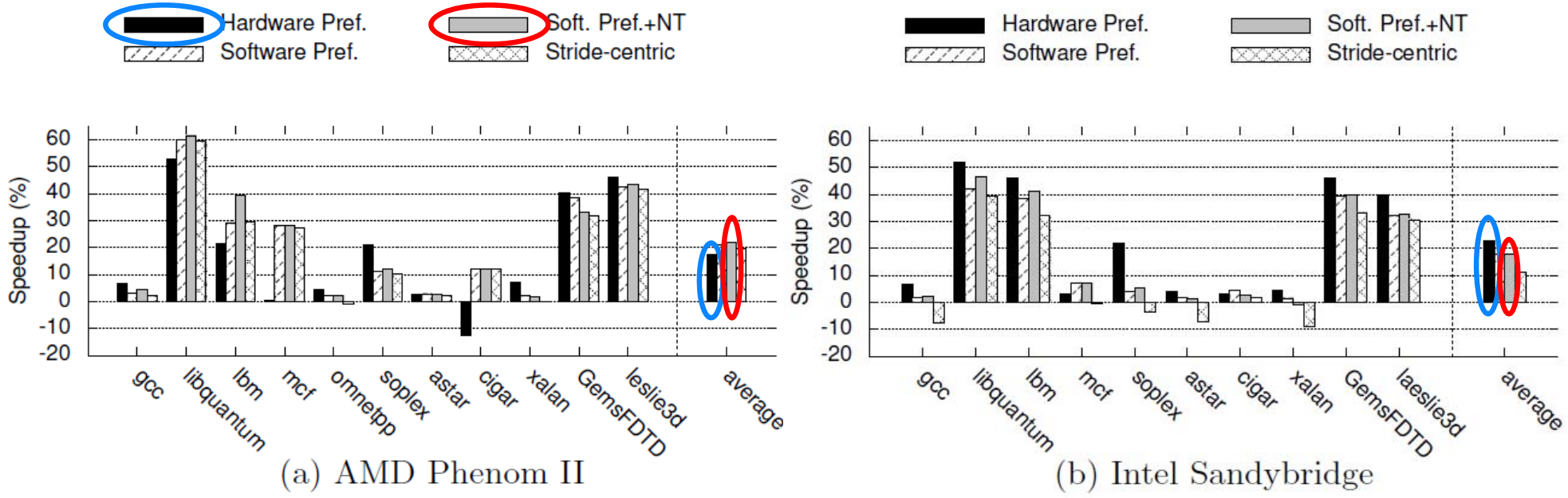
- Use the “UART modeling” (StatStack) to “understand” the application (run once with a 20% overhead)
- Find instructions that miss in the cache
- Capture their popular strides
- Figure out the necessary prefetch distances
- Figure out the best non-temporal strategy
- Do cost/benefit analysis
- Insert prefetches into binary automatically (actually: into ASM)
  - Use the right stride
  - Use the right prefetch distance

AVDARK 2013

Speedup comparison

Existing HW prefetching

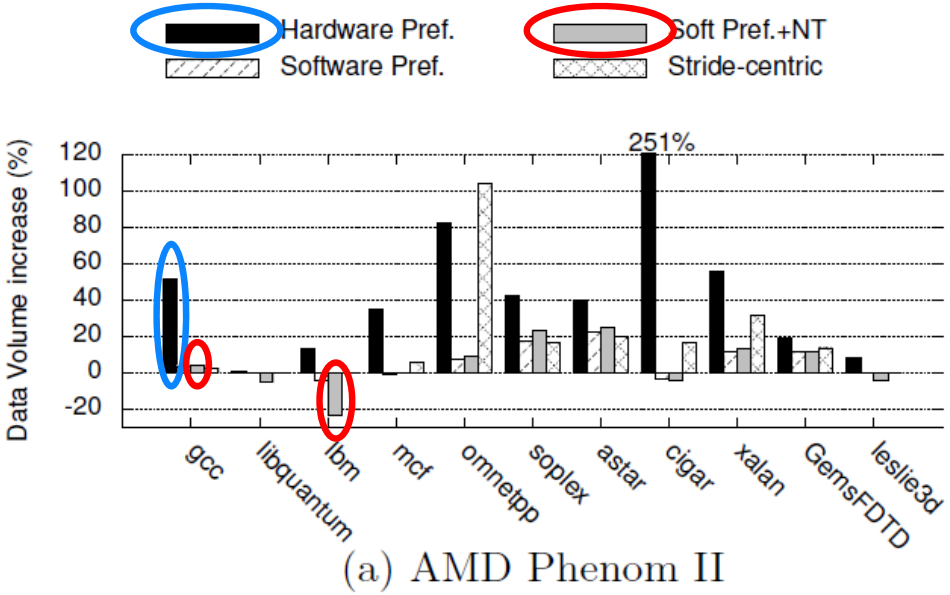
Muneeb's proposal with HW pref. turned off



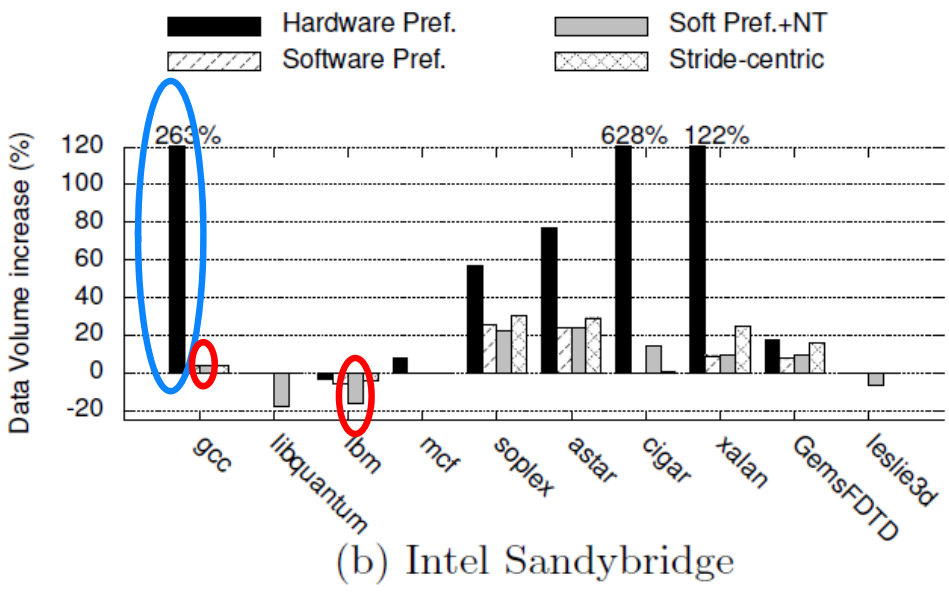
Data Volume Increase Caused by Prefetching

Existing HW prefetching

Muneeb's proposal with HW pref. turned off



(a) AMD Phenom II



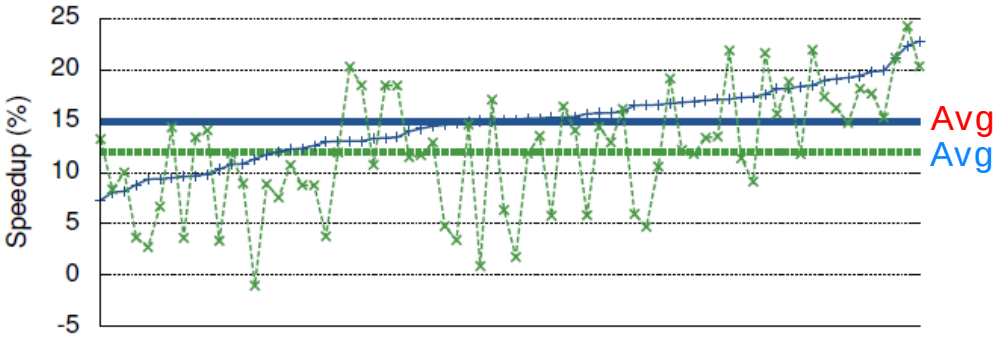
(b) Intel Sandybridge

Running many apps at the same time on a MC

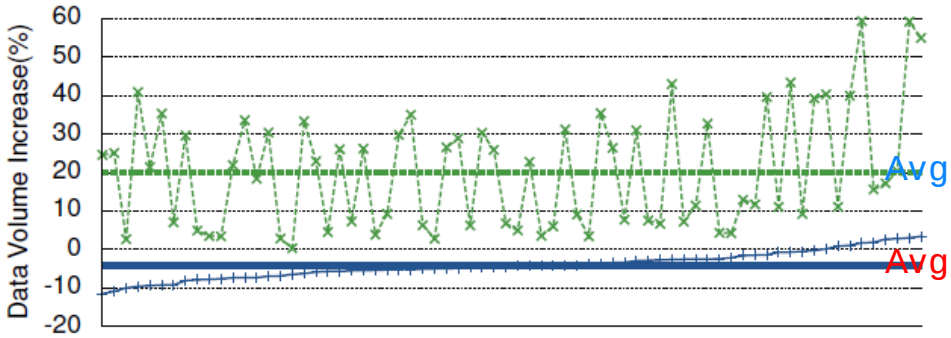
Muneeb's proposal with HW pref. turned off

Existing HW prefetching

Sandybridge

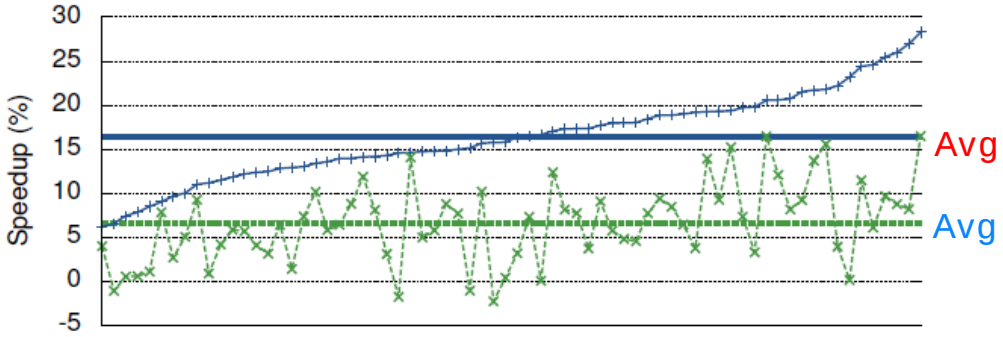


Speedup (higher is better)

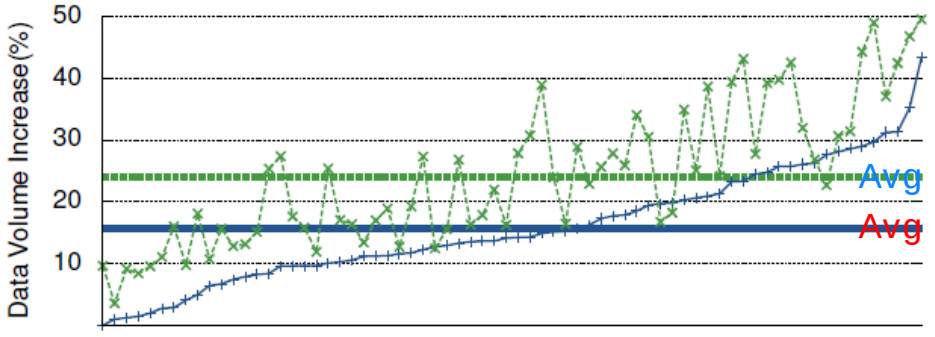


Data Volume (lower is better)

Phenom II

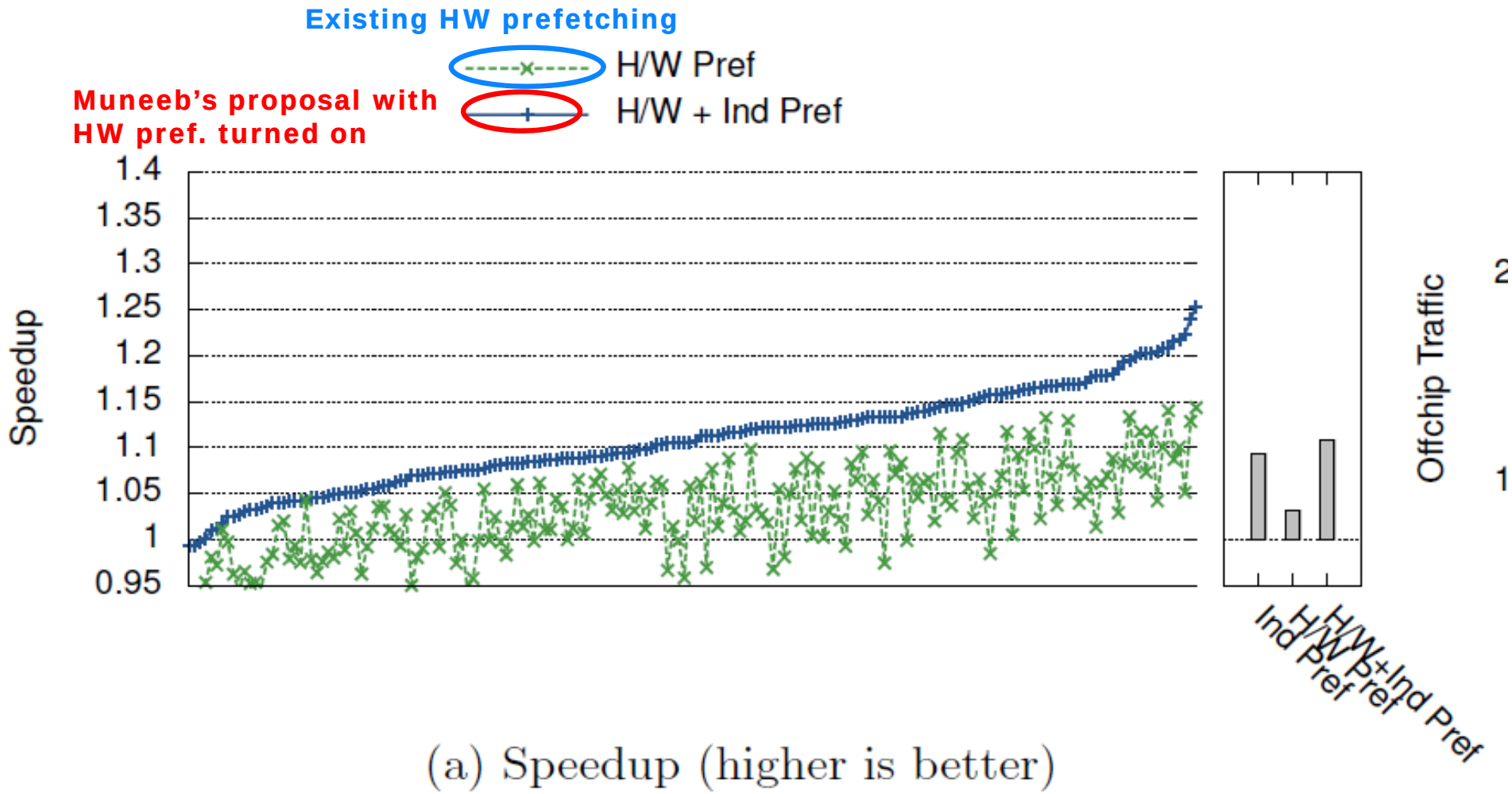


Speedup (higher is better)



Data Volume (lower is better)

Now also non-strided prefetching



(a) Speedup (higher is better)

Example of Performance Tools: Vtune/ThreadSpotter

Erik Hagersten
Uppsala University, Sweden
eh@it.uu.se

UPPSALA UNIVERSITET

Some performance tools

Free tools

- Oprofile
- GNU: gprof
- AMD: code analyst
- Google performance tools
- Virtual Inst: High Productivity Supercomputing (http://www.vi-hps.org/tools/)
- …

Commercial tools

- Intel: Vtune and many more — Covered here
- HP: Multicore toolkit (some free, some not)
- ThreadSpotter (of course☺) — Covered here
- …

AVDARK 2013

UPPSALA UNIVERSITET

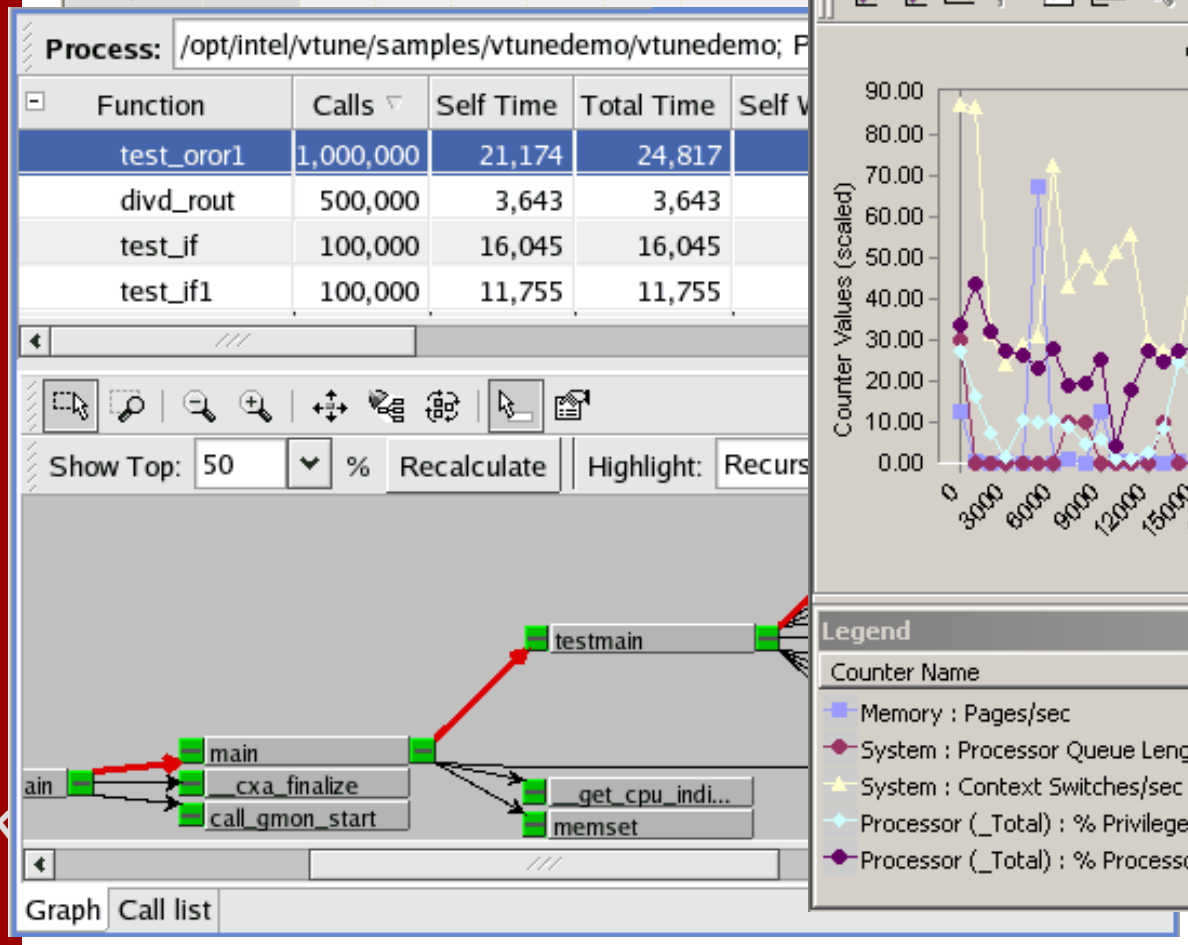
VTune

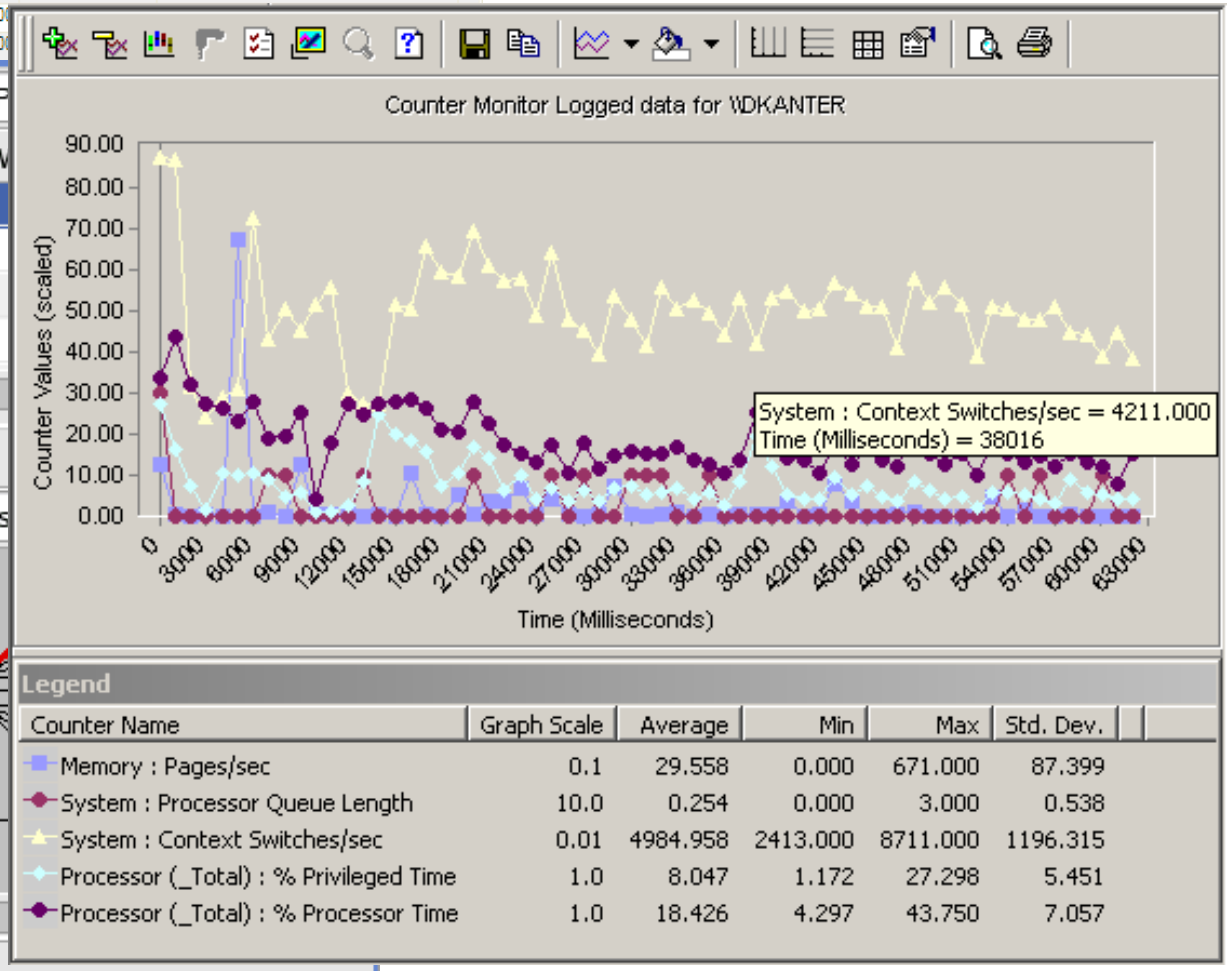
- Broad-spectrum, low-level tool
  - Profiling (where do you spend your time)
  - HW ctr-based information
    - CPI…
    - I1, D1, L2, L3 cache misses…
    - Branch prediction
    - Per: application, function, instruction
- Often requires architecture knowledge to understand the stats (so you should be fine☺)

AVDARK 2013

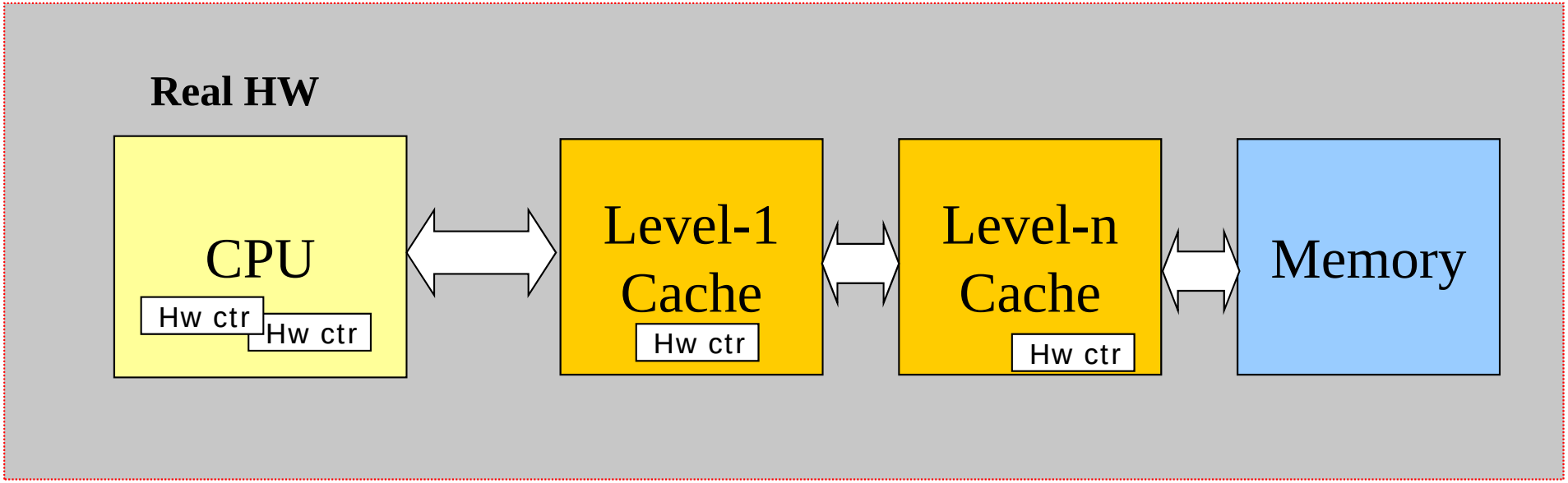
Example of Vtune info

Module	Process	Clock sampl	Instructi Retire...	Cycles per...	Clocktic %	Instructi Retired..	Clockticks events	Instructions Retired events	Process Path	P
ntkrnlpa.exe	firefox.exe	8,867	1,059	8.373	48.52%	7.97%	14,781,289,000	1,765,353,000	c:\proj\mozilla-cvs\firefox-opt\dist\bin\	33
igxpdv32.DLL	firefox.exe	5,452	10,789	0.505	29.83%	81.16%	9,088,484,000	17,985,263,000	c:\proj\mozilla-cvs\firefox-opt\dist\bin\	33
xul.dll	firefox.exe	1,316	671	1.961	7.20%	5.05%	2,193,772,000	1,118,557,000	c:\proj\mozilla-cvs\firefox-opt\dist\bin\	33
win32k.sys	firefox.exe	962	288	3.340	5.26%	2.17%	1,603,654,000	480,096,000	c:\proj\mozilla-cvs\firefox-opt\dist\bin\	33
hal.dll	firefox.exe	905	335	2.701	4.95%	2.52%	1,508,635,000	558,445,000	c:\proj\mozilla-cvs\firefox-opt\dist\bin\	33
igxpgd32.dll	firefox.exe	190	17	11.176	1.04%	0.13%	316,730,000	28,339,000	c:\proj\mozilla-cvs\firefox-opt\dist\bin\	33
ntdll.dll	firefox.exe	126	45	2.800	0.69%	0.34%	210,042,00			
ar5211.sys	firefox.exe	98	0	0.000	0.54%	0.00%	163,366,00			





Using Progammable HW Counters



Runs at near native speed!

ThreadSpotter: ”Cache Tool for Dum”

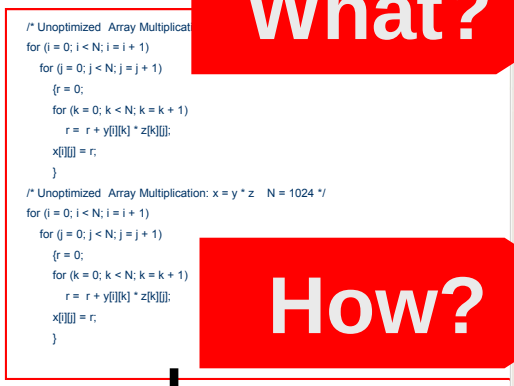
Help!

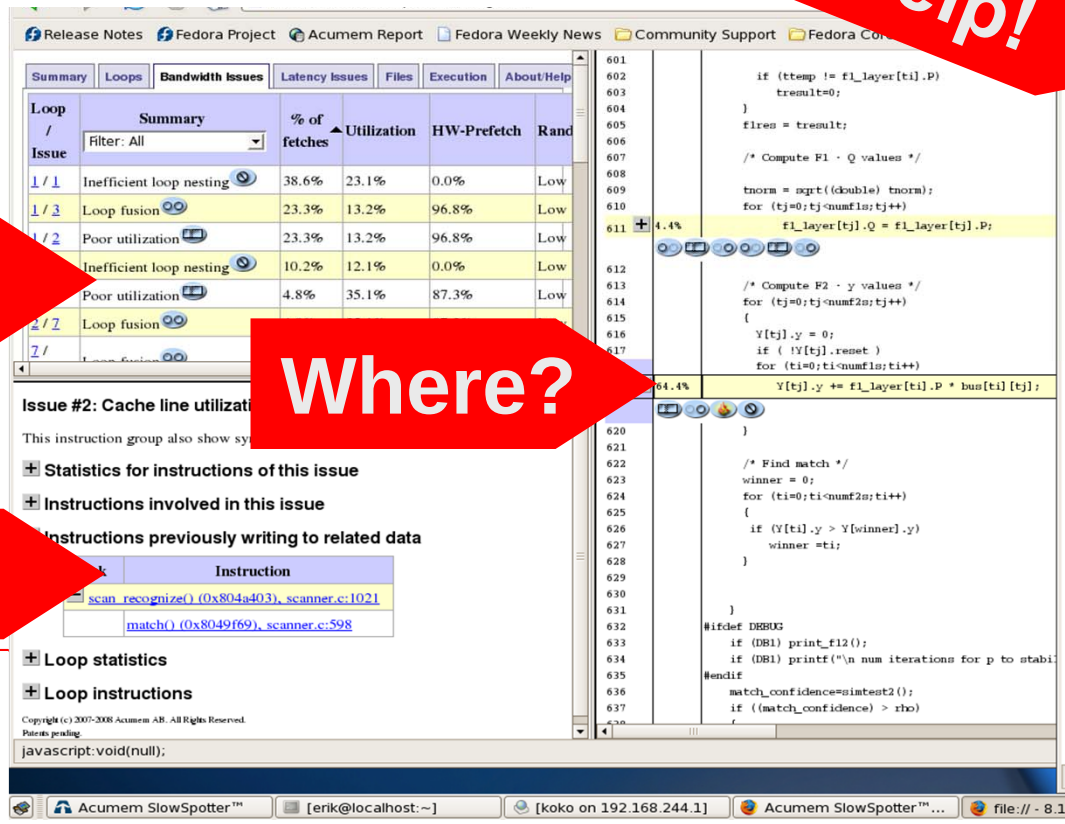
Source:
C, C++, For

What?

Where?

How?





p instructions
(c) 2007-2008 Acumem AB. All Rights Reserved.
ing.

A *poor cache line utilization* issue indicates that a part of the locality, that is, cache lines are only partially used. The unus cache, which means that memory bandwidth and cache spac data is wasted.

The poor utilization issue has these sections:

- Statistics for instructions of this issue
- Instructions involved in this issue
- Instructions previously writing to related data
- Loop statistics
- Loop instructions

Poor cache line utilization can have a number of causes:

- There may be structures with unused fields, see Sectio Structures”.
- There may be padding inserted into structures or betw data alignment, see Section 5.1.3, “Alignment Problen
- There may be housekeeping data from the dynamic m objects, see Section 5.1.4, “Dynamic Memory Allocat
- It may be caused by irregular access patterns, see Sect Pattern”.

Done

Any Compiler

Binary

Host System

”Application locality”

Sampler

Finger Print (~4MB)

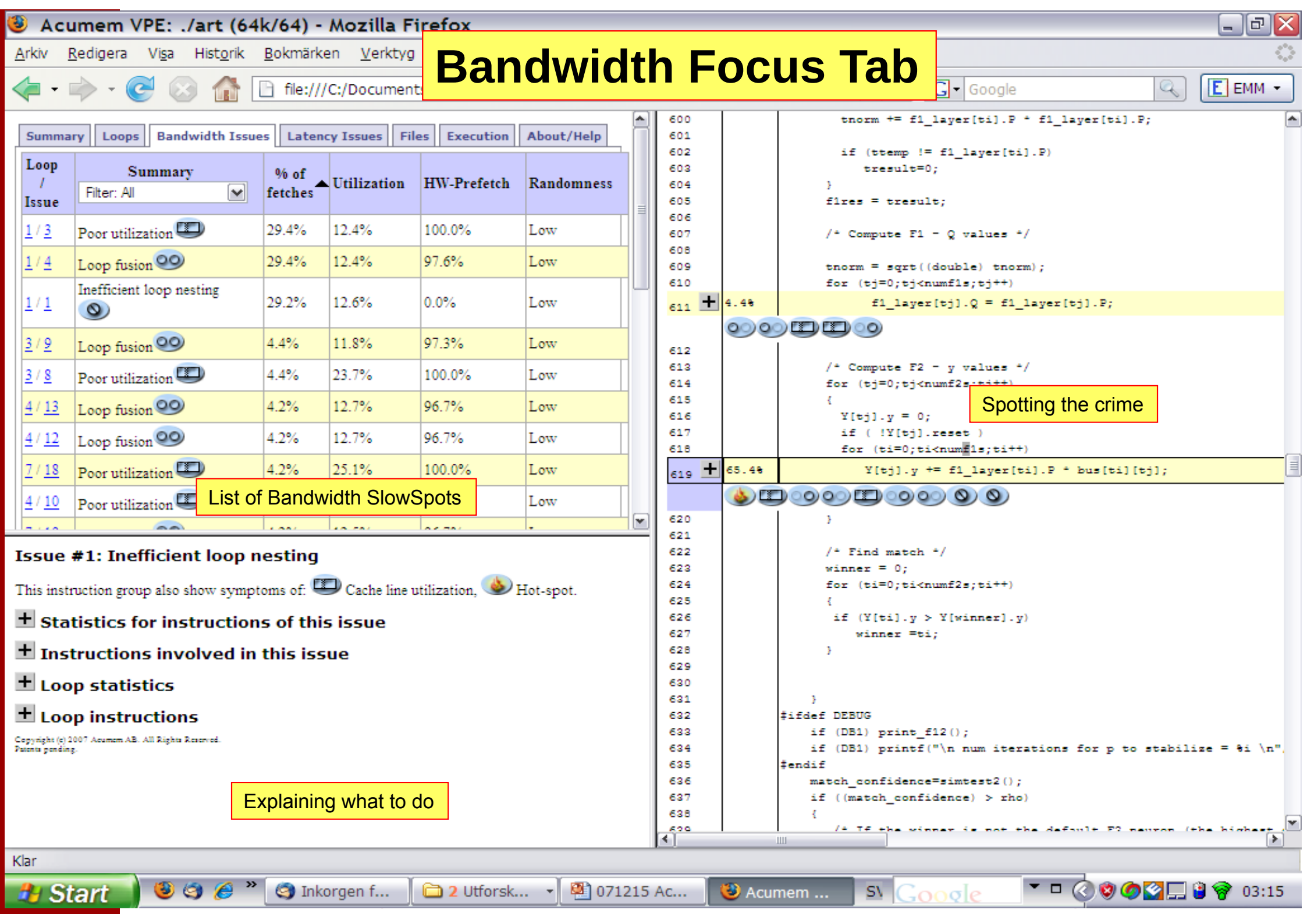
UART technology

Analysis

Advice

Target System Parameters

AVDARK 2013


Acumem VPE: ./art (64k/64) - Mozilla Firefox
Arkiv Redigera Visa Historik Bokmärken Verktyg
Bandwidth Focus Tab
Summary Loops Bandwidth Issues Latency Issues Files Execution About/Help
Loop / Issue
Summary
Filter: All
% of fetches
Utilization
HW-Prefetch
Randomness
1 / 3 Poor utilization 29.4% 12.4% 100.0% Low
1 / 4 Loop fusion 29.4% 12.4% 97.6% Low
1 / 1 Inefficient loop nesting 29.2% 12.6% 0.0% Low
3 / 9 Loop fusion 4.4% 11.8% 97.3% Low
3 / 8 Poor utilization 4.4% 23.7% 100.0% Low
4 / 13 Loop fusion 4.2% 12.7% 96.7% Low
4 / 12 Loop fusion 4.2% 12.7% 96.7% Low
7 / 18 Poor utilization 4.2% 25.1% 100.0% Low
4 / 10 Poor utilization Low
List of Bandwidth SlowSpots
Issue #1: Inefficient loop nesting
This instruction group also show symptoms of: Cache line utilization, Hot-spot.
Statistics for instructions of this issue
Instructions involved in this issue
Loop statistics
Loop instructions
Explaining what to do
Spotting the crime
4.4%
65.4%
Klar
Start
Inkorgen f...
2 Utforsk...
071215 Ac...
Acumem ...
03:15

Acumem VPE: ./art (64k/64) - Mozilla Firefox

Arkiv Redigera Visa Historik Bokmärken Verktyg Hjälp

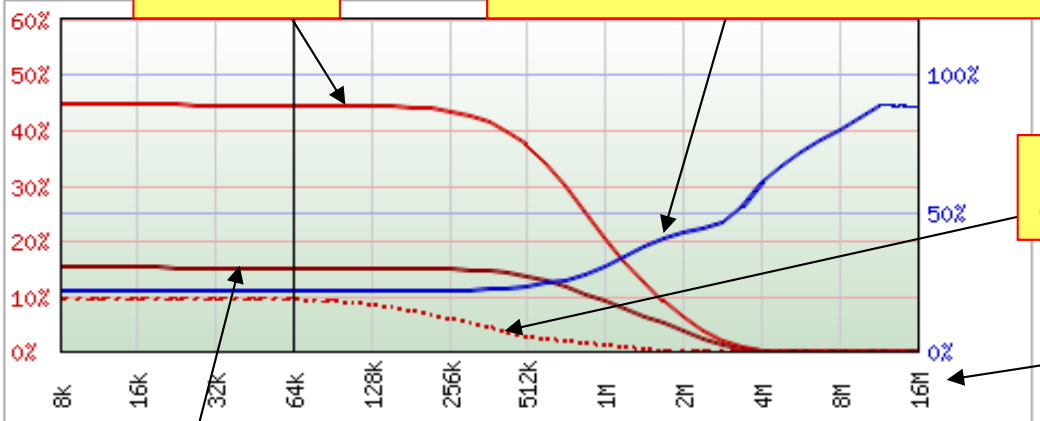
file:///C:/Documents%20and%20Settings/Erik/Mina%20dokument/Demo/071112

Summary | Loops | Bandwidth Issues | Latency Issues | Files | Execution | About/Help

Fetch ratio

Miss rates

Fetch utilization ≈ Fraction of fetched cache data used before eviction



Predicted fetch ratio (if utilization → 100%)

Cache size

Miss ratio

Legend: -- Fetch rate .. Util. corr. fetch rate -- Miss rate -- Utilization

Miss rate	15.1%
Fetch rate	44.4%
Misses	5.45e+07
Fetches	1.60e+08
Cache line utilization	21.9%
Cache size	64k
Line size	64
Replacement policy	random

Select a file in the file table, or follow a source code link from

op description.

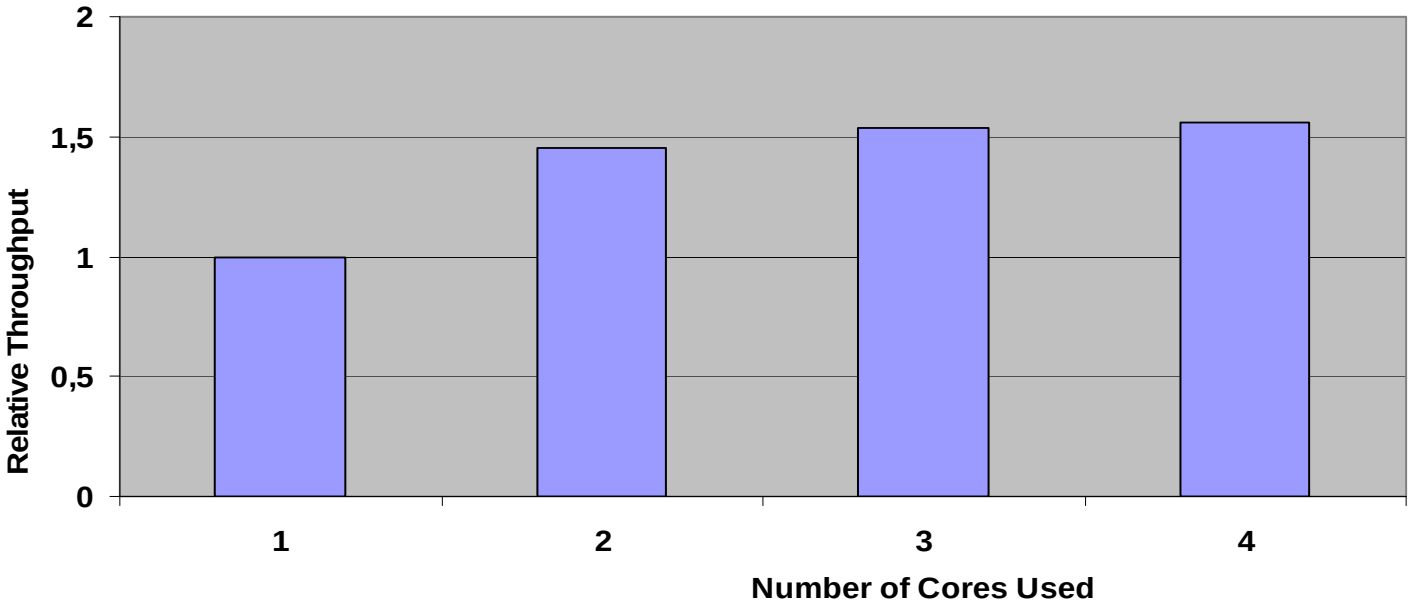
Klar

Start 3 U... Auto... Skick... Micr... Acu... SV Google 18:36

Resource Sharing Example

Libquantum
A quantum computer simulation
Widely used in research (download from: http://www.libquantum.de/)
4000+ lines of C, fairly complex code.
Runs an experiment in ~30 min

Throughput improvement:



Demo Time!

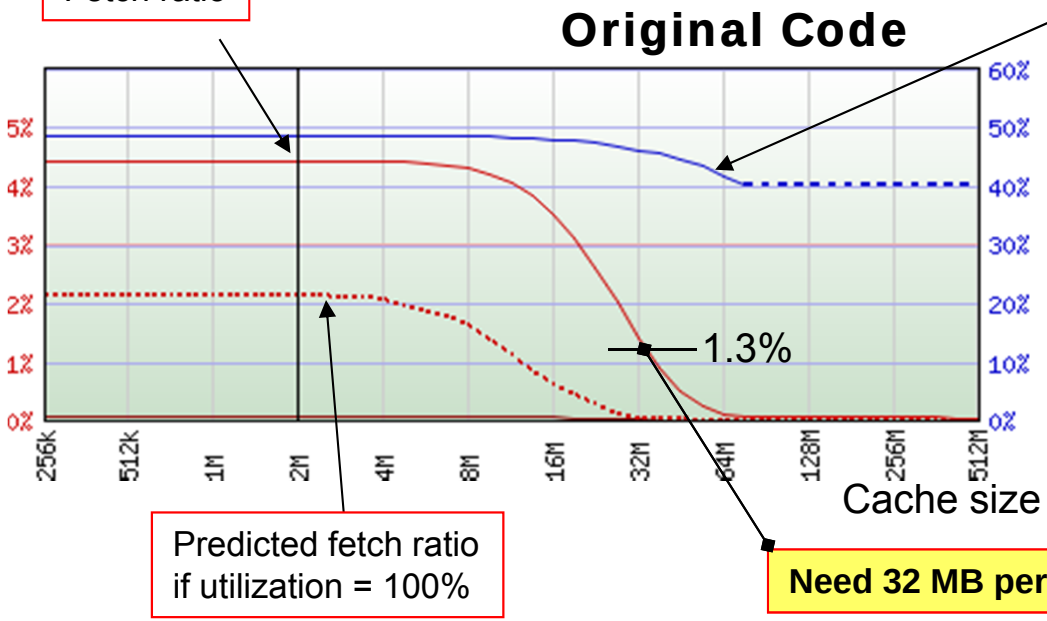
Libquantum
Libquantum

Fetch Utilization Analysis

Libquantum

Fetch utilization ≈ Fraction of cache data utilized

Fetch ratio

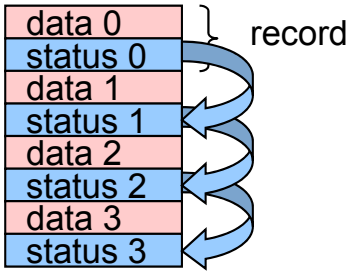



```
for (i=0; i++; i<MAX) {
     ... = huge_data[i].status + ...
 }
```

Predicted fetch ratio if utilization = 100%

Need 32 MB per thread!

Main data structure (vector of structs)

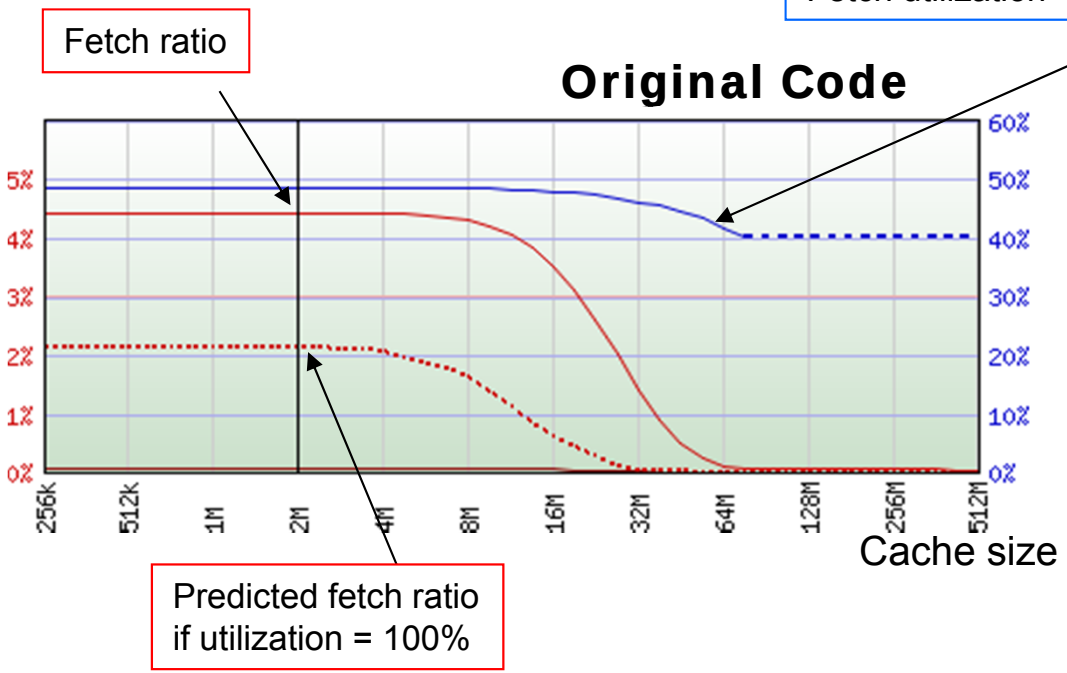


Only accessing status data in main loop

Utilization Analysis

Libquantum

Fetch utilization ≈ Fraction of cache data utilized



Utilization Optimization

```
for (i=0; i++; i<MAX) {
     ... = huge_data[i].status   ...
 }
```

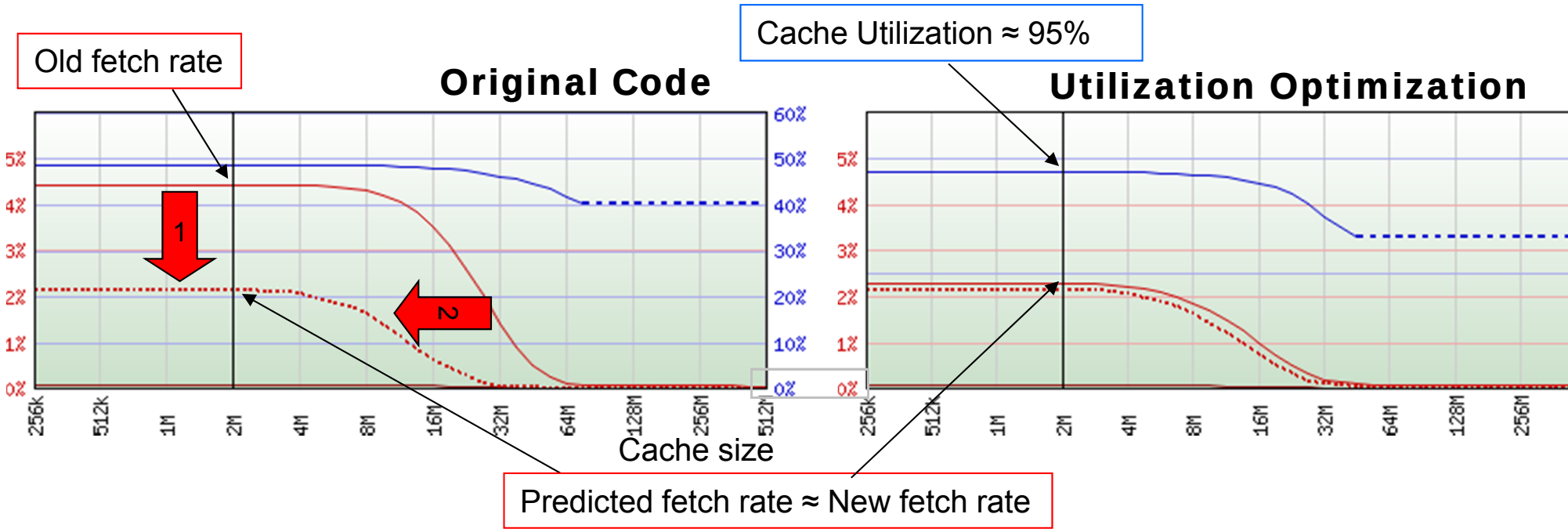
```
for (i=0; i++; i<MAX) {
     ... = huge_data_status[i]   ...
 }
```

ThreadSpotter's First Advice: Improve Utilization

➔Change to "struct of vectors"

- Involves ~20 lines of code
- Takes a non-expert 30 min

Utilization Optimization



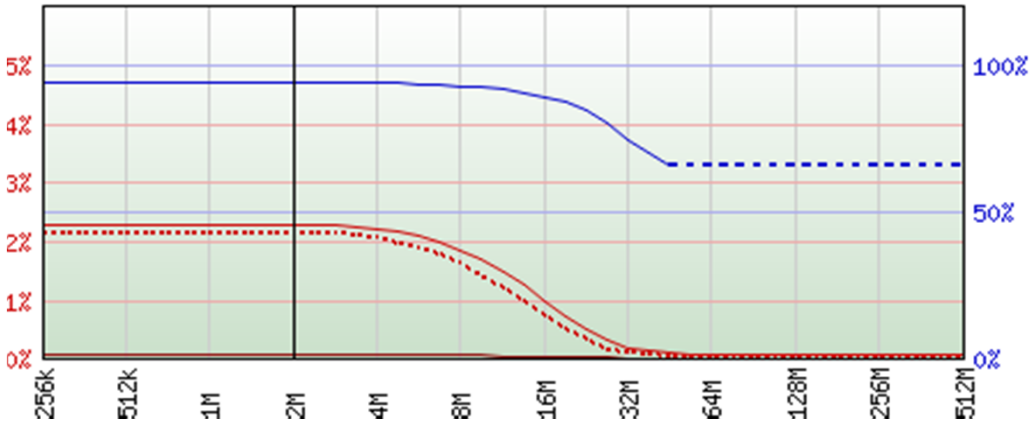
Two positive effects from better utilization

1. Each fetch brings in more useful data → lower fetch ratio
2. The same amount of useful data can fit in a smaller cache → shift left

UPPSALA UNIVERSITET

Reuse Analysis

After Utilization Optimization



Utilization + Fusion Optimization

```
...
toffoli(huge_data_status, ...)
toffoli(huge_data_status, ...)
...
```

```
...
fused_toffoli(huge_data_status,...)
...
```

Second-Fifth ThreadSpotter Advice: Improve reuse of data

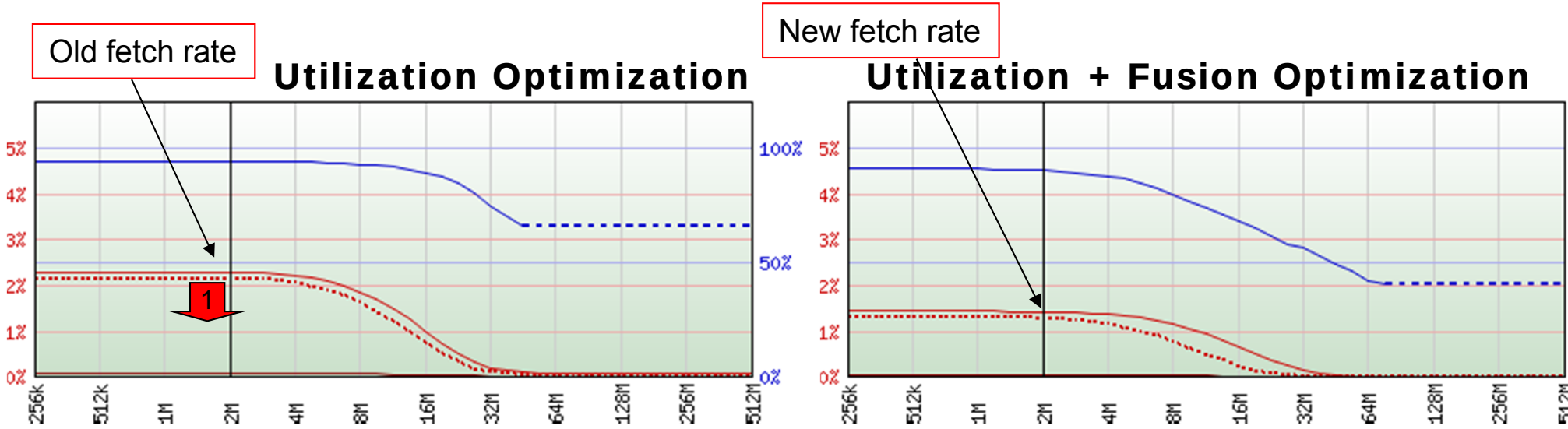
➔Fuse functions traversing the same data

- Here: four new fused functions created
- Takes a non-expert < 2h

AVDARK 2013

Effect: Loop Fusion

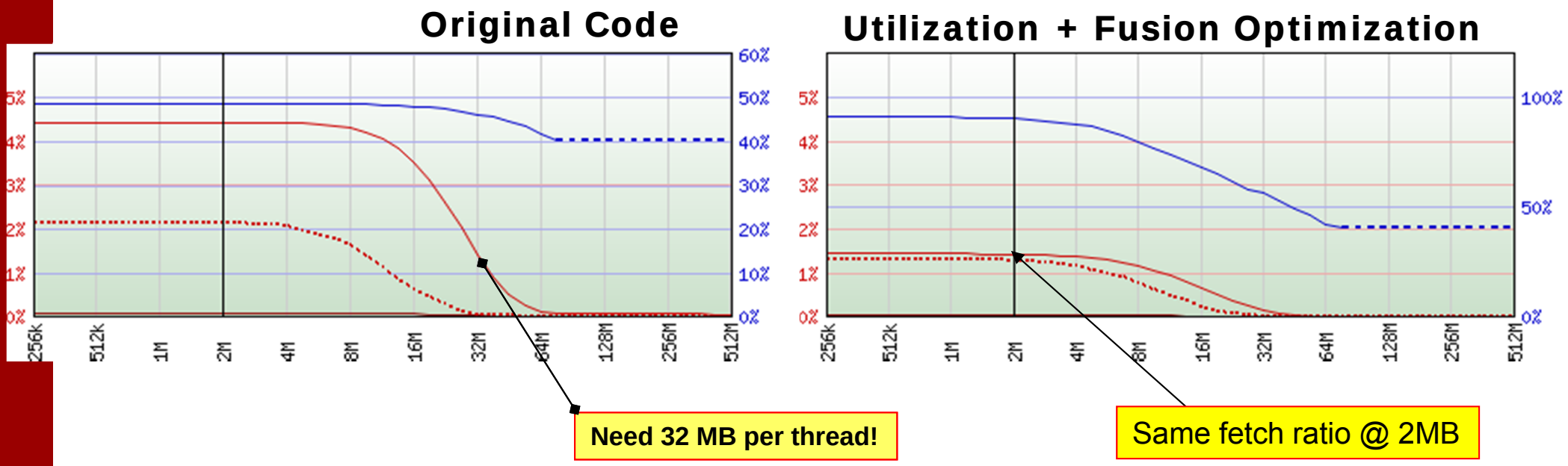
SPEC CPU2006-462.libquantum



- The miss in the second loop goes away
- Still need the same amount of cache to fit “all data”

UPPSALA UNIVERSITET

Utilization + Reuse Optimization

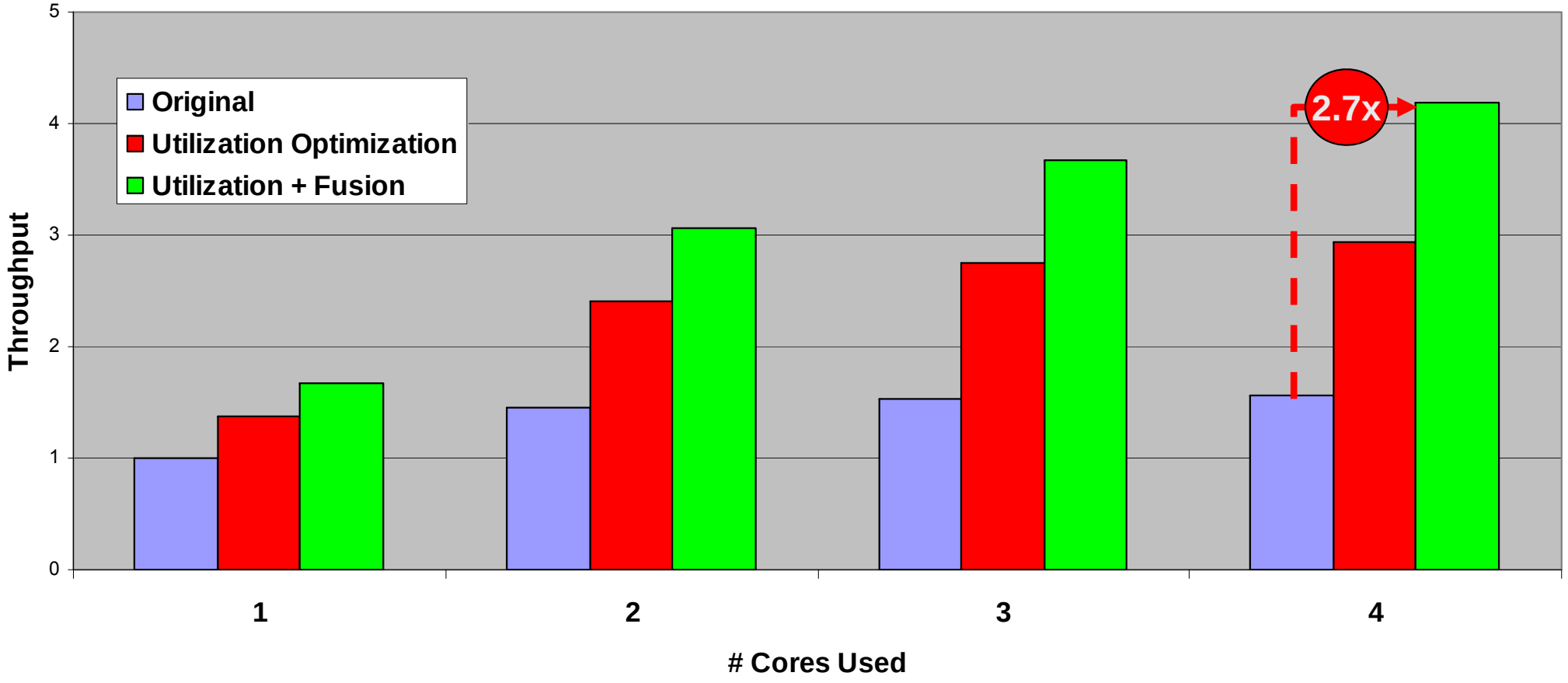


- Fetch rate down to 1.3% for 2MB caches
- Same as a 32 MB cache originally

AVDARK 2013

Summary

Libquantum



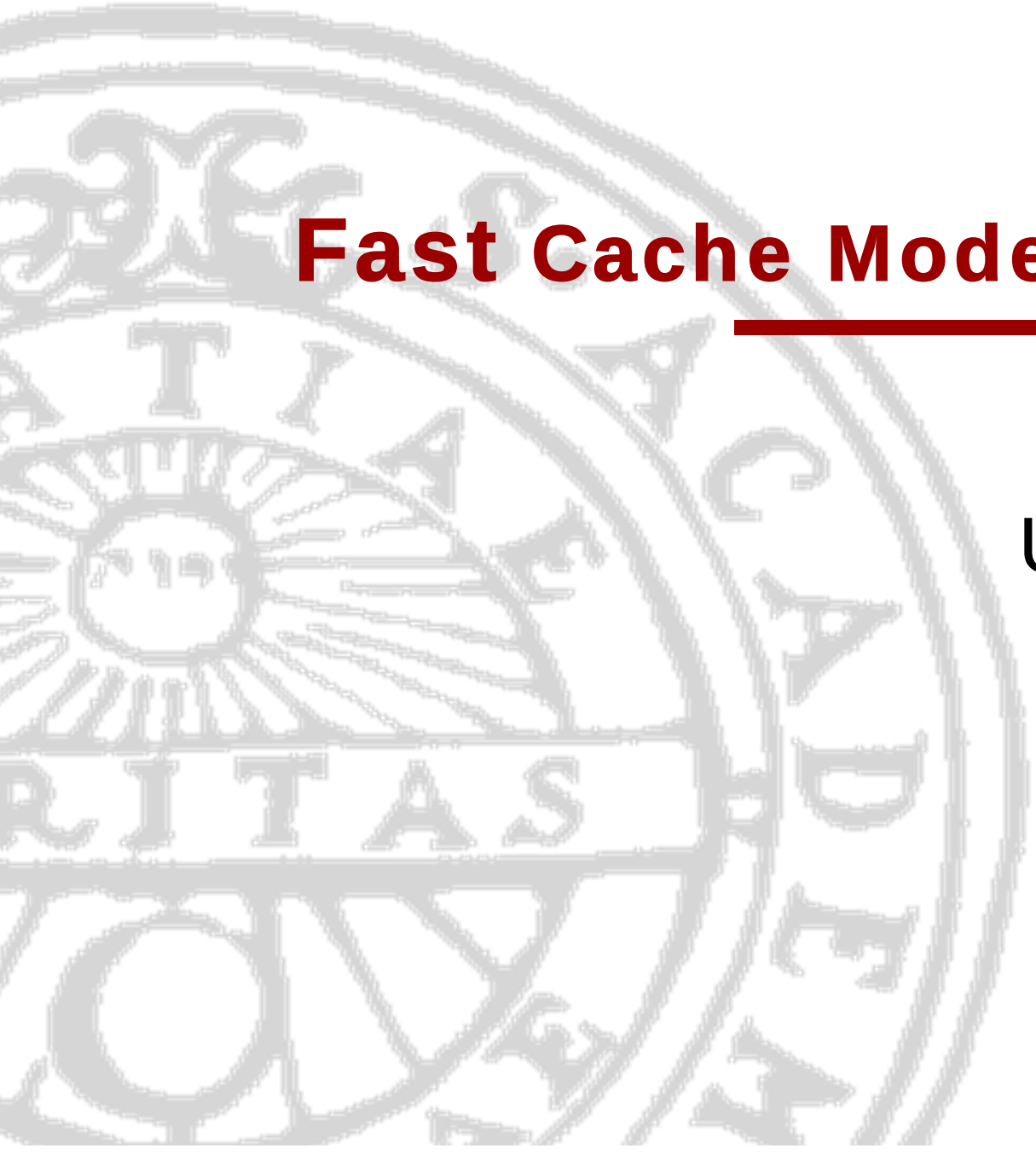
AVDARK 2013

UPPSALA UNIVERSITET

Can Compilers Find These Optimizations?

- That's what they are trying to do…
- They do a good job at small simple examples
- Sometimes fail for large complex applications

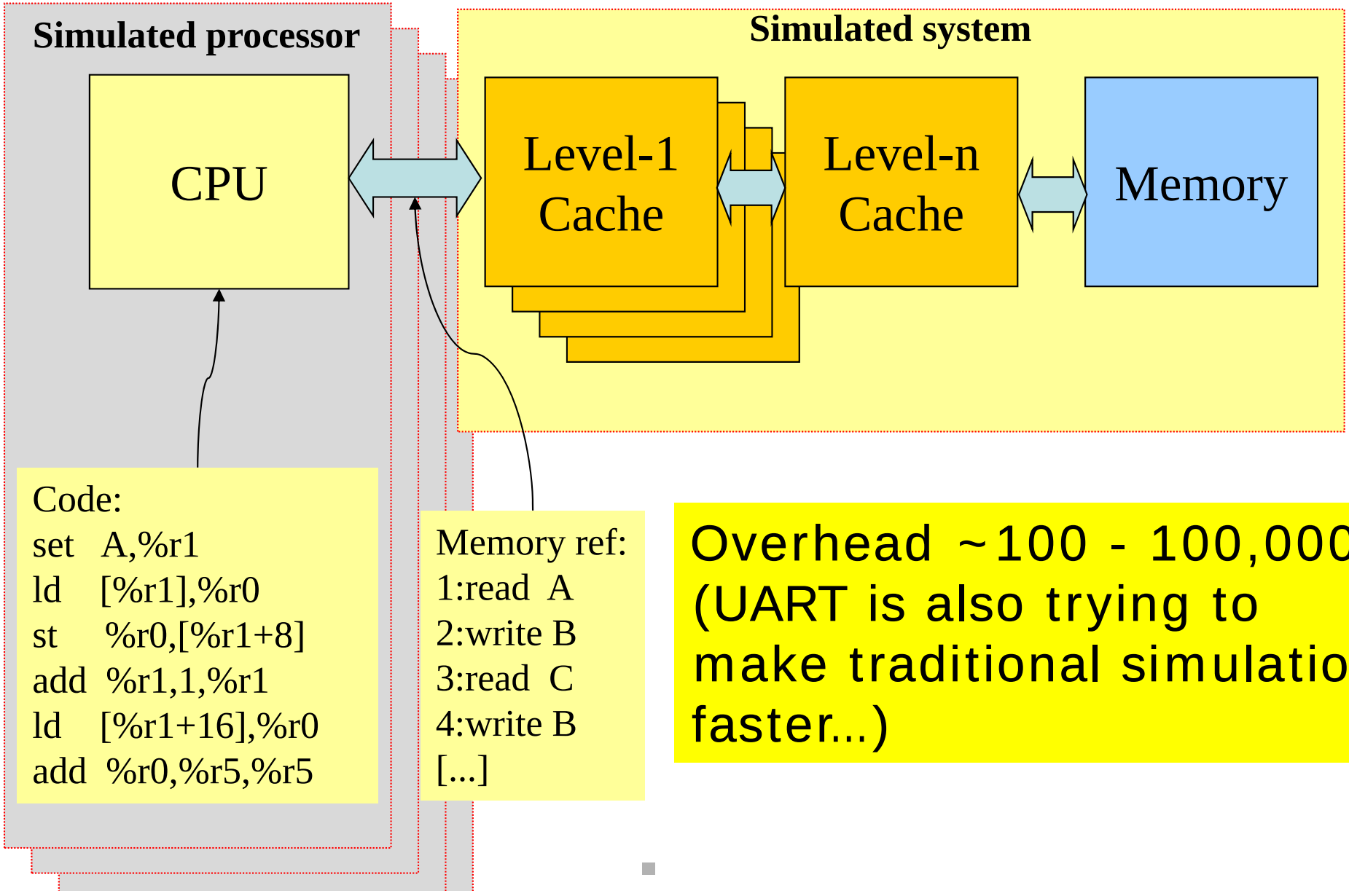
AVDARK 2013



Fast Cache Modeling Technology

Erik Hagersten
Uppsala University

Understanding Performance and Power: Traditional simulation model

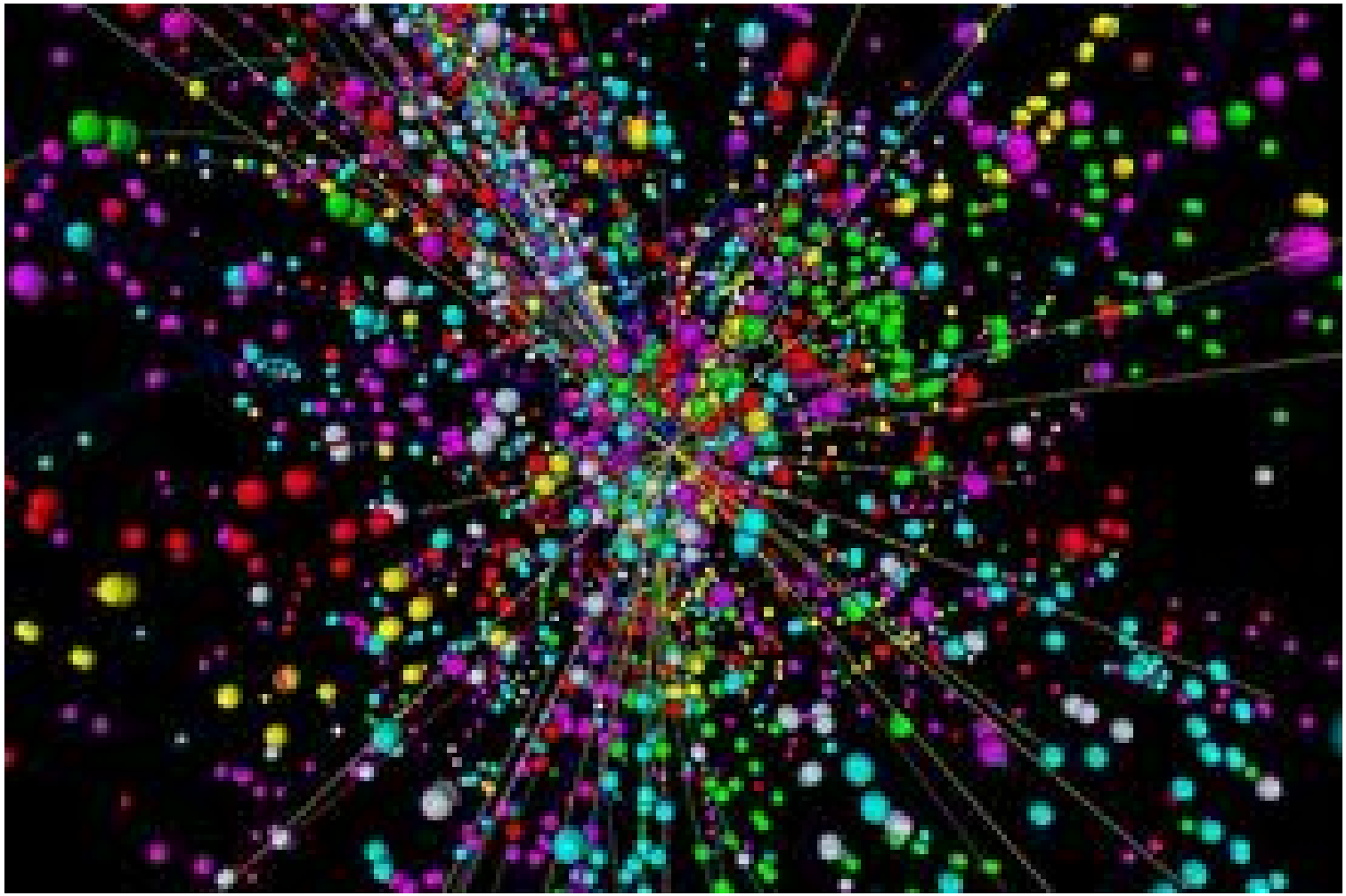


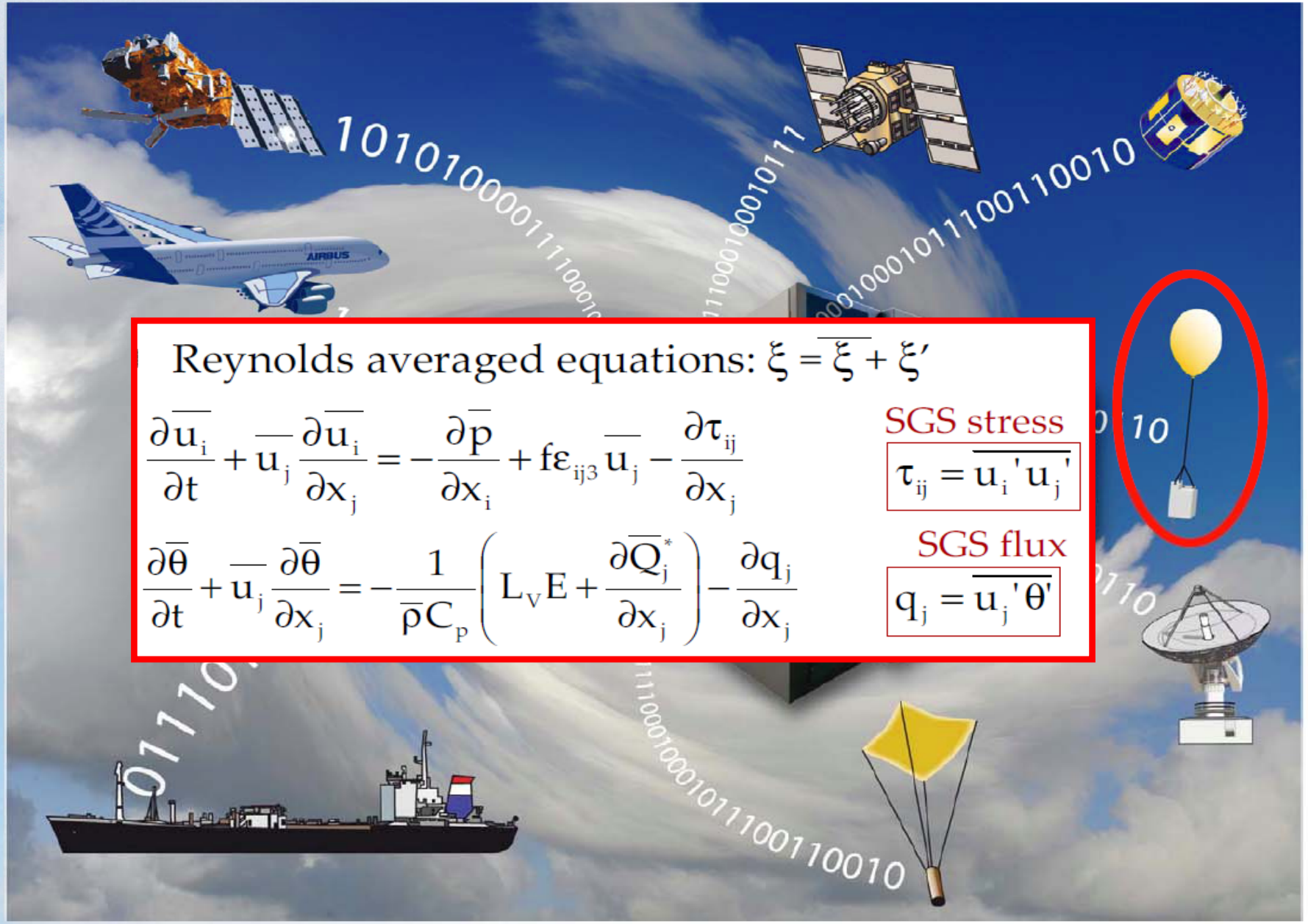
Overhead ~100 - 100,000x (UART is also trying to make traditional simulation faster...)

Our Goal

- Estimate performance and power effects for
  - Programmers
  - Compilers / compiler writers
  - JIT
  - Runtime systems
  - (... and Architects)
- Runtime Overhead <2x (i.e., <100% over native)
- Architecturally-independent in-data
- Flexible models (answer what-if questions)

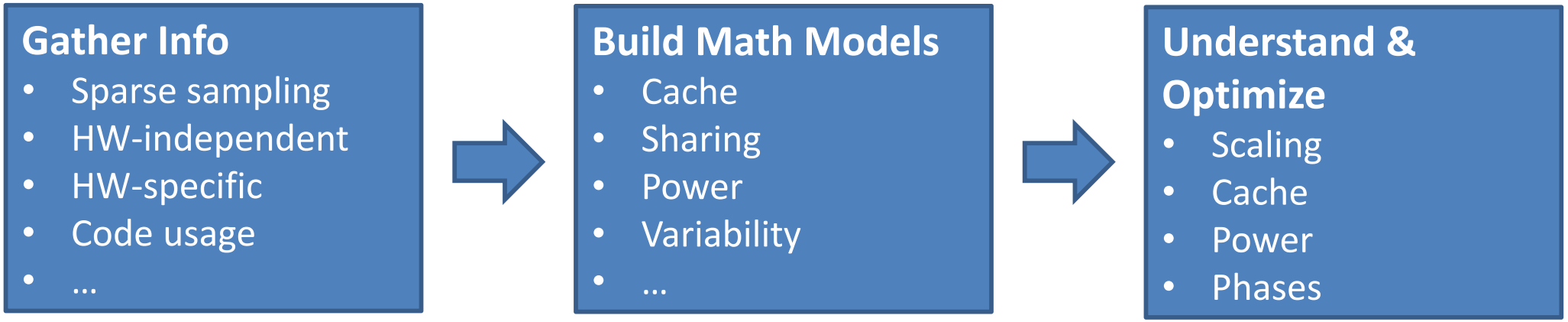
Simulating Particles in Physics

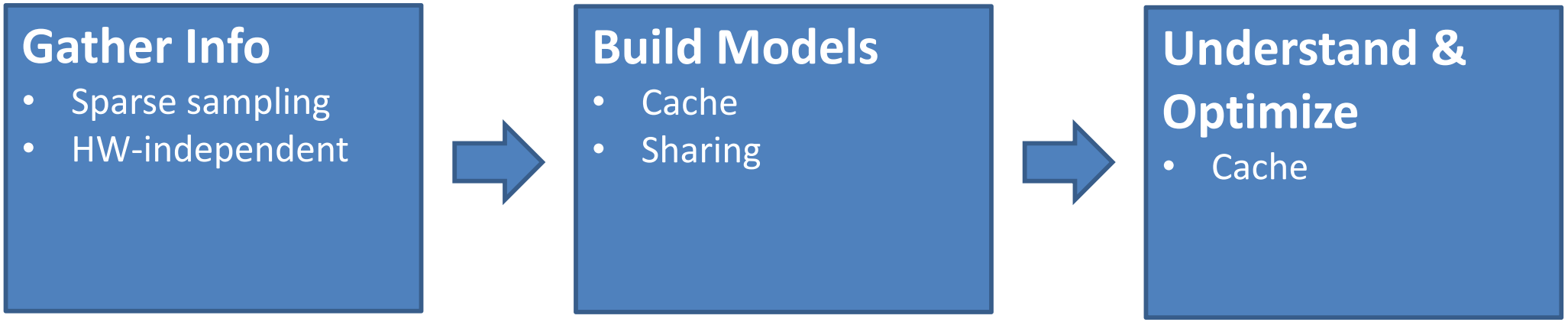




Source: Erland Källén, Celsius/Linnaeus Symposium 2013 & Philippe Drobinski ENS-UPS – 12 May 2011

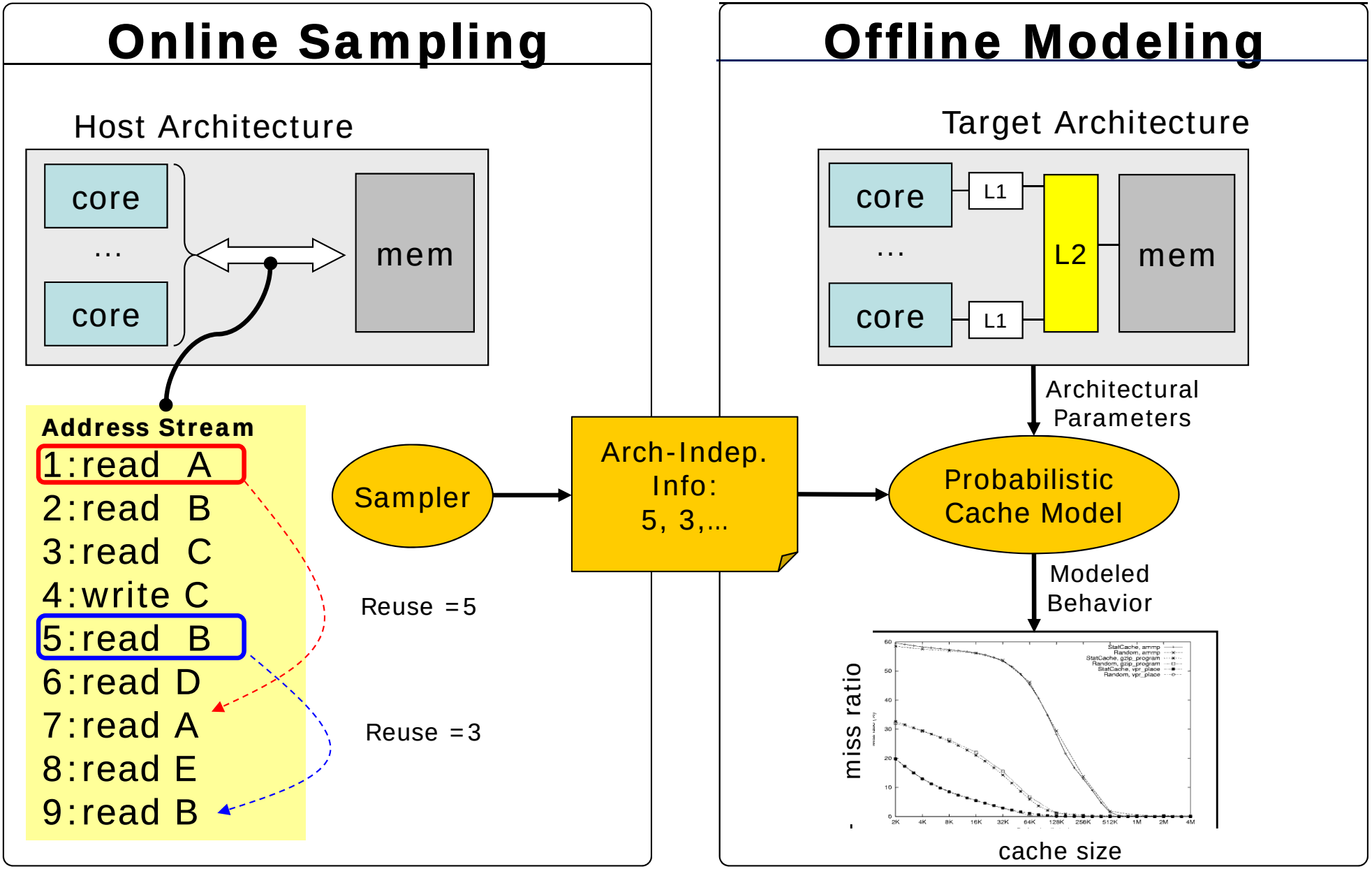
Architecture Research at Uppsala: Overall approach for modeling





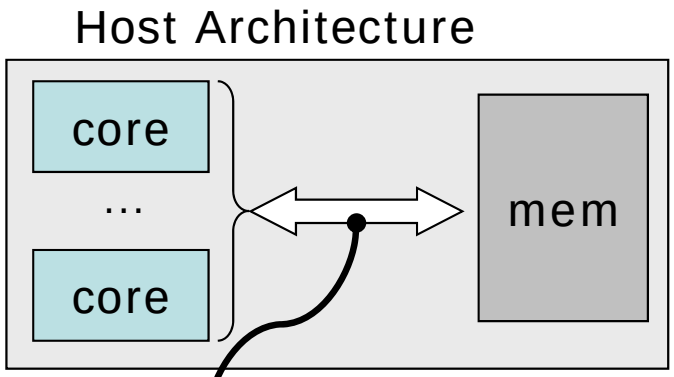
HERE: "SIMULATORS" ON STEROIDS FOR UNDERSTANDING APPLICATION BEHAVIOUR

1. Example: Fast Cache Modeling

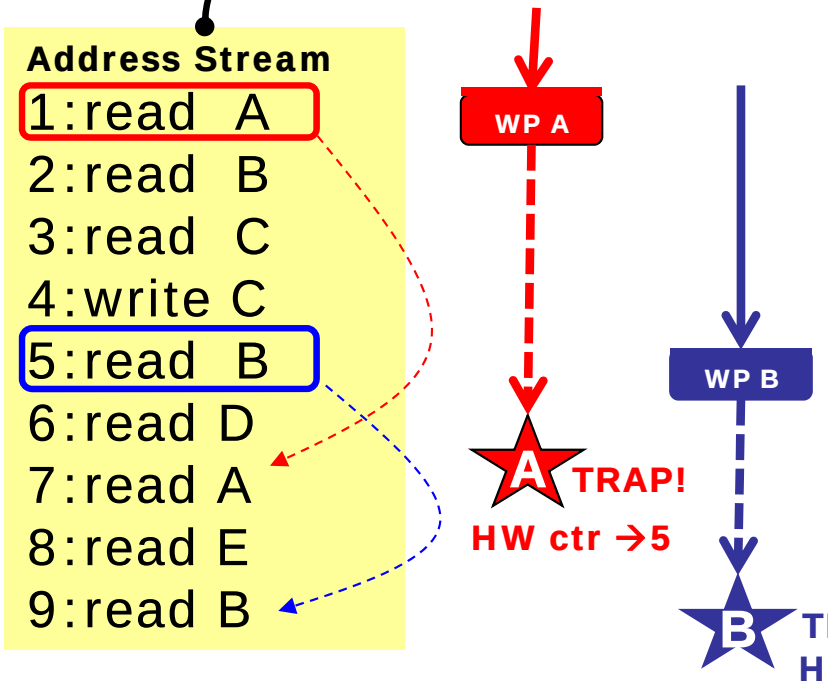


Efficient Reuse Sampling

Online Sampling



Typically, 50.000 samples per applicaiton
Typical sample rate 1:100.000
Typical overhead: 20% over native execution



Init HW ctr to overflow on next instruction to sample

Set watchpoint for the data address it accesses

Resume execution

The execution traps on next access to that address

HW ctrs has been counting intervening mem ops

Modeling caches (random replacement)

P_{hit} after one cache replacement: *(1 – 1/Size)*

$P_{hit}(R)$, after R cache replacements: $(1 - 1/Size)^R$

$P_{miss}(R) = 1 - P_{hit}(R)$: $(1 - (1 - 1/Size)^R)$

*Assume that we know the averga miss ration **M** for the application*

*$P_{miss}(A)$, after A acesses, assuming miss ratio **M**:* $(1 - (1 - 1/Size)^{MA})$

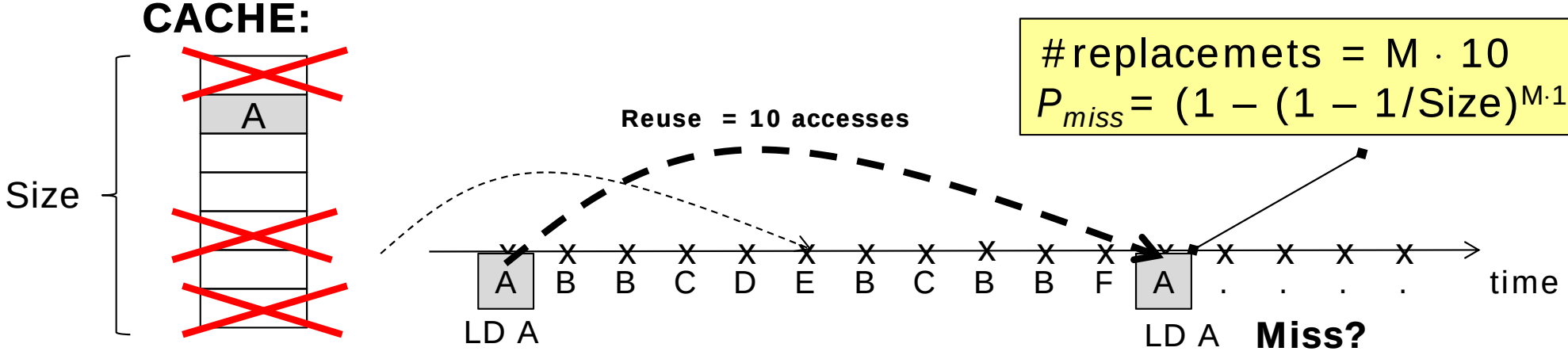
For N samples, each with a reuse r(i), total misses are:

$$\sum_{k=1}^{N} \left(1 - (1 - 1/Size)^{M\, r(k)}\right)$$

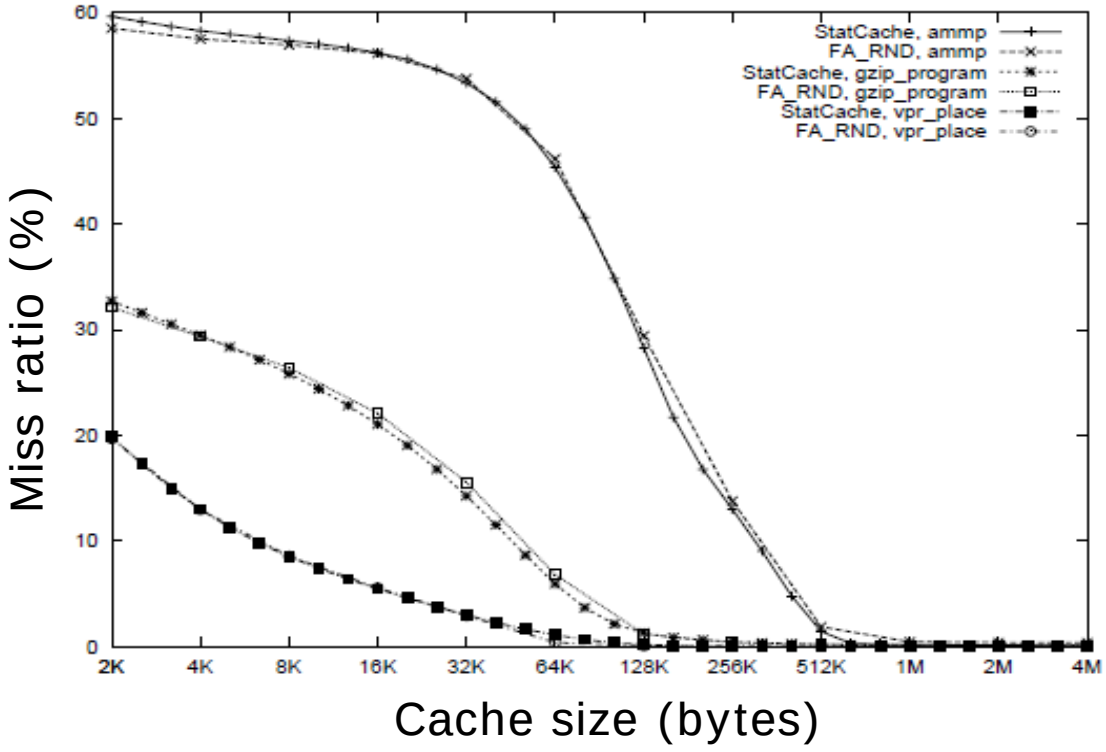
...which should be equal to N time miss ratio (M):

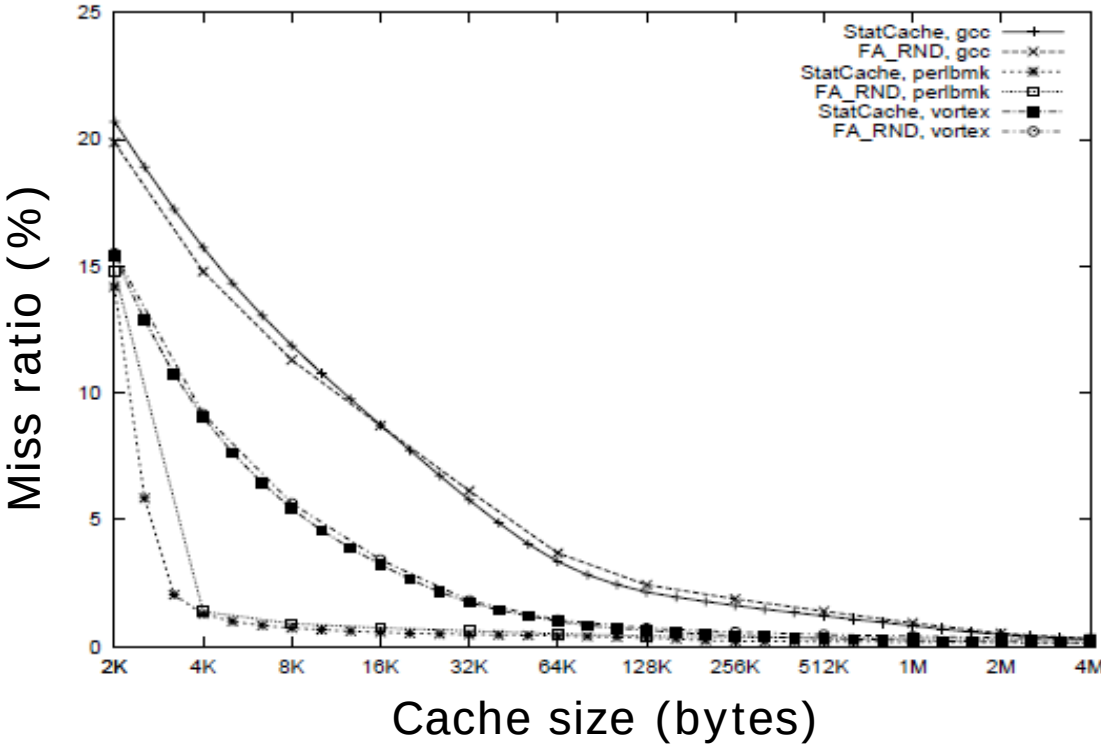
$$N\,M = \sum_{k=1}^{N} \left(1 - (1 - 1/Size)^{M\, r(k)}\right)$$

...where M can be solved numerically for any value of Size



Accuracy for modeling caches with random replacement





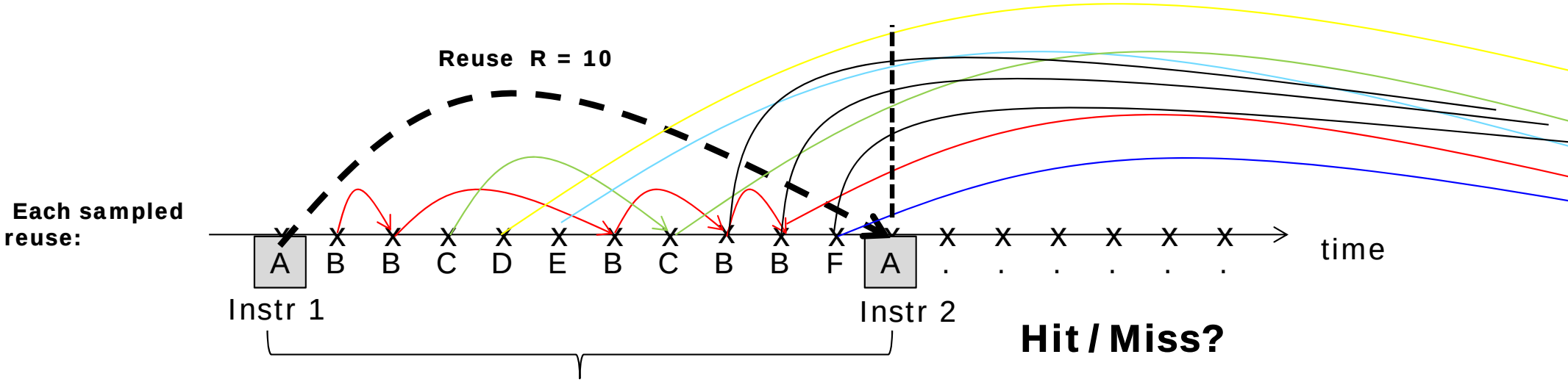
[Berg, ISPASS 2004], [Berg, SIGMETRICS 2005]

- Randomly halting instructions is a more tricky
- Application phase behavior requires ”windowing”

[Sembrant, CGO 2010]

- How to get down to 20% overhead

StatStack: Modeling LRU Caches



Stack distance: #Unique cache lines (=5) → **Hit for cache size > 5**

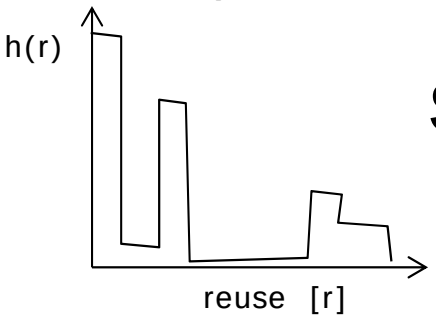
(…but capturing stack distances is expensive ☹)

If we knew all the other reuse distances: #Jump-outs = Stack distance

(…but we do not know all the reuses ☹)

Can be estimated using the reuses [from the vicinity]

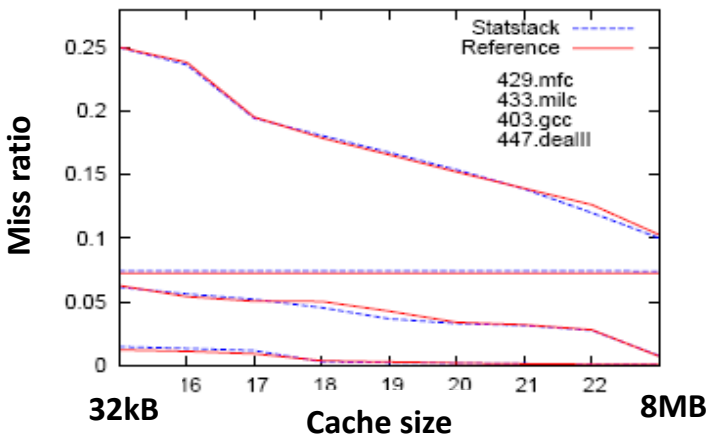
Reuse histogram for all samples

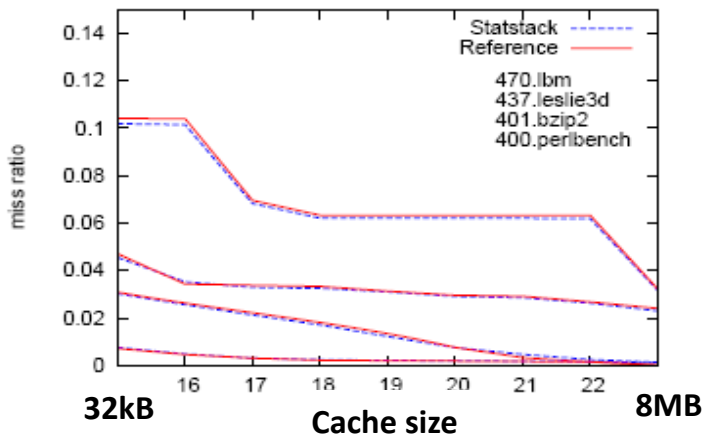


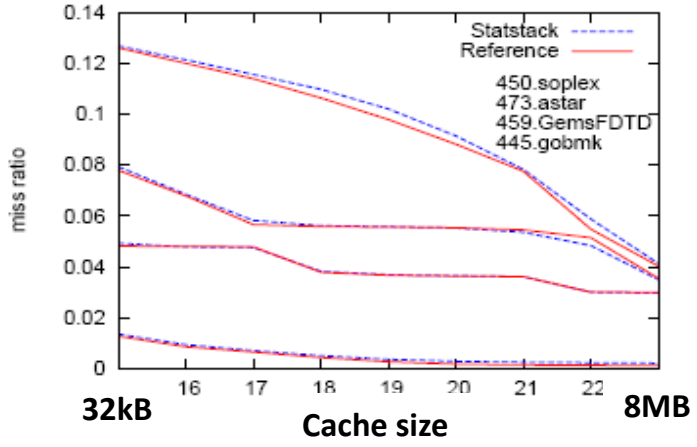
$$\textbf{Stack-distance}(R) = \sum_{i=1}^{R-1}\left(P(reuse > i) \right)$$

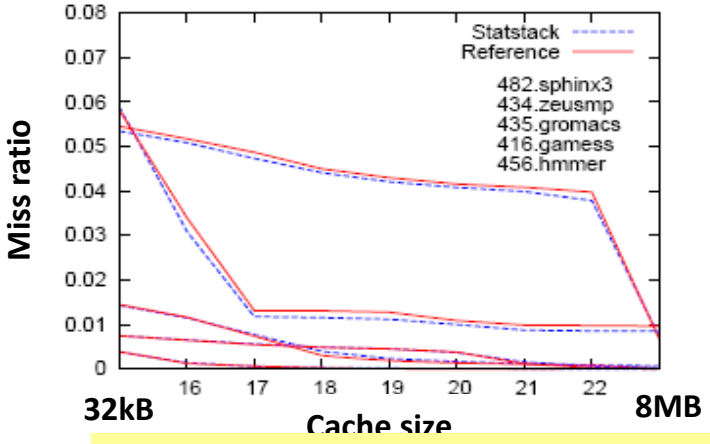
Hit if (cache size > Stack-distance(R))

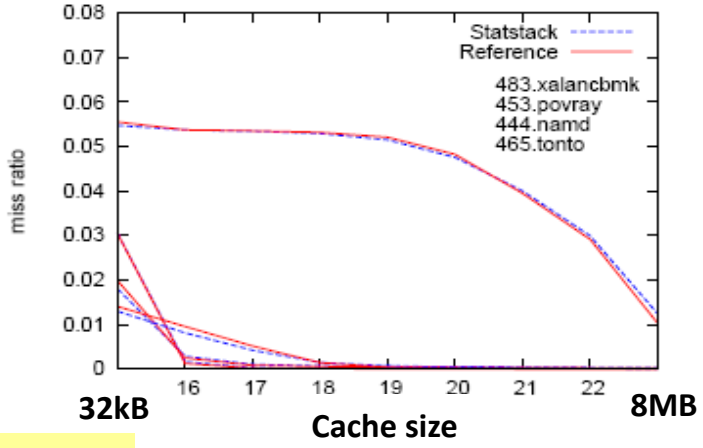
Accuracy for modeling LRU caches of all sizes

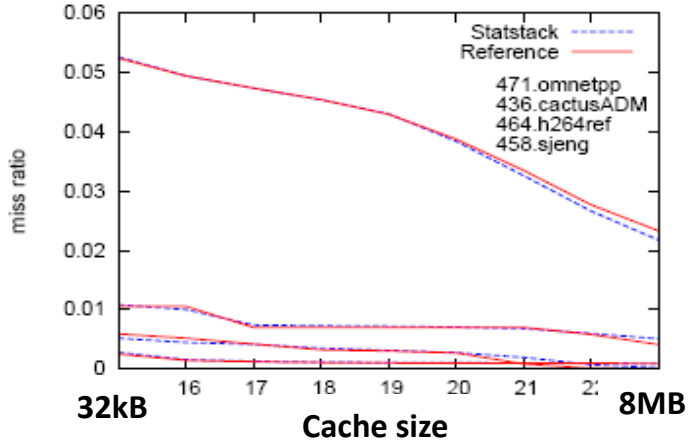






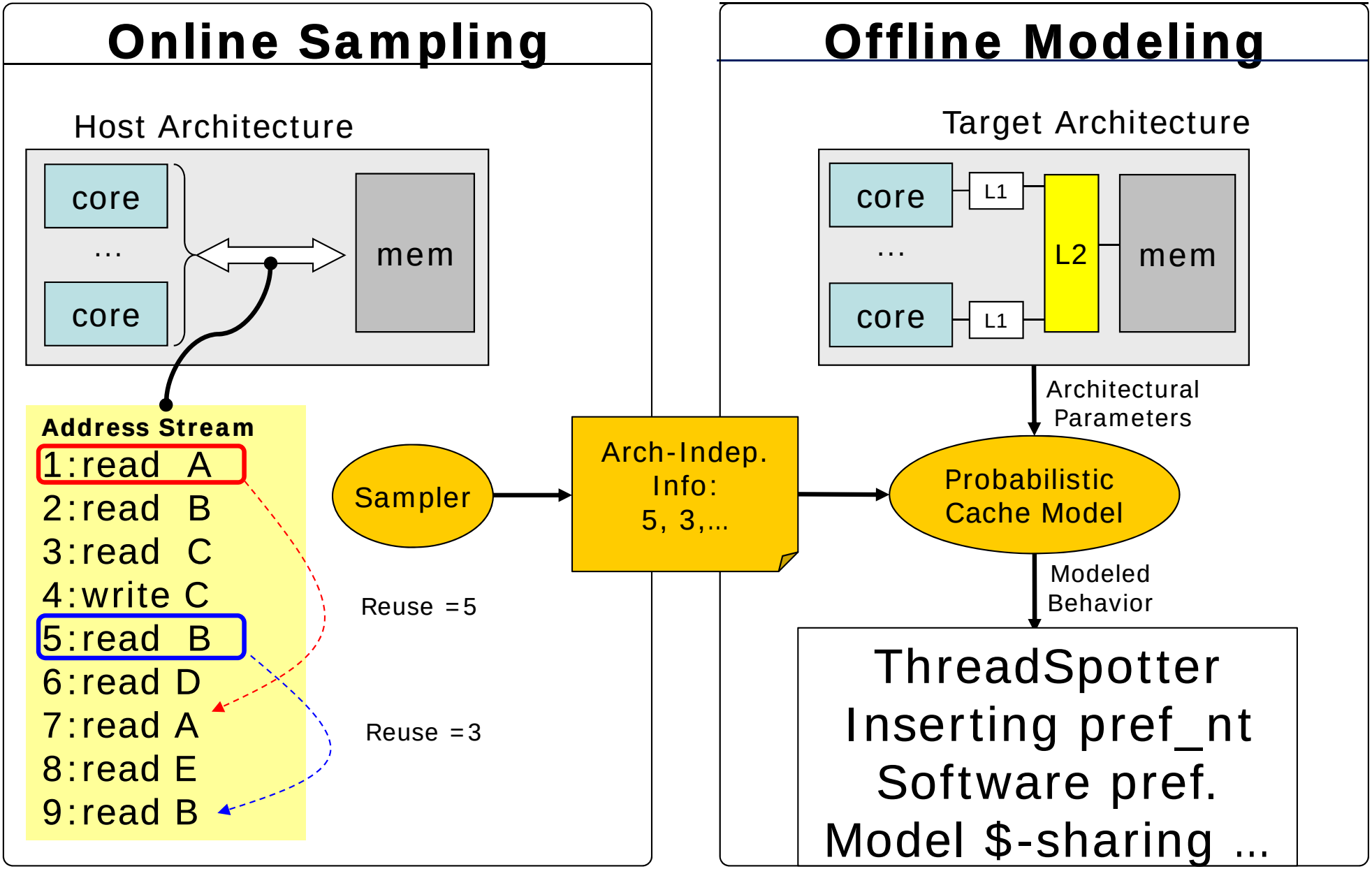






[Eklov, ISPASS 2008]

1. Example: Fast Cache Modeling



UPPSALA UNIVERSITET

UPMARC Uppsala Programming for Multicore Architectures Research Center

- Uppsala Programming for Multicore Architecture Center
- 62 MSEK grant / 10 years [$9M/10y]
  + related additional grants at UU = 130MSEK
- Research areas:

Erik:
- Performance modeling
- New parallel algorithms
- Scheduling of threads and resources
- Testing & verification
- Language technology
- MC in wireless and sensors

AVDARK 2013