

Summary, global statistics

The screenshot displays a web-based performance analysis tool. The browser address bar shows `http://localhost:45570/session/main.html` and the page title is `ThreadSpotter: test2 (2M/64)`. The main content area is divided into a left sidebar and a right main panel. The sidebar, highlighted with a red border, contains a navigation menu with tabs for **Issues**, **Loops**, **Summary**, **Files**, **Execution**, and **About/Help**. The **Summary** tab is active, showing global statistics and analysis parameters.

Global statistics	
Accesses	2.22e+008
Misses	1.19e+006
Fetches	4.34e+007
Write-backs	1.86e+004
Upgrades	0.00e+000
Miss ratio	0.5%
Fetch ratio	19.5%
Write-back ratio	0.0%
Upgrade ratio	0.0%
Communication ratio	0.0%
Fetch utilization	16.5%
Write-back utilization	49.0%
Communication utilization	100.0%

Analysis parameters	
Processor model	Intel(R) Core(TM)2 CPU T7200 @ 2.00GHz (auto)
Number of CPUs	1

The right main panel contains three charts:

- Miss/Fetch ratio**: A line graph showing the percentage of misses relative to fetches. The y-axis ranges from 0% to 25%. The x-axis shows memory addresses from 256k to 512M. A solid red line represents the 'Fetch ratio' (19.5%), a dotted red line represents the 'Utilization corrected fetch ratio', and a dashed red line represents the 'Miss ratio' (0.5%).
- Write-back ratio**: A line graph showing the percentage of write-backs. The y-axis ranges from 0.000% to 0.015%. The x-axis shows memory addresses from 256k to 512M. A solid black line represents the 'Write-back ratio' (0.0%), and a dotted black line represents the 'Utilization corrected write-back ratio'.
- Utilization**: A line graph showing the percentage of utilization. The y-axis ranges from 0% to 60%. The x-axis shows memory addresses from 256k to 512M. A solid red line represents 'Fetch utilization' (16.5%), and a solid green line represents 'Write-back utilization' (49.0%).

Placeholder text: "Placeholder. Click on an issue, loop or file" is visible in the main panel and at the bottom of the page.

Report sections

The screenshot displays the ThreadSpotter application interface. At the top, a navigation bar contains tabs for 'Issues', 'Loops', 'Summary', 'Files', 'Execution', and 'About/Help'. Below this, a sub-navigation bar highlights 'Bandwidth Issues', 'Latency Issues', 'Multi-Threading Issues', and 'Pollution Issues'. A table lists various issues with columns for '#', 'Issue type', 'Filter', '% of bandwidth', '% of fetches', '% of write-backs', 'Fetch utilization', and 'Write back utilization'. Issue #4, 'Fetch utilization', is selected and expanded to show detailed statistics and instructions. The right side of the interface shows a code editor with C++ code, including a function 'add_one' and a loop that iterates over a vector of cars, performing a switch statement on the query type.

#	Issue type	Filter: All	% of bandwidth	% of fetches	% of write-backs	Fetch utilization	Write back utilization
4	Fetch utilization		48.3%	48.3%	0.0%	14.3%	100.0%
7	Spat/temp blocking		48.3%	48.3%	0.0%	14.3%	100.0%
5	Fetch utilization		47.9%	48.0%	0.0%	15.3%	100.0%
9	Spat/temp blocking		47.9%	48.0%	0.0%	15.3%	100.0%
6	Fetch utilization		3.0%	3.0%	0.0%	61.6%	100.0%
10	Spat/temp blocking		3.0%	3.0%	0.0%	61.6%	100.0%
8	Spat/temp blocking		0.1%	0.0%	96.5%	22.8%	22.4%

Issue #4: Fetch utilization

This instruction group also shows symptoms of: Fetch hot-spot.

- Statistics for instructions of this issue
- Instructions involved in this issue
- Loop statistics
- Loop instructions

Copyright (c) 2006-2011 Rogue Wave Software, Inc. All Rights Reserved.
Patents pending.

Placeholder. Click on an issue, loop or file

```
18 | };
19 |
20 |
21 |
22 | void database_2_vector_t::add_one(const car_t &c)
23 | {
24 |     cars.push_back(c);
25 | }
26 |
27 | void database_2_vector_t::finalize_adding()
28 | {
29 |     // std::sort(cars.begin(), cars.end());
30 | }
31 |
32 | void database_2_vector_t::ask_one_question(query
33 | {
34 |     cars_t::const_iterator i = cars.begin(), e =
35 |     for (; i!=e; i++) {
36 |         switch (query.query_type) {
37 |             case 0: // count matching colors
38 |                 if (i->color == query.car.color)
39 |                     query.result++;
40 |                 break;
41 |             case 1: // count same model but heavier
42 |                 if (i->model == query.car.model && i
43 |                     query.result++;
44 |                 break;
45 |             }
46 |         }
47 |     }
48 |     #endif
49 | }
```

Copyright (c) 2006-2011 Rogue Wave Software, Inc. All Rights Reserved.

Navigation by issues

The screenshot shows the ThreadSpotter application interface. At the top, there is a browser address bar with the URL `http://localhost:45570/session/main.html` and a window title `ThreadSpotter: test2 (2M/64)`. Below the browser bar is a navigation menu with tabs: **Issues**, **Loops**, **Summary**, **Files**, **Execution**, and **About/Help**. The **Issues** tab is active, showing a table of issues. A red box highlights the table. Below the table, the details for **Issue #4: Fetch utilization** are displayed, including a description and several expandable sections: **Statistics for instructions of this issue**, **Instructions involved in this issue**, **Loop statistics**, and **Loop instructions**. On the right side of the interface, a code editor displays C++ code. The code includes functions like `add_one`, `finalize_adding`, and `ask_one_question`. The code editor has a vertical line indicating the current position in the code, which corresponds to the highlighted issue in the table.

#	Issue type	Filter: All	Multi-Threading Issues				Pollution Issues	
			% of bandwidth	% of fetches	% of write-backs	Fetch utilization	Write-back utilization	
4	Fetch utilization		48.3%	48.3%	0.0%	14.3%	100.0%	
7	Spat/temp blocking		48.3%	48.3%	0.0%	14.3%	100.0%	
5	Fetch utilization		47.9%	48.0%	0.0%	15.3%	100.0%	
9	Spat/temp blocking		47.9%	48.0%	0.0%	15.3%	100.0%	
6	Fetch utilization		3.0%	3.0%	0.0%	61.6%	100.0%	
10	Spat/temp blocking		3.0%	3.0%	0.0%	61.6%	100.0%	
8	Spat/temp blocking		0.1%	0.0%	96.5%	22.8%	22.4%	

Issue #4: Fetch utilization

This instruction group also shows symptoms of: Fetch hot-spot.

- Statistics for instructions of this issue**
- Instructions involved in this issue**
- Loop statistics**
- Loop instructions**

Copyright (c) 2006-2011 Rogue Wave Software, Inc. All Rights Reserved.
Patents pending.

Placeholder. Click on an issue, loop or file

```
18 | };
19 |
20 |
21 | void database_2_vector_t::add_one(const car_t &c)
22 | {
23 |     cars.push_back(c);
24 | }
25 |
26 | void database_2_vector_t::finalize_adding()
27 | {
28 |     // std::sort(cars.begin(), cars.end());
29 | }
30 |
31 | void database_2_vector_t::ask_one_question(query
32 | {
33 |     cars_t::const_iterator i = cars.begin(), e =
34 |     for (; i!=e; i++) {
35 |         switch (query.query_type) {
36 |             case 0: // count matching colors
37 |                 if (i->color == query.car.color)
38 |                     query.result++;
39 |                     break;
40 |             case 1: // count same model but heavier
41 |                 if (i->model == query.car.model && i
42 |                     query.result++;
43 |                     break;
44 |             }
45 |         }
46 |     }
47 | }
48 | #endif
49 |
```

Copyright (c) 2006-2011 Rogue Wave Software, Inc. All Rights Reserved.

Source code annotation

The screenshot displays the ThreadSpotter application interface. At the top, there are navigation tabs: Issues, Loops, Summary, Files, Execution, and About/Help. Below these are sub-tabs for Bandwidth Issues, Latency Issues, Multi-Threading Issues, and Pollution Issues. A table lists various issues with columns for #, Issue type, % of bandwidth, % of fetches, % of write-backs, Fetch utilization, and Write-back utilization. Issue #4, 'Fetch utilization', is highlighted. Below the table, there is a detailed view for Issue #4, including a description and expandable sections for statistics, instructions, loop statistics, and loop instructions. On the right side, a source code editor shows the implementation of the database_2_vector_t class. A red box highlights a section of code (lines 31-46) corresponding to the 'ask_one_question' method, where the 'Fetch utilization' issue is annotated with a red 'F' icon and a percentage of 48.9%.

#	Issue type	% of bandwidth	% of fetches	% of write-backs	Fetch utilization	Write-back utilization
4	Fetch utilization	48.3%	48.3%	0.0%	14.3%	100.0%
7	Spat/temp blocking	48.3%	48.3%	0.0%	14.3%	100.0%
5	Fetch utilization	47.9%	48.0%	0.0%	15.3%	100.0%
9	Spat/temp blocking	47.9%	48.0%	0.0%	15.3%	100.0%
6	Fetch utilization	3.0%	3.0%	0.0%	61.6%	100.0%
10	Spat/temp blocking	3.0%	3.0%	0.0%	61.6%	100.0%
8	Spat/temp blocking	0.1%	0.0%	96.5%	22.8%	22.4%

Issue #4: Fetch utilization

This instruction group also shows symptoms of: Fetch hot-spot.

- Statistics for instructions of this issue
- Instructions involved in this issue
- Loop statistics
- Loop instructions

```
void database_2_vector_t::ask_one_question(query_t &q)
{
    cars_t::const_iterator i = cars.begin(), e = cars.end();
    for (; i!=e; i++) {
        switch (query.query_type) {
            case 0: // count matching colors
                if (i->color == query.car.color)
                    query.result++;
                    break;
            case 1: // count same model but heavier
                if (i->model == query.car.model && i->weight > query.car.weight)
                    query.result++;
                    break;
        }
    }
}
```

Navigation by loops

The screenshot shows a performance analysis tool interface. At the top, there is a table with columns: Issues, Loops, Summary, Files, Execution, and About/Help. The 'Loops' column is expanded to show a table with columns: Loop, % of misses, % of fetches, Fetch utilization, Write-back utilization, and Issues. Three loops are listed, with Loop 2 highlighted in yellow.

Loop	% of misses	% of fetches	Fetch utilization	Write-back utilization	Issues
2	55.0%	48.9%	14.3%	100.0%	[F] [PI] [ST] [NT]
1	42.1%	48.0%	15.6%	100.0%	[F] [PI] [ST] [NT]
3	2.2%	3.0%	63.6%	100.0%	[F] [PI] [ST] [NT]

Below this table, a detailed view for 'Loop 2' is shown. It includes sections for 'Loop statistics', 'Loop instructions', and 'Bandwidth issues related to this loop'. A table of bandwidth issues is displayed:

#	Issue type	% of bandwidth	% of fetches	% of write-backs	Fetch utilization	Write-back utilization
7	Spat/temp blocking	48.3%	48.3%	0.0%	14.3%	100.0%
4	Fetch utilization	48.3%	48.3%	0.0%	14.3%	100.0%

On the right side, C++ source code is displayed. Performance metrics are overlaid on the code. Line 37 shows a loop with a 48.9% bandwidth issue (Fetch utilization). Line 41 shows a loop with a 51.0% bandwidth issue (Fetch utilization). The code includes functions like `add_one`, `finalize_adding`, and `ask_one_question`.

Copyright (c) 2006-2011 Rome Wave Software, Inc. All Rights Reserved

Issue details

The screenshot shows the ThreadSpotter interface with the 'Issues' tab selected. A table lists two issues: 'Fetch utilization' and 'Spat/temp blocking', both with 48.3% bandwidth and 14.3% fetch utilization. The 'Issue #4: Fetch utilization' details are highlighted with a red box. It shows a 'Fetch hot-spot' and a table of instructions involved in the issue. The instruction table has columns for Stack, Instruction, % of misses, % of fetches, Fetch ratio, Fetch utilization, and W-B Utilization. The instruction at stack frame 1 is highlighted in yellow.

#	Issue type	% of bandwidth	% of fetches	% of write-backs	Fetch utilization
4	Fetch utilization	48.3%	48.3%	0.0%	14.3%
Z	Spat/temp blocking	48.3%	48.3%	0.0%	14.3%

Issue #4: Fetch utilization

This instruction group also shows symptoms of: Fetch hot-spot.

+ Statistics for instructions of this issue

- Instructions involved in this issue

Stack	Instruction	% of misses	% of fetches	Fetch ratio	Fetch utilization	W-B Utilization
-	"test2"!execute()+0x23 (0x80494a3), driver.cc:35					
-	"test2"!ask_questions()+0x27 (0x804b957), database.hh:57					
-	"test2"!ask_one_question()+0x37 (0x804a177) [R]	43.8%	48.3%	43.6%	14.3%	100.0%
-	database_2_vector.hh:37					

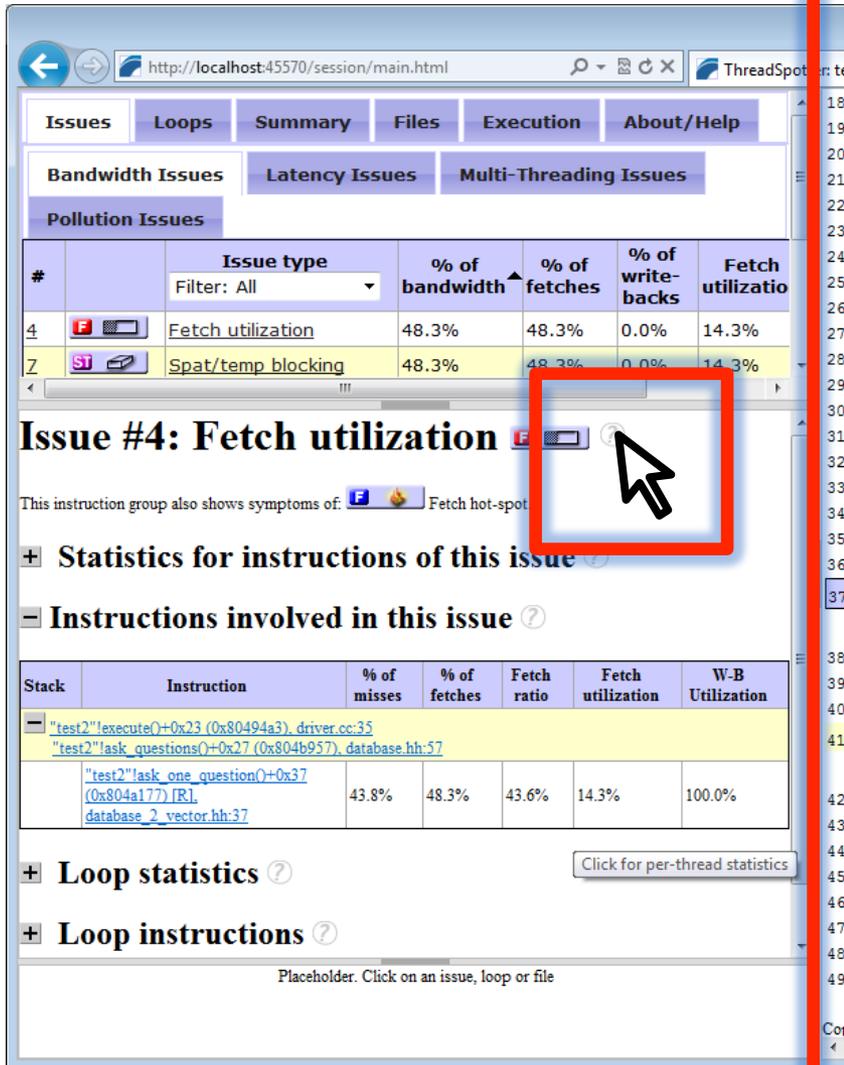
+ Loop statistics Click for per-thread statistics

+ Loop instructions

```
18 };
19
20
21 void database_2_vector_t::add_one(const car_t &c)
22 {
23     cars.push_back(c);
24 }
25
26 void database_2_vector_t::finalize_adding()
27 {
28     // std::sort(cars.begin(), cars.end());
29 }
30
31 void database_2_vector_t::ask_one_question(query_t &query) const
32 {
33     cars_t::const_iterator i = cars.begin(), e = cars.end();
34     for (; i!=e; i++) {
35         switch (query.query_type) {
36             case 0: // count matching colors
37                 if (i->color == query.car.color)
38                     query.result++;
39                 break;
40             case 1: // count same model but heavier
41                 if (i->model == query.car.model && i->weight > query
42                     query.result++;
43                 break;
44             }
45         }
46     }
47 }
48 #endif
49
```

Copyright (c) 2006-2011 Rogue Wave Software, Inc. All Rights Reserved.

Context sensitive help



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

Bandwidth Issues | Latency Issues | Multi-Threading Issues

Pollution Issues

#	Issue type	% of bandwidth	% of fetches	% of write-backs	Fetch utilization
4	Fetch utilization	48.3%	48.3%	0.0%	14.3%
Z	Spat/temp blocking	48.3%	48.3%	0.0%	14.3%

Issue #4: Fetch utilization

This instruction group also shows symptoms of:  Fetch hot-spot

Statistics for instructions of this issue

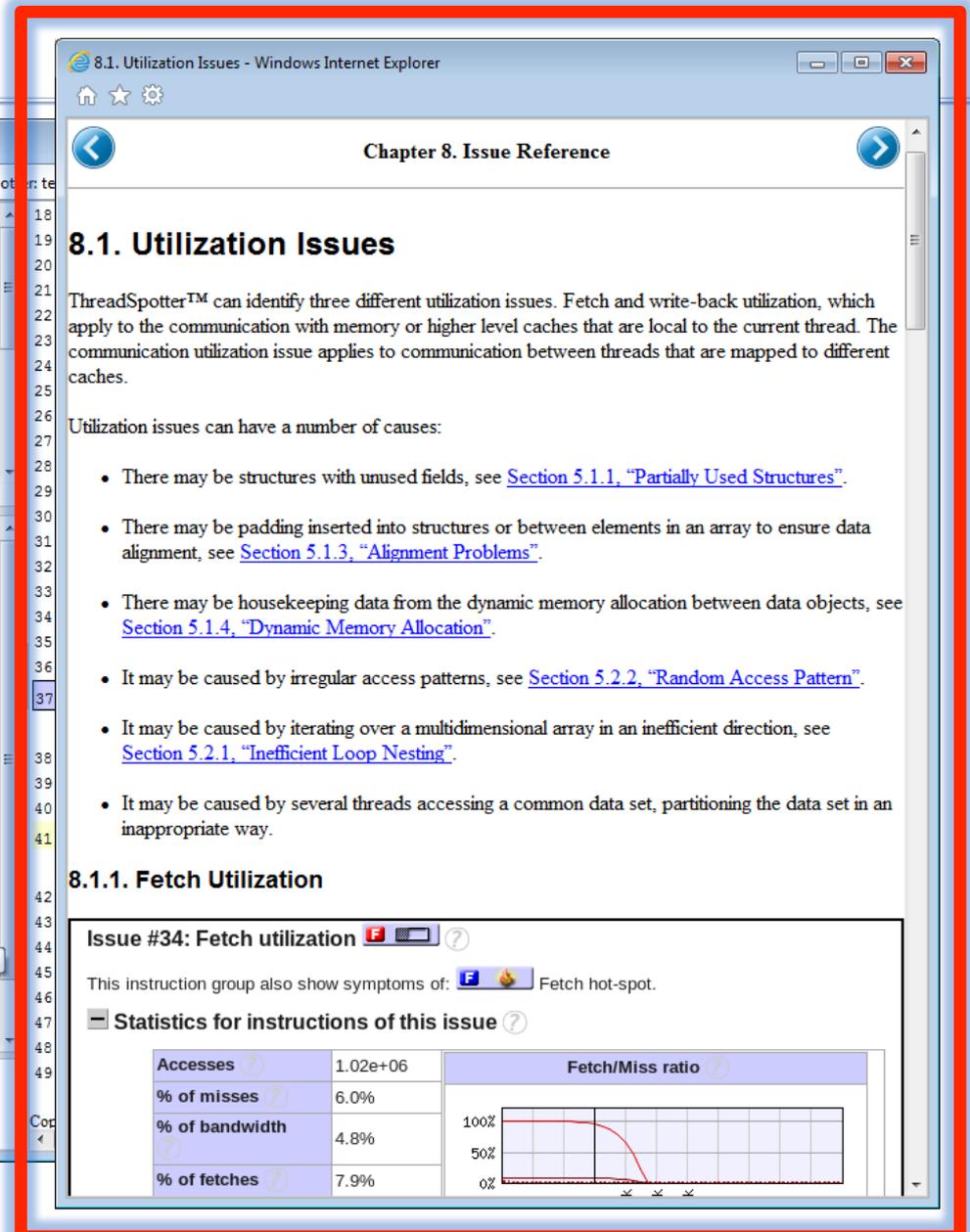
Instructions involved in this issue

Stack	Instruction	% of misses	% of fetches	Fetch ratio	Fetch utilization	W-B Utilization
-	"test2"lexecute()+0x23 (0x80494a3), driver.cc:35					
-	"test2"lask_questions()+0x27 (0x804b957), database.hh:57					
-	"test2"lask_one_question()+0x37 (0x804a177) [R] database_2_vector.hh:37	43.8%	48.3%	43.6%	14.3%	100.0%

Loop statistics

Loop instructions

Placeholder. Click on an issue, loop or file



Chapter 8. Issue Reference

8.1. Utilization Issues

ThreadSpotter™ can identify three different utilization issues. Fetch and write-back utilization, which apply to the communication with memory or higher level caches that are local to the current thread. The communication utilization issue applies to communication between threads that are mapped to different caches.

Utilization issues can have a number of causes:

- There may be structures with unused fields, see [Section 5.1.1, "Partially Used Structures"](#).
- There may be padding inserted into structures or between elements in an array to ensure data alignment, see [Section 5.1.3, "Alignment Problems"](#).
- There may be housekeeping data from the dynamic memory allocation between data objects, see [Section 5.1.4, "Dynamic Memory Allocation"](#).
- It may be caused by irregular access patterns, see [Section 5.2.2, "Random Access Pattern"](#).
- It may be caused by iterating over a multidimensional array in an inefficient direction, see [Section 5.2.1, "Inefficient Loop Nesting"](#).
- It may be caused by several threads accessing a common data set, partitioning the data set in an inappropriate way.

8.1.1. Fetch Utilization

Issue #34: Fetch utilization

This instruction group also show symptoms of:  Fetch hot-spot.

Statistics for instructions of this issue

Accesses	1.02e+06	Fetch/Miss ratio	
% of misses	6.0%		
% of bandwidth	4.8%		
% of fetches	7.9%		

2. Slowspotter Demo

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;
    }
```

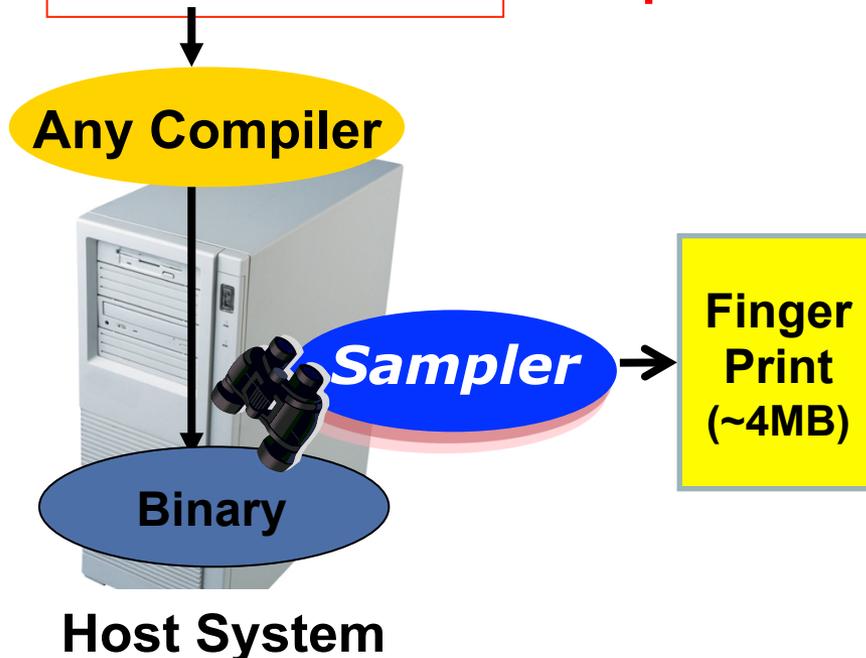
Mission:

Find the SlowSpots™

Asses their importance

Enable for non-experts to fix them

Improve the productivity of performance experts



SlowSpotter

Source:

C, C++

```

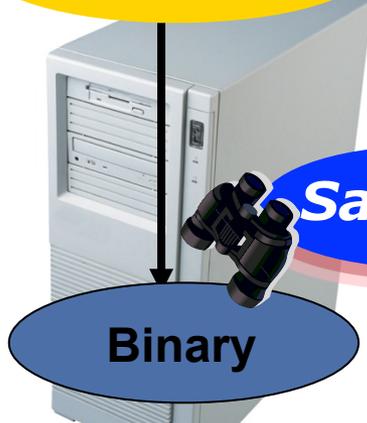
/* Unoptimized Array Multipl
for (i = 0; i < N; i = i + 1)
for (j = 0; j < N; j = j + 1)
{
    r = 0;
    for (k = 0; k < N; k = k + 1)
        r = r + y[i][k] * z[k][j];
    x[i][j] = r;
}

/* 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?

Any Compiler



Host System

Sampler

Finger Print (~4MB)

Analysis

Advice

Target System Parameters

Help!

Loop Issue	Summary	% of fetches	Utilization	HW-Prefetch	Rand
1 / 1	Inefficient loop nesting	38.6%	23.1%	0.0%	Low
1 / 2	Loop fusion	23.3%	13.2%	96.8%	Low
1 / 2	Poor utilization	23.3%	13.2%	96.8%	Low
2 / 2	Inefficient loop nesting	10.2%	12.1%	0.0%	Low
2 / 2	Poor utilization	4.8%	35.1%	87.3%	Low
2 / 2	Loop fusion	4.8%	35.1%	87.3%	Low

Issue #2: Cache line utilization

This instruction group also shows:

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

Copyright (c) 2007-2008 Acumem AB. All Rights Reserved. Patents pending.

```

if (ttemp != f1_layer[tj].P)
    tresult=0;
}
fires = tresult;
/* Compute F1 * Q values */
norm = sqrt((double) tnorm);
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 (ti=0;ti<numf2s;ti++)
{
    if (Y[ti].y > Y[winner].y)
        winner = ti;
}

#ifdef DEBUG
if (DB1) printf("num iterations for p to stable\n");
#endif
match_confidence=rintent2();
if ((match_confidence) > xho)
    
```

Loop instructions

A poor cache line utilization issue indicates that a particular locality, that is, cache lines are only partially used. This means that memory bandwidth and cache capacity are 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 [Structures](#).
- There may be padding inserted into structures for data alignment, see [Section 5.1.3, "Alignment"](#).
- There may be housekeeping data from the dynamic memory objects, see [Section 5.1.4, "Dynamic Memory"](#).
- It may be caused by irregular access patterns, see [Pattern](#).

A One-Click Report Generation

The screenshot shows the Acumem SlowSpotter application window. The interface is divided into several sections:

- Sample source:** Includes options for 'Sample application' (with sub-options 'Launch application' and 'Attach to running application') and 'Read sample file'. The 'Launch application' section has fields for Program (.shor), Arguments (1397 8), and Working directory (/home/erik/demos/libq...).
- Report generation:** Includes fields for Generate report in (/home/erik), Report name (acumem-report), Cache size (2M bytes), and Launch web browser (/usr/bin/htmlview).
- Buttons:** 'Advanced sampling settings...', 'Advanced report settings...', and a large 'Sample application and generate report' button at the bottom.

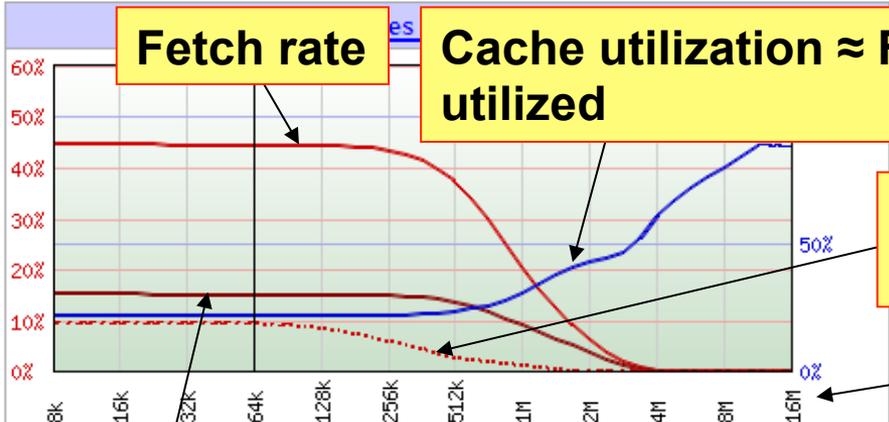
Callout boxes provide the following instructions:

- Fill in the following fields:** Points to the overall form area.
- Application to run** and **Input arguments**: Point to the Program and Arguments fields respectively.
- Working dir (where to run the app)**: Points to the Working directory field.
- (Limit, if you like, data gathered here, e.g., start gathering after after 10 sec. and stop after 10 sec.)**: Points to the Cache size field.
- Cache size of the target system for optimization (e.g., L1 or L2 size)**: Points to the Cache size field.
- Click this button to create a report**: Points to the 'Sample application and generate report' button.

The desktop background is blue with abstract patterns. The system tray at the bottom shows the user [erik@localhost: ~/demos/libquantum/src...], the application Acumem SlowSpotter™, and the terminal [Acumem SlowSpotter™: ./shor (2M/64) -...].

Summary Loops Bandwidth Issues Latency Issues Files Execution About/Help

Select a file in the file table, or follow a source code link from the file table to the file's top description.



Cache utilization \approx Fraction of cache data utilized

Predicted fetch rate (if utilization \rightarrow 100%)

Cache size

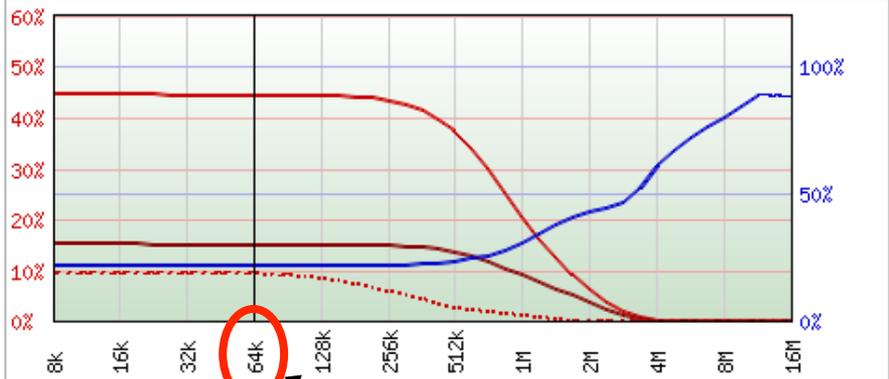
Legend: -- Fetch rate .. Util. corr. fetch rate -- Miss rate -- Utilization

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

Copyright (c) 2007 Acumem AB. All Rights Reserved. Patents pending.

Summary Loops Bandwidth Issues Latency Issues Files Execution About/Help

Miss rates and utilization



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

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

Loop Focus Tab

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%	

List of bad loops

Spotting the crime

```
600 tnorm += f1_layer[ti].P * f1_layer[ti].P;
601
602 if (ttemp != f1_layer[ti].P)
603     tresult=0;
604 }
605 fires = tresult;
606
607 /* Compute F1 - Q values */
608
609 tnorm = sqrt((double) tnorm);
610 for (tj=0;tj<numf1s;tj++)
611 + 4.4% f1_layer[tj].Q = f1_layer[tj].P;
612
613 /* Compute F2 - y values */
614 for (tj=0;tj<numf2s;tj++)
615 {
616     Y[tj].y = 0;
617     if ( !Y[tj].reset )
618         for (ti=0;ti<numf1s;ti++)
619 + 65.4% Y[tj].y += f1_layer[ti].P * bus[ti][tj];
620 }
621
622 /* Find match */
623 winner = 0;
624 for (ti=0;ti<numf2s;ti++)
625 {
626     if (Y[ti].y > Y[winner].y)
627         winner =ti;
628 }
629
630
631 }
632 #ifdef DEBUG
633     if (DB1) print_f12();
634     if (DB1) printf("\n num iterations for p to stabilize = %i \n",j);
635 #endif
636 match_confidence=simtest2();
637 if ((match_confidence) > rho)
638 {
639     /* If the winner is not the default F2 winner (the highest...
```

Explaining what to do

- Loop 1
- + Loop statistics
- + Loop instructions
- + Instruction groups in this loop, summary of issues
- + Instruction group 1
- + Instruction group 2
- + Instruction group 3

Copyright (c) 2007 Acumem AB. All Rights Reserved. Patents pending.

Bandwidth Focus Tab

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				

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

Copyright (c) 2007 Acumem AB. All Rights Reserved. Patents pending.

Explaining what to do

```
600 tnorm += f1_layer[ti].P + f1_layer[ti].P;
601
602 if (ttemp != f1_layer[ti].P)
603     tresult=0;
604 }
605 fires = tresult;
606
607 /* Compute F1 - Q values */
608
609 tnorm = sqrt((double) tnorm);
610 for (tj=0;tj<numf1s;tj++)
611     f1_layer[tj].Q = f1_layer[tj].P;
612
613 /* Compute F2 - y values */
614 for (tj=0;tj<numf2s;tj++)
615 {
616     Y[tj].y = 0;
617     if (!Y[tj].reset)
618         for (ti=0;ti<numf1s;ti++)
619             Y[tj].y += f1_layer[ti].P + bus[ti][tj];
620 }
621
622 /* Find match */
623 winner = 0;
624 for (ti=0;ti<numf2s;ti++)
625 {
626     if (Y[ti].y > Y[winner].y)
627         winner = ti;
628 }
629
630 }
631
632 #ifdef DEBUG
633 if (DB1) print_f12();
634 if (DB1) printf("\n num iterations for p to stabilize = %i \n",
635 #endif
636 match_confidence=simbest2();
637 if ((match_confidence) > rho)
638 {
639     /* If the winner is not the default F2 neuron (the highest
```

Resource Sharing Example

Libquantum

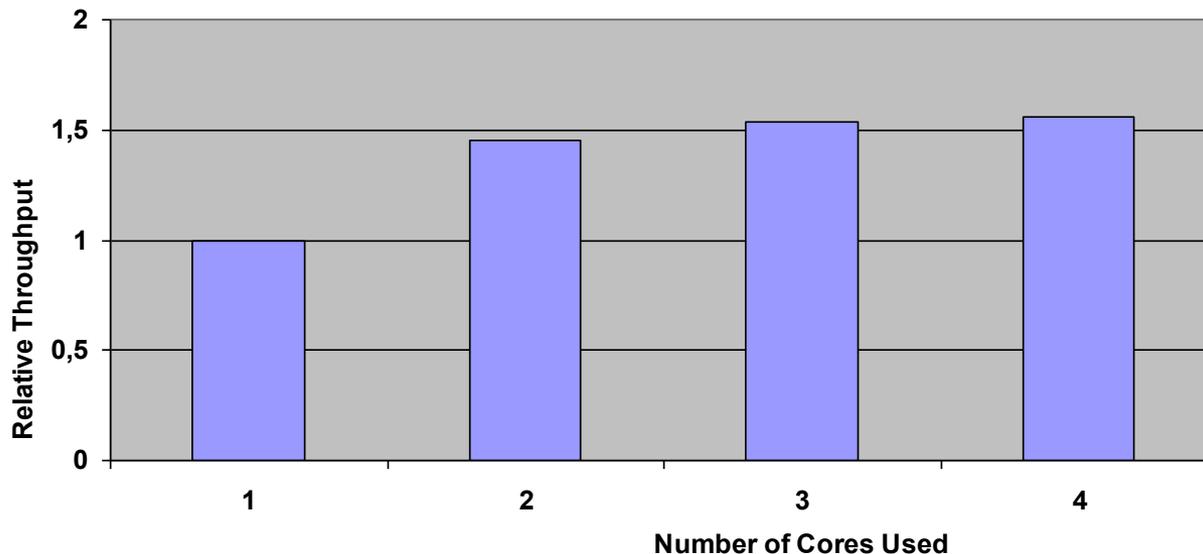
A quantum computer simulation

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

The screenshot shows a Linux desktop with a dark blue background. The top panel includes 'Applications', 'Places', 'System', and system status icons (1.83 GHz, 2:16 PM, Swe). The left sidebar shows icons for 'Computer', 'erik's Home', 'Trash', 'usr', and 'SMB'. The main window is 'Acumem SlowSpotter™' with the following settings:

- Sample source**
 - Sample application
 - Launch application
 - Program: - Arguments:
 - Working directory:
 - Attach to running application
 - Pid:
 - Read sample file
 - Sample file:
- Report generation**
 - Generate report in: - Report name:
 - Cache size: bytes
 - Launch web browser:

Buttons: 'Advanced sampling settings...', 'Sample application', 'Advanced report settings...', and 'Sample application and generate report' (highlighted with a red arrow).

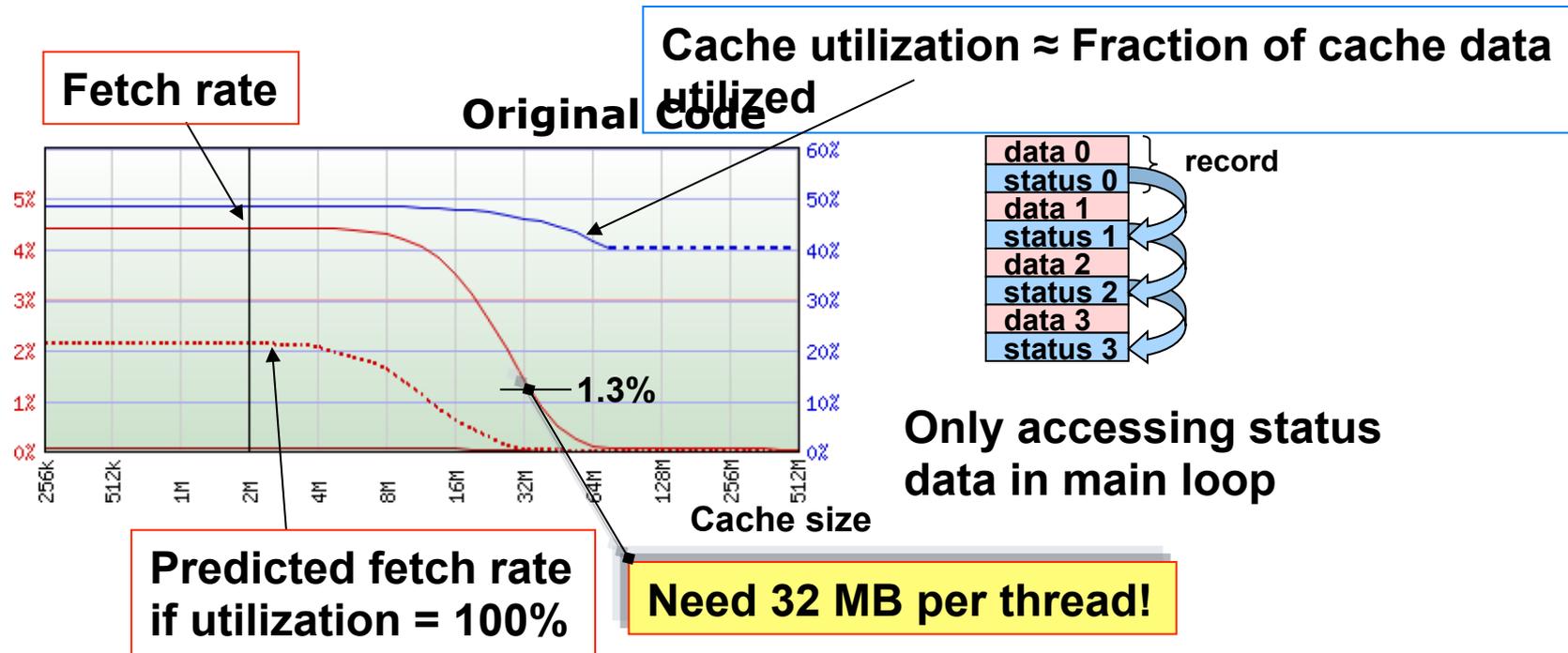
Demo Time!

Libquantum:
Orig code
Spatial opt
Spat + Loop fusion

Edit-compile-analysis cycle \approx 1min

Utilization Analysis

Libquantum



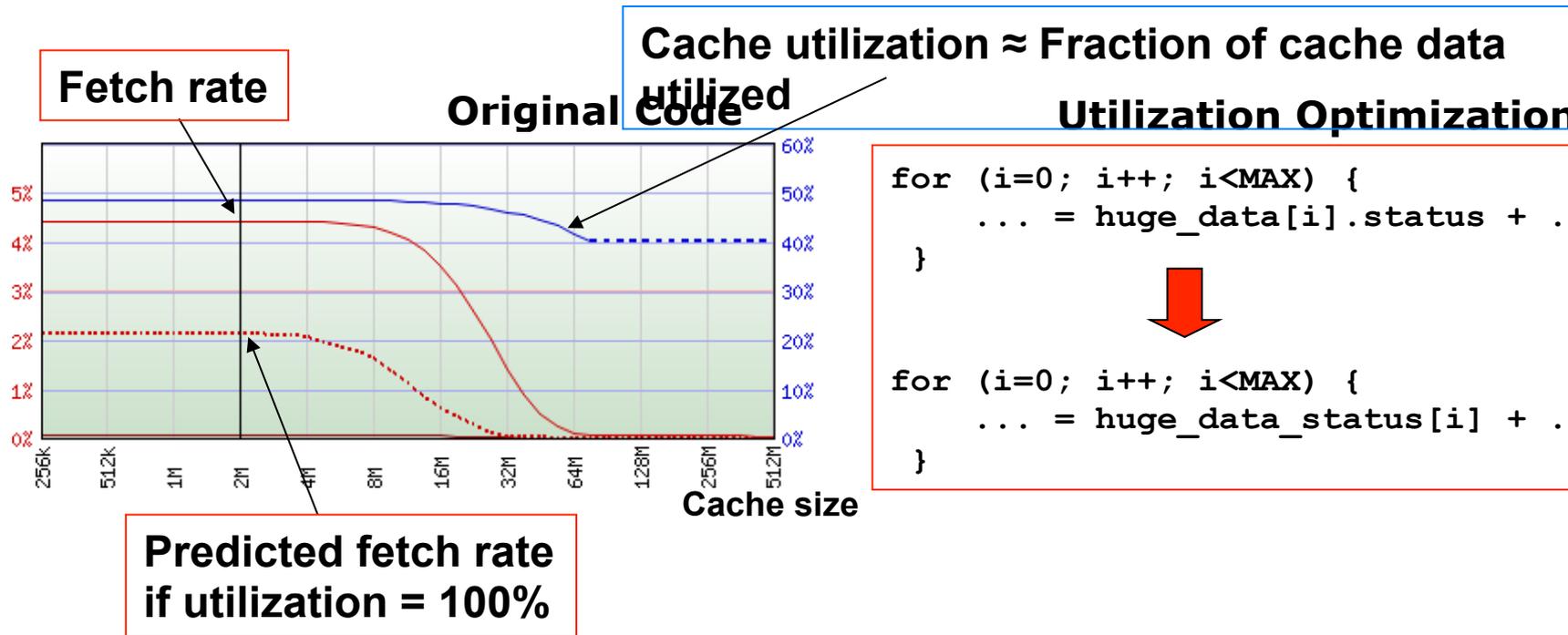
SlowSpotter's First Advice: Improve Utilization

➔ Change one data structure

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

Utilization Analysis

Libquantum



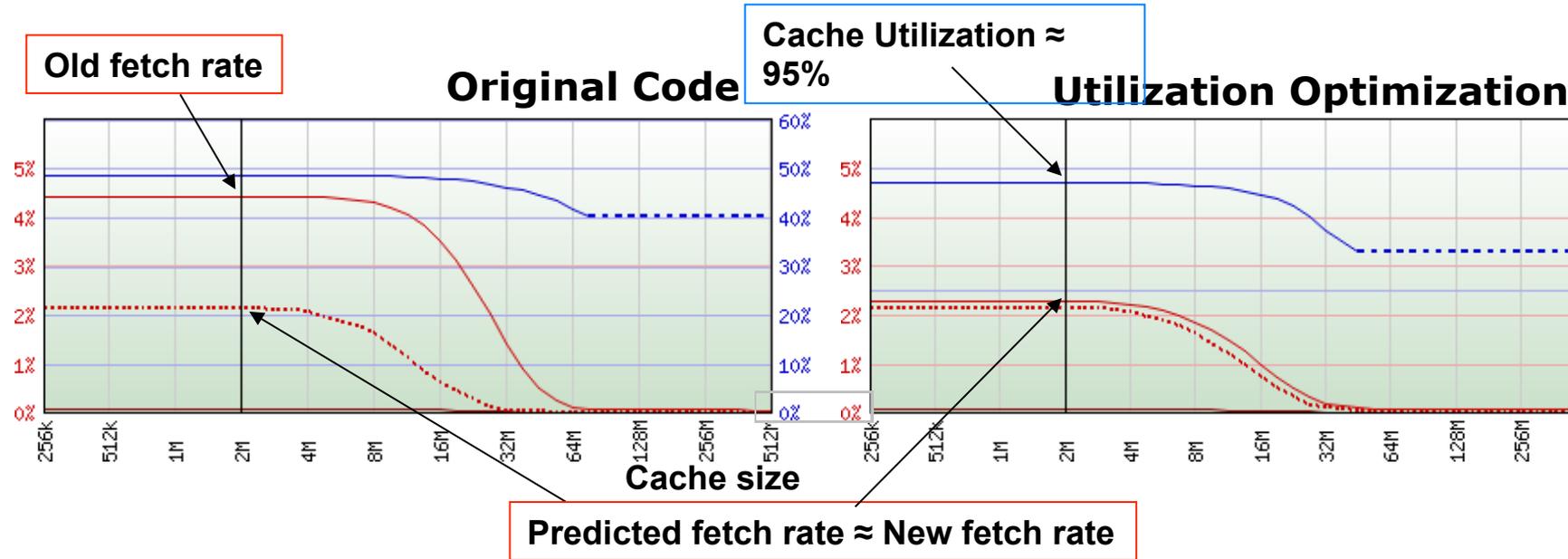
SlowSpotter's First Advice: Improve Utilization

→ Change one data structure

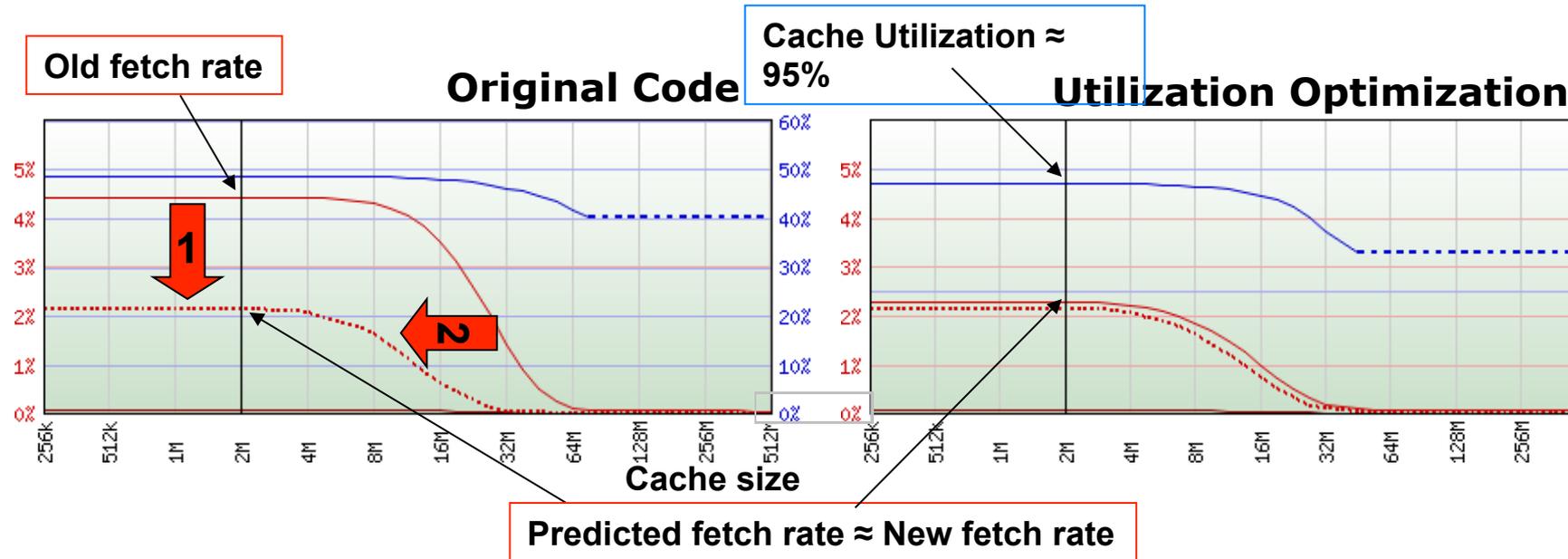
- Involves ~20 lines of code
- Takes a non-expert 30 min

After Utilization Optimization

Libquantum



Utilization Optimization

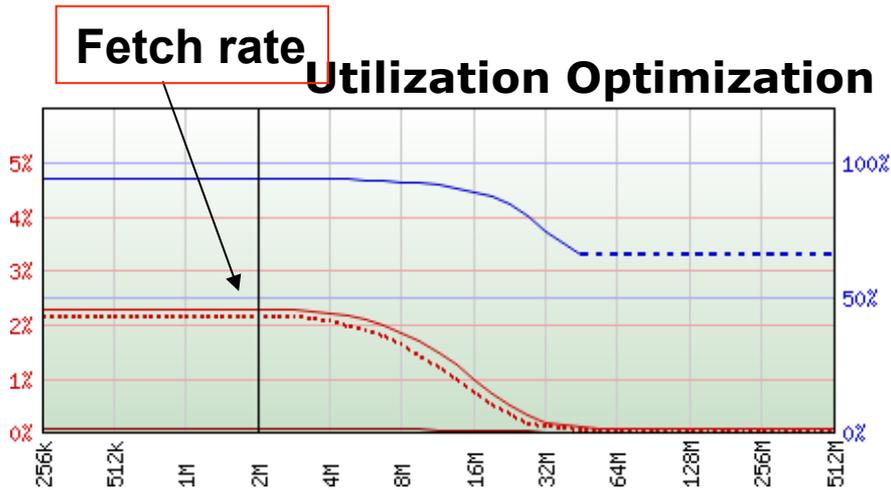


Two positive effects from better utilization

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

Reuse Analysis

Libquantum



Utilization + Fusion Optimization

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

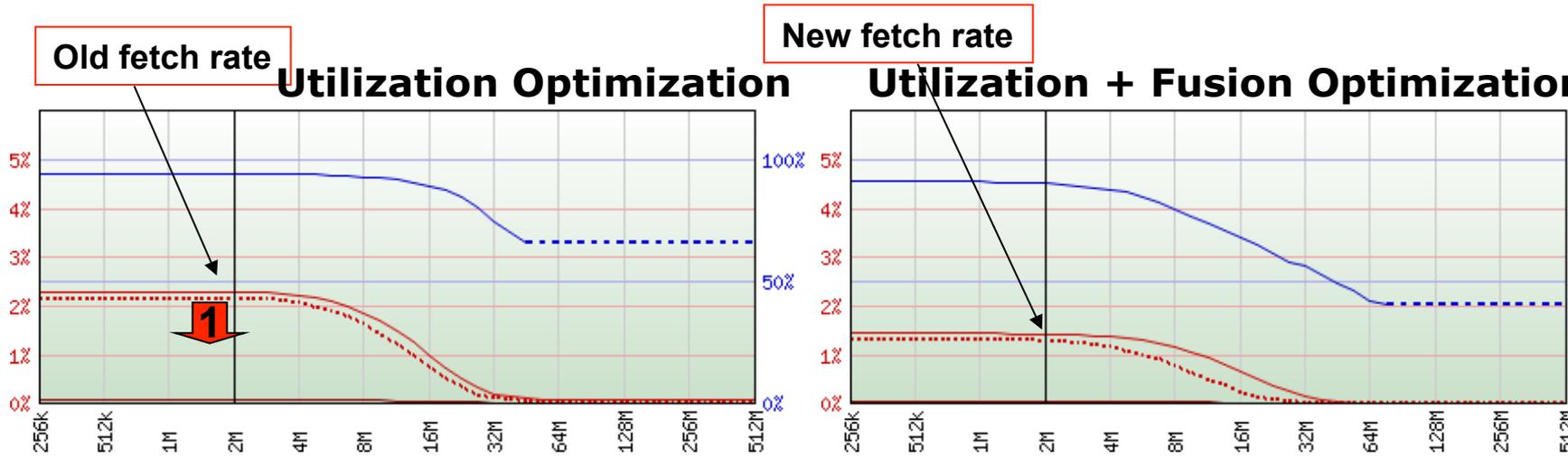
Second-Fifth SlowSpotter Advice: Improve reuse of data

➔ Fuse functions traversing the same data

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

Effect: Reuse Optimization

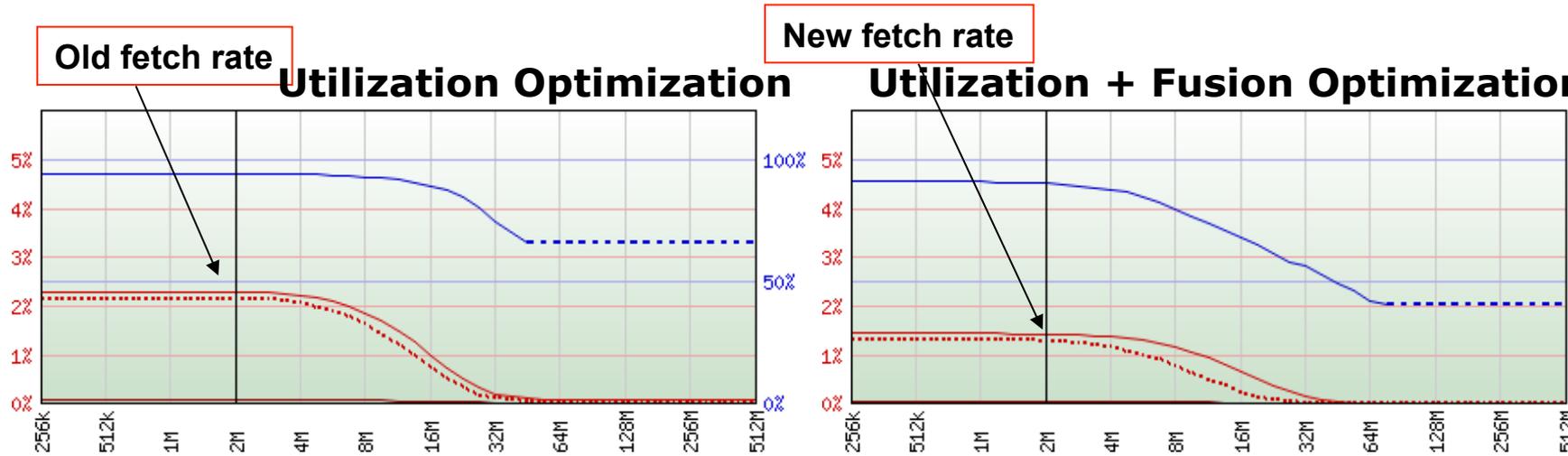
SPEC CPU2006-462.libquantum



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

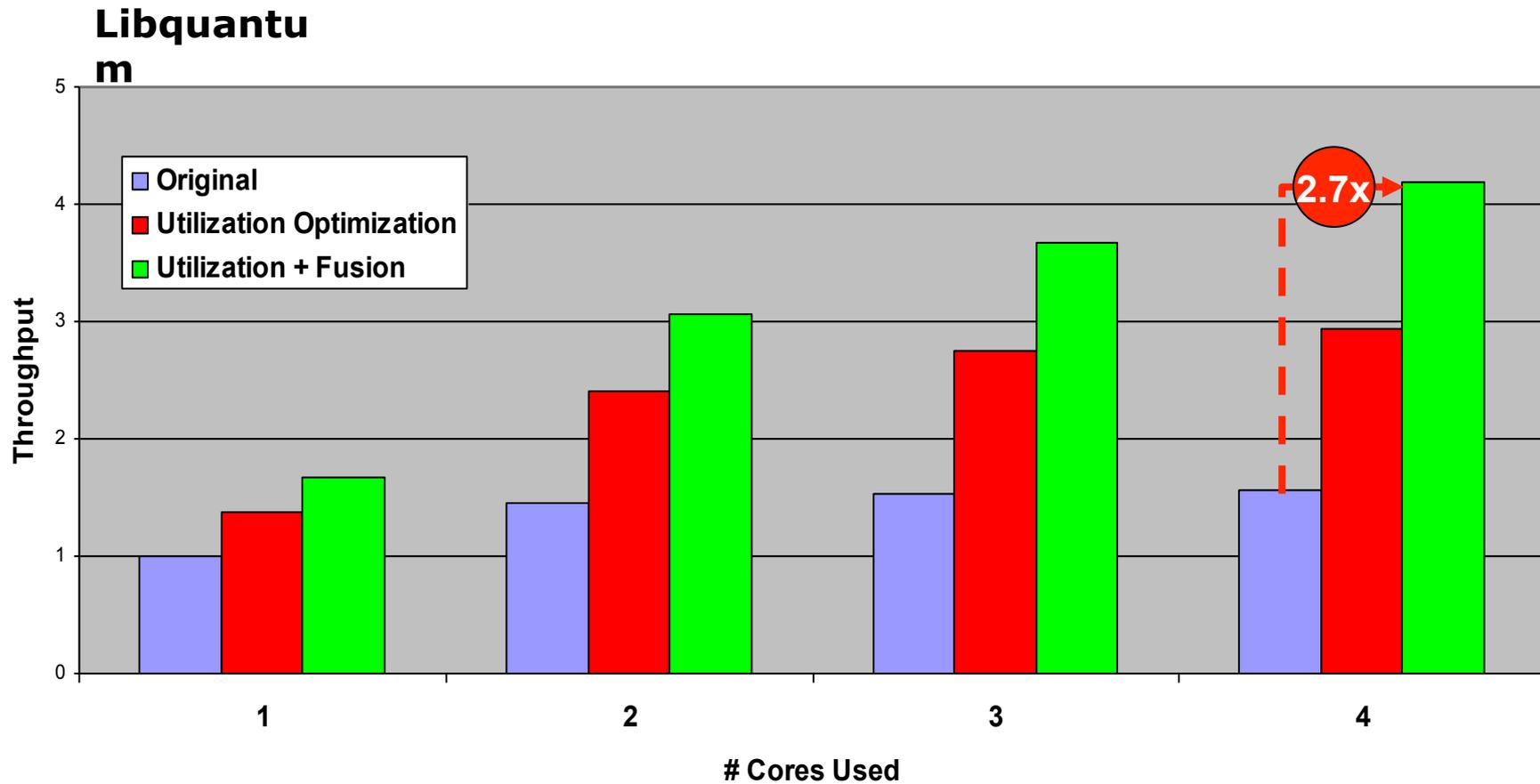
Utilization + Reuse Optimization

Libquantum



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

Summary



Report – front page

The screenshot shows a web browser window with the URL `http://localhost:45570/session/front.html` and a tab titled "ThreadSpotter: test2 (2M/64)". The page content includes:

- ThreadSpotter™** header with a description: "ThreadSpotter™ is a tool to quickly analyze an application for a range of performance problems, particularly related to multicore optimization." and links for "Read more..." and "Manual".
- An "Open the Report" button.
- A red-bordered box containing the "Your application" section, which lists four performance metrics with gauge charts and descriptions:
 - Memory Bandwidth**: The memory bus transports data between the main memory and the processor. The capacity of the memory bus is limited. Abuse of this resource limits application scalability. [Manual: Bandwidth](#)
 - Memory Latency**: The regularity of the application's memory accesses affects the efficiency of the hardware prefetcher. Irregular accesses causes cache misses, which forces the processor to wait a lot for data to arrive. [Manual: Cache misses](#) [Manual: Prefetching](#)
 - Data Locality**: Failure to pay attention to data locality has several negative effects. Caches will be filled with unused data, and the memory bandwidth will waste transporting unused data. [Manual: Locality](#)
 - Thread Communication / Interaction**: Several threads contending over ownership of data in their respective caches causes the different processor cores to stall. [Manual: Multithreading](#)
- A summary statement: "This means that your application shows opportunities to: Avoid major processor stalls and a congested memory bus due to poor data usage." with a "Read more..." link.
- ParaTools** logo.
- Next Steps** section: "The prepared report is divided into sections." followed by a bulleted list:
 - Select the tab **Summary** to see global statistics for the entire application.
 - Select the tabs **Bandwidth Issues**, **Latency Issues** and **MT Issues** to browse through the detected problems.
 - Select the tab **Loops** to browse through statistics and detected problems loop by loop.
- A diagram showing a grid of report sections: "Summary", "Source", "Issue", and "Value details".
- Resources** section with links for "Manual", "Table of Contents", "Optimization Workflow", "Reading the Report", "Rogue Wave Software Web Site", "Rogue Wave Web Site", "Overview", "Concepts", "Issue Reference", and "Tutorials".

Summary, global statistics

The screenshot shows a web browser window with the URL `http://localhost:45570/session/main.html` and a tab titled "ThreadSpotter: test2 (2M/64)". The main content area is divided into a left sidebar and a right main panel. The sidebar, highlighted with a red border, contains a navigation menu with tabs: "Issues", "Loops", "Summary", "Files", "Execution", and "About/Help". The "Summary" tab is active, displaying "Global statistics" and "Analysis parameters".

Global statistics	
Accesses	2.22e+008
Misses	1.19e+006
Fetches	4.34e+007
Write-backs	1.86e+004
Upgrades	0.00e+000
Miss ratio	0.5%
Fetch ratio	19.5%
Write-back ratio	0.0%
Upgrade ratio	0.0%
Communication ratio	0.0%
Fetch utilization	16.5%
Write-back utilization	49.0%
Communication utilization	100.0%

Analysis parameters	
Processor model	Intel(R) Core(TM)2 CPU T7200 @ 2.00GHz (auto)
Number of CPUs	1

The right main panel contains three charts:

- Miss/Fetch ratio:** A line graph showing the percentage of misses and fetches over time. The y-axis ranges from 0% to 25%. The x-axis shows time intervals from 256k to 512M. A solid red line represents the "Fetch ratio" (around 19.5%), a dotted red line represents the "Utilization corrected fetch ratio" (around 16.5%), and a solid black line represents the "Miss ratio" (around 0.5%).
- Write-back ratio:** A line graph showing the percentage of write-backs over time. The y-axis ranges from 0.000% to 0.015%. The x-axis shows time intervals from 256k to 512M. A solid black line represents the "Write-back ratio" (around 0.0%), and a dotted black line represents the "Utilization corrected write-back ratio" (around 0.0%).
- Utilization:** A line graph showing the percentage of utilization over time. The y-axis ranges from 0% to 60%. The x-axis shows time intervals from 256k to 512M. A solid red line represents the "Fetch utilization" (around 16.5%), and a solid green line represents the "Write-back utilization" (around 49.0%).

Placeholder text: "Placeholder. Click on an issue, loop or file" is visible in the main panel and at the bottom of the browser window.

Report sections

The screenshot shows the ThreadSpotter application interface. At the top, there is a navigation bar with tabs: Issues, Loops, Summary, Files, Execution, and About/Help. Below this is a sub-navigation bar with tabs: Bandwidth Issues, Latency Issues, Multi-Threading Issues, and Pollution Issues. A table lists various issues with columns for #, Issue type, Filter, % of bandwidth, % of fetches, write-backs, Fetch utilization, and back utilization. Issue #4 is highlighted as 'Fetch utilization'. Below the table, there is a detailed view for 'Issue #4: Fetch utilization' with expandable sections for statistics, instructions, loop statistics, and loop instructions. On the right side, a code editor displays C++ code with line numbers 18 to 49. The code includes functions like add_one, finalize_adding, and ask_one_question. A yellow highlight is visible on line 37, corresponding to the 'Fetch utilization' issue.

#	Issue type	Filter: All	% of bandwidth	% of fetches	write-backs	Fetch utilization	back utilization
4	Fetch utilization		48.3%	48.3%	0.0%	14.3%	100.0%
7	Spat/temp blocking		48.3%	48.3%	0.0%	14.3%	100.0%
5	Fetch utilization		47.9%	48.0%	0.0%	15.3%	100.0%
9	Spat/temp blocking		47.9%	48.0%	0.0%	15.3%	100.0%
6	Fetch utilization		3.0%	3.0%	0.0%	61.6%	100.0%
10	Spat/temp blocking		3.0%	3.0%	0.0%	61.6%	100.0%
8	Spat/temp blocking		0.1%	0.0%	96.5%	22.8%	22.4%

Issue #4: Fetch utilization

This instruction group also shows symptoms of: Fetch hot-spot.

- Statistics for instructions of this issue
- Instructions involved in this issue
- Loop statistics
- Loop instructions

Copyright (c) 2006-2011 Rogue Wave Software, Inc. All Rights Reserved.
Patents pending.

Placeholder. Click on an issue, loop or file

```
18 | };
19 |
20 |
21 |
22 | void database_2_vector_t::add_one(const car_t &c)
23 | {
24 |     cars.push_back(c);
25 | }
26 |
27 | void database_2_vector_t::finalize_adding()
28 | {
29 |     // std::sort(cars.begin(), cars.end());
30 | }
31 |
32 | void database_2_vector_t::ask_one_question(query
33 | {
34 |     cars_t::const_iterator i = cars.begin(), e =
35 |     for (; i!=e; i++) {
36 |         switch (query.query_type) {
37 |             case 0: // count matching colors
38 |                 if (i->color == query.car.color)
39 |                     query.result++;
40 |                 break;
41 |             case 1: // count same model but heavier
42 |                 if (i->model == query.car.model && i
43 |                     query.result++;
44 |                 break;
45 |             }
46 |         }
47 |     }
48 |     #endif
49 | }
```

Copyright (c) 2006-2011 Rogue Wave Software, Inc. All Rights Reserved.

Navigation by issues

The screenshot shows the ThreadSpotter application interface. At the top, there is a browser address bar with the URL `http://localhost:45570/session/main.html` and a window title `ThreadSpotter: test2 (2M/64)`. Below the browser bar is a navigation menu with tabs: **Issues**, **Loops**, **Summary**, **Files**, **Execution**, and **About/Help**. The **Issues** tab is active, displaying a table of issues. A red box highlights the table. The table has columns for issue number, issue type, and various utilization percentages. Below the table, the details for **Issue #4: Fetch utilization** are shown, including a description and navigation options. On the right side, a code editor displays C++ code with line numbers 18 through 49. The code includes functions like `add_one`, `finalize_adding`, and `ask_one_question`. The code editor also shows a table of issue statistics for the selected code block, with columns for issue number, issue type, and utilization percentages. The table highlights two issues: issue 37 (48.9% Fetch utilization) and issue 41 (51.0% Fetch utilization).

#	Issue type	% of bandwidth	% of fetches	% of write-backs	Fetch utilization	Write-back utilization
4	Fetch utilization	48.3%	48.3%	0.0%	14.3%	100.0%
7	Spat/temp blocking	48.3%	48.3%	0.0%	14.3%	100.0%
5	Fetch utilization	47.9%	48.0%	0.0%	15.3%	100.0%
9	Spat/temp blocking	47.9%	48.0%	0.0%	15.3%	100.0%
6	Fetch utilization	3.0%	3.0%	0.0%	61.6%	100.0%
10	Spat/temp blocking	3.0%	3.0%	0.0%	61.6%	100.0%
8	Spat/temp blocking	0.1%	0.0%	96.5%	22.8%	22.4%

Issue #4: Fetch utilization

This instruction group also shows symptoms of: Fetch hot-spot.

- Statistics for instructions of this issue
- Instructions involved in this issue
- Loop statistics
- Loop instructions

Copyright (c) 2006-2011 Rogue Wave Software, Inc. All Rights Reserved.
Patents pending.

Placeholder. Click on an issue, loop or file

```
18 | };
19 |
20 |
21 | void database_2_vector_t::add_one(const car_t &c)
22 | {
23 |     cars.push_back(c);
24 | }
25 |
26 | void database_2_vector_t::finalize_adding()
27 | {
28 |     // std::sort(cars.begin(), cars.end());
29 | }
30 |
31 | void database_2_vector_t::ask_one_question(query
32 | {
33 |     cars_t::const_iterator i = cars.begin(), e =
34 |     for (; i!=e; i++) {
35 |         switch (query.query_type) {
36 |             case 0: // count matching colors
37 |                 if (i->color == query.car.color)
38 |                     query.result++;
39 |                     break;
40 |             case 1: // count same model but heavier
41 |                 if (i->model == query.car.model && i
42 |                     query.result++;
43 |                     break;
44 |             }
45 |         }
46 |     }
47 | }
48 | #endif
49 |
```

#	Issue type	% of bandwidth	% of fetches	% of write-backs	Fetch utilization	Write-back utilization
37	Fetch utilization	48.9%				
41	Fetch utilization	51.0%				

Copyright (c) 2006-2011 Rogue Wave Software, Inc. All Rights Reserved.

Source code annotation

The screenshot displays the ThreadSpotter application interface. At the top, there are navigation tabs: Issues, Loops, Summary, Files, Execution, and About/Help. Below these are sub-tabs for Bandwidth Issues, Latency Issues, Multi-Threading Issues, and Pollution Issues. A table lists various issues with their respective statistics.

#	Issue type	% of bandwidth	% of fetches	% of write-backs	Fetch utilization	Write-back utilization
4	Fetch utilization	48.3%	48.3%	0.0%	14.3%	100.0%
7	Spat/temp blocking	48.3%	48.3%	0.0%	14.3%	100.0%
5	Fetch utilization	47.9%	48.0%	0.0%	15.3%	100.0%
9	Spat/temp blocking	47.9%	48.0%	0.0%	15.3%	100.0%
6	Fetch utilization	3.0%	3.0%	0.0%	61.6%	100.0%
10	Spat/temp blocking	3.0%	3.0%	0.0%	61.6%	100.0%
8	Spat/temp blocking	0.1%	0.0%	96.5%	22.8%	22.4%

Issue #4: Fetch utilization

This instruction group also shows symptoms of: Fetch hot-spot.

- Statistics for instructions of this issue**
- Instructions involved in this issue**
- Loop statistics**
- Loop instructions**

Copyright (c) 2006-2011 Rogue Wave Software, Inc. All Rights Reserved.
Patents pending.

Placeholder. Click on an issue, loop or file

```
18 | };
19 |
20 |
21 | void database_2_vector_t::add_one(const car_t &c)
22 | {
23 |     cars.push_back(c);
24 | }
25 |
26 | void database_2_vector_t::finalize_adding()
27 | {
28 |     // std::sort(cars.begin(), cars.end());
29 | }
30 |
31 | void database_2_vector_t::ask_one_question(query
32 | {
33 |     cars_t::const_iterator i = cars.begin(), e =
34 |     for (; i!=e; i++) {
35 |         switch (query.query_type) {
36 |             case 0: // count matching colors
37 |                 if (i->color == query.car.color)
38 |                     query.result++;
39 |                     break;
40 |             case 1: // count same model but heavier
41 |                 if (i->model == query.car.model && i
42 |                     query.result++;
43 |                     break;
44 |             }
45 |         }
46 |     }
47 | }
48 | #endif
49 |
```

Navigation by loops

The screenshot displays a performance analysis tool interface. On the left, a table lists various loops with their respective statistics. A red box highlights the top three loops. Below this, a detailed view for 'Loop 2' is shown, including its statistics and a list of bandwidth-related issues. On the right, the source code for the database_2_vector_t class is visible, with specific lines highlighted in yellow to correspond to the loops and issues listed in the tool.

Issues	Loops	Summary	Files	Execution	About/Help
Loop	% of misses	% of fetches	Fetch utilization	Write-back utilization	Issues
2	55.0%	48.9%	14.3%	100.0%	[F] [PI] [ST] [NT]
1	42.1%	48.0%	15.6%	100.0%	[F] [PI] [ST] [NT]
3	2.2%	3.0%	63.6%	100.0%	[F] [PI] [ST] [NT]

Loop 2 ?

+ Loop statistics ?

+ Loop instructions ?

- Bandwidth issues related to this this loop ?

#	Issue type	% of bandwidth	% of fetches	% of write-backs	Fetch utilization	Write-back utilization
7	Spat/temp blocking	48.3%	48.3%	0.0%	14.3%	100.0%
4	Fetch utilization	48.3%	48.3%	0.0%	14.3%	100.0%

Placeholder. Click on an issue, loop or file.

```

};
};
void database_2_vector_t::add_one(const car_t &c)
{
    cars.push_back(c);
}
void database_2_vector_t::finalize_adding()
{
    // std::sort(cars.begin(), cars.end());
}
void database_2_vector_t::ask_one_question(query_t &query) const
{
    cars_t::const_iterator i = cars.begin(), e = cars.end();
    for (; i!=e; i++) {
        switch (query.query_type) {
            case 0: // count matching colors
                if (i->color == query.car.color)
                    query.result++;
                    break;
            case 1: // count same model but heavier
                if (i->model == query.car.model && i->weight > query
                    query.result++;
                    break;
        }
    }
}
#endif
    
```

Copyright (c) 2006-2011 Rome Wave Software, Inc. All Rights Reserved

Issue details

The screenshot shows the ThreadSpotter interface with a red box highlighting the 'Issue #4: Fetch utilization' details. The interface includes a navigation menu, a table of issues, and a detailed view of the selected issue. The detailed view shows the instruction stack, loop statistics, and loop instructions. The code editor on the right shows the source code for the 'ask_one_question' function.

Issue #4: Fetch utilization   

This instruction group also shows symptoms of:   Fetch hot-spot.

+ Statistics for instructions of this issue 

- Instructions involved in this issue 

Stack	Instruction	% of misses	% of fetches	Fetch ratio	Fetch utilization	W-B Utilization
-	"test2"!execute()+0x23 (0x80494a3), driver.cc:35					
	"test2"!ask_questions()+0x27 (0x804b957), database.hh:57					
	"test2"!ask_one_question()+0x37 (0x804a177) [R]	43.8%	48.3%	43.6%	14.3%	100.0%
	database_2_vector.hh:37					

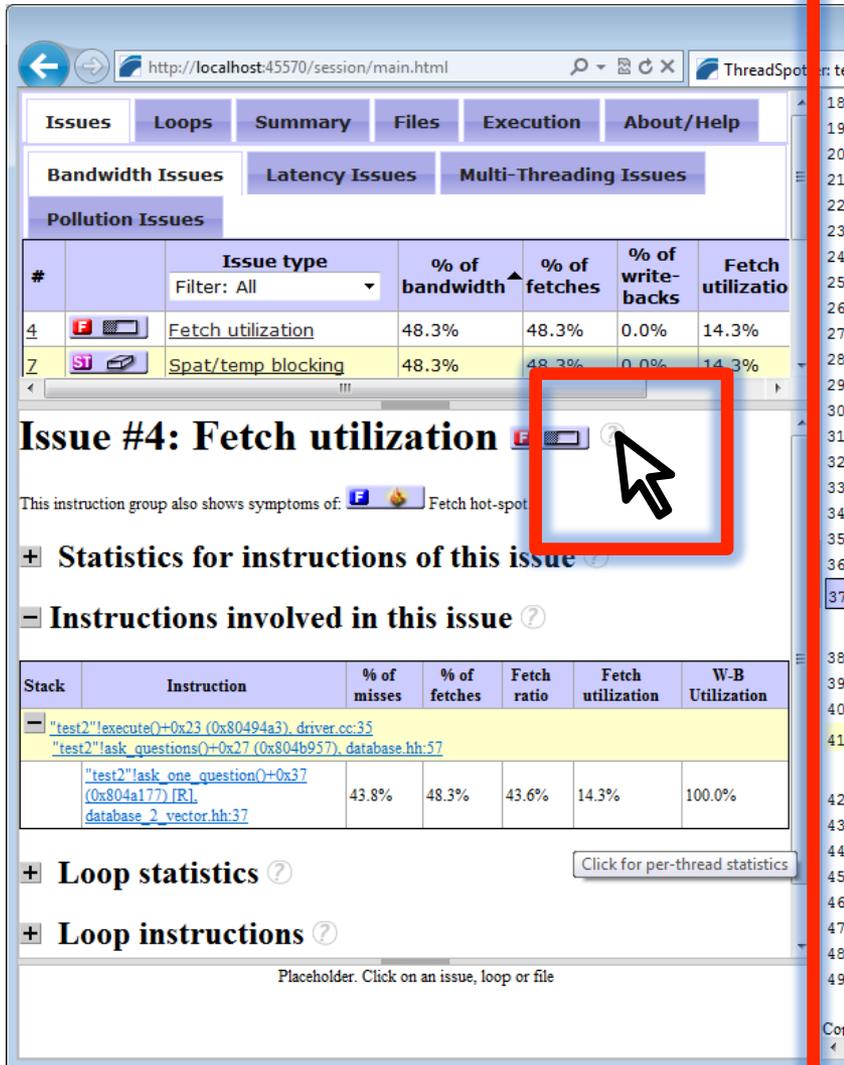
+ Loop statistics  Click for per-thread statistics

+ Loop instructions 

```
18 };
19
20
21 void database_2_vector_t::add_one(const car_t &c)
22 {
23     cars.push_back(c);
24 }
25
26 void database_2_vector_t::finalize_adding()
27 {
28     // std::sort(cars.begin(), cars.end());
29 }
30
31 void database_2_vector_t::ask_one_question(query_t &query) const
32 {
33     cars_t::const_iterator i = cars.begin(), e = cars.end();
34     for (; i!=e; i++) {
35         switch (query.query_type) {
36             case 0: // count matching colors
37                 if (i->color == query.car.color)
38                     query.result++;
39                 break;
40             case 1: // count same model but heavier
41                 if (i->model == query.car.model && i->weight > query
42                     query.result++;
43                 break;
44         }
45     }
46 }
47 #endif
48
49
```

Copyright (c) 2006-2011 Rogue Wave Software, Inc. All Rights Reserved.

Context sensitive help



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

Bandwidth Issues | Latency Issues | Multi-Threading Issues

Pollution Issues

#	Issue type	% of bandwidth	% of fetches	% of write-backs	Fetch utilization
4	Fetch utilization	48.3%	48.3%	0.0%	14.3%
Z	Spat/temp blocking	48.3%	48.3%	0.0%	14.3%

Issue #4: Fetch utilization

This instruction group also shows symptoms of:  Fetch hot-spot

Statistics for instructions of this issue

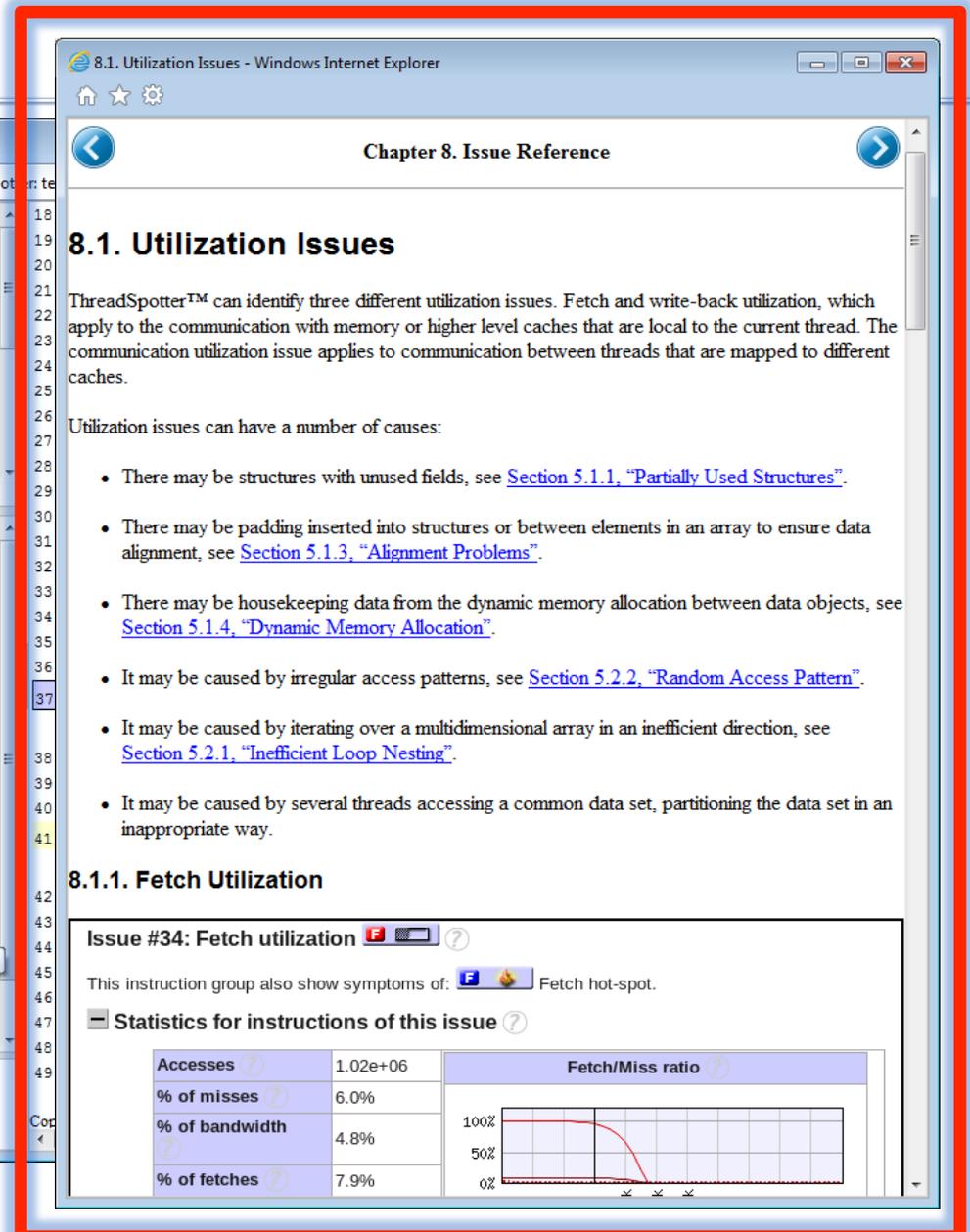
Instructions involved in this issue

Stack	Instruction	% of misses	% of fetches	Fetch ratio	Fetch utilization	W-B Utilization
-	"test2"lexecute()+0x23 (0x80494a3), driver.cc:35					
-	"test2"lask_questions()+0x27 (0x804b957), database.hh:57					
-	"test2"lask_one_question()+0x37 (0x804a177) [R] database_2_vector.hh:37	43.8%	48.3%	43.6%	14.3%	100.0%

Loop statistics

Loop instructions

Placeholder. Click on an issue, loop or file



8.1. Utilization Issues - Windows Internet Explorer

Chapter 8. Issue Reference

8.1. Utilization Issues

ThreadSpotter™ can identify three different utilization issues. Fetch and write-back utilization, which apply to the communication with memory or higher level caches that are local to the current thread. The communication utilization issue applies to communication between threads that are mapped to different caches.

Utilization issues can have a number of causes:

- There may be structures with unused fields, see [Section 5.1.1, "Partially Used Structures"](#).
- There may be padding inserted into structures or between elements in an array to ensure data alignment, see [Section 5.1.3, "Alignment Problems"](#).
- There may be housekeeping data from the dynamic memory allocation between data objects, see [Section 5.1.4, "Dynamic Memory Allocation"](#).
- It may be caused by irregular access patterns, see [Section 5.2.2, "Random Access Pattern"](#).
- It may be caused by iterating over a multidimensional array in an inefficient direction, see [Section 5.2.1, "Inefficient Loop Nesting"](#).
- It may be caused by several threads accessing a common data set, partitioning the data set in an inappropriate way.

8.1.1. Fetch Utilization

Issue #34: Fetch utilization

This instruction group also show symptoms of:  Fetch hot-spot.

Statistics for instructions of this issue

Accesses	1.02e+06	Fetch/Miss ratio
% of misses	6.0%	
% of bandwidth	4.8%	
% of fetches	7.9%	

EXAMPLE
libquantum

Motivating example

Libquantum

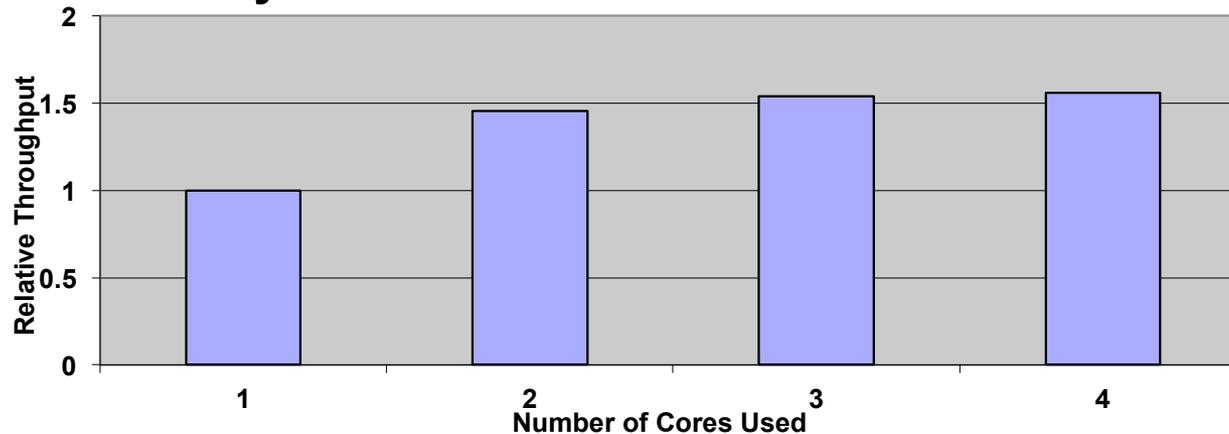
A quantum computer simulation

Widely used in research (download: <http://www.libquantum.de/>)

4000+ lines of C, fairly complex code.

Runs an experiment in ~30 min

Poor scalability



Live demo

Original program



libquantum-orig.tsr

After spatial optimization



libquantum-opt1.tsr

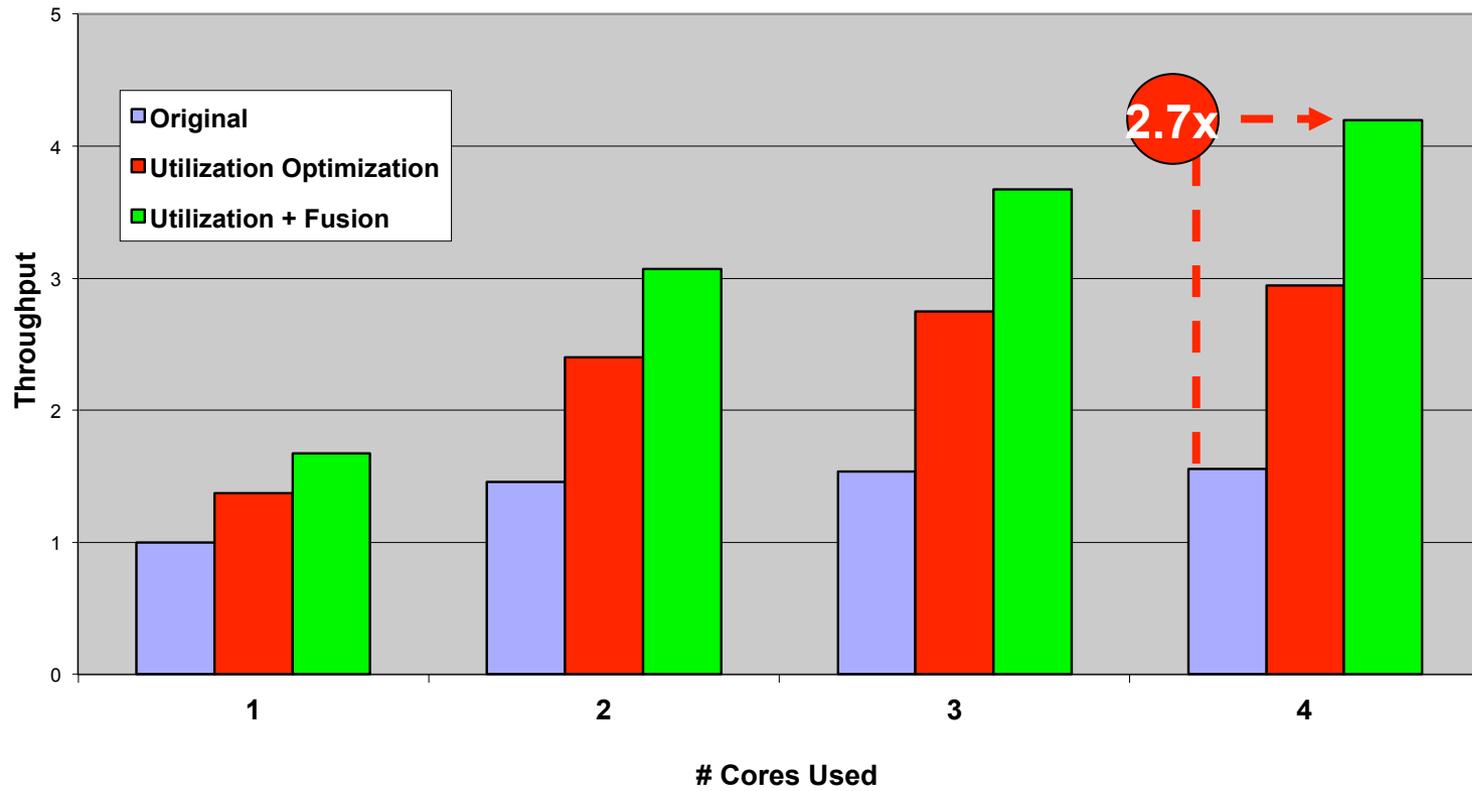
After temporal optimization



libquantum-opt2.tsr

Result

Libquantum



Lunch break

then:

**INSTALL SOFTWARE,
BOOT FROM DVD**

Agenda

- **Installation of software**
- **Individual work with tutorial**
 - 5 Labs,
 - Self study; then
 - we will go through answers and have a short discussion for each lab
- **Presentation of two advanced optimization examples**
 - Blocking
 - False sharing

TUTORIAL

General Workflow

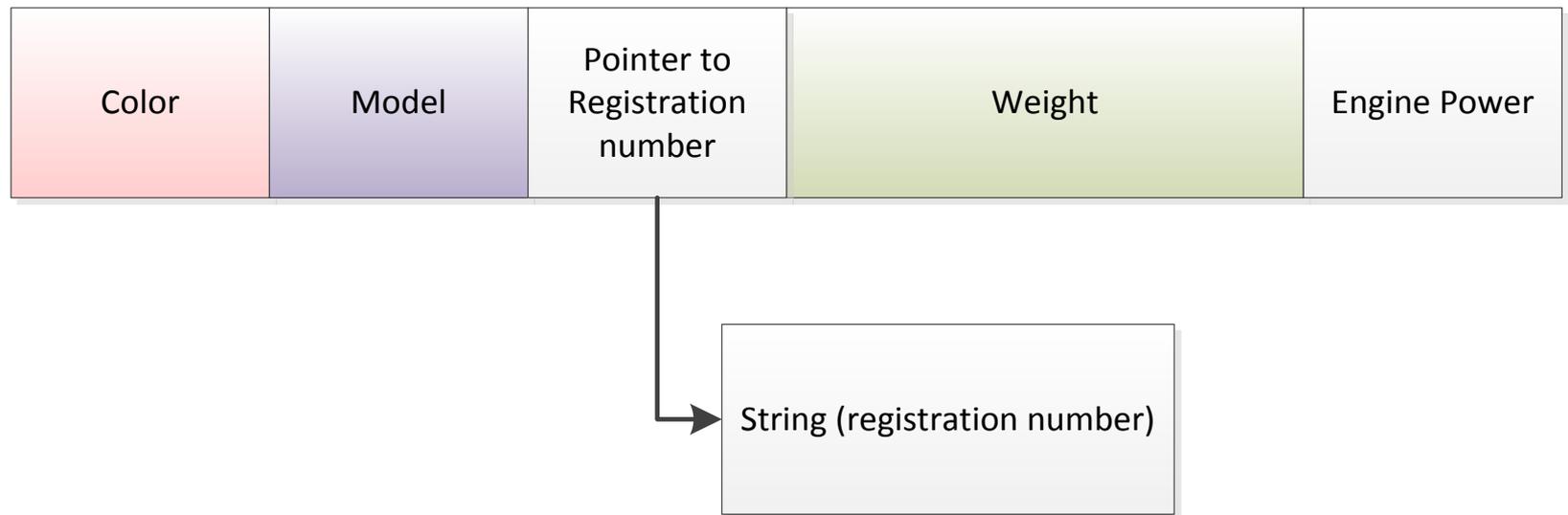
- **Avoid CPU stalls (“cache misses”)**
 - Identify irregular accesses
 - Where is the hardware prefetcher ineffective
 - Convert to consecutive, streaming accesses
 - Hide tricky latencies using prefetches
- **Make better use of cache space**
 - Spatial locality
 - Separate read only fields and read/write fields
- **Improve scalability**
 - Long term data reuse

General Workflow (continued)

- **Inefficient use of shared memory in multithreaded programs:**
 - False sharing
 - Poor communication efficiency
(few bytes transferred per cache line downgrade)
- **Avoiding cache pollution (depending on architecture):**
 - Write combining, a.k.a streaming stores
 - Non-temporal prefetching
- **Other things:**
 - TLB pressure, Cache conflicts.

Example

- **Application performs repeated lookups in a table**
- **Each record consists of several fields:**



- **Different queries:**
 - **Count cars with certain color**
 - **Count cars with certain model and minimum weight**

Example code versions

- **Linked list**
- **Linked list with prefetch hints**
- **Vector**
- **Several vectors**
- **Blocked (a.k.a Tiled)**

Baseline code

```
class database_1_linked_list_t : public single_question_database_t {  
public:  
    virtual void ask_one_question(query_t &query) const  
private:  
    typedef std::list<car_t> cars_t;  
    cars_t cars;  
};
```

```
void database_1_linked_list_t::ask_one_question(query_t &query) const  
{
```

```
    cars_t::const_iterator i = cars.begin(), e = cars.end();
```

```
    for (; i != e; i++) {
```

```
        switch (query.query_type) {
```

```
            case 0: // count matching colors
```

```
                if (i->color == query.car.color)
```

```
                    query.result++;
```

```
                    break;
```

```
            case 1: // count same model but heavier than minimum weight
```

```
                if (i->model == query.car.model &&
```

```
                    i->weight > query.car.weight)
```

```
                    query.result++;
```

```
                    break;
```

```
            }
```

```
        }
```

```
    }
```

Linked list

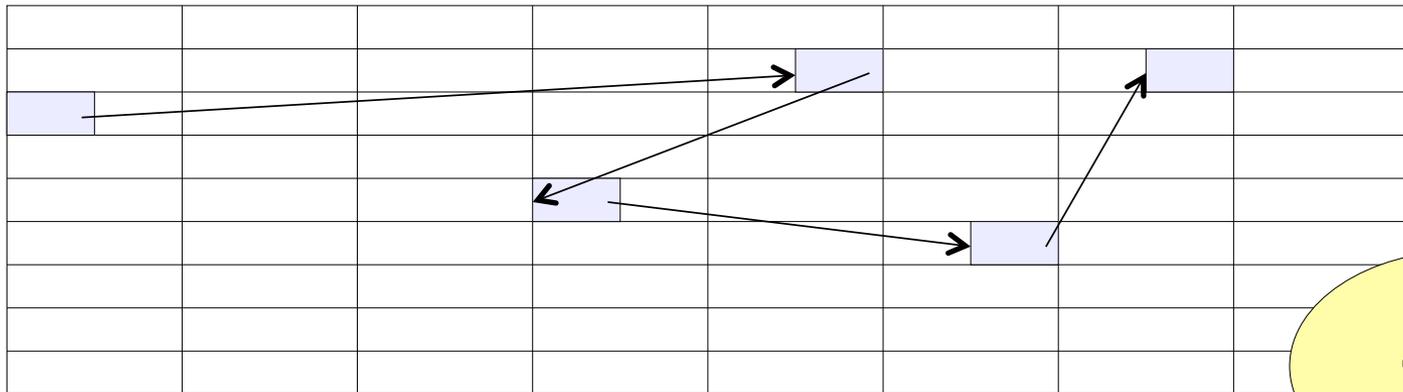
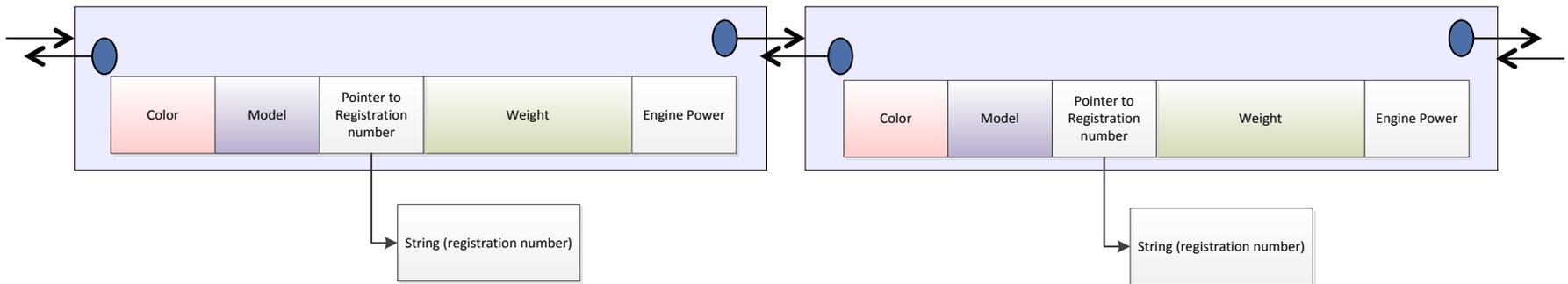
For each query

Traverse entire list

Two variants of
Record access

Linked list (doubly linked, std::list)

A linked list is the worst possible scenario.

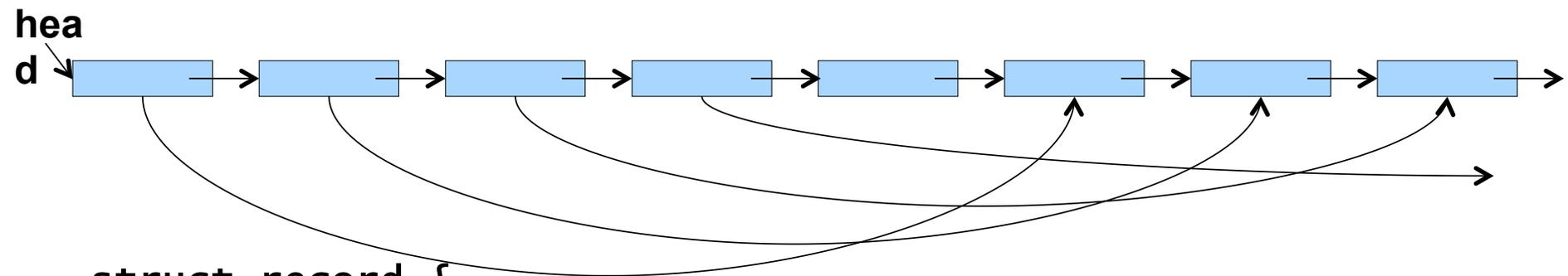


13.6 s



Prefetch Improvements

Hide traversal latencies by prefetching, slightly in advance.



```
struct record {  
    car data;  
    record *next;  
    record *prefetch_hint;  
};
```

```
for (record *i = head; i != NULL; i = i->next) {  
    __builtin_prefetch(i->prefetch_hint); //  
gcc
```

6.6 s

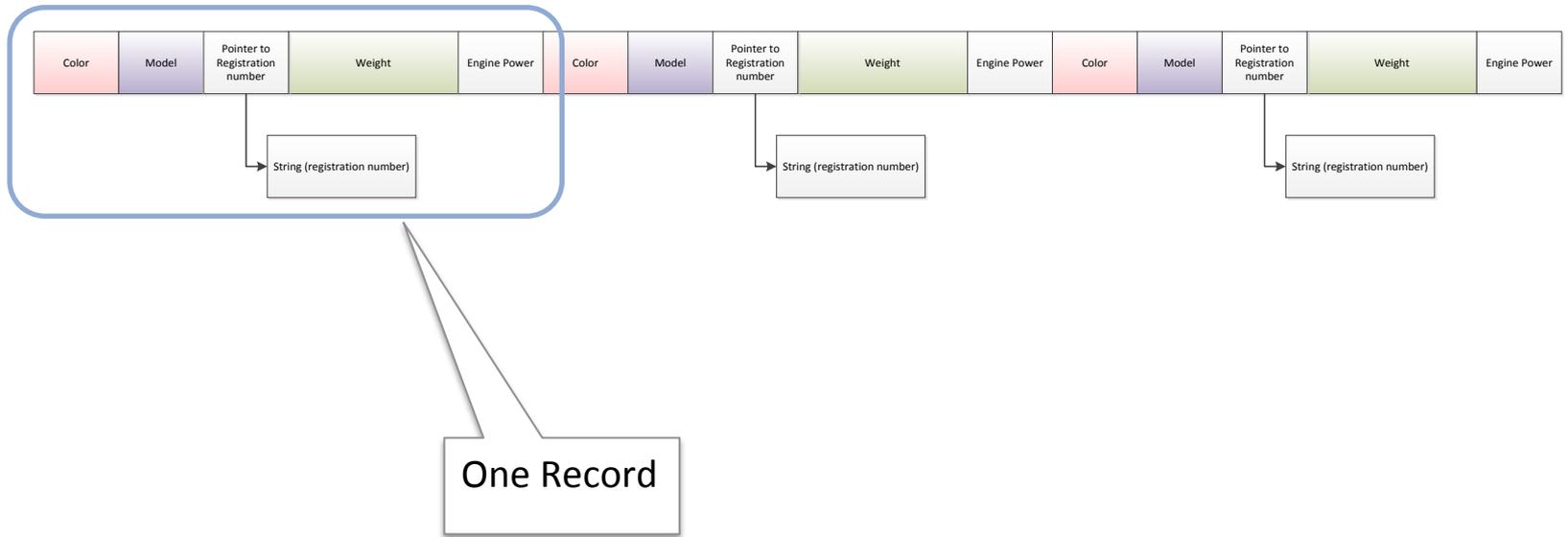


Arrange data consecutively in memory

- **Use custom memory allocators to control dynamic memory layout**
 - (for instance to keep linked list nodes adjacent in memory)
- **Or use data structures that guarantee consecutive storage**
 - Plain old vectors
 - `std::vector`
 - `std::deque`

Improvements: packing

Place data consecutively, i.e. array, vector

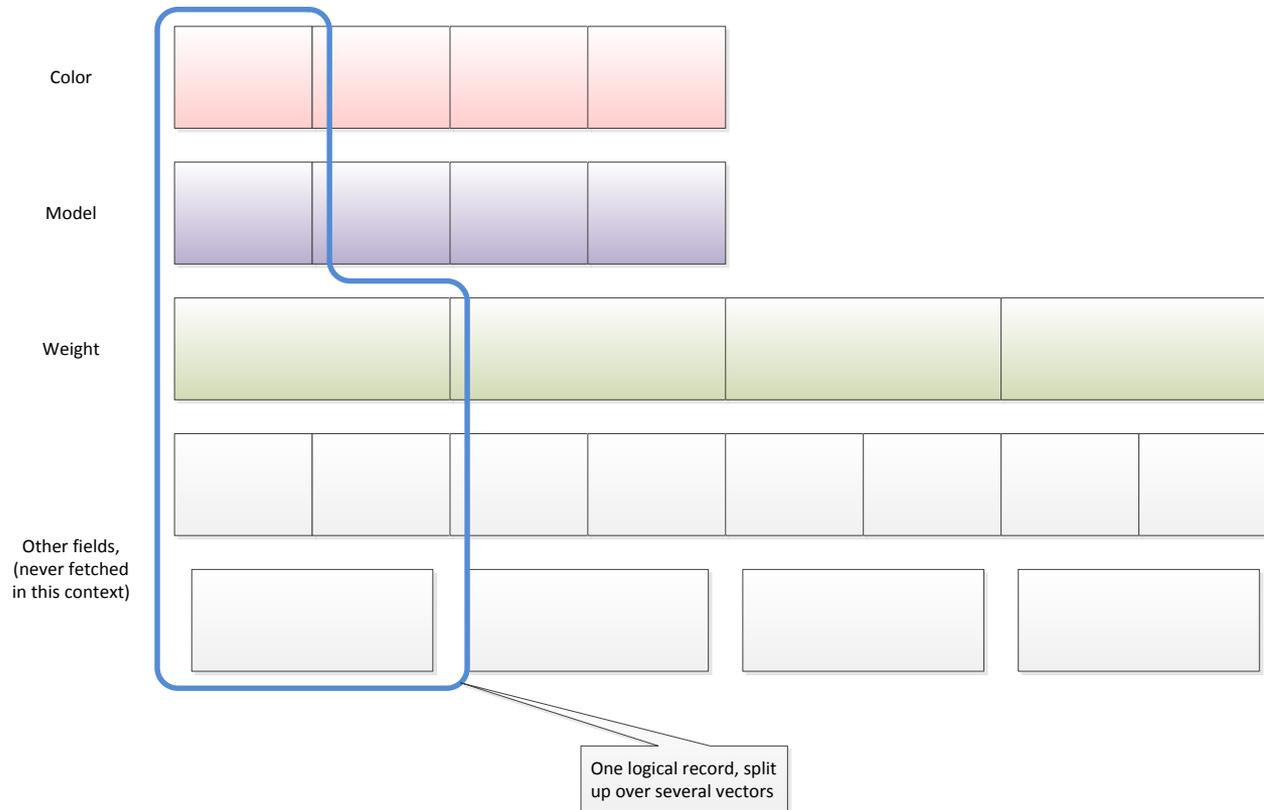


0.88 s



Improvements: packing

Store often used fields together; move less used fields elsewhere.

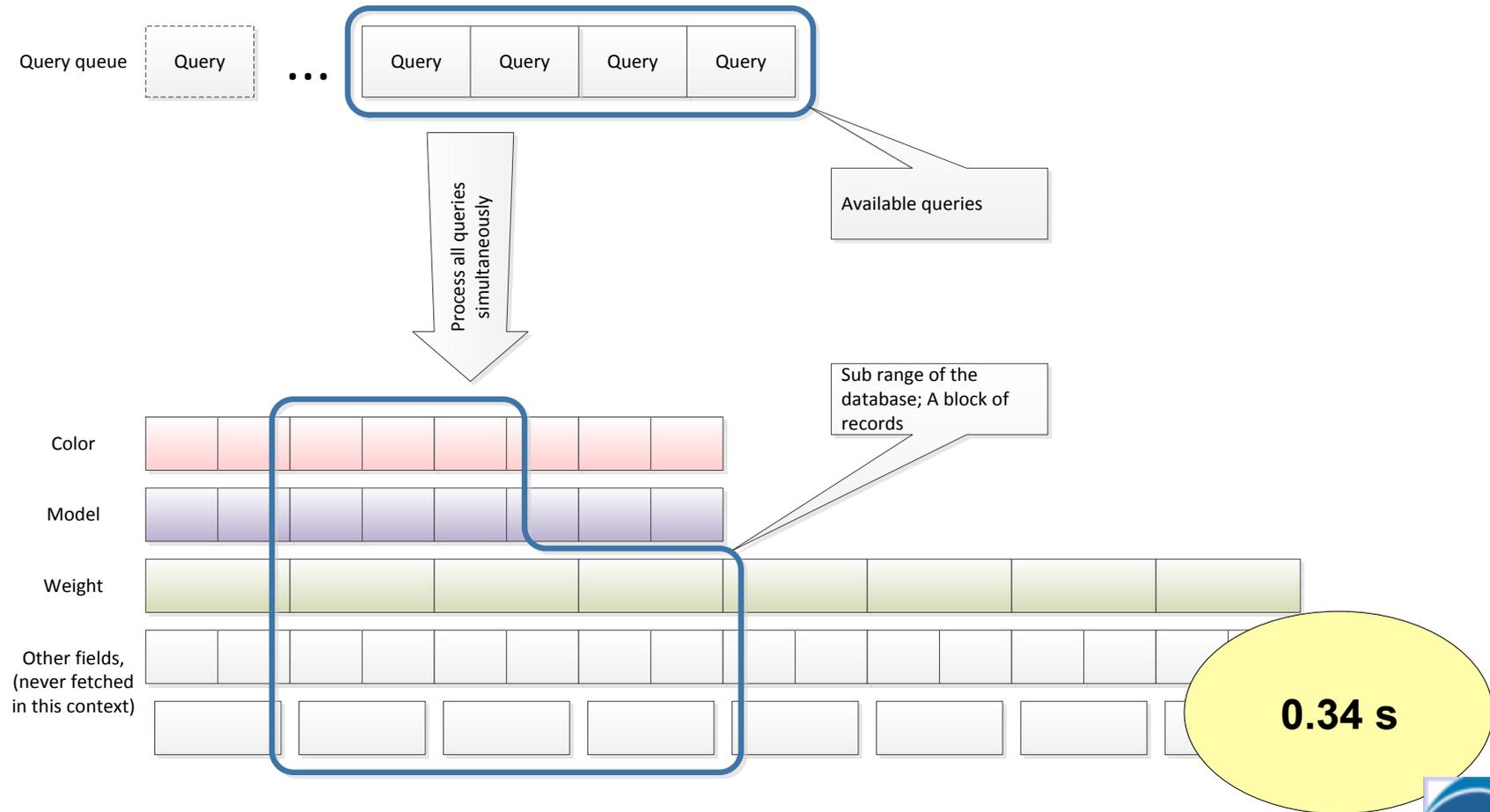


0.45 s



Improve long term reuse

Blocking means batching and subdividing data to fit in cache.

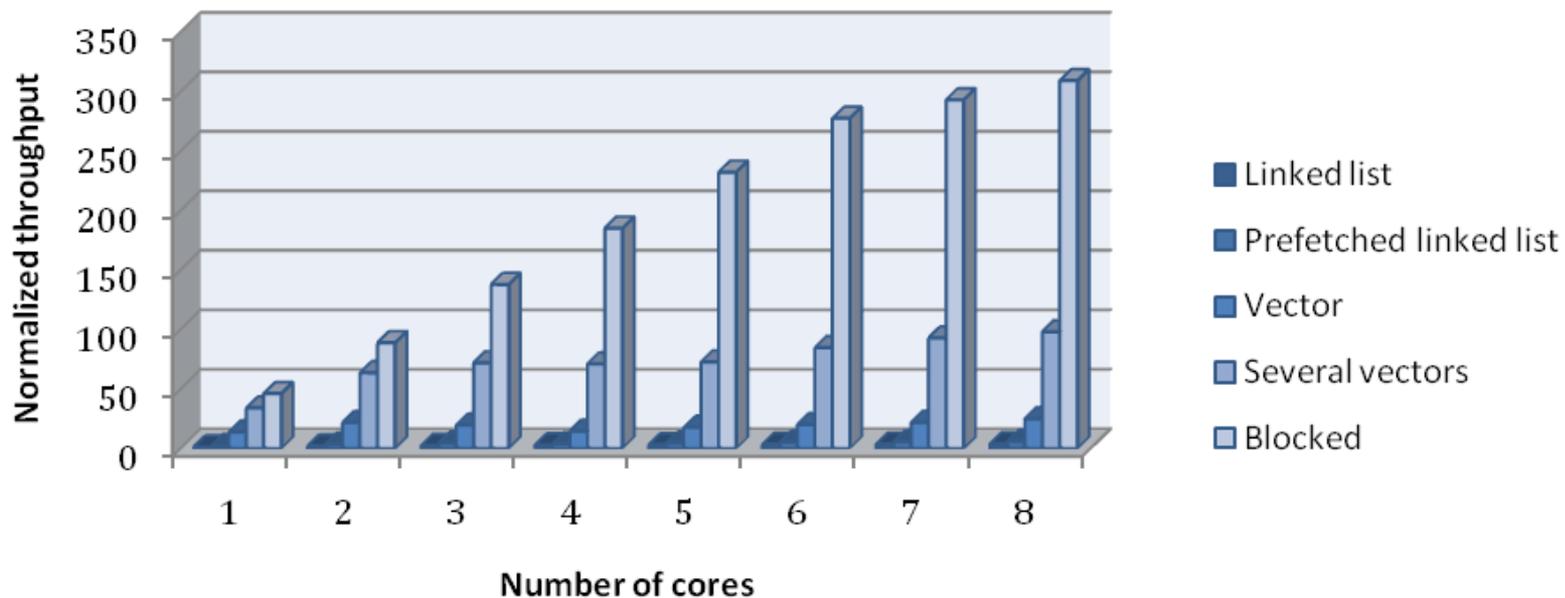


Summary example timings

- **Linked list:** **13.6 s**
- **Linked list w/ prefetch** **6.6 s**
- **Vector** **0.88 s**
- **Several vectors** **0.45 s**
- **Blocking** **0.34 s**

Multithreaded app, scaling properties

# cores:	1	2	3	4	5	6	7	8
1 – Linked list	≡ 1	1.8	2.4	2.8	3.1	3.5	3.5	3.7
2 – Prefetched linked list	2.2	3.5	4.4	3.8	4.5	5.1	5.4	5.9
3 – Vector	14	22	19	15	18	20	22	25
4 – Several vectors	34	63	72	71	73	85	93	98
5 – Blocked	46	89	138	185	232	277	293	309



EXAMPLE (BLOCKING)

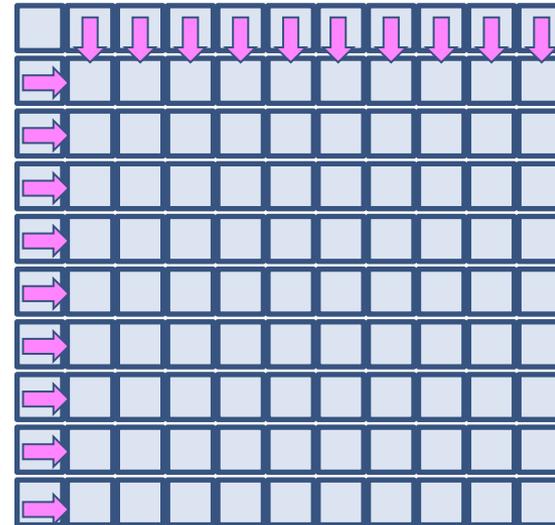
Gaussian Elimination with pivoting
(Forward elimination step)

Overview of the forward elimination step

```
for i=1 to n-1
  → find pivotPos in column i
  if pivotPos ≠ i
    exchange rows(pivotPos,i)
  end if

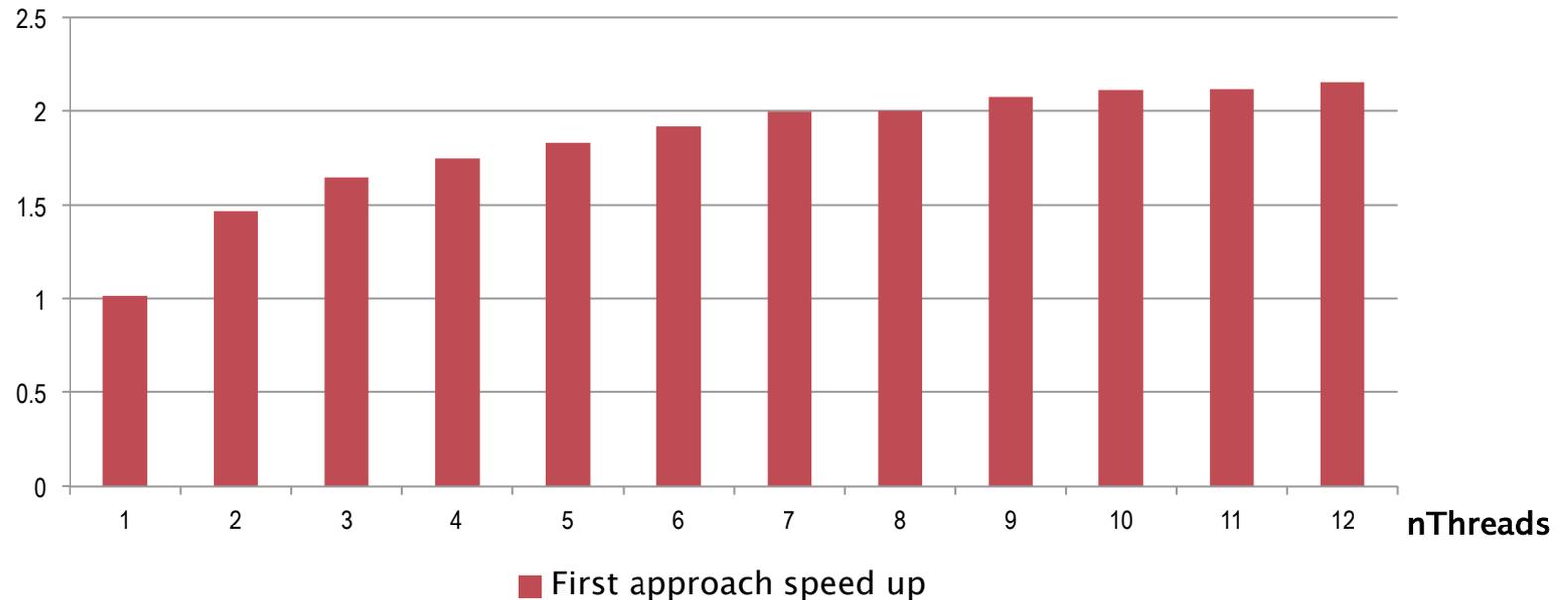
  for j=i+1 to n
    A(i,j) = A(i,j)/A(i,i)
  end for j
  !omp parallel do private ( i , j )

  for j=i+1 to n+1
    for k=i+1 to n
      A(k,j)=A(k,j)-A(k,i)×A(i,j)
    end for k
  end for j
end for i
```



First approach speed up

Speed up w.r.t. sequential version



What went wrong?

- For each prepared pivot, the whole matrix is accessed. The algorithm requires pivots to be calculated in order.
- Repeated eviction of the matrix' cache lines.
- **Observation: Each column is an accumulation of eliminations using previous columns!**
- **Temporal Blocking Advice says:**
 - Use each column many times before it gets evicted.
- **How? To use each column more times means we have to:**
 - **Arrange code to make more pivots available!**

Blocking GE

The row exchange turned into a two-element swap before column elimination. We need this auxiliary storage for the original pivot location.

for k=1 to n-1, step C

BlockEnd=min(k+C-1,n)

GE on A(k:n,k:BlockEnd) &

Store C pivots' positions

!\$omp parallel do private (i , j)

for each column j after BlockEnd

for i=k to BlockEnd

swap using pivots(i)

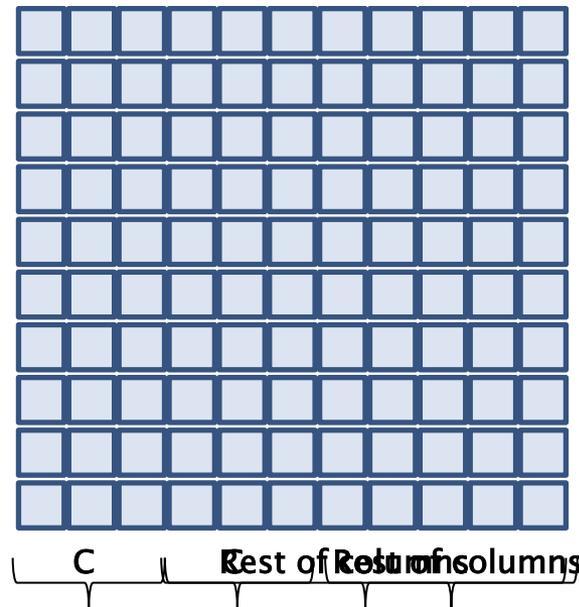
elimination i on j

end for i

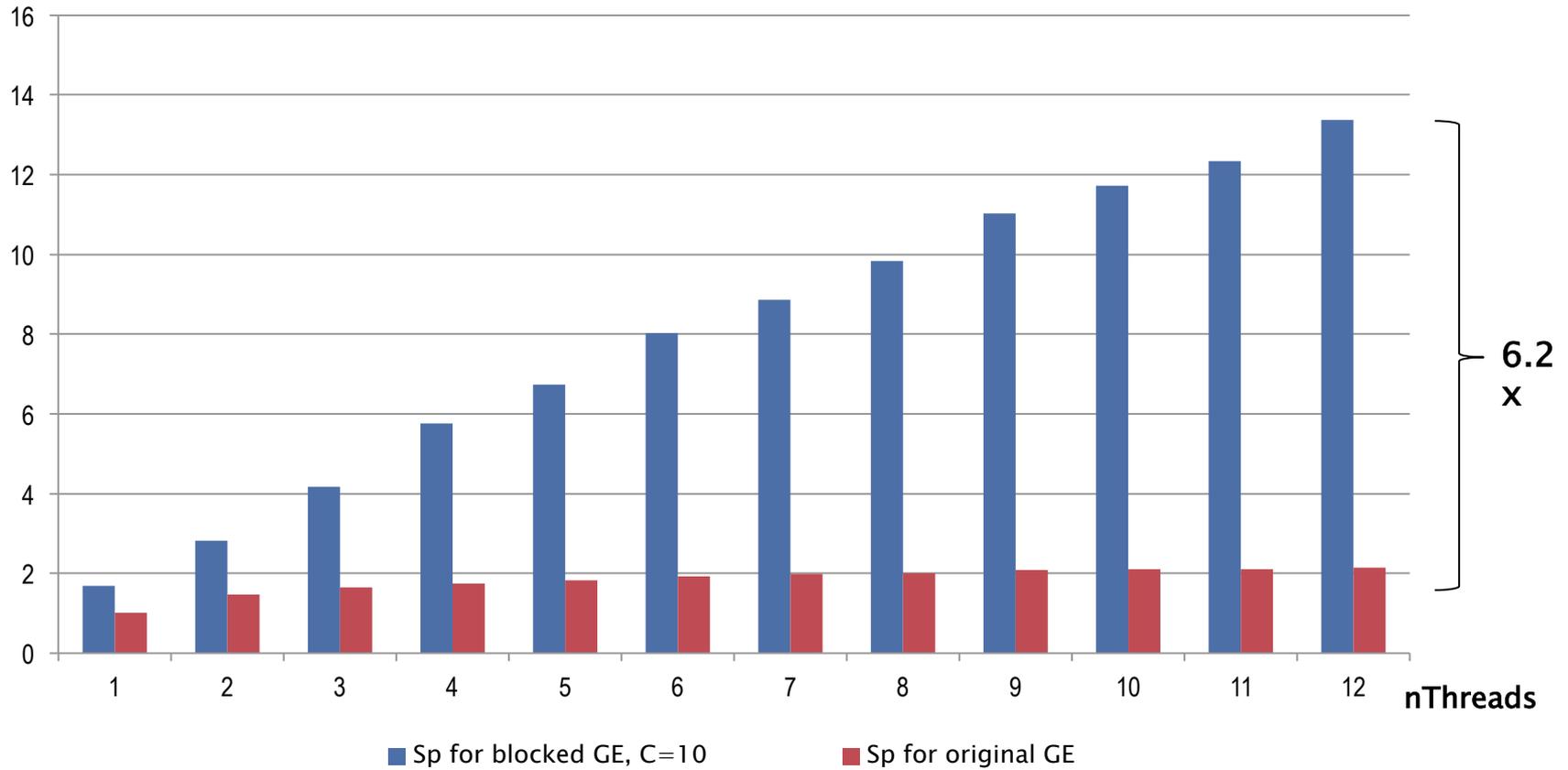
end for each j

End for k

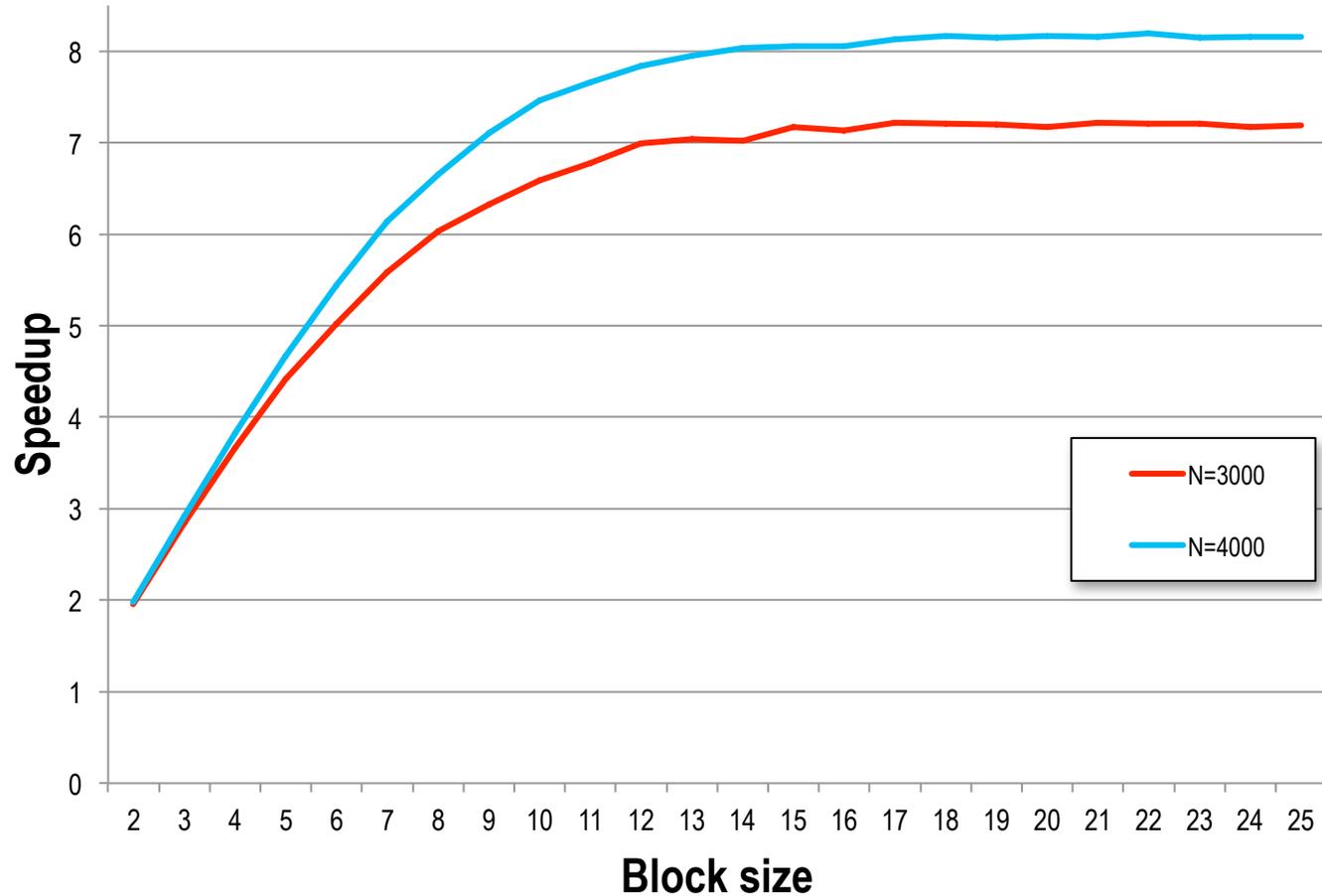
Pivots array



Speed-up relative to the sequential time



Selecting a Good Blocking Size



EXAMPLE (FALSE SHARING)

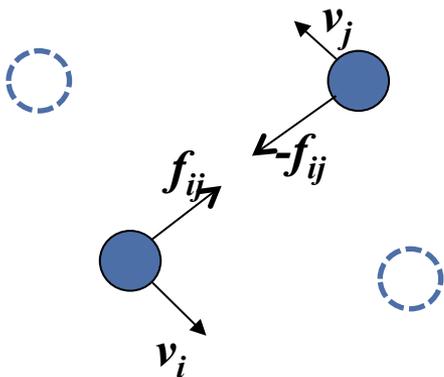
N-Body

Simulation of Gravitational N-body problem

- Initialize bodies
- for time= start to end step by Δt
 - Calculate forces
 - Move bodies
- end for time

for each body $i=1$ to n
 $d\vec{v} = \vec{f}_i / m_i \times \Delta t$
 $d\vec{p} = (\vec{v}_i + d\vec{v} / 2) \times \Delta t$
 $\vec{v}_i += d\vec{v}$
 $\vec{p}_i += d\vec{p}$
 $\vec{f}_i = \vec{0}$
end for each

for each body $i=1$ to $n-1$
for each neighbour $j=i+1$ to n
calculate:
 $r_{ij} = |\vec{p}_i - \vec{p}_j|$
 $\vec{f}_{ij} = \frac{Gm_i m_j}{r_{ij}^2} \frac{(\vec{p}_i - \vec{p}_j)}{r_{ij}}$
 $\vec{f}_i += \vec{f}_{ij}$
 $\vec{f}_j -= \vec{f}_{ij}$
end for j
end for i



Algorithm

```
#pragma omp parallel private(i,j)
for (time=start to end, step dt)
{ #pragma omp for
  for(i=0 to n-1, step 1)
    for(j=i+1 to n-1, step 1)
      CalculateForce(bodyArr[i], bodyArr[j]);

  #pragma omp for
  for(i=0 to n-1, step 1)
    Move(bodyArr[i]);
}
```

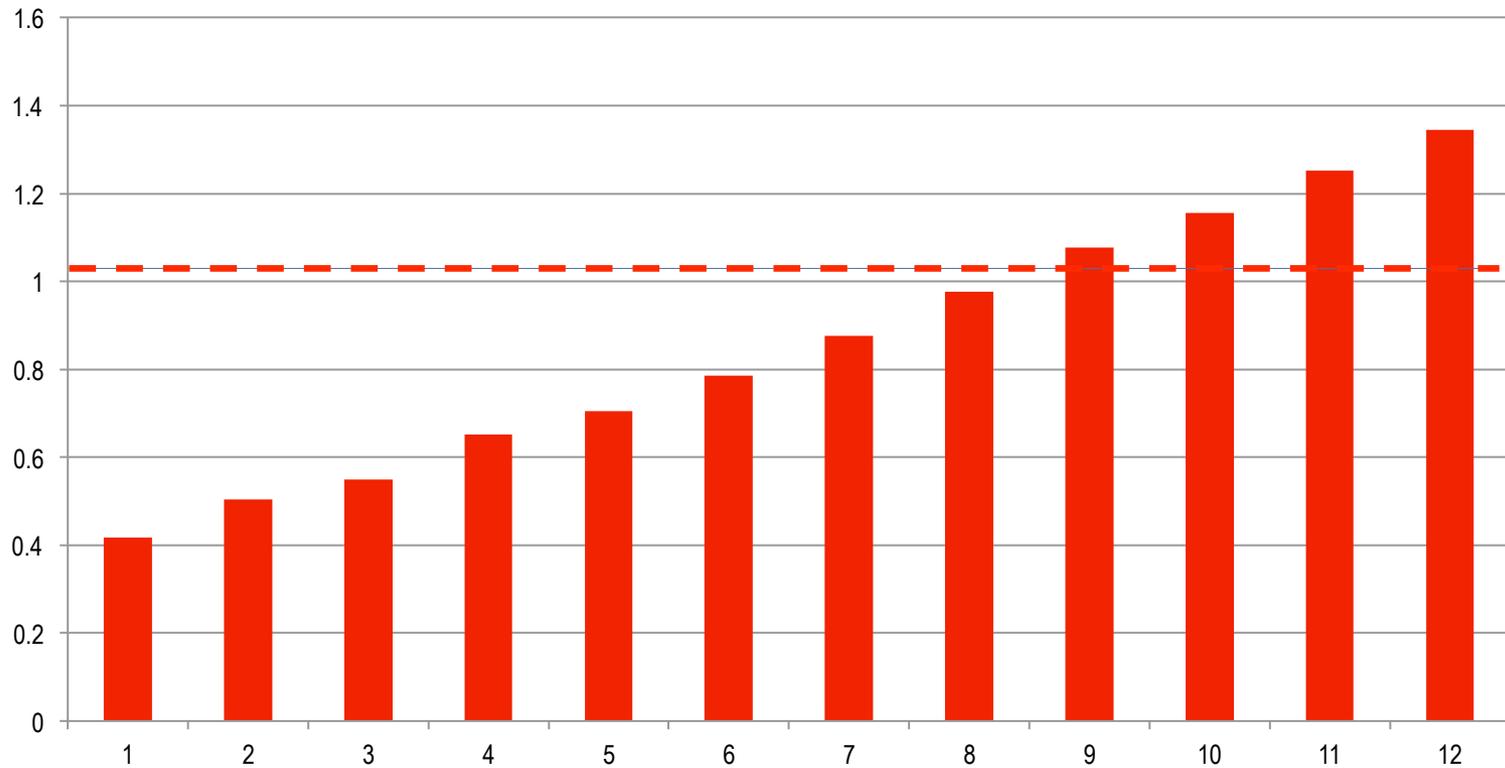
typedef struct

```
{
  double px,py;
  double vx,vy;
  double fx,fy;
  double m;
} body;
```

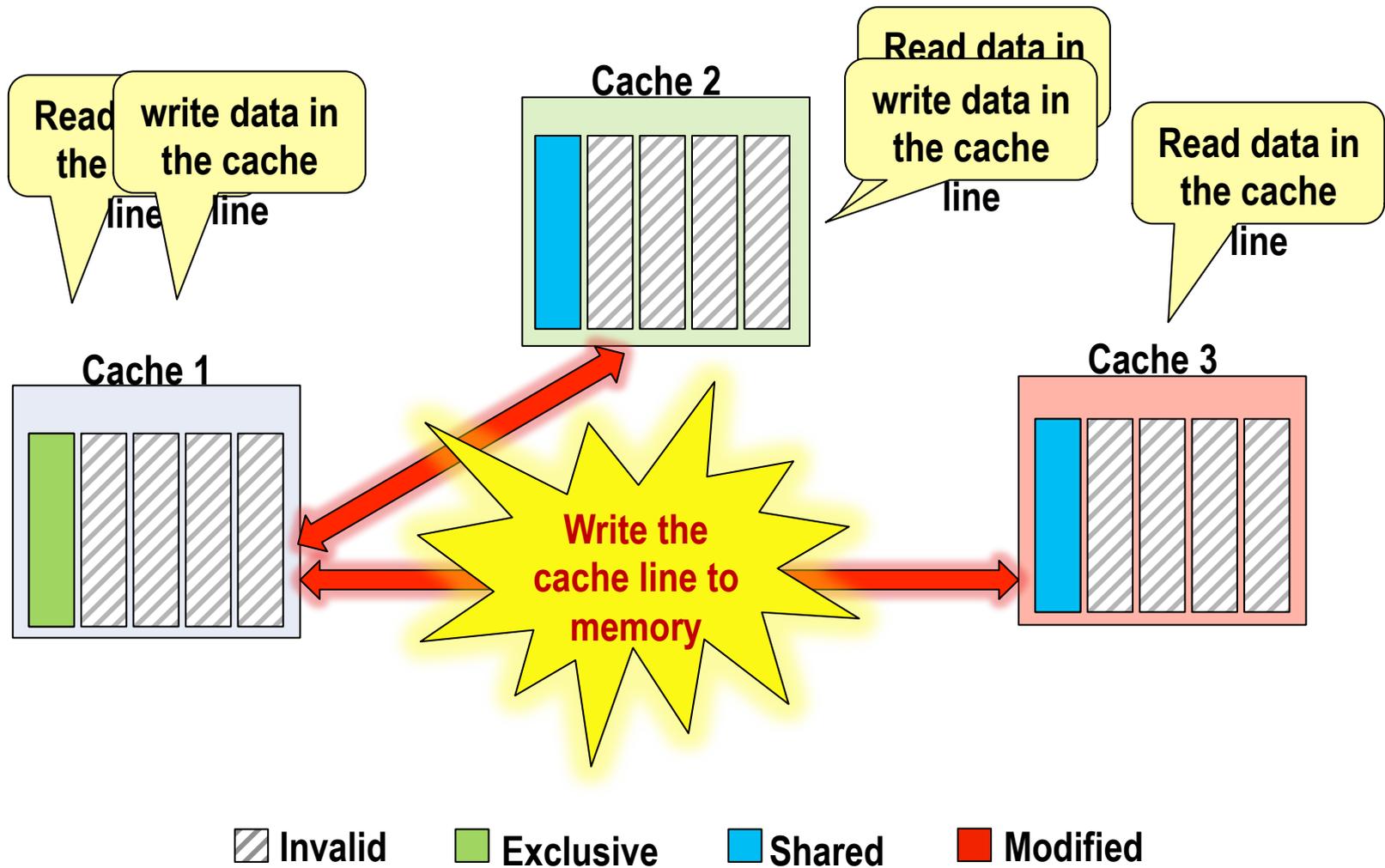
#pragma omp atomic
force updates

Speed up or slow down?!

Original



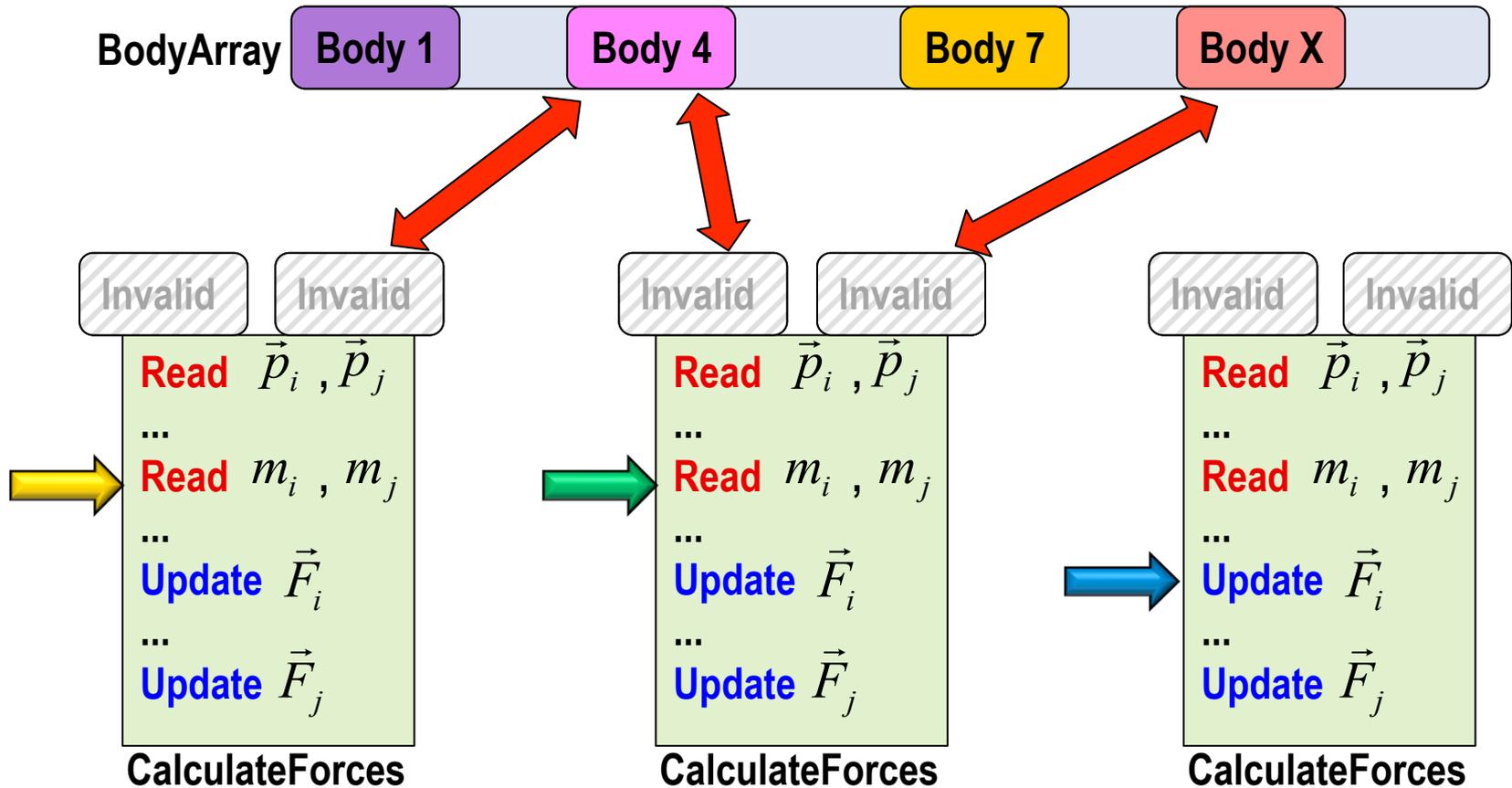
Cache coherence



Communication overheads in force calculations

- **Symmetric updates to the 'force' vector causing false sharing:**
 - Fighting over ownership of the corresponding cachelines.
 - Negative side-effect: No fast access to read-only variable 'position'.
- **Low write-back utilization:**
 - Dirty cache lines are written back to memory before re-updating force fields.
- **Expected communication overheads due to atomic updates.**

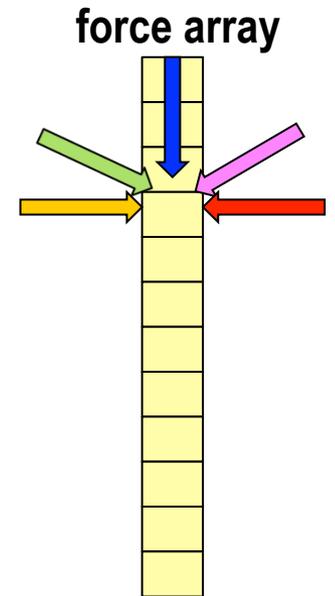
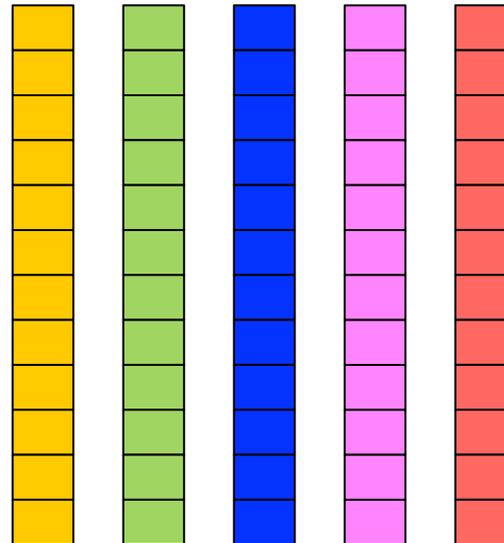
Communication overheads in force calculations



Avoid false sharing!

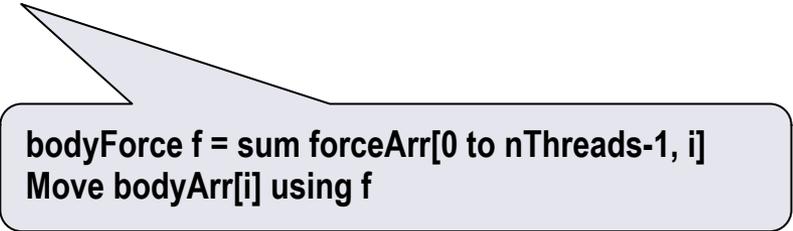
```
typedef struct  
{  
    double px,py;  
    double vx,vy;  
    double fx,fy;  
    double m;  
} body;
```

Avoid atomic updates using thread private force arrays!



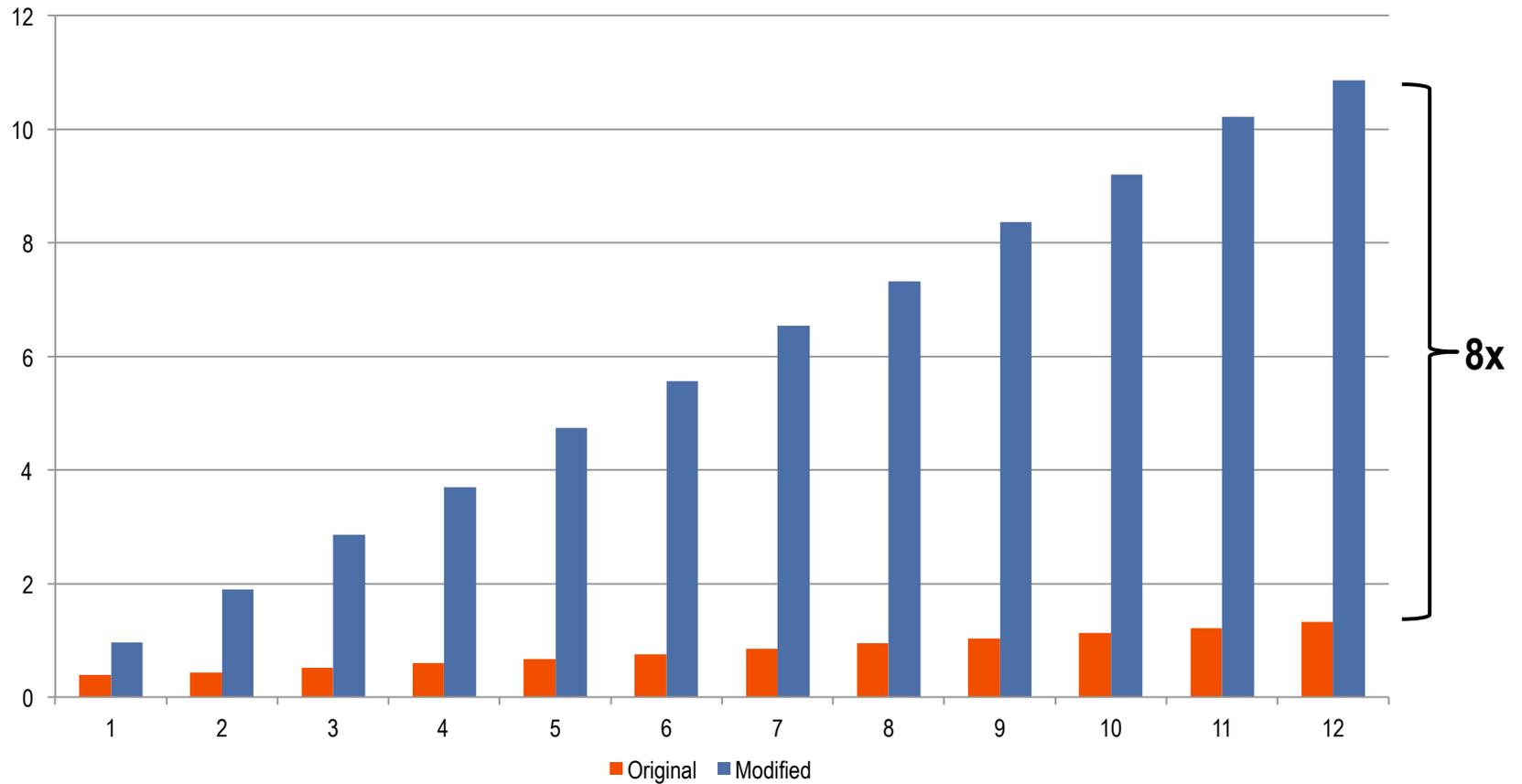
Modified algorithm

```
#pragma omp parallel private(i, j, id)
{
  id=omp_get_thread_num();
  for (time=start to end, step dt)
  {   forceArr[id, 0 to n-1] = 0
    for (i=id to n-1, step nThreads)   //Explicit scheduling, open for smarter load balancing?
      for (j=i+1 to n-1, step 1)
        CalculateForce (bodyArr[i], bodyArr[j], forceArr[id,i], forceArr[id,j]);
    #pragma omp barrier
    for (i=id to n-1, step nThreads)   //move objects
      SumForcesAndMove (bodyArr[i], forceArr, i, nThr, n);
    #pragma omp barrier
  }
}
```



bodyForce f = sum forceArr[0 to nThreads-1, i]
Move bodyArr[i] using f

Overall performance



SUMMARY

Summary

- ParaTools ThreadSpotter is a tool for working with performance for **serial** and **multi-threaded** programs.
- **Large** performance benefits in paying attention to architecture.
- Exploit **locality**, by making sure that data **memory layout** and data **traversal patterns** agree and are **linear**.
- Conserve **memory bandwidth**, **cache space** and avoid **coherency traffic**.

Thank you

ParaTools ThreadSpotter - Report

Acumem ThreadSpotter™
Acumem ThreadSpotter™ is a tool to quickly analyze an application for a range of performance problems, particularly related to multicore optimization.
[Read more... Manual](#)

Your application
Application: ./tripleissue

Memory Bandwidth

The memory bus transports data between the main memory and the processor. The capacity of the memory bus is limited. Abuse of this resource limits application scalability.
[Manual: Bandwidth](#)

Memory Latency

The regularity of the application's memory accesses affects the efficiency of the hardware prefetcher. Irregular accesses causes cache misses, which forces the processor to wait a lot for data to arrive.
[Manual: Cache misses](#) [Manual: Prefetching](#)

Data Locality

Failure to pay attention to data locality has several negative effects. Caches will be filled with unused data, and the memory bandwidth will waste transporting unused data.
[Manual: Locality](#)

Thread Communication / Interaction

Several threads contending over ownership of data in their respective caches causes the different processor cores to stall.
[Manual: Multithreading](#)

This means that your application shows opportunities to:
Tune cache utilization to avoid processor stalls.
[Read more...](#)

ParaTools

Next Steps
The prepared report is divided into sections.

- Select the tab **Summary** to see global statistics for the entire application.
- Select the tabs **Bandwidth Issues**, **Latency Issues** and **MT Issues** to browse through the detected problems.
- Select the tab **Loops** to browse through statistics and detected problems loop by loop.

The Issue and Source windows contain details and annotated source code for the detected problems.

Summary	Source
Issue	
Value details	

Resources
Manual
[Table of Contents](#) [Overview](#)
[Optimization Workflow](#) [Concepts](#)
[Reading the Report](#) [Issue Reference](#)
Rogue Wave Software Web Site
[Rogue Wave Web Site](#) [Tutorials](#)

Summary

Issues	Loops	Summary	Files	Execution	About/Help
Global statistics					
Accesses ?		6.14e+07			
Misses ?		1.48e+06			
Fetches ?		1.96e+06			
Write-backs ?		4.72e+05			
Upgrades ?		0.00e+00			
Miss ratio ?		2.4%			
Fetch ratio ?		3.2%			
Write-back ratio ?		0.8%			
Upgrade ratio ?		0.0%			
Communication ratio ?		0.0%			
Fetch utilization ?		43.7%			
Write-back utilization ?		52.0%			
Communication utilization ?		100.0%			
Analysis parameters					
Processor model ?		Genuine Intel(R) CPU T2500 @ 2.00GHz (auto)			
Number of CPUs ?		1			
Number of caches ?		1			
Cache level ?		2			
Cache size ?		2M			
Line size ?		64			
Replacement policy ?		random			
Software prefetches active		Yes			

Miss/Fetch ratio ?

— Fetch ratio
..... Utilization corrected fetch ratio
— Miss ratio

Write-back ratio ?

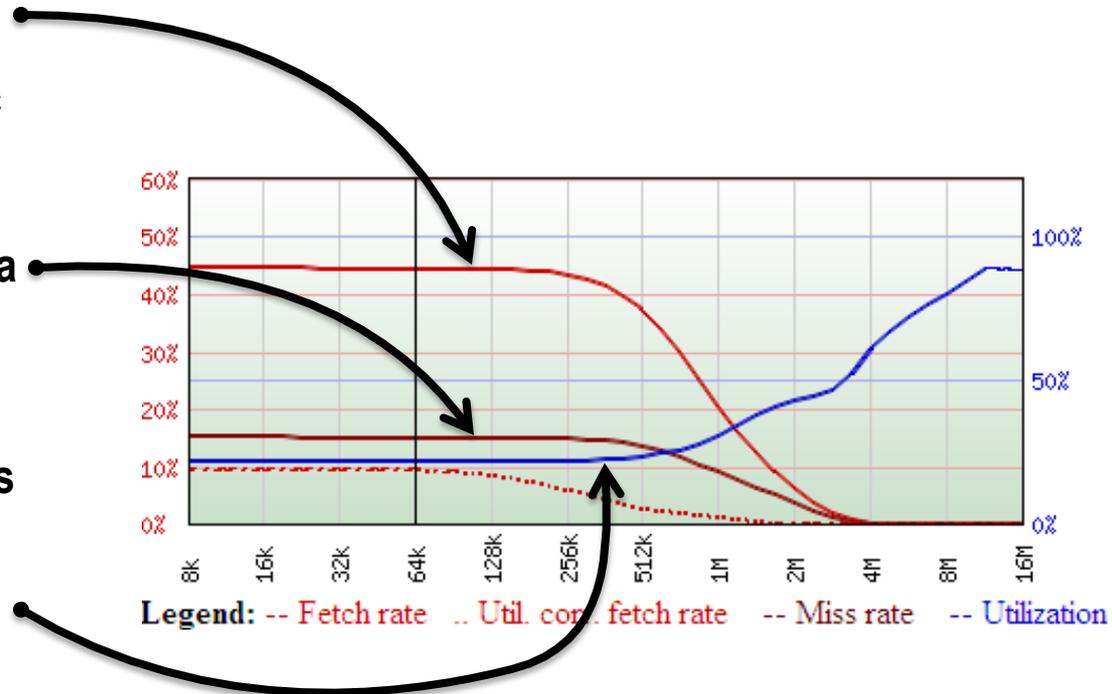
— Write-back ratio
..... Utilization corrected write-back ratio

Utilization ?

— Fetch utilization
— Write-back utilization

Metrics as a function of cache size

- **Fetch ratio**
 - The likelihood that a memory access causes memory bus traffic
- **Miss ratio**
 - The likelihood that a memory access doesn't find requested data in the cache
- **Fetch utilization**
 - How much of every fetched cache line that the application really uses



Issues by Severity

Issues							
Bandwidth Issues		Latency Issues		Multi-Threading Issues		Pollution Issues	
#	Issue type	% of bandwidth	% of fetches	% of write-backs	Fetch utilization	Write-back utilization	
8	Fetch utilization	50.0%	49.8%	50.8%	50.0%	51.0%	
9	Write back utilization	50.0%	49.8%	50.8%	50.0%	51.0%	
6	Fetch utilization	29.4%	24.7%	49.2%	25.1%	53.0%	
7	Write back utilization	29.4%	24.7%	49.2%	25.1%	53.0%	
10	Fetch utilization	20.5%	25.5%	0.0%	49.2%	100.0%	

Copyright (c) 2006-2011 Rogue Wave Software, Inc. All Rights Reserved.
Patents pending.

Issue #8: Fetch utilization ?

This instruction group also show symptoms of: Fetch hot-spot, Write-back hot-spot.

+ Statistics for instructions of this issue ?

+ Instructions involved in this issue ?

+ Instructions previously writing to related data ?

+ Loop statistics ?

+ Loop instructions ?

Copyright (c) 2006-2011 Rogue Wave Software, Inc. All Rights Reserved.
Patents pending.

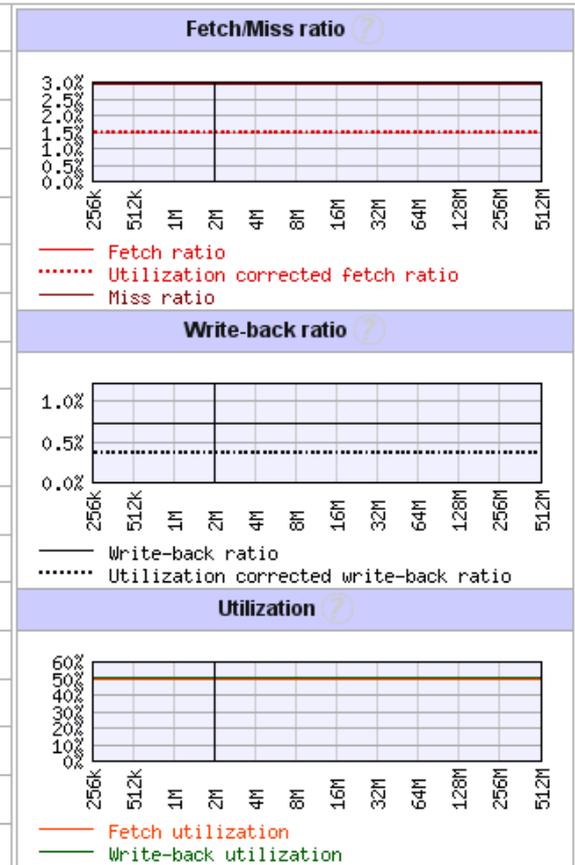
Statistics of an Issue

Issues Loops Summary Files Execution About/Help

Bandwidth Issues Latency Issues Multi-Threading Issues Pollution Issues

Statistics for instructions of this issue ?

Accesses ?	3.27e+07
% of misses ?	66.1%
% of bandwidth ?	50.0%
% of fetches ?	49.8%
% of write-backs ?	50.8%
% of upgrades ?	---
Miss ratio ?	3.0%
Fetch ratio ?	3.0%
Write-back ratio ?	0.7%
Upgrade ratio ?	0.0%
Communication ratio ?	0.0%
Fetch utilization ?	50.0%
Write-back utilization ?	51.0%
Communication utilization ?	100.0%
False sharing ratio ?	0.0%
HW prefetch probability ?	0.0%
Access randomness ?	Low
Worst instruction ?	tripleissue!main()+0x6f(0x80490ad) [R], tripleissue.cpp:53



If the program was changed as to reach 100% fetch utilization, fetches in this instruction group would be reduced with 50.0%, and total number of fetches would be reduced with 24.9%.

Reference to Source Code

```

45 //Partially used structure
46
47 TS_sampler_start(200);
48
49 pMyData = new DATA[10*1024*1024];
50
51 + 24.7% for (long i=0; i<10*1024*1024; i++)
52 {
53 - 75.3% pMyData[i].a = pMyData[i].b;

```

% of fetches	Miss ratio	Fetch ratio	WB ratio	Fetch Util	WB Util	PC	Type	Issues
14.1%	3.4%	3.4%	0.0%	49.2%	100.0%	0x8049096	R	
12.5%	3.0%	3.0%	0.0%	50.0%	51.0%	0x804909b	R	
11.4%	2.7%	2.7%	0.0%	49.2%	100.0%	0x80490a8	R	
12.7%	3.0%	3.0%	0.0%	50.0%	51.0%	0x80490ad	R	
12.8%	0.0%	4.1%	0.0%	25.1%	53.0%	0x80490b9	R	
11.9%	0.0%	3.8%	3.8%	25.1%	53.0%	0x80490bc	W	

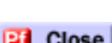
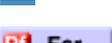
```

54 }
55
56 delete pMyData;
57
58 TS_sampler_stop();
59
60
61
62 //Inefficient loop nesting
63
64 char * p = new char[SIZE];
65
66 long nbRows = NBROWS;
67 long sRowSize = ROWSIZE;
68

```

Used Icons

Slowspot Issues

-  Fetch utilization
-  Write back utilization
-  Communication utilization
-  Inefficient loop nesting
-  Random access
-  Prefetch: too close
-  Prefetch: too distant
-  Prefetch: unnecessary
-  False sharing

Opportunity issues

-  Spatial blocking
-  Temporal blocking
-  Spat/temp blocking
-  Loop fusion
-  Non-temporal data
-  Non-temporal store possible
-  Fetch hot-spot
-  Write-back hot-spot
-  Communication hot-spot

Resource Sharing Example

Libquantum

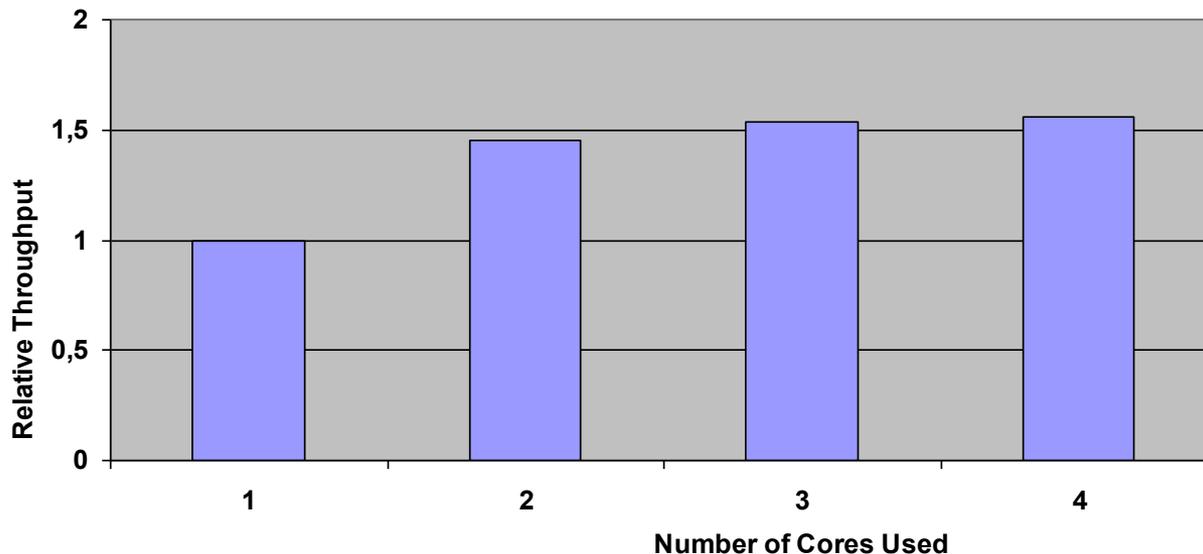
A quantum computer simulation

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

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: - Arguments:
 - Working directory:
 - Attach to running application
 - Pid:
- Read sample file
 - Sample file:

Report generation

- Generate report in: - Report name:
- Cache size: bytes
- Launch web browser:

Libquantum:
Orig code
Spatial opt
Spat + Loop fusion

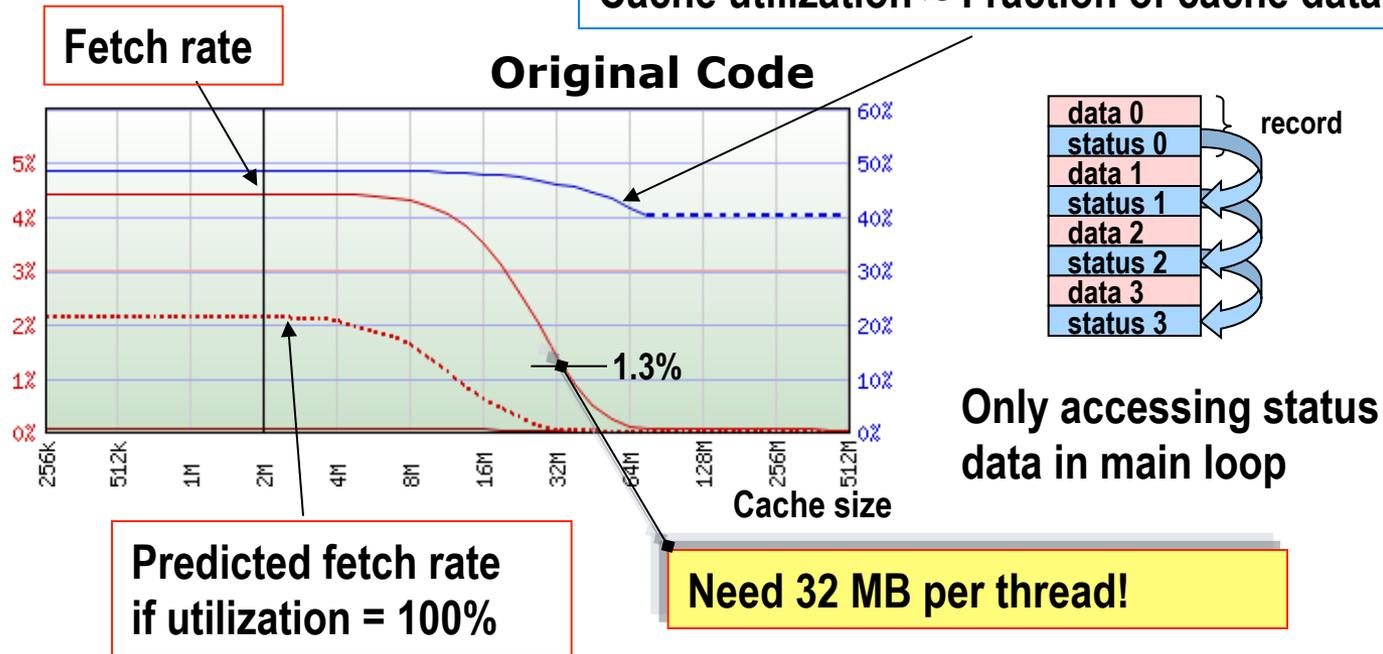
Edit-compile-analysis cycle \approx 1min

[erik@localhost... [koko on 192.1... [Acumem Slow... [erik] [demos] Acumem SlowS... file:/// - 8.4. Loo... Starting Take Scr...

Utilization Analysis

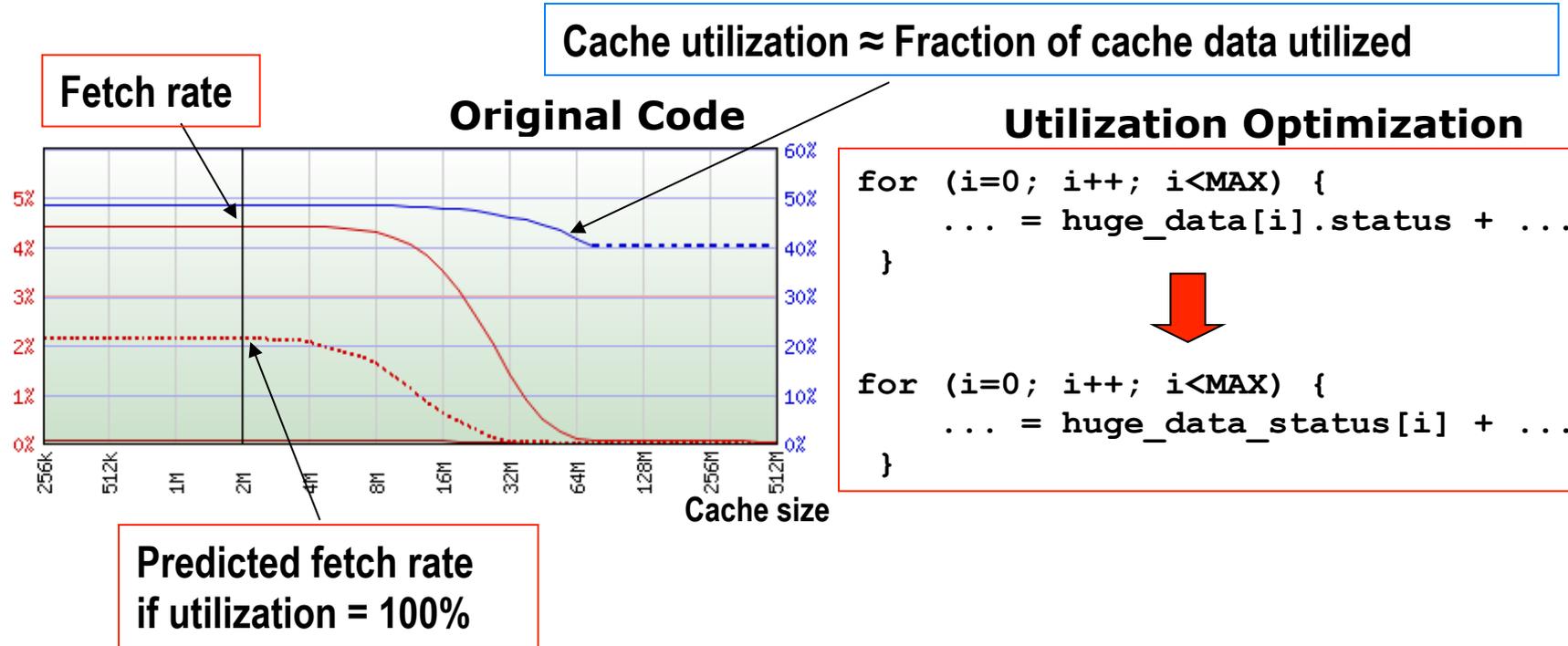
Libquantum

Cache utilization \approx Fraction of cache data utilized



Utilization Analysis

Libquantum

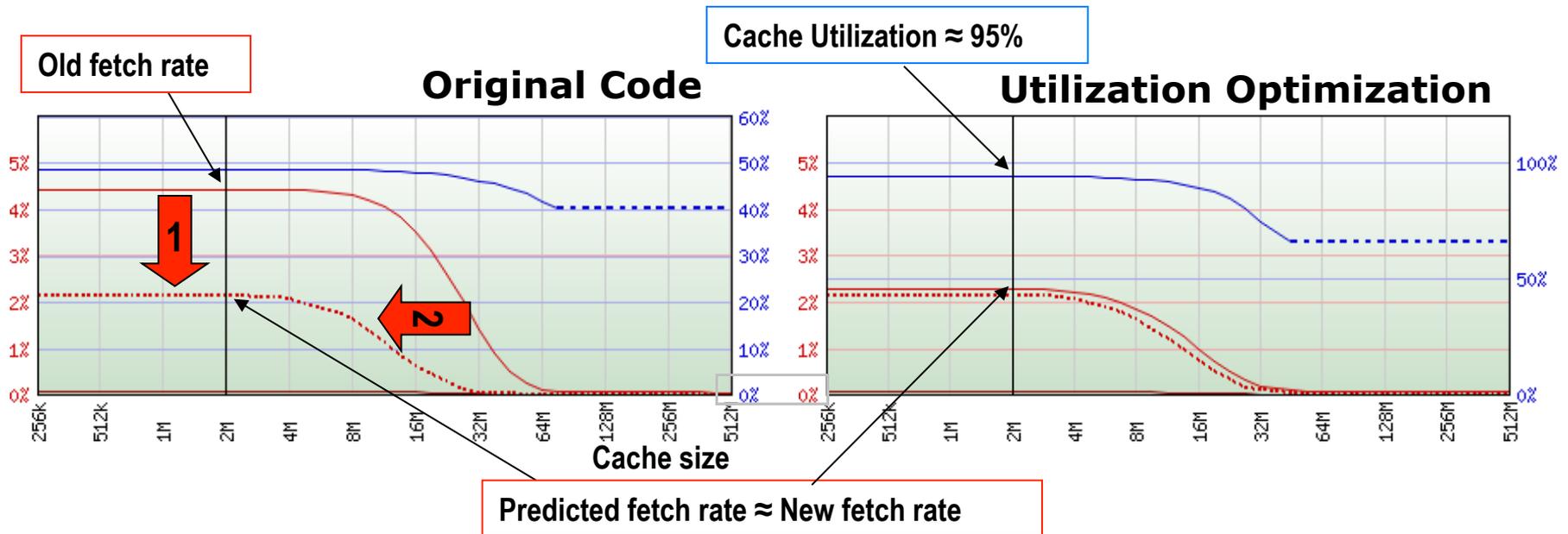


ParaTools ThreadSpotter's First Advice: Improve Utilization

→ Change one data structure

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

Utilization Optimization

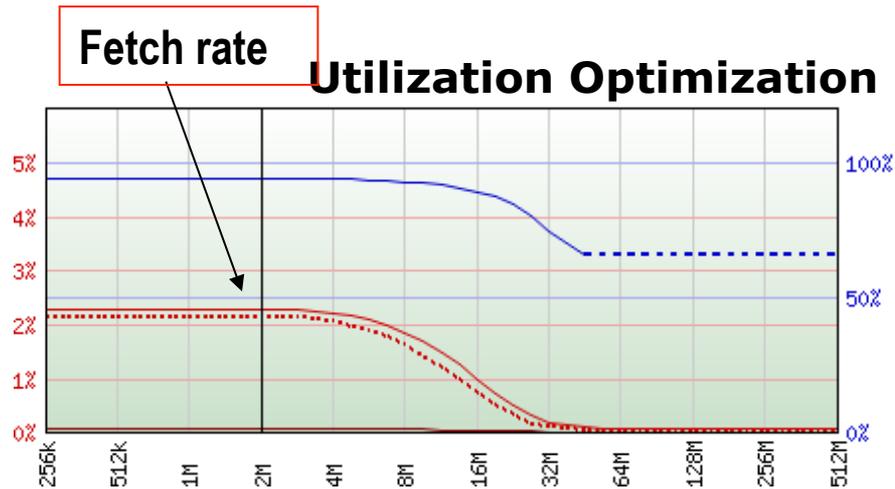


Two positive effects from better utilization

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

Loop Fusion

Libquantum



Utilization + Fusion Optimization

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

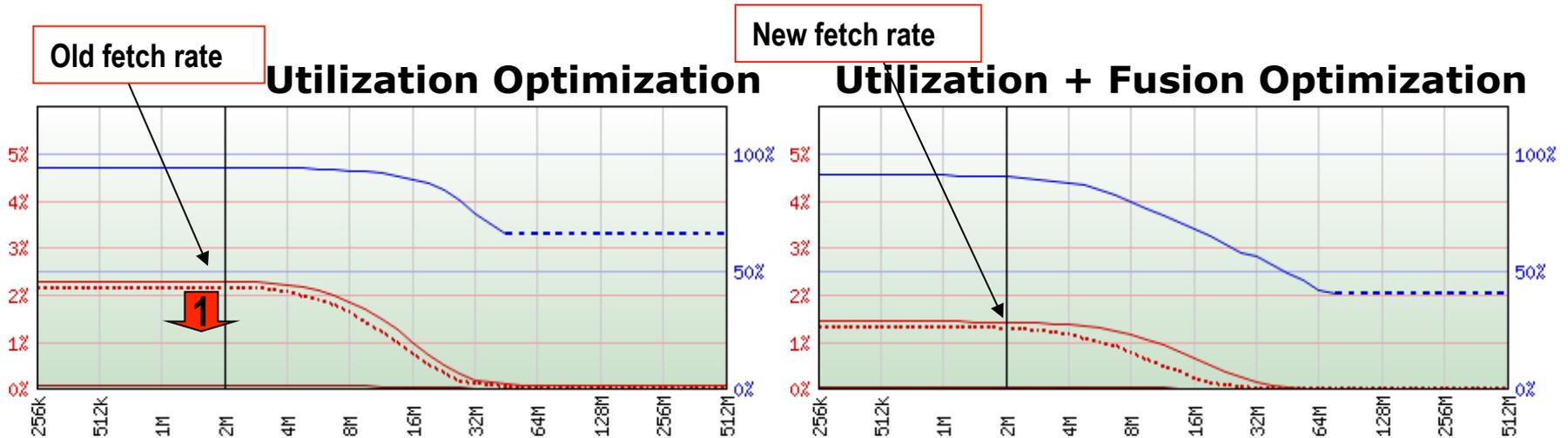
Second-Fifth ParaTools ThreadSpotter Advice: Improve reuse of data through loop fusion

➔ **Fuse functions traversing the same data**

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

Effect: Loop Fusion

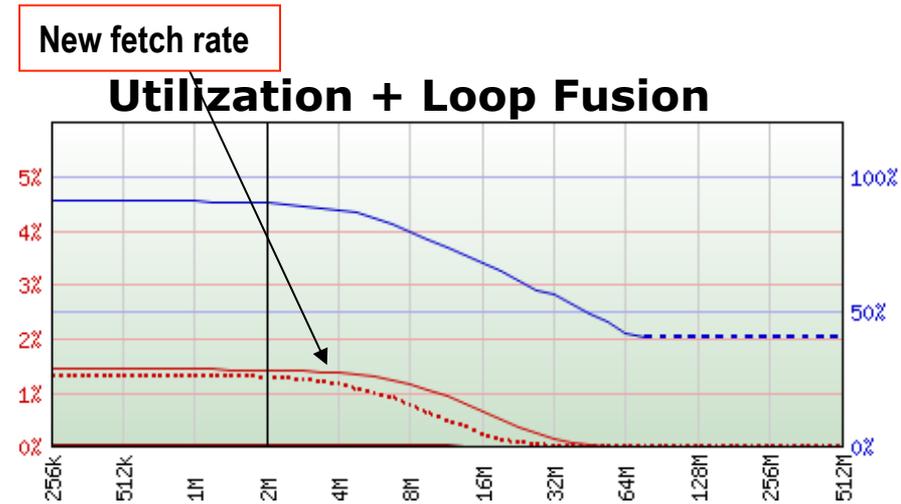
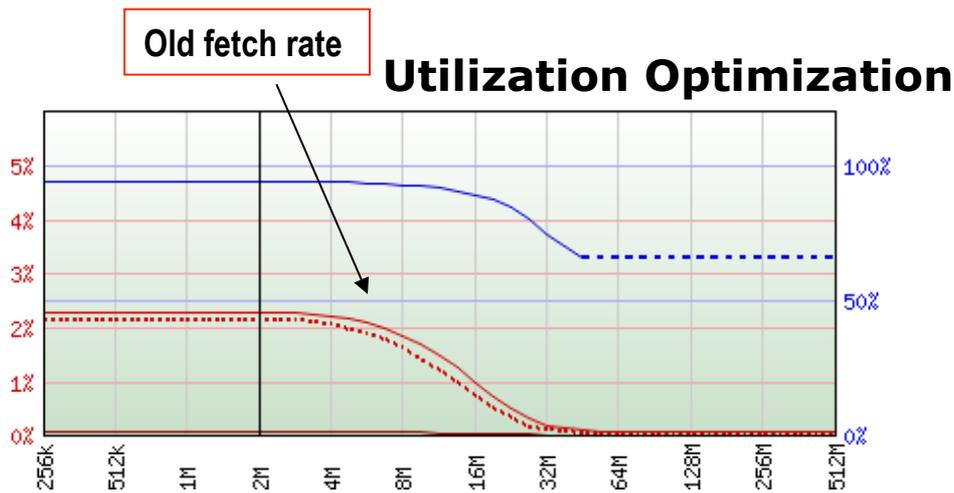
SPEC CPU2006-462.libquantum



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

Utilization + Loop Fusion

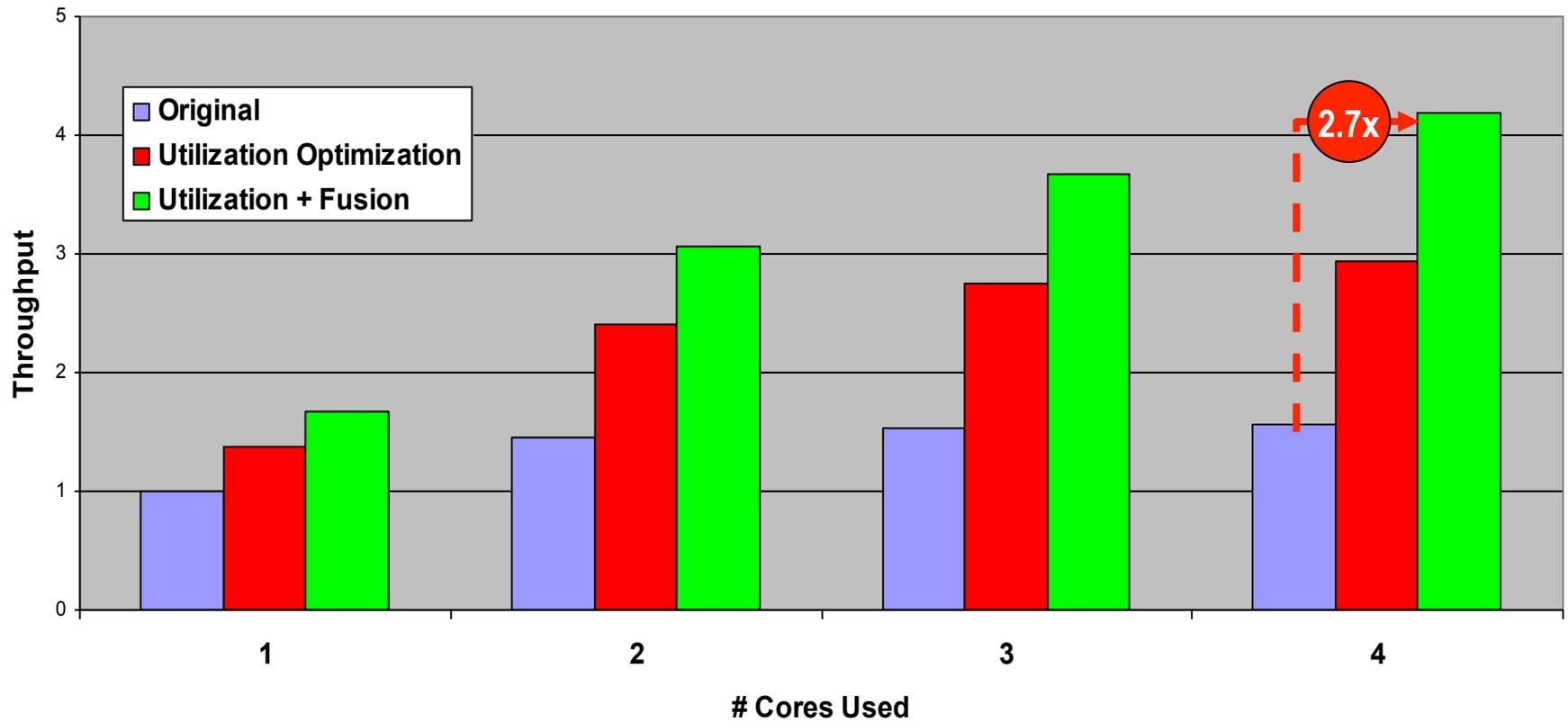
Libquantum



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

Summary

Libquantum



Another Demo – N-body

Simulation of Gravitational N-body Problem

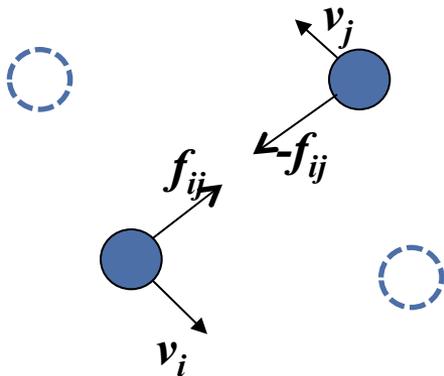
- Initialize bodies
- for time = start to end step by Δt
 - calculate forces
 - move bodies
- end for time

```

for each body i=1 to n
   $d\vec{v} = \vec{f}_i / m_i \times \Delta t$ 
   $d\vec{p} = (\vec{v}_i + d\vec{v} / 2) \times \Delta t$ 
   $\vec{v}_{i+} = d\vec{v}$ 
   $\vec{p}_{i+} = d\vec{p}$ 
   $\vec{f}_i = \vec{0}$ 
end for each
  
```

```

for each body i=1 to n-1
  for each neighbour j=i+1 to n
    calculate
       $\vec{f}_{ij}$ 
       $\vec{f}_{i+} = \vec{f}_{ij}$ 
       $\vec{f}_{j-} = \vec{f}_{ij}$ 
    end for j
  end for i
  
```



Algorithm

```
#pragma omp parallel private(i,j)  
#pragma omp for  
for (time=start to end, step dt)  
{  
    for(i=0 to n, step 1)  
        for(j=i+1 to n, step 1)  
            CalculateForce(bodyArr[i], bodyArr[j]);  
  
    #pragma omp for  
    for(i=0 to n, step 1)  
        Move(bodyArr[i]);  
}
```

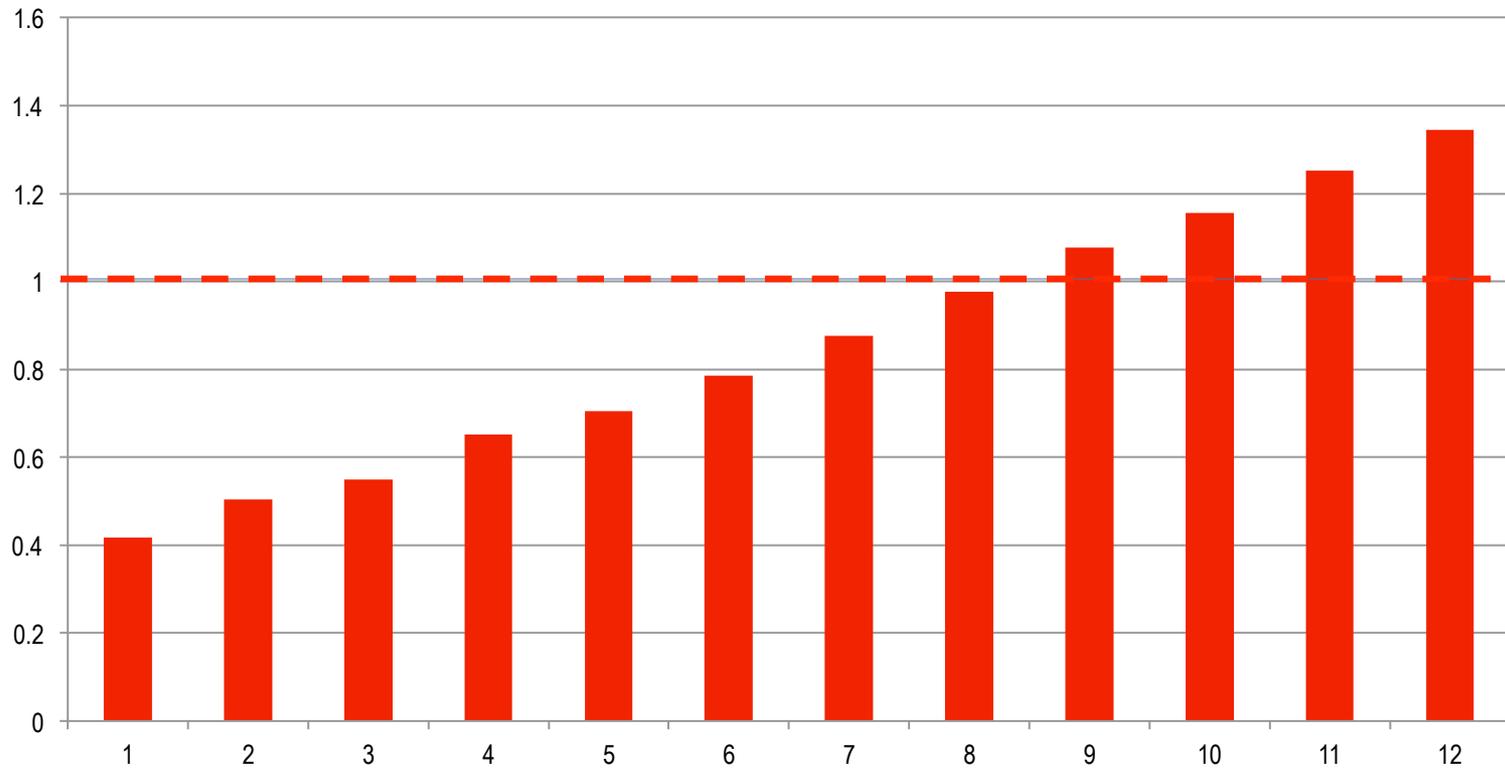
typedef struct

```
{  
    double px,py;  
    double vx,vy;  
    double fx,fy;  
    double m;  
} body;
```

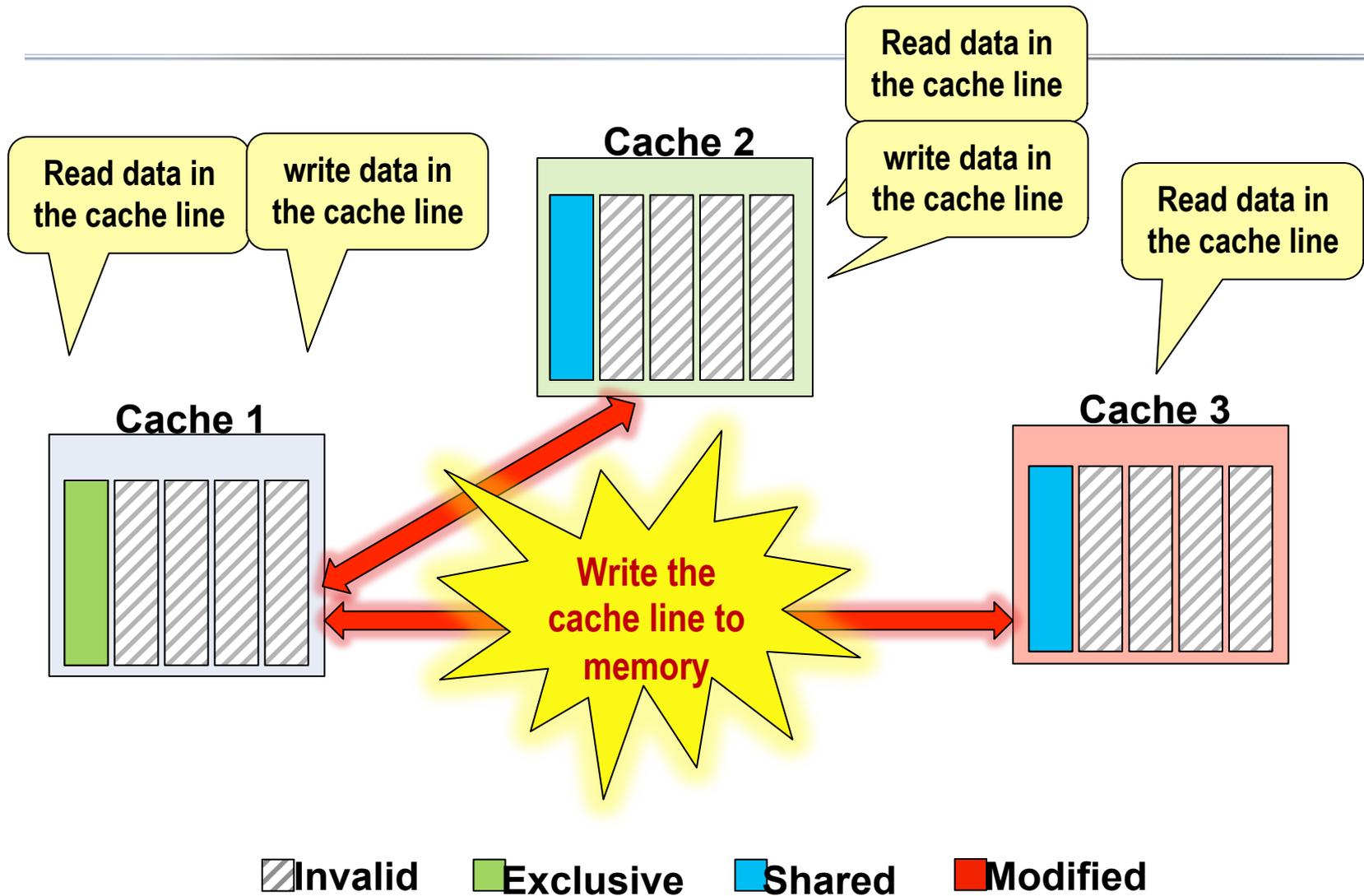
#pragma omp atomic
force updates

Speed Up or Slow Down?

Original



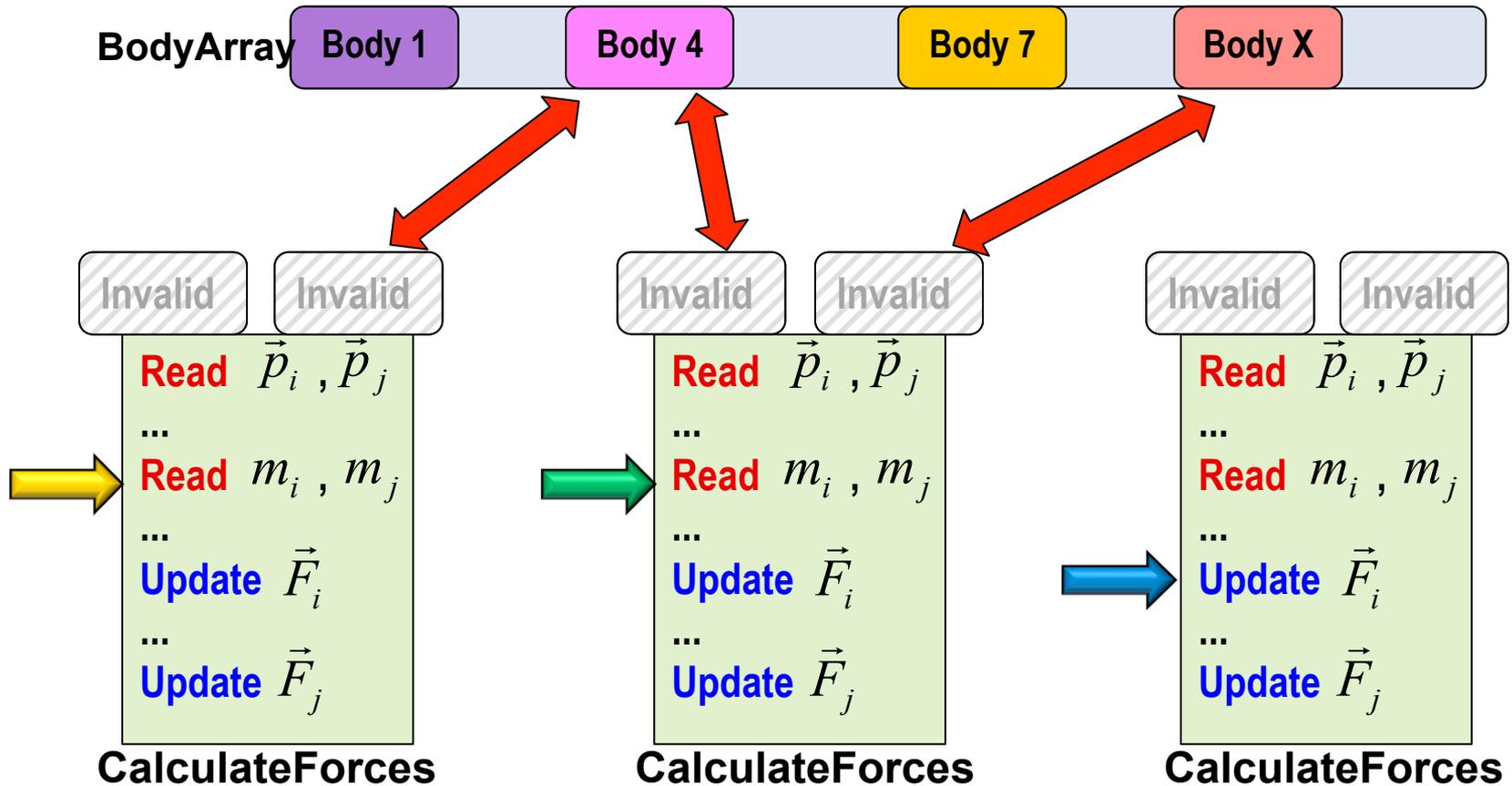
Cache Coherence



Communication Overheads in Force Calculations

- **Symmetric updates to the 'force' vector causing false sharing:**
 - Fighting over ownership of the corresponding cachelines.
 - Negative side-effect: No fast access to read-only variable 'position'.
- **Low write-back utilization:**
 - Dirty cache lines are written back to memory before re-updating force fields.
- **Expected communication overheads due to atomic updates.**

Communication Overheads in Force Calculations



Avoid False Sharing

```
typedef struct
```

```
{
```

```
    double px,py;
```

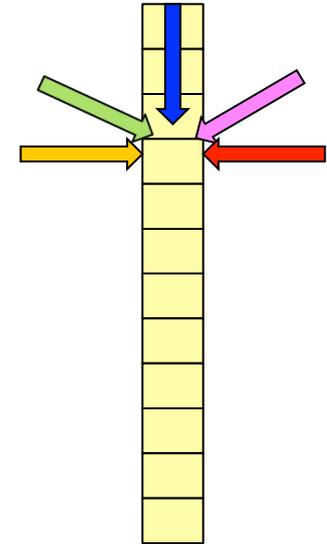
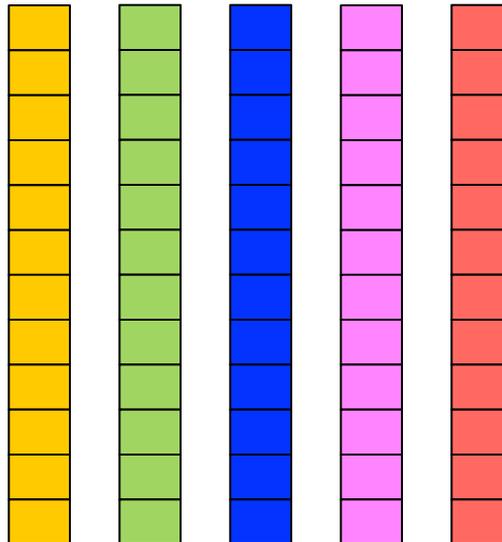
```
    double vx,vy;
```

```
    double fx,fy;
```

```
    double m;
```

```
} body;
```

Avoid atomic updates using thread private force buffer!



Modified Algorithm

```
#pragma omp parallel private(i, j, id)
{
  id=omp_get_thread_num();
  for (time=start to end, step dt)
  {
    forceArr[id, 0 to n] = 0
    for (i=id to n, step nThreads) //Now able to scatter for load balancing
      for (j=i+1 to n, step 1)
        CalculateForce (bodyArr[i], bodyArr[j], forceArr[id,i], forceArr[id,j] );
    #pragma omp barrier
    for (i=id to n, step nThreads) //move objects
      SumForcesAndMove (bodyArr[i], forceArr, i, nThr, n);
    #pragma omp barrier
  }
}
```

bodyForce f = sum forceArr[0 to nThreads, i]
Move bodyArr[i] using f

Overall Performance

