



## Methodology for Application Performance Tuning

Andres Charif-Rubial, Souad Koliai, [Bettina  
Krammer](#), William Jalby, UVSQ  
Quang Dinh, Dassault-Aviation

**// ParMA**

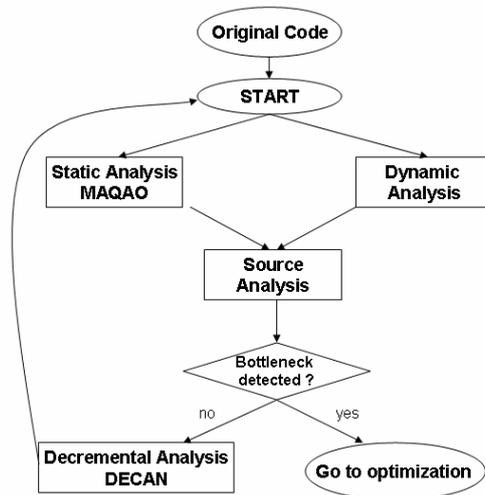
## Outline

- Motivation
- Methodology
  - Static analysis: MAQAO tool
  - Dynamic analysis: Hardware performance counters / MAQAO
  - DECAN (Decremental performance analysis)
- Example
  - ITRSOL by Dassault
- Conclusions

## Motivation

- How to optimise use of resources (CPU, RAM, I/O,...)?
  - Identify hotspots using different tools (see following talks...)
  - Platform dependent
- How much can be gained by optimising? When to stop optimising?
  - What is the theoretical peak performance of your code?
  - Cost of further optimisation: is it worth spending a lot of time on squeezing out 5% performance gain?
  - Using a different algorithm might sometimes be the better solution

## Methodology (semi-automatic)



## Static analysis with MAQAO

- **Modular Assembly Quality Analyzer and Optimizer**
  - Parses assembly code and builds the structure of code (call graph,...)
  - Correlation with source code if available
- **Supports Itanium and x86 platforms**
- **Main metrics:**
  - Vectorisation instruction usage (load, store, add, multiply)
    - Developer e.g. to insert directives if compiler failed to vectorise
  - Performance estimation for full vectorisation
  - Execution port usage
  - Performance estimation in L1 (number of cycles spent for one loop iteration assuming all operands are in L1) → optimal lower bound for execution
  - Similar: Performance estimation in L2/RAM (stride 1)
  - Loop attribute profiling (number of iterations, instructions per iteration)



Parco'09, Lyon, France  
2 September 2009



# Static analysis with MAQAO

Maqao Web Interface

New... Load profiling... Profile Loops Run

Functions

- Enter Here Project Name
- L\_rbgau5s\_50\_\_tree\_reduce2\_2.0
- L\_rbgau5s\_21\_\_par\_region0\_2.1
  - Loop SRC L30
  - Loop
 

Loop	DDG
rbgauss	DDG
Loop	0
Bounds:	
Predecoder bound	14*N
Decoder bound	14*N
Reorder Buffer bound	15*N
Execution ports (opt) bound	19*N
Execution ports (agner) bound	19.00*N
Front-end/Back-end ratio	0.79
Intructions/Cycles ratio	2.13
Packed Degree	0.00
Packed LOAD ratio	0.00
Packed STORE ratio	0.00
Packed MUL ratio	0.00
Packed ADD SUB ratio	0.00
Execution ports dispatch:	
P0	8
P1	10
P2	19
P3	1
P4	1
P5	5
L1 prediction:	19
L2 prediction min:	32.95
L2 prediction avg:	32.95
RAM prediction:	82.36

Loop's DDG

TS

ation of Arithmetic Instructions  
on units are the Bottleneck  
P2 is saturated  
and. Try to use instructions which don't use the same execution port. Example : avoid using (if it is possible)  
instruction (which generates 3 uops executed on port P1) with 'addps/d' and 'mulps/d' instructions (which  
es 1 uop executed on port P1)

ation of Load Instructions

```

3 #include <omp.h>
4 #include <stdint.h>
5 #include <xmmintrin.h>
6
7
8 #define _FABS(x) ((x)>0?(x):-x)
9
10 extern float resi;
11 extern float rsum;
12
13 void rbgauss(float *phi, int64_t nvar, float *su,
14 loop, float (*sarray)[800000], float (*rarray)[800
15 inpd, int64_t jnpd)
16 {
17     float urel,dlphi,hamb;
18     urel = 0.5;
19     nij = (inpd*jnpd);
20
21     #pragma omp parallel firstprivate(dlphi,hamb,i
22     {
23         #pragma omp single
24         {
25             resi = 0;
26             rsum = 0;
27         }
28
29         #pragma omp for reduction(-:resi,rsum)
30         for (ido=0; ido<nredd; ido++) {
31
32             inc = indir[id0]-1;
33
34             hamb = am0[inc] * phi[inc+1] +
35             am1[inc] * phi[inc-1] +
36             am2[inc] * phi[inc-1] +
    
```

## Dynamic analysis

- hardware performance counters
  - ~1200 counters on Xeon
  - Intel PTU
- Memory traces (MAQAO)
  - Detect stride access

## Decremental Analysis (DECAN)

- **Concept:**
  - measure original code (timing, cache miss rates)
  - Remove one or more instructions (may lead to incorrect results, instructions causing crash or alternating control flow are not removed)
    - Source level (check with MAQAO that compiler optimisation is still the same)
    - Assembly level
  - Measure again and find impact of this specific instruction
- **Helps identify the loads with poor memory access patterns**
  - Optimisation strategies: data reshaping, add software prefetch instructions
- **DECAN is tedious: automation at binary level in future**



Parco'09, Lyon, France  
2 September 2009



## Example: ITRSOL by Dassault

- Iterative solver for Navier-Stokes equation
- Most time-consuming subroutine: EUFLUXm
  - Sparse matrix-vector product
  - Two groups of 4-level-nested loops

## Example: ITRSOL by Dassault

```
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,omp1)
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) + omp1(e,k,l)*vecx(i,l)
        enddo
    enddo
enddo
enddo
```

## Example: ITRSOL by Dassault

- Static analysis (MAQAO): no vectorisation (SSE instructions) done by compiler
  - Loads cannot be vectorised due to non unit stride
  - But add/multiply could have been
  - Execution port usage: execution units ports can be improved, loads port a bottleneck
- Dynamic analysis
  - Loop attribute profiling: 2 innermost loop bounds (ndof) are small (4-10), outer loop bounds larger and vary throughout execution
  - Memory tracing: 2 innermost loops access arrays row-wise
  - Indirect addressing for accessing first dimension of arrays
  - PTU: 2-dim arrays in L2, 3-dim arrays from RAM
- DECAN: not performed (static and dynamic analysis sufficient)

## Example: ITRSOL by Dassault

- **Optimisation:**

- **Hardwire ndof (2 innermost loop bounds):**

- Replace ndof by its proper value (4) helps compiler fully unroll the 2 innermost loops

- **Loop interchange**

- Second loop (ig) becomes innermost loop
- Arrays now accessed column-wise
- But: still no vectorisation due to indirect addressing



*Parco'09, Lyon, France  
2 September 2009*

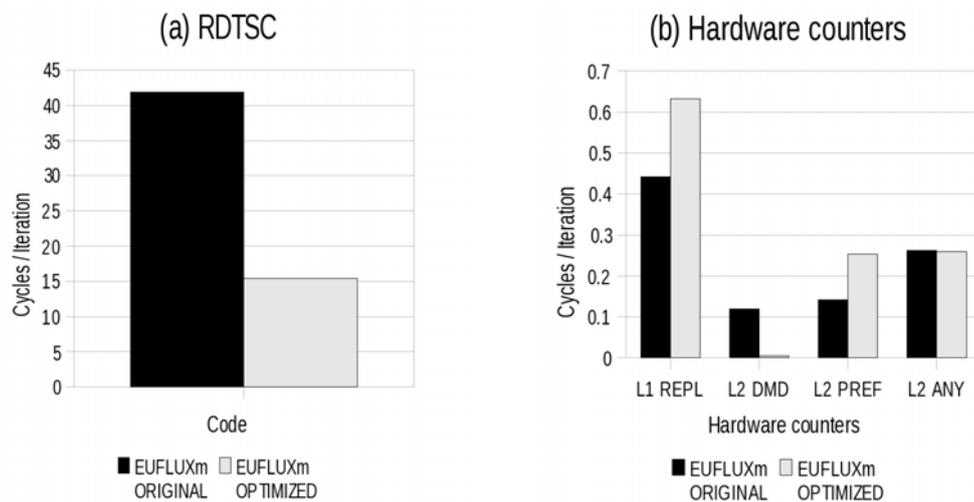
**ParMA**

## Example: ITRSOL by Dassault

- Experimental platform:

- Compute node with 4 Xeon X7350 (Tigerton), quad-core 2.93 GHz
- 2 4MB L2 caches shared between 2 cores
- 32kB L1 data cache (private to each core)
- 48 GB RAM
- Intel compilers v 10.1

## Example: ITRSOL by Dassault



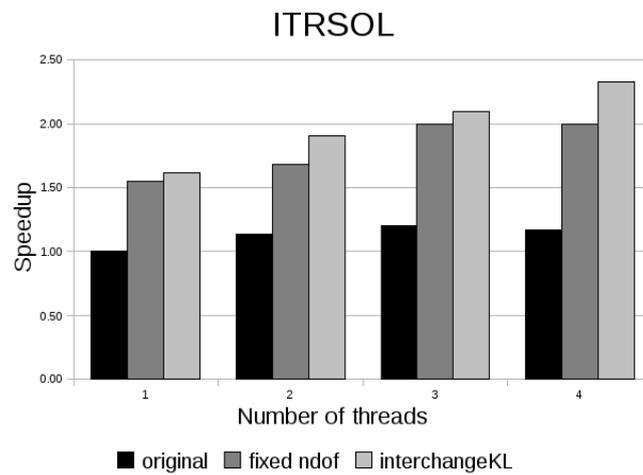
Unicore results



Parco'09, Lyon, France  
2 September 2009



## Example: ITRSOL by Dassault



Multicore results



Parco'09, Lyon, France  
2 September 2009

ParMA

## Conclusions and future work

- Semi-automatic performance analysis by combining tools for static and dynamic analysis (MAQAO, hardware performance counters) enables successful low-level code optimisation for real-world applications (factor of 2)
- This process can be further automated (e.g. analysis of memory traces, DECAN)

Thanks for your attention.



*Parco'09, Lyon, France*  
*2 September 2009*



Backup



*Parco'09, Lyon, France*  
2 September 2009

