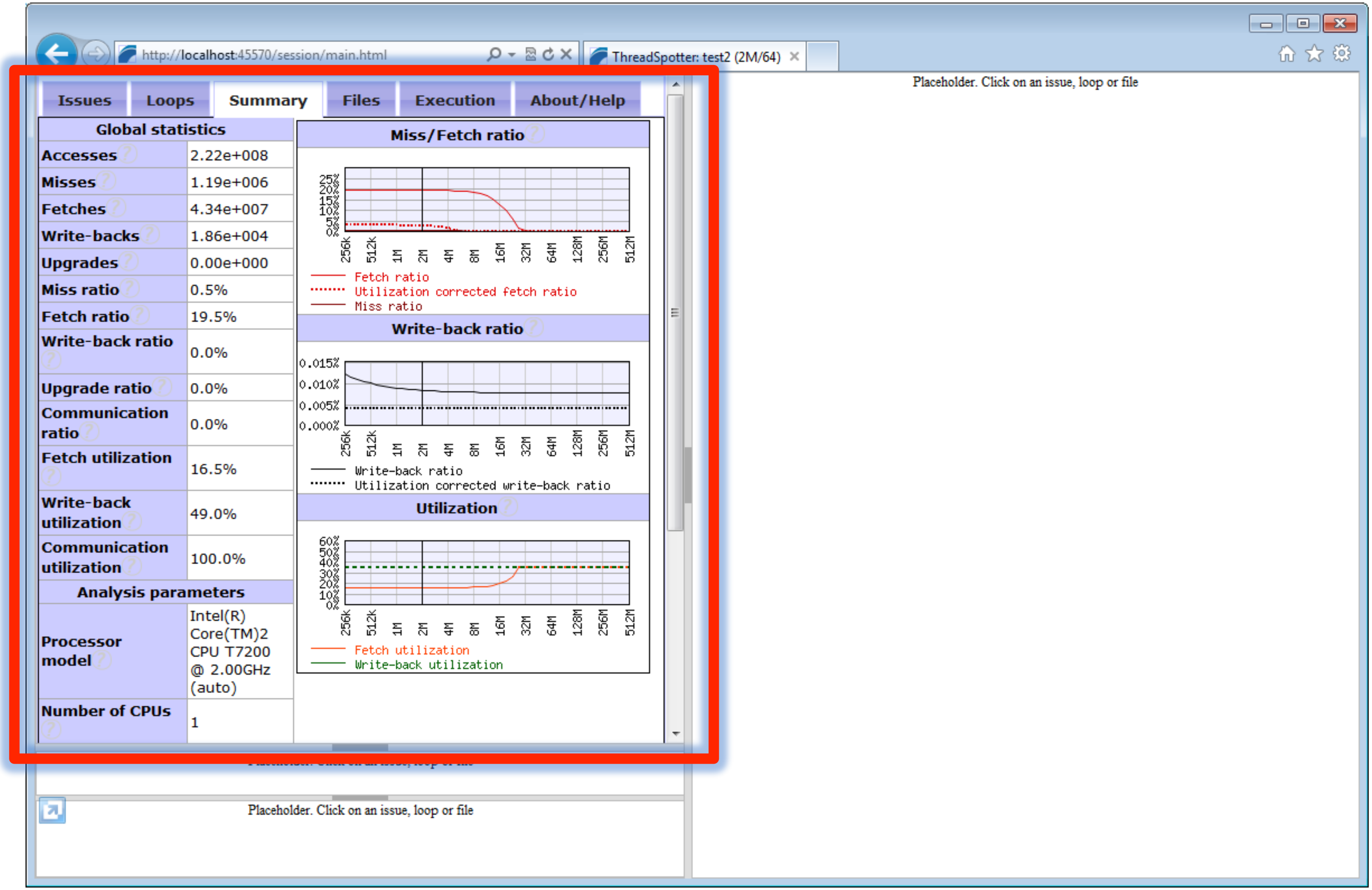
Summary, global statistics

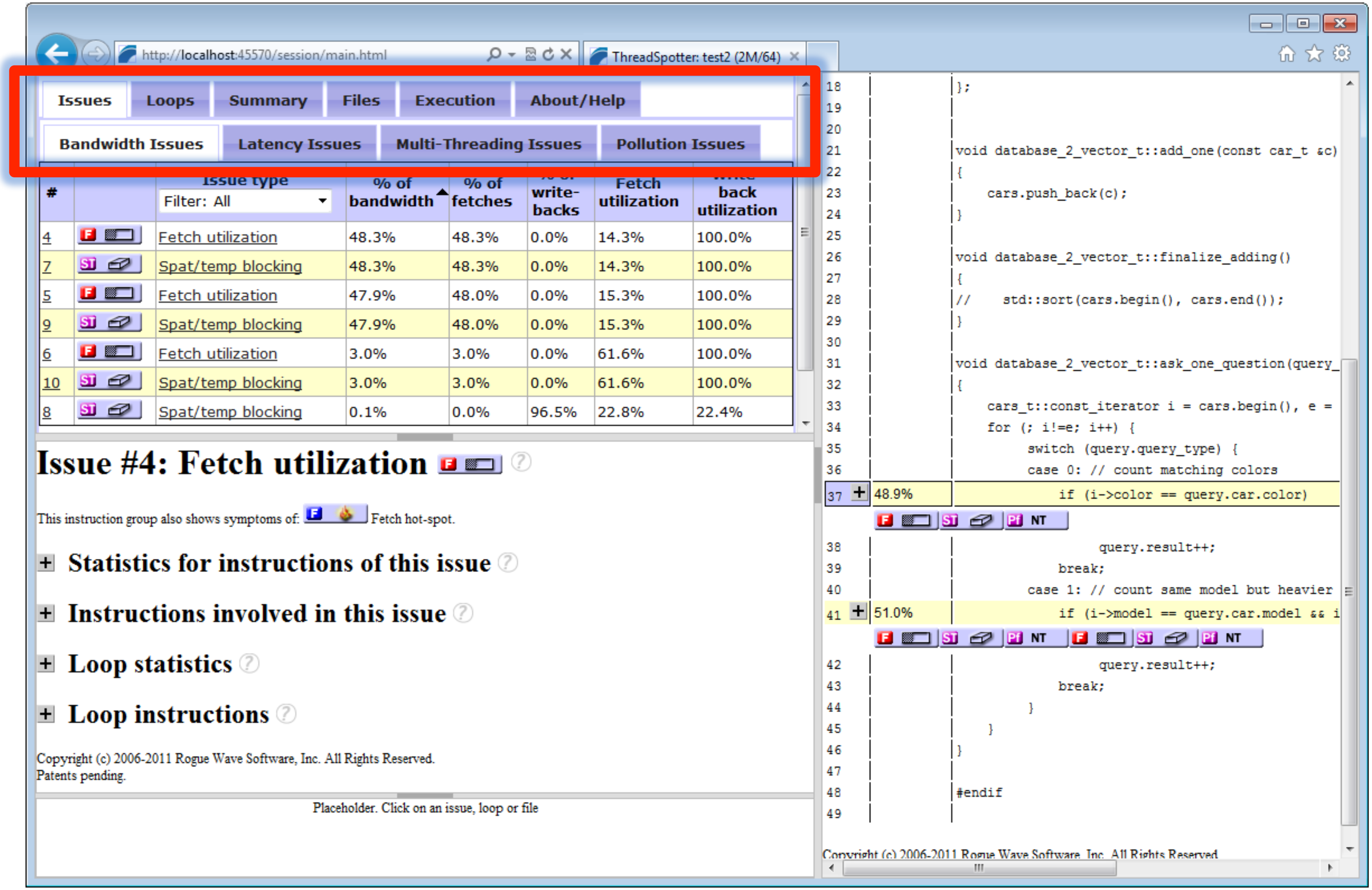


Global statistics	
Accesses	2.22e+008
Misses	1.19e+006
Fetches	4.34e+007
Write-backs	1.86e+004
Upgrades	0.00e+000
Miss ratio	0.5%
Fetch ratio	19.5%
Write-back ratio	0.0%
Upgrade ratio	0.0%
Communication ratio	0.0%
Fetch utilization	16.5%
Write-back utilization	49.0%
Communication utilization	100.0%
Analysis parameters	
Processor model	Intel(R) Core(TM)2 CPU T7200 @ 2.00GHz (auto)
Number of CPUs	1

Placeholder. Click on an issue, loop or file

Placeholder. Click on an issue, loop or file

Report sections



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

Bandwidth Issues | Latency Issues | Multi-Threading Issues | Pollution Issues

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

Issue #4: Fetch utilization

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

+ Statistics for instructions of this issue

+ Instructions involved in this issue

+ Loop statistics

+ Loop instructions

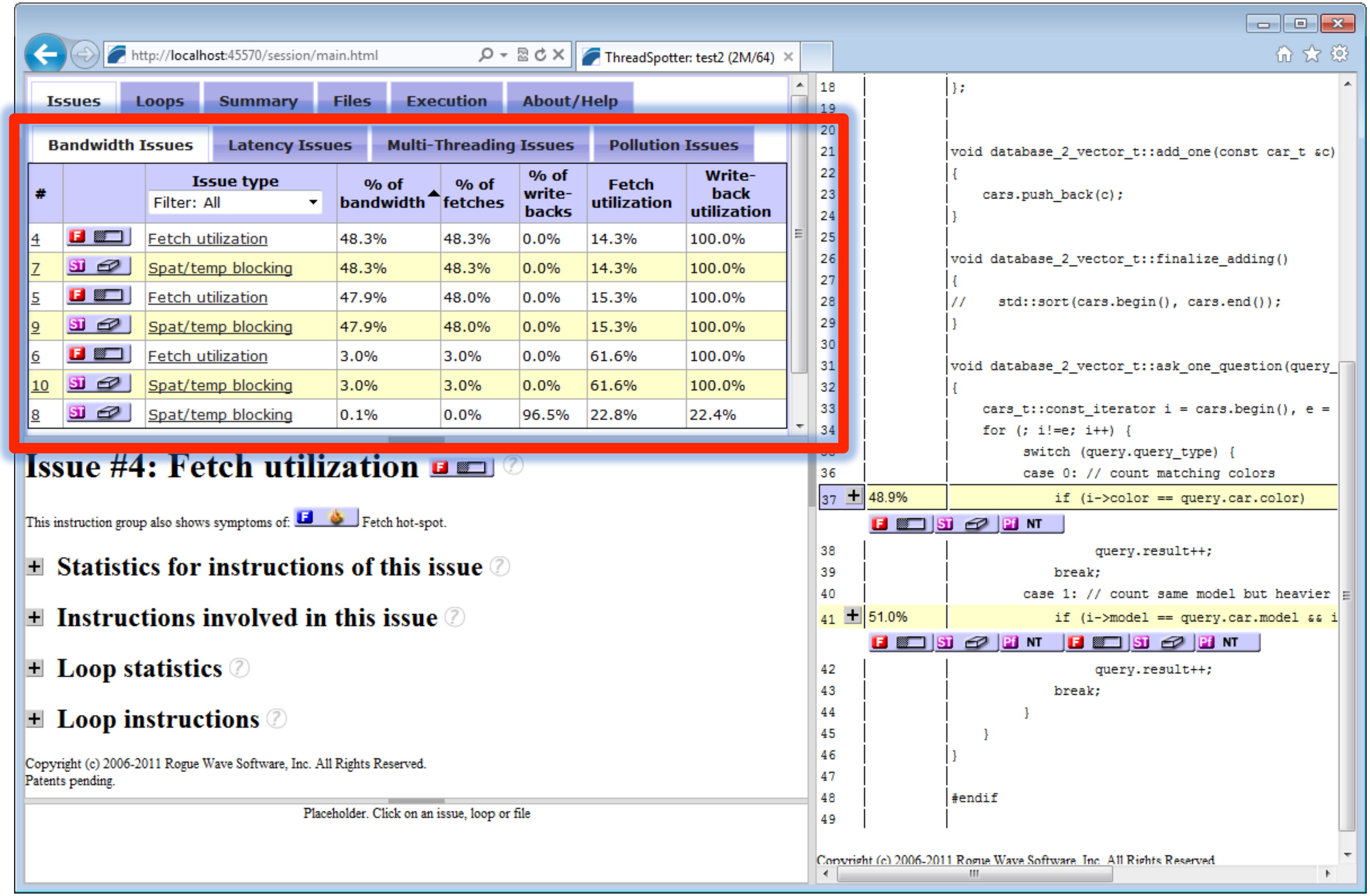
Copyright (c) 2006-2011 Rogue Wave Software, Inc. All Rights Reserved.
Patents pending.

Placeholder. Click on an issue, loop or file

```
};


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_
{
    cars_t::const_iterator i = cars.begin(), e =
    for (; i!=e; i++) {
        switch (query.query_type) {
        case 0: // count matching colors
+ 48.9%     if (i->color == query.car.color)
            F  ST  Pf NT
                query.result++;
            break;
        case 1: // count same model but heavier
+ 51.0%     if (i->model == query.car.model && i
            F  ST  Pf NT  F  ST  Pf NT
                query.result++;
            break;
        }
    }
}

#endif
```

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

Navigation by issues



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

Issue #4: Fetch utilization

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

+ Statistics for instructions of this issue

+ Instructions involved in this issue

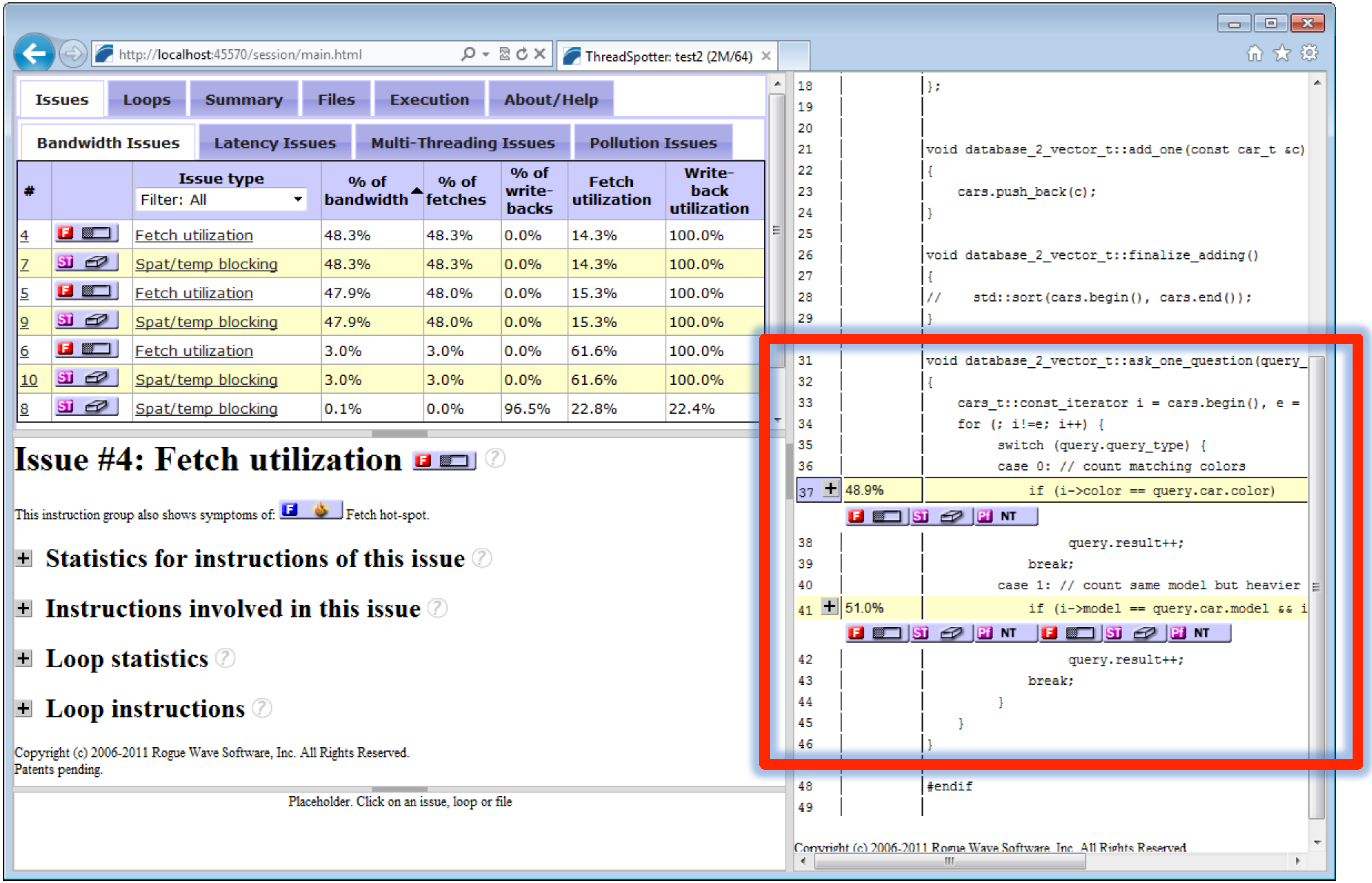
+ Loop statistics

+ Loop instructions

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_
{
    cars_t::const_iterator i = cars.begin(), e =
    for (; i!=e; i++) {
        switch (query.query_type) {
        case 0: // count matching colors
48.9%           if (i->color == query.car.color)
                    query.result++;
                break;
        case 1: // count same model but heavier
51.0%           if (i->model == query.car.model && i
                    query.result++;
                break;
        }
    }
}

#endif
```

Source code annotation



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

Issue #4: Fetch utilization F

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

+ Statistics for instructions of this issue

+ Instructions involved in this issue

+ Loop statistics

+ Loop instructions

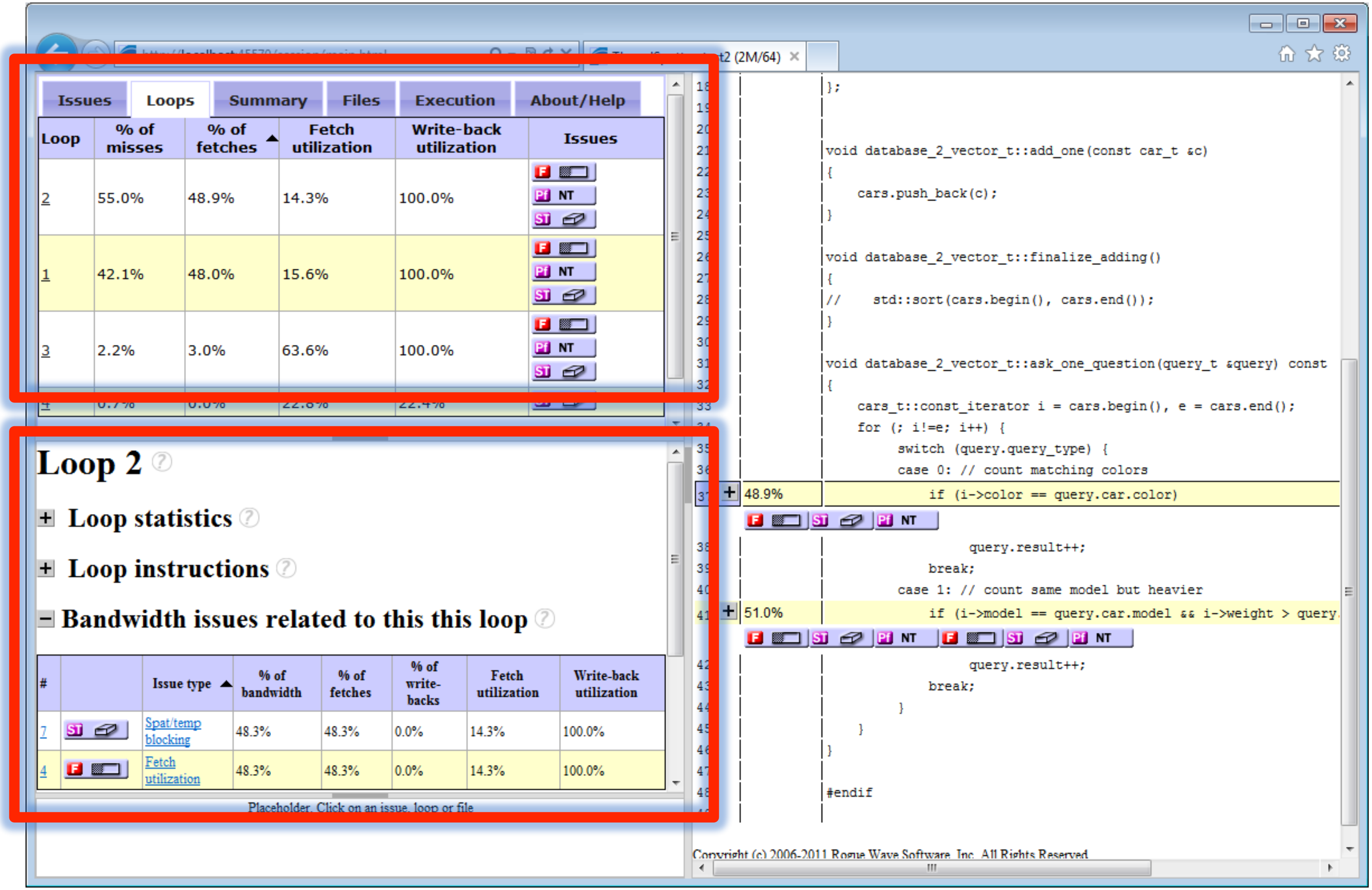
Copyright (c) 2006-2011 Rogue Wave Software, Inc. All Rights Reserved.
Patents pending.

Placeholder. Click on an issue, loop or file

```
                  };


                  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_
                  {
                      cars_t::const_iterator i = cars.begin(), e =
                      for (; i!=e; i++) {
                          switch (query.query_type) {
                          case 0: // count matching colors
+ 48.9%                       if (i->color == query.car.color)
      F  ST  Pf NT
                                  query.result++;
                              break;
                          case 1: // count same model but heavier
+ 51.0%                       if (i->model == query.car.model && i
      F  ST  Pf NT  F  ST  Pf NT
                                  query.result++;
                              break;
                          }
                      }
                  }

                  #endif
```

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

Navigation by loops



Loop	% of misses	% of fetches	Fetch utilization	Write-back utilization	Issues
2	55.0%	48.9%	14.3%	100.0%	F Pf NT ST
1	42.1%	48.0%	15.6%	100.0%	F Pf NT ST
3	2.2%	3.0%	63.6%	100.0%	F Pf NT ST
4	0.7%	0.0%	22.8%	22.4%	

Loop 2

+ Loop statistics

+ Loop instructions

− Bandwidth issues related to this this loop

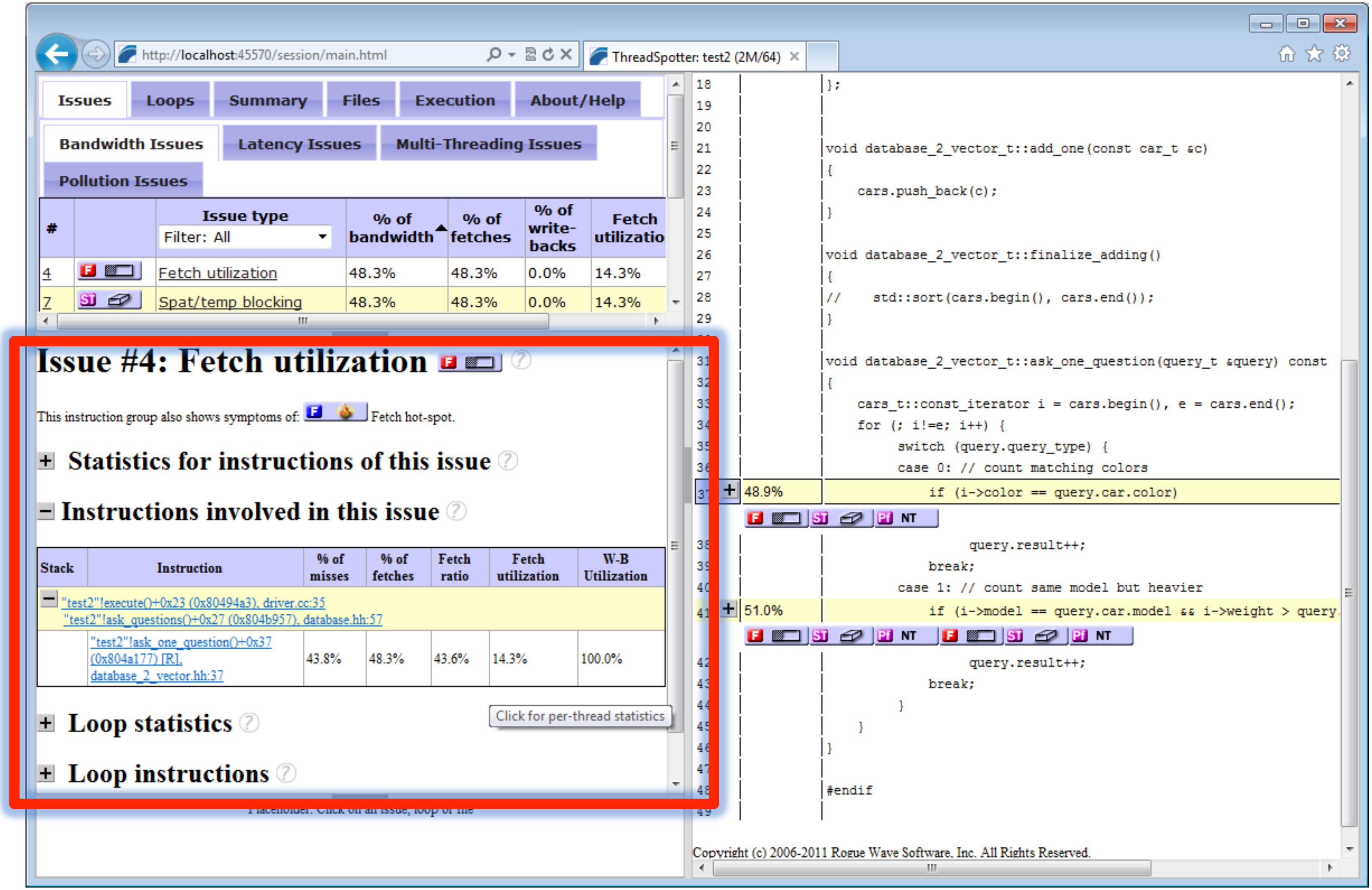
#		Issue type	% of bandwidth	% of fetches	% of write-backs	Fetch utilization	Write-back utilization
7	ST	Spat/temp blocking	48.3%	48.3%	0.0%	14.3%	100.0%
4	F	Fetch utilization	48.3%	48.3%	0.0%	14.3%	100.0%

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
48.9%       if (i->color == query.car.color)
                query.result++;
            break;
        case 1: // count same model but heavier
51.0%       if (i->model == query.car.model && i->weight > query.
                query.result++;
            break;
        }
    }
}

#endif
```

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

Issue details



#		Issue type Filter: All	% of bandwidth	% of fetches	% of write-backs	Fetch utilizatio
4	F	Fetch utilization	48.3%	48.3%	0.0%	14.3%
7	ST	Spat/temp blocking	48.3%	48.3%	0.0%	14.3%

Issue #4: Fetch utilization

This instruction group also shows symptoms of: F 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], database_2_vector.hh:37	43.8%	48.3%	43.6%	14.3%	100.0%

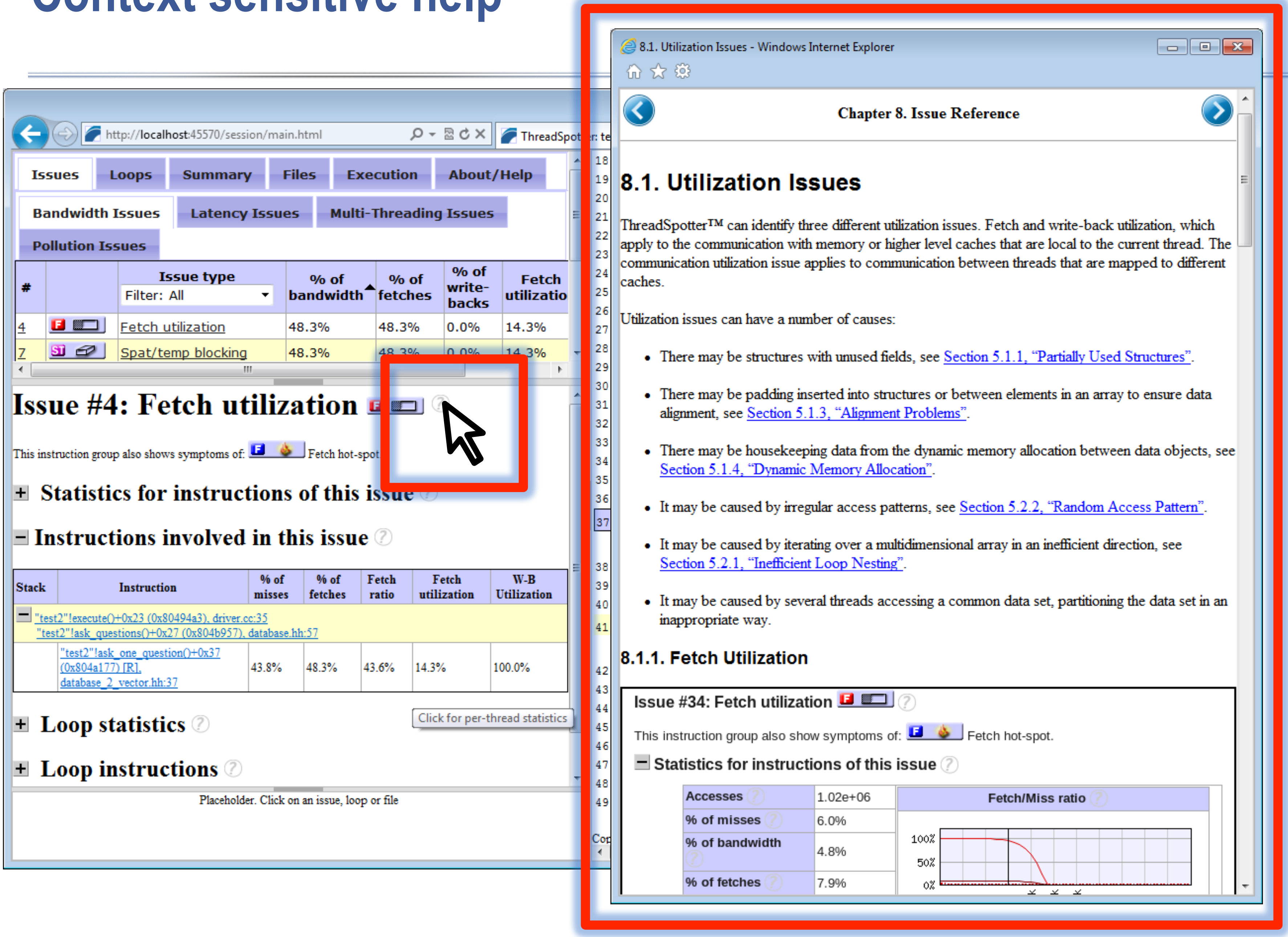
+ Loop statistics

+ Loop instructions

```
        };


        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
+ 48.9%             if (i->color == query.car.color)
        F  ST  Pf NT
                        query.result++;
                    break;
                case 1: // count same model but heavier
+ 51.0%             if (i->model == query.car.model && i->weight > query.
        F  ST  Pf NT  F  ST  Pf NT
                        query.result++;
                    break;
                }
            }
        }

        #endif
```

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

Context sensitive help



#		Issue type Filter: All	% of bandwidth	% of fetches	% of write-backs	Fetch utilizatio
4	F	Fetch utilization	48.3%	48.3%	0.0%	14.3%
7	ST	Spat/temp blocking	48.3%	48.3%	0.0%	14.3%

Issue #4: Fetch utilization

This instruction group also shows symptoms of: F 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], 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 F

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

− Statistics for instructions of this issue

Accesses	1.02e+06
% of misses	6.0%
% of bandwidth	4.8%
% of fetches	7.9%

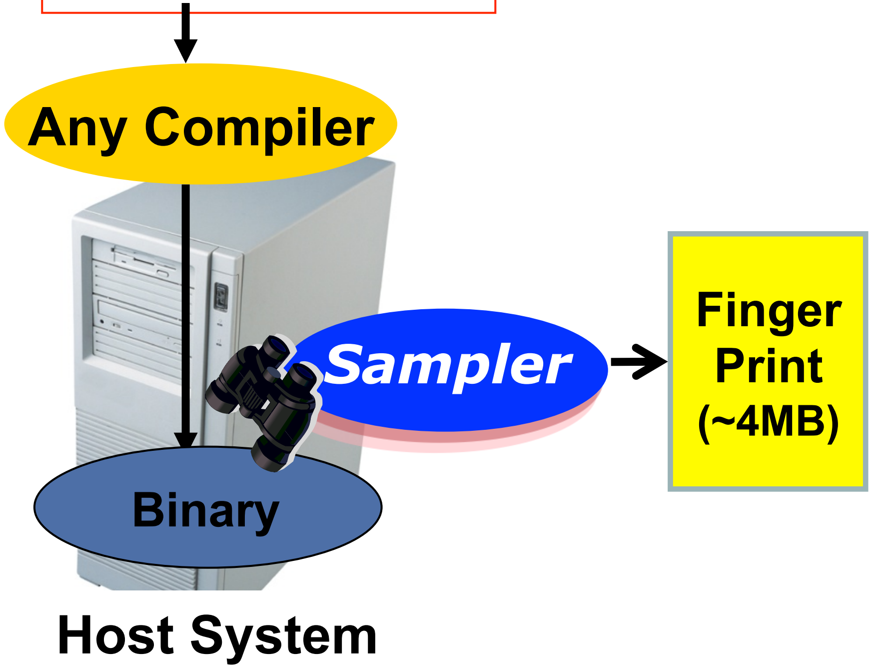
2. Slowspotter Demo

SlowSpotter™

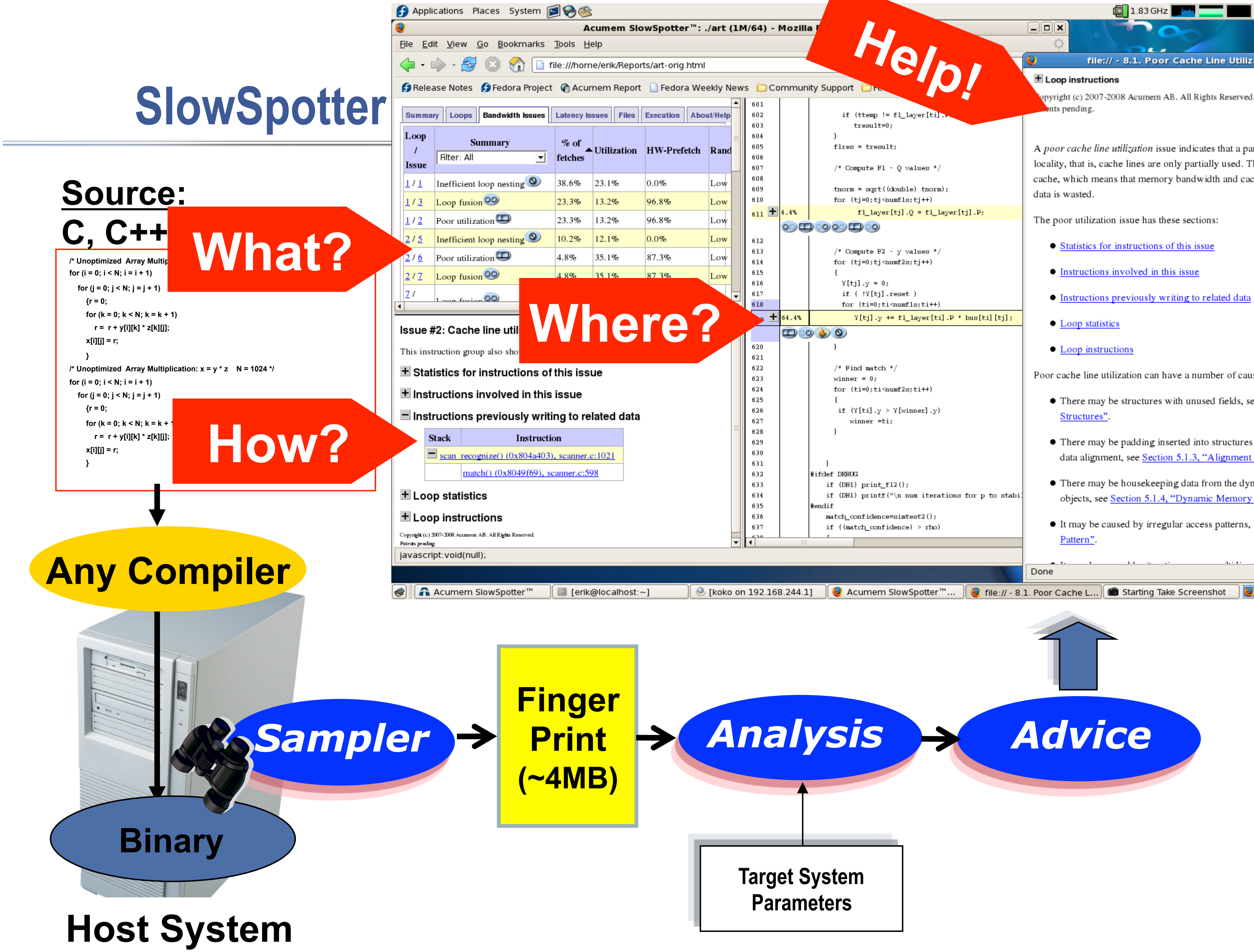
Source:
C, C++, Fortran, OpenMP…

```
/* 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 expe



SlowSpotter



Source:

C, C++

```
/* Unoptimized  Array Multip
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 + 
         r =  r + y[i][k] * z[k][j];
      x[i][j] = r;
      }
```

Loop / Issue	Summary Filter: All	% of fetches	Utilization	HW-Prefetch	Rand
1 / 1	Inefficient loop nesting	38.6%	23.1%	0.0%	Low
1 / 3	Loop fusion	23.3%	13.2%	96.8%	Low
1 / 2	Poor utilization	23.3%	13.2%	96.8%	Low
2 / 5	Inefficient loop nesting	10.2%	12.1%	0.0%	Low
2 / 6	Poor utilization	4.8%	35.1%	87.3%	Low
2 / 7	Loop fusion	4.8%	35.1%	87.3%	Low
7 /	Loop fusion				

Issue #2: Cache line util

This instruction group also sho

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

Stack	Instruction
scan_recognize() (0x804a403), scanner.c:1021	
	match() (0x8049f69), scanner.c:598

- Loop statistics
- Loop instructions

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

```
            if (ttemp != f1_layer[ti].
               tresult=0;
         }
         flres = tresult;

         /* Compute F1 - Q values */

         tnorm = sqrt((double) tnorm);
         for (tj=0;tj<numf1s;tj++)
4.4%           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++)
64.4%            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) print_f12();
      if (DB1) printf("\n num iterations for p to stabi
#endif
      match_confidence=simtest2();
      if ((match_confidence) > rho)
```

Loop instructions

opyright (c) 2007-2008 Acumem AB. All Rights Reserved.
ents pending.

A *poor cache line utilization* issue indicates that a pa locality, that is, cache lines are only partially used. T cache, which means that memory bandwidth and cac data is wasted.

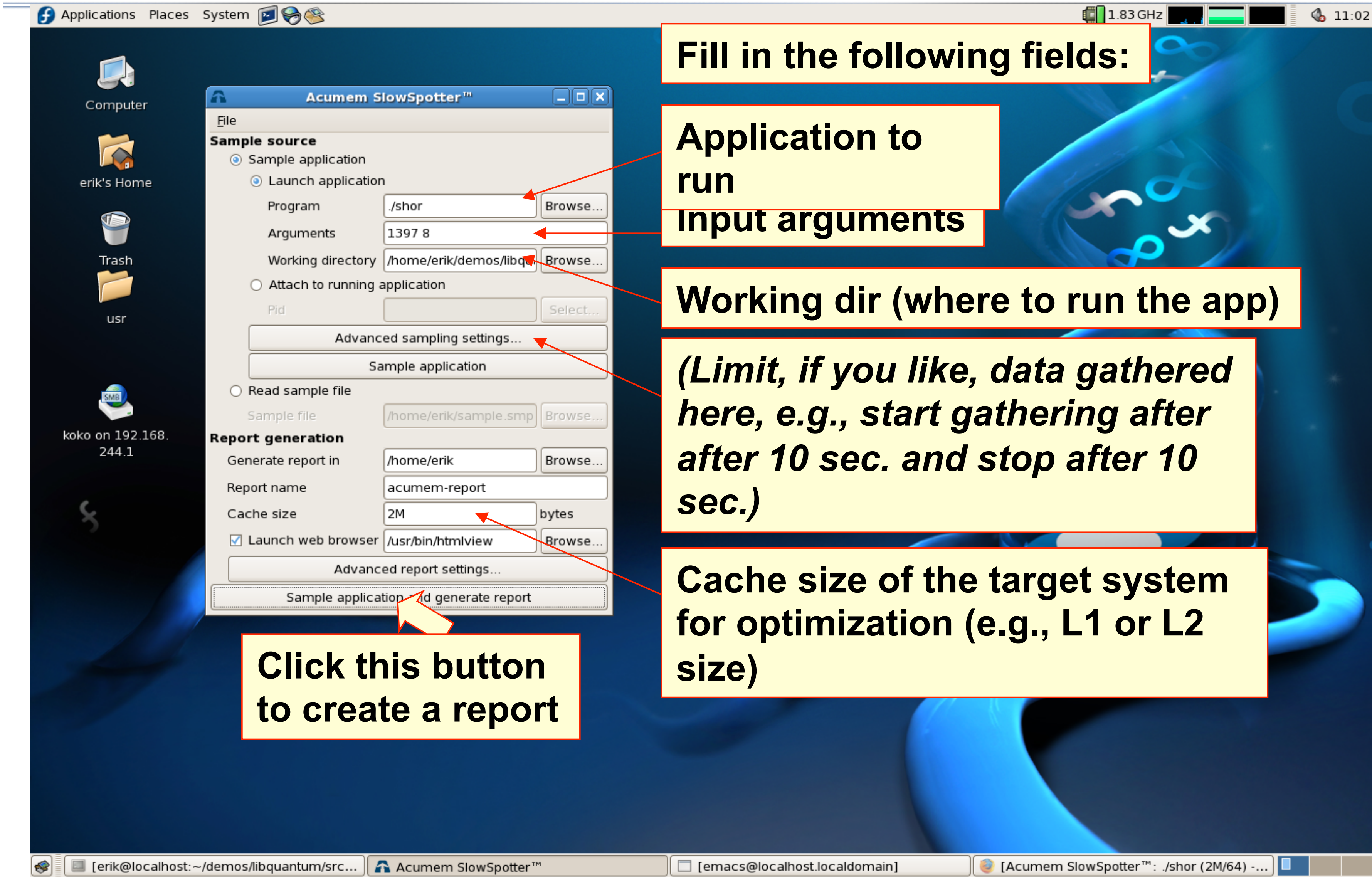
The poor utilization issue has these sections:

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

Poor cache line utilization can have a number of cau

- There may be structures with unused fields, se Structures".
- There may be padding inserted into structures data alignment, see Section 5.1.3, "Alignment
- There may be housekeeping data from the dyn objects, see Section 5.1.4, "Dynamic Memory
- It may be caused by irregular access patterns, Pattern".

A One-Click Report Generation



Sample source

● Sample application

● Launch application

Program: ./shor

Arguments: 1397 8

Working directory: /home/erik/demos/libq

○ Attach to running application

Pid:

○ Read sample file

Sample file: /home/erik/sample.smp

Report generation

Generate report in: /home/erik

Report name: acumem-report

Cache size: 2M bytes

☑ Launch web browser: /usr/bin/htmlview

Fill in the following fields:

Application to run

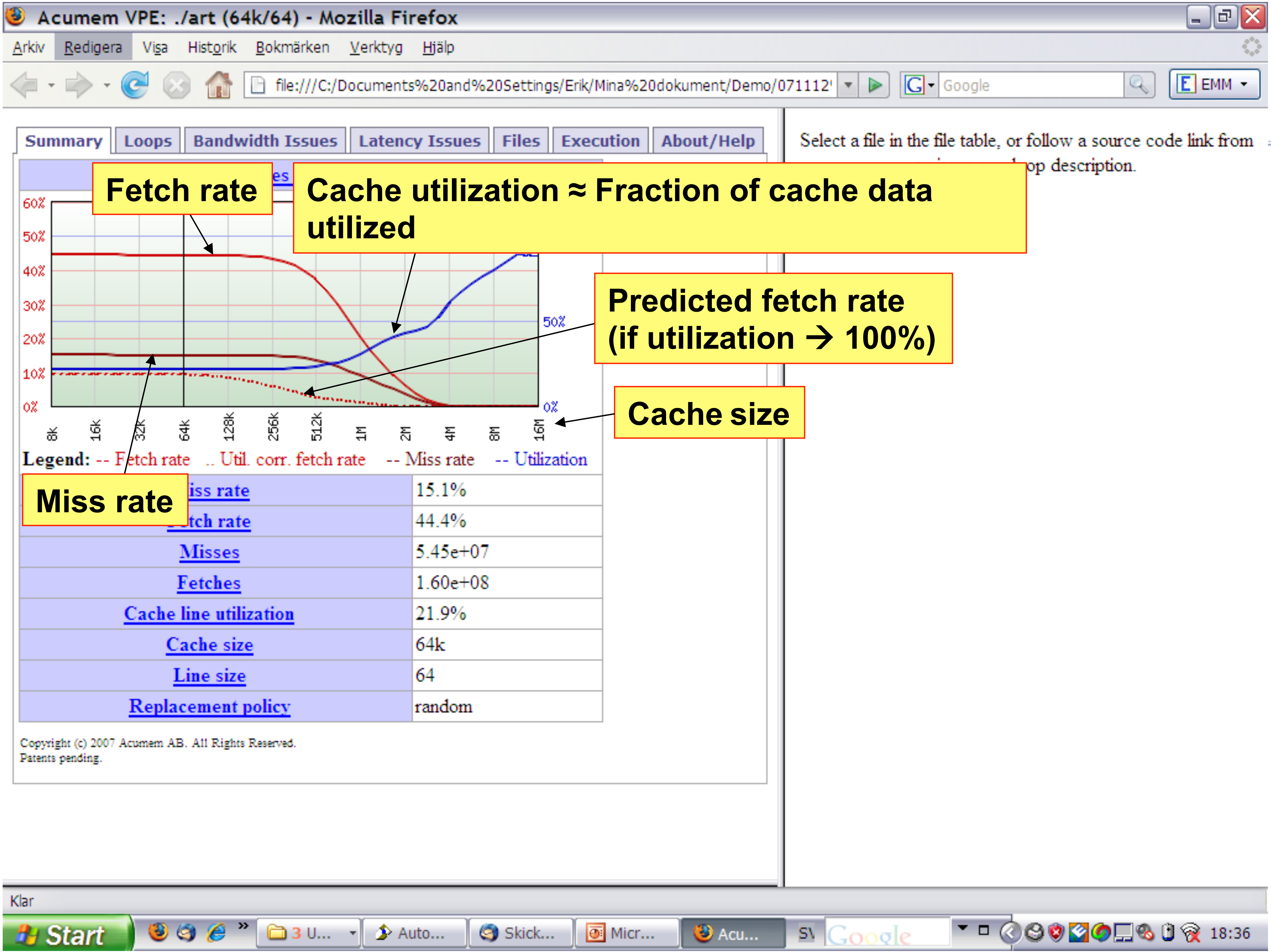
Input arguments

Working dir (where to run the app)

(Limit, if you like, data gathered here, e.g., start gathering after after 10 sec. and stop after 10 sec.)

Cache size of the target system for optimization (e.g., L1 or L2 size)

Click this button to create a report



Fetch rate

Cache utilization ≈ Fraction of cache data utilized

Predicted fetch rate (if utilization → 100%)

Cache size

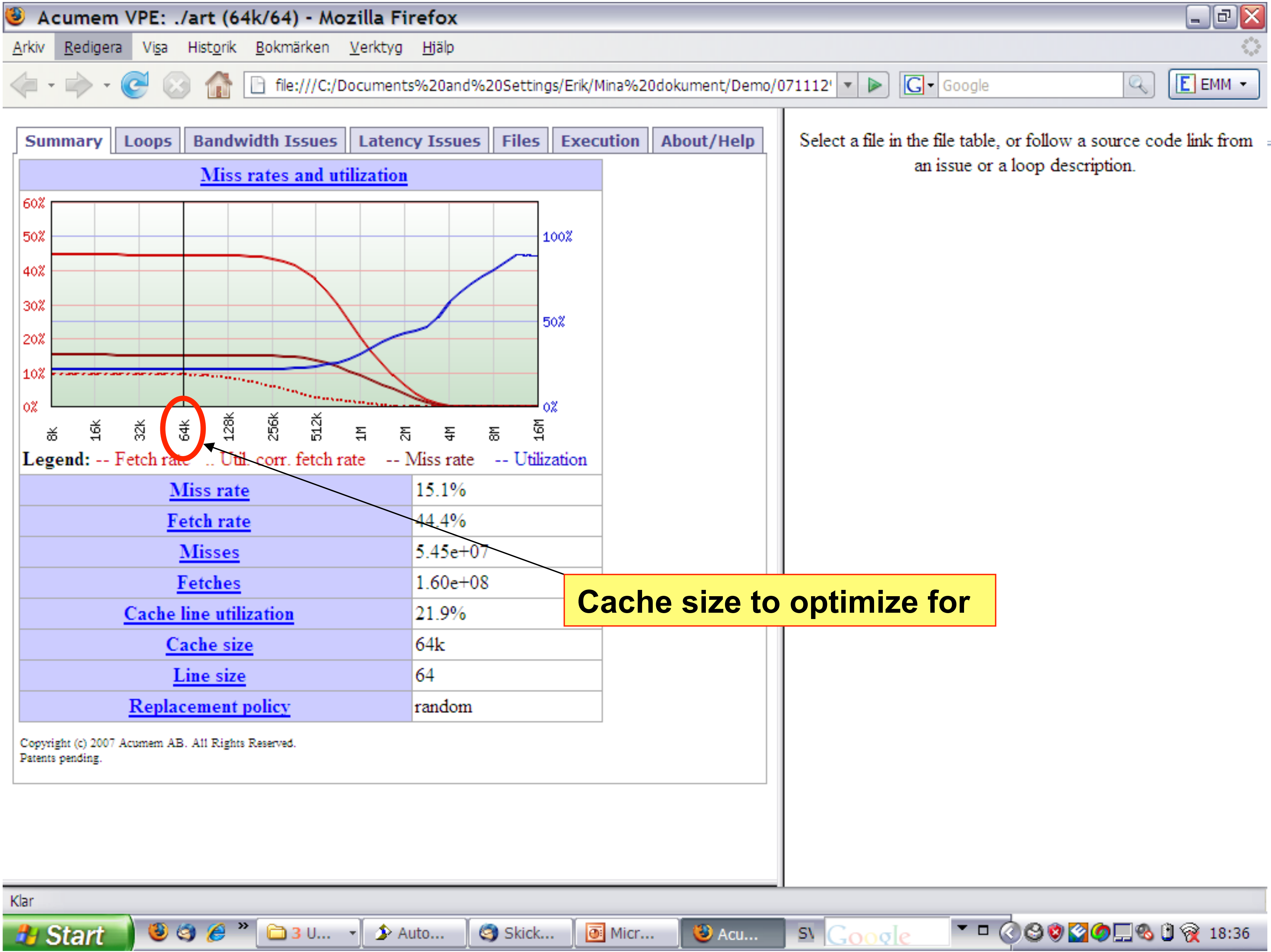
Miss rate

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

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

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

op description.



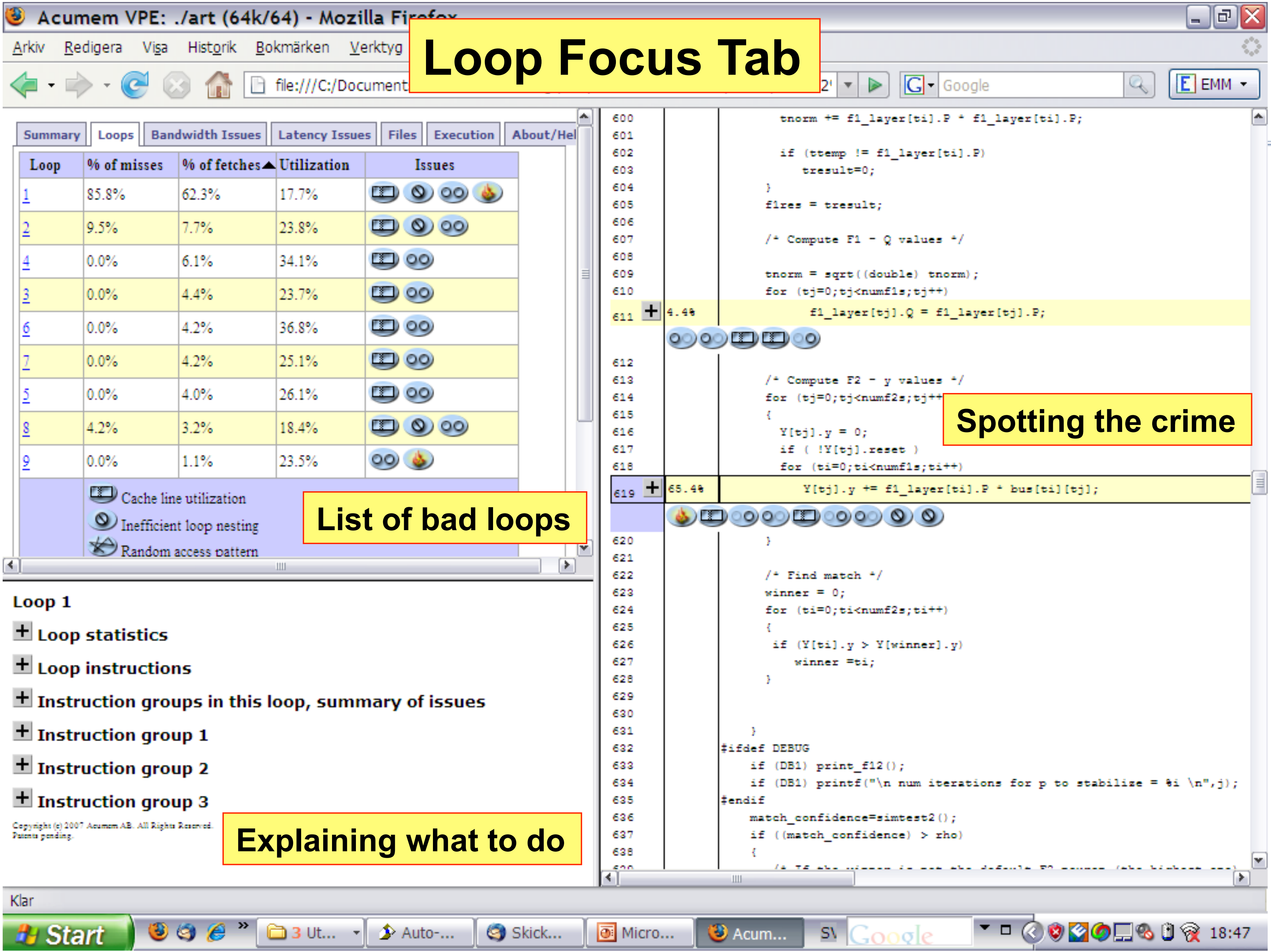
Select a file in the file table, or follow a source code link from an issue or a loop description.

Miss rates and utilization

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

Cache size to optimize for

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%	○○ 🔥

▭ Cache line utilization
⊘ Inefficient loop nesting
✕ Random access pattern

List of bad loops

Loop 1

+ Loop statistics

+ Loop instructions

+ Instruction groups in this loop, summary of issues

+ Instruction group 1

+ Instruction group 2

+ Instruction group 3

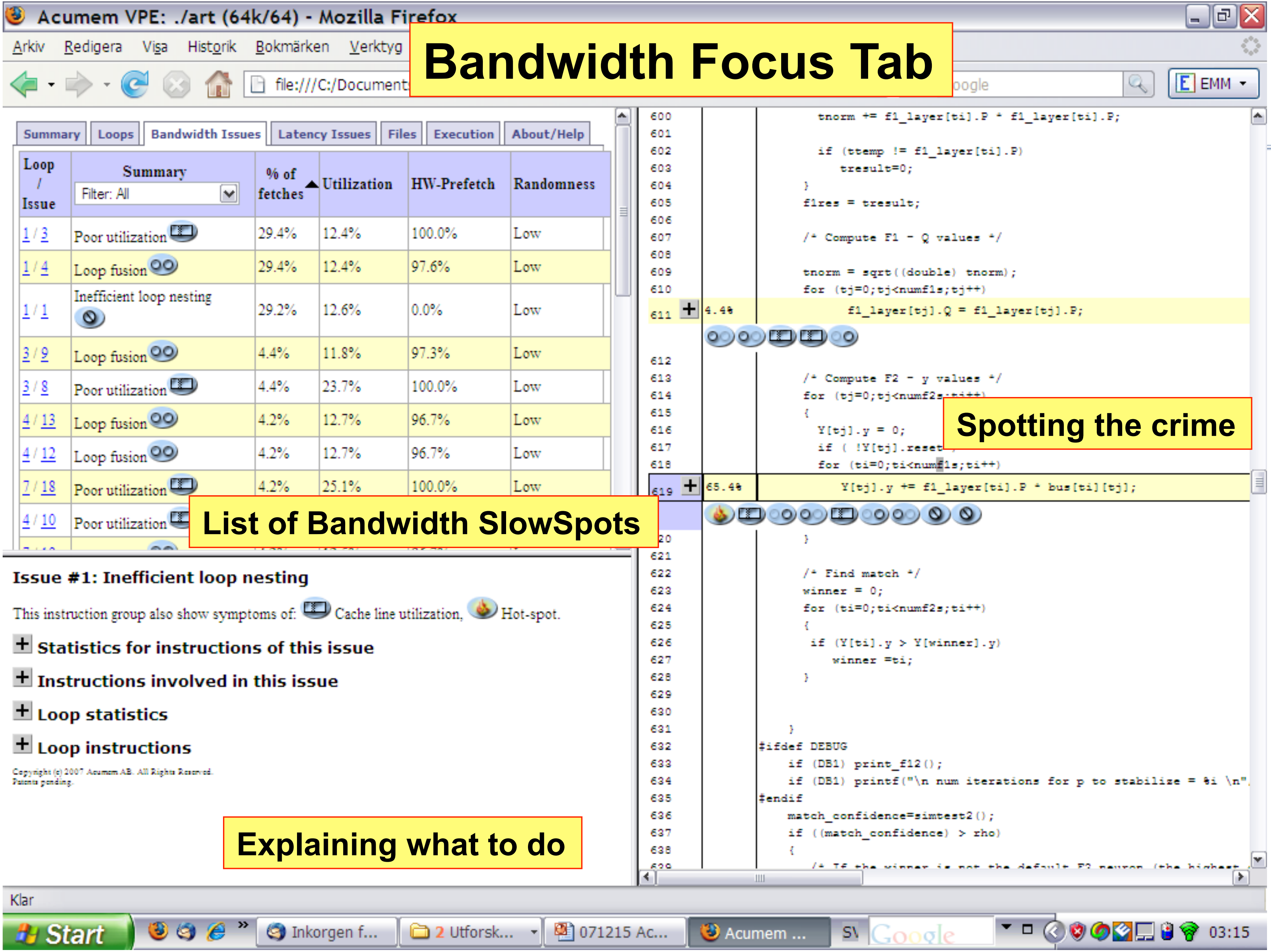
Copyright (c) 2007 Acumem AB. All Rights Reserved.
Patents pending.

Explaining what to do

```
                 tnorm += f1_layer[ti].P * f1_layer[ti].P;

                 if (ttemp != f1_layer[ti].P)
                    tresult=0;
               }
               f1res = tresult;

               /* Compute F1 - Q values */

               tnorm = sqrt((double) tnorm);
               for (tj=0;tj<numf1s;tj++)
+ 4.4%              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++)
+ 65.4%            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) print_f12();
         if (DB1) printf("\n num iterations for p to stabilize = %i \n",j);
#endif
         match_confidence=simtest2();
         if ((match_confidence) > rho)
         {
```

Spotting the crime

Bandwidth Focus Tab



Loop / Issue	Summary Filter: All	% of fetches	Utilization	HW-Prefetch	Randomness
1 / 3	Poor utilization	29.4%	12.4%	100.0%	Low
1 / 4	Loop fusion	29.4%	12.4%	97.6%	Low
1 / 1	Inefficient loop nesting	29.2%	12.6%	0.0%	Low
3 / 9	Loop fusion	4.4%	11.8%	97.3%	Low
3 / 8	Poor utilization	4.4%	23.7%	100.0%	Low
4 / 13	Loop fusion	4.2%	12.7%	96.7%	Low
4 / 12	Loop fusion	4.2%	12.7%	96.7%	Low
7 / 18	Poor utilization	4.2%	25.1%	100.0%	Low
4 / 10	Poor utilization				

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.

```
                  tnorm += f1_layer[ti].P * f1_layer[ti].P;

                  if (ttemp != f1_layer[ti].P)
                     tresult=0;
                }
                f1res = tresult;

                /* Compute F1 - Q values */

                tnorm = sqrt((double) tnorm);
                for (tj=0;tj<numf1s;tj++)
+ 4.4%                 f1_layer[tj].Q = f1_layer[tj].P;


                /* Compute F2 - y values */
                for (tj=0;tj<numf2s
                {
                  Y[tj].y = 0;
                  if ( !Y[tj].reset
                  for (ti=0;ti<numf1s;ti++)
+ 65.4%                  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) print_f12();
        if (DB1) printf("\n num iterations for p to stabilize = %i \n",
#endif
        match_confidence=simtest2();
        if ((match_confidence) > rho)
        {
           /* If the winner is not the default F2 neuron (the highest
```

Spotting the crime

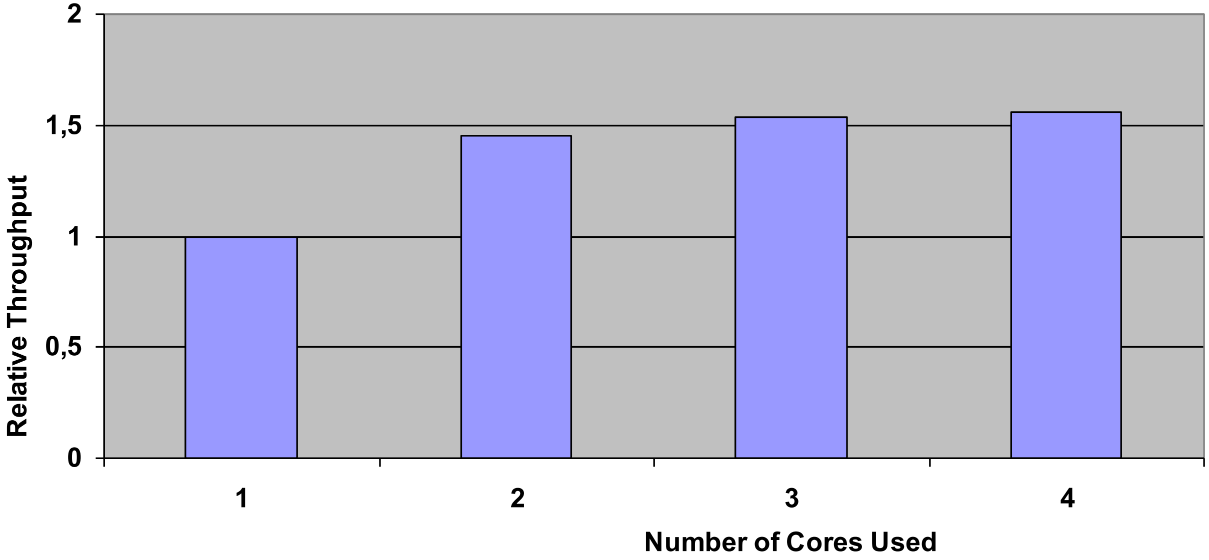
List of Bandwidth SlowSpots

Explaining what to do

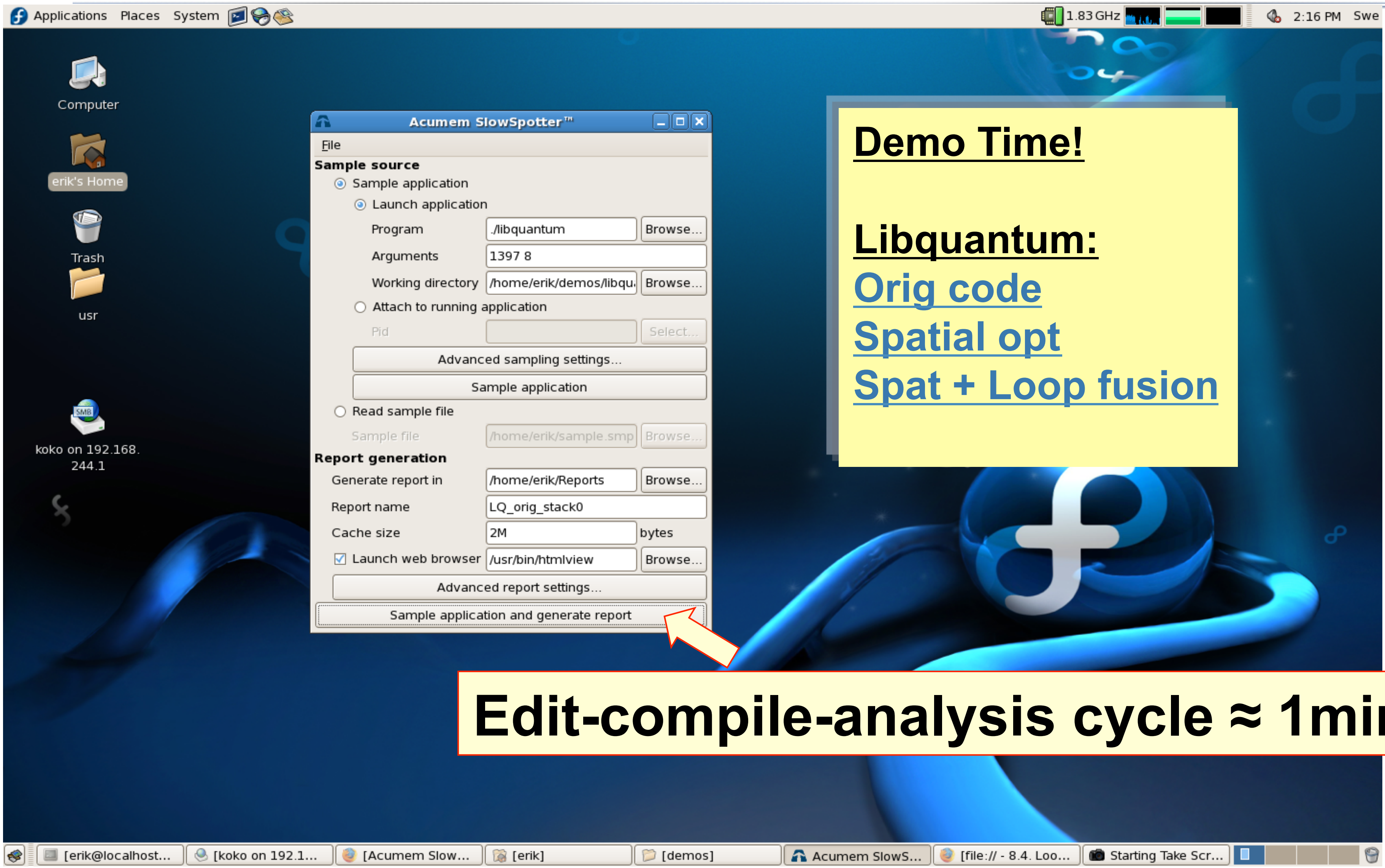
Resource Sharing Example

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



Sample source

● Sample application

● Launch application

Program: ./libquantum

Arguments: 1397 8

Working directory: /home/erik/demos/libqu

○ Attach to running application

Pid:

○ Read sample file

Sample file: /home/erik/sample.smp

Report generation

Generate report in: /home/erik/Reports

Report name: LQ_orig_stack0

Cache size: 2M bytes

☑ Launch web browser: /usr/bin/htmlview

Demo Time!

Libquantum:

Orig code

Spatial opt

Spat + Loop fusion

Edit-compile-analysis cycle ≈ 1min

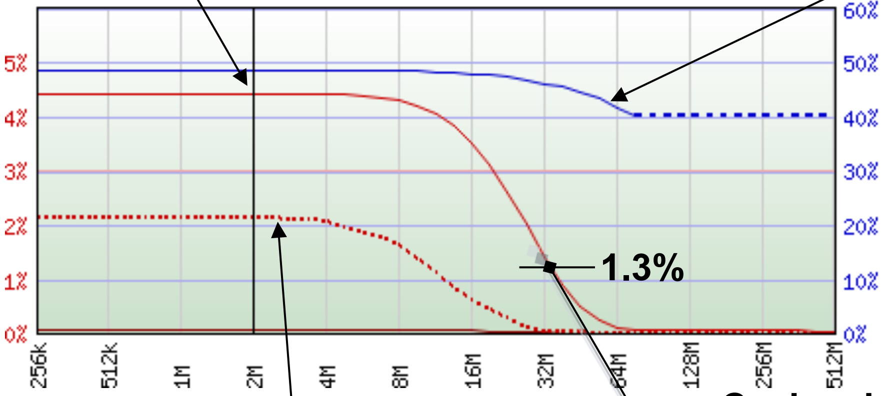
Utilization Analysis

Libquantum

Fetch rate

Original Code

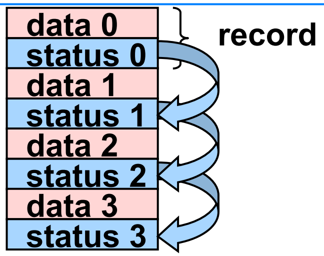
Cache utilization ≈ Fraction of cache data utilized



Cache size

Predicted fetch rate
if utilization = 100%

Need 32 MB per thread!



Only accessing status data in main loop

SlowSpotter’s First Advice: Improve Utilization

➔Change one data structure

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

Utilization Analysis

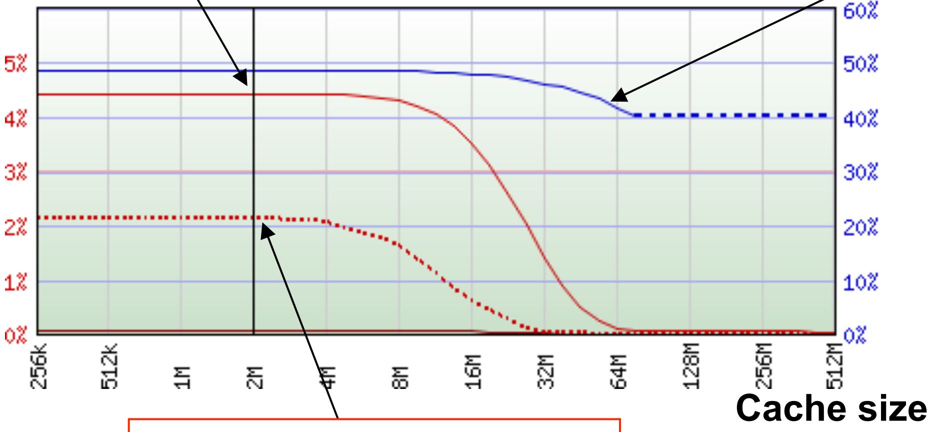
Libquantum

Cache utilization ≈ Fraction of cache data utilized

Fetch rate

Original Code

Utilization Optimization



Predicted fetch rate
if utilization = 100%

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

⬇

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

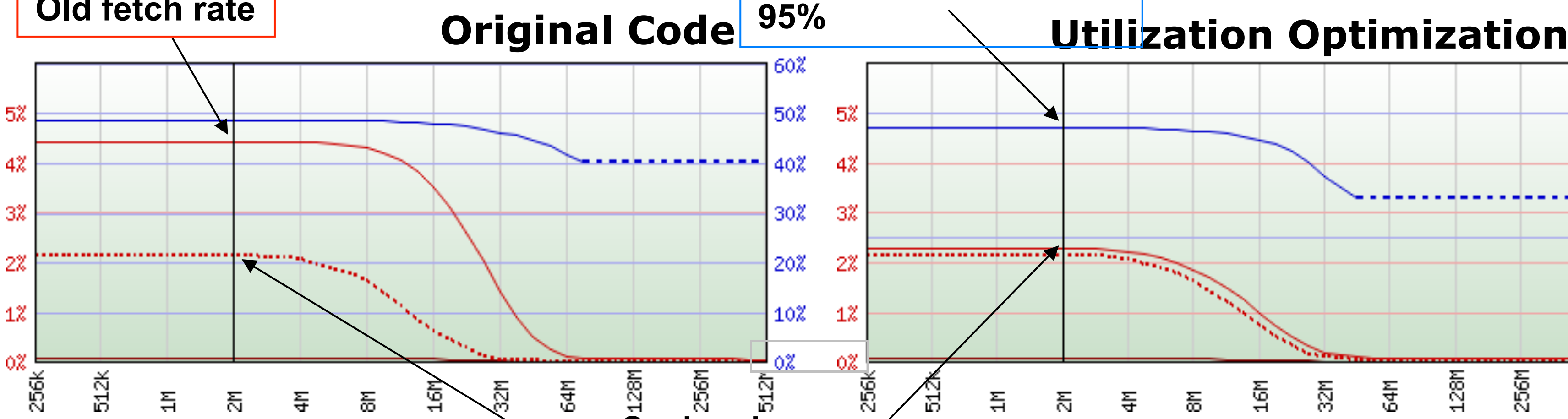
SlowSpotter’s First Advice: Improve Utilization

➔Change one data structure

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

After Utilization Optimization

Libquantum



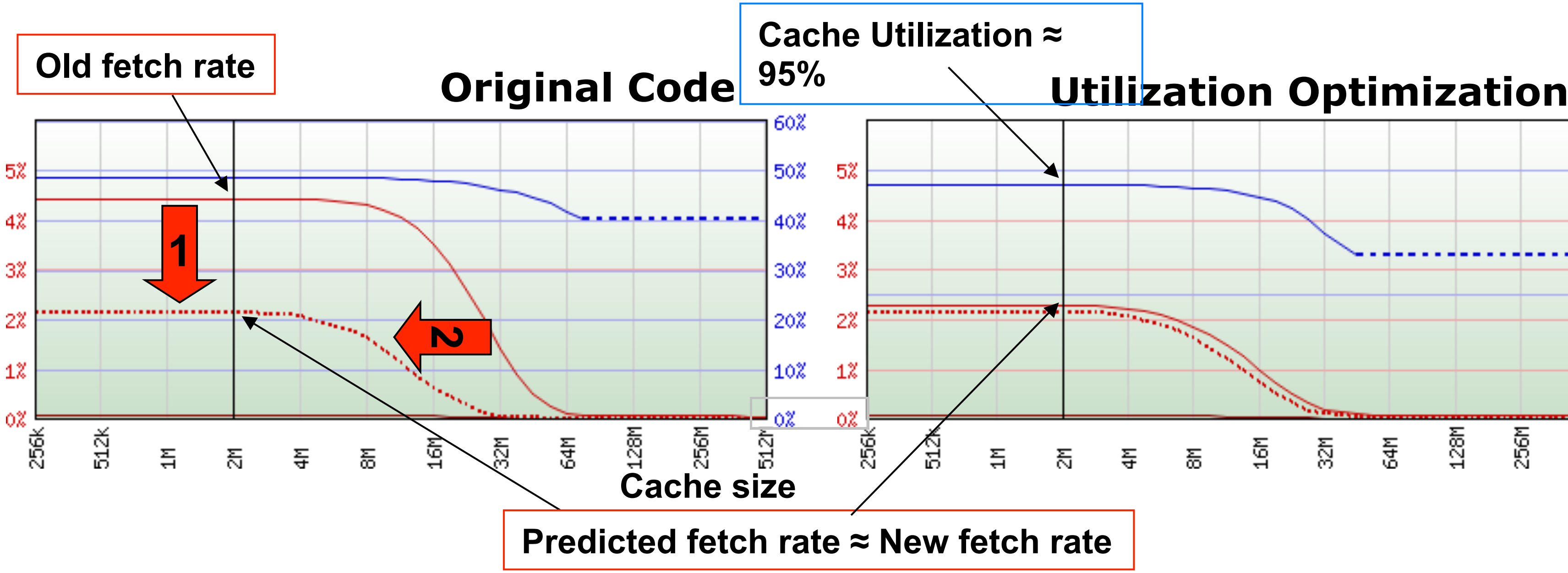
Old fetch rate

Cache Utilization ≈ 95%

Cache size

Predicted fetch rate ≈ New fetch rate

Utilization Optimization



Old fetch rate

Cache Utilization ≈ 95%

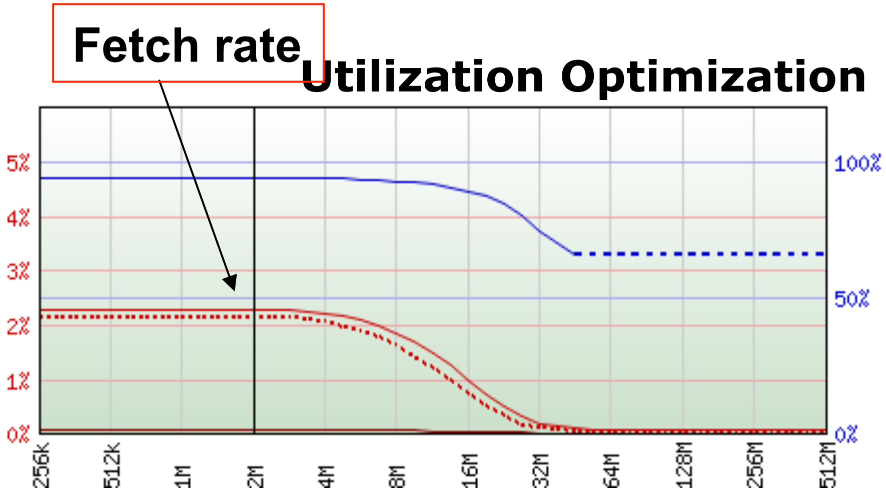
Predicted fetch rate ≈ New fetch rate

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 Optimization

Utilization + Fusion Optimizatio

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

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

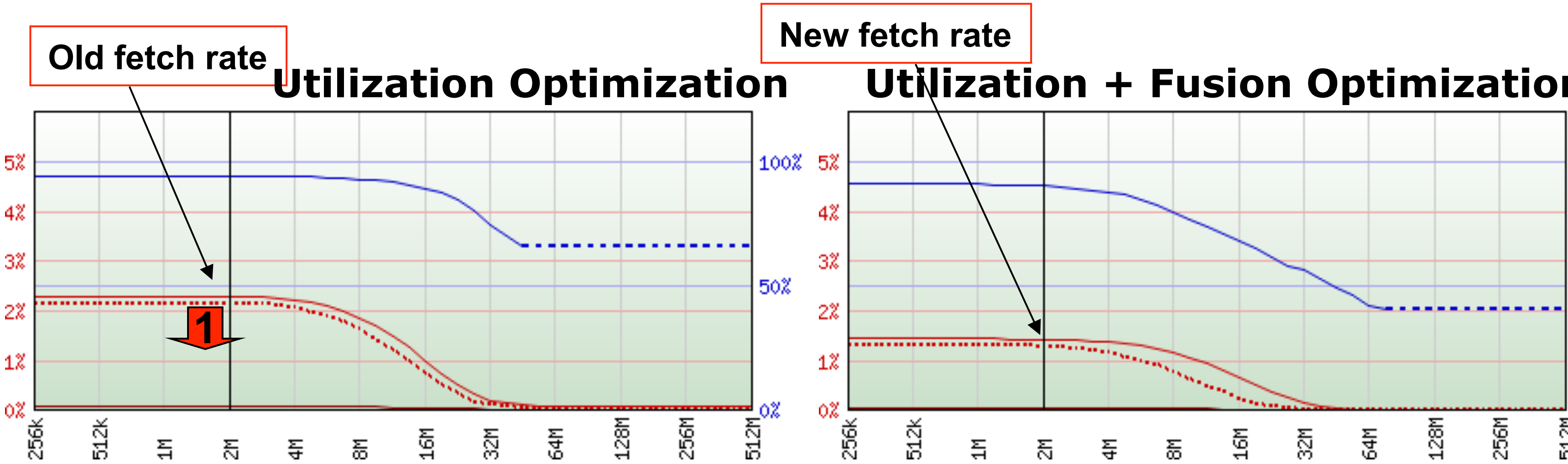
Second-Fifth SlowSpotter Advice: Improve reuse of data

➔Fuse functions traversing the same data

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

Effect: Reuse Optimization

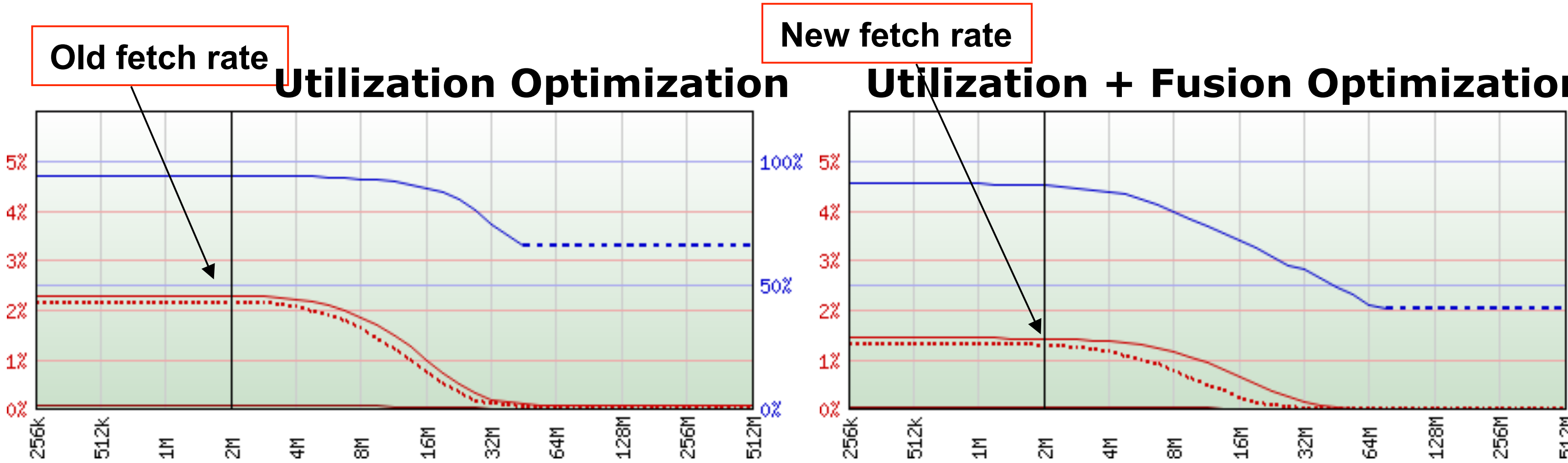
SPEC CPU2006-462.libquantum



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

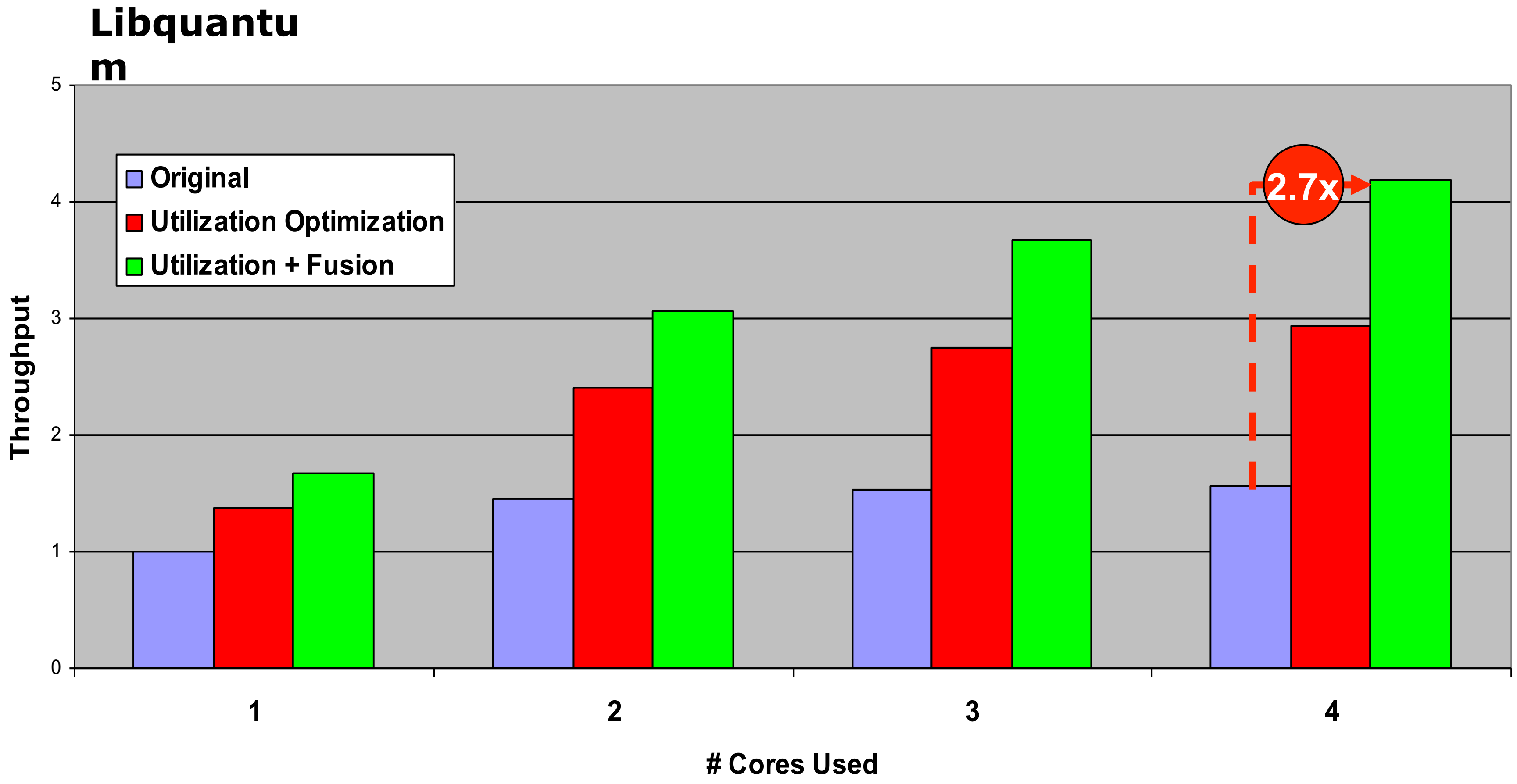
Utilization + Reuse Optimization

Libquantum

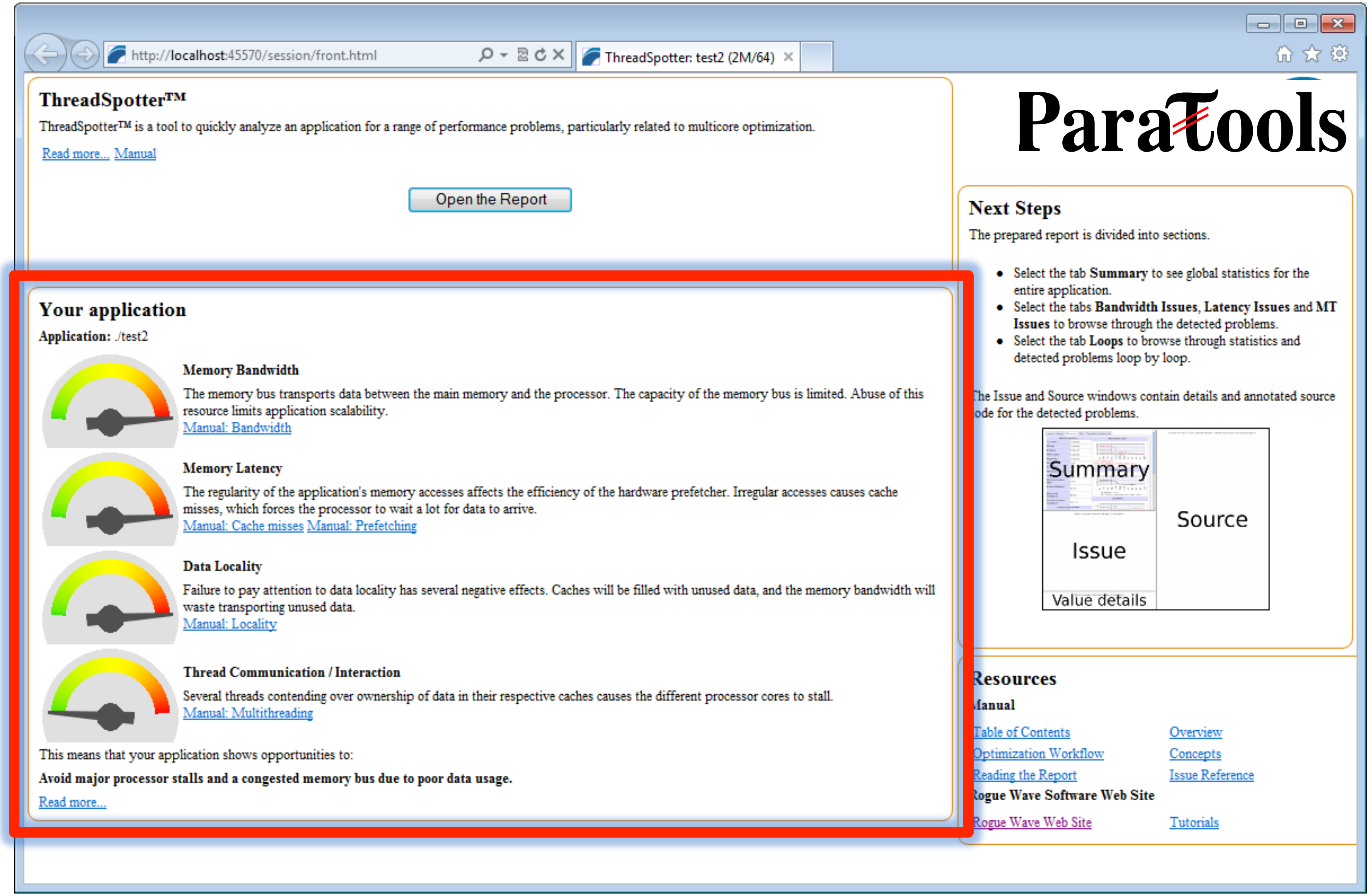


- Fetch rate down to 1.3% for 2MB
- Same as a 32 MB cache originally

Summary


Libquantum
5
4
3
2
1
0
Throughput
Original
Utilization Optimization
Utilization + Fusion
2.7x
1
2
3
4
Cores Used

Report – front page



ThreadSpotter™

ThreadSpotter™ is a tool to quickly analyze an application for a range of performance problems, particularly related to multicore optimization.

Read more... Manual

Open the Report

Your application

Application: ./test2

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:

Avoid major processor stalls and a congested memory bus due to poor data usage.

Read more...

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 ode for the detected problems.

Resources

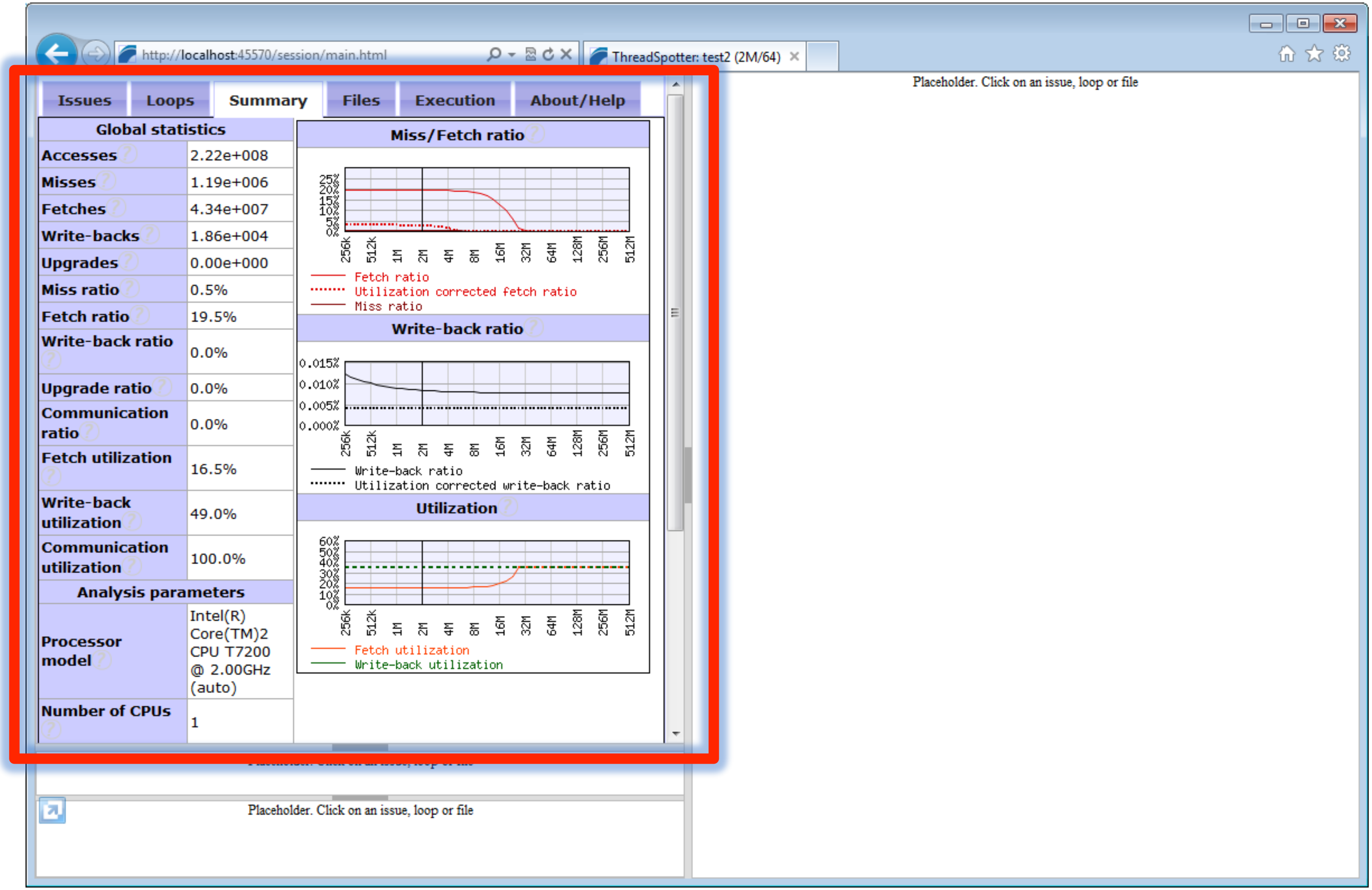
Manual

Table of Contents	Overview
Optimization Workflow	Concepts
Reading the Report	Issue Reference

Rogue Wave Software Web Site

Rogue Wave Web Site	Tutorials

Summary, global statistics

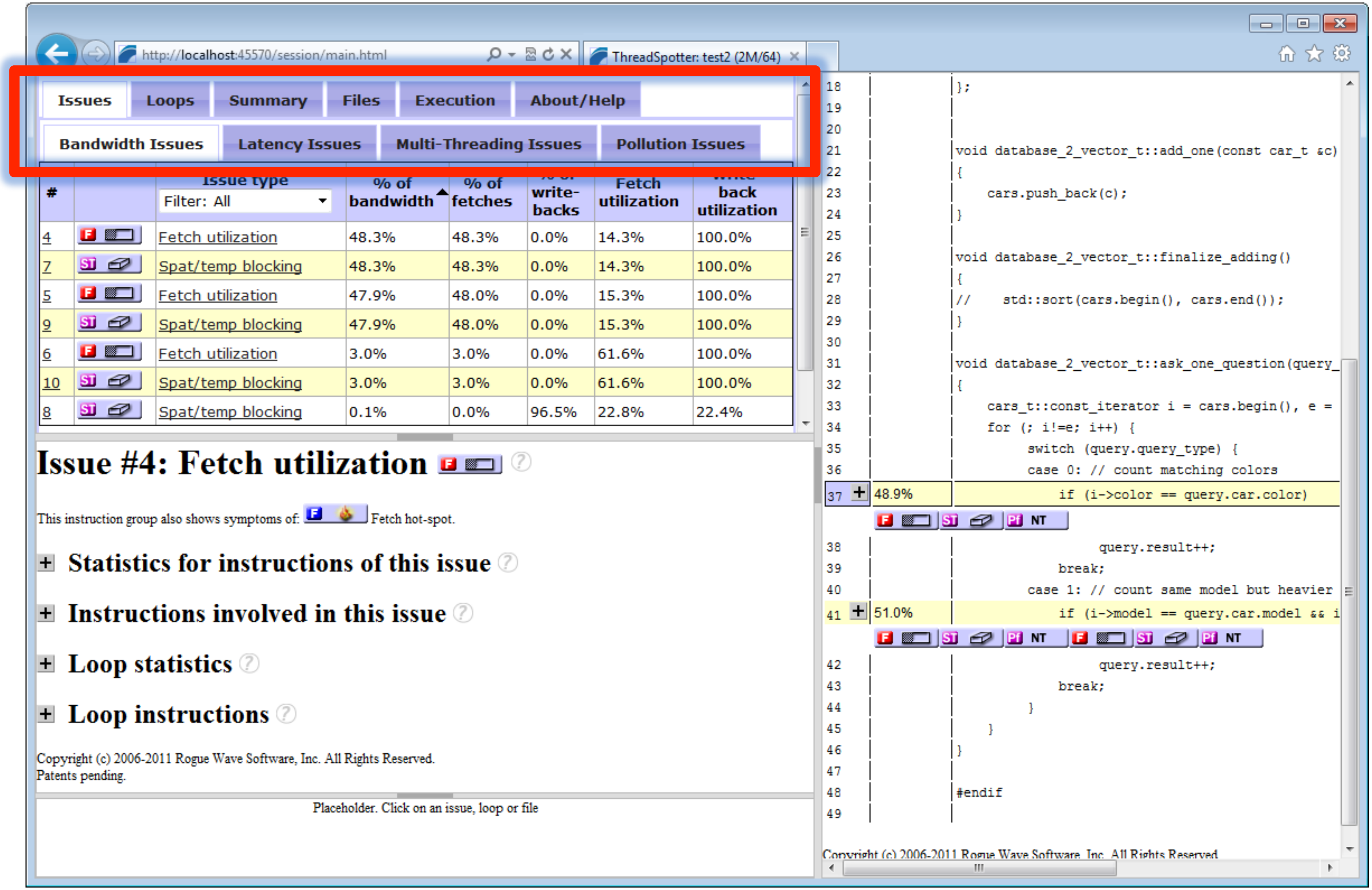


Global statistics	
Accesses	2.22e+008
Misses	1.19e+006
Fetches	4.34e+007
Write-backs	1.86e+004
Upgrades	0.00e+000
Miss ratio	0.5%
Fetch ratio	19.5%
Write-back ratio	0.0%
Upgrade ratio	0.0%
Communication ratio	0.0%
Fetch utilization	16.5%
Write-back utilization	49.0%
Communication utilization	100.0%
Analysis parameters	
Processor model	Intel(R) Core(TM)2 CPU T7200 @ 2.00GHz (auto)
Number of CPUs	1

Placeholder. Click on an issue, loop or file

Placeholder. Click on an issue, loop or file

Report sections



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

Issue #4: Fetch utilization

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

+ Statistics for instructions of this issue

+ Instructions involved in this issue

+ Loop statistics

+ Loop instructions

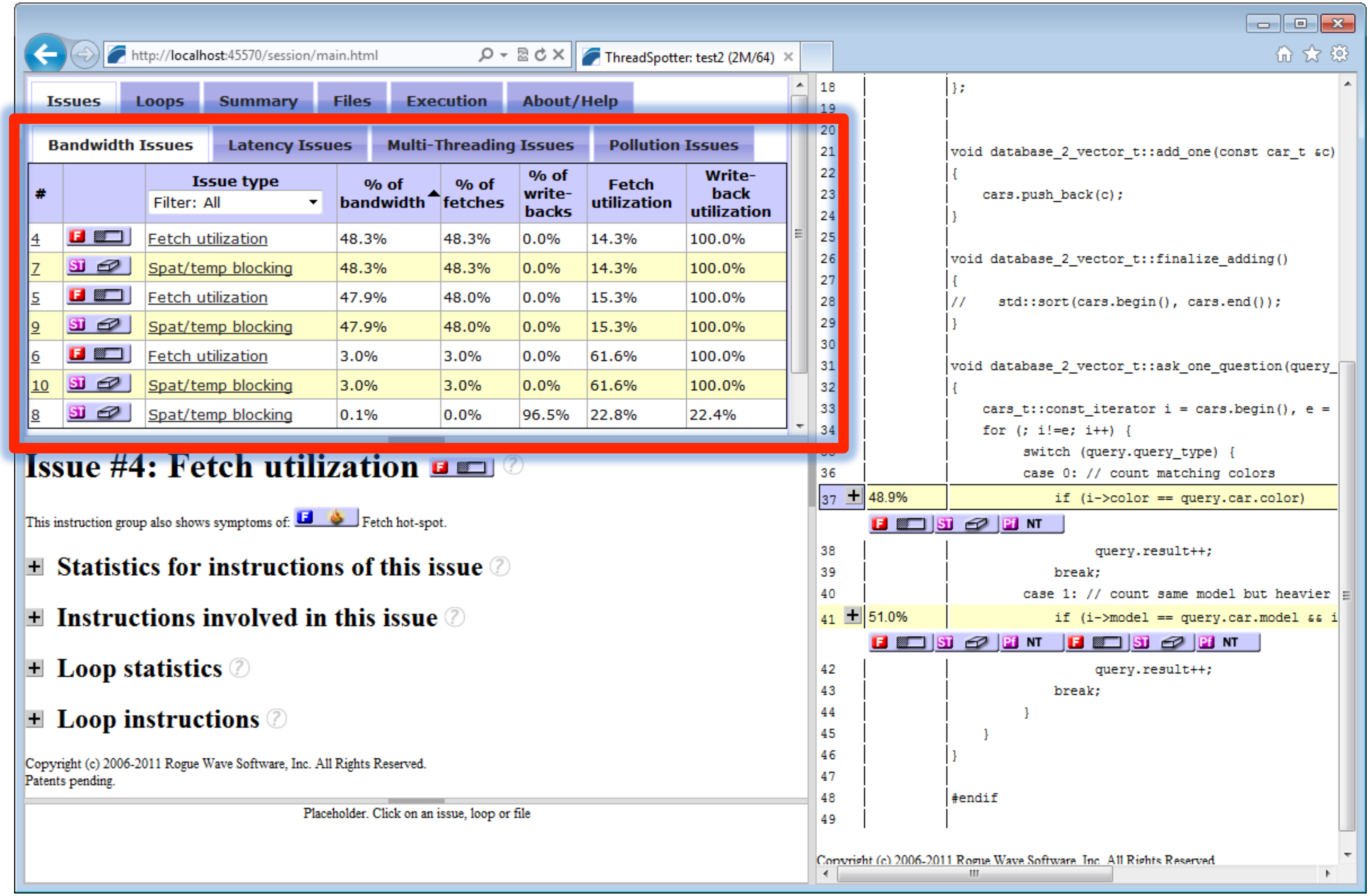
Copyright (c) 2006-2011 Rogue Wave Software, Inc. All Rights Reserved.
Patents pending.

Placeholder. Click on an issue, loop or file

```
};


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_
{
    cars_t::const_iterator i = cars.begin(), e =
    for (; i!=e; i++) {
        switch (query.query_type) {
        case 0: // count matching colors
+ 48.9%        if (i->color == query.car.color)
                F  ST  Pf NT
                query.result++;
                break;
        case 1: // count same model but heavier
+ 51.0%        if (i->model == query.car.model && i
                F  ST  Pf NT  F  ST  Pf NT
                query.result++;
                break;
        }
    }
}

#endif

```

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

Navigation by issues



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

Issue #4: Fetch utilization

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

\+ Statistics for instructions of this issue

\+ Instructions involved in this issue

\+ Loop statistics

\+ Loop instructions

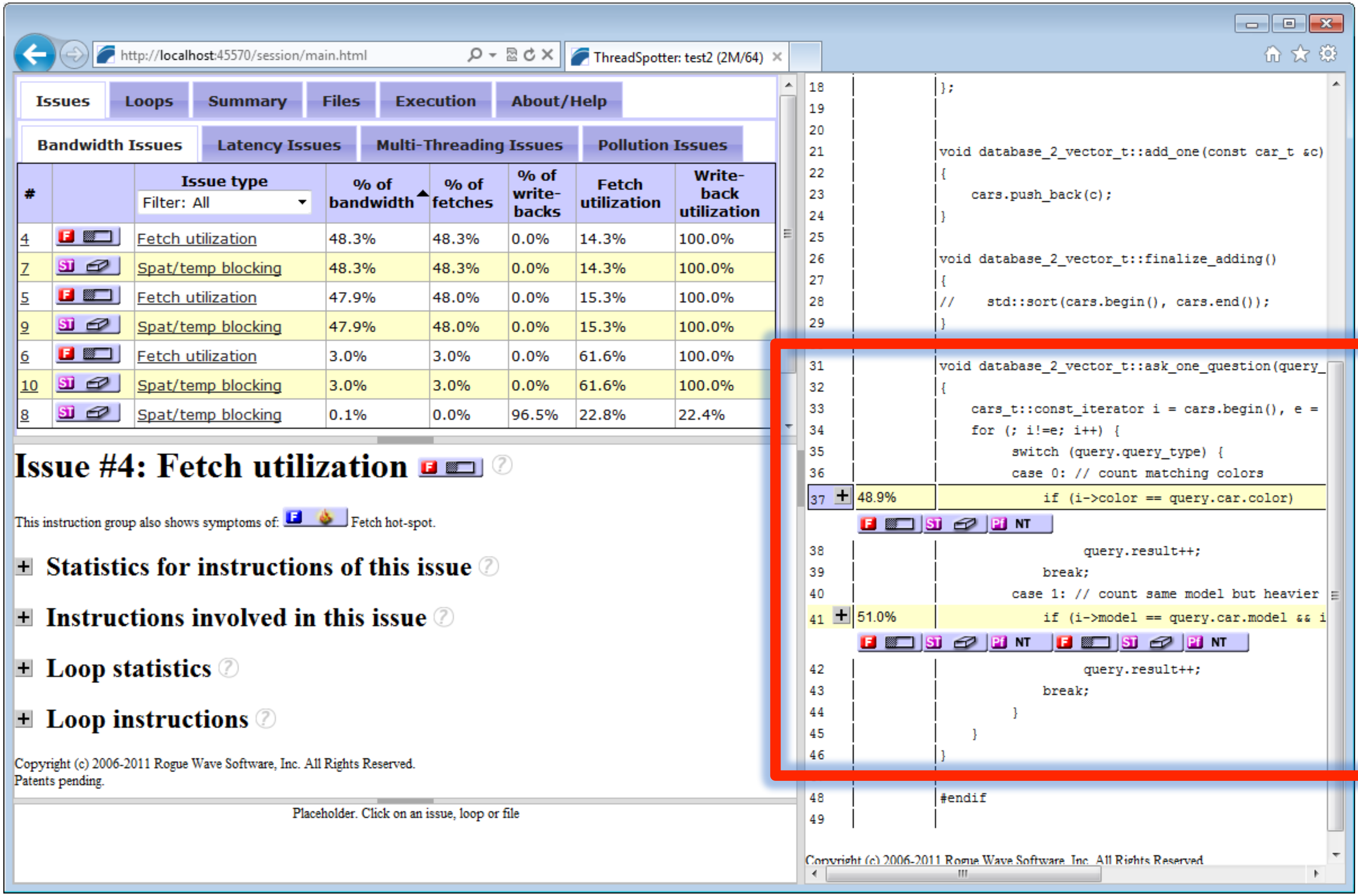
Copyright (c) 2006-2011 Rogue Wave Software, Inc. All Rights Reserved.
Patents pending.

Placeholder. Click on an issue, loop or file

```
        };


        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_
        {
            cars_t::const_iterator i = cars.begin(), e =
            for (; i!=e; i++) {
                switch (query.query_type) {
                case 0: // count matching colors
+ 48.9%                 if (i->color == query.car.color)
        F ST Pf NT
                        query.result++;
                    break;
                case 1: // count same model but heavier
+ 51.0%                 if (i->model == query.car.model && i
        F ST Pf NT F ST Pf NT
                        query.result++;
                    break;
                }
            }
        }

        #endif
```

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

Source code annotation



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

Issue #4: Fetch utilization F

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

+ Statistics for instructions of this issue

+ Instructions involved in this issue

+ Loop statistics

+ Loop instructions

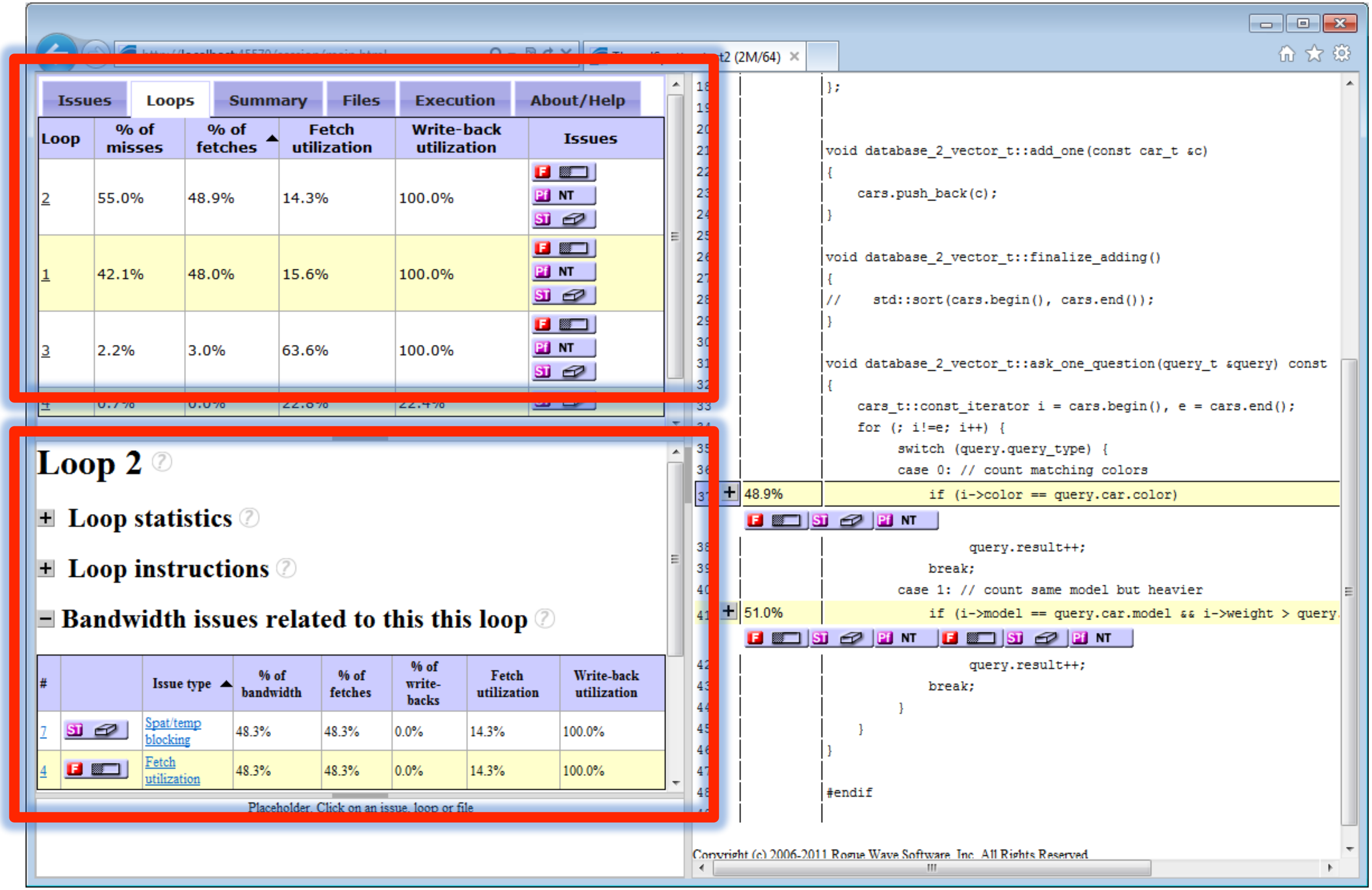
Copyright (c) 2006-2011 Rogue Wave Software, Inc. All Rights Reserved.
Patents pending.

Placeholder. Click on an issue, loop or file

```
            };


            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_
            {
                cars_t::const_iterator i = cars.begin(), e =
                for (; i!=e; i++) {
                    switch (query.query_type) {
                    case 0: // count matching colors
+  48.9%                    if (i->color == query.car.color)
         F  ST  Pf NT
                            query.result++;
                        break;
                    case 1: // count same model but heavier
+  51.0%                    if (i->model == query.car.model && i
         F  ST  Pf NT  F  ST  Pf NT
                            query.result++;
                        break;
                    }
                }
            }

            #endif
```

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

Navigation by loops



Loop	% of misses	% of fetches ▲	Fetch utilization	Write-back utilization	Issues
2	55.0%	48.9%	14.3%	100.0%	F Pf NT ST
1	42.1%	48.0%	15.6%	100.0%	F Pf NT ST
3	2.2%	3.0%	63.6%	100.0%	F Pf NT ST
4	0.7%	0.0%	22.8%	22.4%	

Loop 2

+ Loop statistics

+ Loop instructions

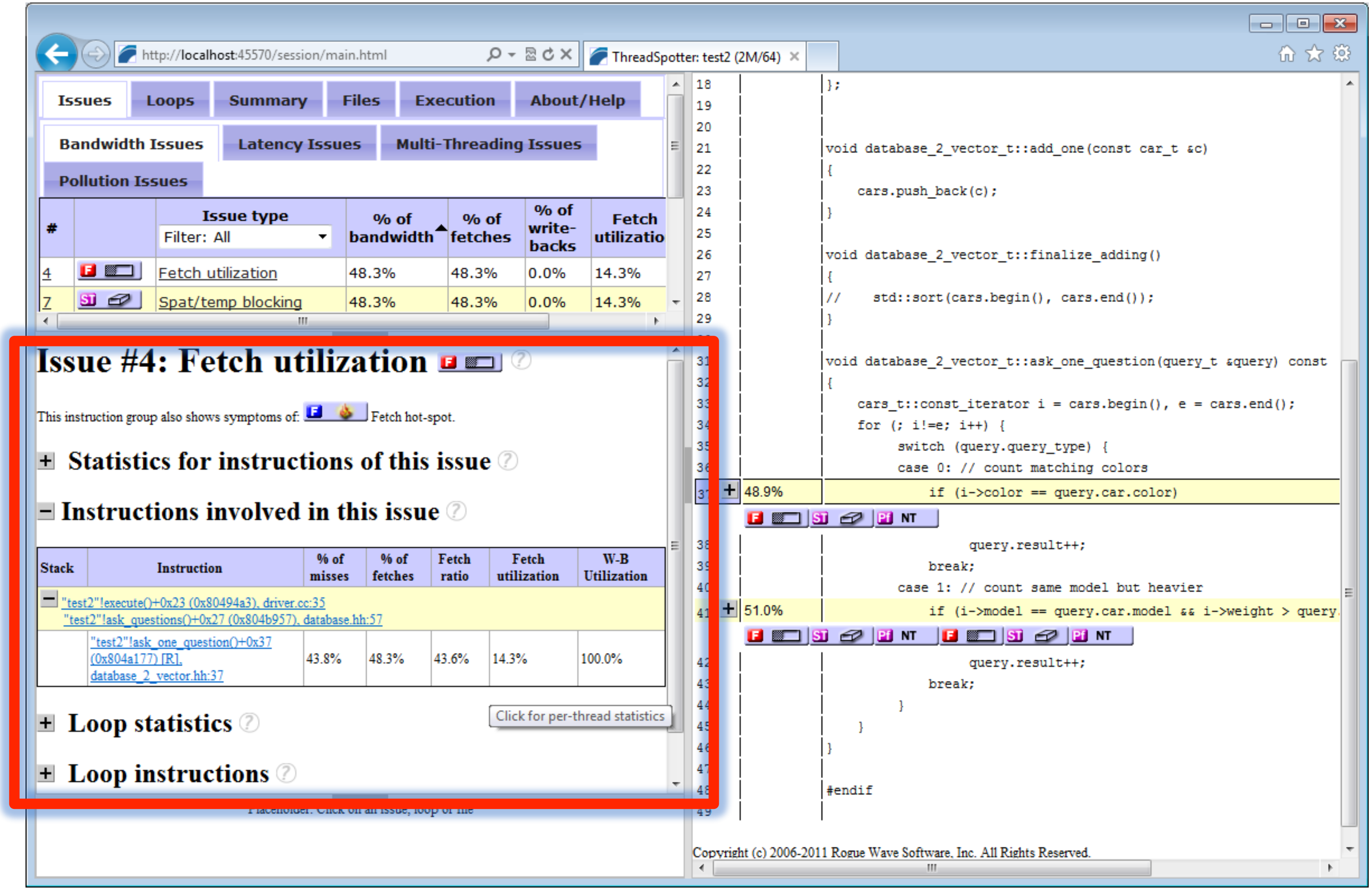
− Bandwidth issues related to this this loop

#		Issue type ▲	% of bandwidth	% of fetches	% of write-backs	Fetch utilization	Write-back utilization
7	ST	Spat/temp blocking	48.3%	48.3%	0.0%	14.3%	100.0%
4	F	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
+ 48.9%     if (i->color == query.car.color)
F  ST  Pf NT
                query.result++;
            break;
        case 1: // count same model but heavier
+ 51.0%     if (i->model == query.car.model && i->weight > query.
F  ST  Pf NT  F  ST  Pf NT
                query.result++;
            break;
        }
    }
}

#endif
```

Issue details



#		Issue type Filter: All	% of bandwidth	% of fetches	% of write-backs	Fetch utilizatio
4	F	Fetch utilization	48.3%	48.3%	0.0%	14.3%
7	ST	Spat/temp blocking	48.3%	48.3%	0.0%	14.3%

Issue #4: Fetch utilization

This instruction group also shows symptoms of: F 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], database_2_vector.hh:37	43.8%	48.3%	43.6%	14.3%	100.0%

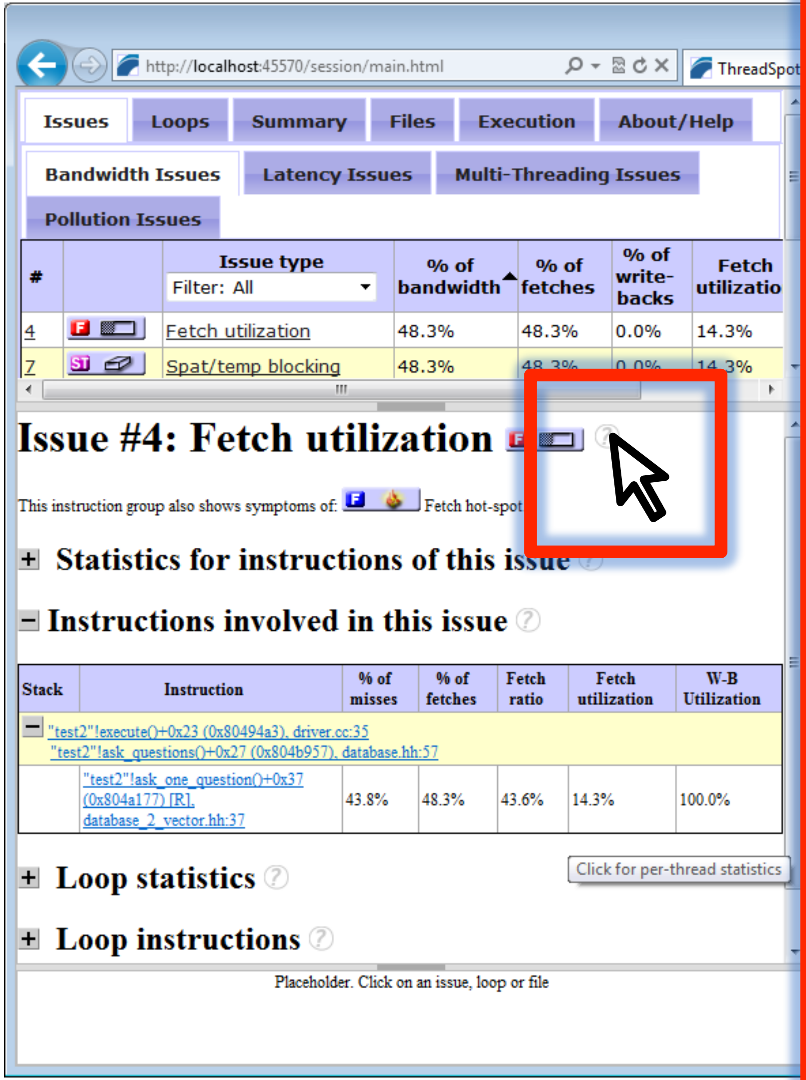
+ Loop statistics

+ Loop instructions

```
};


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
+ 48.9%         if (i->color == query.car.color)
F ST Pf NT
                query.result++;
            break;
        case 1: // count same model but heavier
+ 51.0%         if (i->model == query.car.model && i->weight > query.
F ST Pf NT F ST Pf NT
                query.result++;
            break;
        }
    }
}

#endif
```

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

Context sensitive help



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

Issue #4: Fetch utilization

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

+ Statistics for instructions of this issue

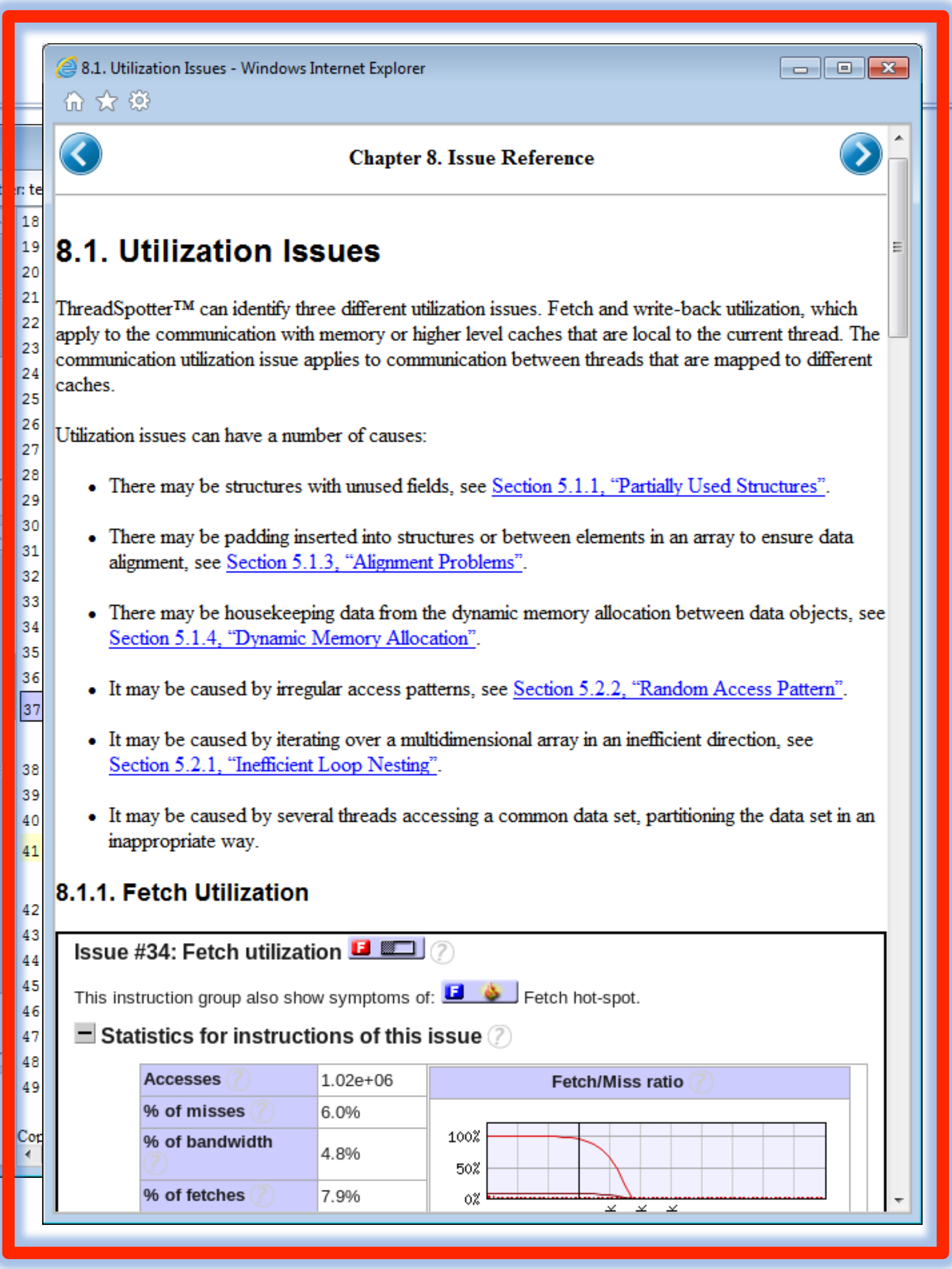
− Instructions involved in this issue

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], 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

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 F

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

− Statistics for instructions of this issue

Accesses	1.02e+06
% of misses	6.0%
% of bandwidth	4.8%
% of fetches	7.9%

EXAMPLE

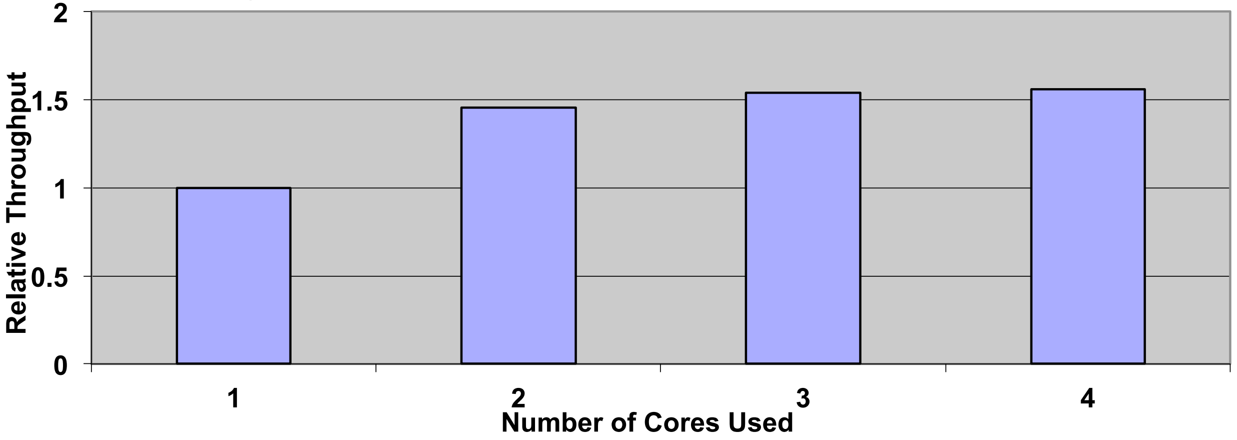
libquantum

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

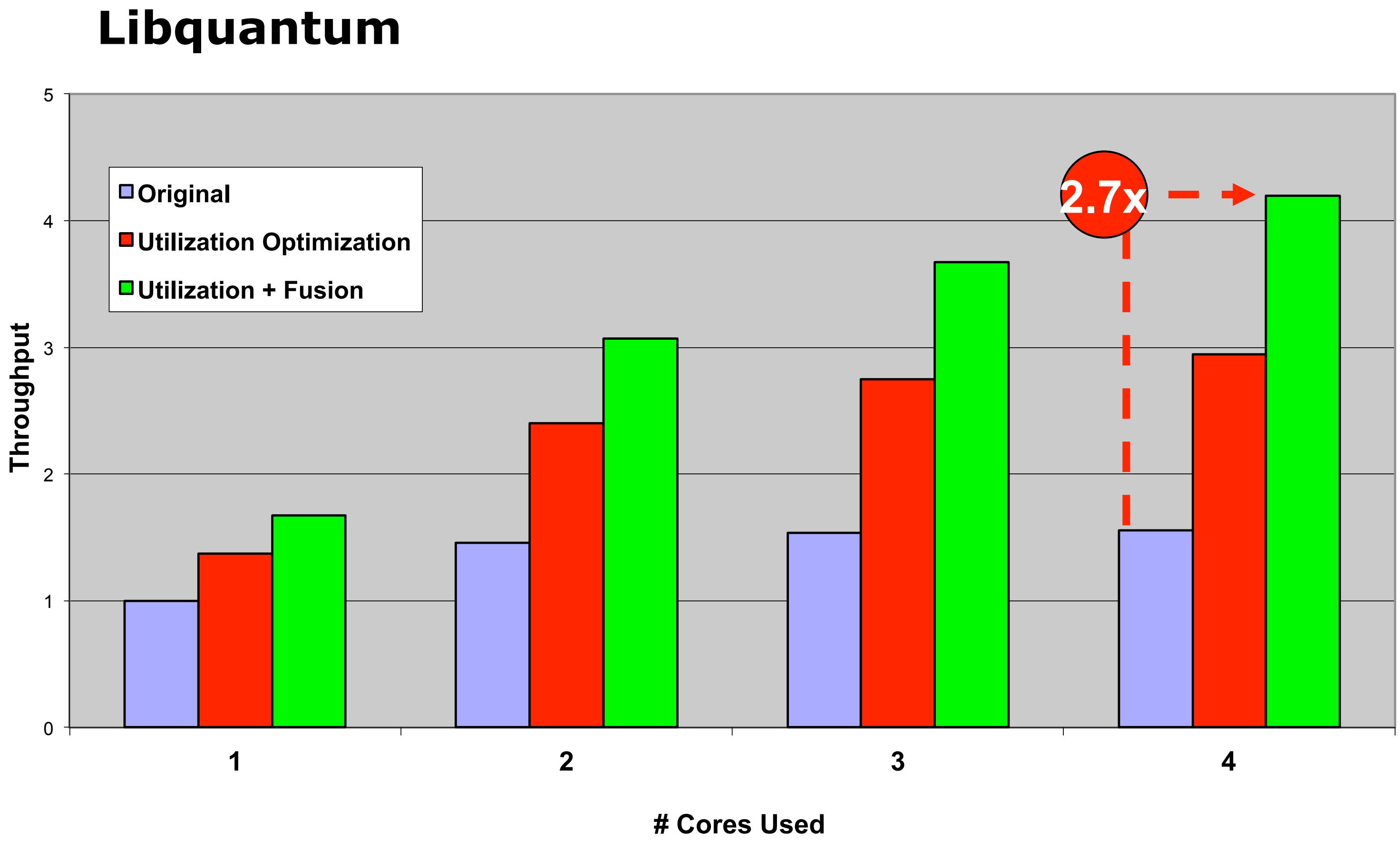
After spatial optimization

libquantum-opt1.tsr

After temporal optimization

libquantum-opt2.tsr

Result


Libquantum
Original
Utilization Optimization
Utilization + Fusion
2.7x
Throughput
0
1
2
3
4
5
1
2
3
4
Cores Used

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)

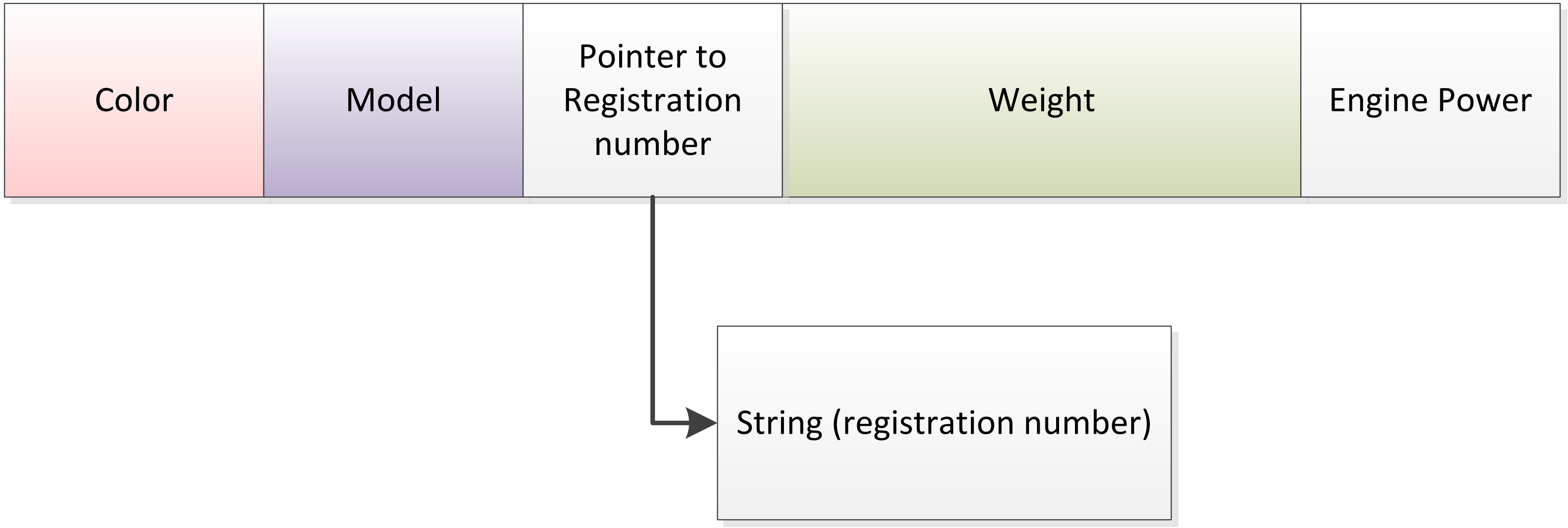
- Inefficient use of shared memory in multithreaded programs:
  - False sharing
  - Poor communication efficiency
    (few bytes transferred per cache line downgrade)

- 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.e
    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

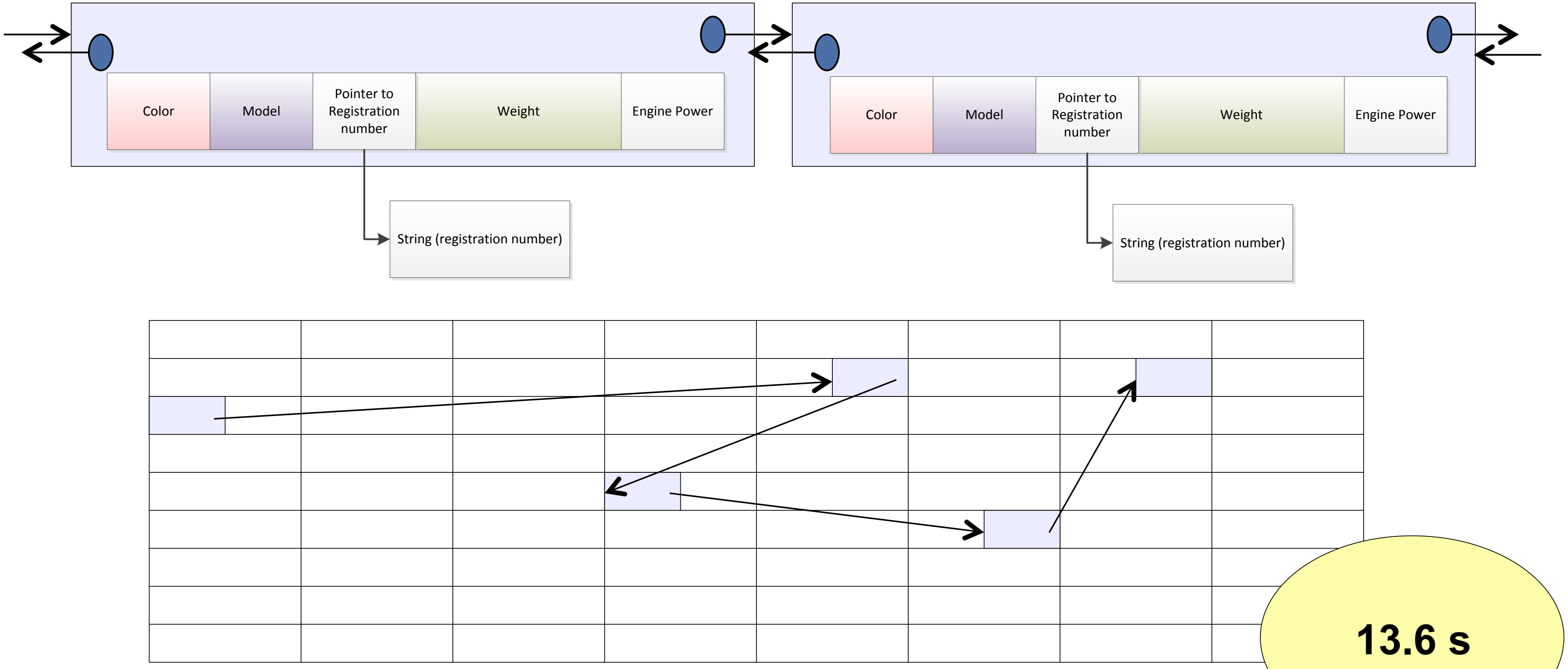
For each query

Traverse entire list

Two variants of Record access

Linked list (doubly linked, std::list)

A linked list is the worst possible scenario.

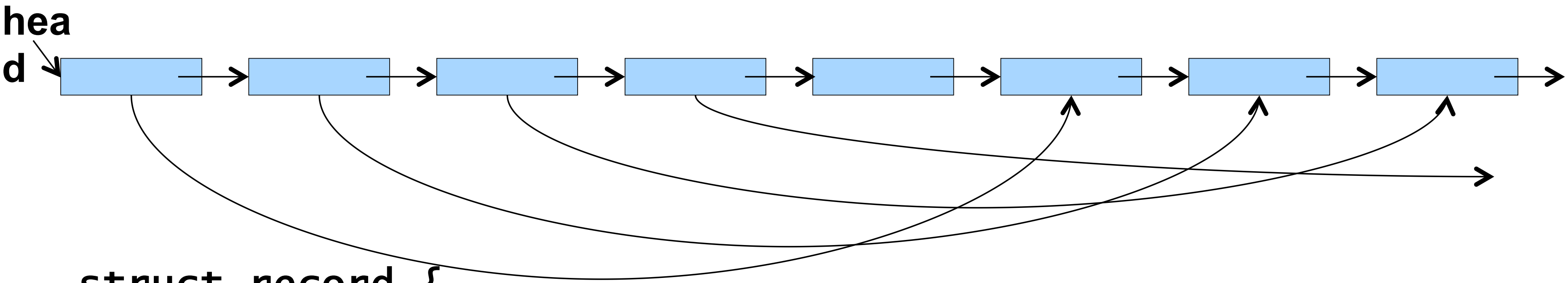


13.6 s

test1.tsr

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

test1c.tsr

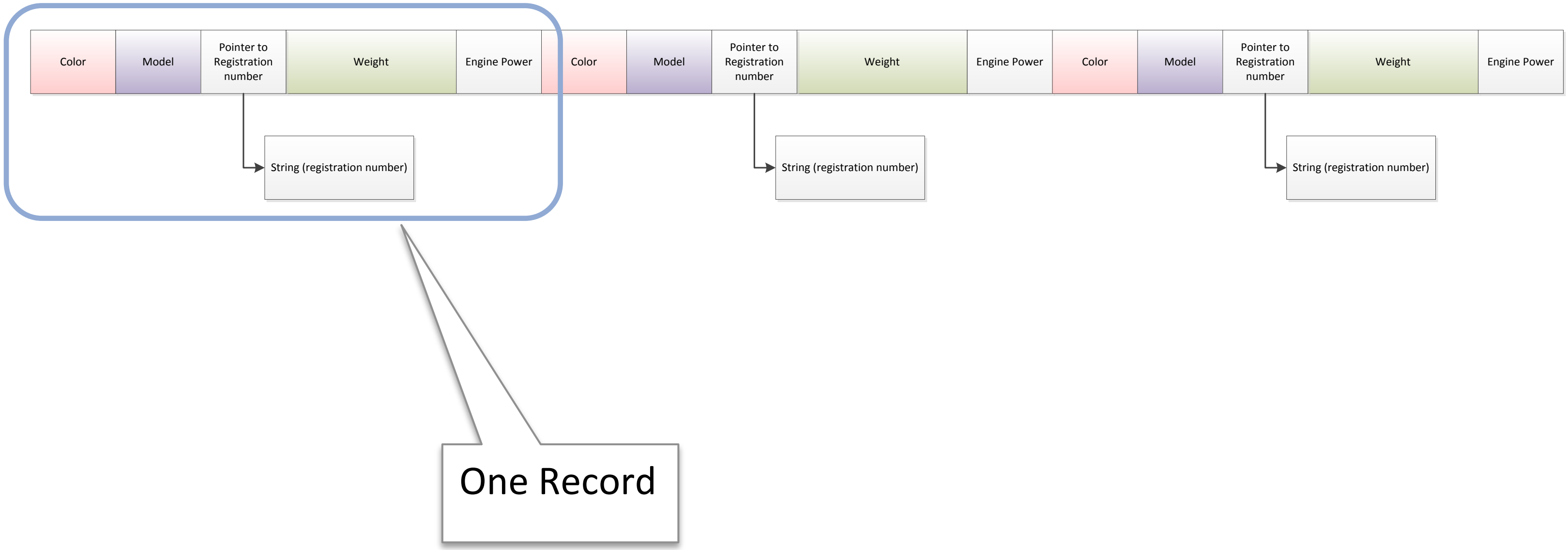
Arrange data consecutively in memory

- 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

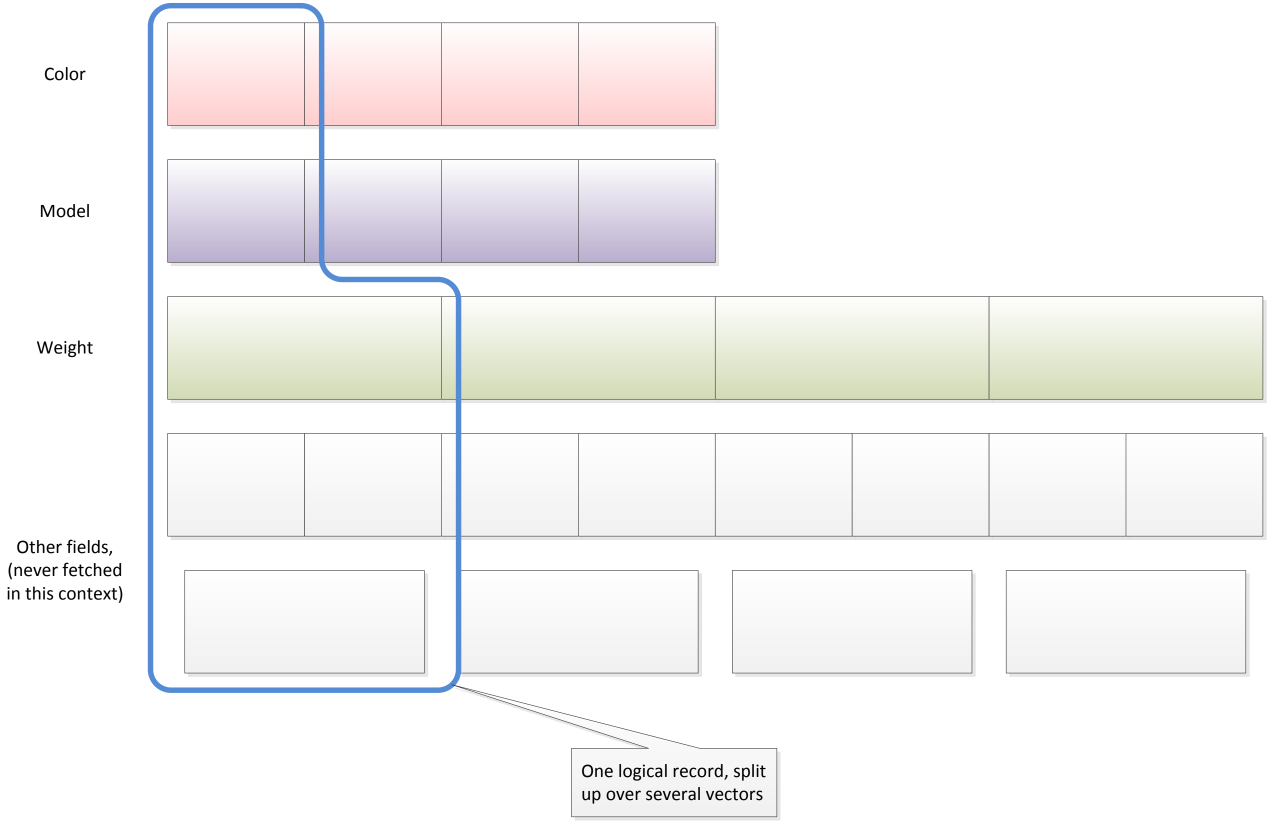


0.88 s

test2.tsr

Improvements: packing

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

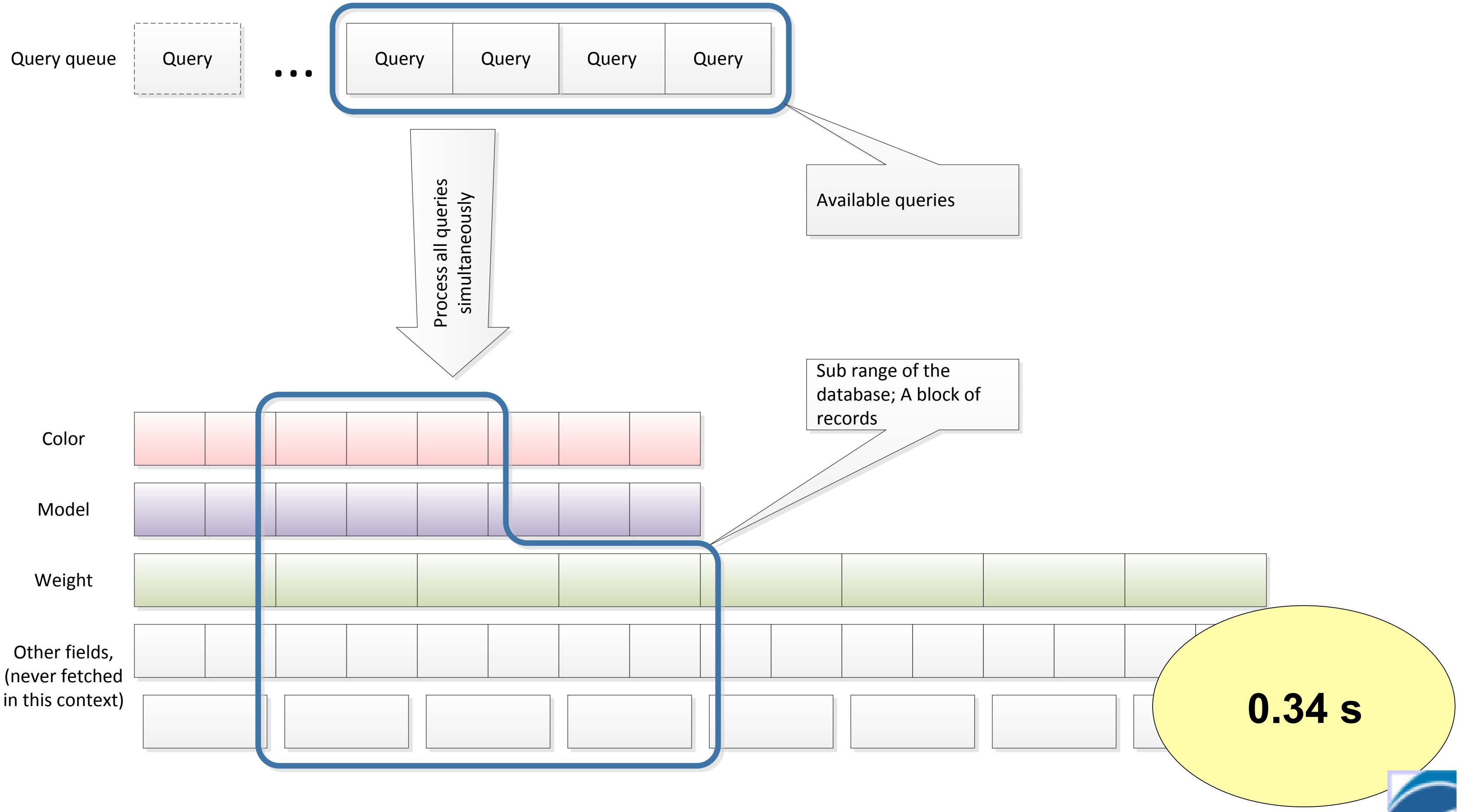


0.45 s

test3.tsr

Improve long term reuse

Blocking means batching and subdividing data to fit in cache.

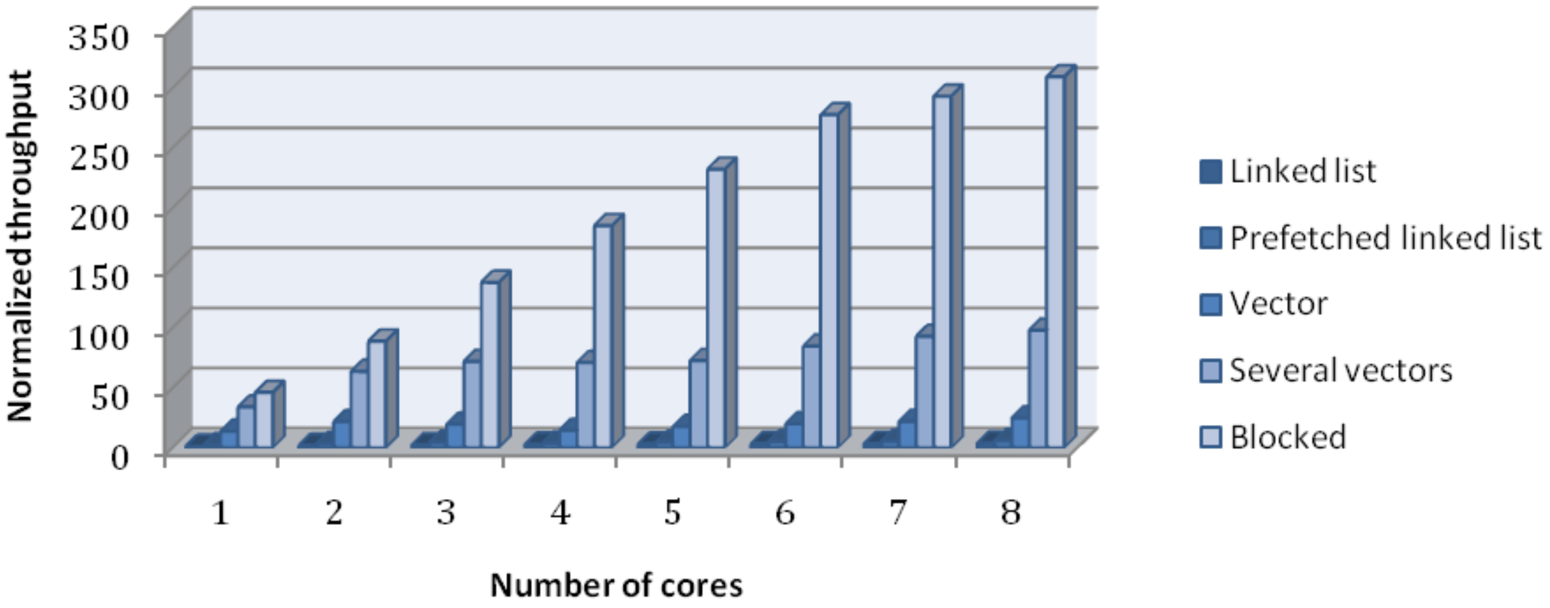


Summary example timings

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



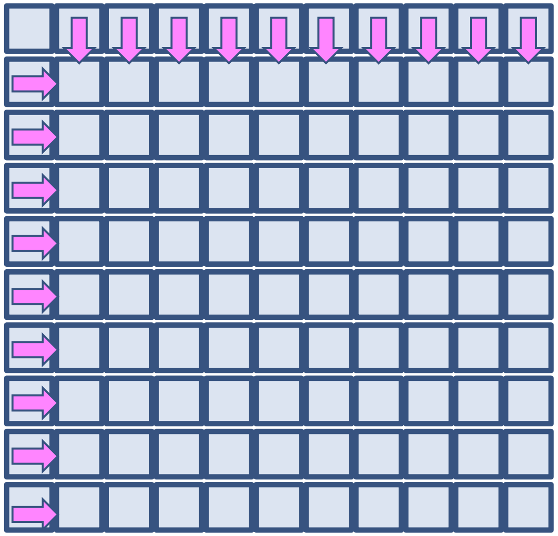
EXAMPLE (BLOCKING)

Gaussian Elimination with pivoting

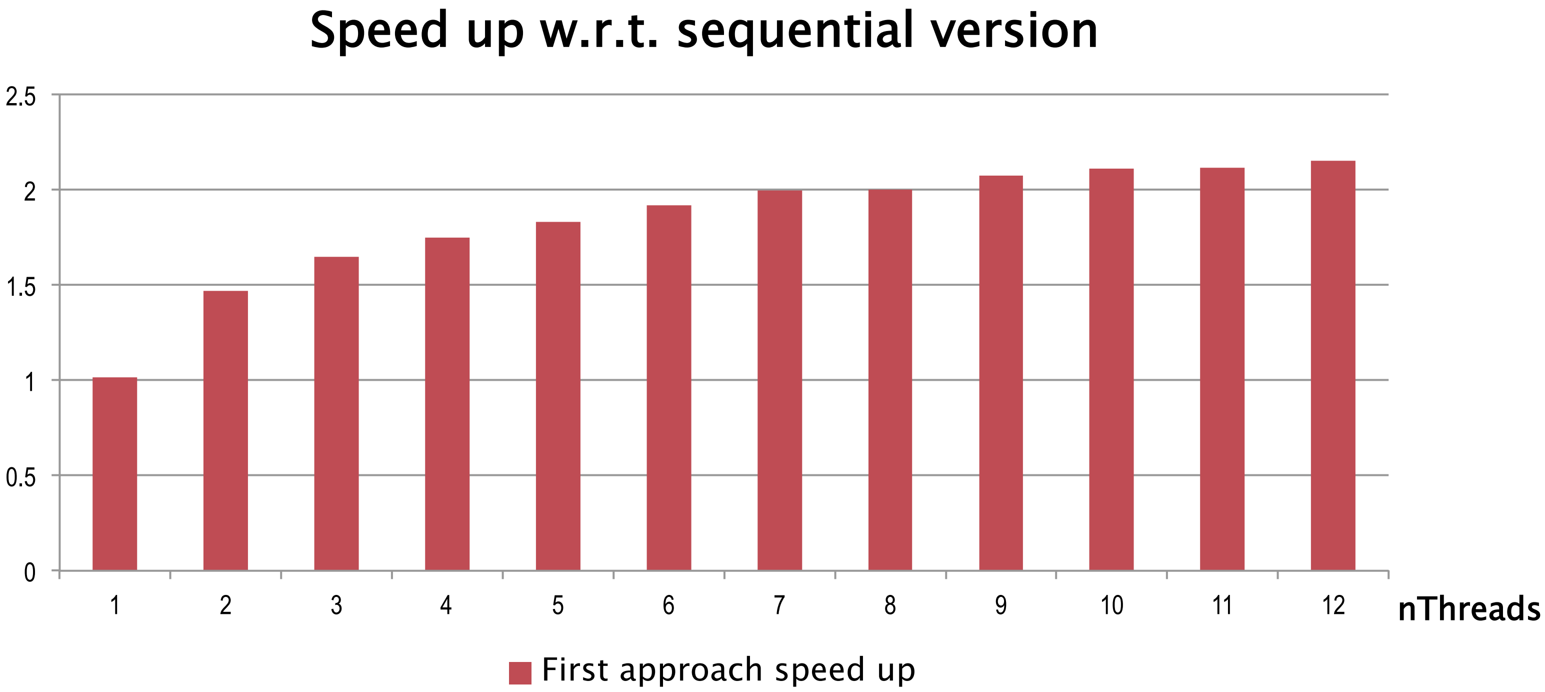
(Forward elimination step)

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



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.

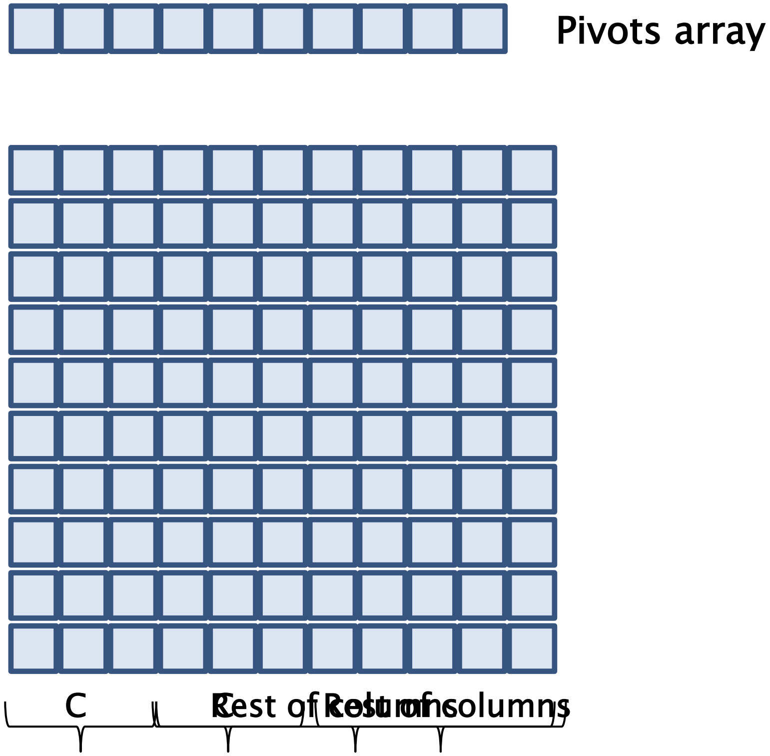
- How? To use each column more times means we have to:

➢ Arrange code to make more pivots available!

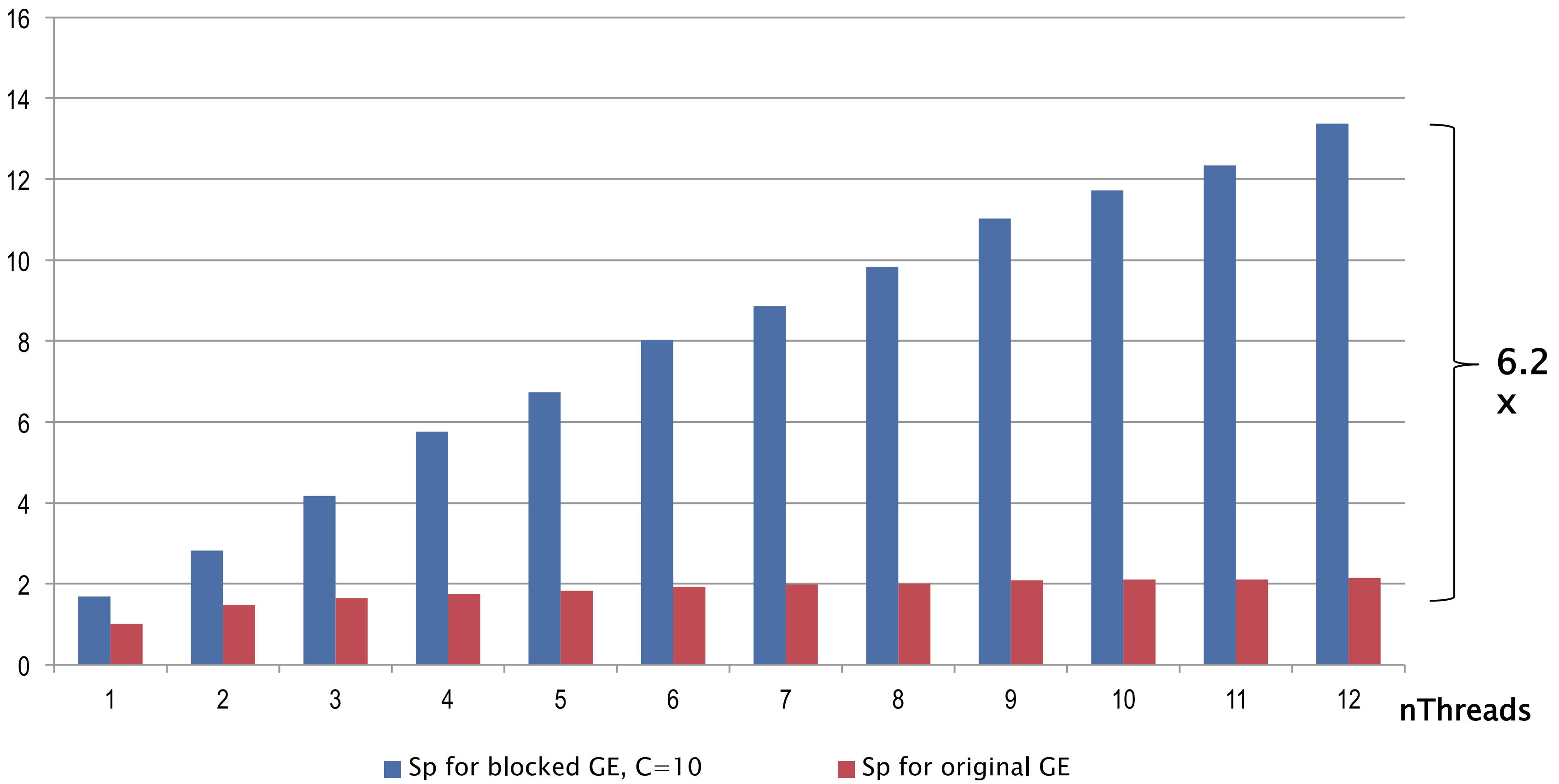
Blocking GE

```
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
```

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

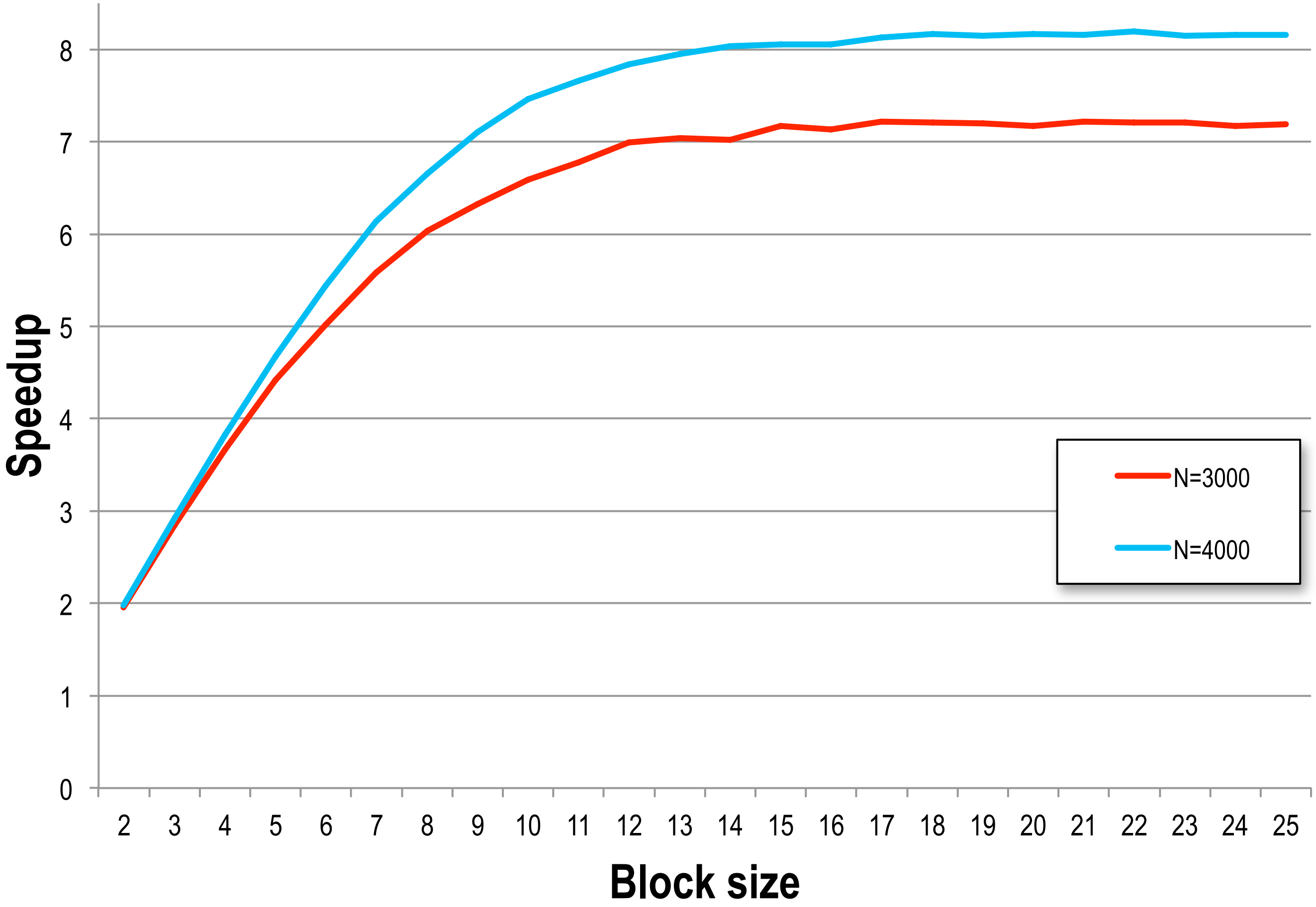


Speed-up relative to the sequential time



6.2 x

Selecting a Good Blocking Size



EXAMPLE (FALSE SHARING)

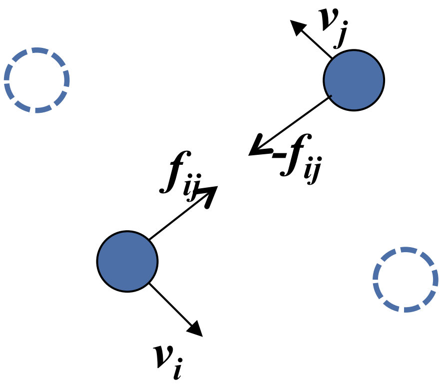
N-Body

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

```
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
```



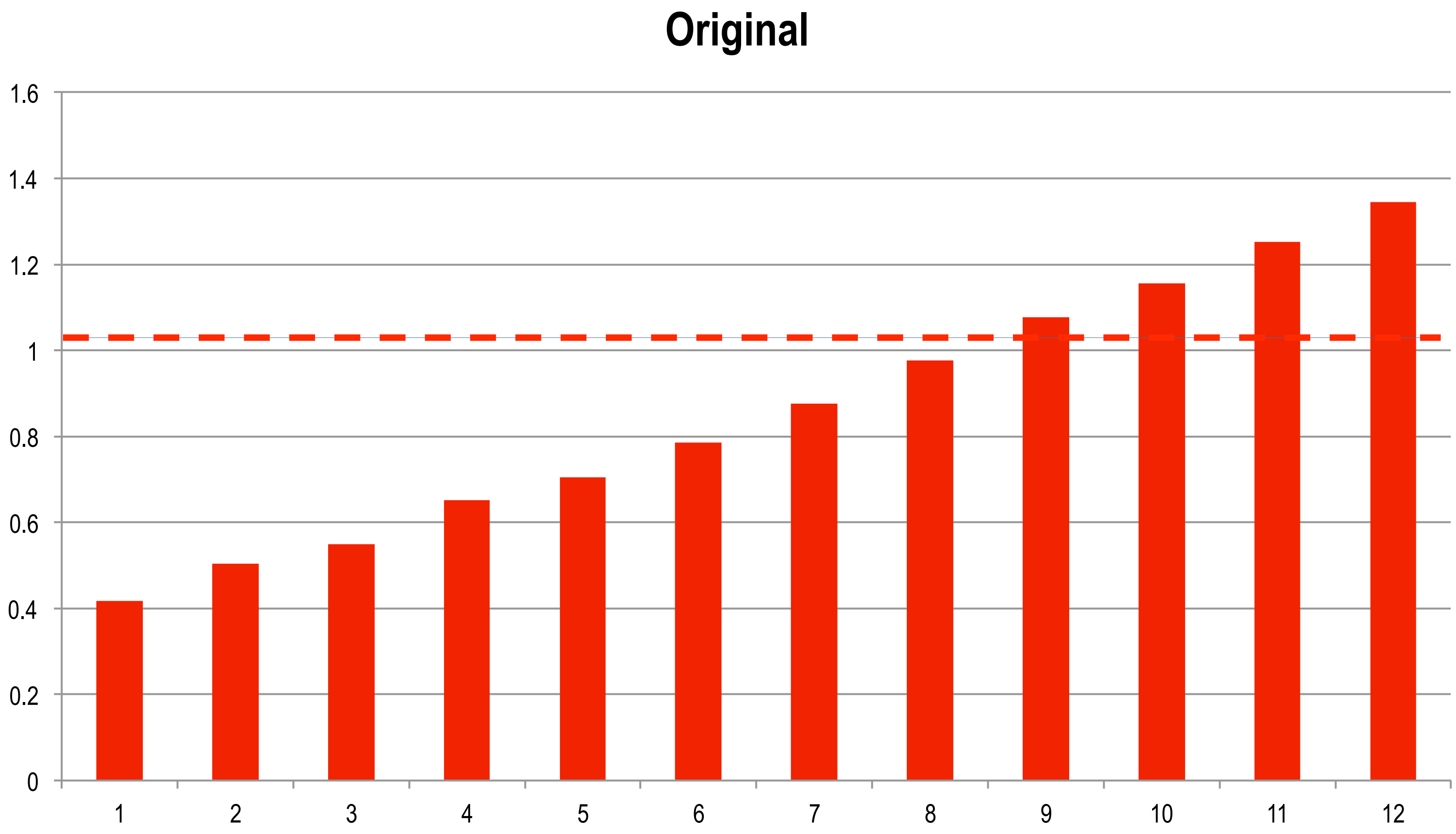
Algorithm

```
#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;
```

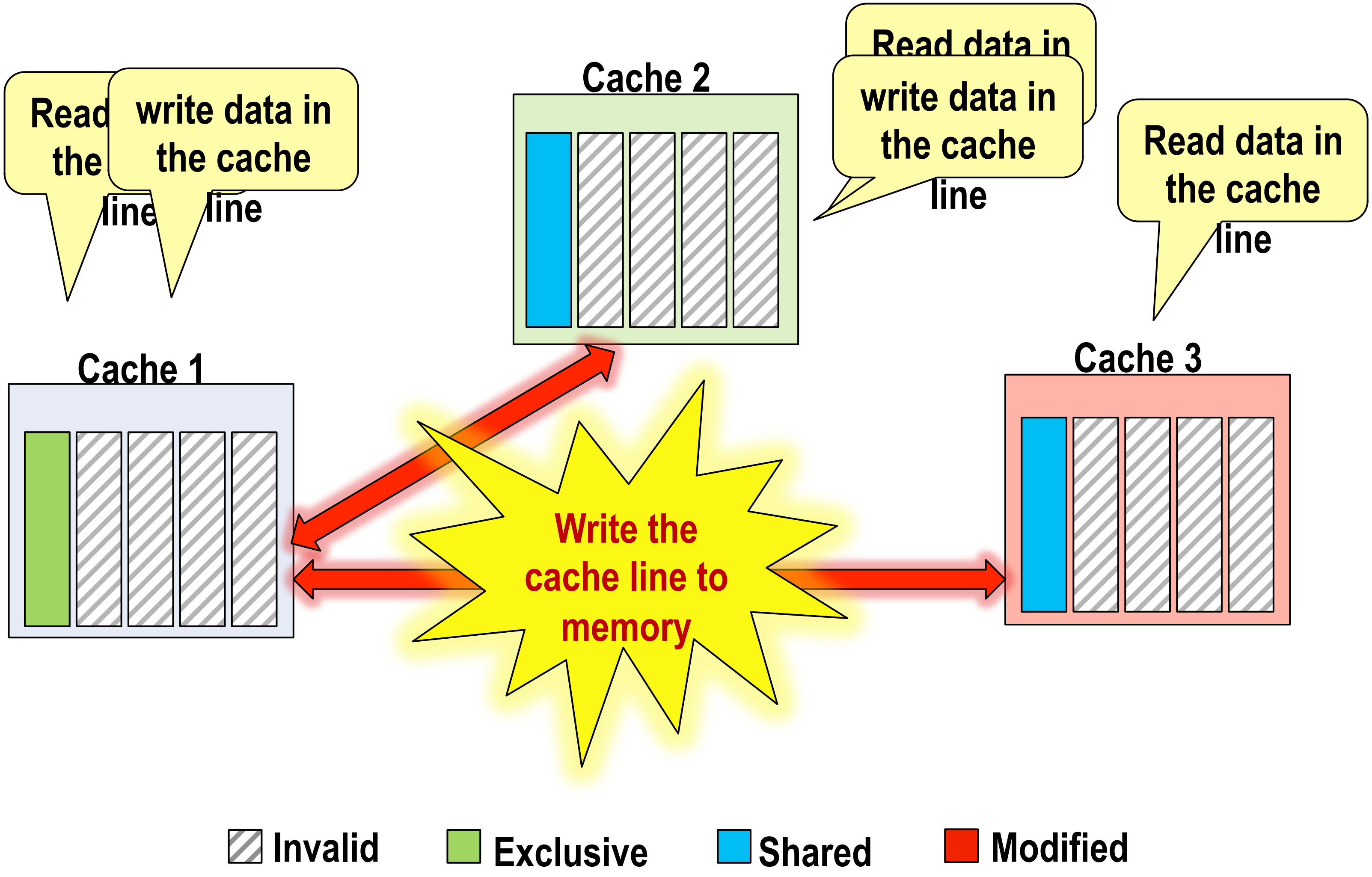
#pragma omp atomic
force updates

Speed up or slow down?!



nbody-orig.tsr

Cache coherence



Read the line

write data in the cache line

Cache 2

Read data in

write data in the cache line

Read data in the cache line

Cache 1

Write the cache line to memory

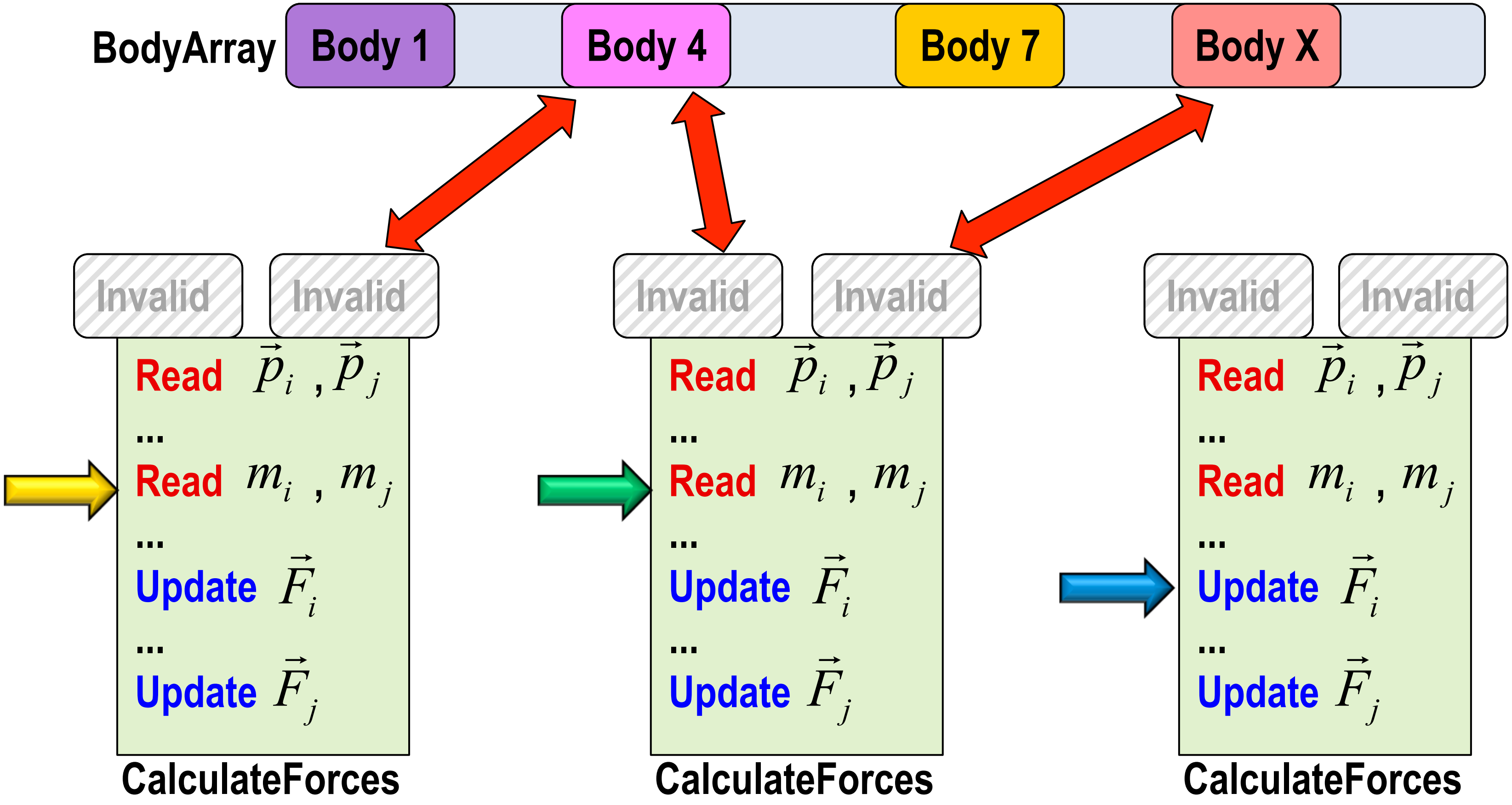
Cache 3

▨ Invalid 🟩 Exclusive 🟦 Shared 🟥 Modified

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



BodyArray **Body 1** **Body 4** **Body 7** **Body X**

Invalid Invalid

Read $\vec{p}_i , \vec{p}_j$

…

Read m_i , m_j

…

Update $\vec{F}_i$

…

Update $\vec{F}_j$

CalculateForces

Invalid Invalid

Read $\vec{p}_i , \vec{p}_j$

…

Read m_i , m_j

…

Update $\vec{F}_i$

…

Update $\vec{F}_j$

CalculateForces

Invalid Invalid

Read $\vec{p}_i , \vec{p}_j$

…

Read m_i , m_j

…

Update $\vec{F}_i$

…

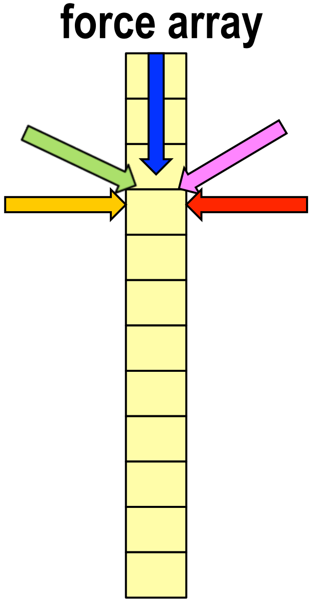
Update $\vec{F}_j$

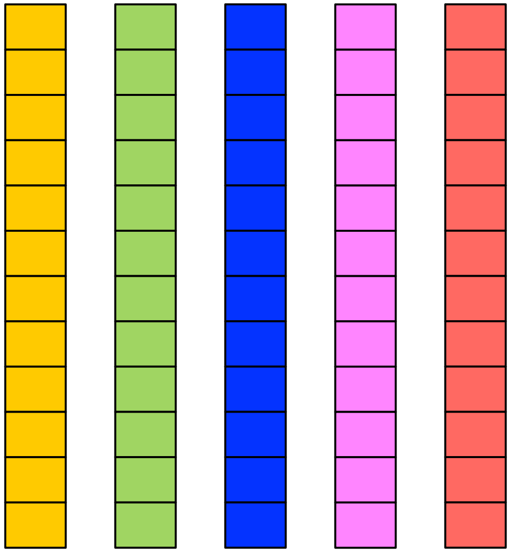
CalculateForces

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!



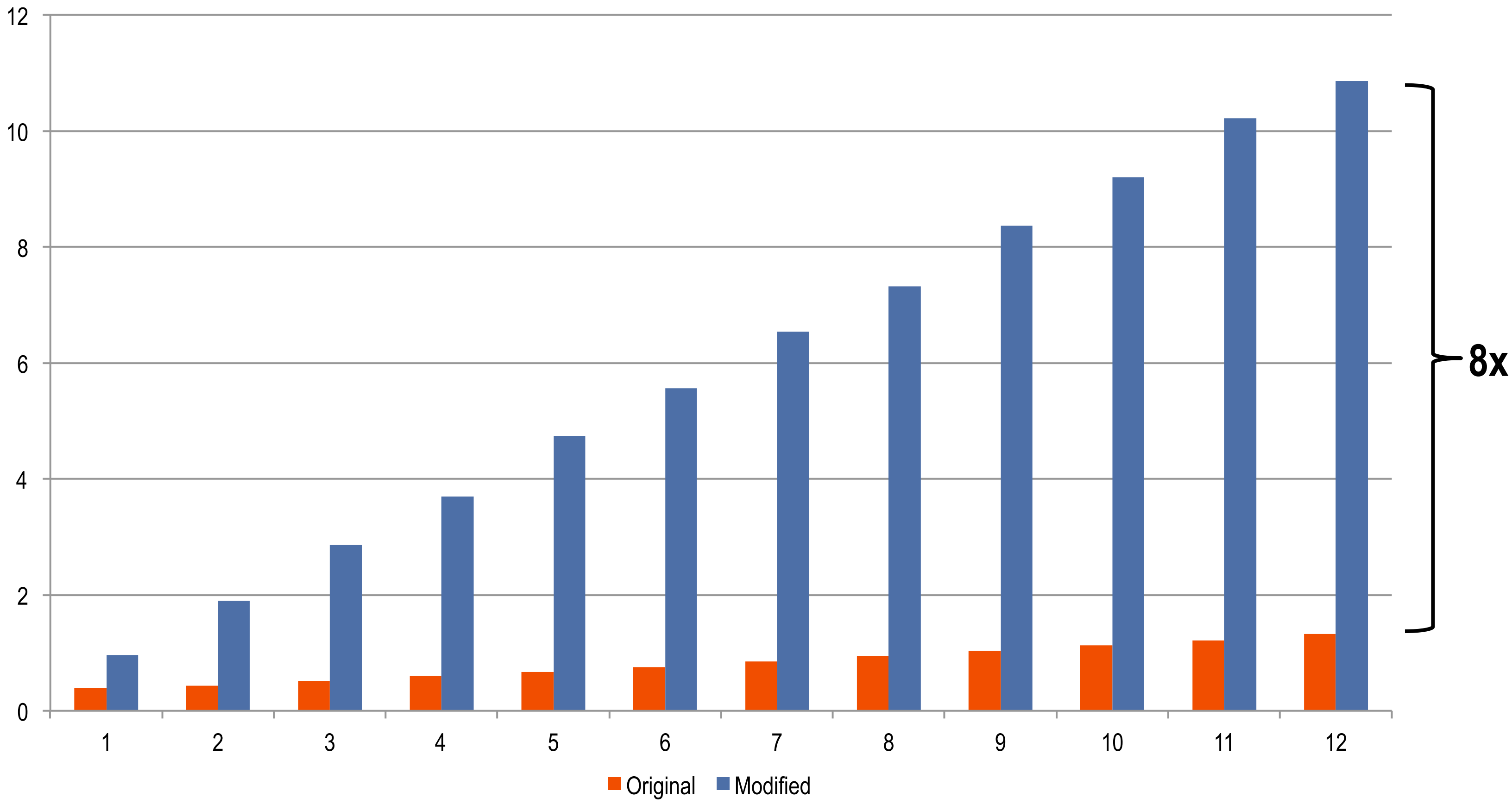


Modified algorithm

```
#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



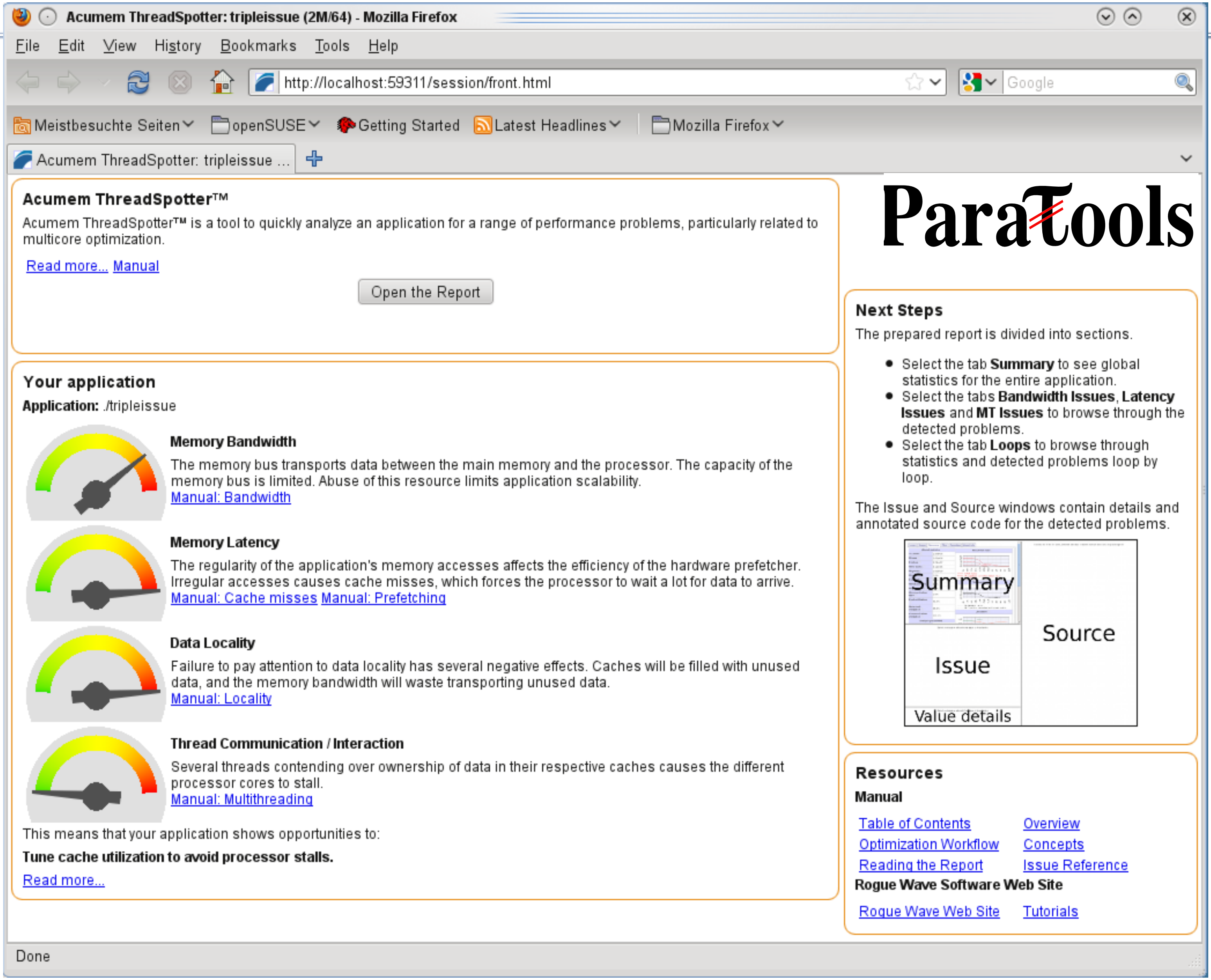
SUMMARY

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...

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.

Resources

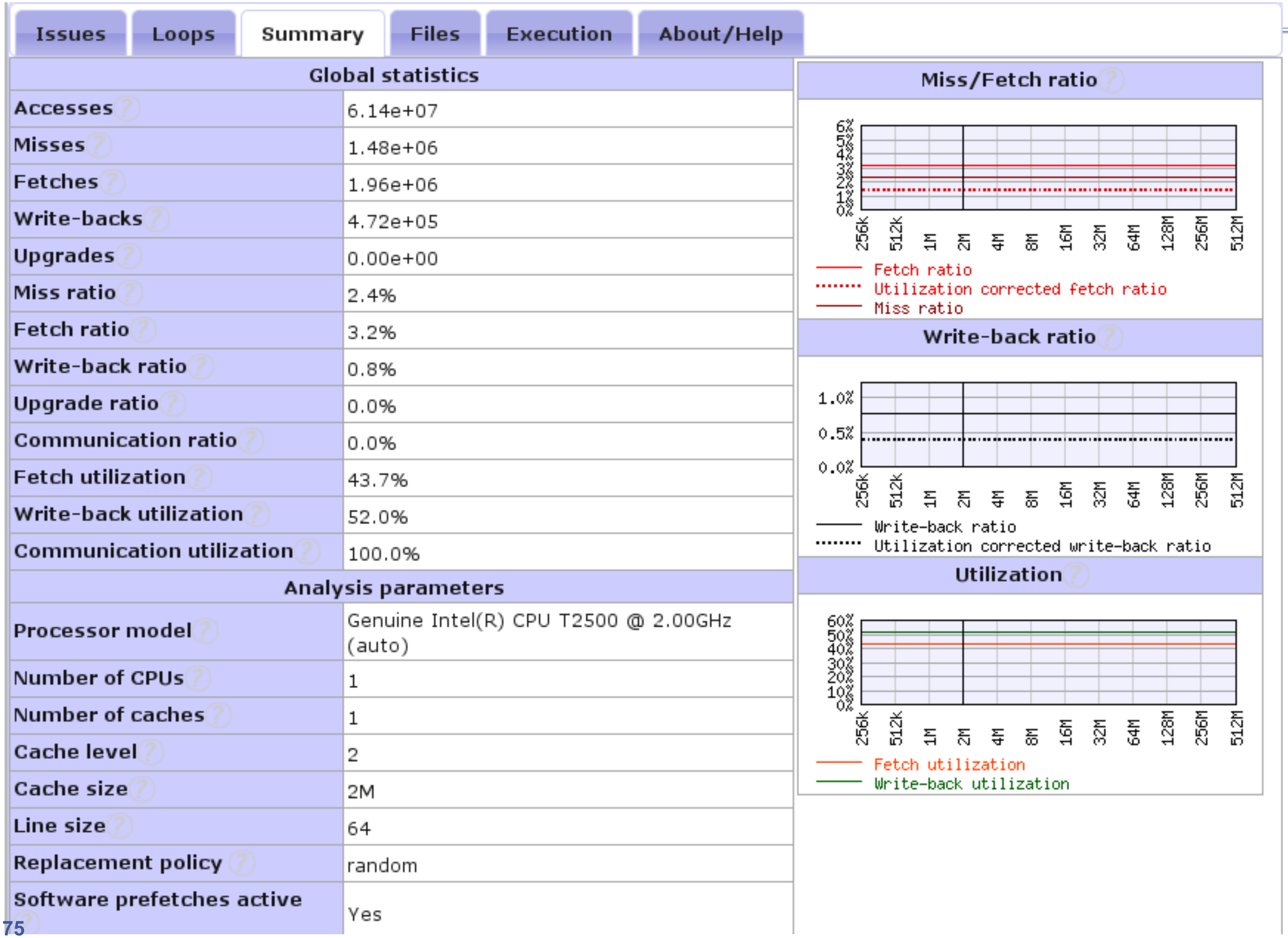
Manual

Table of Contents	Overview
Optimization Workflow	Concepts
Reading the Report	Issue Reference

Rogue Wave Software Web Site

Rogue Wave Web Site Tutorials

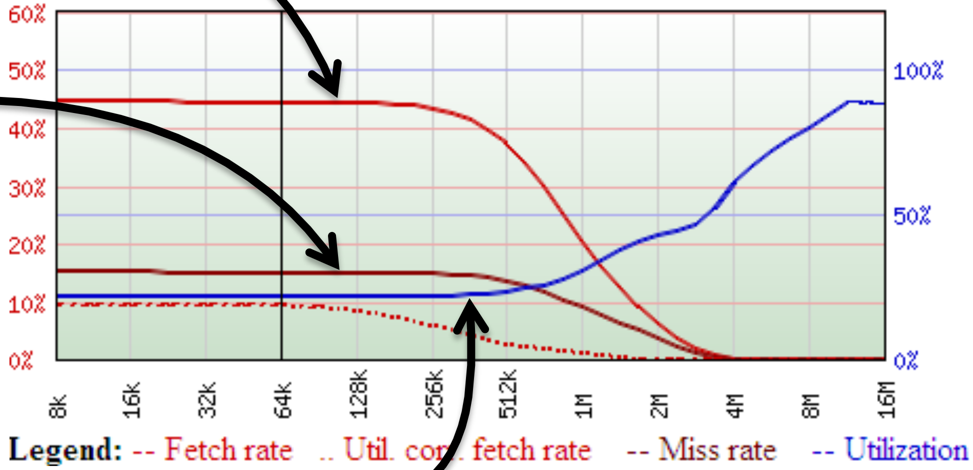
Summary



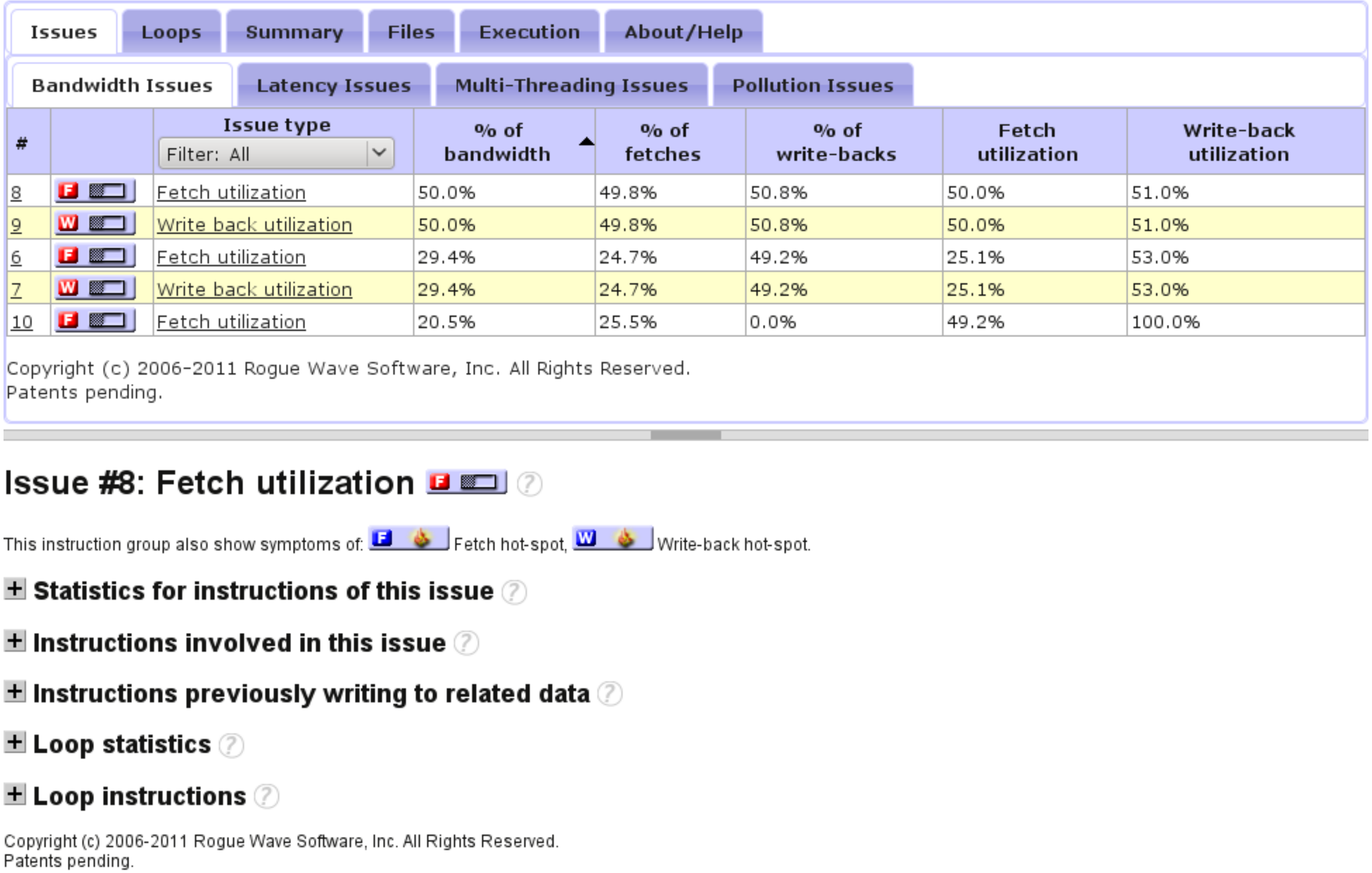
Global statistics	
Accesses	6.14e+07
Misses	1.48e+06
Fetches	1.96e+06
Write-backs	4.72e+05
Upgrades	0.00e+00
Miss ratio	2.4%
Fetch ratio	3.2%
Write-back ratio	0.8%
Upgrade ratio	0.0%
Communication ratio	0.0%
Fetch utilization	43.7%
Write-back utilization	52.0%
Communication utilization	100.0%
Analysis 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

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



#		Issue type	% of bandwidth ▲	% of fetches	% of write-backs	Fetch utilization	Write-back utilization
8	F	Fetch utilization	50.0%	49.8%	50.8%	50.0%	51.0%
9	W	Write back utilization	50.0%	49.8%	50.8%	50.0%	51.0%
6	F	Fetch utilization	29.4%	24.7%	49.2%	25.1%	53.0%
7	W	Write back utilization	29.4%	24.7%	49.2%	25.1%	53.0%
10	F	Fetch utilization	20.5%	25.5%	0.0%	49.2%	100.0%

Issue #8: Fetch utilization F ?

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

+ Statistics for instructions of this issue ?

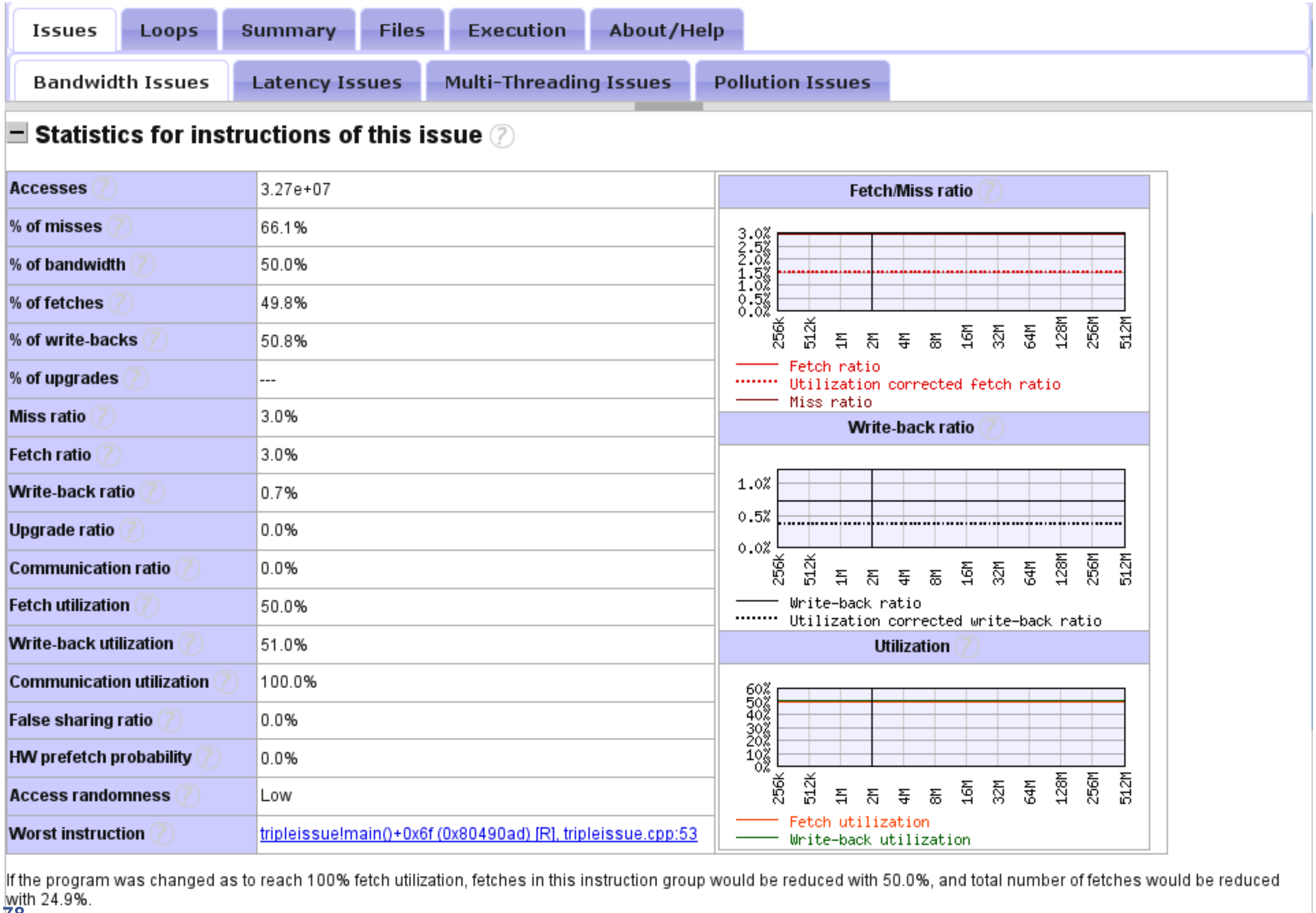
+ Instructions involved in this issue ?

+ Instructions previously writing to related data ?

+ Loop statistics ?

+ Loop instructions ?

Statistics of an Issue

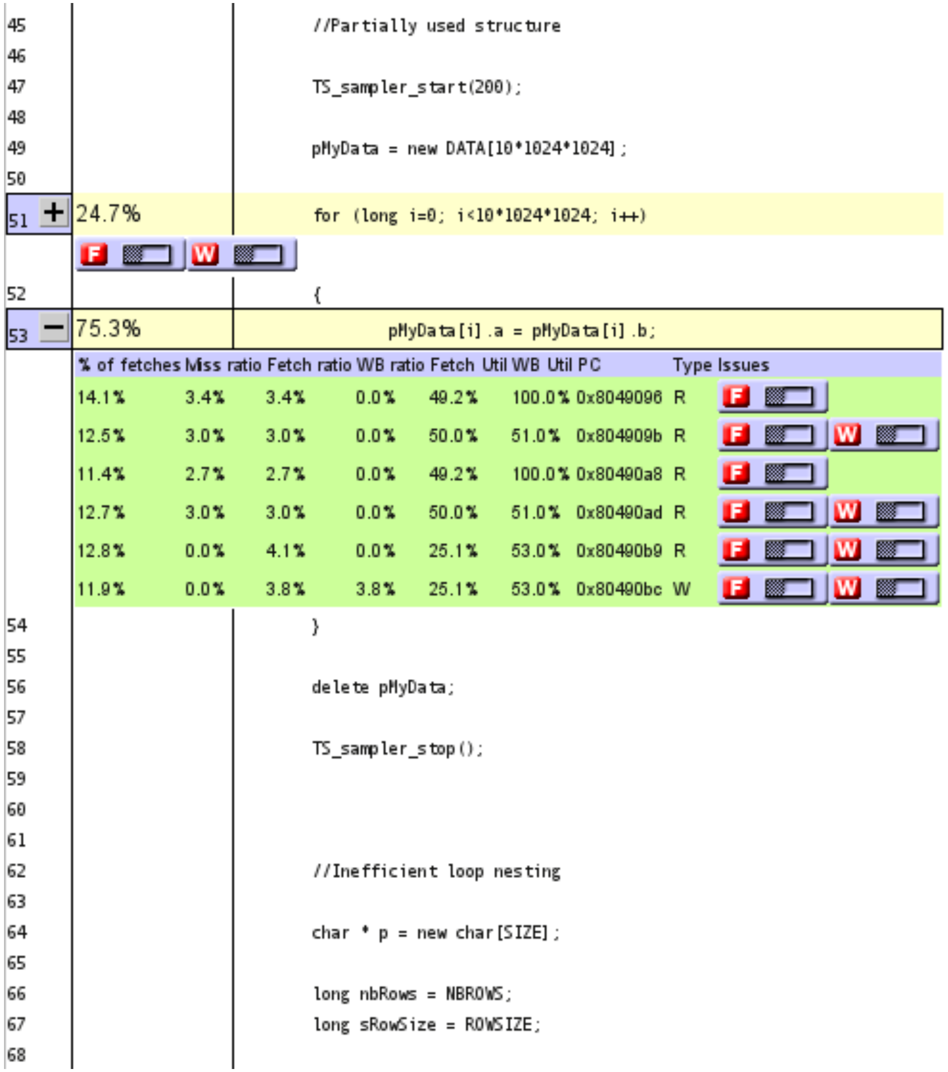


Statistics for instructions of this issue

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



```
            //Partially used structure

            TS_sampler_start(200);

            pMyData = new DATA[10*1024*1024];

24.7%       for (long i=0; i<10*1024*1024; i++)
```

```
            {
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	F
12.5%	3.0%	3.0%	0.0%	50.0%	51.0%	0x804909b	R	F W
11.4%	2.7%	2.7%	0.0%	49.2%	100.0%	0x80490a8	R	F
12.7%	3.0%	3.0%	0.0%	50.0%	51.0%	0x80490ad	R	F W
12.8%	0.0%	4.1%	0.0%	25.1%	53.0%	0x80490b9	R	F W
11.9%	0.0%	3.8%	3.8%	25.1%	53.0%	0x80490bc	W	F W

```
            }

            delete pMyData;

            TS_sampler_stop();



            //Inefficient loop nesting

            char * p = new char[SIZE];

            long nbRows = NBROWS;
            long sRowSize = ROWSIZE;

```

Used Icons

Slowspot Issues

F ▭	Fetch utilization
W ▭	Write back utilization
C ▭	Communication utilization
🟥 ⦸	Inefficient loop nesting
🟥 ✭	Random access
Pf Close	Prefetch: too close
Pf Far	Prefetch: too distant
Pf Hit	Prefetch: unnecessary
🟥 False	False sharing

Opportunity issues

S ▱	Spatial blocking
T ▱	Temporal blocking
ST ▱	Spat/temp blocking
🟪 OO	Loop fusion
Pf NT	Non-temporal data
W NT	Non-temporal store possible
F 🔥	Fetch hot-spot
W 🔥	Write-back hot-spot
C 🔥	Communication hot-spot

Resource Sharing Example

<u>Libquantum</u>

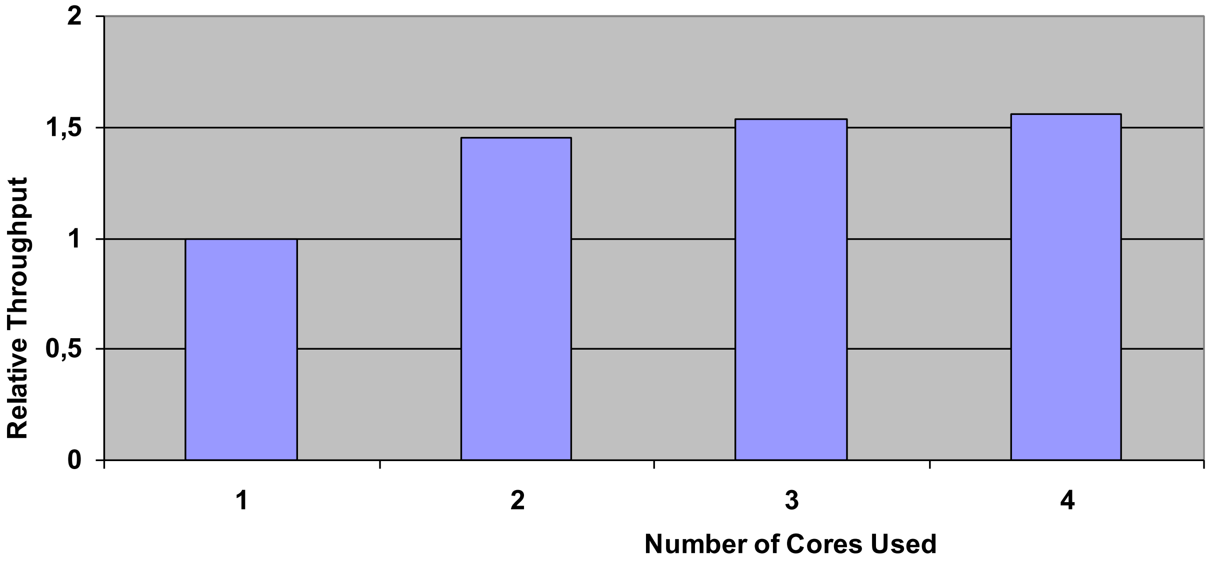
A quantum computer simulation

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

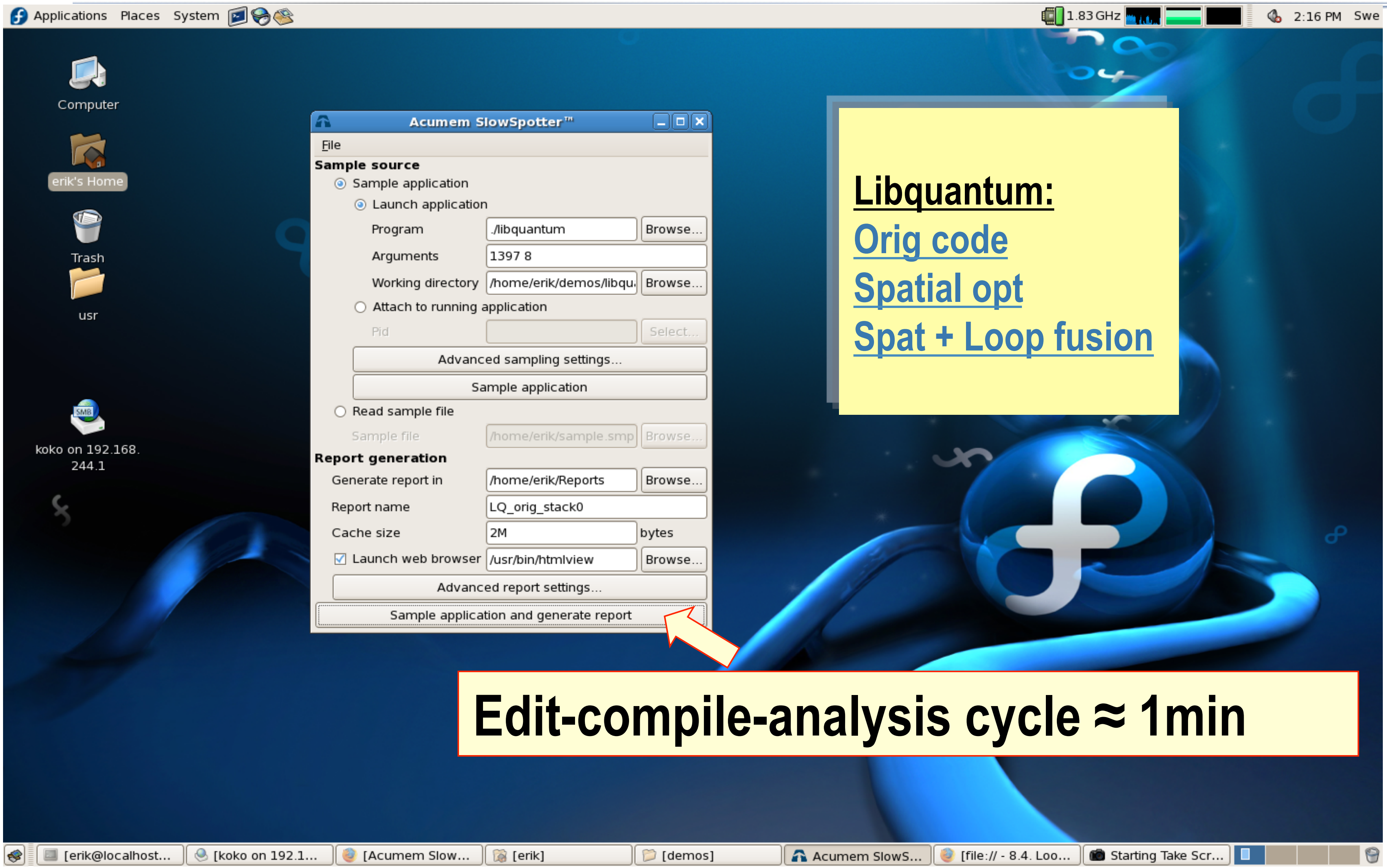
4000+ lines of C, fairly complex code.

Runs an experiment in ~30 min

Throughput improvement:



Demo



Acumem SlowSpotter™

File

Sample source

● Sample application

● Launch application

Program: ./libquantum

Arguments: 1397 8

Working directory: /home/erik/demos/libqu

○ Attach to running application

Pid:

○ Read sample file

Sample file: /home/erik/sample.smp

Report generation

Generate report in: /home/erik/Reports

Report name: LQ_orig_stack0

Cache size: 2M bytes

☑ Launch web browser: /usr/bin/htmlview

Libquantum:

Orig code

Spatial opt

Spat + Loop fusion

Edit-compile-analysis cycle ≈ 1min

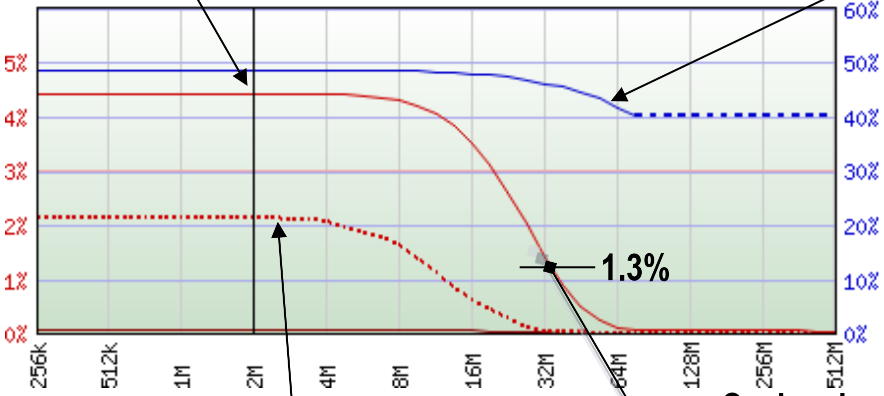
Utilization Analysis

Libquantum

Cache utilization ≈ Fraction of cache data utilized

Fetch rate

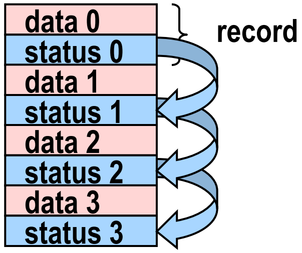
Original Code



Cache size

Predicted fetch rate if utilization = 100%

Need 32 MB per thread!



Only accessing status data in main loop

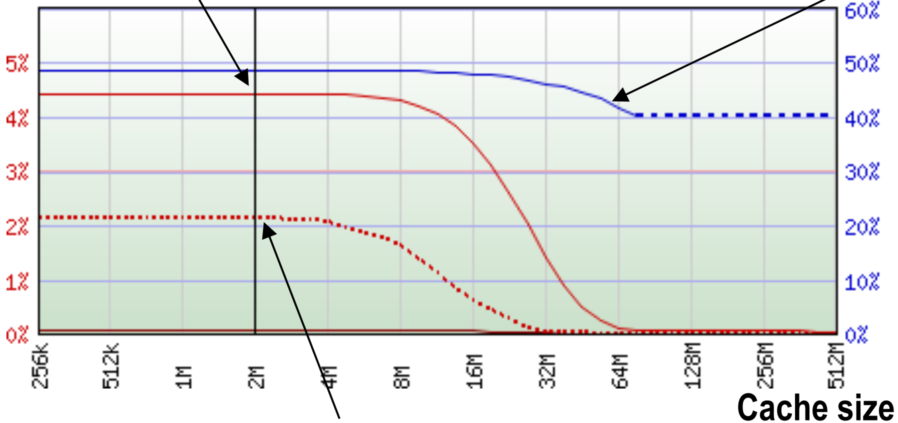
Utilization Analysis

Libquantum

Cache utilization ≈ Fraction of cache data utilized

Fetch rate

Original Code



Predicted fetch rate
if utilization = 100%

Utilization Optimization

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

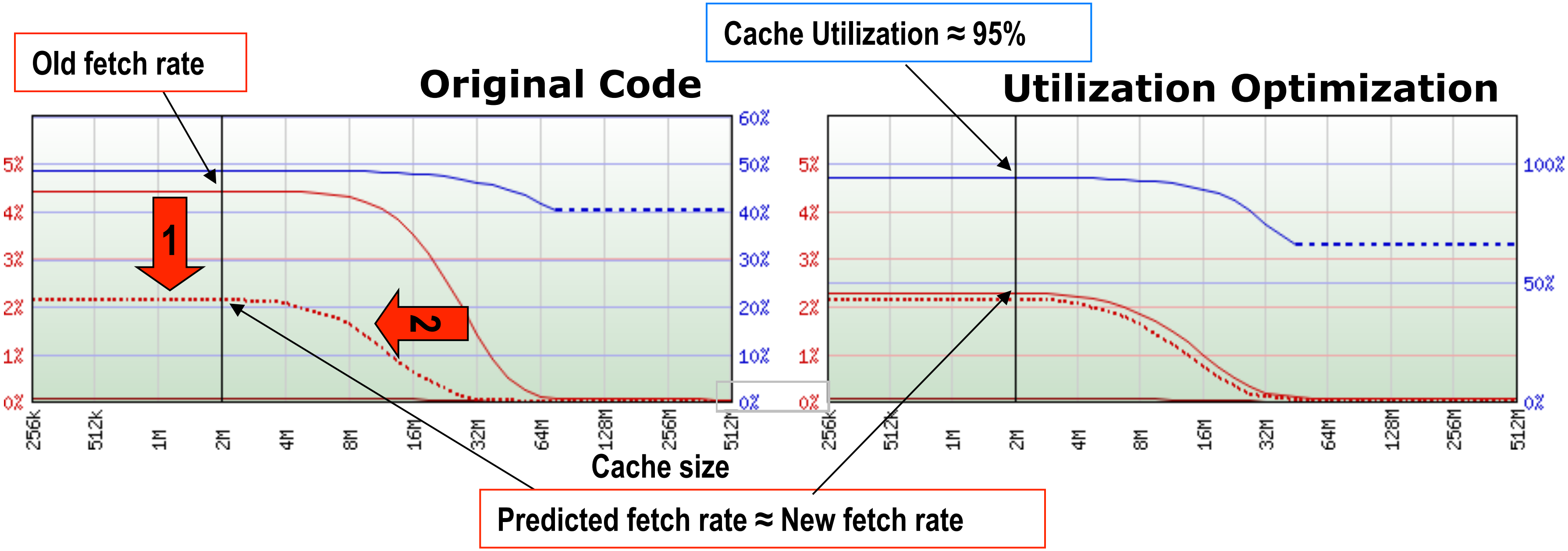
```
for (i=0; i++; i<MAX) {
     ... = huge_data_status[i] + ...
 }
```

ParaTools ThreadSpotter's First Advice: Improve Utilization

➔Change one data structure

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

Utilization Optimization



Old fetch rate

Cache Utilization ≈ 95%

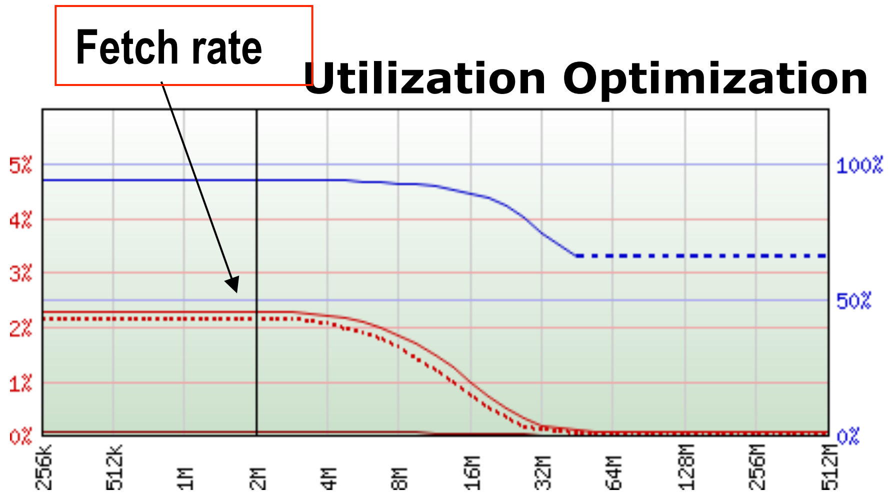
Predicted fetch rate ≈ New fetch rate

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


Fetch rate

Utilization Optimization

Utilization + Fusion Optimization

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

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

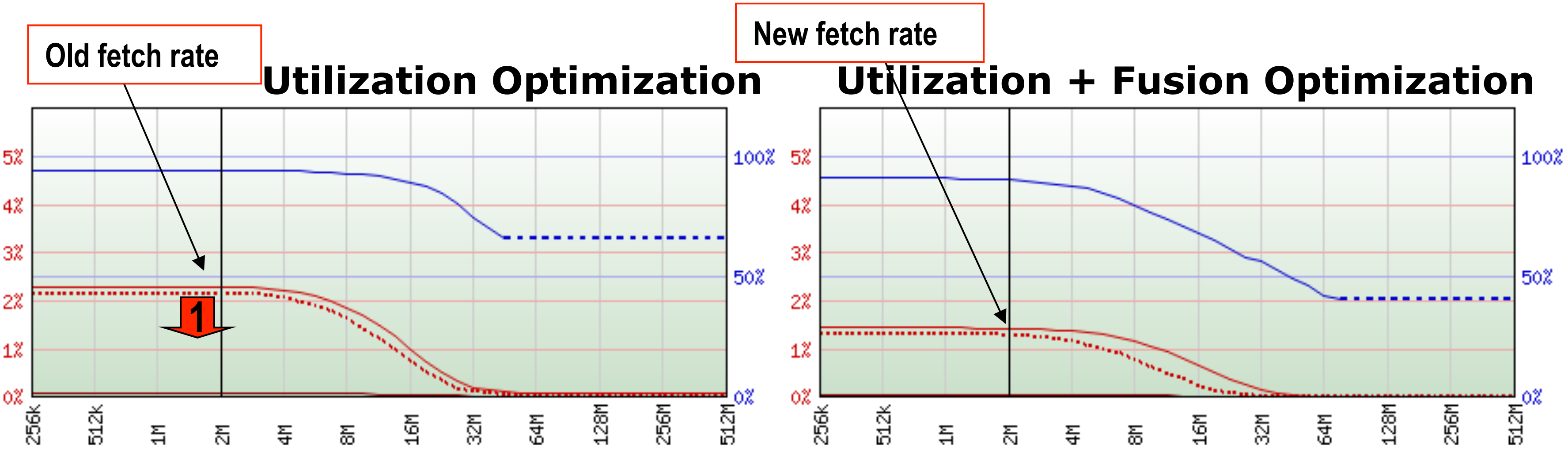
Second-Fifth ParaTools ThreadSpotter Advice: Improve reuse of data through loop fusion

➔**Fuse functions traversing the same data**

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

Effect: Loop Fusion

SPEC CPU2006-462.libquantum



Old fetch rate

New fetch rate

Utilization Optimization

Utilization + Fusion Optimization

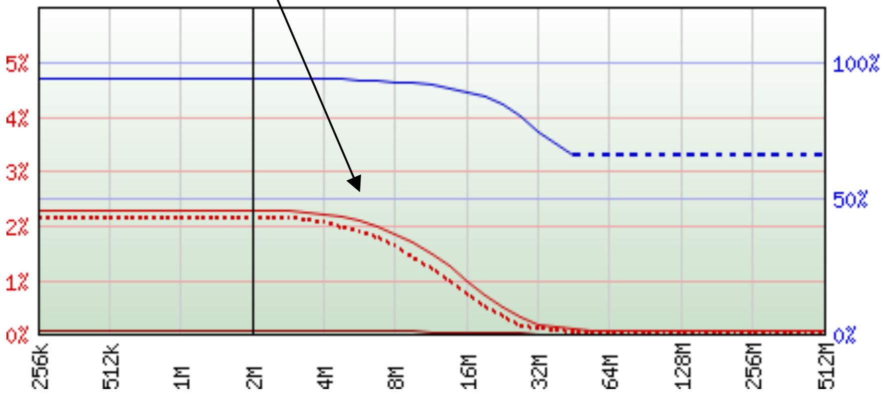
- The miss in the second loop goes away
- Still need the same amount of cache to fit “all data”

Utilization + Loop Fusion

Libquantum

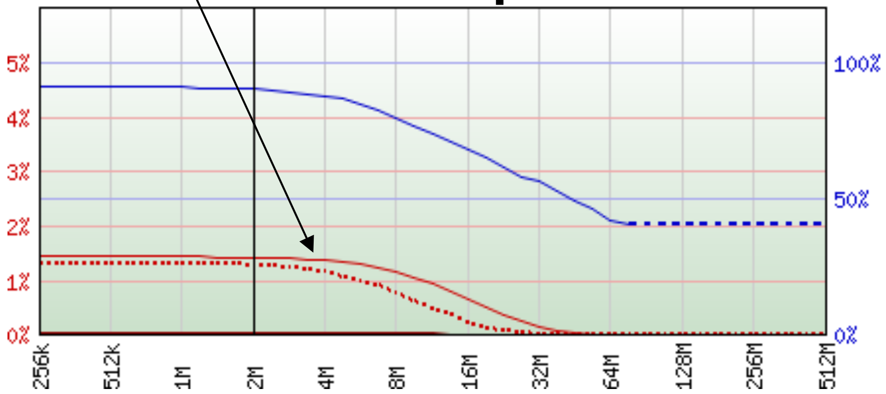
Old fetch rate

Utilization Optimization



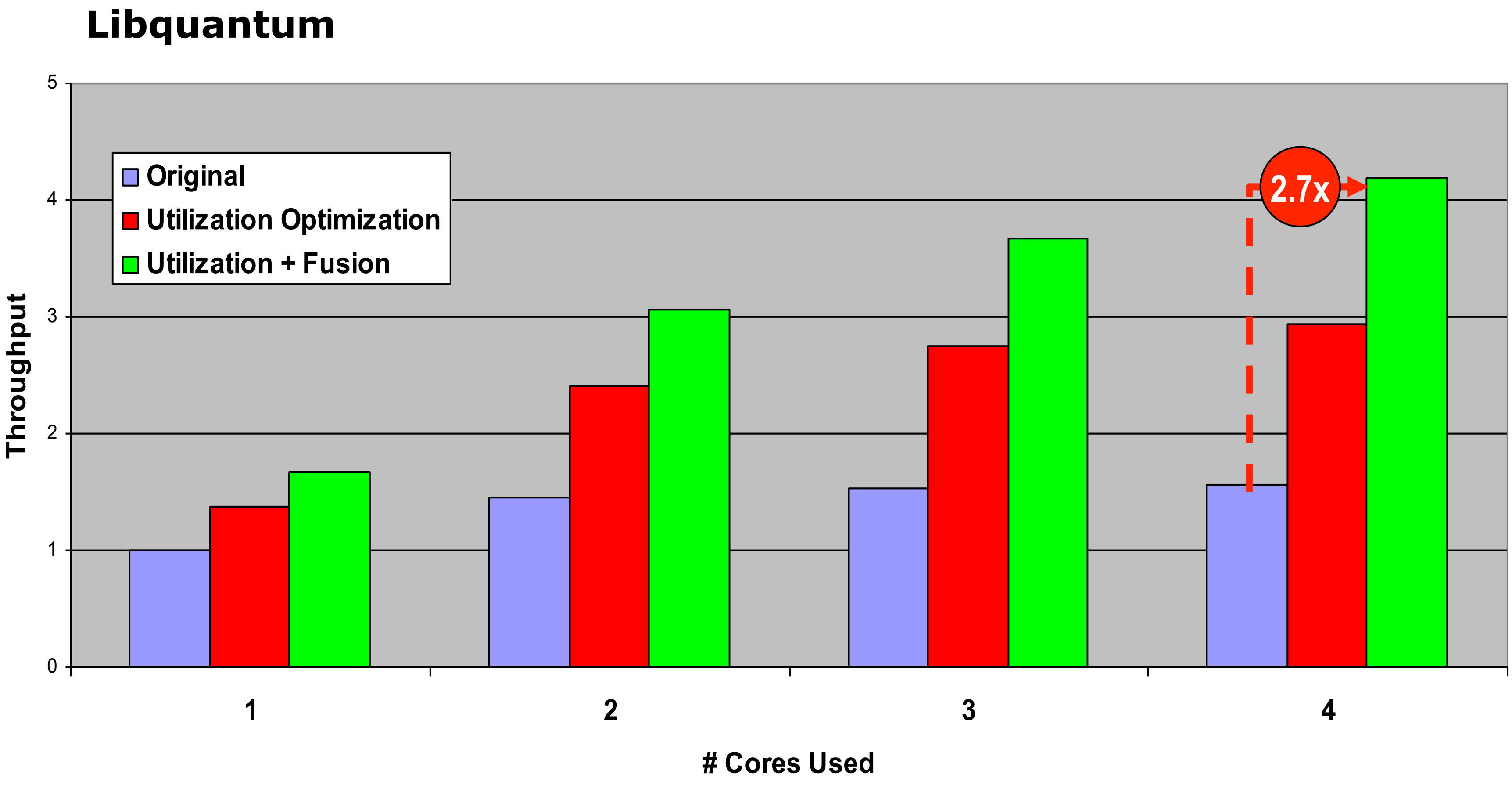
New fetch rate

Utilization + Loop Fusion



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

Summary


Libquantum
5
4
3
2
1
0
Throughput
Original
Utilization Optimization
Utilization + Fusion
2.7x
1
2
3
4
Cores Used

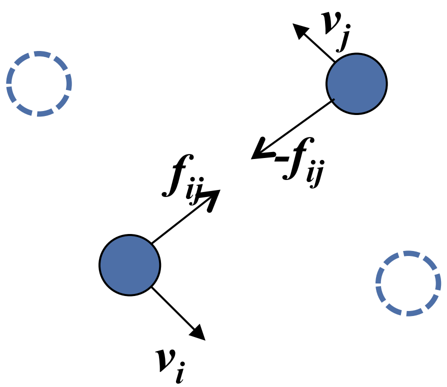
Another Demo – N-body

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
```



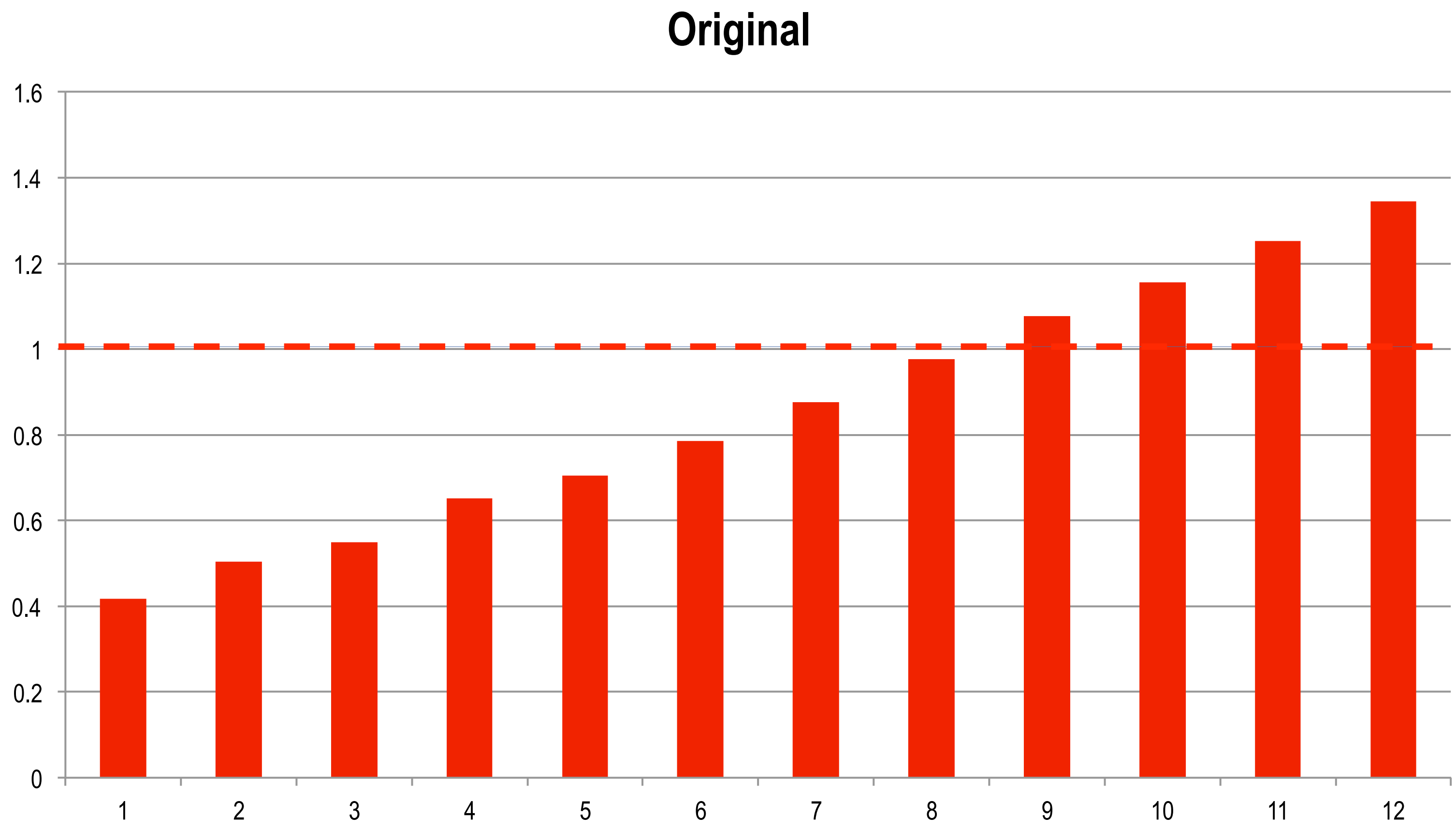
Algorithm

```
#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]);
}
```

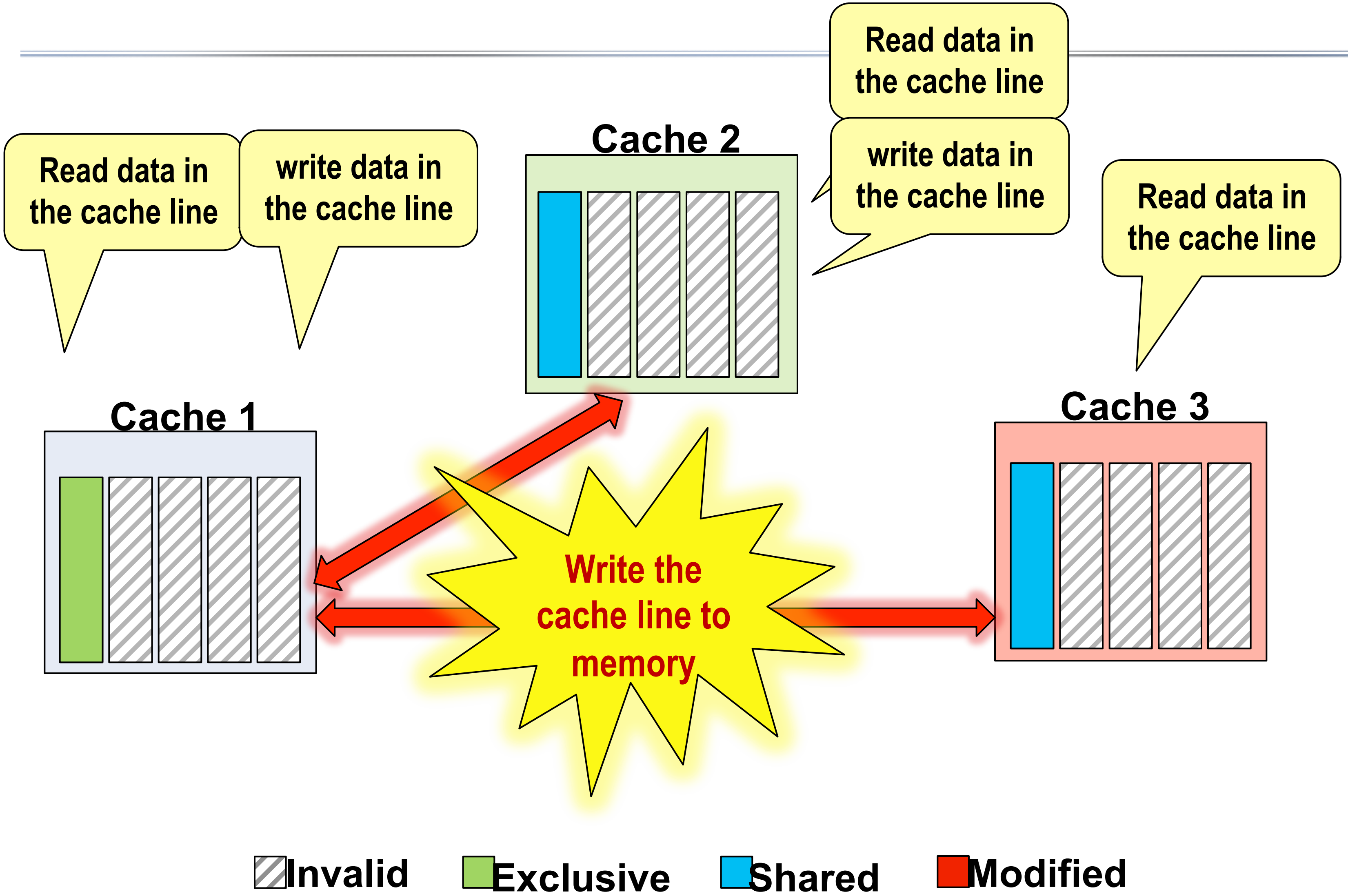
```
typedef struct
{
        double px,py;
        double vx,vy;
        double fx,fy;
        double m;
} body;
```

```
#pragma omp atomic
    force updates
```

Speed Up or Slow Down?



Cache Coherence



Read data in the cache line

write data in the cache line

Cache 1

Cache 2

Read data in the cache line

write data in the cache line

Read data in the cache line

Cache 3

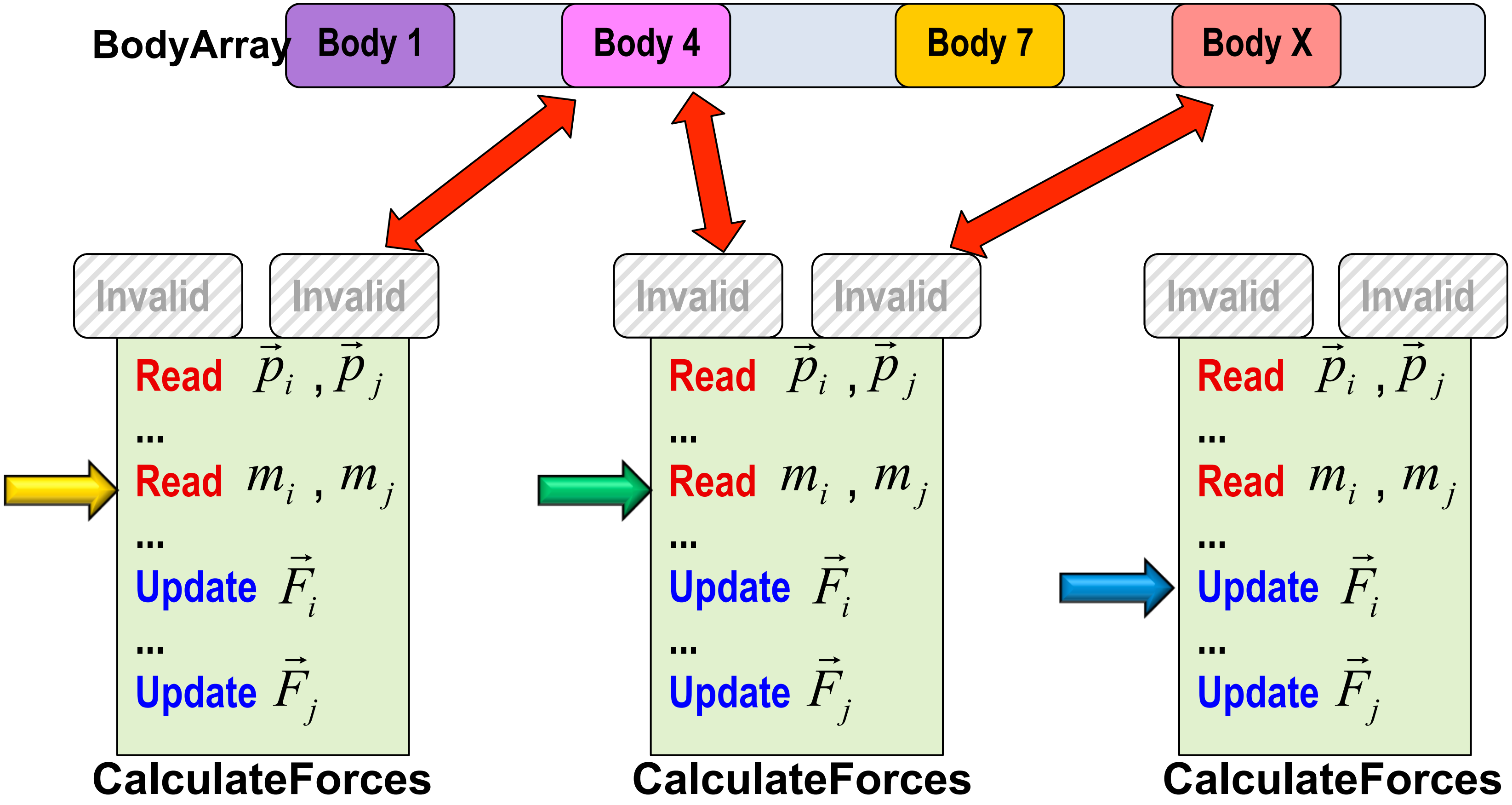
Write the cache line to memory

▨ Invalid 🟩 Exclusive 🟦 Shared 🟥 Modified

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



BodyArray Body 1 Body 4 Body 7 Body X

Invalid Invalid Invalid Invalid Invalid Invalid

Read $\vec{p}_i , \vec{p}_j$

…

Read m_i , m_j

…

Update $\vec{F}_i$

…

Update $\vec{F}_j$

CalculateForces

Read $\vec{p}_i , \vec{p}_j$

…

Read m_i , m_j

…

Update $\vec{F}_i$

…

Update $\vec{F}_j$

CalculateForces

Read $\vec{p}_i , \vec{p}_j$

…

Read m_i , m_j

…

Update $\vec{F}_i$

…

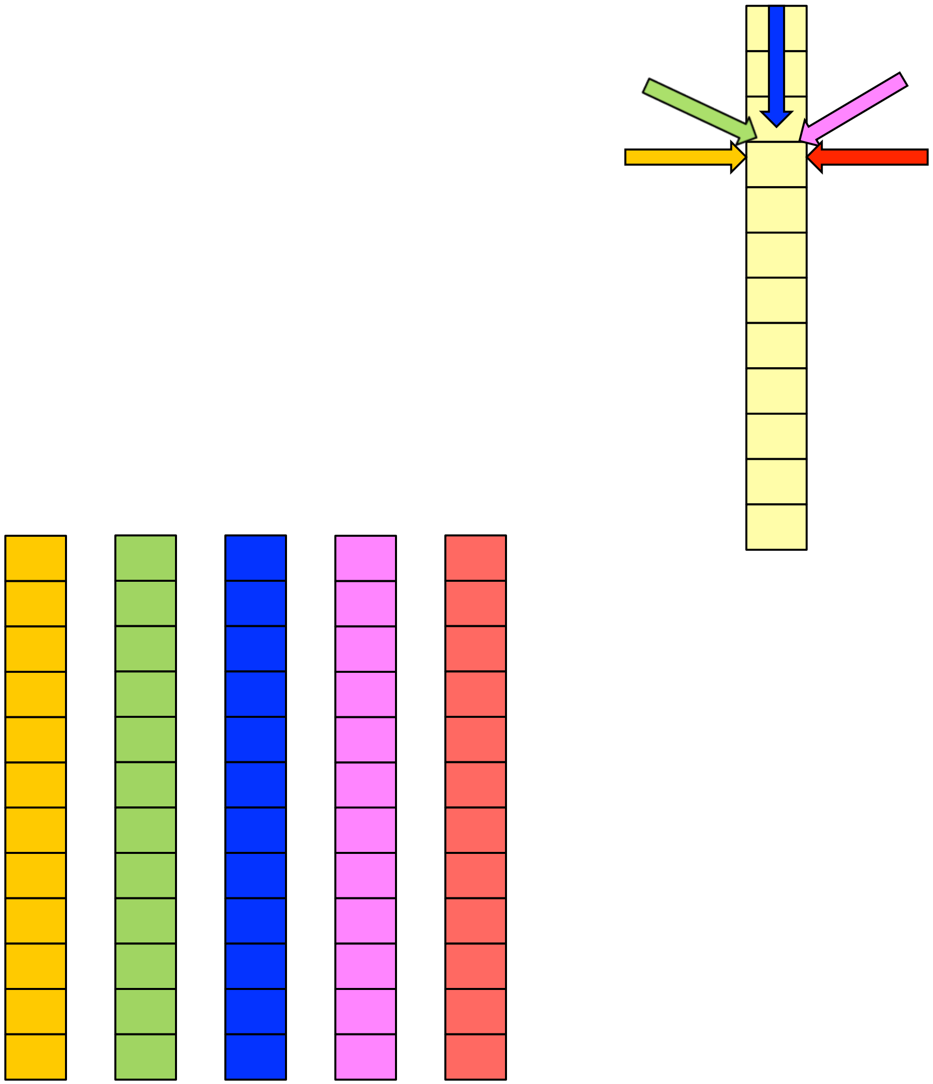
Update $\vec{F}_j$

CalculateForces

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