Exascale-Lab
CEA GENCI INTEL UVSQ

A Decremental Analysis Tool for Fine-Grained Bottleneck Detection

Souad Koliaï[1,2] **Sébastien Valat[1,2]** **Tipp Moseley[3]**

Jean-Thomas Acquaviva[1,2] **William Jalby[1,2]**

[1]University of Versailles Saint-Quentin-en-Yvelines, France

[2]Exatec-Lab, France

[3]Google, Mountain View, CA

Outline

Exatec-Lab

- Introduction: my personal view on hardware performance counters
- DECAN: what?
- DECAN: how?
- Case studies
- Future work

How to deal with performance issues (1)

Exatec-Lab

- First (well known) technique: profiling
  - Down to a few hot routines
  - Then analyze loop behavior
  - Four key issues: source code, compiler, OS, hardware
- Second analyze loops statically (source code, compiler
  - Static analysis (MAQAO)
  - Allows to detect compiler inefficiencies
  - Provides performance estimates and bottleneck analysis
- In general discrepancy between static estimates and measurements
  - What is the next step ??
  - Use performance counters to get an idea of hardware performance behavior

How to deal with performance issues (2)

Exatec-Lab

- Once you know the performance issues, analyze/evaluate them
  - Sort them out by performance impact importance (ROI)
  - Trade off between cost and potential performance gains
- After performance problem analysis, fix performance issues
  - The main "performance knob" at our disposal are instructions
  - Change the source code or assembly to remove performance issues
- Importance of ROI (Return On Investment)
  - Routine A consumes 40% of execution time and performance gains are estimated on routine A at 10%: overall gain 4%
  - Routine B consumes 20% of execution time and performance gains are estimated on routine B at 50%: overall gain 10%

Hardware performance counters/events

Exatec-Lab

In general, performance events give an aggregate view of the routine/loop behavior:

- Number of cache misses
- All of the instructions are "lumped" together: no individual view/report of an instruction
- REMEMBER: our main knob is at instruction level

Conflict on address disambiguation

Exatec-Lab

Consider the C kernel :

```
for (int i = 0 ; i < SIZE ; ++i )
    a[ i ] = b[ i – offset ]
```

If we have addresses such as :

a % 4kB = b % 4kB (same low order 12 bits)

With **offset = 1**, there is a conflict between :

The ***store*** **a[(i)]** from iteration **i**

The ***load*** **b[(i+1) - 1]** from iteration **i+1**

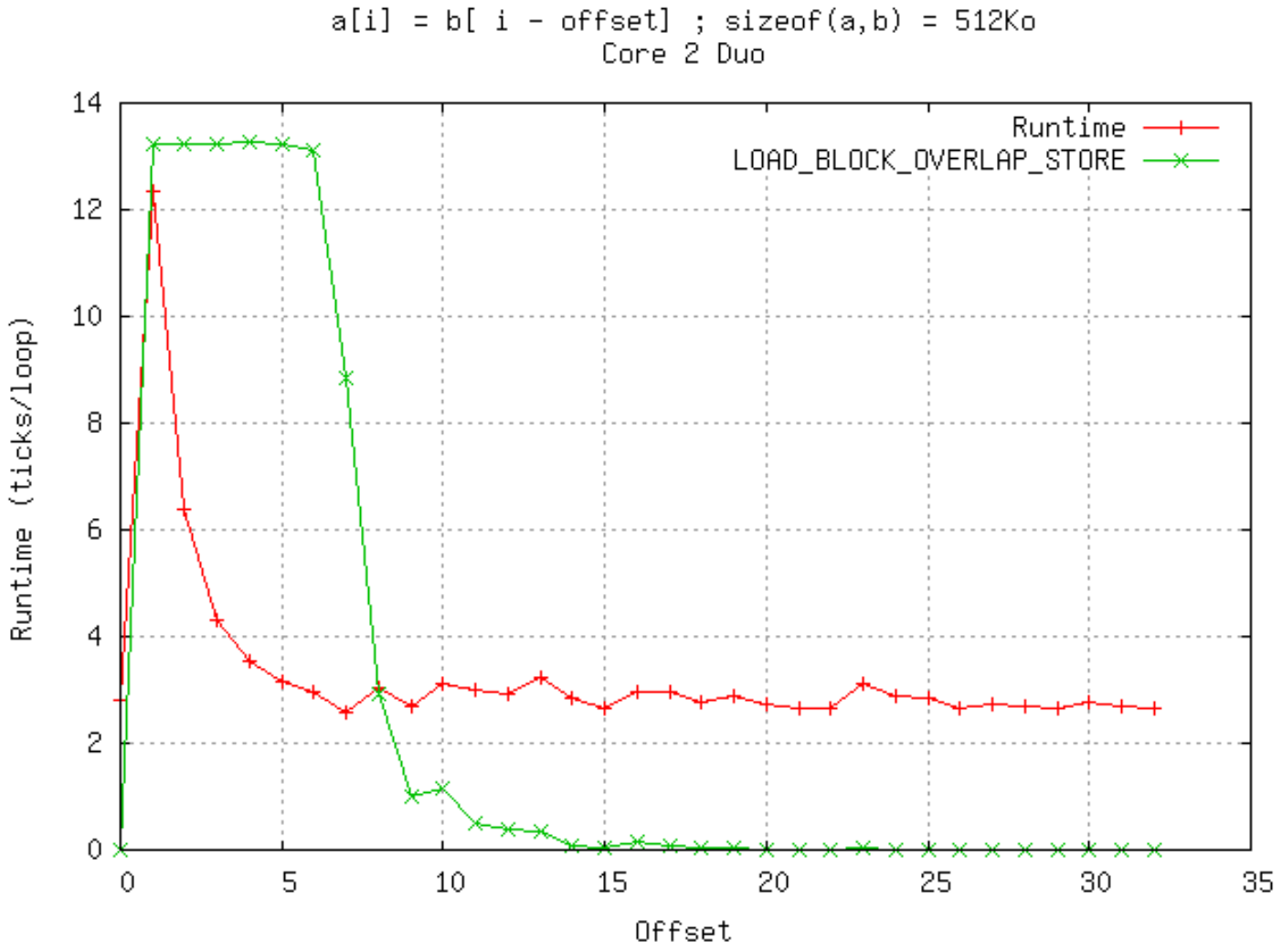
THIS IS KNOWN AS THE 4 KB ALIASING PROBLEM

This can be detected with hardware counter :

LOAD_BLOCK.OVERLAP_STORE

Performance on Intel CORE 2 duo

Exatec-Lab



Results Analysis

Exatec-Lab

Sensible impact up to :

- **Offset = 10** in terms of **counter**
- **Offset = 4** in terms of time **cost**

The counter DETECTS the issue, but not the cost.

WHAT WE CARE ABOUT IS PERFORMANCE IMPACT

Hardware performance counters/events issues (1)

Exatec-Lab

- Detects the source of the problem not the performance impact
  - Counts the number of 4 KB alias conflicts but not the cost
  - Counts the number of cache misses not the latency (except EAR on IA64 and mem lat counter on I7) and in fact you want the exposed latency ☺
- Sampling bias and threshold
  - Quantum measurement: every 100 000 cache misses, update counters
  - In general unable to assign the cost to the right/offending instruction
  - Delays between the counter overflow and the interrupt handler
  - Too many instructions in flight
  - Several instructions retiring at the same time
  - IN CONCLUSION BAD ACCOUNTING: NO GOOD CORRELATION WITH SOURCE CODE

Hardware performance counters/events issues (2)

Exatec-Lab

- Other Key issues with performance counters/events:
  - TOO MANY OF PERFORMANCE EVENTS: Over 1200 on core I7
  - TOO FEW COUNTERS: typically 4, getting values for all events would require 400 runs
  - Deals with low level hardware and gives you a fragmented picture: counts the number of times prefetch are launched including the aborted cases
  - Documentation is usually poor
  - Needs to know very well micro architecture and in general corresponding info is not available
  - Not consistent across architectures even on successive X86 generations

- An interesting OLD idea: Profile me (DEC)
  - Sample instructions
  - Reports all stalls occurring to an instruction

Introduction to DECAN (1)

Exatec-Lab

- Be a physicist:
  - Consider the machine as a black box
  - Send signals in: code fragments
  - Observe/measure signals out: time and maybe other metrics
- Signals in/Signals out
  - Slightly modify incoming signals and observe difference/variations in signals out
  - Tight control on incoming signal
- In coming signal: code
  - Modify source code: easy but dangerous: the compiler is in the way
  - Modify assembly/binary: much finer control but cautious about correlation with source code

Introduction to DECAN (2)

Exatec-Lab

- GOAL 1: detect the offending/delinquent operations
- GOAL 2: get an idea of potential performance gain

DECAN: What?

Exatec-Lab

- A tool for fine grained detection of the bottleneck (ie. assembly instruction level)
- Focus on the hottest region of an application using automatic kernel extraction (AKE)
- DECAN performs on a binary and on loop level
- DECAN uses MAQAO/MADRAS disassembler tool chain

DECAN: General Concept

Exatec-Lab

- DECAN's concept is simple:
  - Measure the original binary
  - Patch the memory access instructions in the original binary
  - New binary is generated for each patch
  - Measure new binaries
  - Measurements are represented in a CSV file: analysis and comparison

DECAN: Automatic Kernel Extraction 1/2

Exatec-Lab

- Strategy for performance measurements: Automatic driver to extract a kernel from a given application
- Goal:
  - focus on only a small part of the application (the hottest subroutine = the kernel)
  - Extract the kernel and its memory context
  - Build a driver to run the kernel in its original execution environment

DECAN: Automatic Kernel Extraction 2/2

Exatec-Lab

- Kernel extraction methodology
  - Dump the memory context of the kernel using GDB
  - Dump the parameters addresses of the kernel using GDB
  - Map the memory context dumped
  - Pass the parameters addresses dumped to the correct registers/stack location → generates a caller to the kernel
  - Original memory context + correct calling convention → operational loader
  - Bypass the main of the original application to branch to the loader → run the kernel in its original execution environment

DECAN: Instruction Removal

Exatec-Lab

- DECAN focuses on SSE memory access instructions (ie. SSE loads and stores)
- Memory access instruction patching:
  - Replace the memory access instruction by a nop operation or a pxor to avoid extra dependencies
  - Example:

```
movaps (%rsi),%xmm1 → nop r/m or pxor %xmm1, %xmm1
movaps %xmm2,(%rsi) → nop r/m
```

- Each patched instruction generates a new binary

DECAN: Instruction Patching

Exatec-Lab

- If n SSE instructions then n+3 different binaries + grouping version of binaries are generated:
  - One_Load binary
  - One_Store binary
  - All_Loads binary
  - All_Stores binary
  - All_Loads_Stores binary
  - Grouping

- Each new binary has the following file name format:
  `<func name> loopID OPT`

`OPT = loads|stores|loads stores|(ld|st) @inst lineSRC`

Example:

`rbgauss loop3 ld 0x402f4c line97` → in loop 3 of rbgauss function, the load instruction at 0x402f4c address has been modified (source line: 97)

DECAN: Performance Measurement

Exatec-Lab

- The original binary and the new binaries are measured using the automatic kernel extraction
- Performance measurements are gathered in a CSV file
- The CSV format allows to make easy the comparison between the original binary and the modified binaries and to pinpoint the delinquent memory access instruction

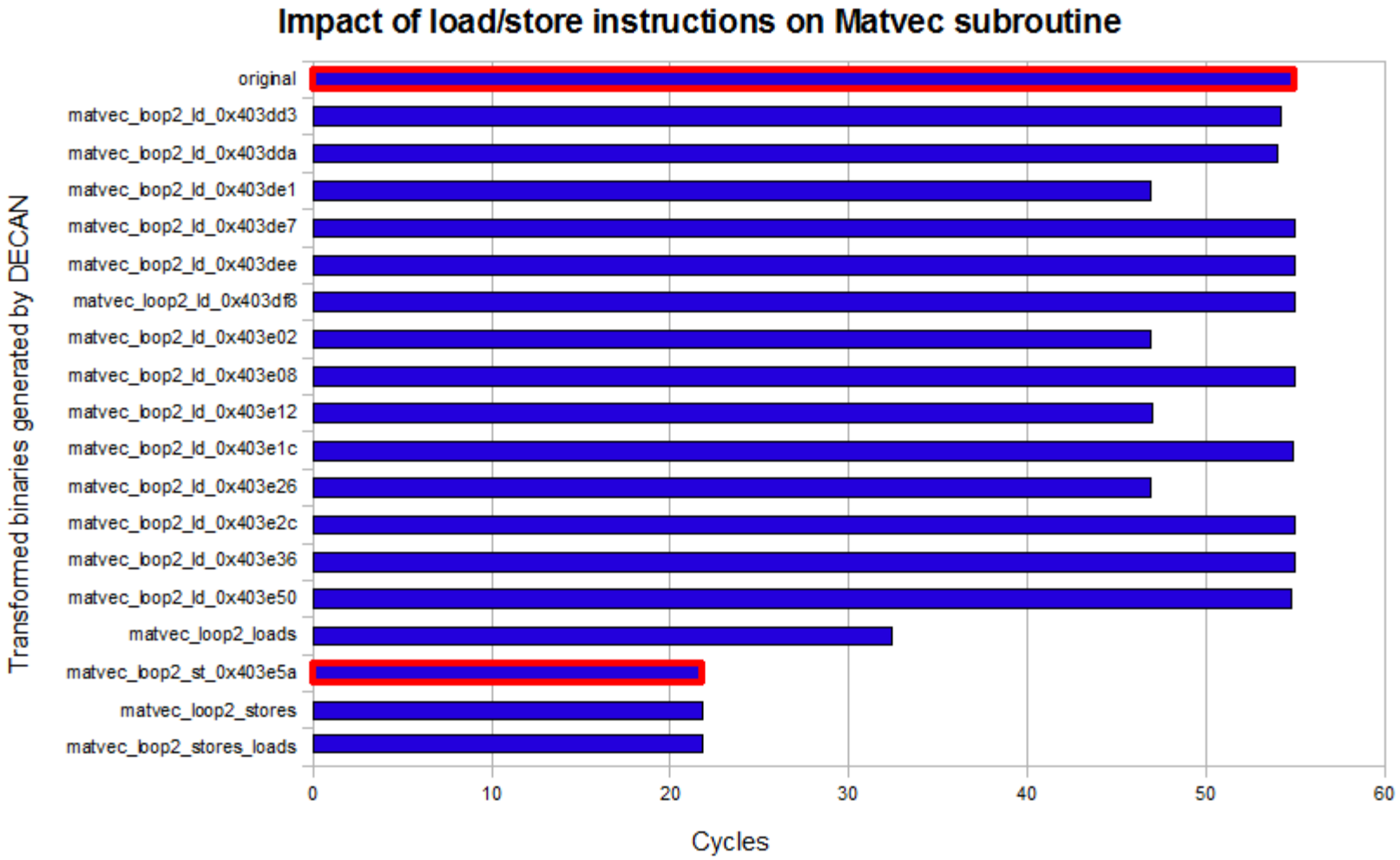
DECAN: Case Studies - MAGMA

Exatec-Lab

- MAGMA is an application for the simulation of casting processes
- The hottest subroutine in MAGMA application is CGSolv
- The target loop in CGSolv is Matvec shown in Fig.3
- Applying DECAN on Matvec generates a set of binaries (when modifying memory access instructions)
- Performance measurements are gathered in MATVEC.csv file

```
do k = anf3, end3
    do j = anf2, end2
        do  i = anf1, end1
            vhilf(i,j,k) = temp(i,j,k) - (
     &              ( acx(i-1,j   ,k  )  * temp(i-1 ,j  ,k    )
     &            + acx(i   ,j   ,k   ) * temp(i+1,j  ,k    )
     &            + acy(i   ,j-1,k   ) * temp(i   ,j-1,k    )
     &            + acy(i   ,j   ,k   ) * temp(i   ,j+1,k   )
     &            + acz(i   ,j   ,k-1) * temp(i   ,j    ,k-1)
     &            + acz(i   ,j   ,k   ) * temp(i   ,j    ,k+1))
     &     ) / coeffd(i,j,k)
        end do
    end do
end do
```

Fig. 3. Target Loop in CGSolv


Impact of load/store instructions on Matvec subroutine
Transformed binaries generated by DECAN
original
matvec_loop2_ld_0x403dd3
matvec_loop2_ld_0x403dda
matvec_loop2_ld_0x403de1
matvec_loop2_ld_0x403de7
matvec_loop2_ld_0x403dee
matvec_loop2_ld_0x403df8
matvec_loop2_ld_0x403e02
matvec_loop2_ld_0x403e08
matvec_loop2_ld_0x403e12
matvec_loop2_ld_0x403e1c
matvec_loop2_ld_0x403e26
matvec_loop2_ld_0x403e2c
matvec_loop2_ld_0x403e36
matvec_loop2_ld_0x403e50
matvec_loop2_loads
matvec_loop2_st_0x403e5a
matvec_loop2_stores
matvec_loop2_stores_loads
0
10
20
30
40
50
60
Cycles

DECAN: Case Studies - MAGMA

Exatec-Lab

- When replacing one load at the same time, there is some performance impact of the replaced load : however some loads have a larger impact than others
- When replacing all loads, performance is improved by a factor of 2.5
- When replacing A SINGLE store, performance is improved by a factor of 2.5 → this store seems to be the bottleneck.
- Conclusion: the conflict between the loads and a store seems to be the bottleneck !
- A 4K-aliasing load-store conflict between vhilf (the array being stored), temp and acx (the arrays being loaded).

DECAN: Case Studies - RECOM

Exatec-Lab

- RECOM application builds a 3D-model of industrial-scale furnaces.
- The hottest subroutine in Recom application is RBgauss
- The target loop in RBgauss is shown in Fig.1
- 3D structures (arrays, loops) are linearized
- Regular geometry but with holes: use of indirect access to jump over holes
- RB stands for Red Black: many access are stride 2

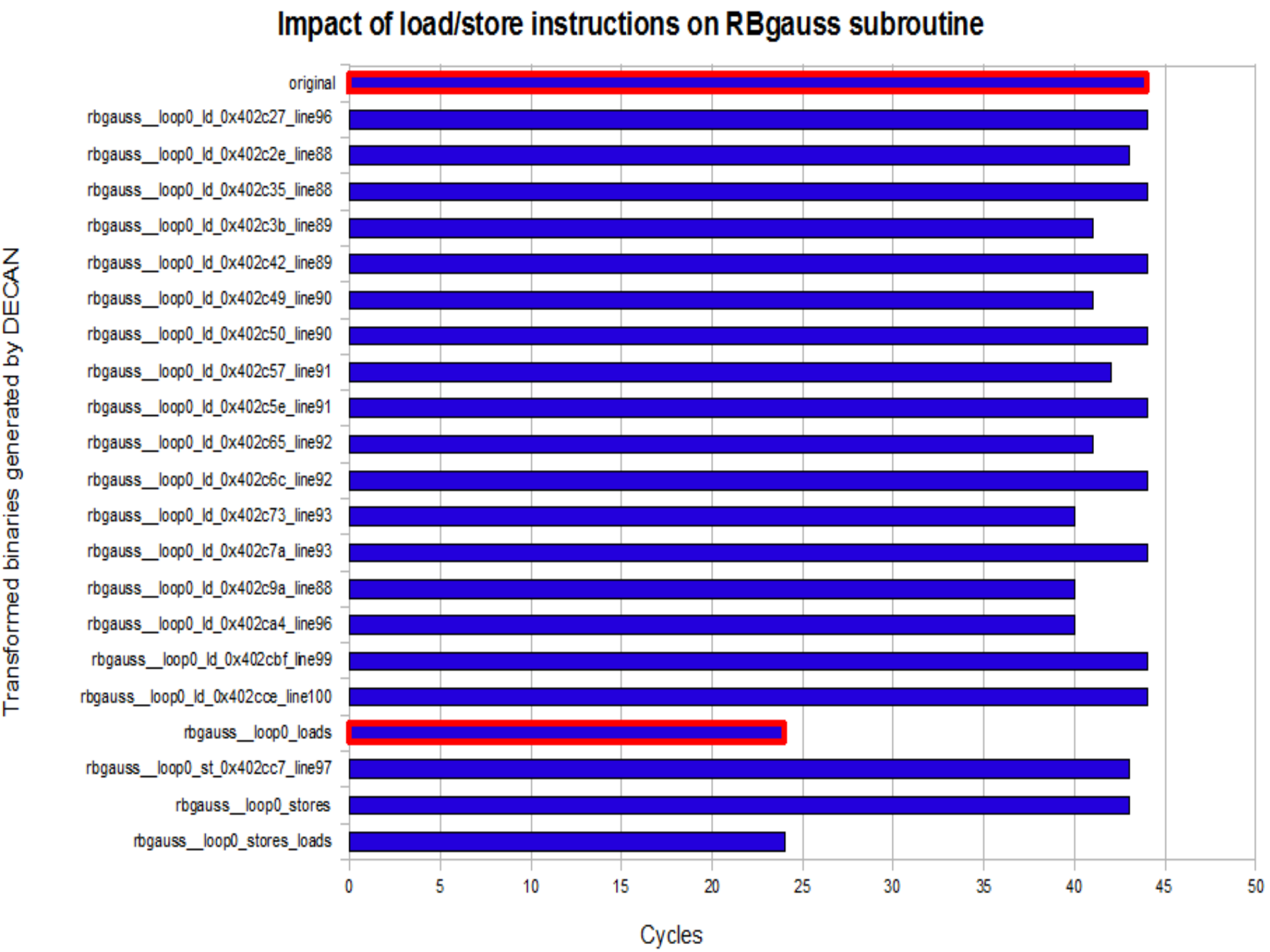
```
DO IDO=1,NREDD
 INC    =    INDINR(IDO)

 HANB   =    AM(INC,1)*PHI(INC+1)   &
           + AM(INC,2)*PHI(INC-1)   &
           + AM(INC,3)*PHI(INC+INPD)   &
           + AM(INC,4)*PHI(INC-INPD)   &
           + AM(INC,5)*PHI(INC+NIJ)   &
           + AM(INC,6)*PHI(INC-NIJ) &
           + SU(INC)

 DLTPHI = UREL*( HANB/AM(INC,7) - PHI(INC) )
 PHI(INC) = PHI(INC) + DLTPHI

 RESI = RESI + ABS(DLTPHI)
 RSUM = RSUM + ABS(PHI(INC))
ENDDO
```

Fig. 1. Target Loop in RBgauss

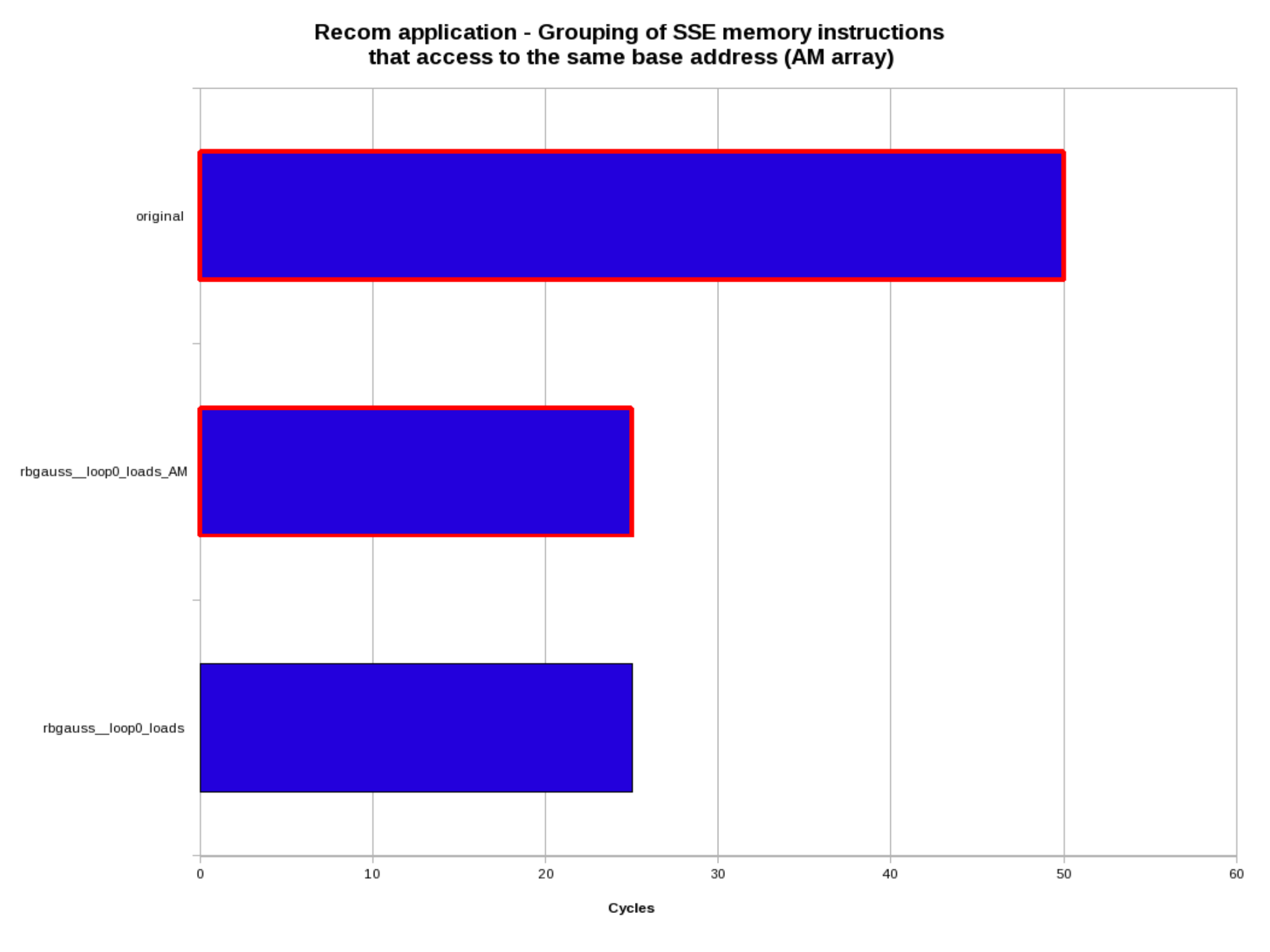

Impact of load/store instructions on RBgauss subroutine
Transformed binaries generated by DECAN
original
rbgauss__loop0_ld_0x402c27_line96
rbgauss__loop0_ld_0x402c2e_line88
rbgauss__loop0_ld_0x402c35_line88
rbgauss__loop0_ld_0x402c3b_line89
rbgauss__loop0_ld_0x402c42_line89
rbgauss__loop0_ld_0x402c49_line90
rbgauss__loop0_ld_0x402c50_line90
rbgauss__loop0_ld_0x402c57_line91
rbgauss__loop0_ld_0x402c5e_line91
rbgauss__loop0_ld_0x402c65_line92
rbgauss__loop0_ld_0x402c6c_line92
rbgauss__loop0_ld_0x402c73_line93
rbgauss__loop0_ld_0x402c7a_line93
rbgauss__loop0_ld_0x402c9a_line88
rbgauss__loop0_ld_0x402ca4_line96
rbgauss__loop0_ld_0x402cbf_line99
rbgauss__loop0_ld_0x402cce_line100
rbgauss__loop0_loads
rbgauss__loop0_st_0x402cc7_line97
rbgauss__loop0_stores
rbgauss__loop0_stores_loads
0
5
10
15
20
25
30
35
40
45
50
Cycles

DECAN refinement: instruction grouping

Exatec-Lab

- An example:
  - B(i) = A(I) + A(I+1)
  - Let us assume A coming from memory: 1 miss followed by a hit
  - Nopping A(I) generates one miss A(I+1)
  - Nopping A(I+1) generates one miss on A(I)

- Basic idea of grouping
  - Group together loads which are dependant upon each other
  - Group loads accessing the same array
- How to implement grouping
  - Analyze start array address
  - Group together loads which corresponds to “close” start array address


Recom application - Grouping of SSE memory instructions
that access to the same base address (AM array)
original
rbgauss__loop0_loads_AM
rbgauss__loop0_loads
0
10
20
30
40
50
60
Cycles

DECAN: Case Studies - RECOM

Exatec-Lab

- When nooping one load at the same time, there is limited effect of the nopped load.
- When replacing all loads, performance is improved by a factor of 1.75
- Grouping shows that most of the performance loss is associated with **access to a 1D array** : AM
- Conclusion: AM access seems to be the bottleneck !
- A memory trace tool is used to detect how AM is accessed
- AM is accessed with a STRIDE 2 !: solution: restructure splitting AM into two distinct arrays: one for the RED, one for the BLACK

DECAN: Case Studies - RECOM

Exatec-Lab

- Limiting array restructuring to AM is much simpler: read only structure
- Restructuring PHI is much harder: complex access and read/write operations

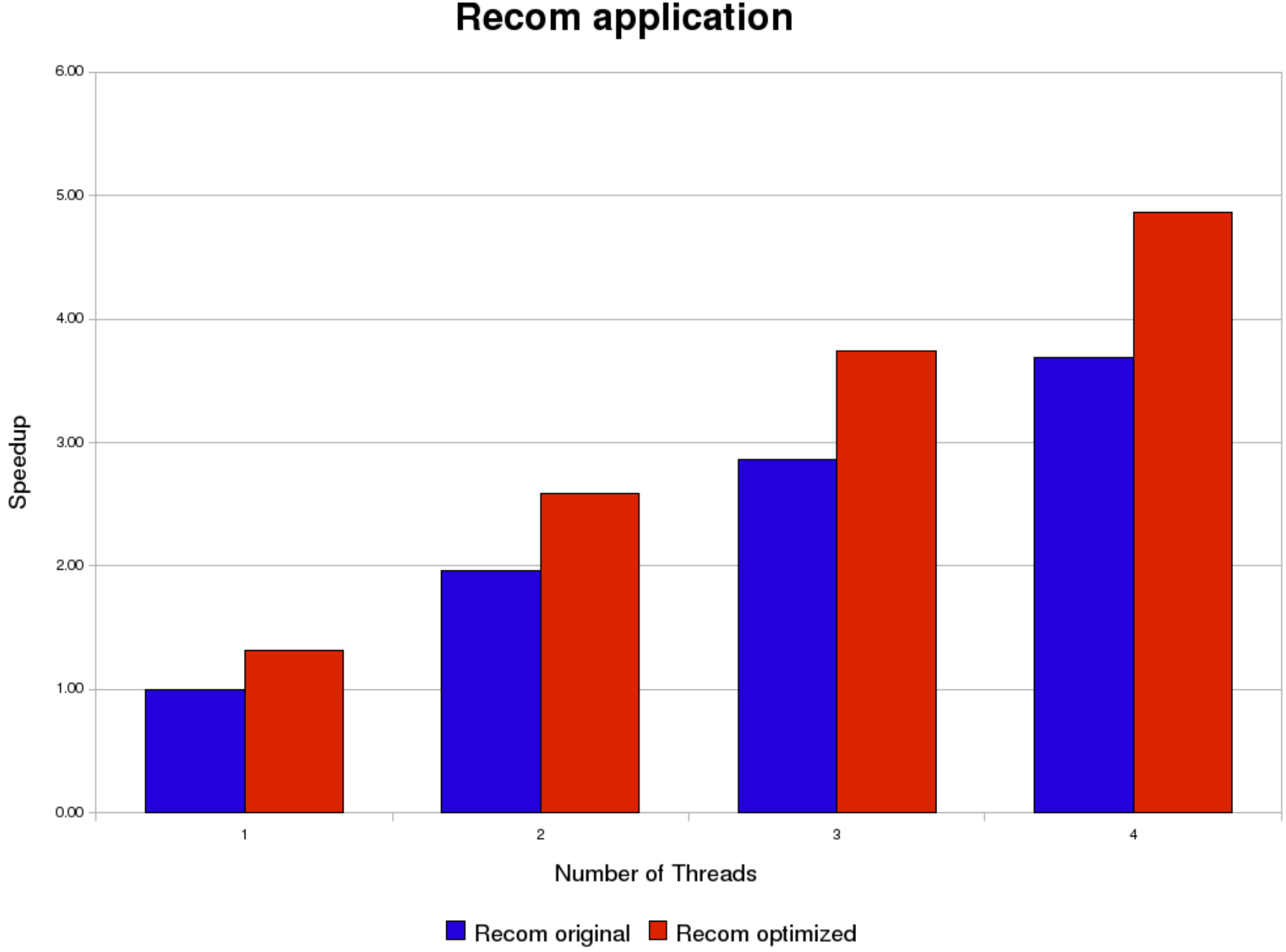
```
DO IDO=1,NREDD
 INC    =    INDINR(IDO)

 HANB   =    AM(INC,1)*PHI(INC+1)   &
           + AM(INC,2)*PHI(INC-1)   &
           + AM(INC,3)*PHI(INC+INPD)   &
           + AM(INC,4)*PHI(INC-INPD)   &
           + AM(INC,5)*PHI(INC+NIJ)   &
           + AM(INC,6)*PHI(INC-NIJ)  &
           + SU(INC)

 DLTPHI = UREL*( HANB/AM(INC,7) - PHI(INC) )
 PHI(INC) = PHI(INC) + DLTPHI

 RESI = RESI + ABS(DLTPHI)
 RSUM = RSUM + ABS(PHI(INC))
ENDDO
```

Fig. 1. Target Loop in RBgauss


Recom application
6.00
5.00
4.00
3.00
2.00
1.00
0.00
Speedup
1
2
3
4
Number of Threads
Recom original
Recom optimized

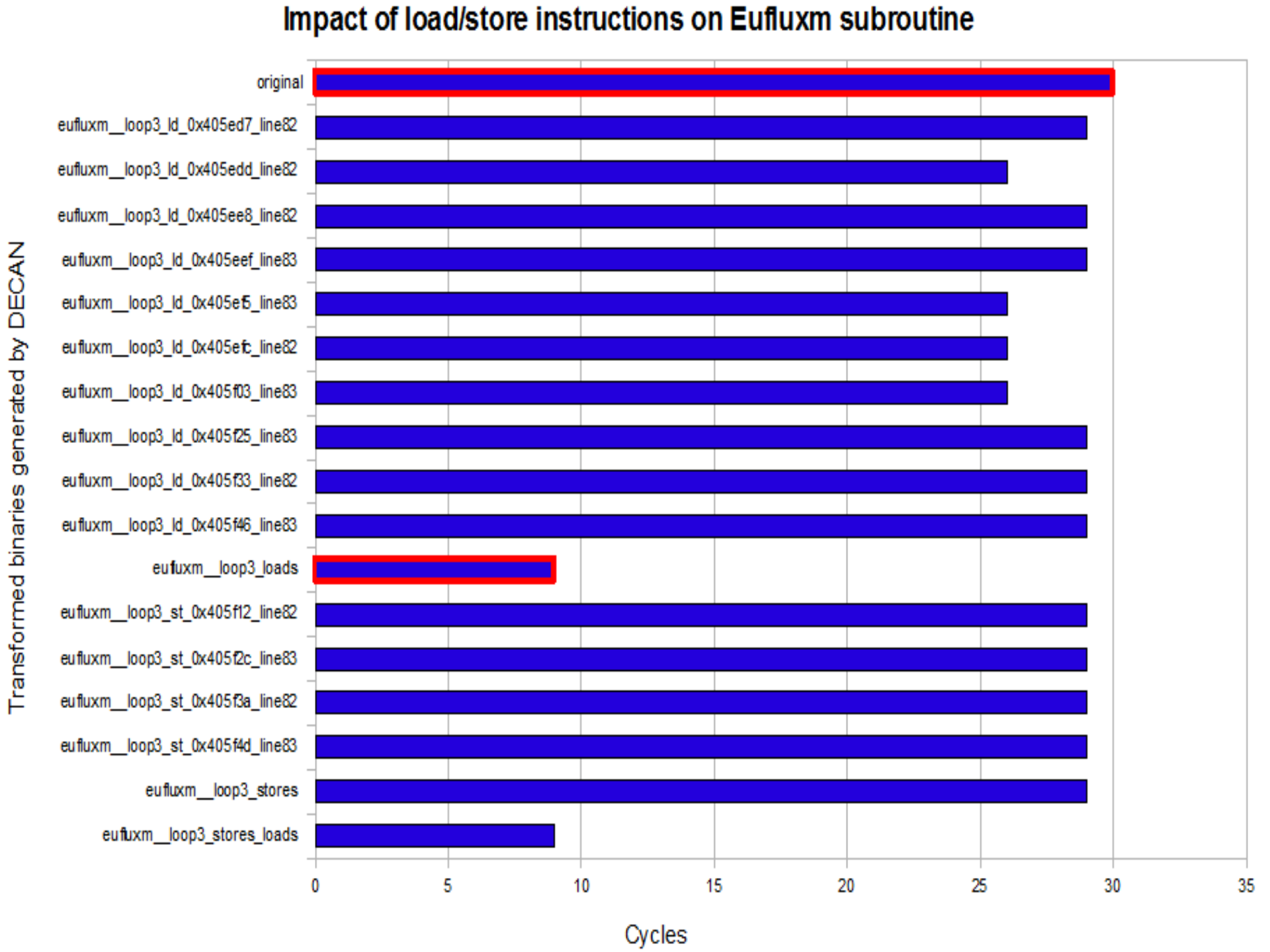
DECAN: Case Studies - DASSAULT

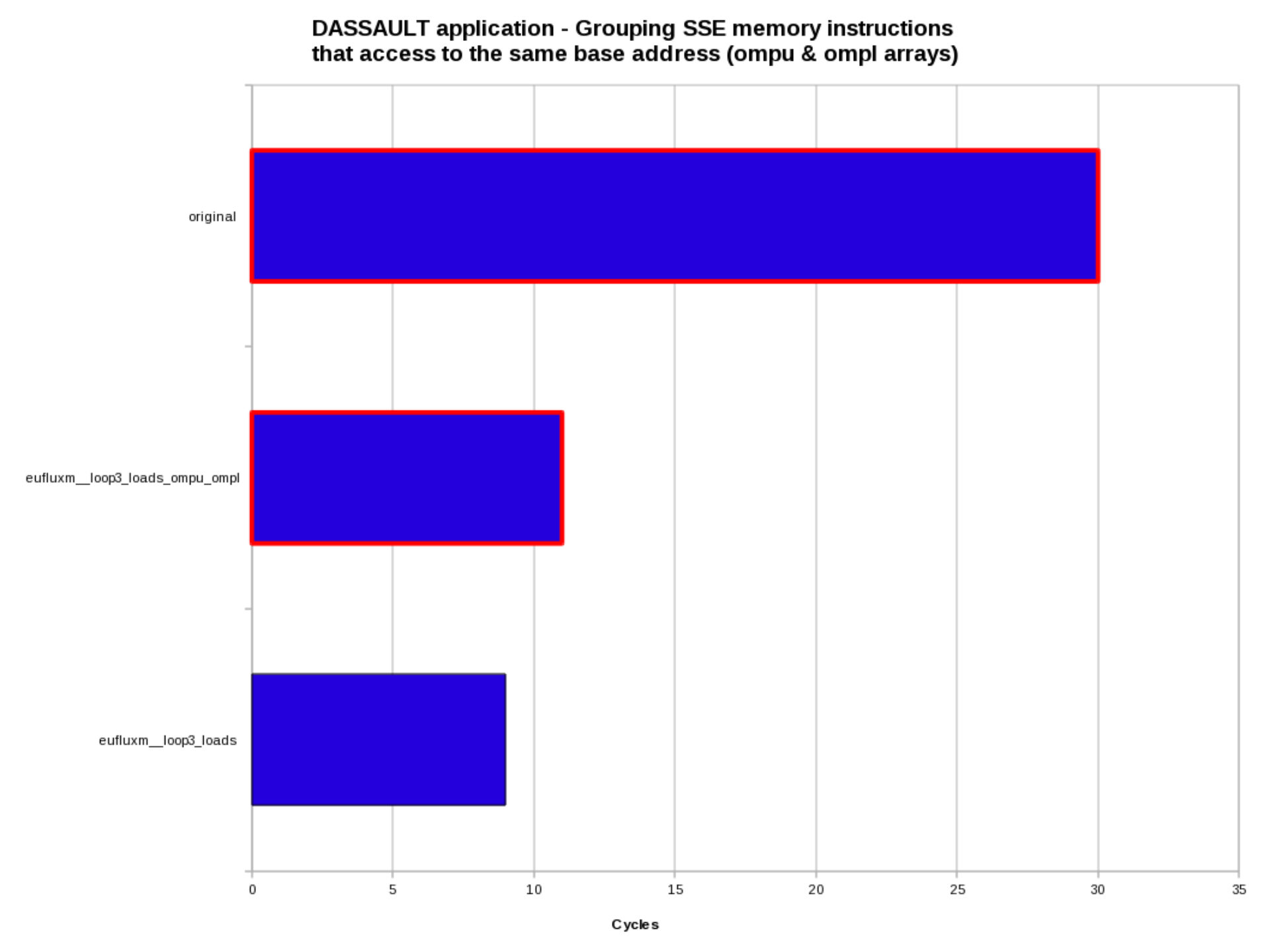
Exatec-Lab

- DASSAULT application solves the Navier-Stokes equation using computational fluid dynamics based on an iterative solver
- The hottest subroutine in Dassault application is Eufluxm
- The target loop in Eufluxm is shown in Fig.2
- Bad access (strides) to arrays

```
      do cb=1,ncbt
        igp = isg
        isg = icolb(icb+1)
        igt = isg + igp
c$OMP  PARALLEL DO DEFAULT(NONE)
c$OMP& SHARED(igt,igp,nnbar,vecy,vecx,ompu,ompl)
c$OMP& PRIVATE(ig,e,i,j,k,l)
        do ig=1,igt
          e = ig + igp
          i = nnbar(e,1)
          j = nnbar(e,2)
cDEC$ IVDEP
          do k=1,ndof
cDEC$ IVDEP
            do l=1,ndof
              vecy(i,k) = vecy(i,k) + ompu(e,k,l)*vecx(j,l)
              vecy(j,k) = vecy(j,k) + ompl(e,k,l)*vecx(i,l)
            enddo
          enddo
        enddo
      enddo
```

Fig. 2. Target Loop in Eufluxm

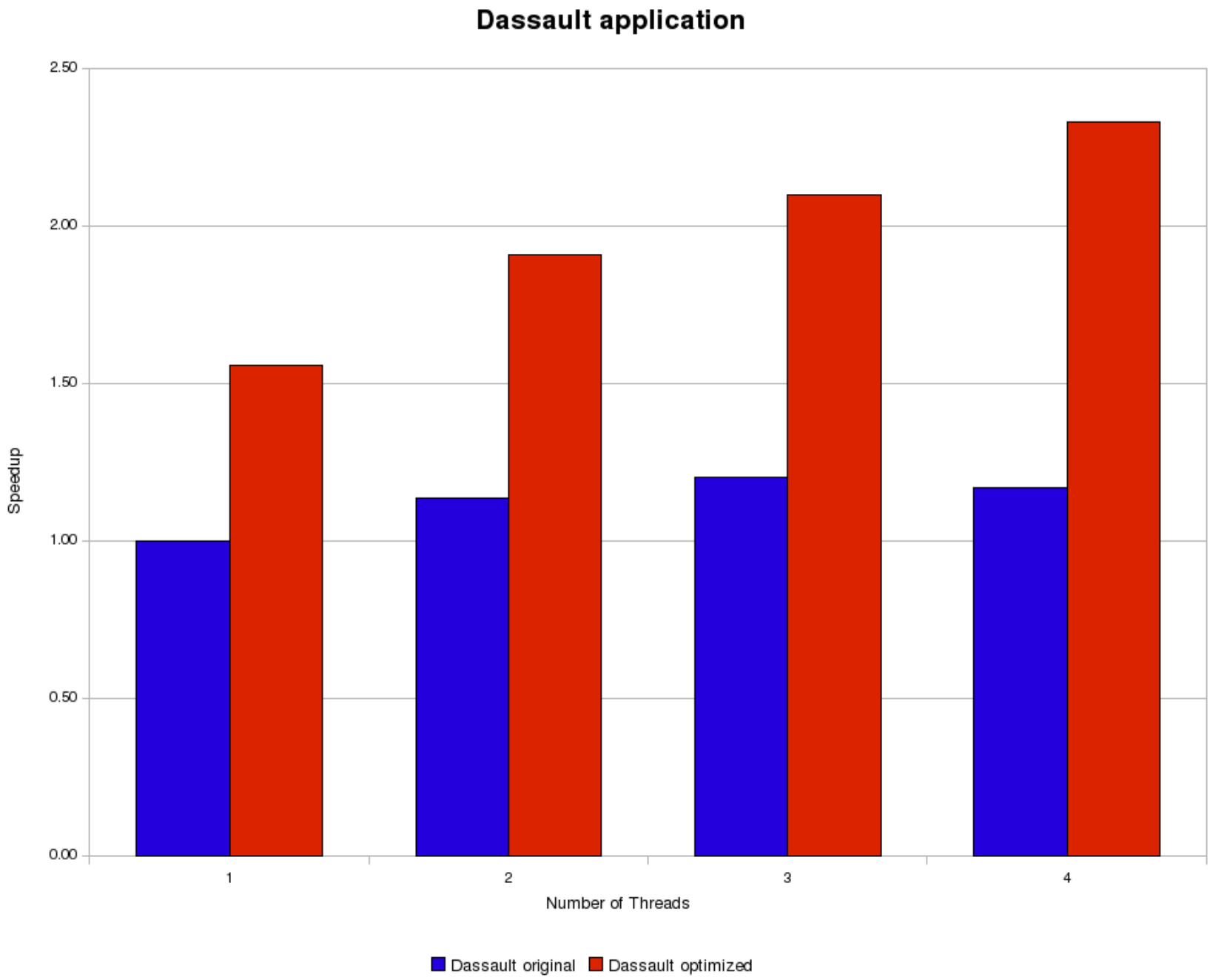

Impact of load/store instructions on Eufluxm subroutine
Transformed binaries generated by DECAN
original
eufluxm__loop3_ld_0x405ed7_line82
eufluxm__loop3_ld_0x405edd_line82
eufluxm__loop3_ld_0x405ee8_line82
eufluxm__loop3_ld_0x405eef_line83
eufluxm__loop3_ld_0x405ef5_line83
eufluxm__loop3_ld_0x405efc_line82
eufluxm__loop3_ld_0x405f03_line83
eufluxm__loop3_ld_0x405f25_line83
eufluxm__loop3_ld_0x405f33_line82
eufluxm__loop3_ld_0x405f46_line83
eufluxm__loop3_loads
eufluxm__loop3_st_0x405f12_line82
eufluxm__loop3_st_0x405f2c_line83
eufluxm__loop3_st_0x405f3a_line82
eufluxm__loop3_st_0x405f4d_line83
eufluxm__loop3_stores
eufluxm__loop3_stores_loads
0
5
10
15
20
25
30
35
Cycles


DASSAULT application - Grouping SSE memory instructions
that access to the same base address (ompu & ompl arrays)
original
eufluxm__loop3_loads_ompu_ompl
eufluxm__loop3_loads
0
5
10
15
20
25
30
35
Cycles

DECAN: Case Studies - DASSAULT

Exatec-Lab

- When replacing one load at the same time, there is no effect of the replaced load.
- When replacing all loads, performance is improved by a factor of 3 → some "dependent" loads seem to be the bottleneck.
- Grouping shows that most of the performance loss is due to **access to two 3D arrays** : ompu & ompl
- Conclusion: ompu & ompl access seems to be the bottleneck !
- A memory trace tool is used to detect how these arrays are accessed: Ompl & ompu are accessed with a LARGE STRIDE ! (iterating on the wrong dimension)
- Only ompu and ompl need to be restructured


Dassault application
2.50
2.00
1.50
1.00
0.50
0.00
Speedup
1
2
3
4
Number of Threads
Dassault original
Dassault optimized

Increasing DECAN functionalities (1)

Exatec-Lab

- Compare performance impact with microbenchmark results
  - Use to detect/guess operand location: L1, L2, L3, RAM
  - Use to evaluate prefetch efficiency
- Go beyond nopping:
  - Instead of a NOP use a register move (pay attention to dependencies)
  - Instead of a NOP, perform an access to a given (invariant memory location on the stack (keep cache access latency impact)
- NOP other instructions than memory operations
  - Arithmetic complex instructions: divide, square root
  - Analyze impact of out of order

Increasing DECAN functionalities (2)

Exatec-Lab

- NOP branches
  - Two variants: force fall through or taken branch
  - Analyze impact of branch misprediction
- Detection of multicore issues:
  - Detection of false sharing
  - Detection of contention

DECAN limitations (1)

Exatec-Lab

- Dealing with side effects:
  - "Nopping" instructions is not exactly neutral
  - Large set of experiments allows to "recoup"
- SEMANTICS is lost
  - From a performance point of view, limited importance but pay attention to some corner cases
  - Some experiments in the DECAN series can crash: for example NOP the access to indirection vectors
- Dealing with If within loop bodies
  - Typical case: if (A(I)) > 0) THEN …. ELSE
  - Nopping A(I) is equivalent to Nopping the branch
  - DECAN provides info but care has to be taken

DECAN limitations (2)

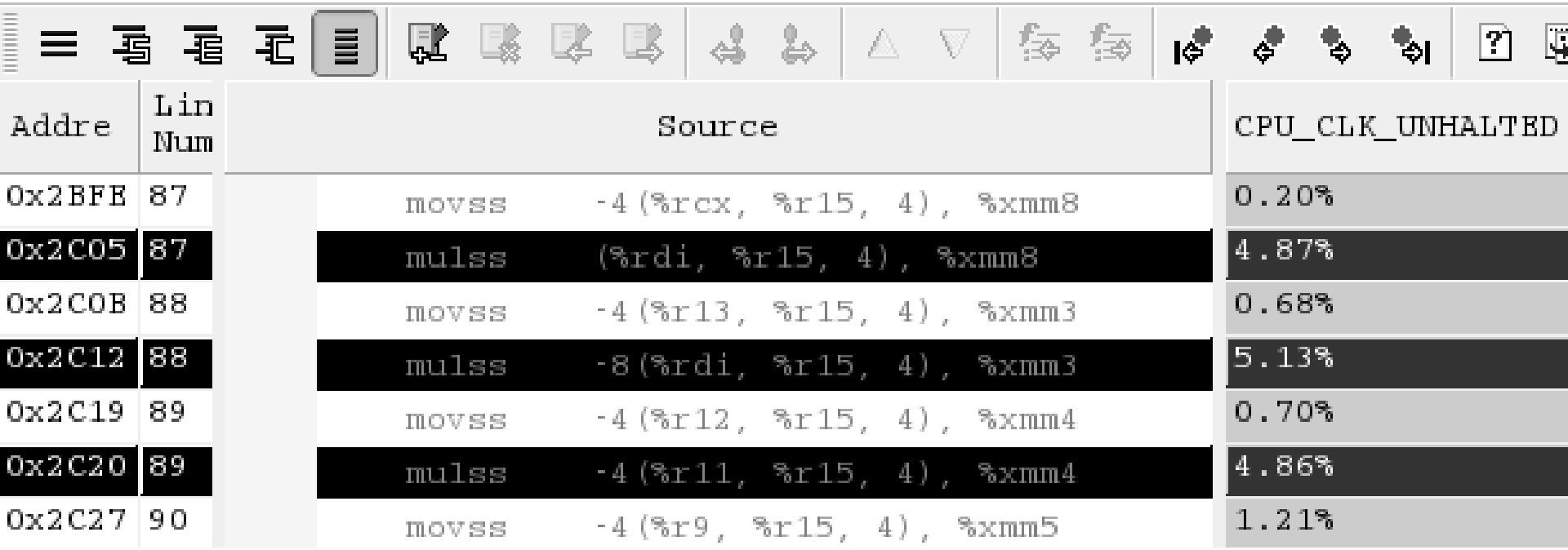
Exatec-Lab

- DECAN is a microscope: applicable to loops only
  - Needs to be coupled with good profiling
- Measurement accuracy
  - Let us think of a loop with 200 vector loads,
  - Some experiments in the DECAN series can crash: for example NOP the access to indirection vectors

DECAN Vs VTune

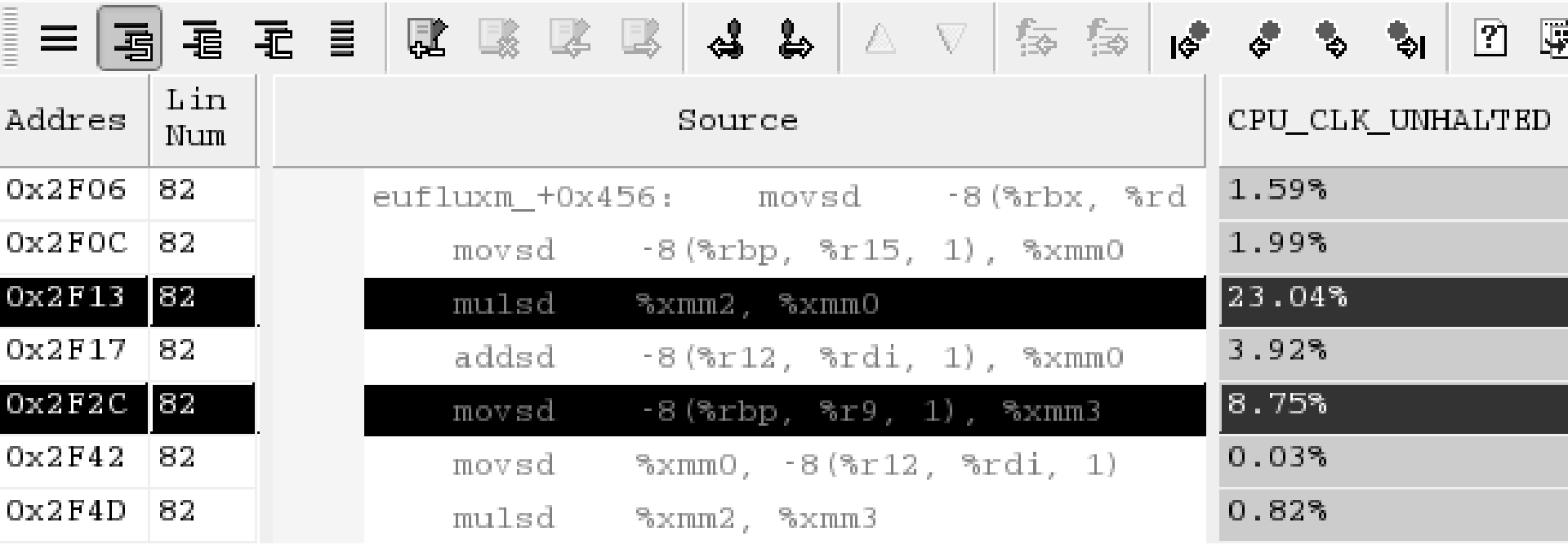
Exatec-Lab

- VTune is an event-based sampling tool that uses hardware counters
- VTune collects data from processor using timer interrupts
- RBgauss and EUFLUXm routines are profiled with VTune (Fig. 1 & Fig. 2)
- VTune detect a large set of instructions that are not all delinquent
- This inaccuracy is inherent to any sampling scheme
- Sampling is useful for a broad diagnostic when DECAN gives a more precise bottleneck detection



Addre	Lin Num	Source	CPU_CLK_UNHALTED
0x2BFE	87	movss -4(%rcx, %r15, 4), %xmm8	0.20%
0x2C05	87	mulss (%rdi, %r15, 4), %xmm8	4.87%
0x2C0B	88	movss -4(%r13, %r15, 4), %xmm3	0.68%
0x2C12	88	mulss -8(%rdi, %r15, 4), %xmm3	5.13%
0x2C19	89	movss -4(%r12, %r15, 4), %xmm4	0.70%
0x2C20	89	mulss -4(%r11, %r15, 4), %xmm4	4.86%
0x2C27	90	movss -4(%r9, %r15, 4), %xmm5	1.21%

Fig. 1. RBgauss profiled with VTune



Addres	Lin Num	Source	CPU_CLK_UNHALTED
0x2F06	82	eufluxm_+0x456: movsd -8(%rbx, %rd	1.59%
0x2F0C	82	movsd -8(%rbp, %r15, 1), %xmm0	1.99%
0x2F13	82	mulsd %xmm2, %xmm0	23.04%
0x2F17	82	addsd -8(%r12, %rdi, 1), %xmm0	3.92%
0x2F2C	82	movsd -8(%rbp, %r9, 1), %xmm3	8.75%
0x2F42	82	movsd %xmm0, -8(%r12, %rdi, 1)	0.03%
0x2F4D	82	mulsd %xmm2, %xmm3	0.82%

Fig. 2. EUFLUXm profiled with VTune

Conclusion & Future Work

Exatec-Lab

- DECAN: a tool for automatic decremental performance analysis.
- DECAN identifies delinquent memory operations
- DECAN gets an estimate of potential performance gain
- Test DECAN on more applications
- Improve user feedback: synthesis of DECAN results
- Extend DECAN to address branch instructions to detect miss-prediction

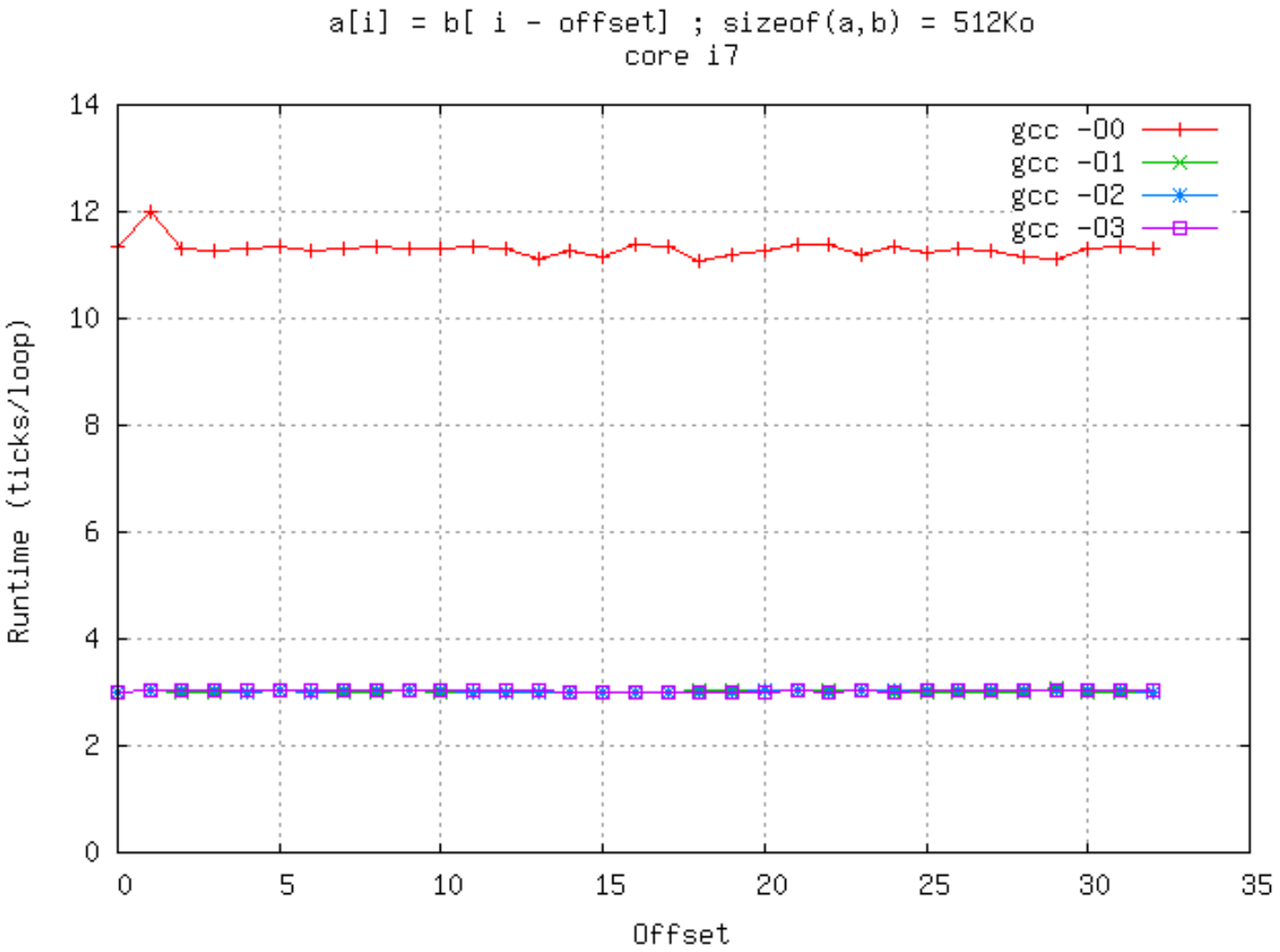
ANNOUNCEMENT

Exatec-Lab

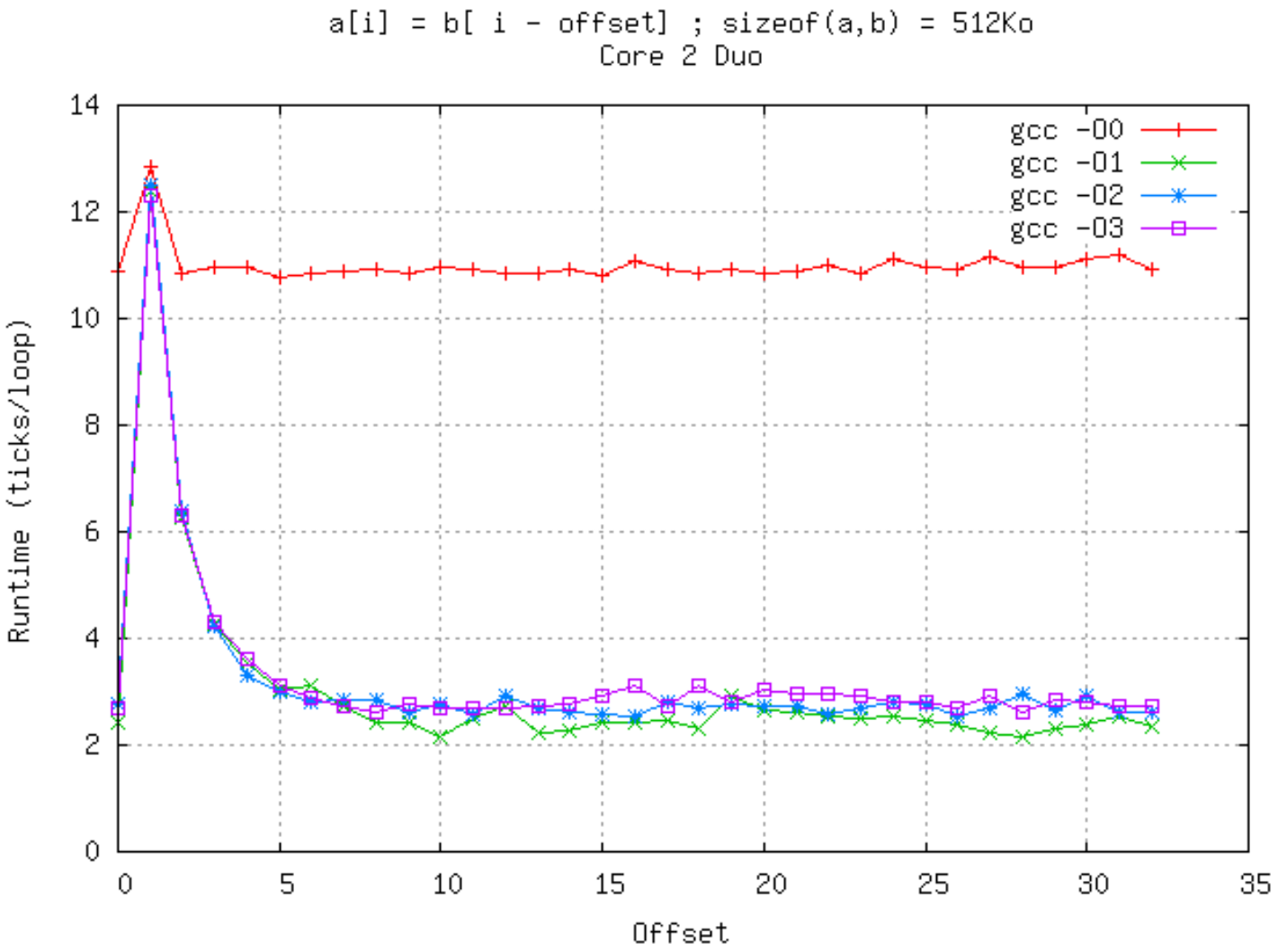
- EXATEC LAB grand opening will take place on October 25th at UVSQ in Versailles
- You are all invited and welcome!!
- See http://www.uvsq.fr : front page

Questions ?

Testing on Intel Core i7



All optimization on Intel Core 2 Duo



Introduction

- Optimization process:
  - Gathering data (ie. code characterization)
  - Diagnosing the problem
  - Prescribing a solution
- Tedious process
  - Complex modern processors
  - Limited existing methodologies
  - Performance counters not up to the job
- Characterization process
  - Code analysis to extract code characteristics
  - Applying different types of code analysis
  - Get different views of the code behavior