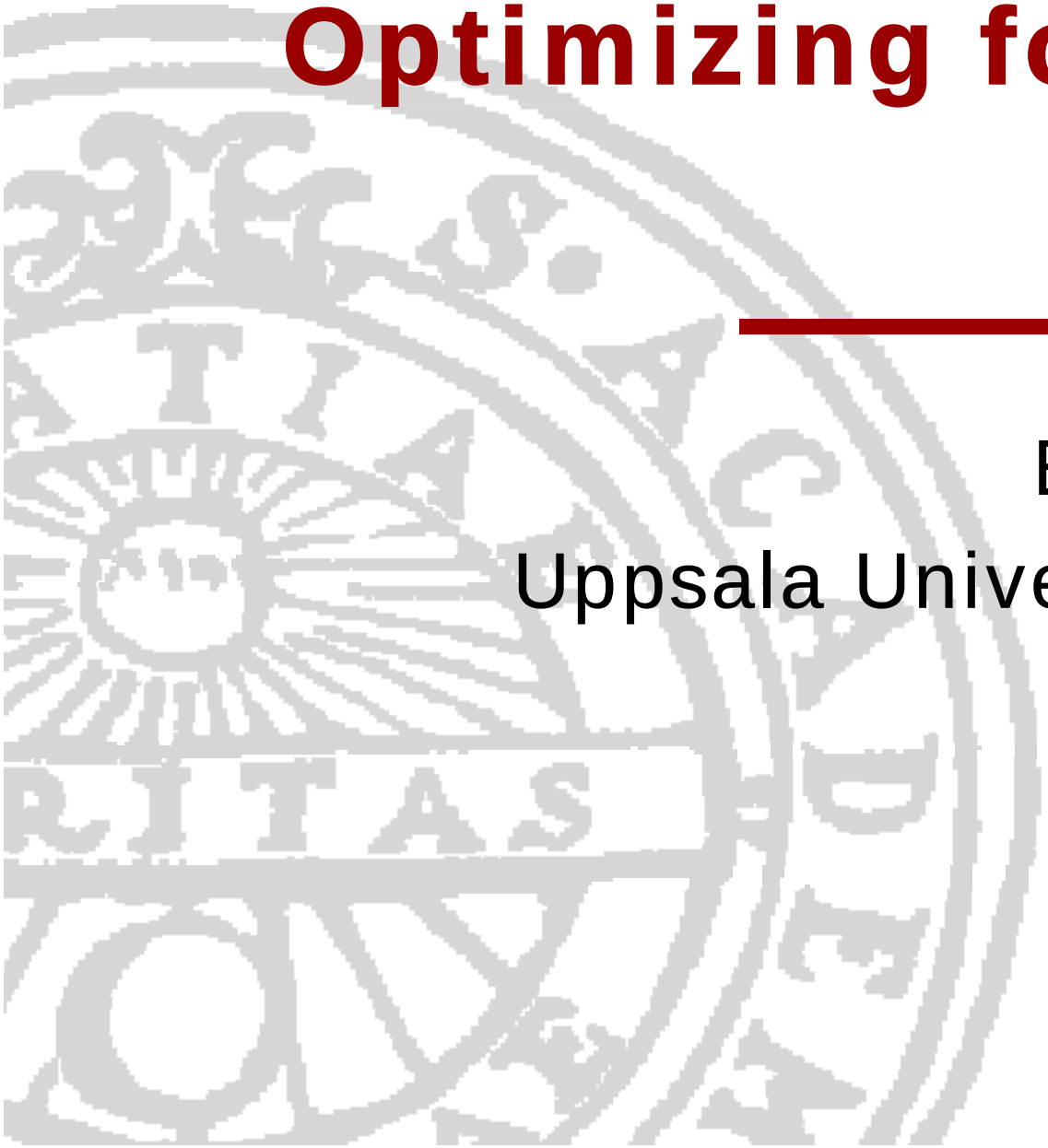


Optimizing for Speed on Multicores

Erik Hagersten

Uppsala University, Sweden

eh@it.uu.se

UPPSALA UNIVERSITET

What is the potential gain?

- Latency difference L1$ and mem: ~50x
- Bandwidth difference L1$ and mem: ~20x
- Repeated TLB misses adds a factor ~2-3x
- Execute from L1$ instead from mem ==> 50-150x improvement
- At least a factor 2-4x is within reach

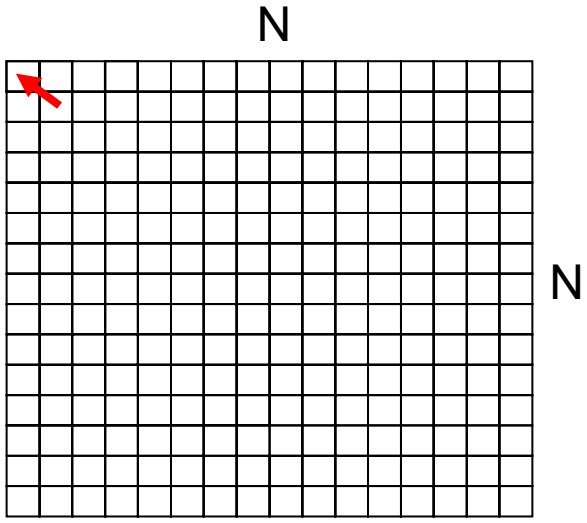
UPPSALA UNIVERSITET

Optimizing for cache 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
- Let the CPU that already has the data in its cache do the job
- …

What can go Wrong? A Simple Example...

Perform a diagonal copy 10 times

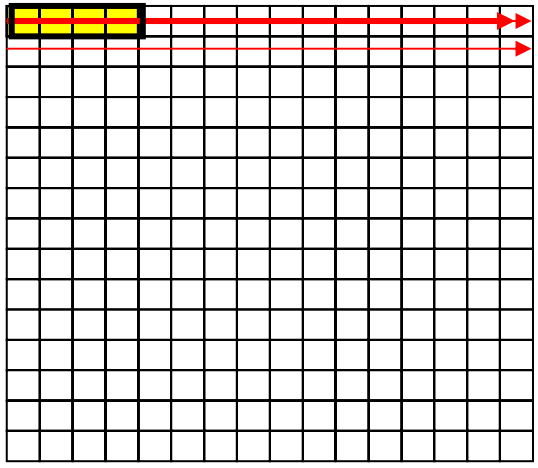


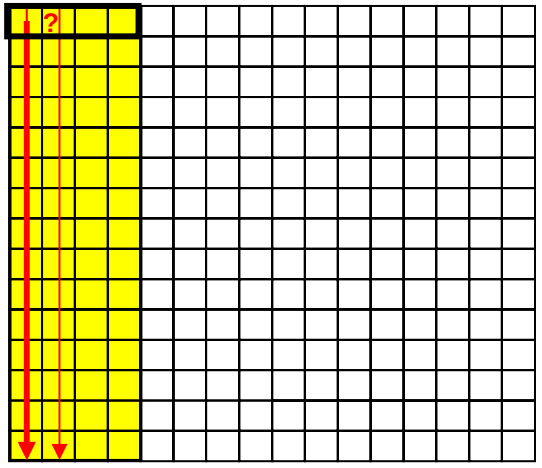
UPPSALA UNIVERSITET

Example: Loop order

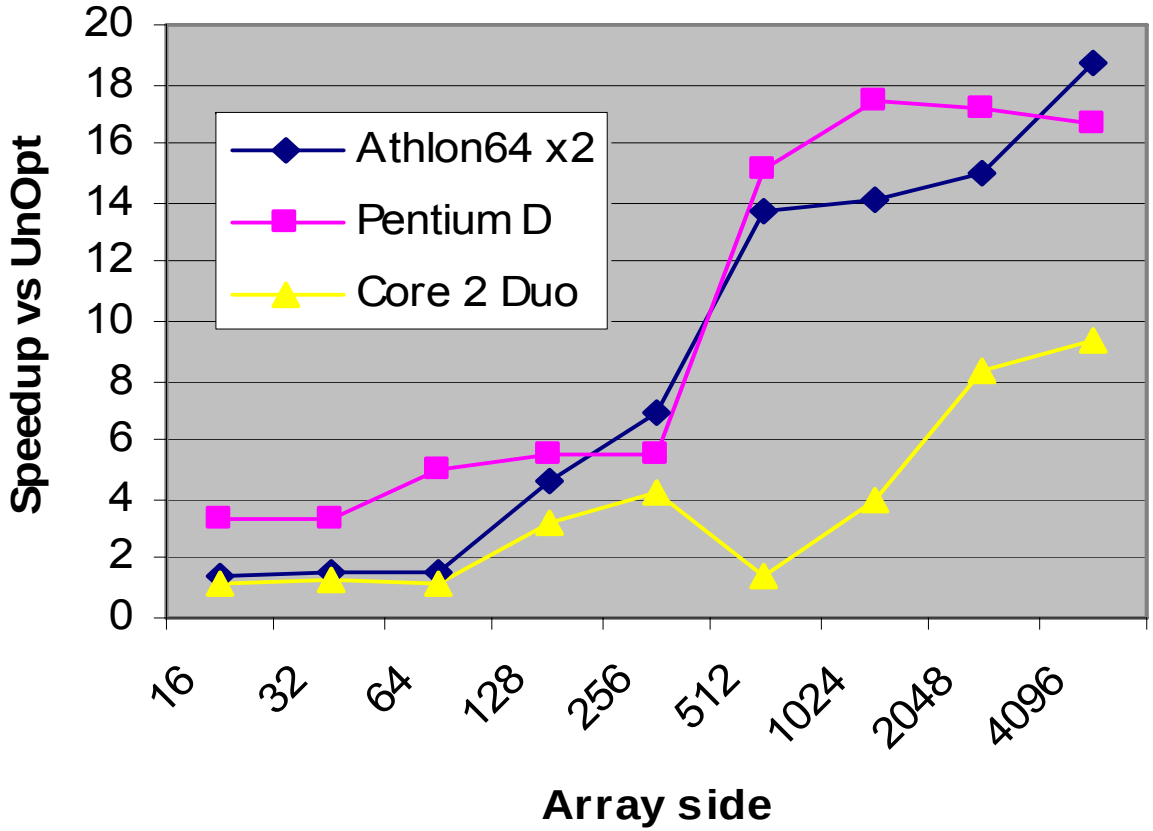
```
//Optimized Example A
for (i=1; i<N; i++) {
  for (j=1; j<N; j++) {
    A[i][j]= A[i-1][j-1];
  }
}
```

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



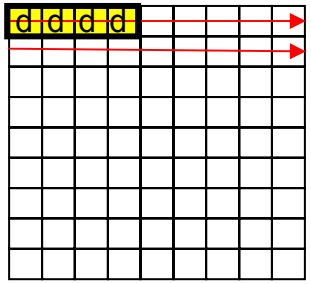


Performance Difference: Loop order

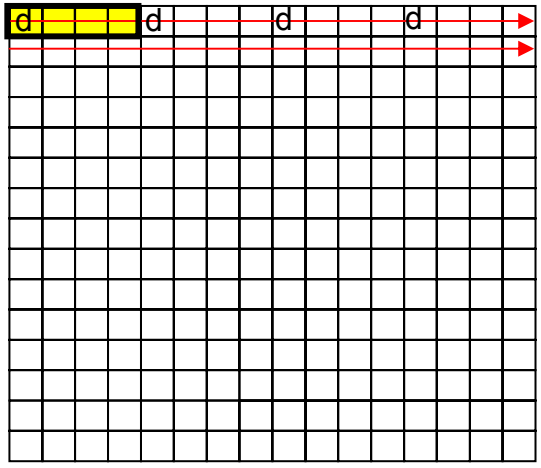


Example: Sparse data

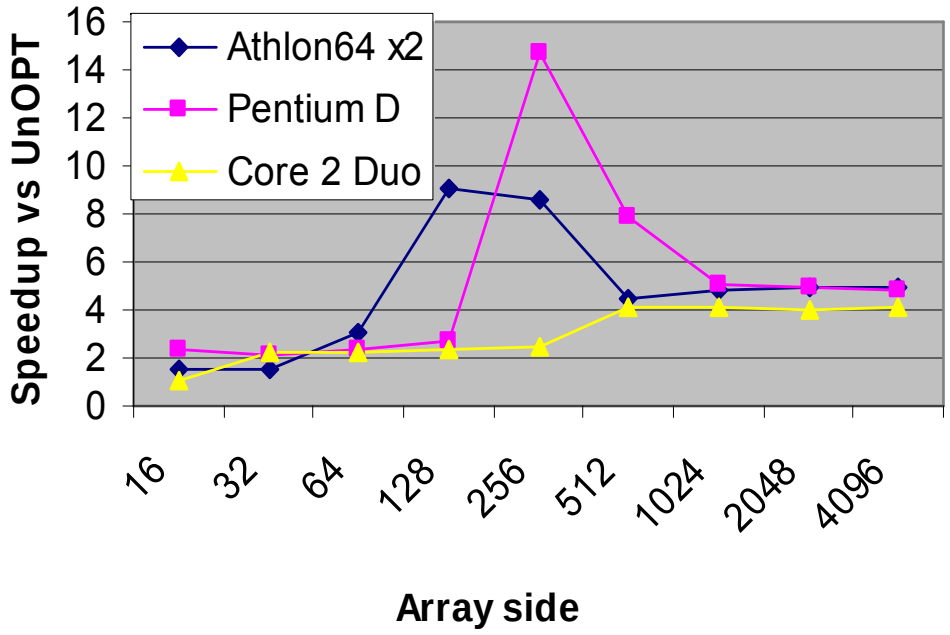
```
//Optimized Example A
for (i=1; i<N; i++) {
  for (j=1; j<N; j++) {
    A_data[i][j]= A_data[i-1][j-1];
  }
}
```




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



Performance Difference: Sparse Data



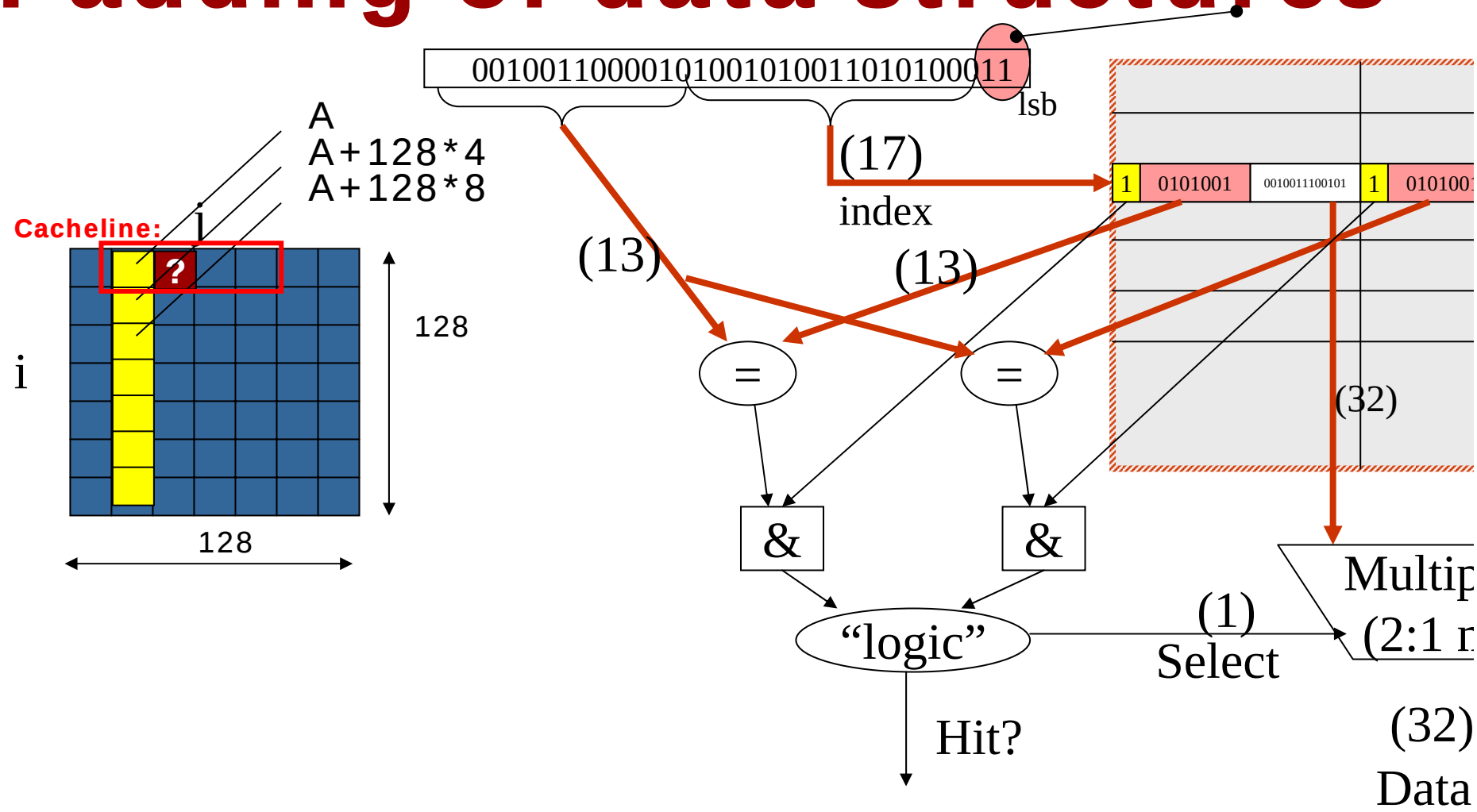
UPPSALA UNIVERSITET

Loop Merging

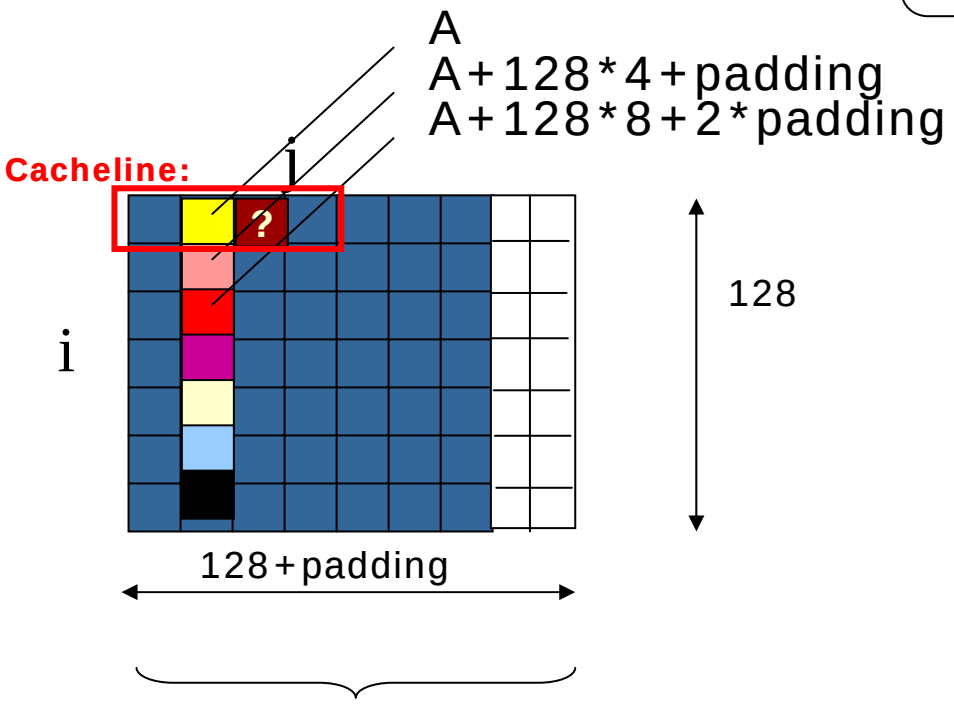
```
/* Unoptimized */
for (i = 0; i < N; i = i + 1)
   for (j = 0; j < N; j = j + 1)
        a[i][j] = 2 * b[i][j];
for (i = 0; i < N; i = i + 1)
   for (j = 0; j < N; j = j + 1)
        c[i][j] = K * b[i][j] + d[i][j]/2

/* 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;
```

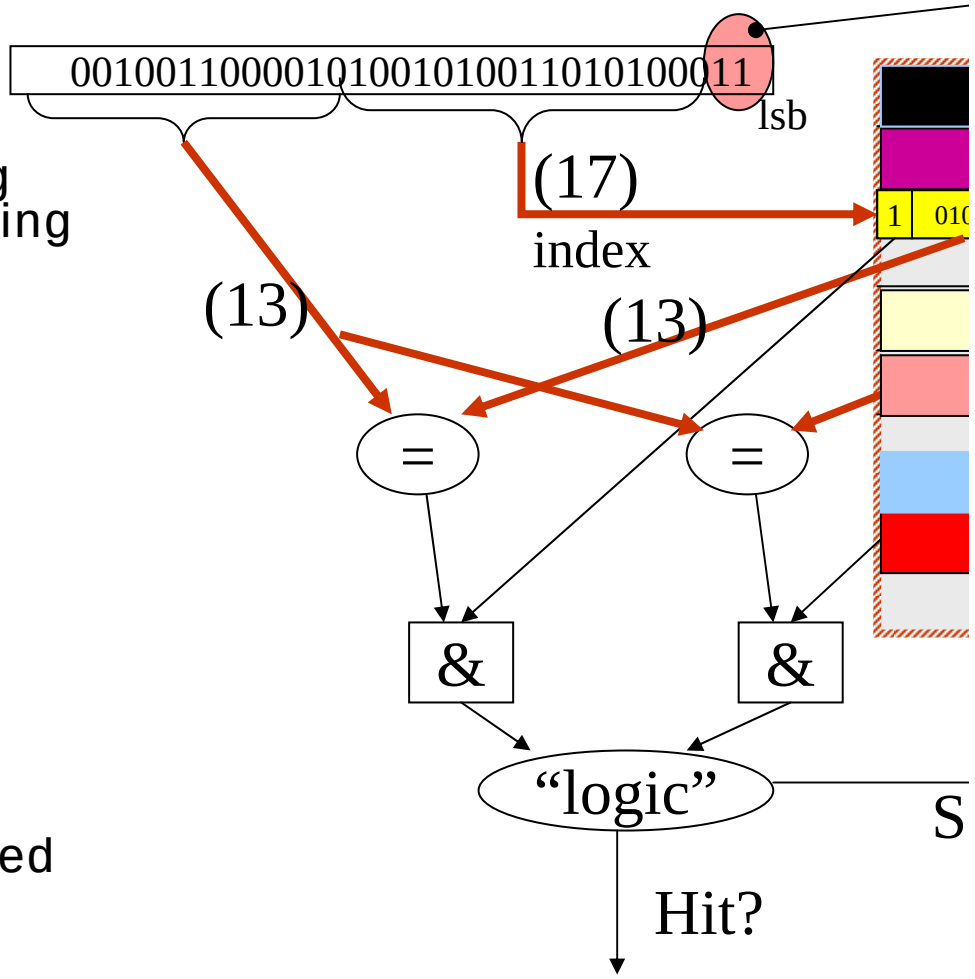
Padding of data structures



Padding of data structures



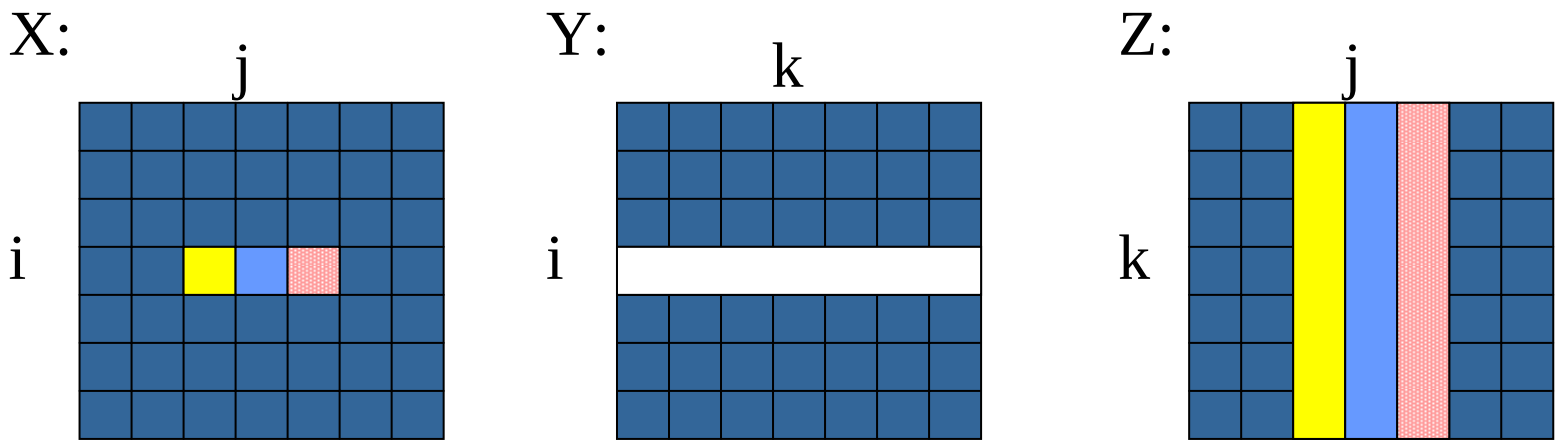
allocate more memory than needed



UPPSALA UNIVERSITET

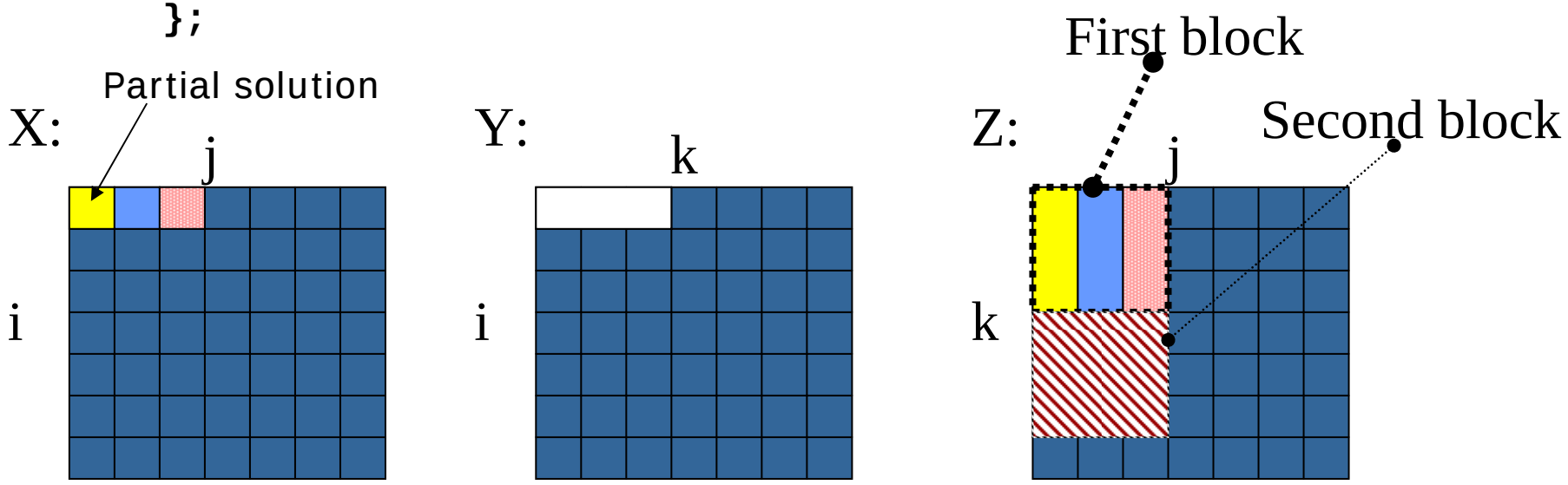
Blocking

```
/* 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;
        };
```



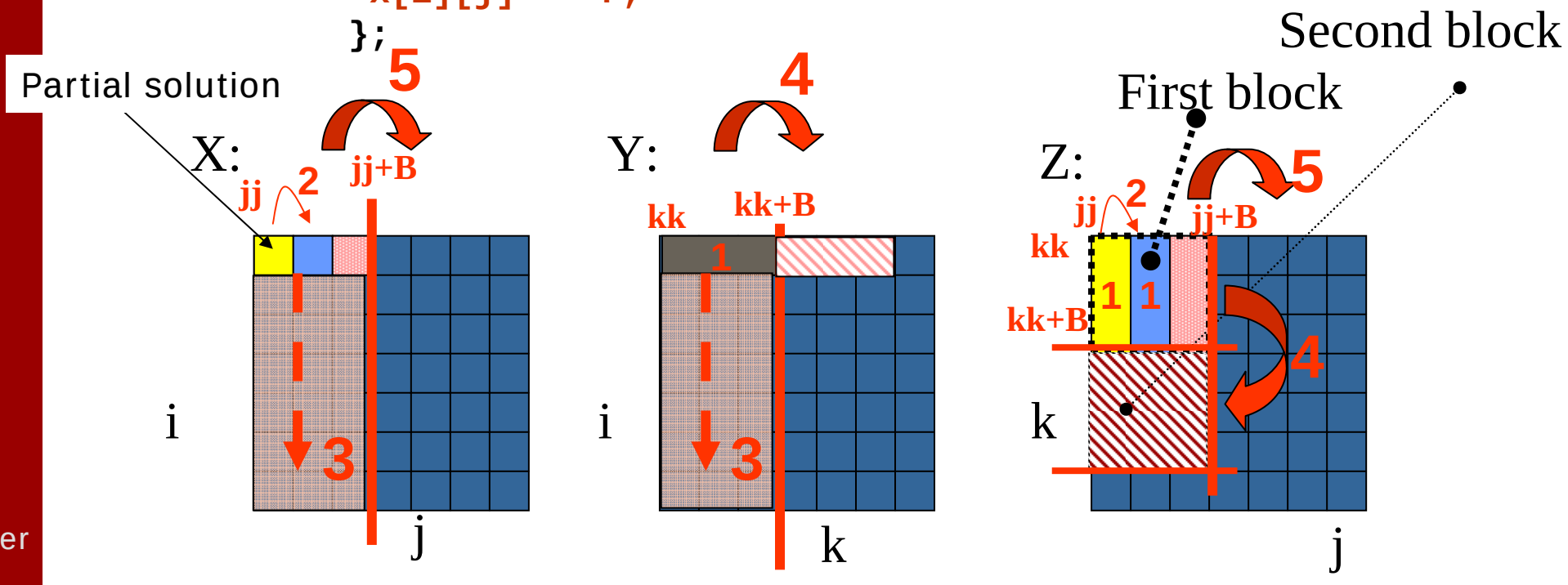
Blocking

```
/* 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;
        };
```



OPT 14

UPPSALA UNIVERSITET

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)

UPPSALA UNIVERSITET

Cache Affinity

- Schedule the process on the processor it last ran
- Allocate and free data buffers in a LIFO order

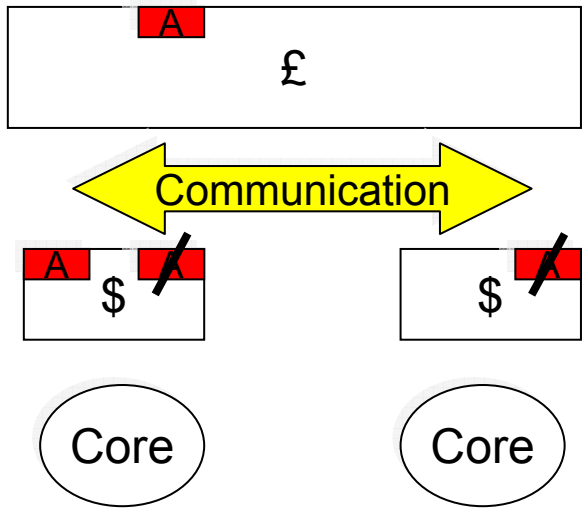
UPPSALA UNIVERSITET

Optimize for ”other caches”

- TLB
  - Avoid random accesses to huge data structs (Ex. Huge hashing table)
  - Avoid few access per page (very sparse data)

...

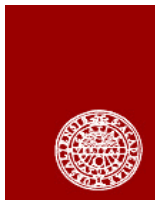
Communication



A:= 1

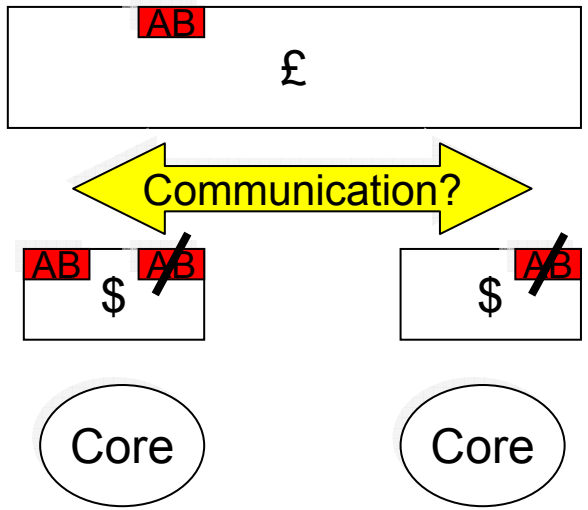
A := 2

A:= 3



UPPSALA UNIVERSITET

False sharing



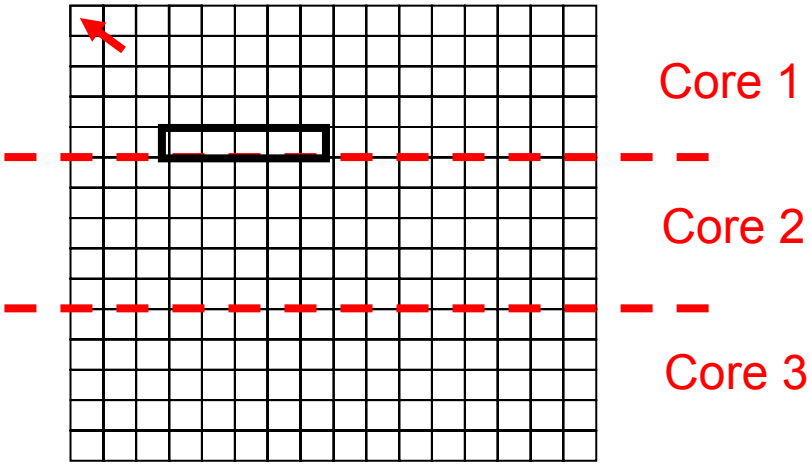
A:= 1

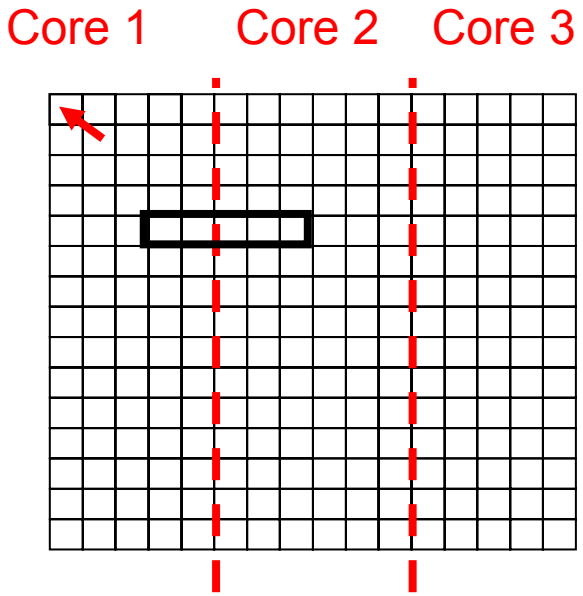
B := 1

A:= 2

Slicing the work

= cache line





Commercial Break: Acumem's Multicore Tools

Erik Hagersten

Uppsala University, Sweden

eh@it.uu.se

UPPSALA UNIVERSITET

Acumem SlowSpotter™

Source:

C, C++, Fortran, OpenMP…

```
/* Unoptimized  Array Multiplication: x = y * z    N = 1024 */
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;
      }
/* Unoptimized  Array Multiplication: x = y * z    N = 1024 */
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;
      }
```

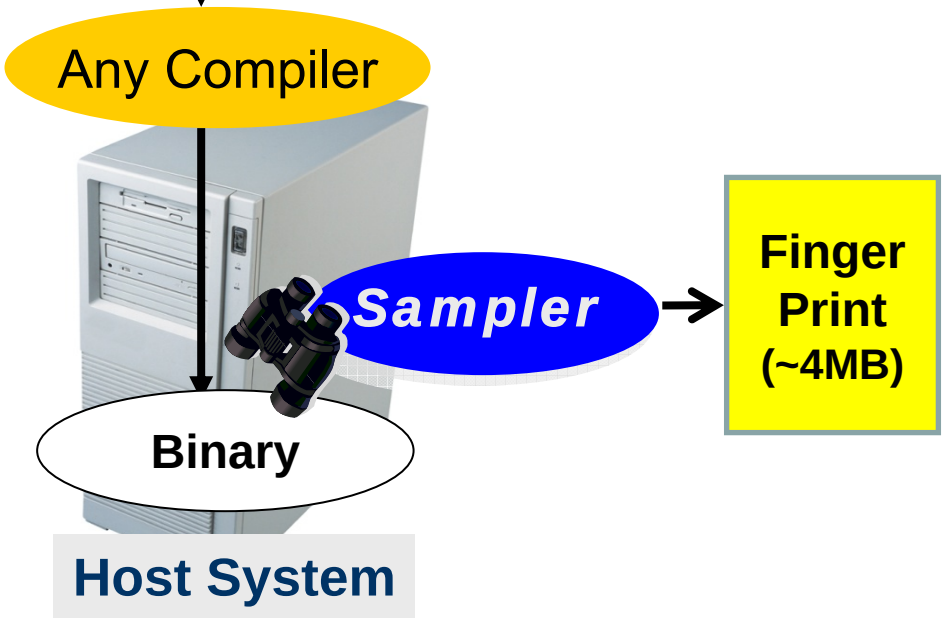
Mission:

Find the SlowSpots™

Asses their importance

Enable for non-experts to fix them

Improve the productivity of performance experts



PDC Summer School 2010

UPPSALA UNIVERSITET

Acume

Source:

C, C++,

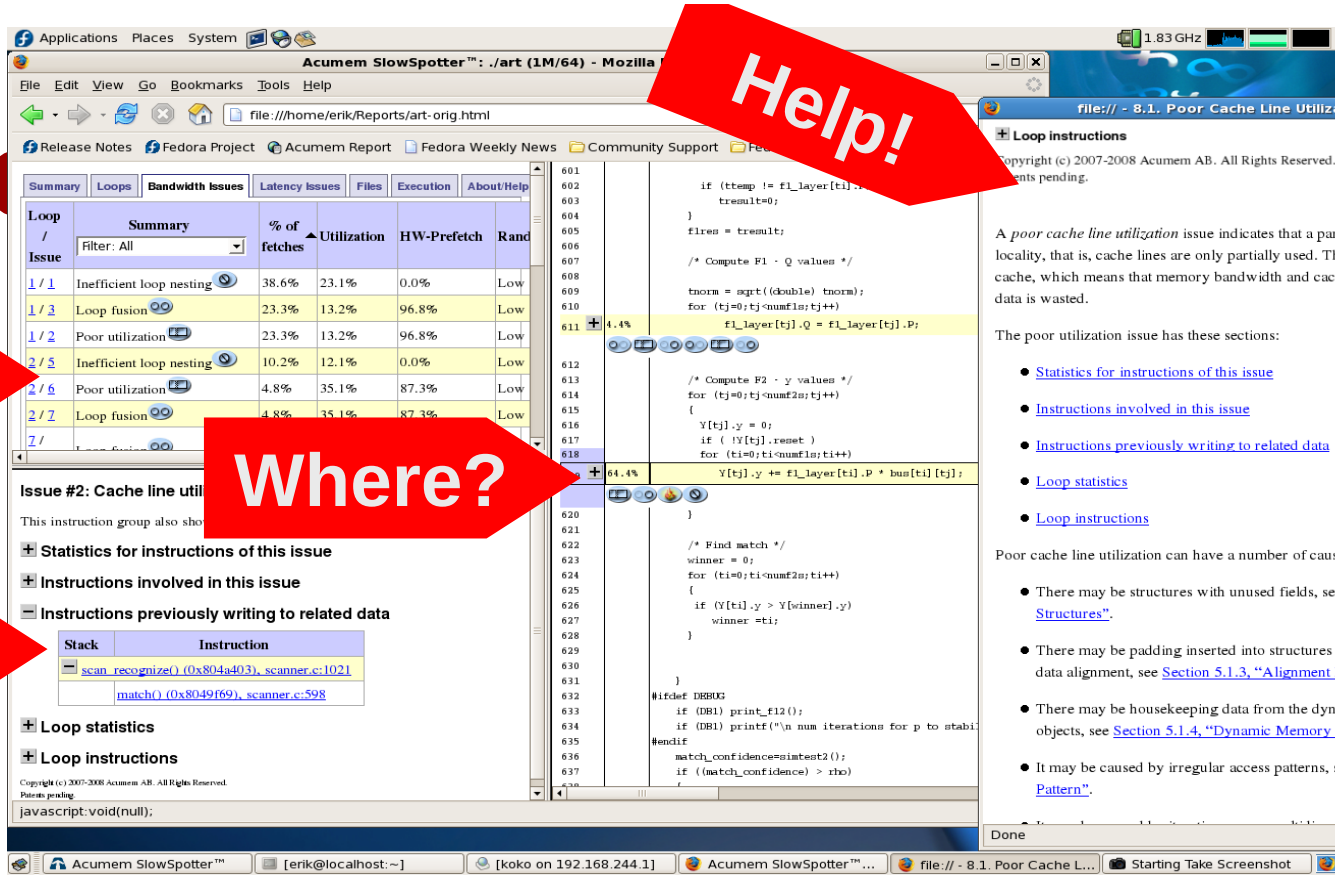
```
/* Unoptimized  Array Multiplica
for (i = 0; i < N; i = i + 1)
   for (j = 0; j < N; j = j + 1)
      {r = 0;
      for (k = 0; k < N; k = k + 1)
         r =  r + y[i][k] * z[k][j];
      x[i][j] = r;
      }
/* Unoptimized  Array Multiplication: x = y * z    N = 1024 */
for (i = 0; i < N; i = i + 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;
      }
```

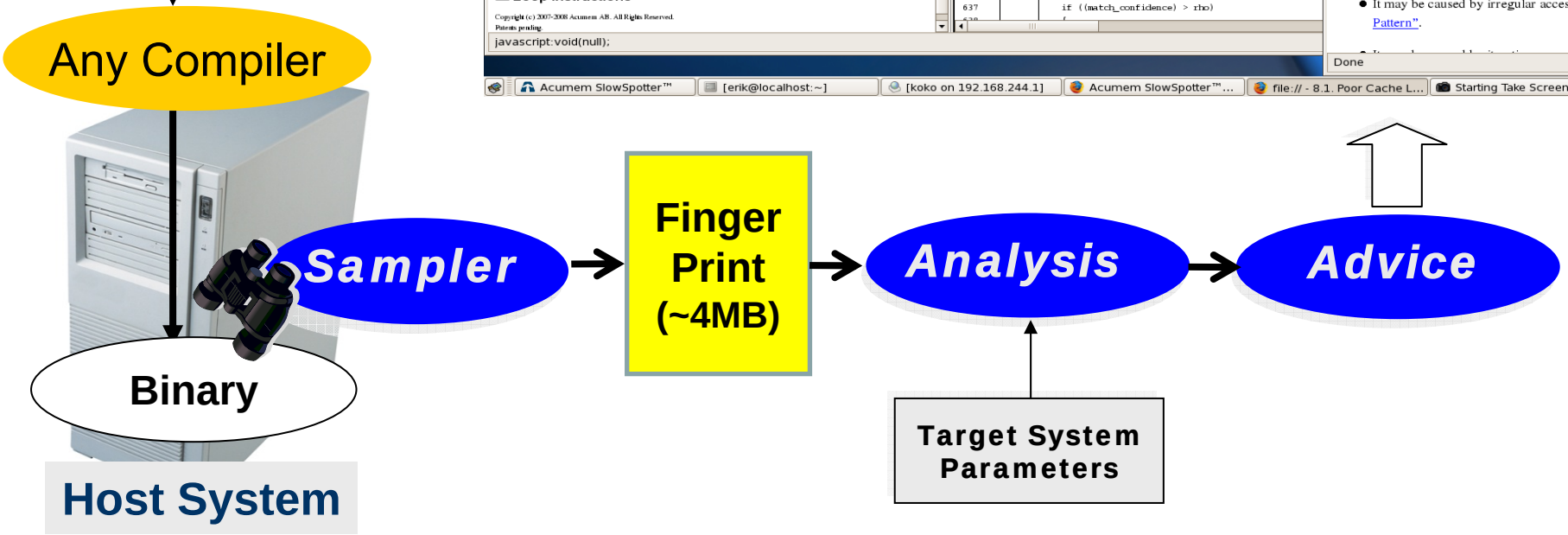
What?

How?

Where?

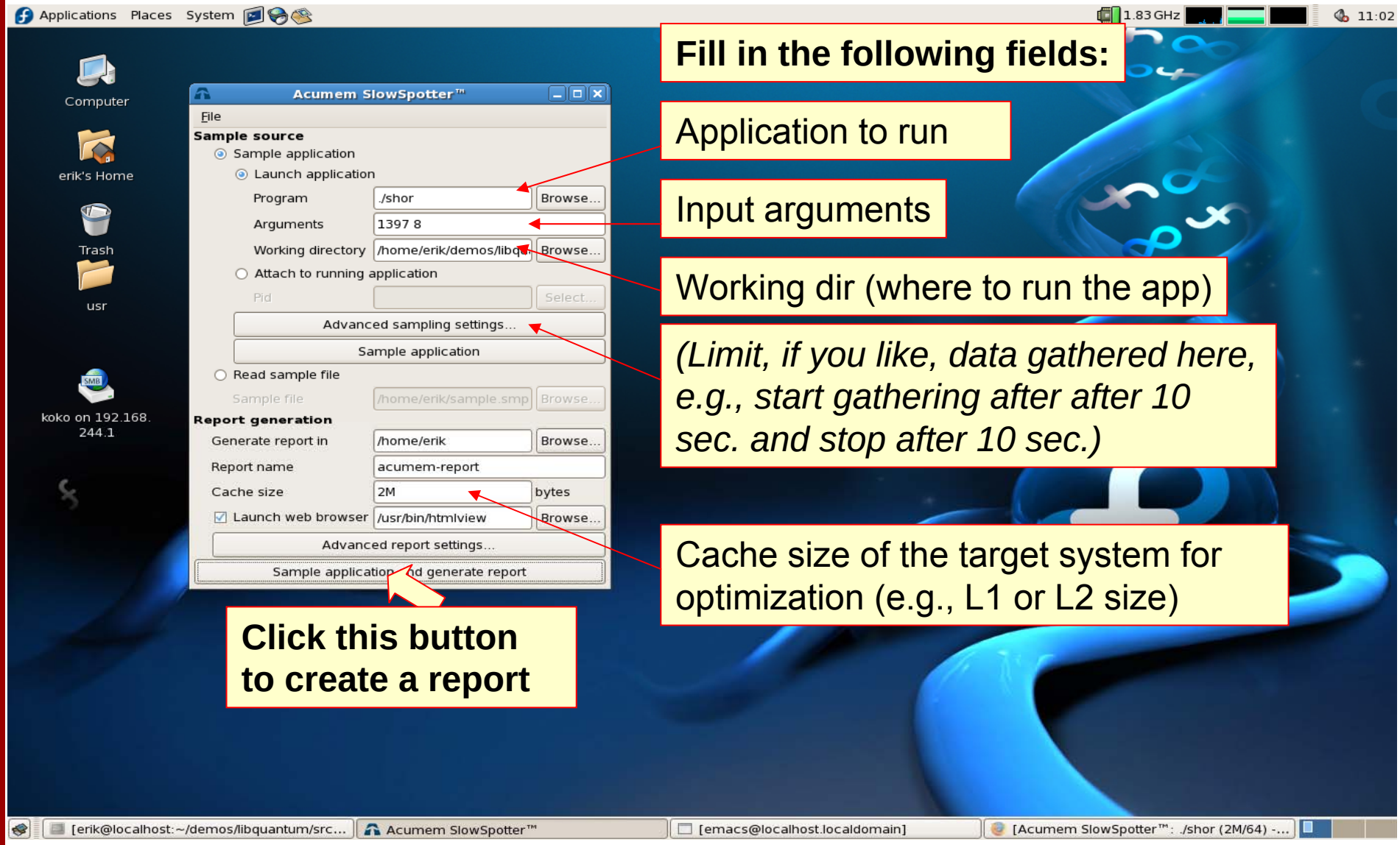
Help!





PDC Summer School 2010

A One-Click Report Generation

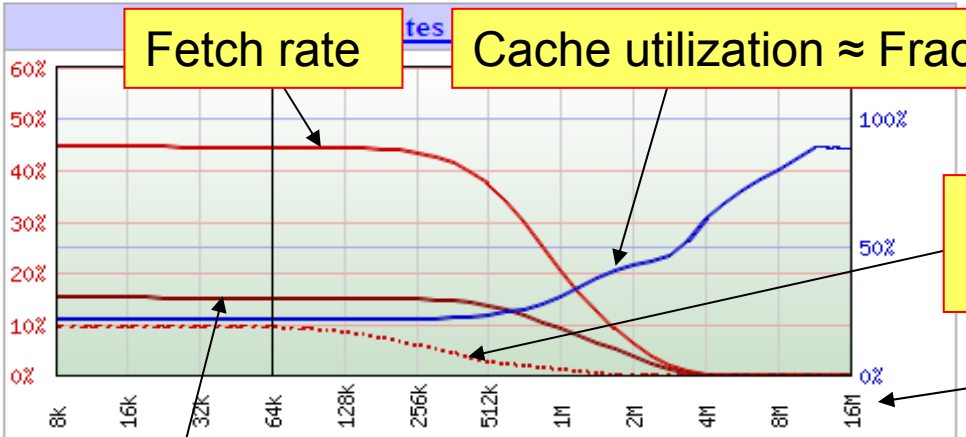


Acumem VPE: ./art (64k/64) - Mozilla Firefox

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 rate

Cache utilization ≈ Fraction of cache data utilized

Predicted fetch rate
(if utilization → 100%)

Cache size

Miss rate

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

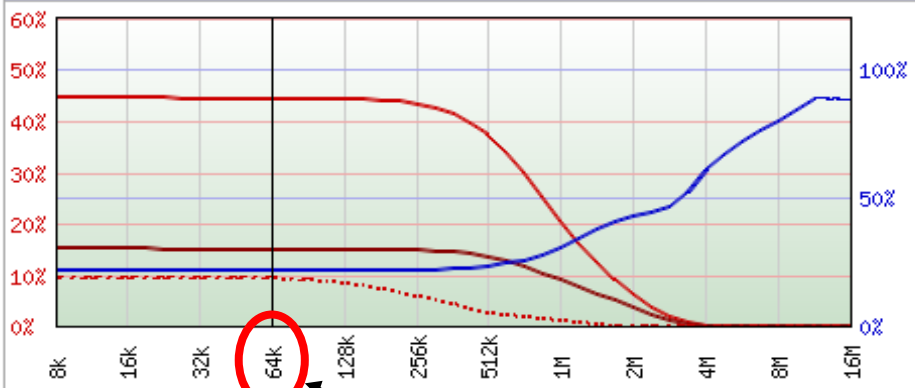
Acumem VPE: ./art (64k/64) - Mozilla Firefox

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

Miss rates and utilization



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

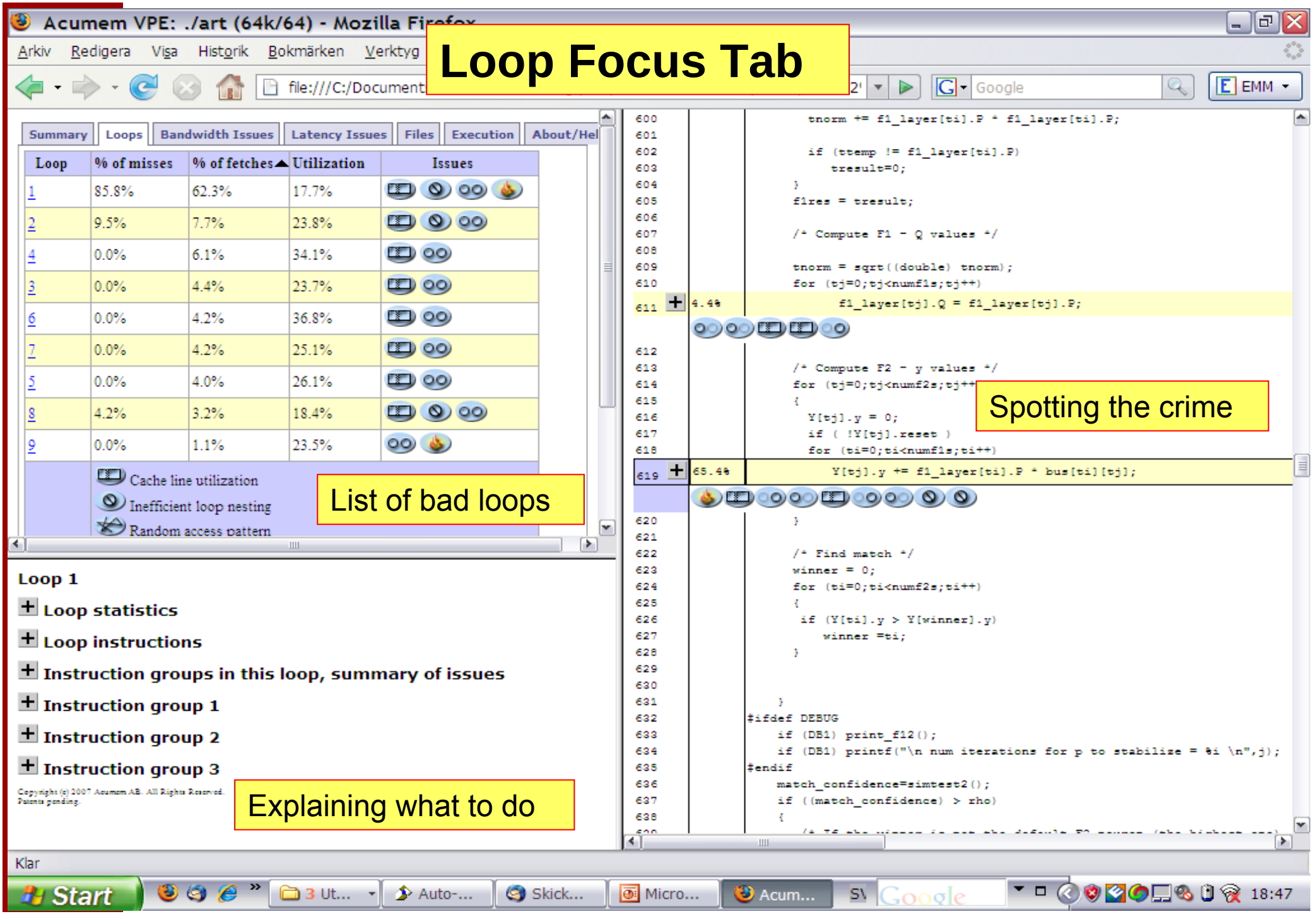
Miss rate	15.1%
Fetch rate	44.4%
Misses	5.45e+07
Fetches	1.60e+08
Cache line utilization	21.9%
Cache size	64k
Line size	64
Replacement policy	random

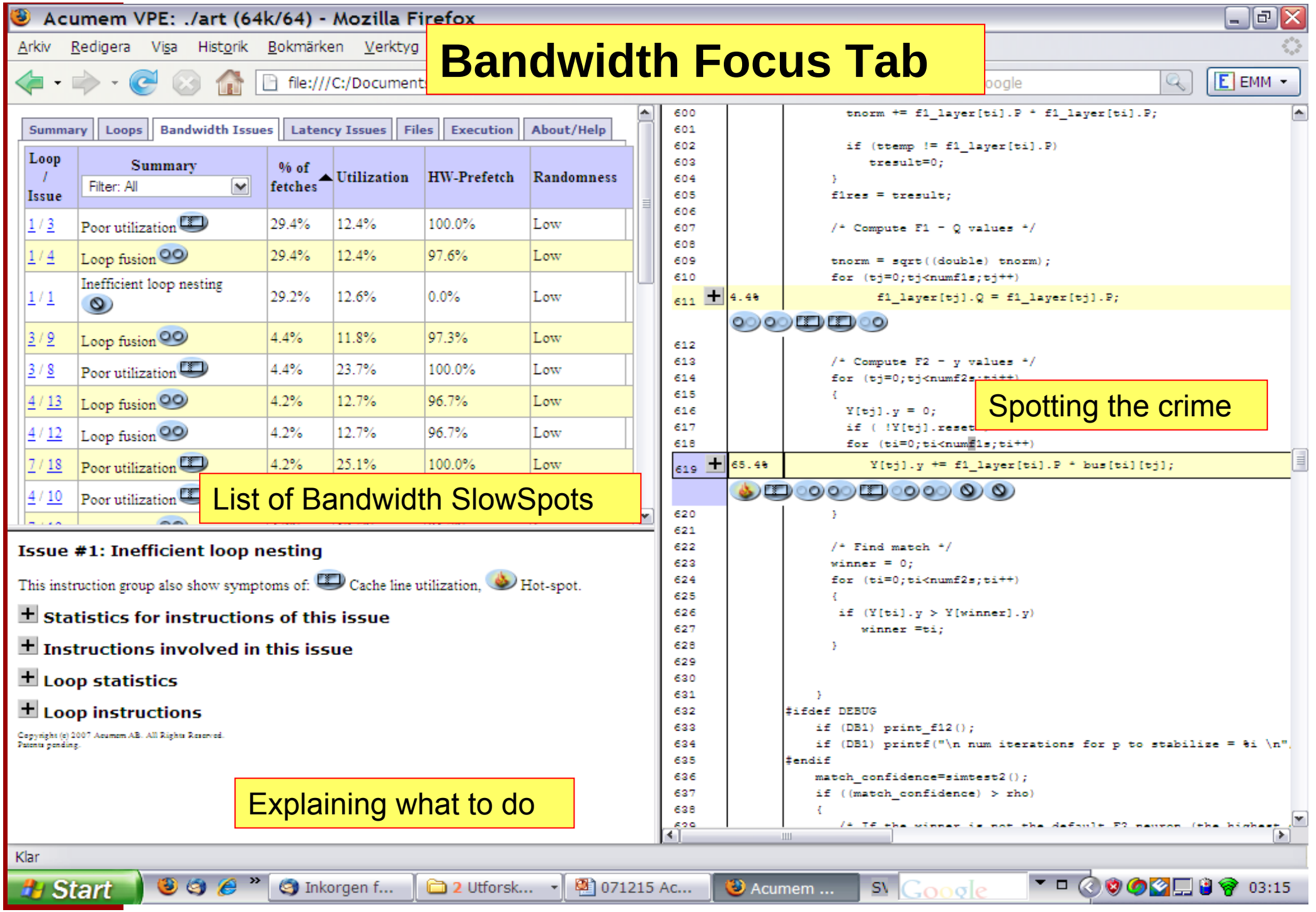
Cache size to optimize for

Select a file in the file table, or follow a source code link from an issue or a loop description.

Klar

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


Acumem VPE: ./art (64k/64) - Mozilla Firefox
Loop Focus Tab
Summary
Loops
Bandwidth Issues
Latency Issues
Files
Execution
Loop
% of misses
% of fetches
Utilization
Issues
1 85.8% 62.3% 17.7%
2 9.5% 7.7% 23.8%
4 0.0% 6.1% 34.1%
3 0.0% 4.4% 23.7%
6 0.0% 4.2% 36.8%
7 0.0% 4.2% 25.1%
5 0.0% 4.0% 26.1%
8 4.2% 3.2% 18.4%
9 0.0% 1.1% 23.5%
Cache line utilization
Inefficient loop nesting
Random access pattern
List of bad loops
Spotting the crime
Loop 1
Loop statistics
Loop instructions
Instruction groups in this loop, summary of issues
Instruction group 1
Instruction group 2
Instruction group 3
Explaining what to do
Klar

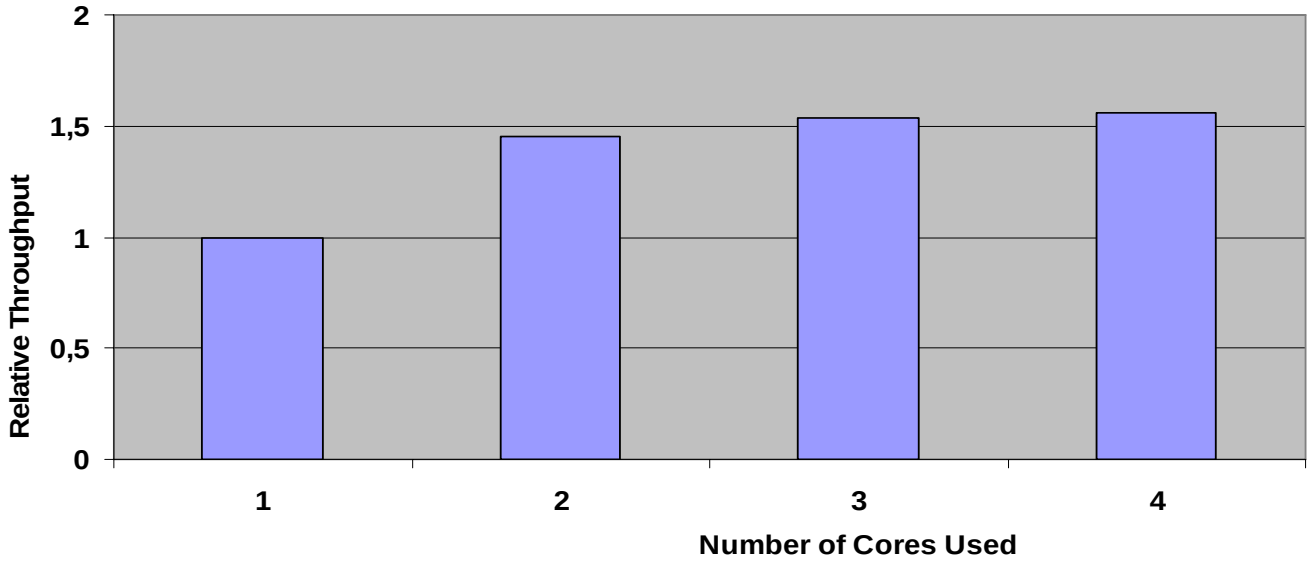

Acumem VPE: ./art (64k/64) - Mozilla Firefox
Arkiv Redigera Visa Historik Bokmärken Verktyg
Bandwidth Focus Tab
Summary Loops Bandwidth Issues Latency Issues Files Execution About/Help
Loop / Issue
Summary
Filter: All
% of fetches
Utilization
HW-Prefetch
Randomness
1 / 3 Poor utilization 29.4% 12.4% 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
List of Bandwidth SlowSpots
Spotting the crime
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
Klar
Start
Inkorgen f...
2 Utforsk...
071215 Ac...
Acumem ...
03:15

Resource Sharing Example

Libquantum

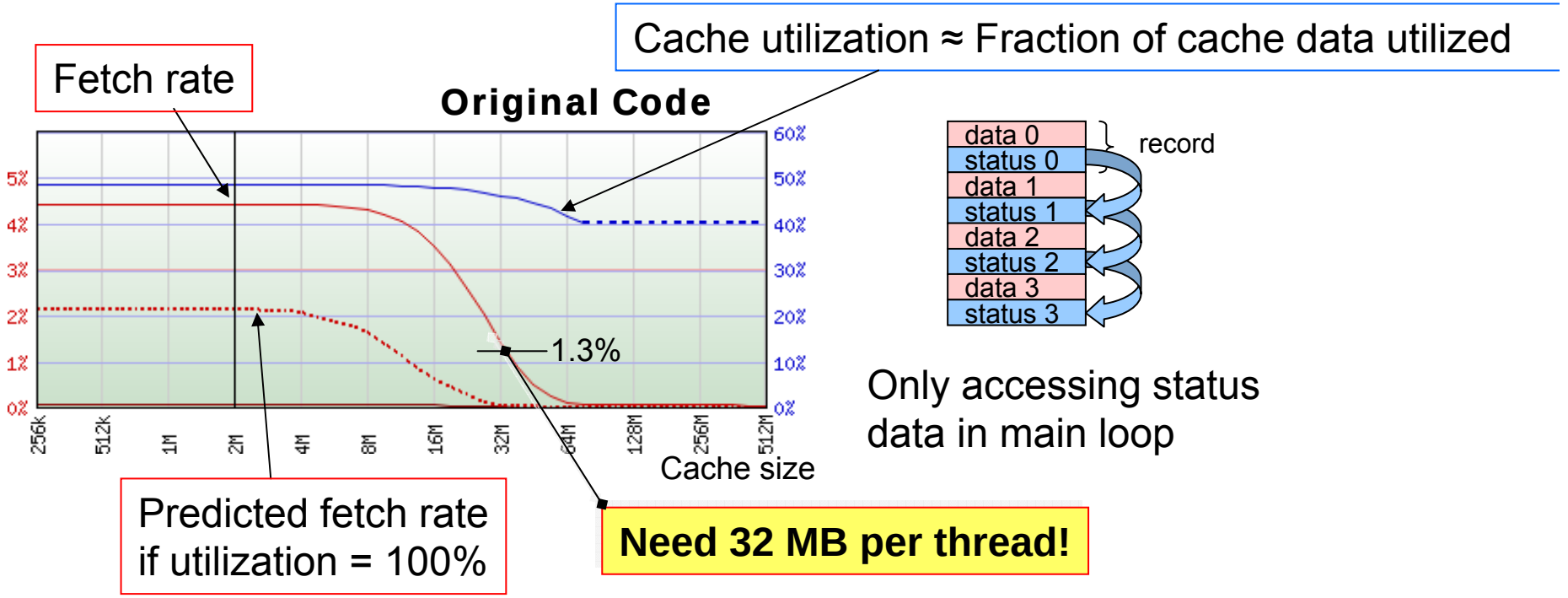
A quantum computer simulation
Widely used in research (download from: http://www.libquantum.de/)
4000+ lines of C, fairly complex code.
Runs an experiment in ~30 min

Throughput improvement:



Utilization Analysis

Libquantum



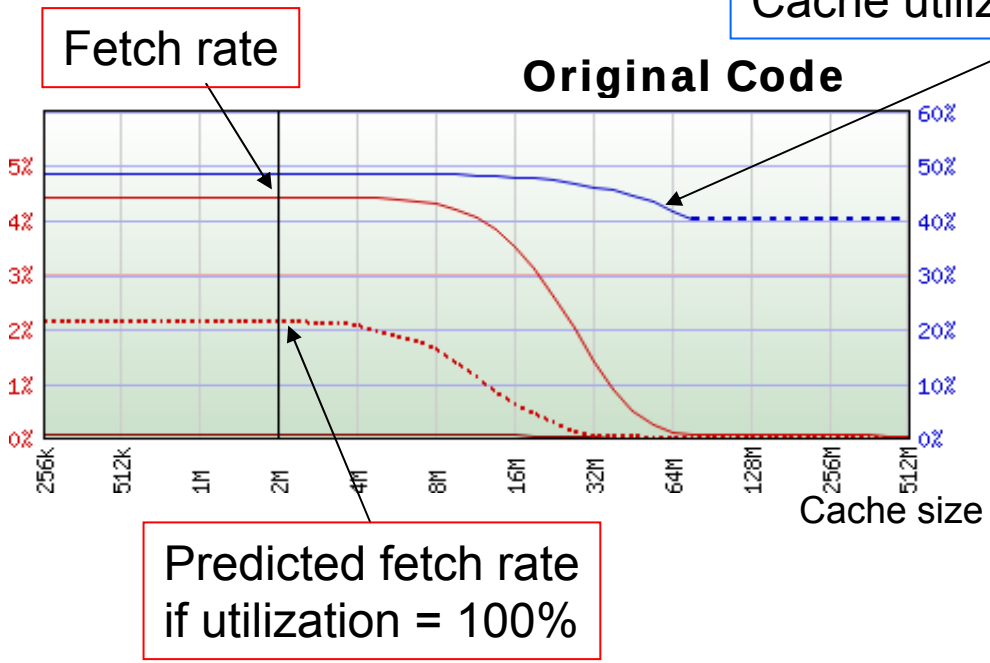
SlowSpotter's First Advice: Improve Utilization

➔Change one data structure

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

Utilization Analysis

Libquantum




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

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

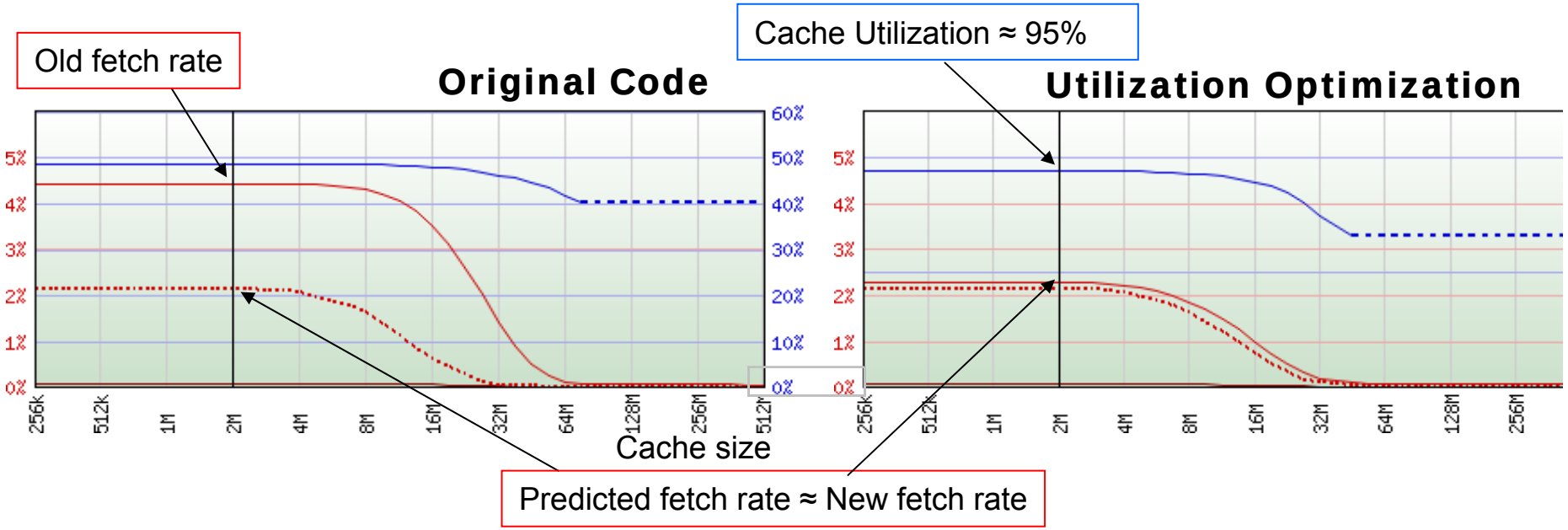
SlowSpotter's First Advice: Improve Utilization

➔Change one data structure

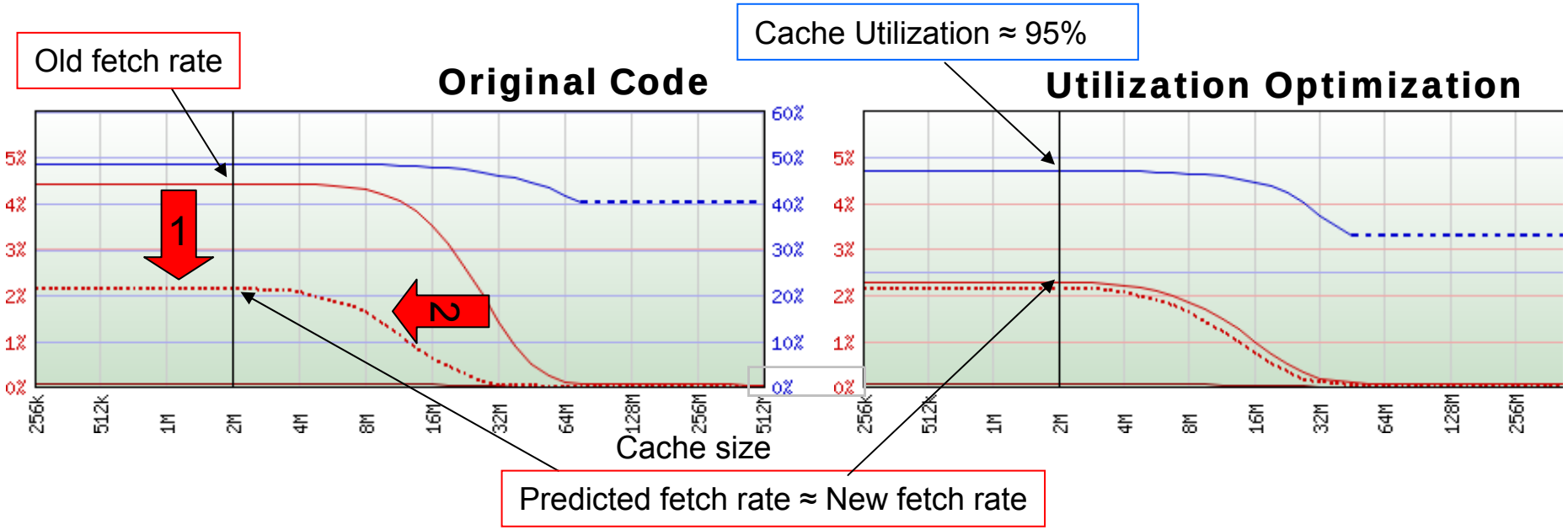
- Involves ~20 lines of code
- Takes a non-expert 30 min

After Utilization Optimization

Libquantum



Utilization Optimization

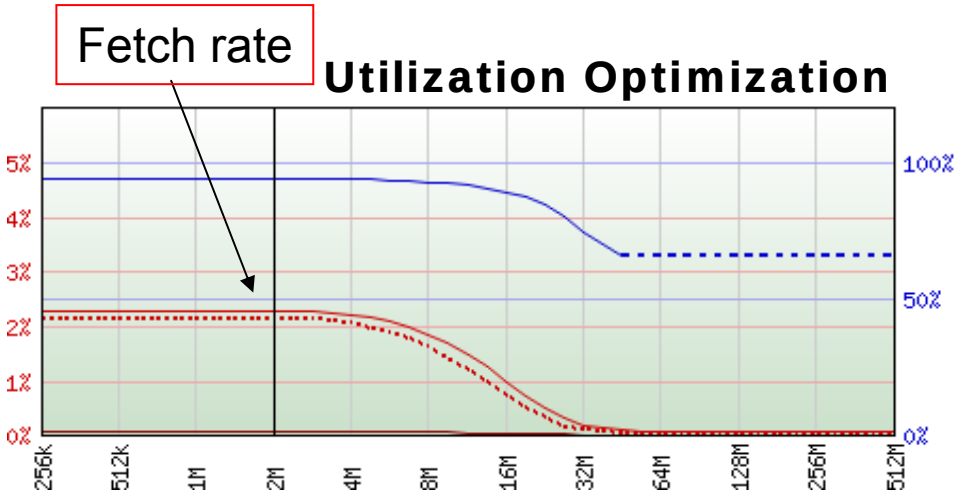


Two positive effects from better utilization

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

Reuse Analysis

Libquantum



Utilization + Fusion Optimization

```
...
toffoli(huge_data, ...)
cnot(huge_data, ...
...

...
fused_toffoli_cnot(huge_data,...)
...
```

Second-Fifth SlowSpotter Advice: Improve reuse of data

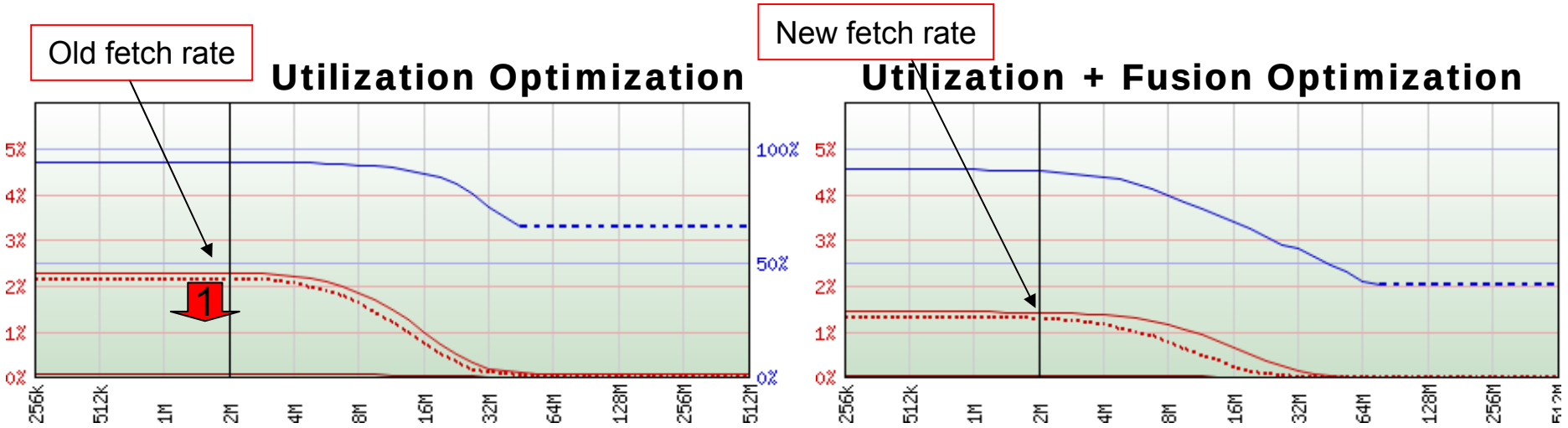
➔Fuse functions traversing the same data

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

UPPSALA UNIVERSITET

Effect: Reuse Optimization

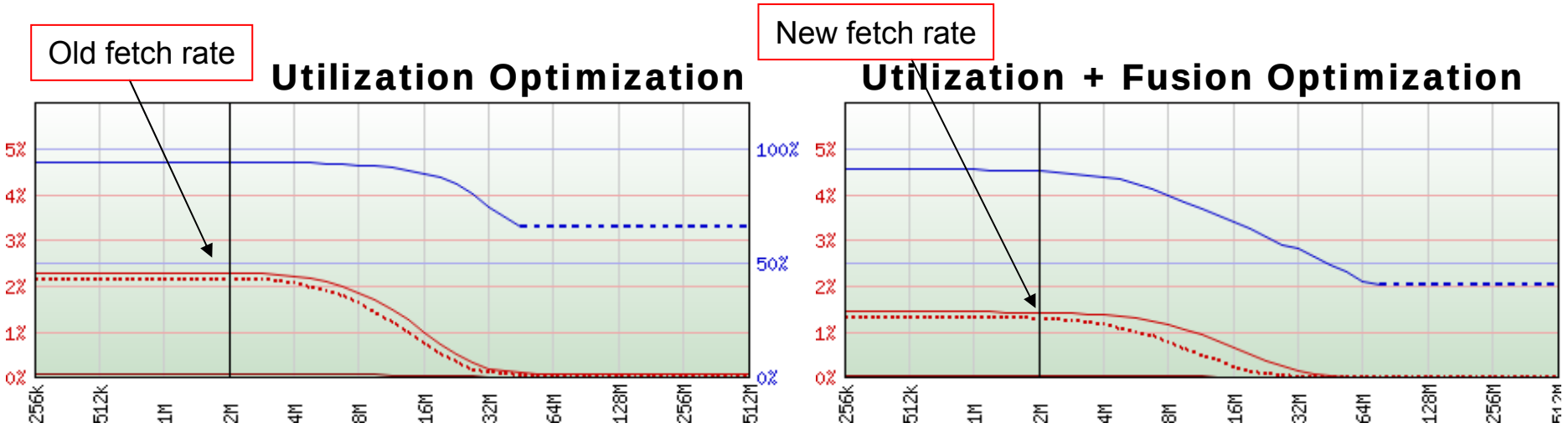
SPEC CPU2006-462.libquantum



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

Utilization + Reuse Optimization

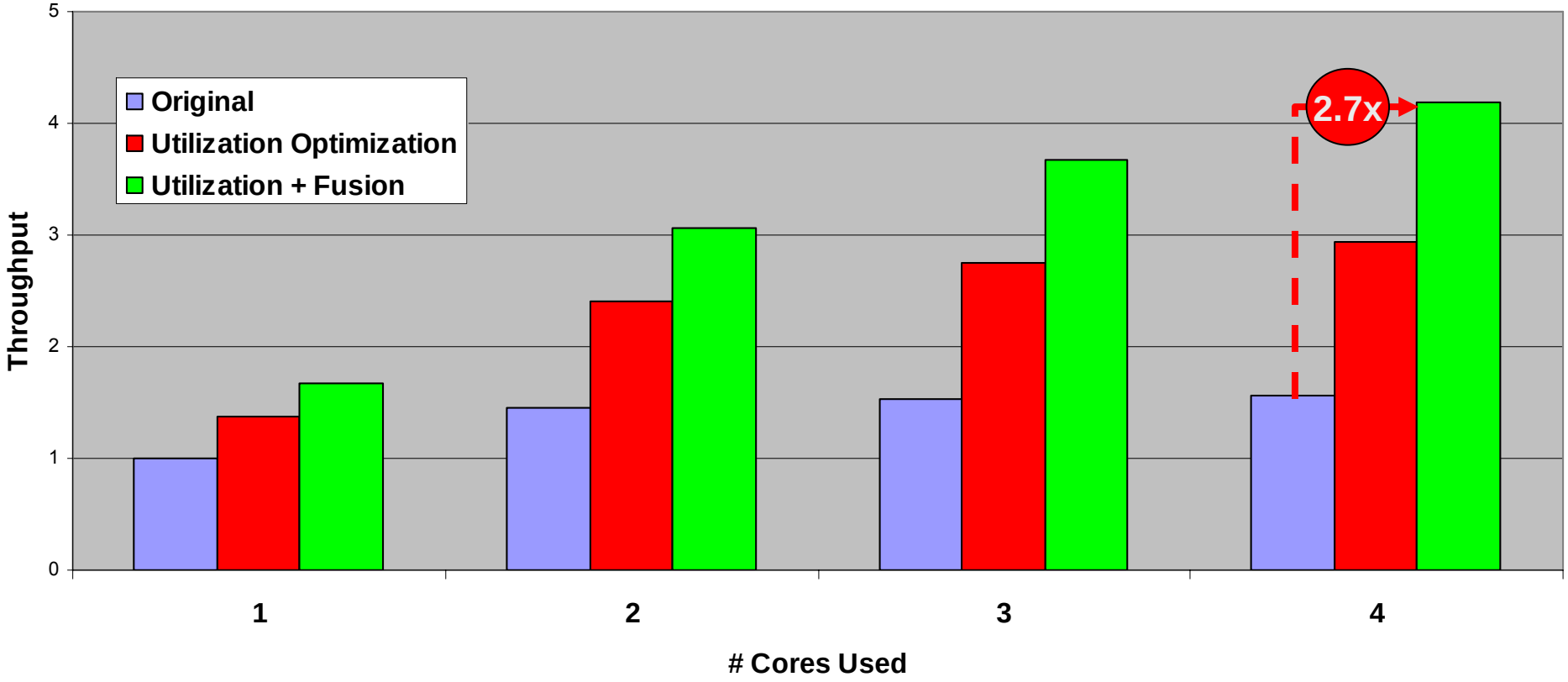
Libquantum



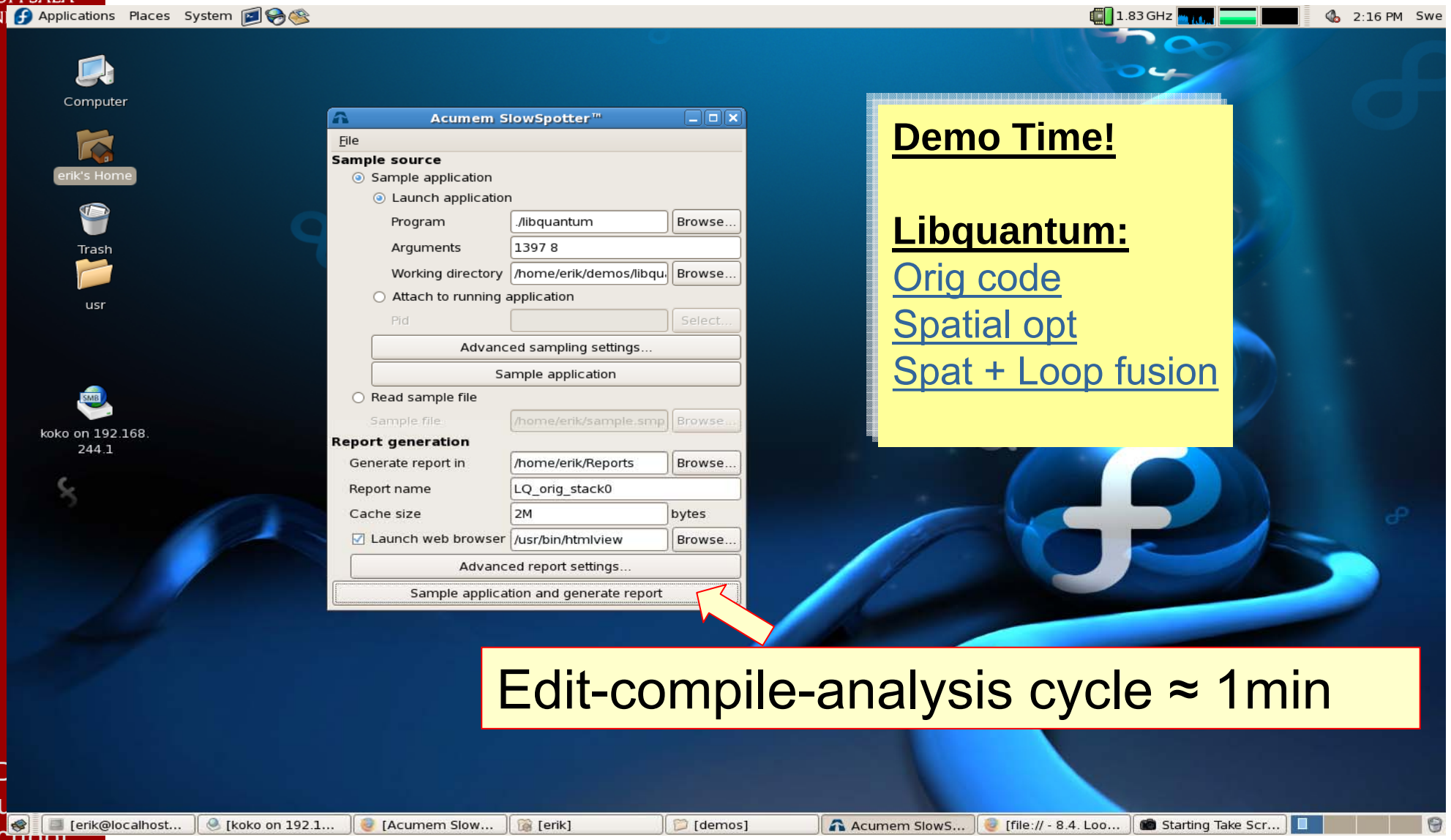
- Fetch rate down to 1.3% for 2MB
- Same as a 32 MB cache originally

Summary

Libquantum



Demo

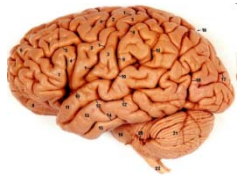

Applications Places System
1.83 GHz
2:16 PM Swe
Computer
erik's Home
Trash
usr
koko on 192.168.244.1
Acumem SlowSpotter™
File
Sample source
Sample application
Launch application
Program ./libquantum Browse...
Arguments 1397 8
Working directory /home/erik/demos/libqu. Browse...
Attach to running application
Pid Select...
Advanced sampling settings...
Sample application
Read sample file
Sample file /home/erik/sample.smp Browse...
Report generation
Generate report in /home/erik/Reports Browse...
Report name LQ_orig_stack0
Cache size 2M bytes
Launch web browser /usr/bin/htmlview Browse...
Advanced report settings...
Sample application and generate report
Demo Time!
Libquantum:
Orig code
Spatial opt
Spat + Loop fusion
Edit-compile-analysis cycle ≈ 1min
[erik@localhost...
[koko on 192.1...
[Acumem Slow...
[erik]
[demos]
Acumem SlowS...
[file:// - 8.4. Loo...
Starting Take Scr...

Making Legacy Code MC-ready

Optimize sequential code THEN go parallel

Tools to use

0. Start with an overall plan (application knowledge)



1. Optimize seq. code for the MC memory hierarchy

or

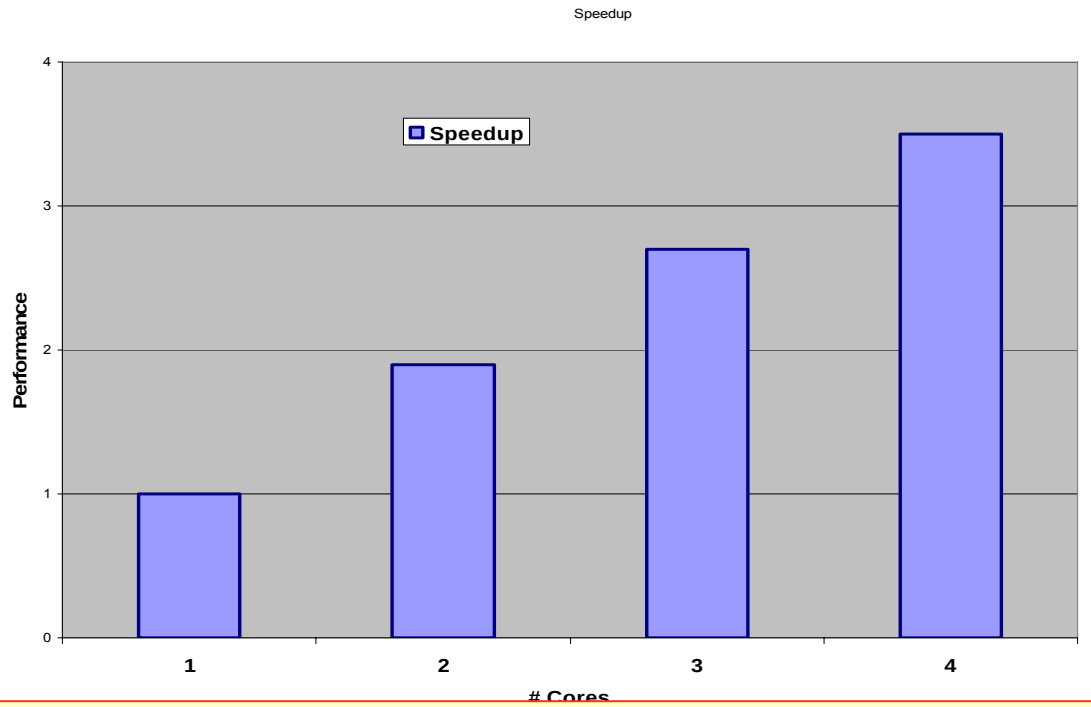
2. Parallelize and debug

OpenMP, TBB, Cilk, pthreads,...
+ Debuggers
+ Profilers

3. Optimize parallel code for MC

Example application: Cigar
(Study by Muneeb Khan, Uppsala University)

Cigar is a genetic algorithm widely used for machine learning. Available as open source.
More on the Cigar app at http://ecsl.cse.unr.edu/ and http://www.cse.unr.edu/~sushil/



Demo Time!

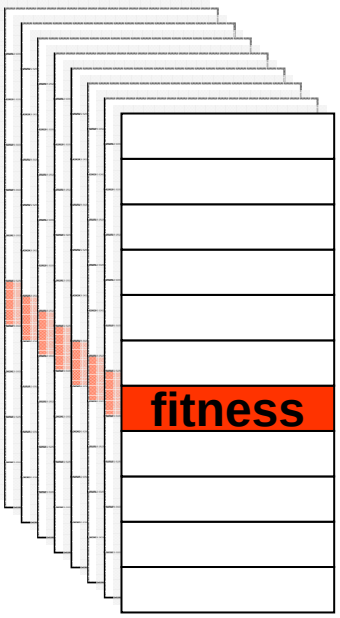
Cigar:
Original code
Optimized

This application is easy to parallelize and shows a great speedup on a multicore: 3,5x on 4 cores. It looks like we are done. Actually not! Up to 33x performance improvement can easily be achieve given the right tools.

UPPSALA UNIVERSITET

The Big Picture: Cigar

rank[ ]




```
for(current = 0; current < howmany - 1  ; current++) {
   max = current;
   /* Find Next best */
   for(i = current + 1; i < size ; i++) {
     if((rank[i])->fitness > ( rank[max])->fitness) {
       max = i;
     }
    }
   SwapInt(&rank[current], &rank[max]);
}
```

The data structure “rank” contain thousands of vector entries, where each entry is a struct containing 11 members of 8 bytes each.

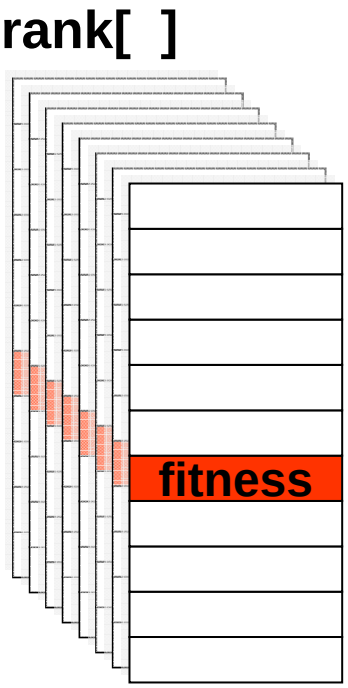
The important loop is only looking at the “fitness” member of each vector entry to find the “max” entry. Eventually, the max entry will be involved in a SwapInt().

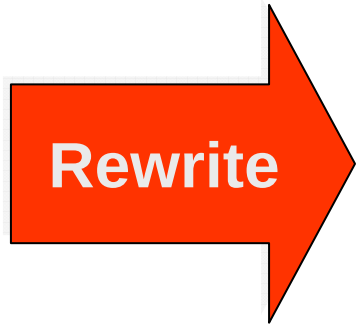
In order to make the fitness test ☺, an entire cache line of 64 bytes will have to be brought in from memory, from which only 8 bytes will be accessed – a bad idea!

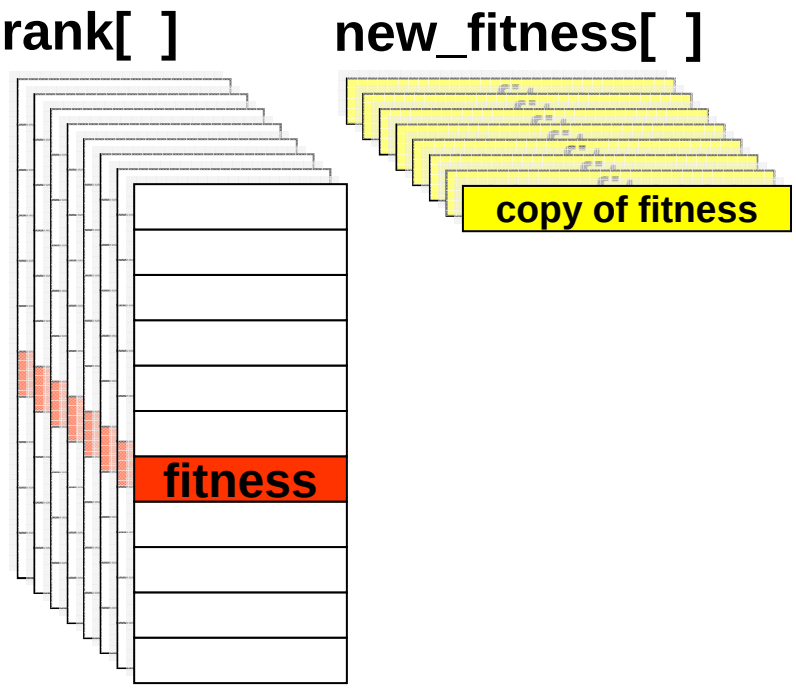
Clearly, we could break up the struct into 11 separate vectors to fix the problem, but that may complicate the rest of the code, such as the implementation of SwapInt().

Instead a “new_fitness” vector is introduced and its values made to correspond to the fitness values.

Adding a shadow data structure





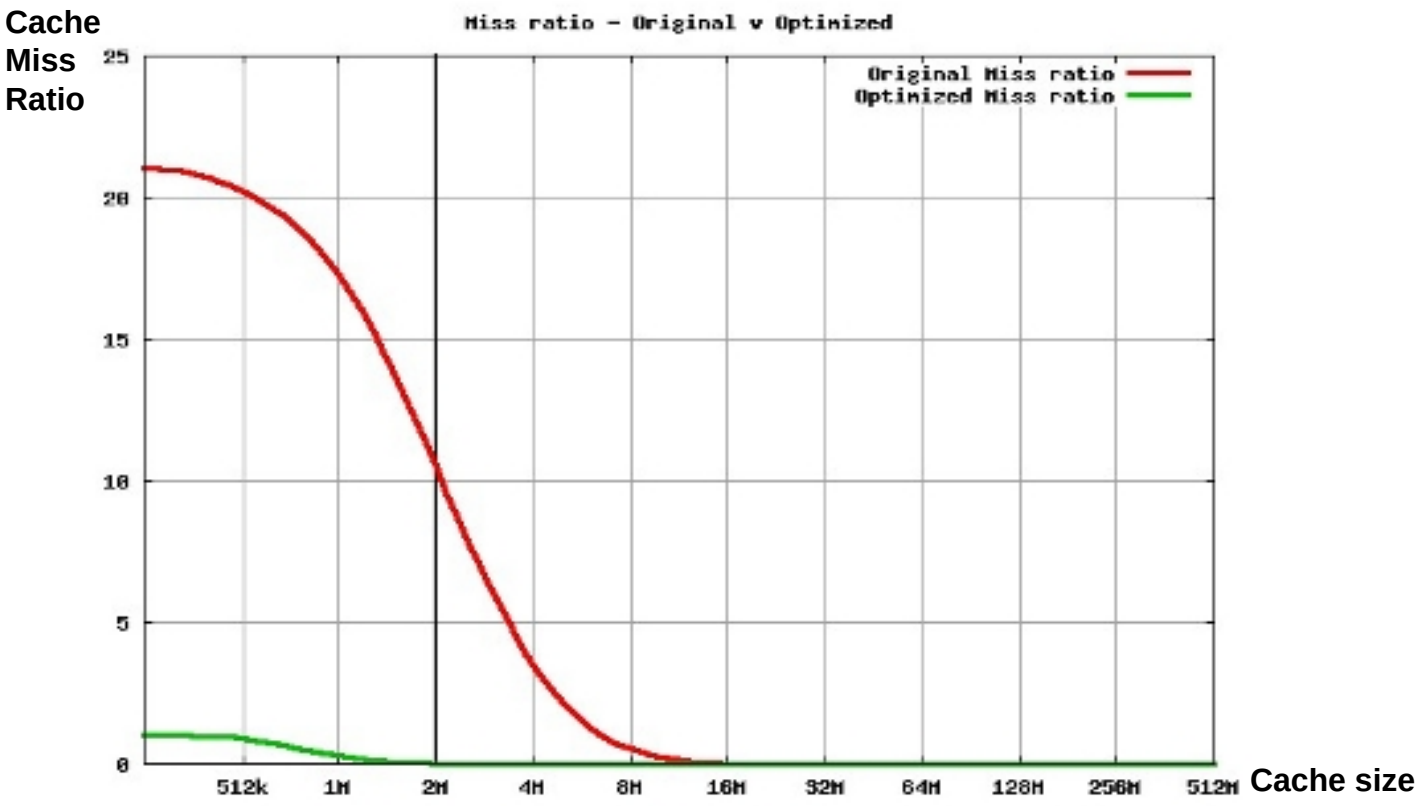



```
for(current = 0; current < howmany - 1  ; current++) {
   max = current;
   /* Find Next best */
   for(i = current + 1; i < size ; i++) {
     if((rank[i])->fitness > ( rank[max])->fitness) {
       max = i;
     }
    }
   SwapInt(&rank[current], &rank[max]);
}
```

```
for(current = 0; current < howmany - 1  ; current++) {
   max = current;
   /* Find Next best */
   for(i = current + 1; i < size ; i++) {
      if(new_fitness[rank[i]] > new_fitness[rank[max]]) {
         max = i;
     }
    }
   SwapInt(&rank[current], &rank[max]); //UNCHANGED
}
```

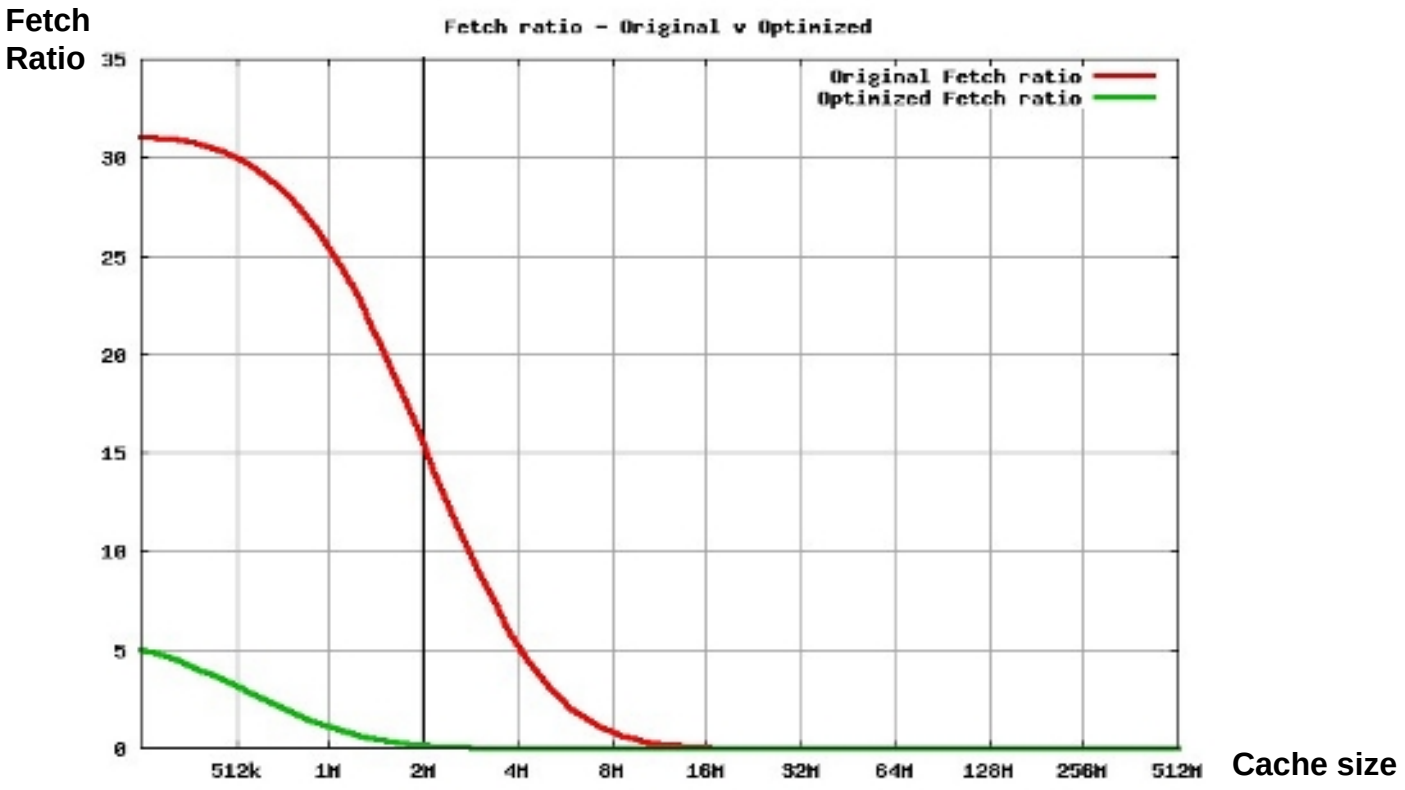
UPPSALA UNIVERSITET

Impact on Miss Ratio



For caches in the 2-8 Mbyte area, the change in Miss Ratio is drastic, going from several percent down to pretty much nothing. This will in effect remove most of the CPU stalls due to cache misses.

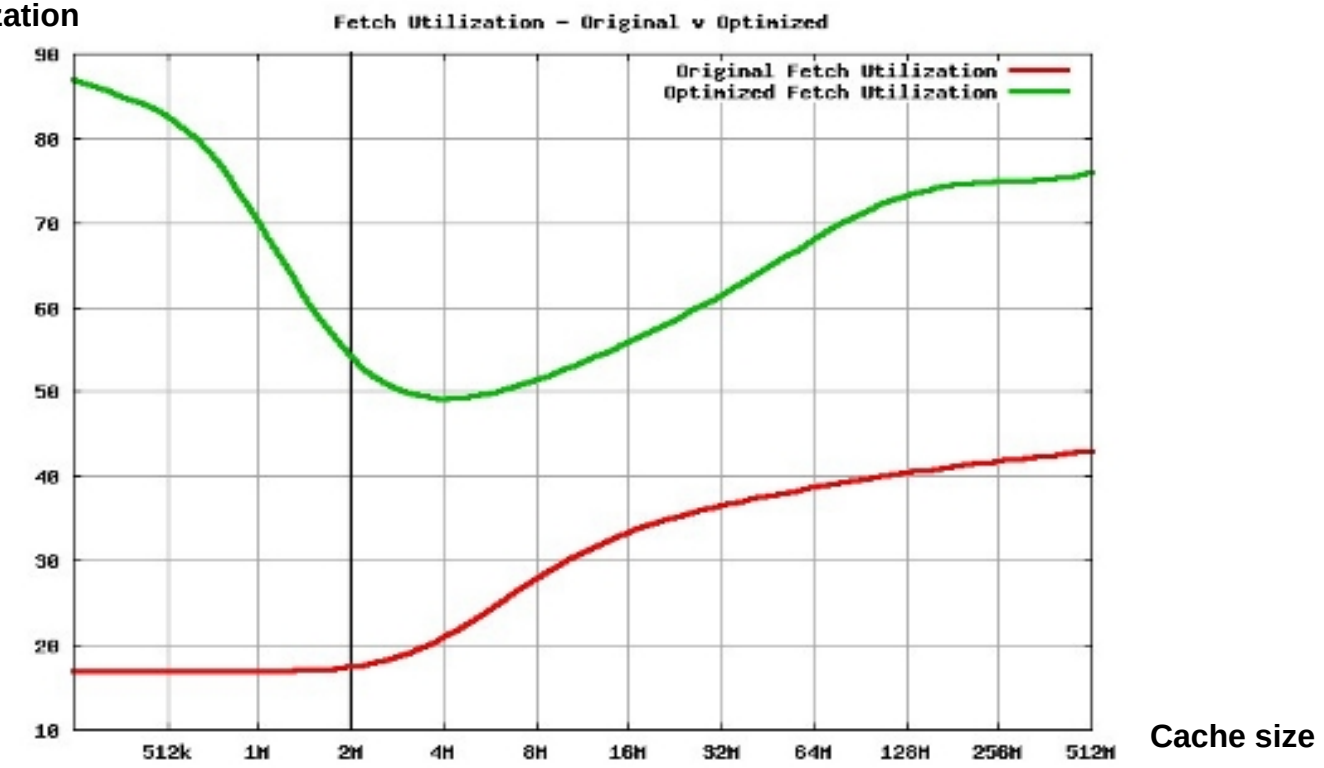
Impact on Fetch Rate



Fetch rate is the probability that a ld/st operation would cause a fetch across the DRAM interface. This corresponds to the cache misses + the hardware prefetch activities.

For caches in the 2-8Mbyte range, the change is again drastic, going from several percent down to pretty much nothing. This will in effect remove most of the DRAM accesses.

Impact on Fetch Utilization

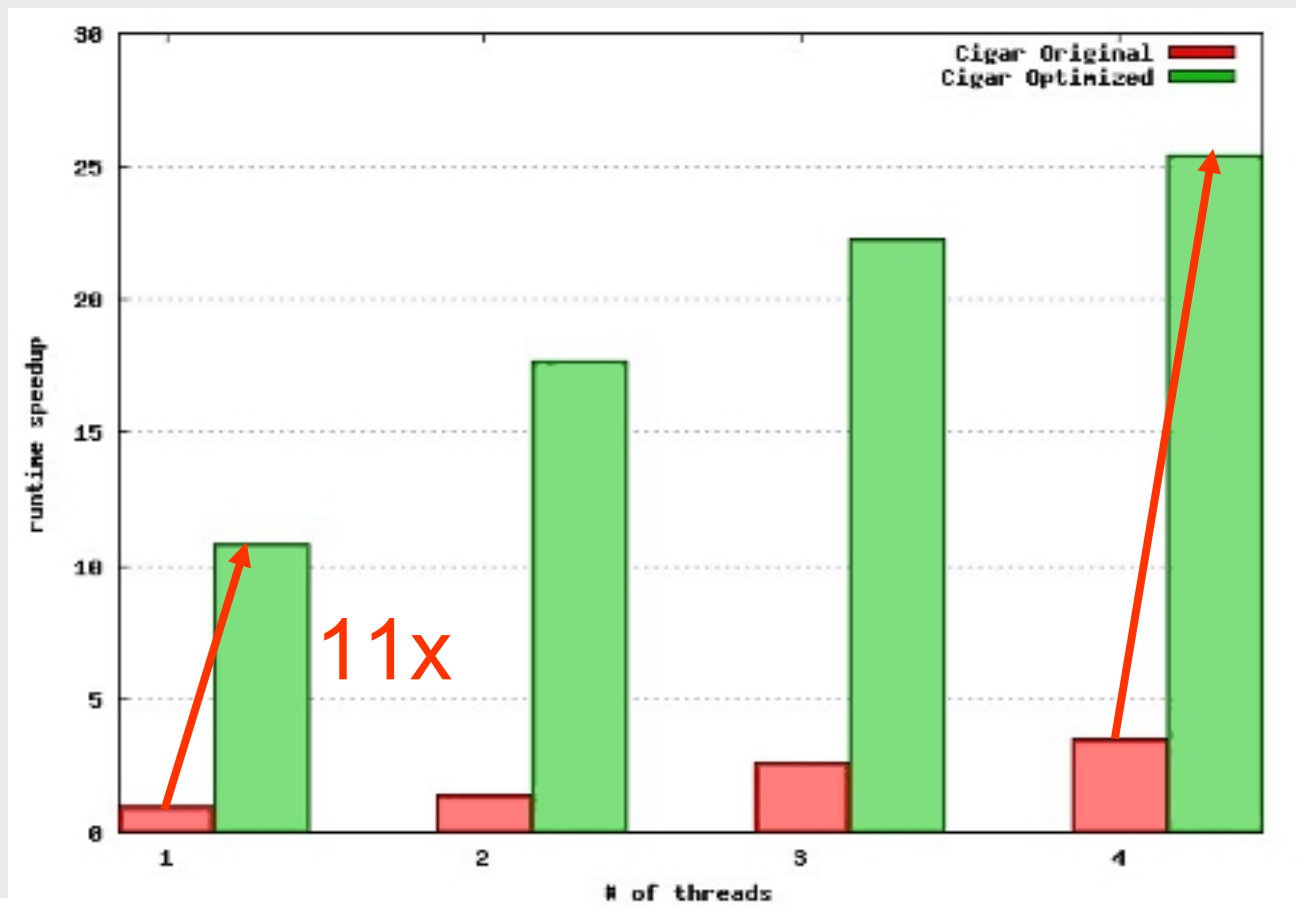


Also the fetch utilization, i.e., the fraction of data brought to the cache which is ever used before cache eviction, can be studied.
For caches in the 2-8Mbyte range, the overall fetch utilization for the entire cigar application goes from 18-25 percent to 50 percent. This grants a better usage of the cache resources which is the main reason for the performance gain.

UPPSALA UNIVERSITET

Parallelizing the Optimized Version

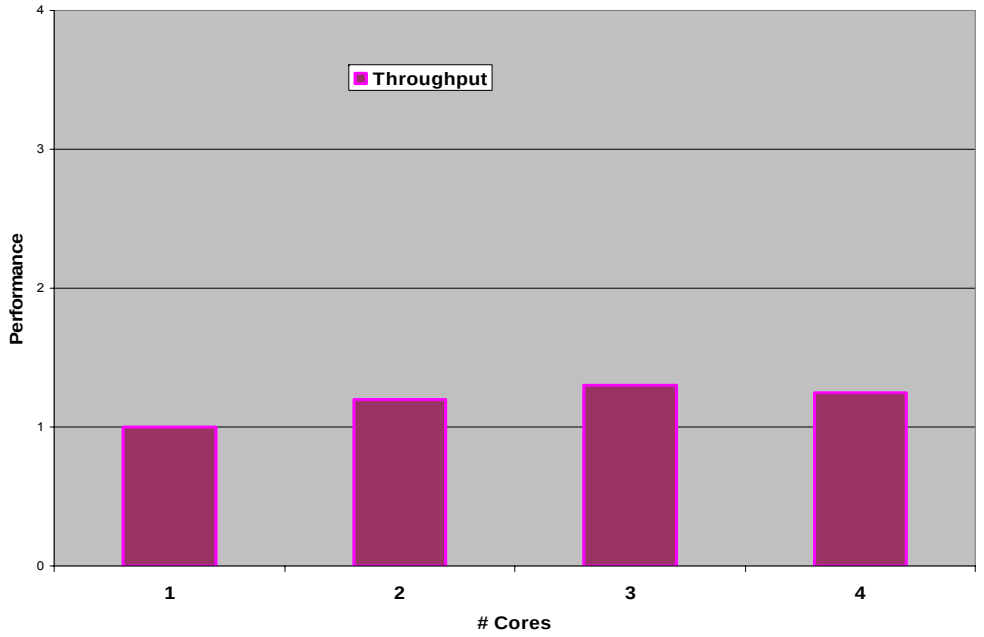
Intel Core2 (Intel Xeon E5345)



Now, when the serial code has been optimized, we can once more parallelize the application. The optimized parallel performance is shown in this graph relative to the original single-threaded performance. Single-core performance is 11x faster and 4-core performance is 7.3x faster

UPPSALA UNIVERSITET

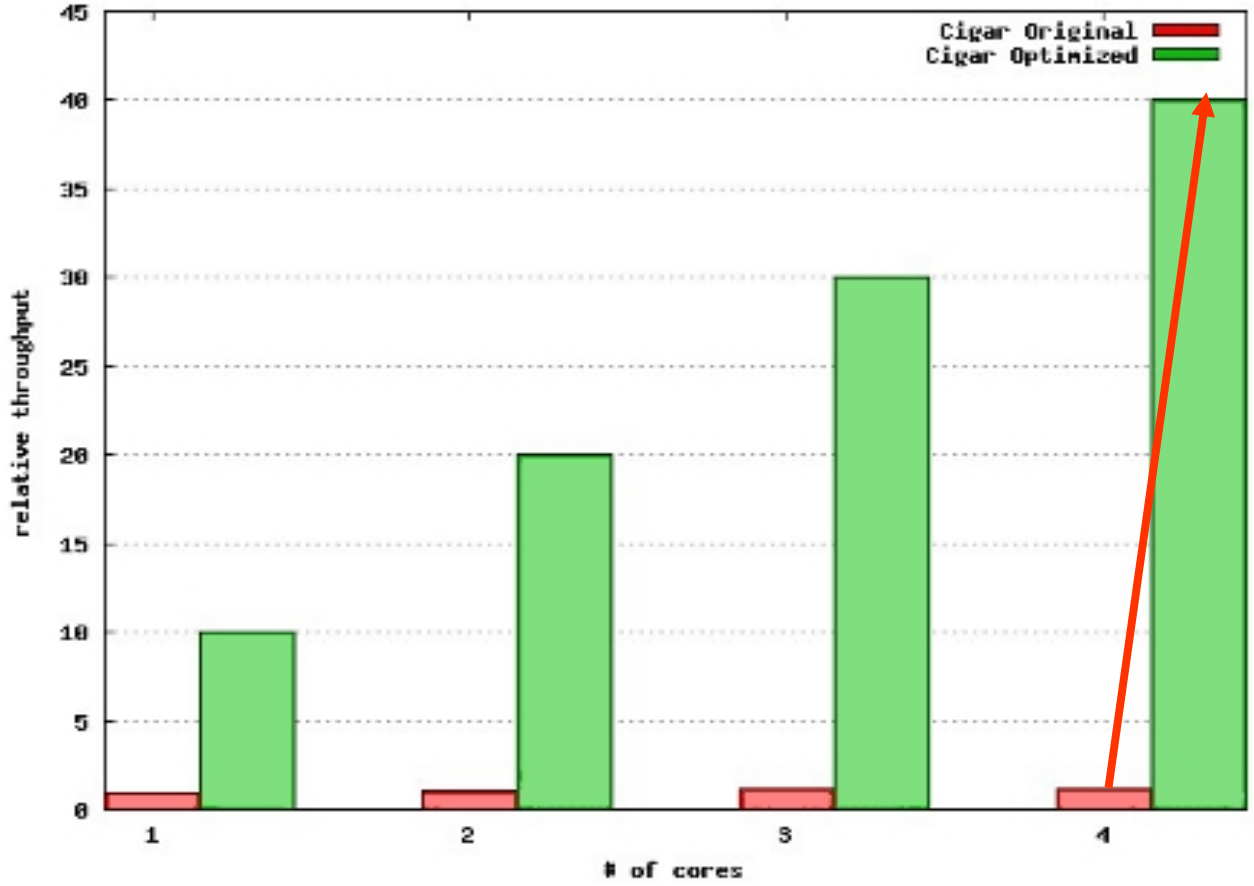
Original Cigar Throughput



Throughput scalability is a different way to look at the performance of an application. Here, several single-threaded instances of the application is run at the same time. Even though the different instances do not explicitly depend on each other, they will nevertheless fight over the shared resources, e.g., running four threads on four cores implies that each thread will get one quarter of the shared cache.

A system using four cores to run four instances of Cigar will actually result in a lower throughput than if only three cores were used.

Throughput Performance
Intel Core2 (Intel Xeon E5345)

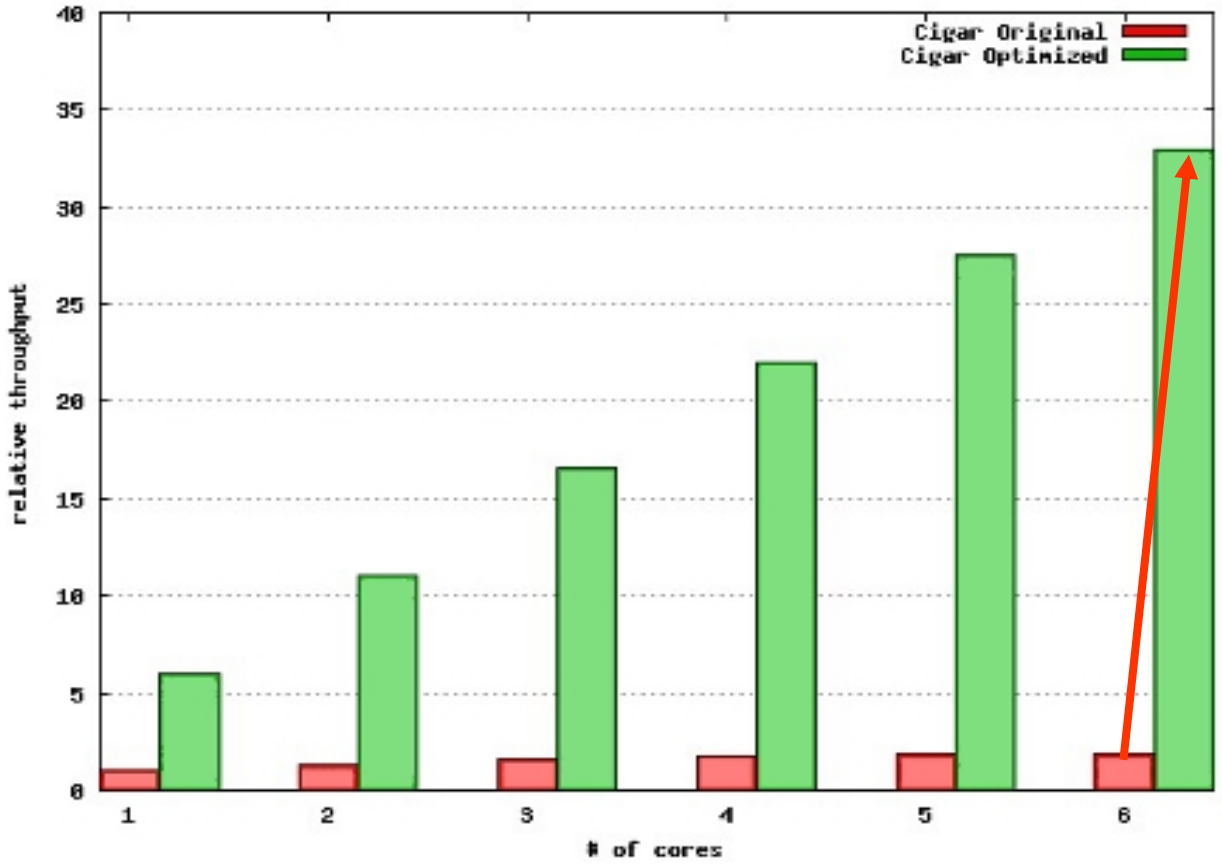


33x

The optimization puts a much lower pressure on the shared cache resulting in a 33x better throughput for four cores.

UPPSALA UNIVERSITET

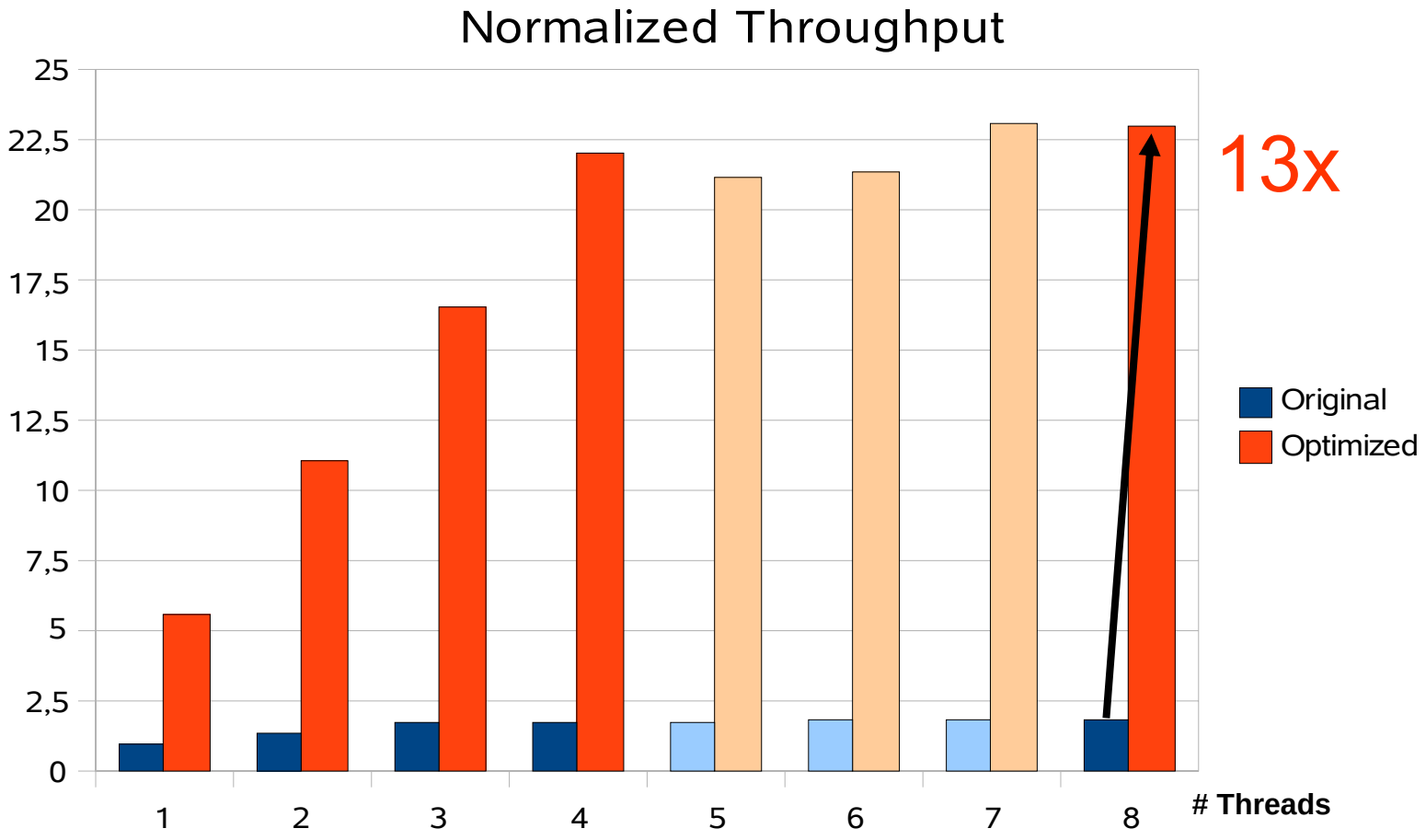
Throughput Performance (AMD's Istanbul)



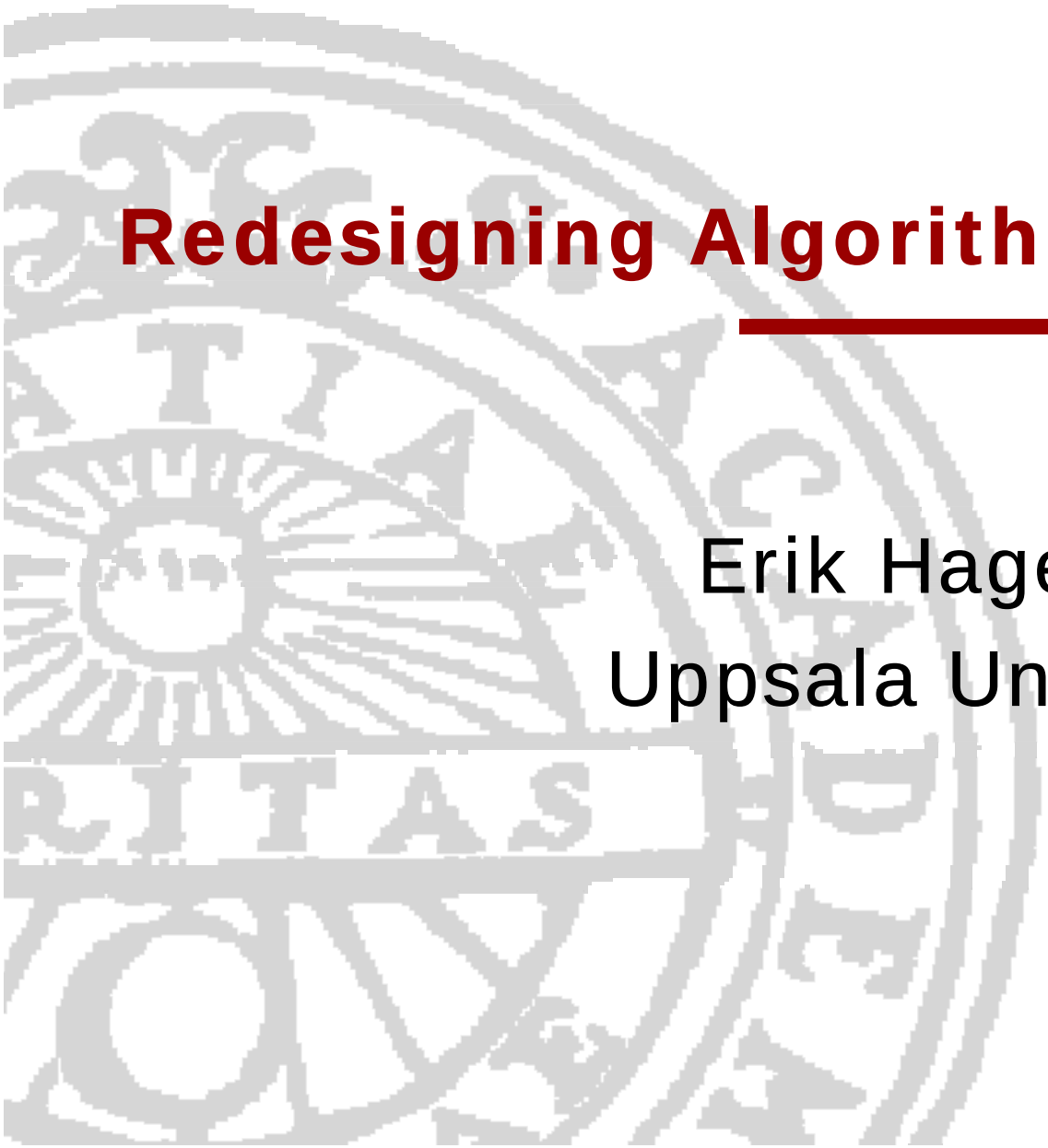
17x

AMDs new six-core Istanbul processor can enjoy a 17x better throughput due to the optimization on six cores

Throughput Performance (Intel i7)



Intel's new four-core i7 (Nehalem) processor enjoy a 13x better throughput due to the Optimization on four cores. Note that each core can run up to two threads.



Redesigning Algorithms for Multicores

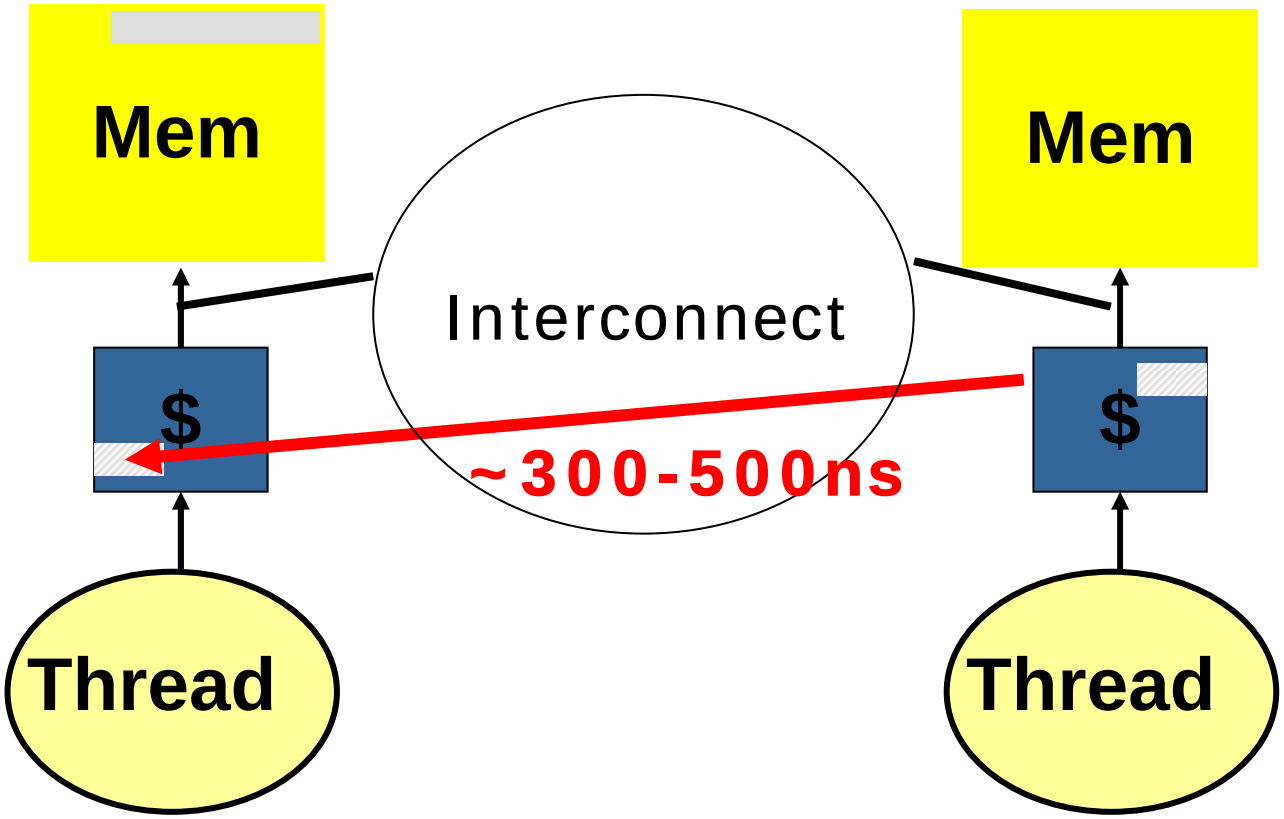
Erik Hagersten
Uppsala Universitet

UPPSALA UNIVERSITET

One slide summary

- Today's algorithms are based on the assumption that inter-thread communication is expensive
- The communication cost of future processors is close to zero
- Memory bandwidth is the bottleneck of the future
- Should we redesign supercomputer algorithms to face this fact?
- Yes: We demonstrate 3x performance gain!

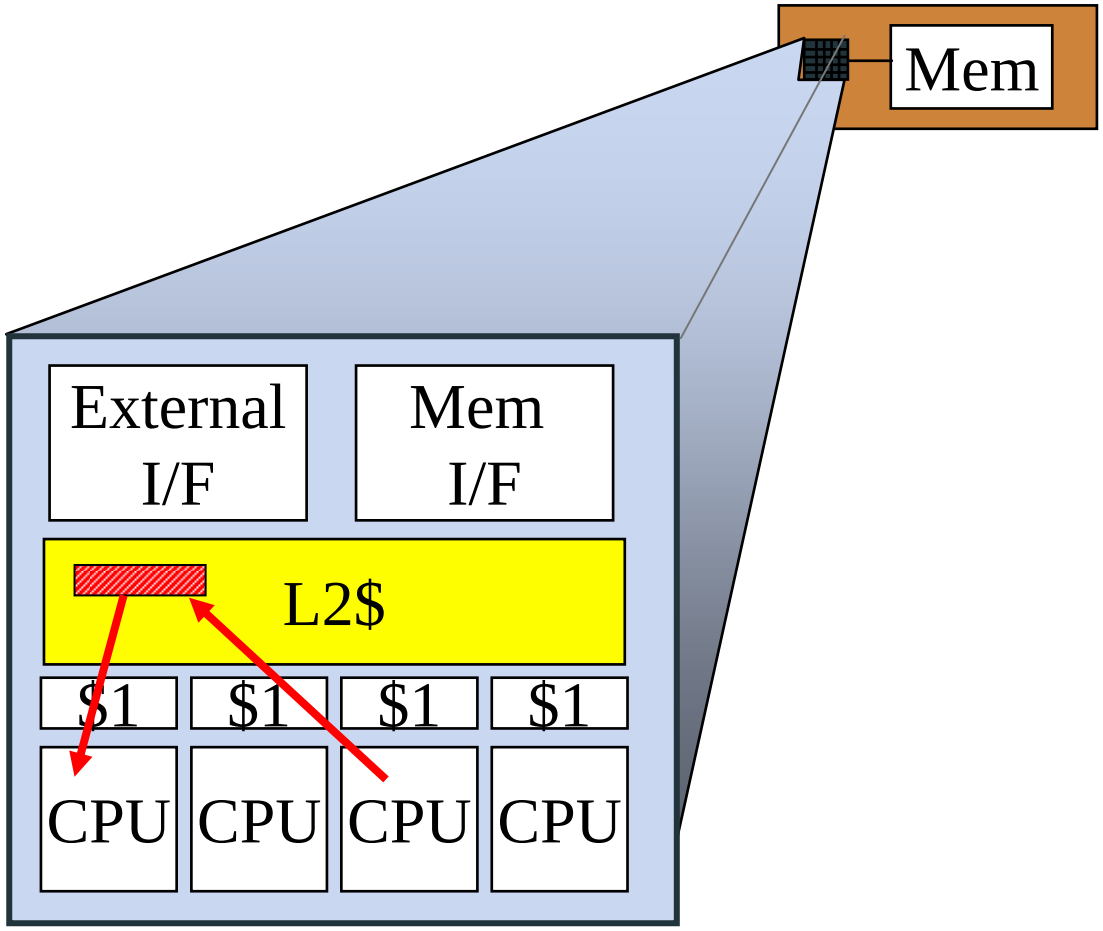
Non-uniform Architectures: NUMA
➔ Communication cost is much worse!



Read A
Read A
…
…
Read A

…
Read A
…
Write A

Multicores:



~10ns

UPPSALA UNIVERSITET

Criteria for HPC Algorithms

- Past:
  - Minimize communication
  - Maximize scalability (1000s of CPUs)
- Multicore today:
  - On-chip communication is “for free”
  - Scalability is limited to ~10 threads
  - The caches are tiny
  - Memory bandwidth is the bottleneck

➔ Data locality is key!

Case Study: Gauss-Seidel on Multicores

Erik Hagersten
Uppsala University
Sweden

Thanks: Dan Wallin(arch), Henrik Löf (sci comp) and Sverker Holmgren (sci comp)
From Wallin et al, ICS 2006

UPPSALA UNIVERSITET

Selected HPC Wire Articles

More Than 16 Cores May Well Be Pointless

Sandia Labs, Dec 07 2008

Up Against the Memory Wall

”Never mind the cores. Just hand over the cache”

Michael Feltman, Dec 11 2008

HPC@Intel: When to Say No to Parallelism

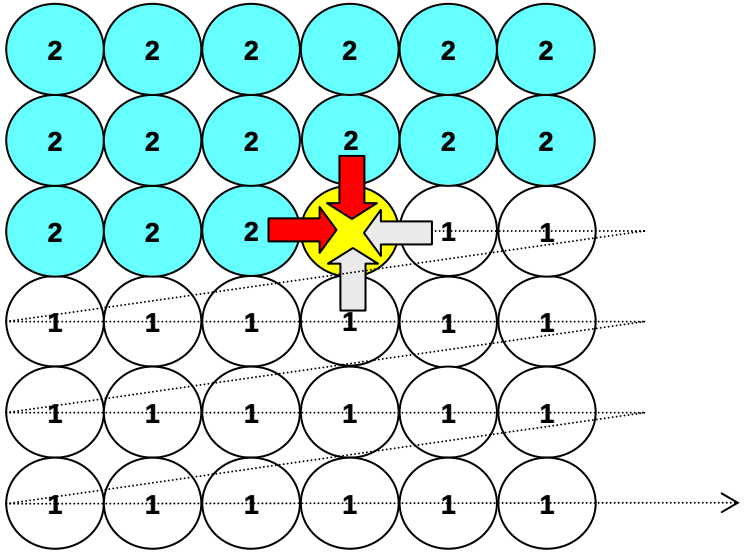
Sanjiv Shah, Intel. January 14 2009

Finding a Door in the Memory Wall

Erik Hagersten, Acumem. Feb-April 2009

Example: Gauss Seidel

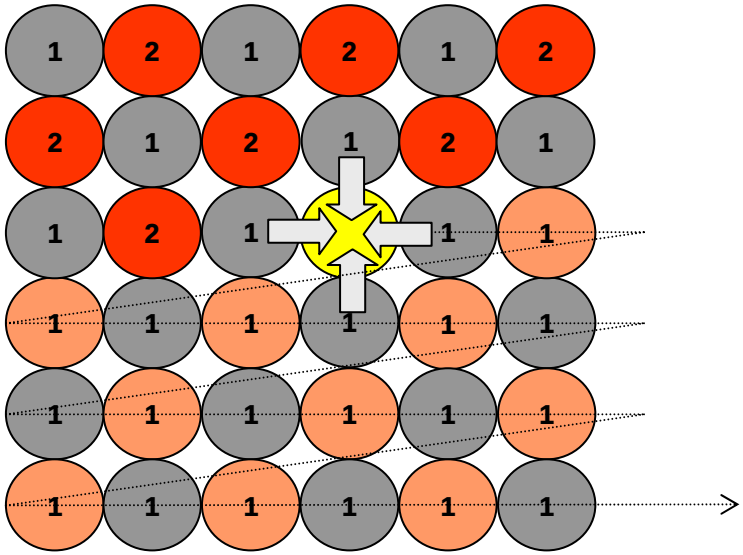
Mission: **"Maximize the parallelism and minimize the inter-thread communication"**



LOOP:
  UPDATE ALL POINTS
  IF (convergence_test)
    <done>

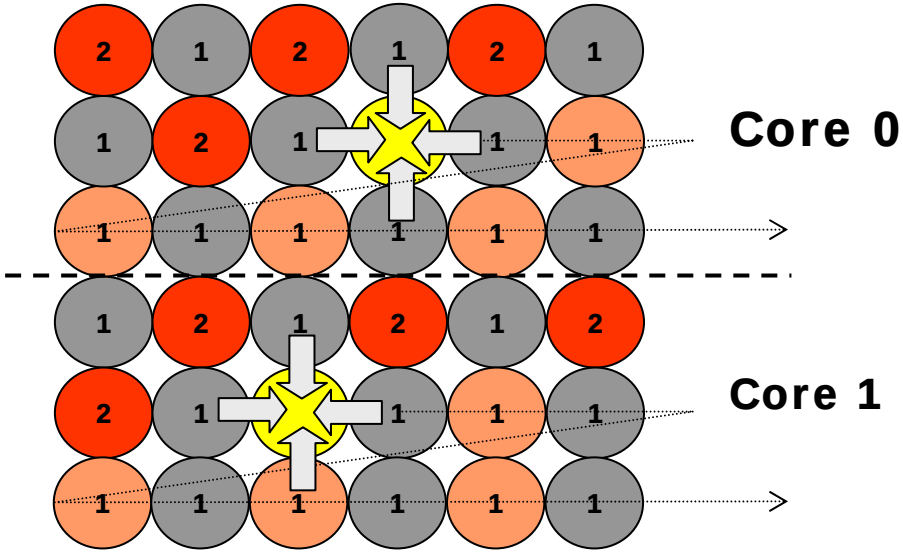
(Longer explanation: Finding a Door in the Memory Wall @ **HPCWire**)

State-of-the-art: Removing Dependence: Red/Black



LOOP:
  UPDATE ALL **RED** POINTS
  UPDATE ALL **BLACK** POINTS
  IF (convergence_test)
    <done>

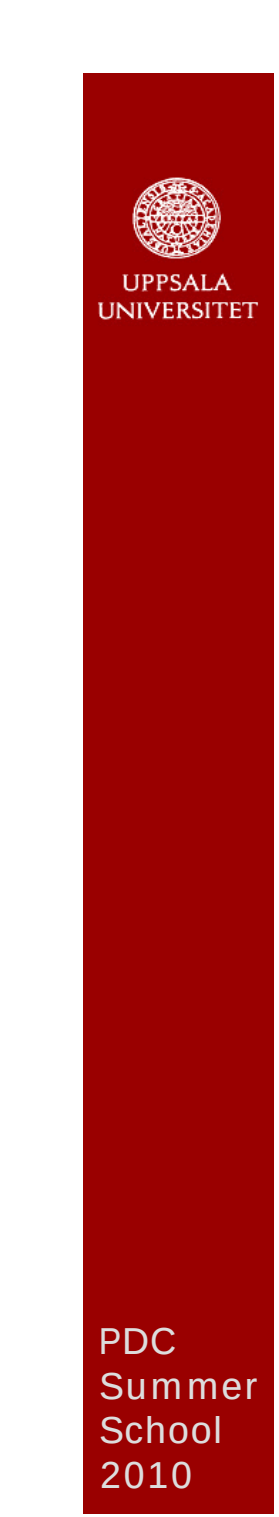
State-of-the-art: Red/Black, Parallelism = $N^2/2$



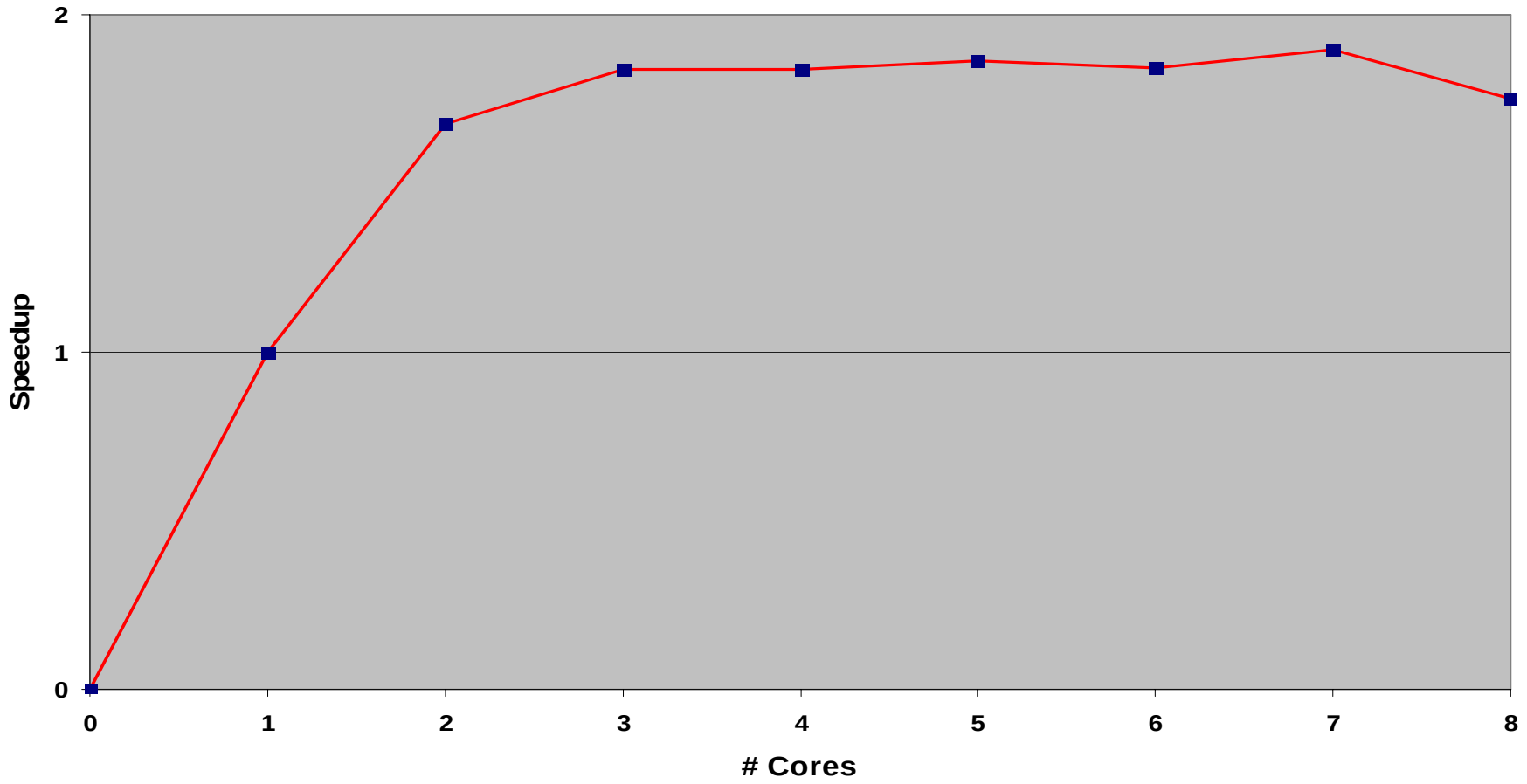
LOOP:
  IN PARALLEL: UPDATE ALL **RED** POINTS
  <barrier>
  IN PARALELL: UPDATE ALL **BLACK** POINTS
  <barrier>
  IF (convergence_test)
    <done>

Limited communication ☺
$N^2/2$ parallelism ☺
Done!
Only one problem…

OPT

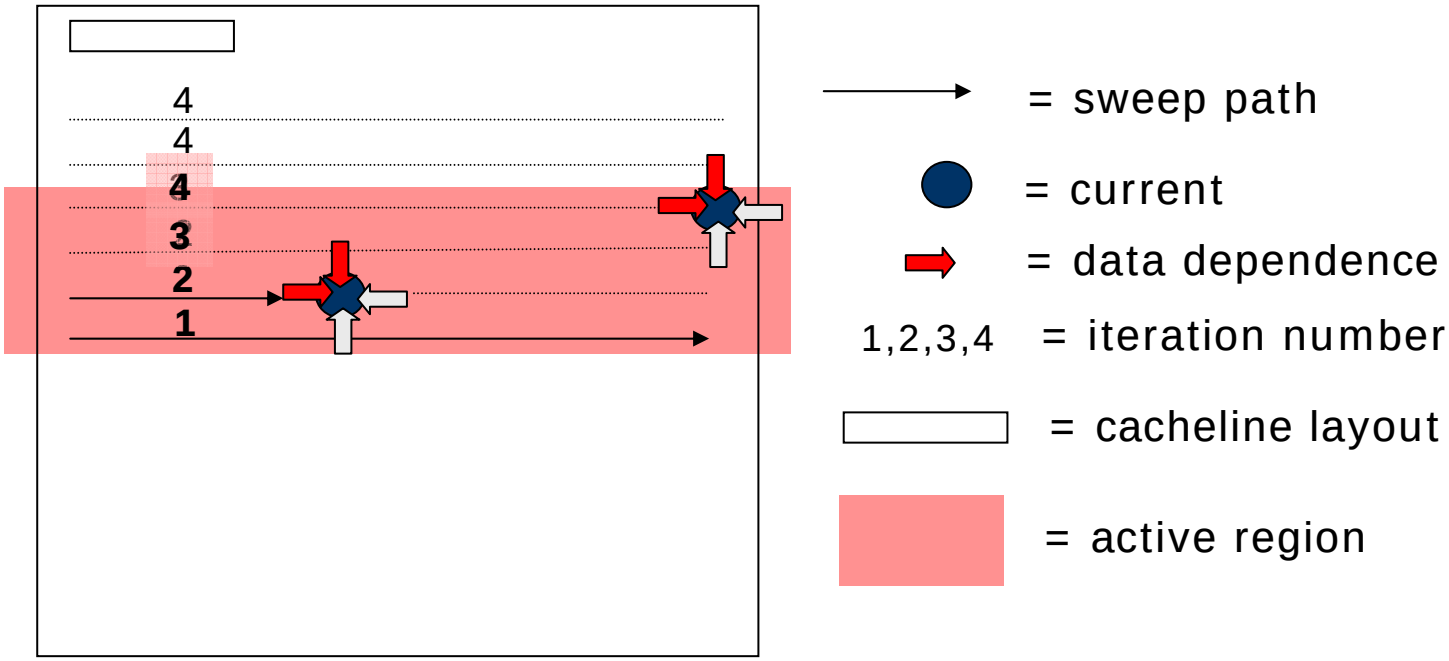


Only One Problem: Speed



Back to the drawing board: Temporal blocking for seq. code

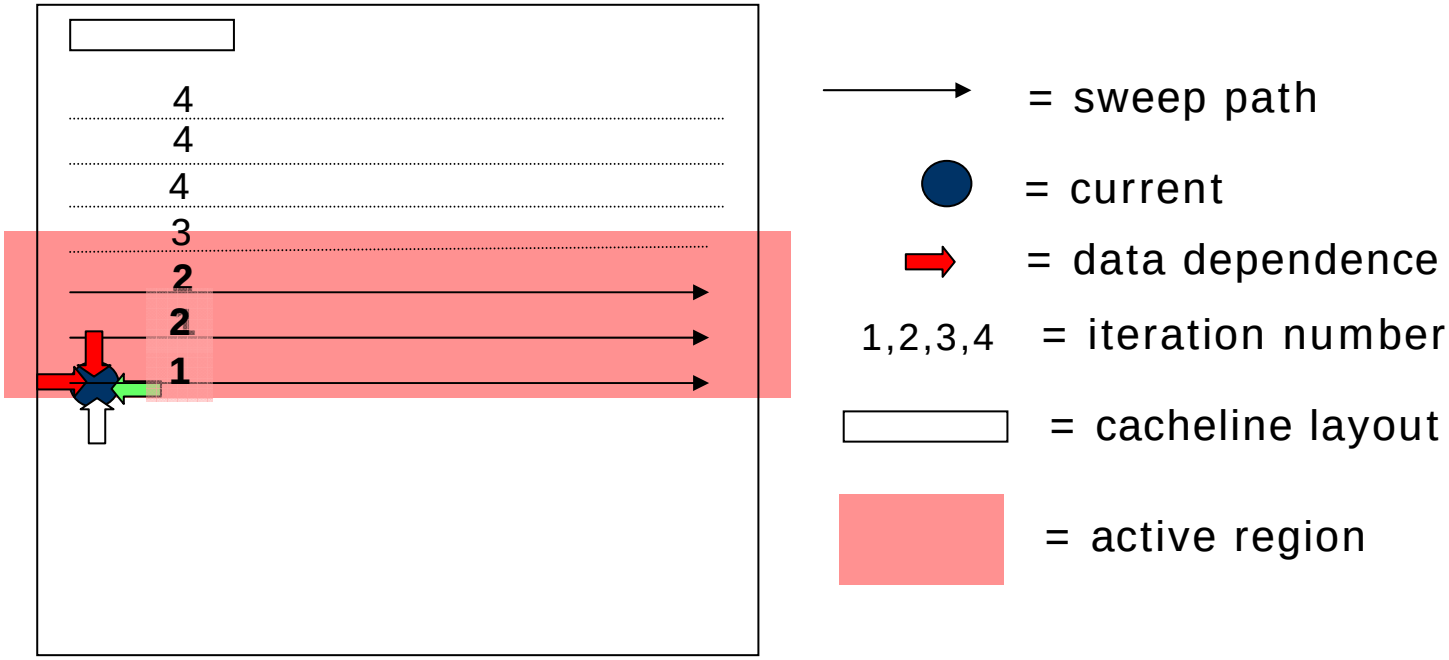
Communication is "for free" and moderate parallelism is OK
Priority 1: limit bandwidth needs!




```
LOOP:
   LOOP:
      UPDATE ALL POINTS IN ACTIVE REGION
      SLIDE DOWN THE REGION
   IF (convergence_test)
    <done>
```

Back to the drawing board: Temporal blocking for seq. code

Communication is "for free" and moderate parallelism is OK
Priority 1: limit bandwidth need!



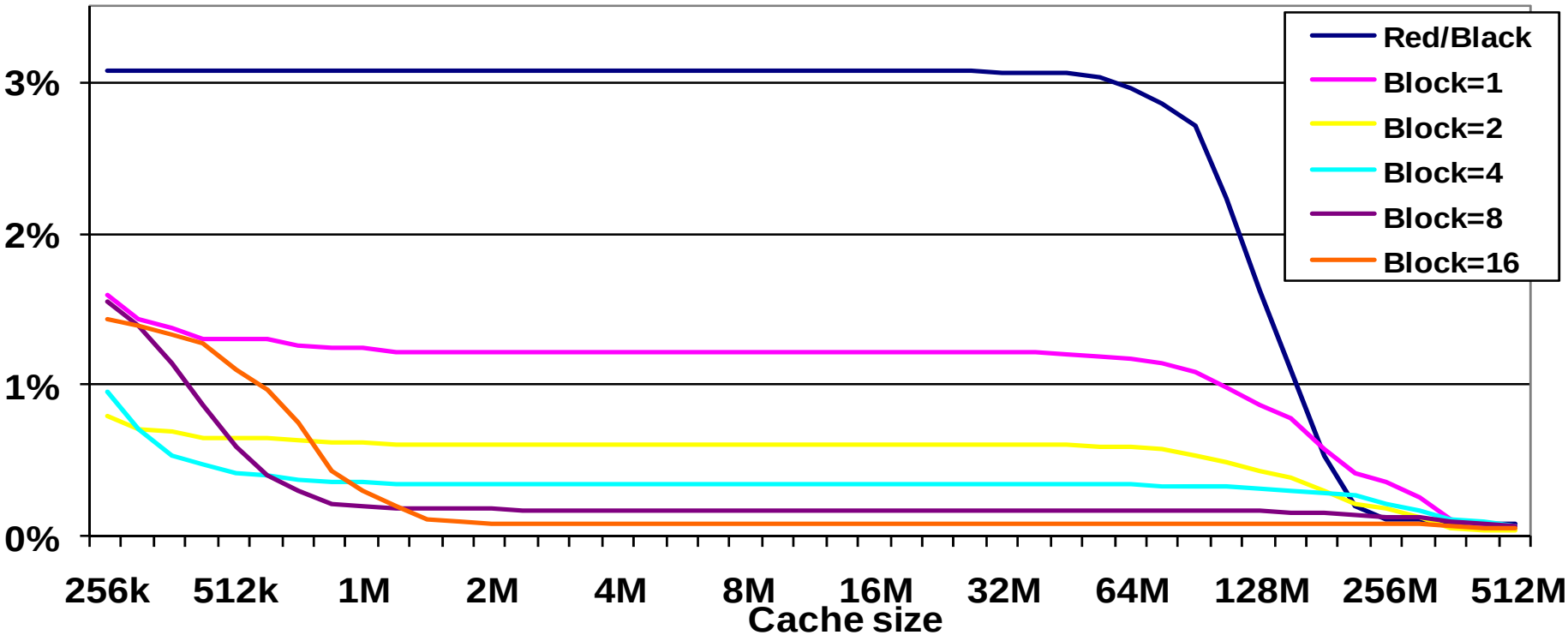

```
LOOP:
   LOOP:
      UPDATE ALL POINTS IN ACTIVE REGION
      SLIDE DOWN THE REGION
   IF (convergence_test)
    <done>
```

4 iterations in one sweep!

UPPSALA UNIVERSITET

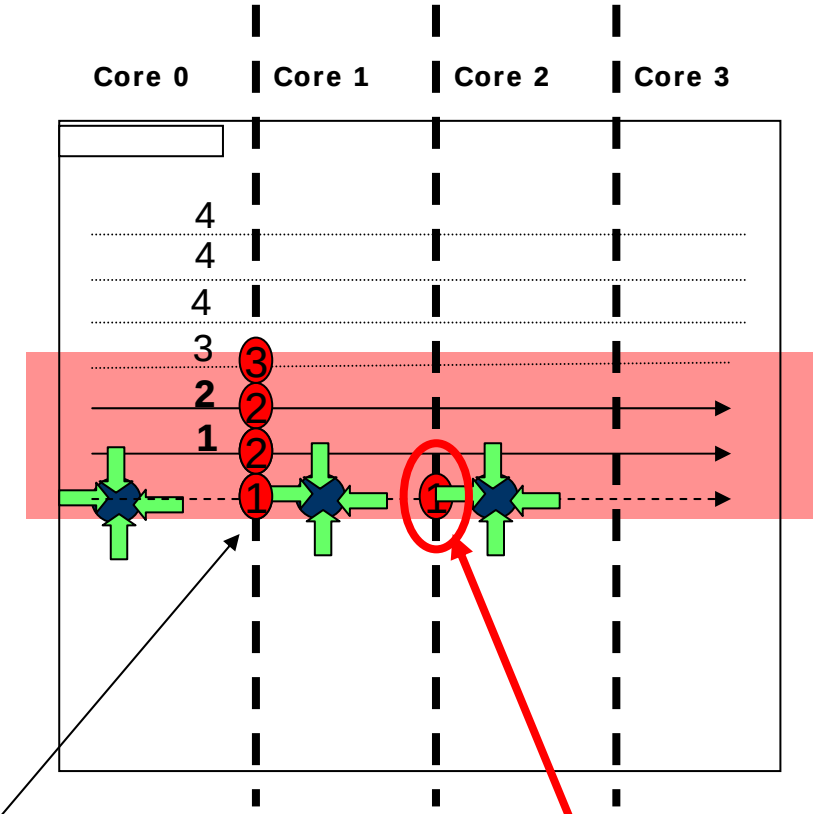
DRAM_traffic(cache_size)

Fetch Rate,
i.e, fraction of mem_ops generating DRAM traffic



OPT 64

G-S, temp block Parallelism = N



= sweep path

= current

= data dependence

1,2,3,4 = iteration number

= cacheline layout

= active region

1 = sync flag iteration no

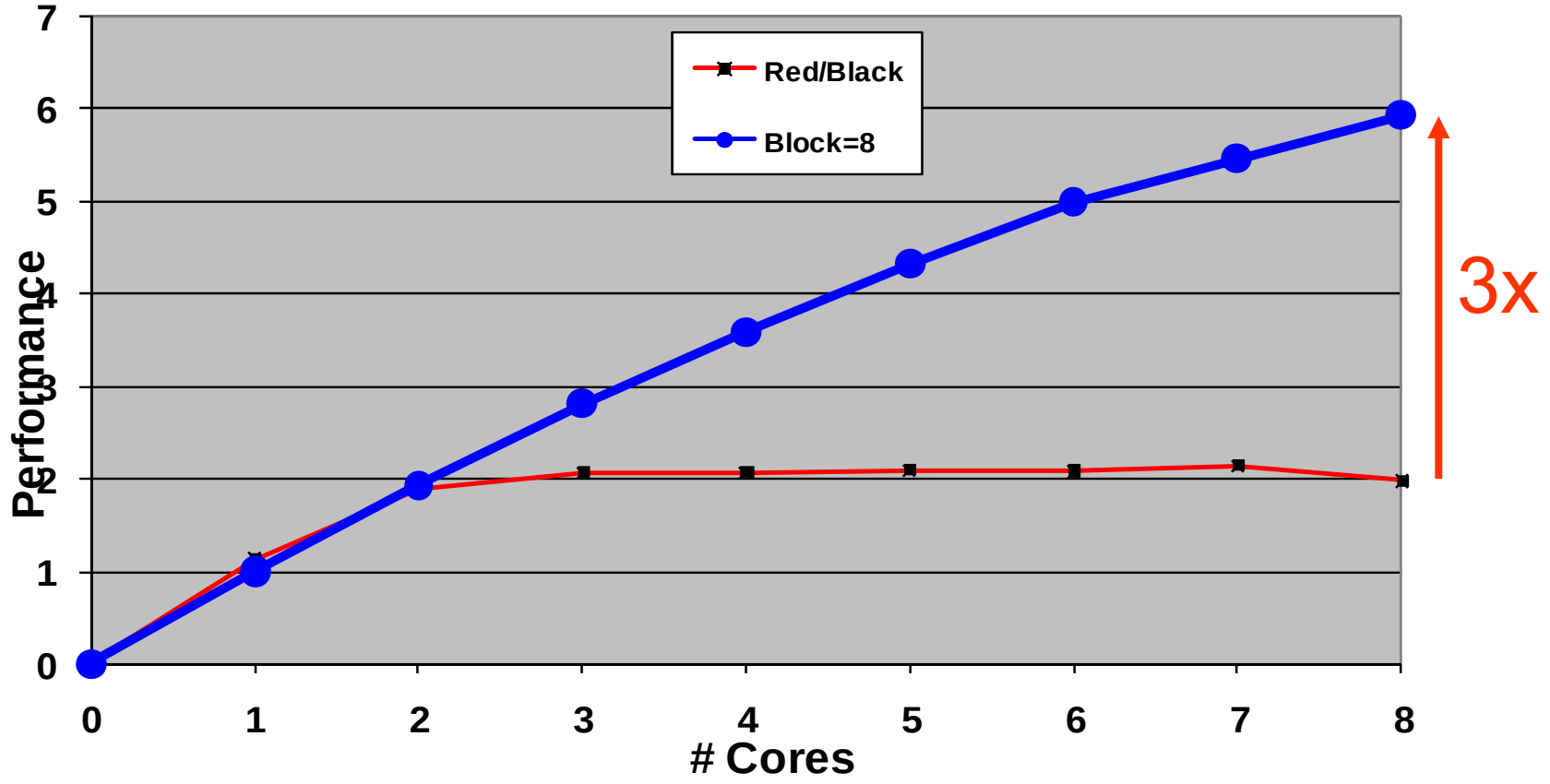
Synchronization flags

Wait until ”lefty” is done:
Lots of communication ☹
- Producer/Consumer Flag
- Sharing of data values

Only N-fold parallelism ☹

Lessons Learned: Open the Door in the Memory Wall BEFORE Parallelizing

Mission: "Optimize for data locality first, THEN maximize parallelism and minimize inter-thread comm."



[Wallin, Löf, Holmgren, Hagersten @ ICS 2006]

UPPSALA UNIVERSITET

Conclusion temporal blocking

When the rules change, the game change!

Multicores change the rules

Techniques described here can be applicable to:

- iterative algorithms
- time-step algorithm

~2-3X performance, ~10X less memory bandwidth

Is it time to revisit more algorithms?

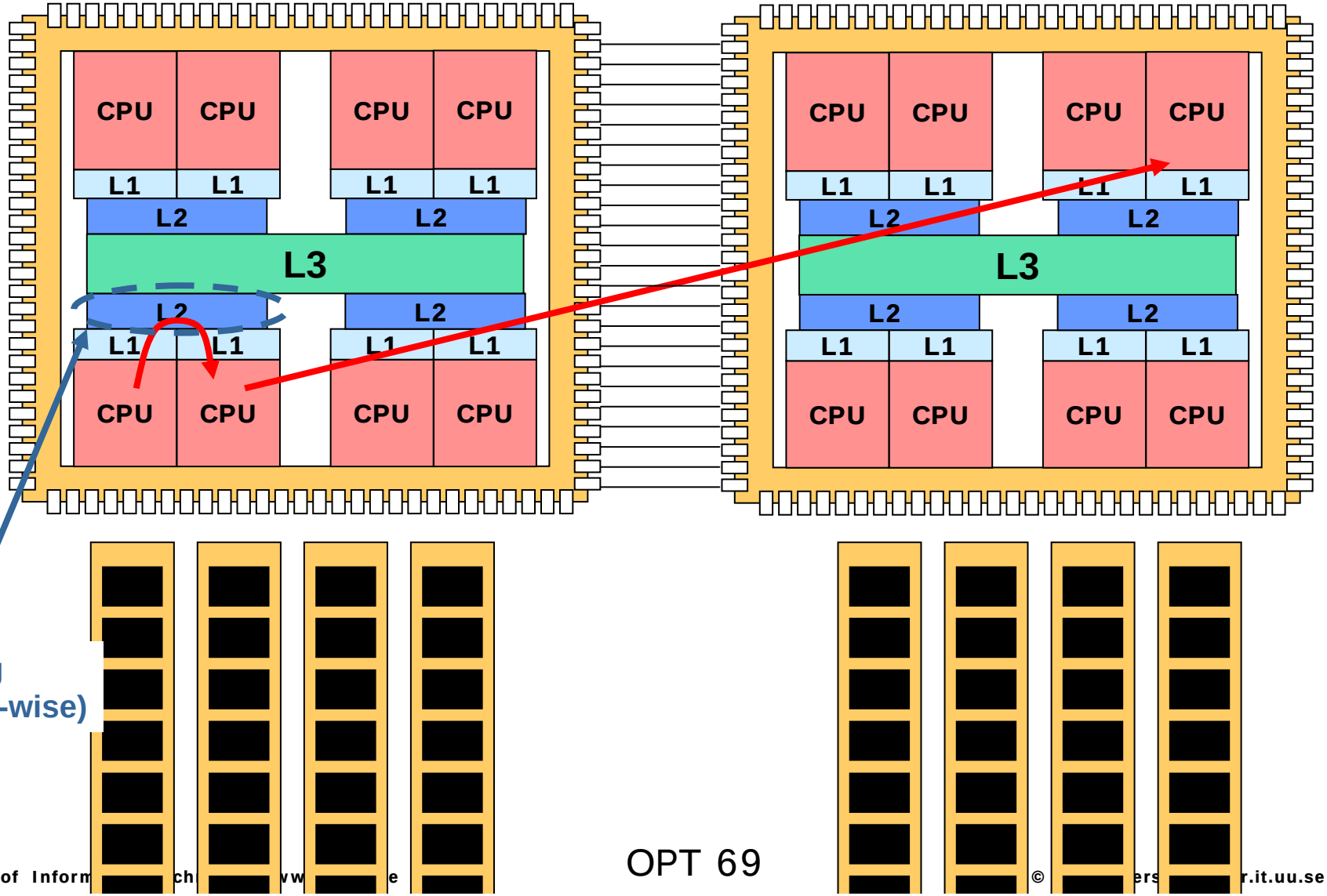
UPPSALA
UNIVERSITET

Multisocket Issues

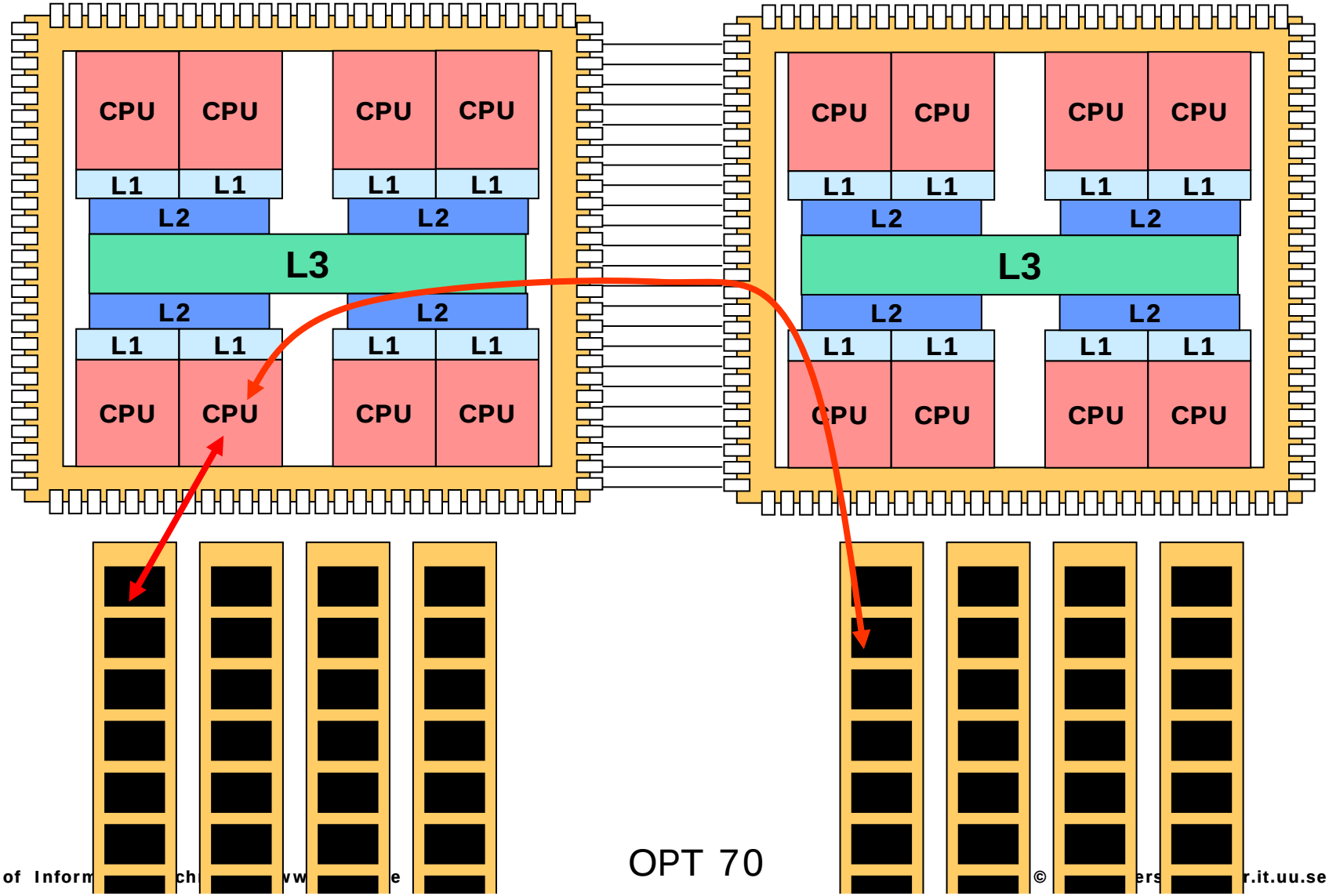
Erik Hagersten
Uppsala University
Sweden

PDC Summer School 2010

Multicore Challenges (Multisocket) Non-uniformity Communication



Multicore Challenges (Multisocket) Non-uniformity Memory

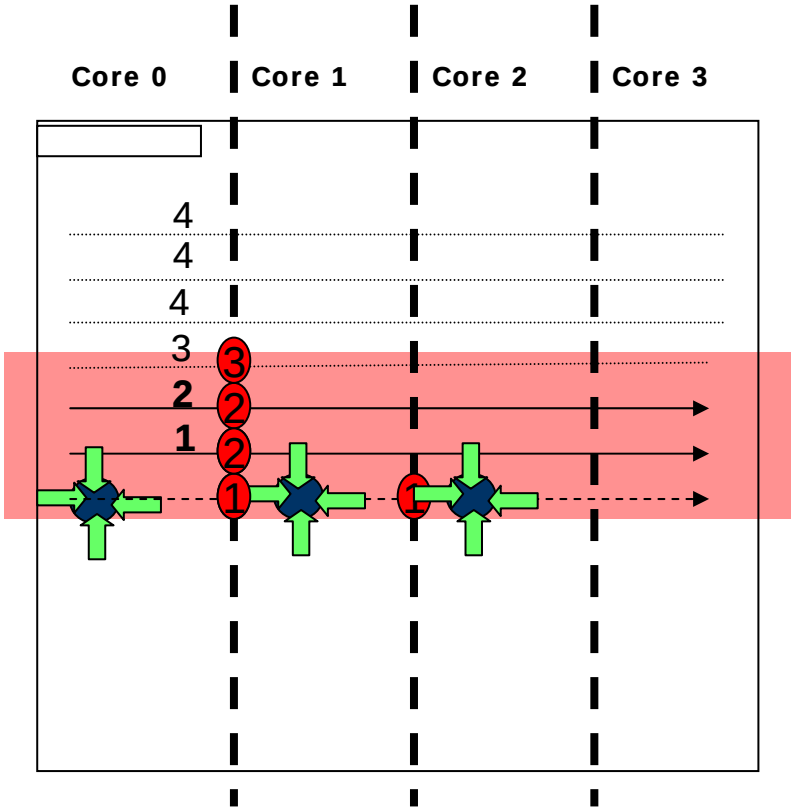


UPPSALA UNIVERSITET

Multisocket optimization

- Bind threads to cores
- Allocate memory by the thread that will use it
  - First touch: OS will try to allocate memory on that socket (if NUMA optimization is supported)
- Minimize inter-socket communication

Example: G-S, temp blocking



= sweep path

= current

= data dependence

1,2,3,4 = iteration number

= cacheline layout

= active region

1 = sync flag iteration no

Demo Time!

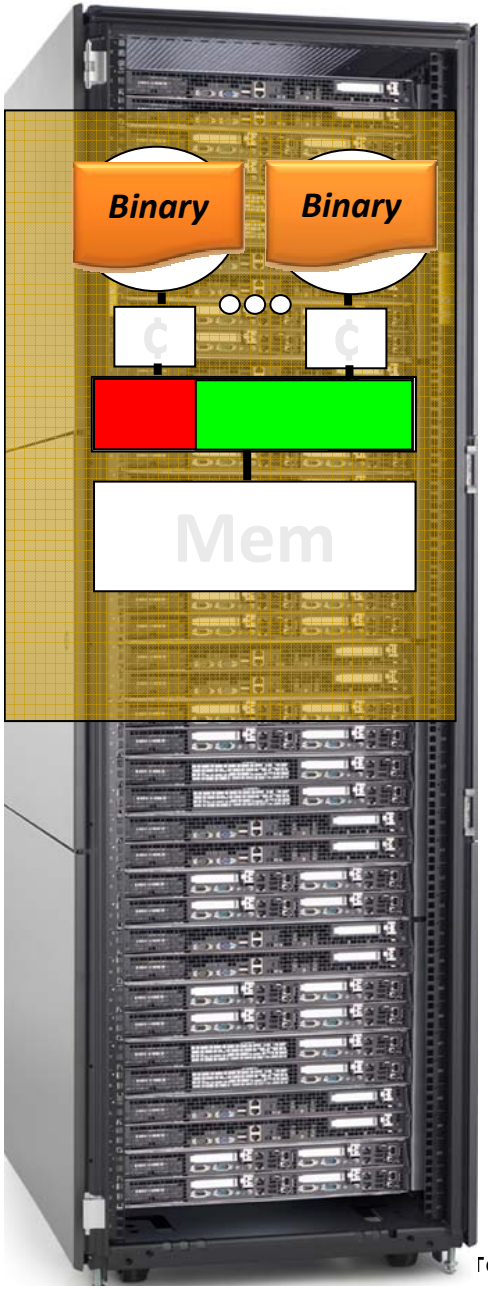
G-S:
Original code
Optimized

UPPSALA
UNIVERSITET

Cache sharing issues

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

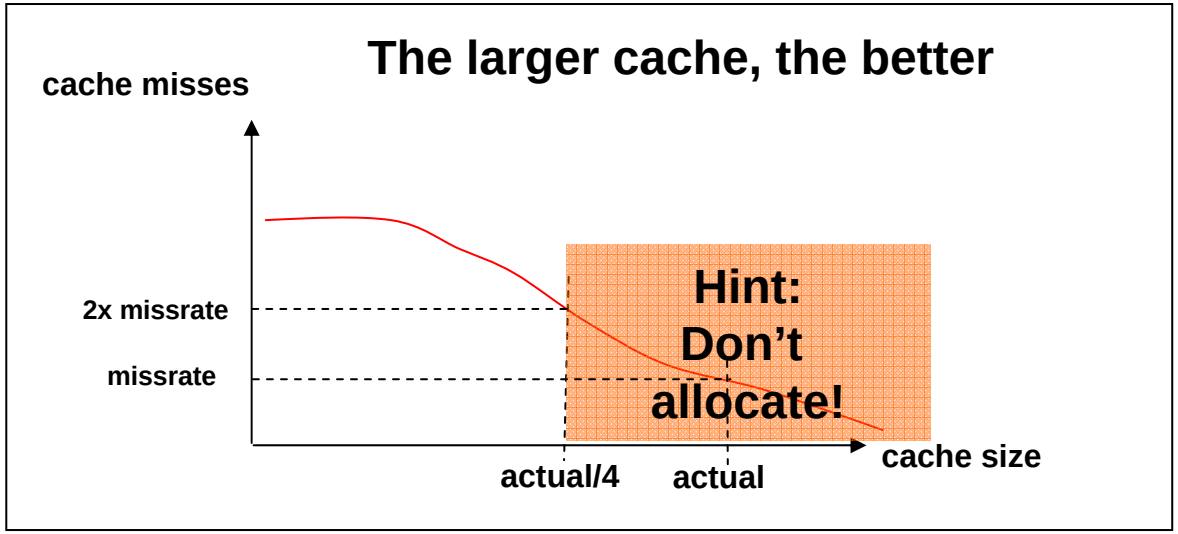
Fighting for shared resources

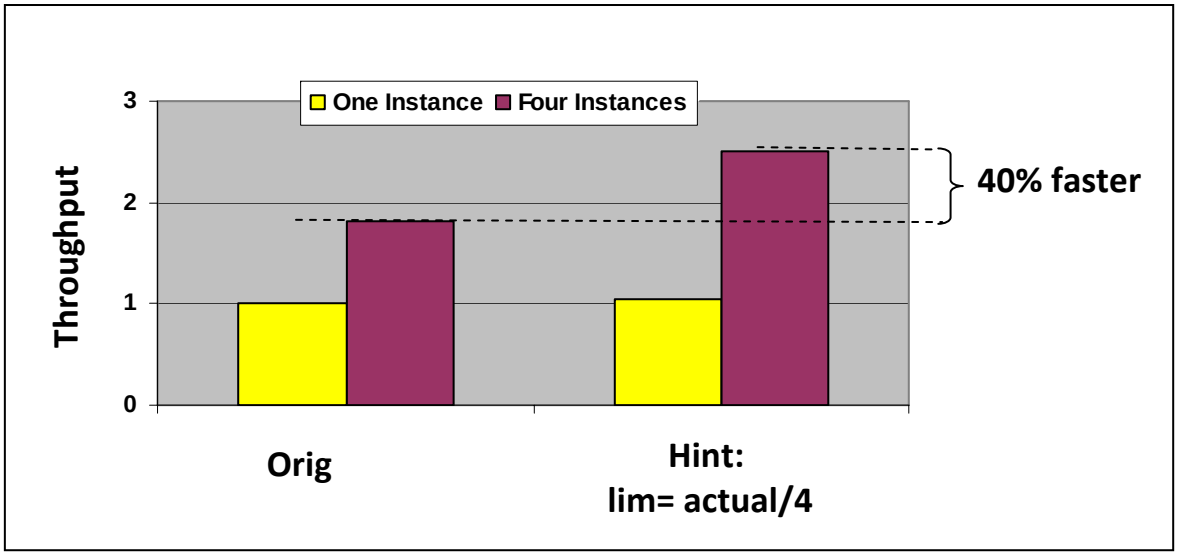


1st Order MC Performance Problems

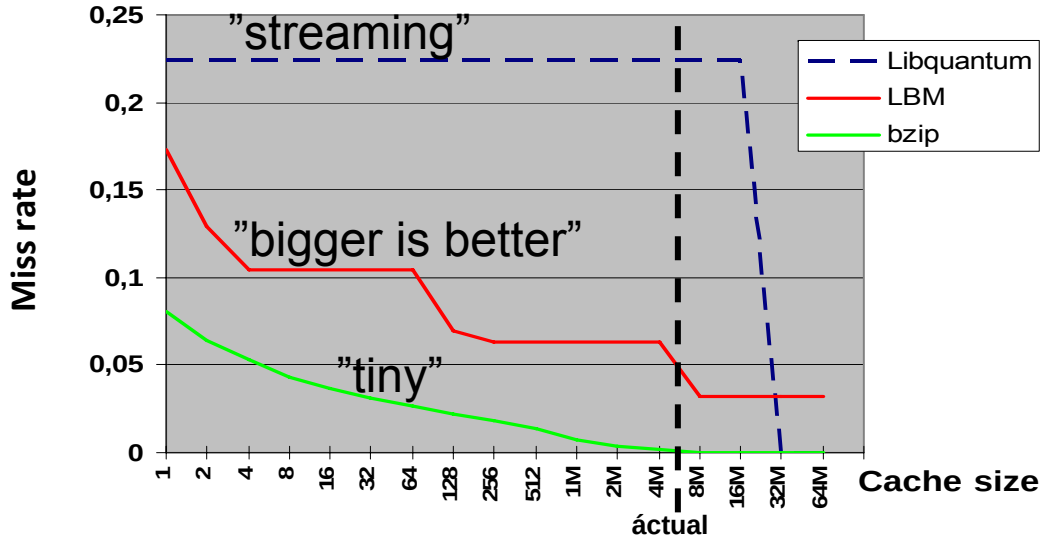
- Additional multicore issues:
  - Even less cache resources per application
  - Sharing of cache resources
  - Wasted cache usage

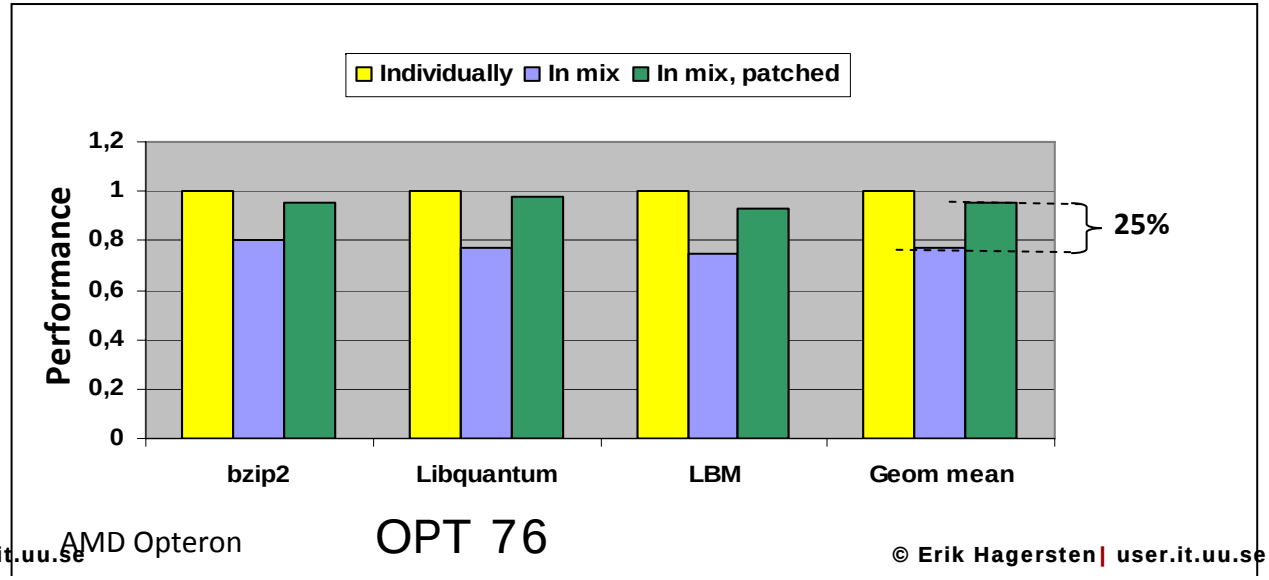
Example: Hints to avoid cache pollution (non-temporal prefetches)





Example: Hints for mixed workloads
(non-temporal prefetches)





AMD Opteron

OPT 76

UPPSALA UNIVERSITET

Some performance tools

Free licenses

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

Not free

- Intel: Vtune and many more
- Alinea, TotalView,… (for MPI…)
- Acumem (of course☺)
- HP: Multicore toolkit (some free, some not)

UPPSALA UNIVERSITET

Summing up: Multicore

- Thread-level parallelism on the chip
- New walls: Bandwidth, locality and parallelism
- Important R&D areas (again)
  - Algorithms
  - Parallelization
  - Verification
  - Modeling/Simulation/Tools
  - Managing data locality
  - Bandwidth optimizations
  - …
- Welcome to the MC club!

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