Generalization of the Decremental Performance Analysis to Differential Analysis

Zakaria Bendifallah

advisors
Pr. William jalby
Jean-Thomas Acquaviva

UVSQ - Perfcloud, Exascale Computing Research

Application performance analysis

Application performance analysis is becoming a difficult art !

application characterization → bottleneck detection → improve performance

- Complex software: thousands of lines and several programing paradigms
- Multiple granularities: cluster level, node level, core level
- Wide range of analysis tools and techniques with different accuracies and overheads

Bottleneck detection

In general

- Detect if a performance pathology limits performance
- Done in two phases
  detect the performance pathology → determine is the pathology is a bottleneck

Fine grain bottleneck detection

- Done at the node level (processor, core) and deals with processor complexity
  - Out of order execution
  - Complex memory sub-system

Bottleneck detection: Hardware counters

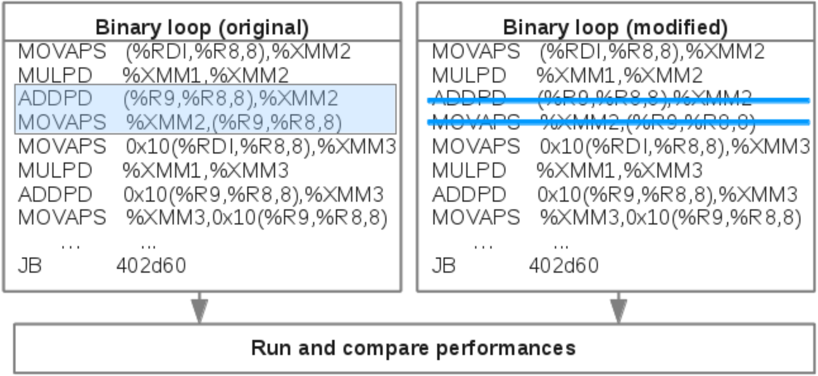
A set of counters that can monitor various hardware generated events

Issues

- Not the same between different micro-architectures
- Good in estimating only quantity not cost
- Difficulty to correlate with source code

A promising technique: Decremental Analysis

A measurement technique based on modification of the program



A promising technique: Decremental Analysis

Advantages

- Accurate pinpointing of delinquent instructions
- Associates a cost to a group of instructions
- Good correlation to binary source code

Technical choices

- Binary level analysis (binary patching tool DECAN)
  Source code → Compiler → Assembly code
- Loop centric (innermost loops)

A promising technique: Decremental Analysis

Limitations

- Simple view of a pathology (= instruction)
- Simple transformation process (no flexibility)
- Poor handling of semantic loss (In-vitro)
- Sequential codes only

Contributions

- Design, test and validate new techniques and use cases to Decremental Analysis
- More sophisticated transformation process
- Extend and fine tune the technical part:
  - Side effects management
  - Parallel codes support
  - Precise measurements process
- Integrate DECAN into an analysis methodology (PAMDA)

Outline

Overview

Differential Analysis

- Continuity of Decremental Analysis
  - More elaborate analyses
  - More advanced transformation process
- Relies and extends the same binary patching tool: DECAN

Terminology

Loop variant

A version of the loop in which assembly instructions have been modified.

DECAN variant

The binary resulting from the process of loop variant creation

Loop variant creation

Identify instruction subsets → Construct transformations requests

Examples of Instruction subsets

- Load & store
- FP arithmetic
- Division
- Reduction

Examples of transformations

- Deletion
- Replacement
- Modification

Memory and arithmetic streams analysis - LS/FP

LS variant

Arithmetic operations are deleted

```
MOVAPS   %XMM2,(%R9,%R8,8)
MOVAPS   0x10(%RDI,%R8,8),%XMM3
MULPD    %XMM1,%XMM3
ADDPD    0x10(%R9,%R8,8),%XMM3
```

```
MOVAPS   %XMM2,(%R9,%R8,8)
MOVAPS   0x10(%RDI,%R8,8),%XMM3

MOVAPS   0x10(%R9,%R8,8),%XMM3
```

FP variant

memory operations are deleted

```
MOVAPS   %XMM2,(%R9,%R8,8)
MOVAPS   0x10(%RDI,%R8,8),%XMM3
MULPD    %XMM1,%XMM3
ADDPD    0x10(%R9,%R8,8),%XMM3
```

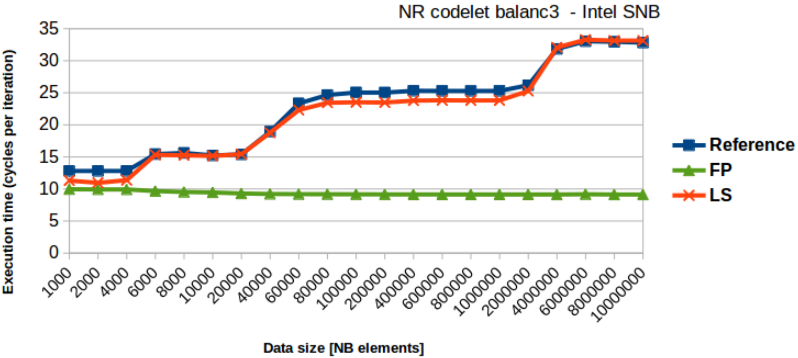
```


MULPD    %XMM1,%XMM3
ADDPD    %XMM3,%XMM3
```

Effect

- CPU and memory sub-system behaviours highlighted independently

Memory and arithmetic streams analysis - LS/FP



Memory operations investigation - DL1

DL1 variant

Replace a memory access to a data structure by an access to a single memory location

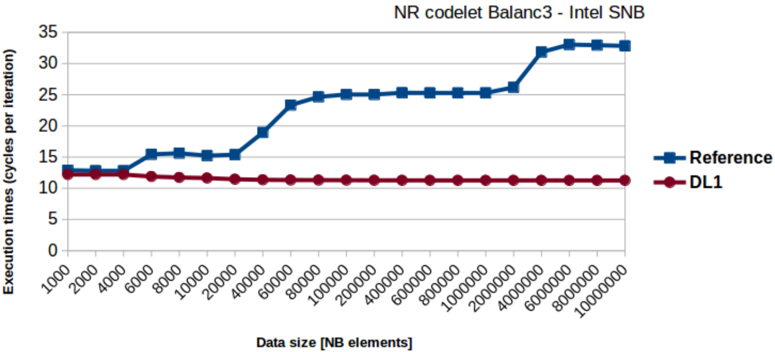
```
MOVAPS (%RDI,%R8,8),%XMM2
```

```
MOVAPS 456876(%RIP),%XMM2
```

Effect

- Simulates the case of an ideal memory bahaviour (L1 access)

Memory operations investigation - DL1



Memory operations investigation - S2L

S2L variant

Transform a store operation into a load operation

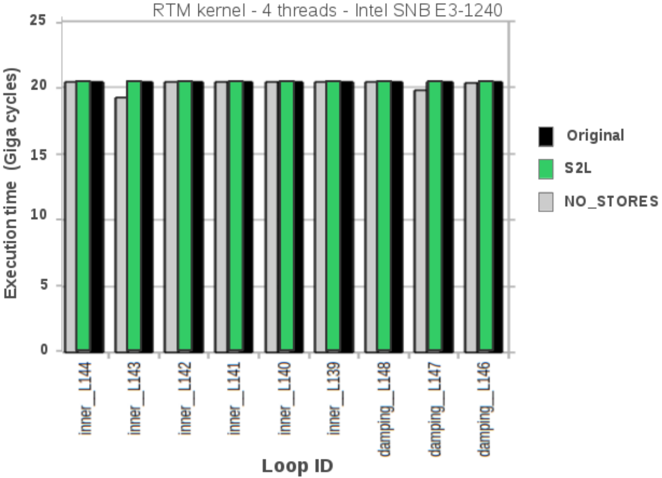
```
MOVAPS  %XMM2,(%RDI,%R8,8)
```

```
MOVAPS  (%RDI,%R8,8),%XMM2
```

Effect

- Disables all the cache effects caused by stores (cache coherency issues)

Memory operations investigation - S2L

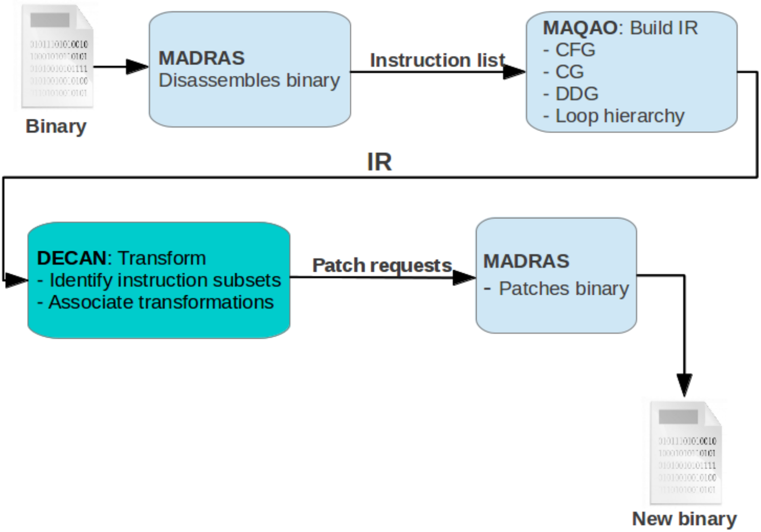


Concerns

- Destroying loop semantic can corrupt the control flow
- Transforming instructions may change the entire behaviour of the loop
- How are parallel codes handled
- How good measurements are

Outline

DECAN variant creation process



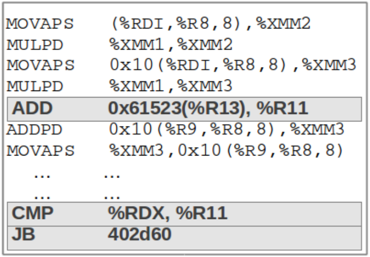
Control flow corruption

```
For ( cond ){
   If( cond ){
      . . .
   }else{
      . . .
   }
   For(cond){
      . . .
      . . .
   }
   If(cond){
      . . .
   }
```

Types

- Inner control flow
- Outer control flow

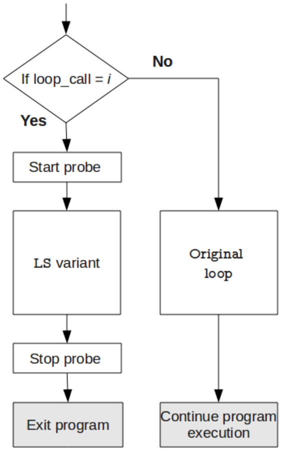
Inner control flow: instruction blacklist



Instruction blacklist

- Construction of Loop control instructions subset
- **blacklist** the subset

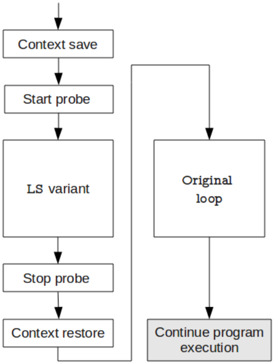
Outer control flow: instance mode



Instance mode

- Two variants of the loop
- Early end of program execution
- Sampling on loop calls

Outer control flow: recovery mode



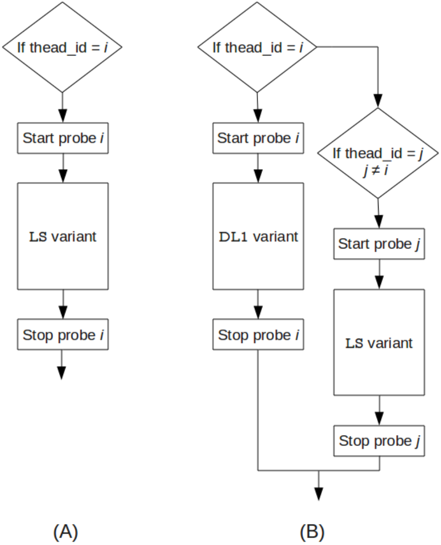
Recovery mode

- Two variants of the loop
- Full program execution

Other side effects

Side effect	Workarounds
Code layout change	Replace deleted instructions with NOPs
Data dependency	Micro-benchmarking to detect dependency subtleties
Variable latency instructions	Control latency by loading the operands
Floating point exceptions	Deactivate software exception handling

Parallel codes: thread based



Operatory modes

- **(A)** Homogeneous
- **(B)** Heterogeneous

Parallel codes: process based

MPI

- Each process is considered as an individual application
- All processes execute the same loop variant
- Each process has its own reports

Measurements: Studied aspects

Stability

- Related to the reproducibility of measurements
- Also known as measurement bias

Precision

- Related to probe placement and lightweightness
- The ability to measure only the events of the target area

Intrusiveness

- Related to probe quality
- The ability to separate probe noise from the measurements

Experimental methodology

- Measurements were done on 22 NR codelets
- Several data size points used (462)
- Compare real measures against reference measures

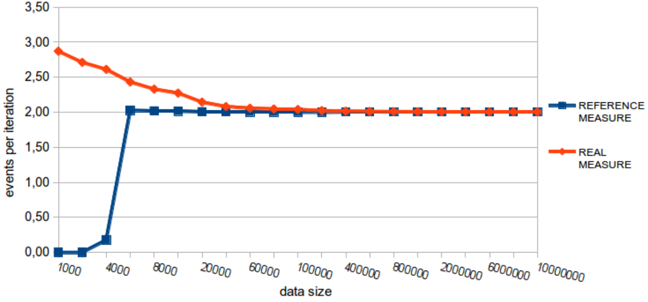
Reference measures

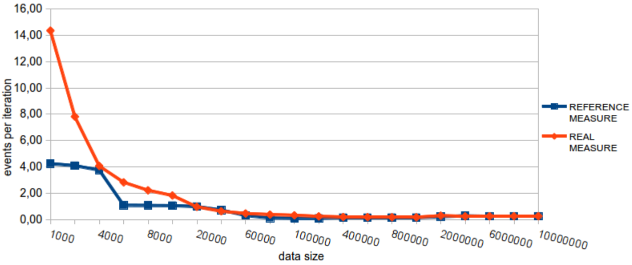
```
Data: codelet data
Result: codelet results
begin
    Monitoring_Start()
    for rep = 1 to NREP do
        Codelet()
    Monitoring_Stop()
end
```

Real measures

```
Data: codelet data
Result: codelet results
begin
    for rep = 1 to NREP do
        Monitoring_Start()
        Codelet()
        Monitoring_Stop()
end
```

Measurement precision





Goal

The possiblity to define a threshold on event count

Outline

1. Differential Analysis
2. Technical challenges
3. PAMDA
4. Conclusion

Observations

Differential Analysis

- A range of loop characterization capabilities
- pathology cost assessment

MAQAO

- A set of specialized tools: CQA, MTL, PROFILER
- Common view of the binary (loops, functions,..)

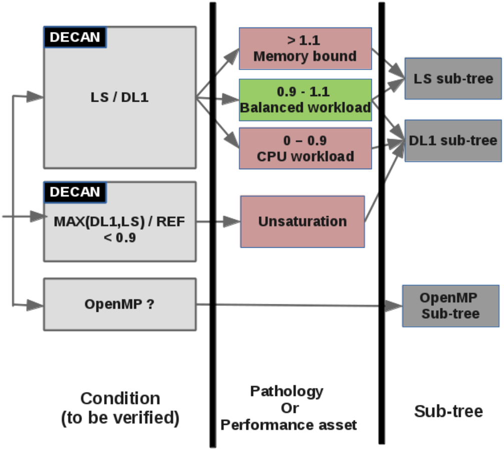
Idea: analysis methodology

Use Differential Analysis as a coordination means between multiple tools

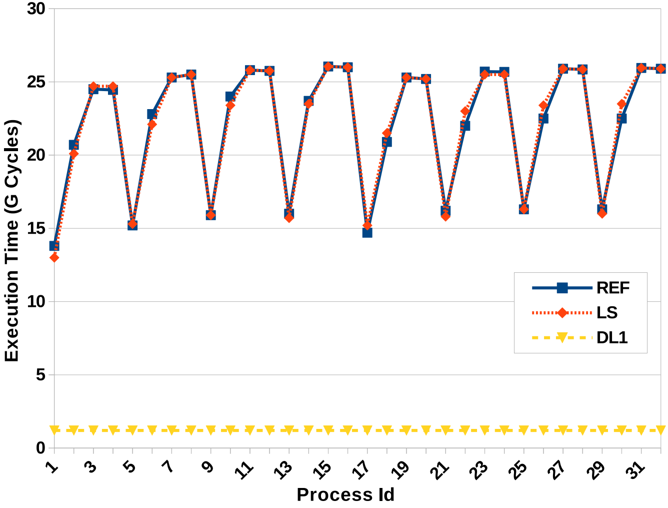
Case Study: PNBench

- PNbench is an application used at the CEA
- OpenMP/MPI code

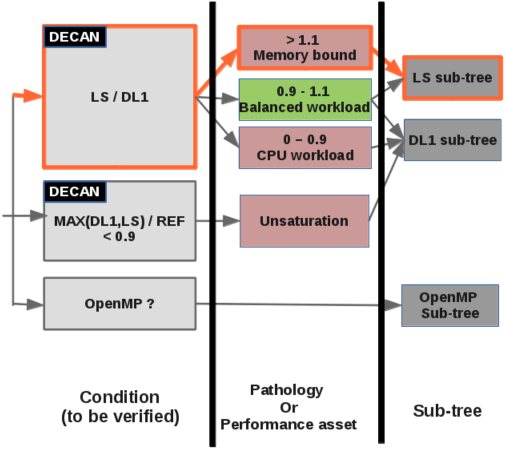
PAMDA analysis scheme



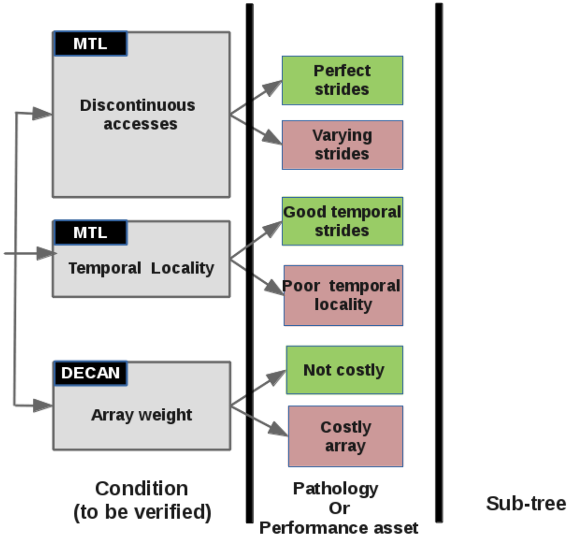
Differential Analysis: LS and DL1



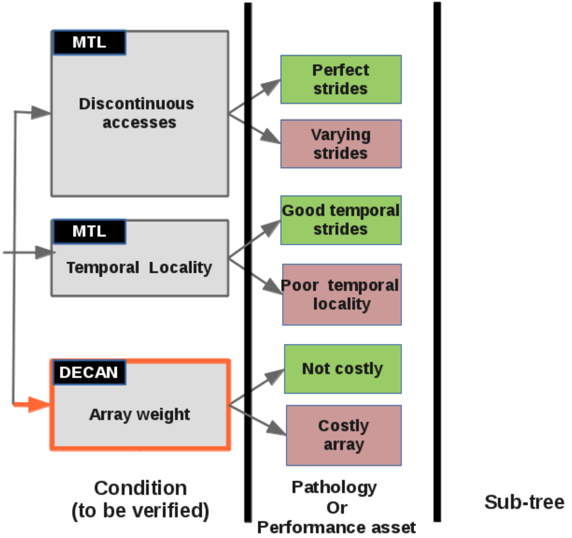
LS sub-tree selection



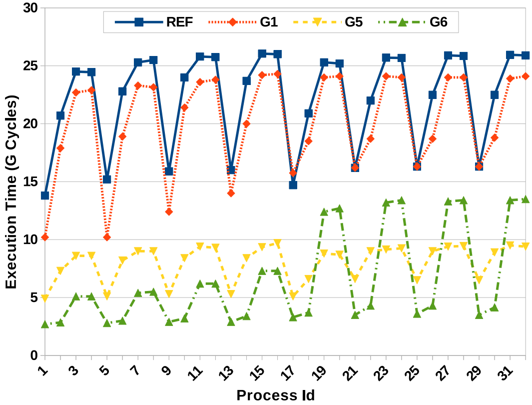
LS sub-tree


MTL
Discontinuous accesses
MTL
Temporal Locality
DECAN
Array weight
Perfect strides
Varying strides
Good temporal strides
Poor temporal locality
Not costly
Costly array
Condition (to be verified)
Pathology Or Performance asset
Sub-tree

LS sub-tree


MTL
Discontinuous accesses
Perfect strides
Varying strides
MTL
Temporal Locality
Good temporal strides
Poor temporal locality
DECAN
Array weight
Not costly
Costly array
Condition (to be verified)
Pathology Or Performance asset
Sub-tree

Differential Analysis: array weight



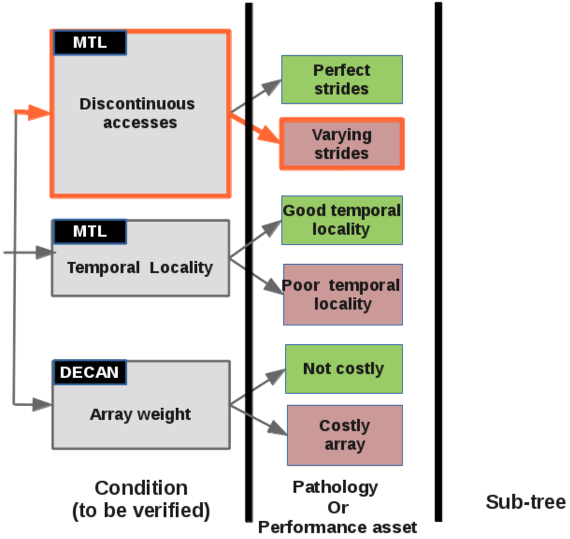
Loop arrays

- Three arrays G1, G5, G6

Array weight

- Determine memory operations group cost by deleting it

LS sub-tree



MTL

- Trace only G6 memory operations
- G6 had complex varying strides
- Hint: loop interchange

Outline

Contributions

Differential analysis

- Design, test and validation on real applications of new variants
- More advanced instruction modification process
- The ability to target more subtle perofrmance pathologies
- Use Differential Analysis outside the performance analysis field: SW/HD codesign

Contributions

DECAN tool

- Special handling of control flow corruption (In vivo)
- Support for parallel programs: thread (OpenMP) and process based
- Defining solutions and workarounds for a wide number of side effects
- A stistical study on the accuracy of measurements

Analysis methodology: PAMDA

- Use Differential Analysis as a coordiantion means between multiple analysis tools
- use the right effort (analysis technique) at the right moment

Future work

- The analysis method
  - Continue the exploration of new variants following the analysis needs
  - Explore the use of the method in other areas: energy and SW/HD codesign
  - Integrate more tools in the analysis methodology PAMDA
- The tool
  - Improve the analysis time (multiple loops in a single run)
  - Extend the tool to handle multi-path loops
  - Develop support for other platforms (ARM)

Thank you !

Principle of Differential analysis

Identify the potential costly instructions

```
MOVAPS  0(%RDI,%R8,8),%XMM4
MOVAPS  0x10(%RDI,%R8,8),%XMM5
DIVPD   %XMM1,%XMM4
DIVPD   %XMM1,%XMM5
MOVAPS  0x20(%RDI,%R8,8),%XMM6
MOVAPS  0x30(%RDI,%R8,8),%XMM7
DIVPD   %XMM1,%XMM6
DIVPD   %XMM1,%XMM7
MOVAPS  %XMM4,0(%RDI,%R8,8)
MULPD   %XMM4,%XMM4
MOVAPS  %XMM5,0x10(%RDI,%R8,8)
MULPD   %XMM5,%XMM5
ADDPD   %XMM4,%XMM3
ADDPD   %XMM5,%XMM2
...
ADDPD   %XMM6,%XMM3
ADDPD   %XMM7,%XMM2
CMP     %RAX,%R8
JB      Loop
```

Memory operations investigation - array cost

Groups subset (static analysis)

Two instructions are part of the same group if they target an address using the same base and index register values

- ADDSS 12(%RDI, %R8, 4), %XMM0
- ADDSS 24(%RDI, %R8, 4), %XMM1

Fast memory tracer (dynamic analysis)

Dynamic tracing of memory references of the loop. Groups are constructed following the rules:

- I1 = [@L1,@H1] and I2 = [@L2,@H2]
- if $I1 \cap I2 \neq \{\emptyset\} \rightarrow G = \{I1, I2\}$

Minimum loop slowdown is $\simeq$7 and maximum is $\simeq$37

Memory operations investigation - array cost

EUFLUX (3D finite element CFD app)

Sparse matrix-vector product in a quadruply nested loop

Loop code

```
do icb=1,ncbt
    ...
   do ig=1,igt
      ...
     do k=1,ndof
         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
```

Motivations

Several arrays accessed: need to detect the delinquent ones

Memory operations investigation - array cost

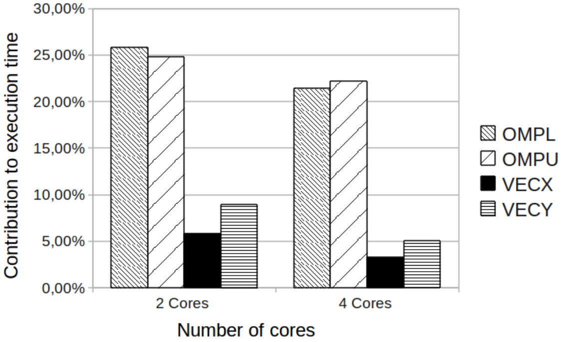
Analysis

- Detect instruction groups

Analysis	groups detected	analysis cost
Static analysis	10	0
Dynamic analysis	4	12.27

- link assembly groups to source arrays with debug information
- Delete an array at a time and monitor performance

Quantifying the access to individual memory structure (Results)



Conclusion:

- $OMPL$ and $OMPU$ are the delinquent arrays
- Focus on these two arrays: How they are accessed, the interaction with the other arrays

Principle of Differential analysis

Identify the potential costly instructions → Transform them

```
MOVAPS  0(%RDI,%R8,8),%XMM4
MOVAPS  0x10(%RDI,%R8,8),%XMM5
XORPS   %XMM1,%XMM4
XORPS   %XMM1,%XMM5
MOVAPS  0x20(%RDI,%R8,8),%XMM6
MOVAPS  0x30(%RDI,%R8,8),%XMM7
XORPS   %XMM1,%XMM6
XORPS   %XMM1,%XMM7
MOVAPS  %XMM4,0(%RDI,%R8,8)
MULPD   %XMM4,%XMM4
MOVAPS  %XMM5,0x10(%RDI,%R8,8)
MULPD   %XMM5,%XMM5
ADDPD   %XMM4,%XMM3
ADDPD   %XMM5,%XMM2
...
ADDPD   %XMM6,%XMM3
ADDPD   %XMM7,%XMM2
CMP     %RAX,%R8
JB      Loop
```

Principle of Differential analysis

Identify the potential costly instructions → Transform the instructions → Run and Compare

Generated binary

↓

Original binary

↓

Compare performance

Analysis example

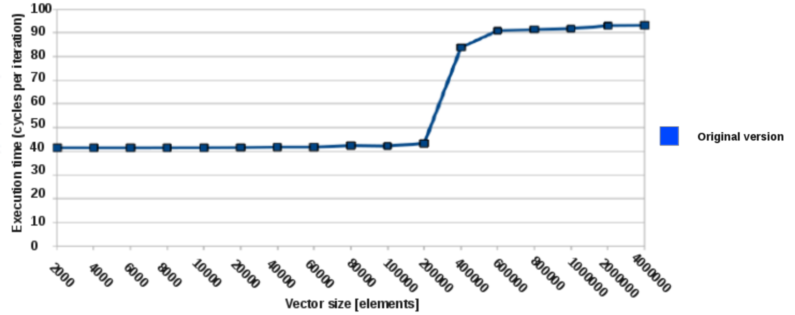
Sample code:

```
real * 8 A(N,16), scal, s(16) {Column oriented storage}
DO i = 1,16 (Parallel loop)
  DO k= 1, N
   A(k,i) = A(k,i)/scal
   s(i) = s(i) + A(k,i) * A(k,i)
  ENDDO
ENDDO
```

Characteristics

- Stride one, perfect load balance
- Two potential problems: Divide and Reduction

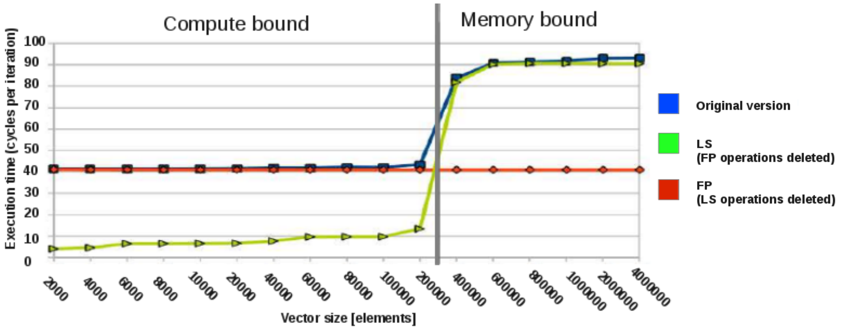
Analysis example (2)



Step 1:

A time profile is performed on the original version of the code for multiple data sets

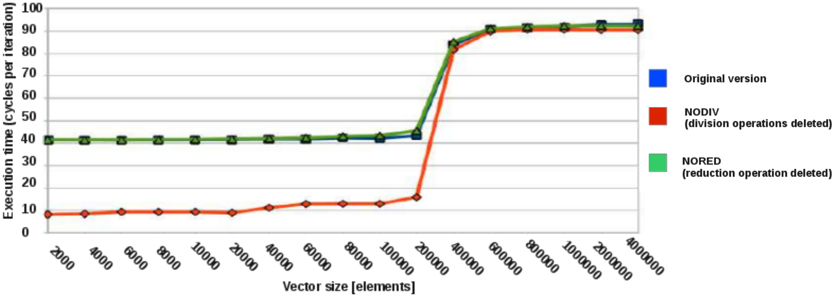
Analysis example (3) - LS/FP analysis



Step 2:

Isolate the memory stream (LS) and the FP arithmetic stream (FP)

Analysis example (4) - Expensive instructions analysis



Step 3:

Isolate the two important operations of the FP stream: division and reduction

Identify instruction subsets → Construct transformations requests → Inject monitoring probes → Generate a new binary

Generated binary

↓

Run program

↓

Original binary

↓

Run program

↓

Compare performance