

Optimizing for Speed

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

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

AVDARK
2010

Dept of Information Technology | www.it.uu.se

OPT 2

© Erik Hagersten | user.it.uu.se/~eh

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
- ...

AVDARK
2010

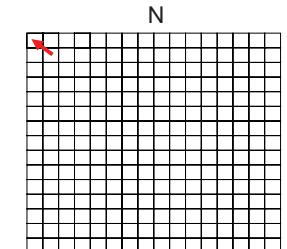
Dept of Information Technology | www.it.uu.se

OPT 3

© Erik Hagersten | user.it.uu.se/~eh

What can go Wrong? A Simple Example...

Perform a diagonal copy 10 times



AVDARK
2010

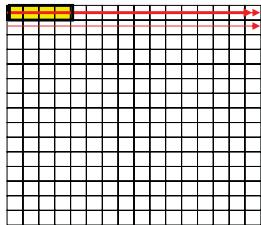
Dept of Information Technology | www.it.uu.se

OPT 4

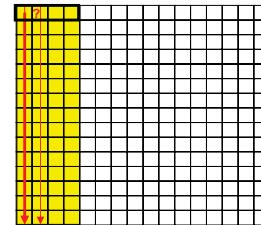
© Erik Hagersten | user.it.uu.se/~eh

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];
    }
}
```

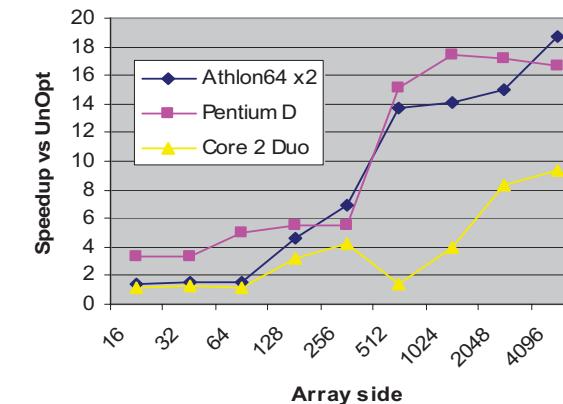
AVDARK
2010

Dept of Information Technology | www.it.uu.se

OPT 5

© Erik Hagersten | user.it.uu.se/~eh

Performance Difference: Loop order

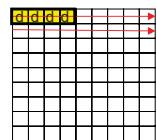


OPT 6

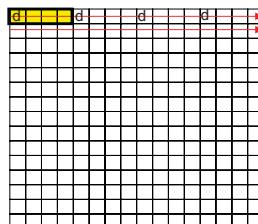
© Erik Hagersten | user.it.uu.se/~eh

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;
    }
}
```

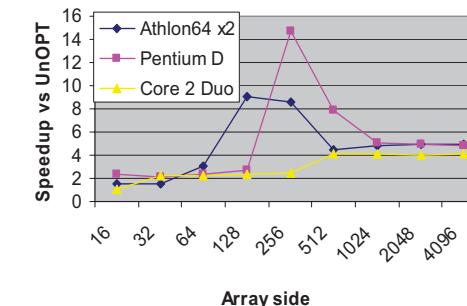
AVDARK
2010

Dept of Information Technology | www.it.uu.se

OPT 7

© Erik Hagersten | user.it.uu.se/~eh

Performance Difference: Sparse Data



OPT 8

© Erik Hagersten | user.it.uu.se/~eh

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;
```

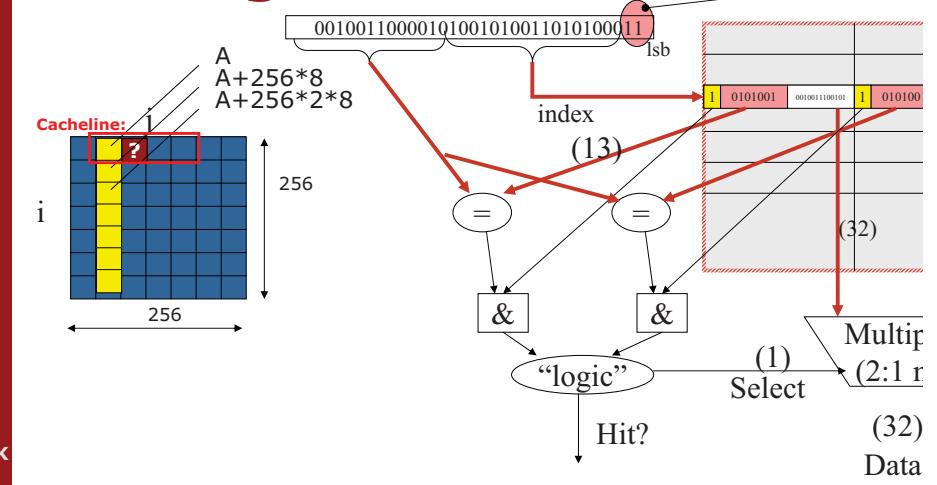
AVDARK
2010

Dept of Information Technology | www.it.uu.se

OPT 9

© Erik Hagersten | user.it.uu.se/~eh

Padding of data structures

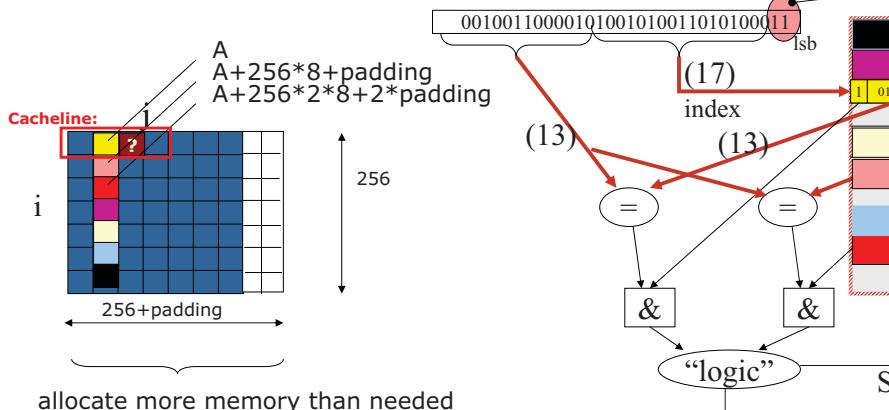


OPT 10

© Erik Hagersten | user.it.uu.se/~eh

Dept of Information Technology | www.it.uu.se

Padding of data structures

AVDARK
2010

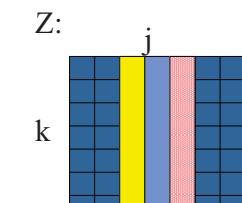
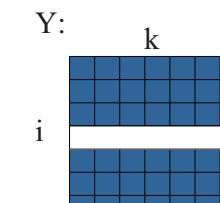
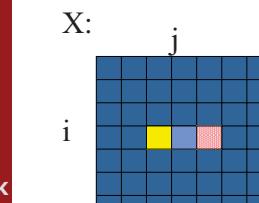
Dept of Information Technology | www.it.uu.se

OPT 11

© Erik Hagersten | user.it.uu.se/~eh

Blocking

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



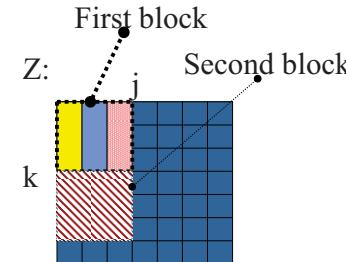
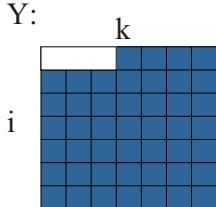
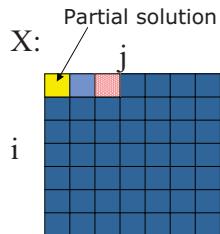
Dept of Information Technology | www.it.uu.se

OPT 12

© Erik Hagersten | user.it.uu.se/~eh

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;
        };
    }
```



OPT 13

© Erik Hagersten | user.it.uu.se/~eh

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)

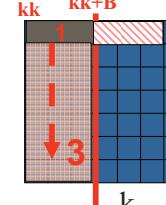
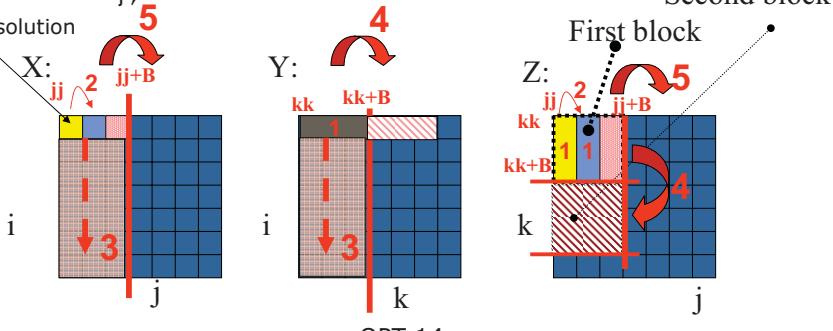
OPT 15

© Erik Hagersten | user.it.uu.se/~eh

Blocking: the Movie!

```
/* 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;
        };
    }
```

Partial solution



OPT 14

© Erik Hagersten | user.it.uu.se/~eh

Cache Affinity

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

OPT 16

© Erik Hagersten | user.it.uu.se/~eh

Optimize for "other caches"

■ TLB

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

...

Commercial Break: Acumem's Multicore Tools

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

Acumem SlowSpotter™

Source:
C, C++, Fortran, OpenMP...

Mission:
Find the SlowSpots™
Asses their importance
Enable for non-experts to fix them
Improve the productivity of performance experts

Any Compiler



Finger
Print
(~4MB)

Acumem SlowSpotter™

Source:
C, C++,

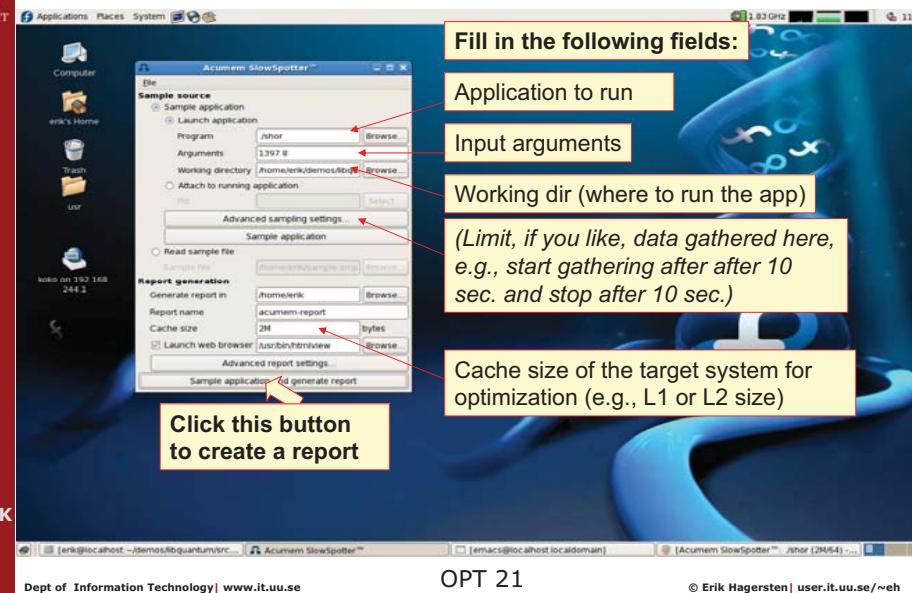
What?

How?

Where?

Help!

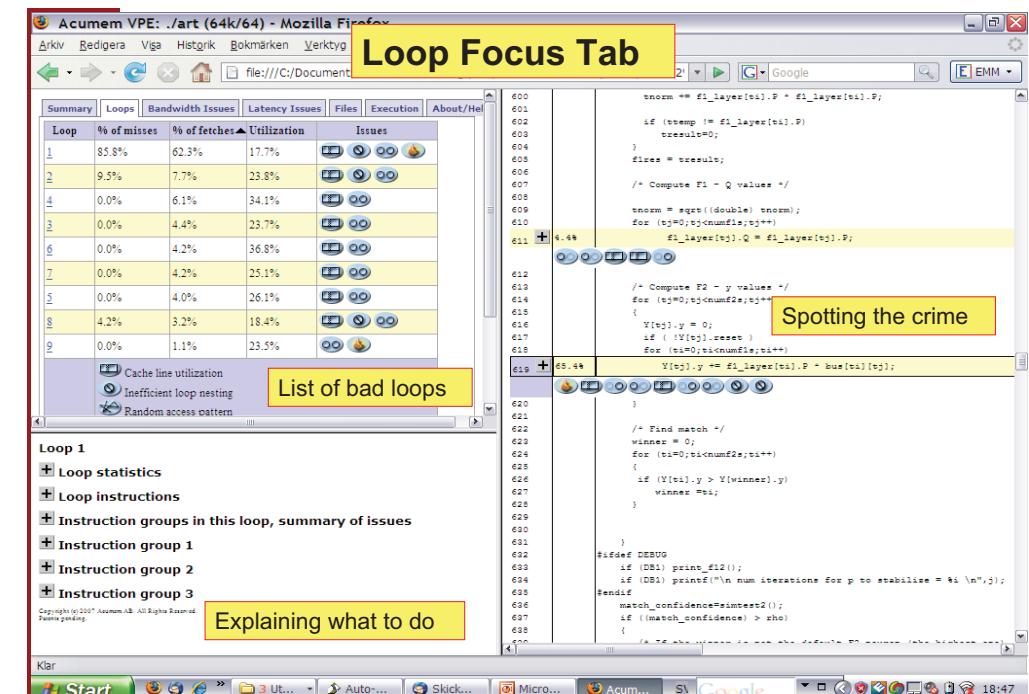
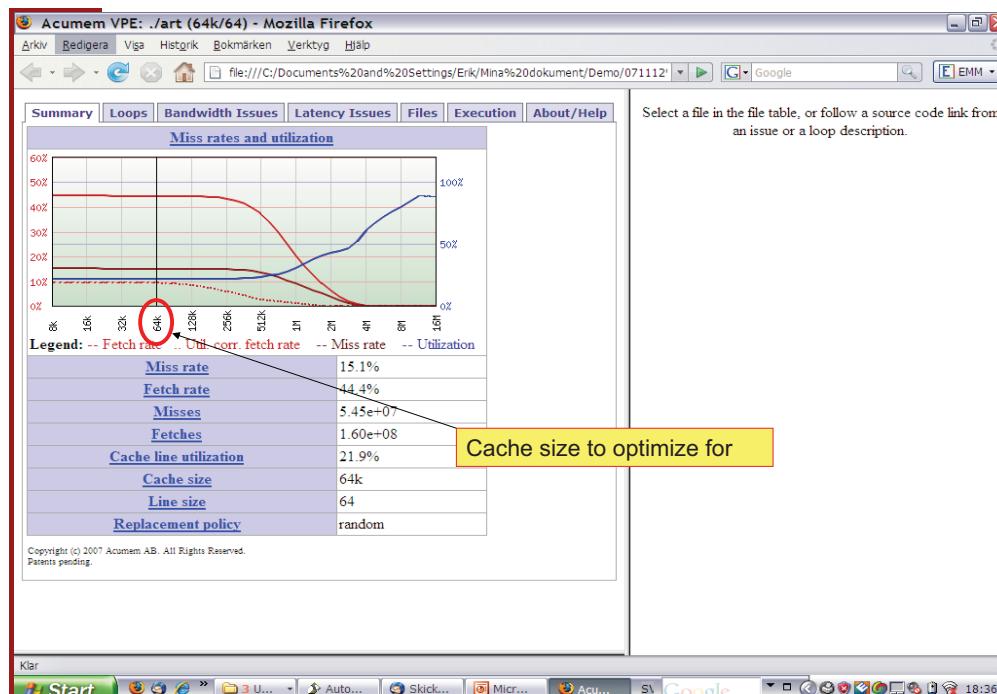
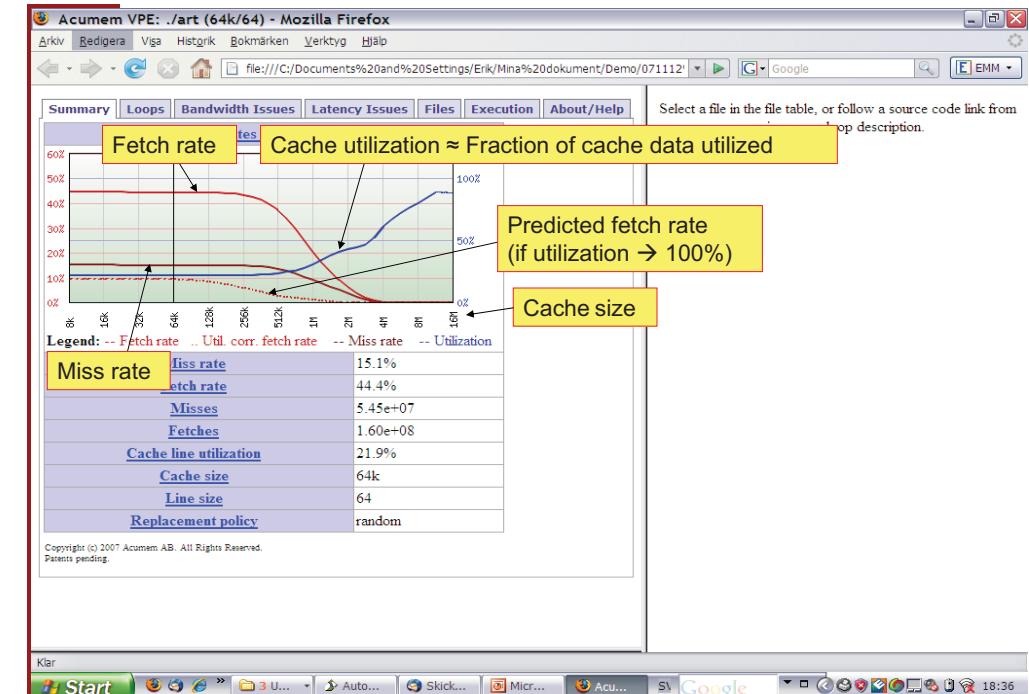
A One-Click Report Generation


 AVDARK
2010

Dept of Information Technology | www.it.uu.se

OPT 21

© Erik Hagersten | user.it.uu.se/~eh



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

Arkiv Redigera Viga Histgrik Bokmärken Verktyg file:///C:/Document

Bandwidth Focus Tab

page EMM

Summary	Loops	Bandwidth Issues	Latency Issues	Files	Execution	About/Help
Loop / Issue	Summary	% 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	4.2%	25.1%	100.0%	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

Copyright © 2007 Acumem AB. All Rights Reserved.
Software pending

Explaining what to do

```
tmax += f1_layer[t1].P * f1_layer[t1].P;
if (tmax != f1_layer[t1].P)
    tresult=0;
}
tresult = tresult;

/* Compute F1 = Q values */

tmax = sqrt((double) tmax);
for (tj=0; tj<numf1s; tj++)
{
    f1_layer[tj].Q = f1_layer[tj].P;

    /* Compute F2 = y values */
    for (tj=0; tj<numf2s; tj++)
    {
        Y[tj].y = 0;
        if (!Y[tj].reset)
            for (ti=0; ti<numf1s; ti++)
                Y[tj].y += f1_layer[ti].P * bus[ti][tj];
    }

    /* Find match */
    winner = 0;
    for (t1=0; t1<numf2s; t1++)
    {
        if (Y[t1].y > Y[winner].y)
            winner = t1;
    }

    #ifdef DEBUG
    (DBL) printf("%d\n");
    if (DBL) printf("In num iterations for p to stabilize = %d\n");
    #endif
    match_confidence+=match2();
    if ((match_confidence) > xlo)
    {
        /* If the winner is not the default P0 source (the highest
           priority), then we have found a match. */
        if (winner != 0)
            match_confidence+=1;
    }
}
```

Klar

Start Inkorgen f... 071215 Ac... Acumem ... Google

Resource Sharing Example

Libquantum

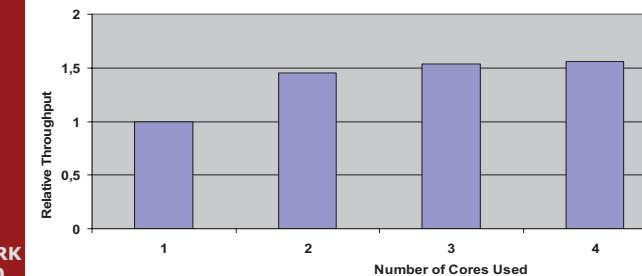
A quantum computer simulation

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

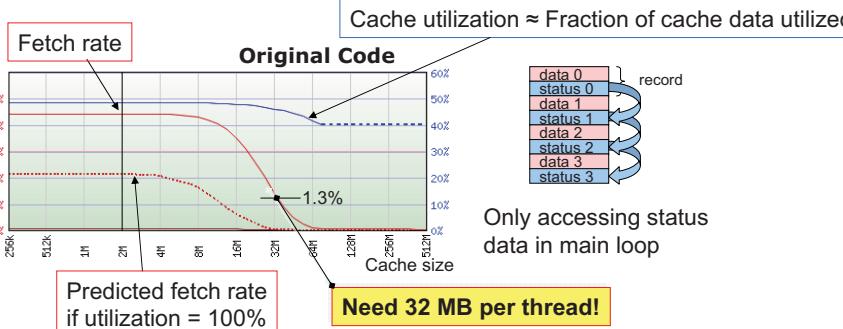


OPT 20

© Erik Hagersten | user.it.uu.se 26

Utilization Analysis

Libquantum



SlowSpotter's First Advice: Improve Utilization

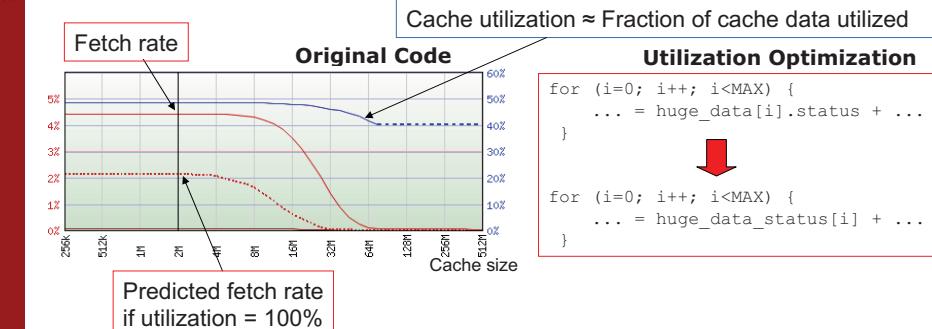
→ Change one data structure

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

**AVDARK
2010**

Utilization Analysis

Libquantum



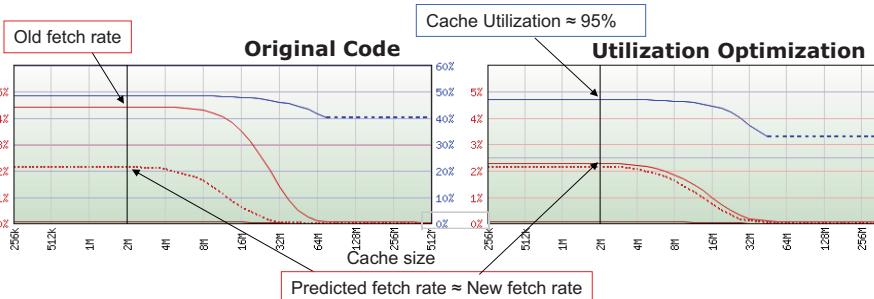
SlowSpotter's First Advice: Improve Utilization

→ Change one data structure

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

AVDA
201

After Utilization Optimization

Libquantum

**AVDARK
2010**

 Dept of Information Technology | www.it.uu.se
OPT 29

 © Erik Hagersten | user.it.uu.se/~eh

Utilization Optimization

**AVDARK
2010**

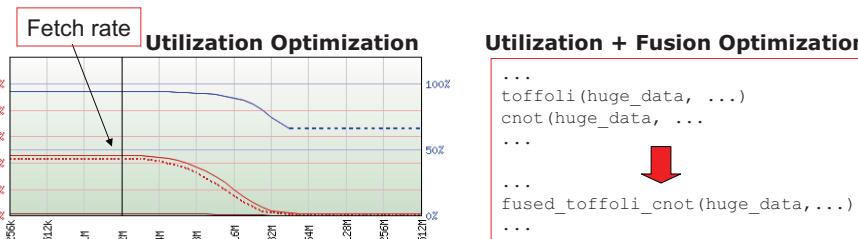
 Dept of Information Technology | www.it.uu.se
OPT 30

 © Erik Hagersten | user.it.uu.se/~eh

Two positive effects from better utilization

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

Reuse Analysis

Libquantum

**AVDARK
2010**

Second-Fifth SlowSpotter Advice: Improve reuse of data

➔ Fuse functions traversing the same data

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

OPT 31

 © Erik Hagersten | user.it.uu.se/~eh

Effect: Reuse Optimization

SPEC CPU2006-462.libquantum
**AVDARK
2010**

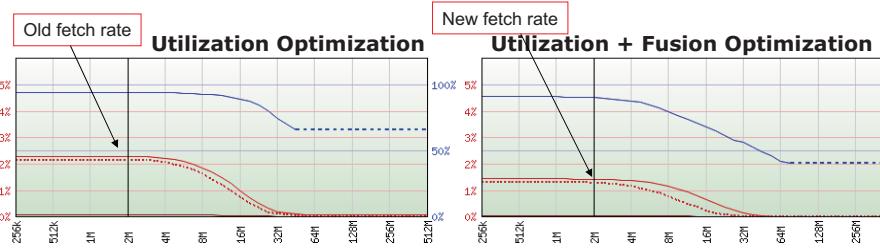
 Dept of Information Technology | www.it.uu.se
OPT 32

 © Erik Hagersten | user.it.uu.se/~eh

- 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

AVDARK
2010

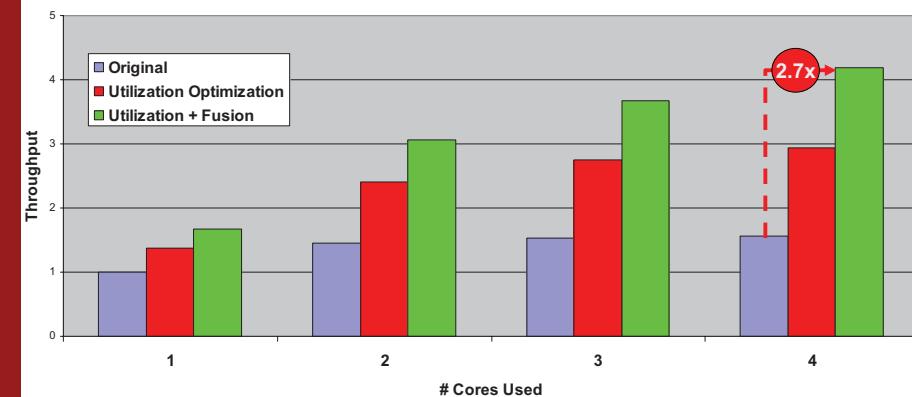
Dept of Information Technology | www.it.uu.se

OPT 33

© Erik Hagersten | user.it.uu.se/~eh

Summary

Libquantum

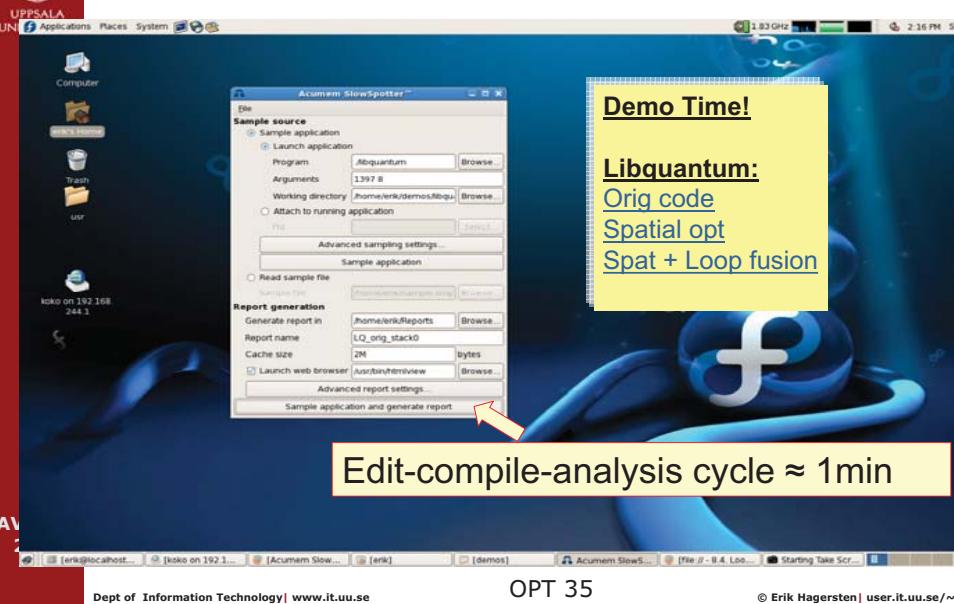


Dept of Information Technology | www.it.uu.se

OPT 34

© Erik Hagersten | user.it.uu.se/~eh

Demo



AV

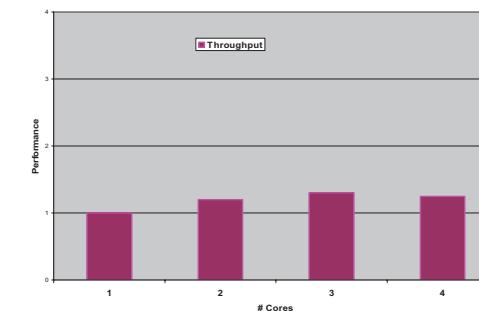
Dept of Information Technology | www.it.uu.se

OPT 35

© Erik Hagersten | user.it.uu.se/~eh

Original Cigar Throughput

UPPSALA
UNIVERSITET



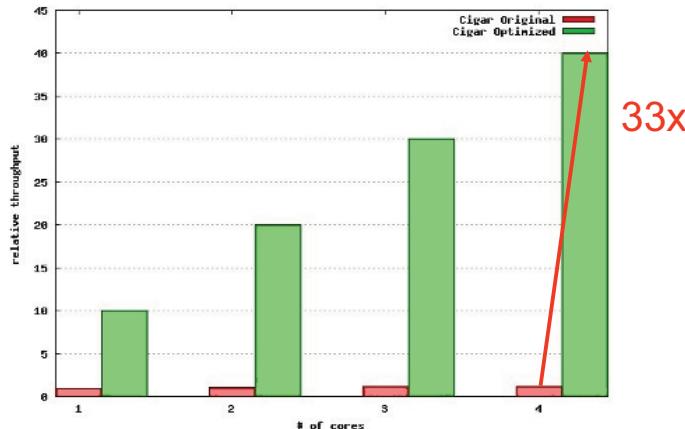
Throughput scalability is a different way to look at the performance of an application. Here, several single-threaded instances of the application are run at the same time. Even though the different instances do not explicitly depend on each other, they will nevertheless fight over the shared resources, 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.

OPT 36

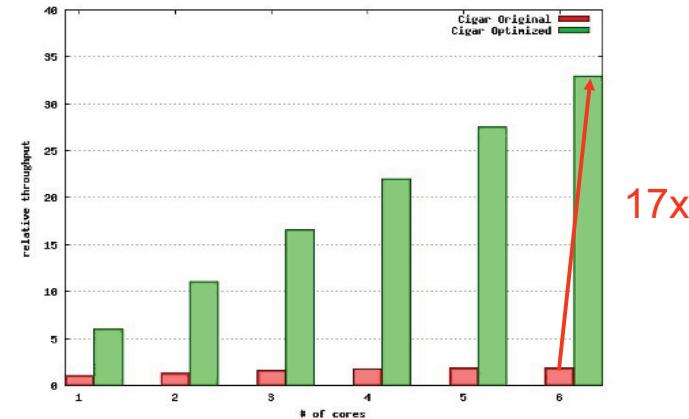
© Erik Hagersten | user.it.uu.se/~eh

Throughput Performance Intel Core2 (Intel Xeon E5345)



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

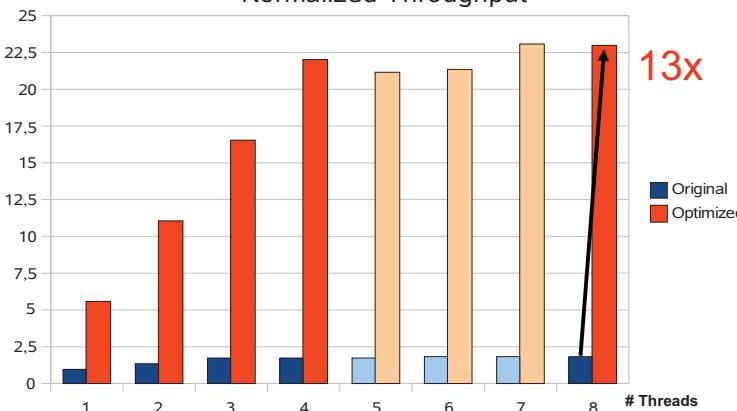
Throughput Performance (AMD's Istanbul)



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

Throughput Performance (Intel i7)

Normalized Throughput

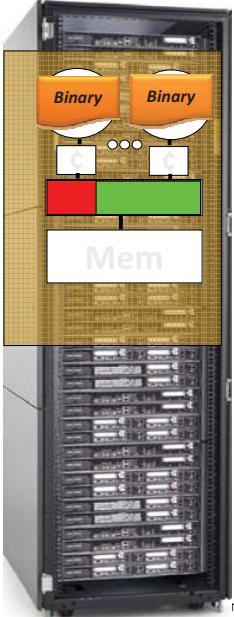


Intel's new four-core i7 (Nehalem) processor enjoy a 13x better throughput due to the Optimization on four cores. Note that each core can run up to two threads.

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

OPT 41

© Erik Hagersten | user.it.uu.se/~eh

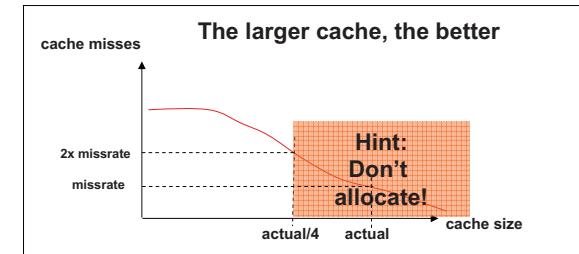
Mem

Technology | www.it.uu.se

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

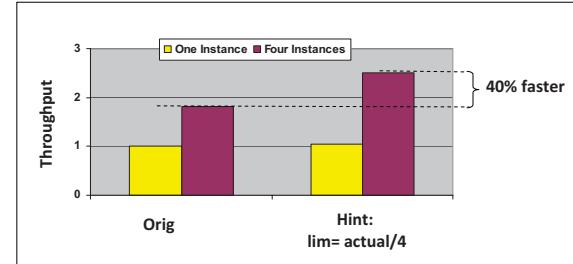


Dept of Information Technology | www.it.uu.se



OPT 42

© Erik Hagersten | user.it.uu.se/~eh

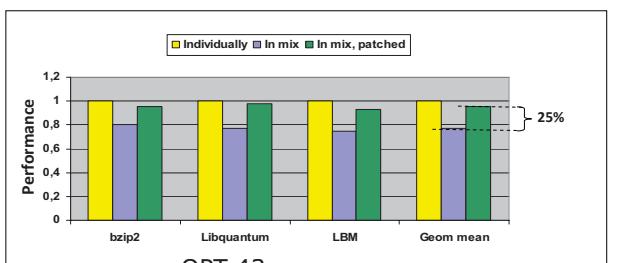
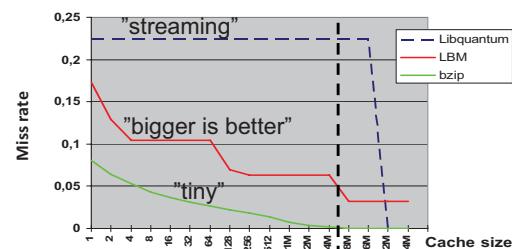


Example: Hints for mixed workloads (non-temporal prefetches)



AVDARK
2010

Dept of Information Technology | www.it.uu.se



OPT 43

© Erik Hagersten | user.it.uu.se/~eh

AMD Opteron

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)

OPT 44

© Erik Hagersten | user.it.uu.se/~eh

AVDARK
2010

Dept of Information Technology | www.it.uu.se