Methodology for Application Performance Tuning

Andres Charif-Rubial, Souad Koliai, Bettina Krammer, William Jalby, UVSQ

Quang Dinh, Dassault-Aviation

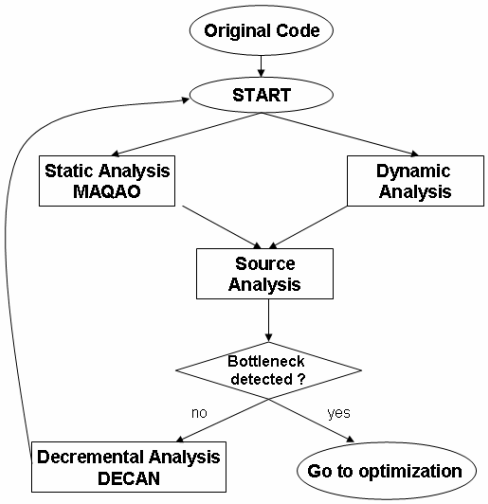
Outline

- 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

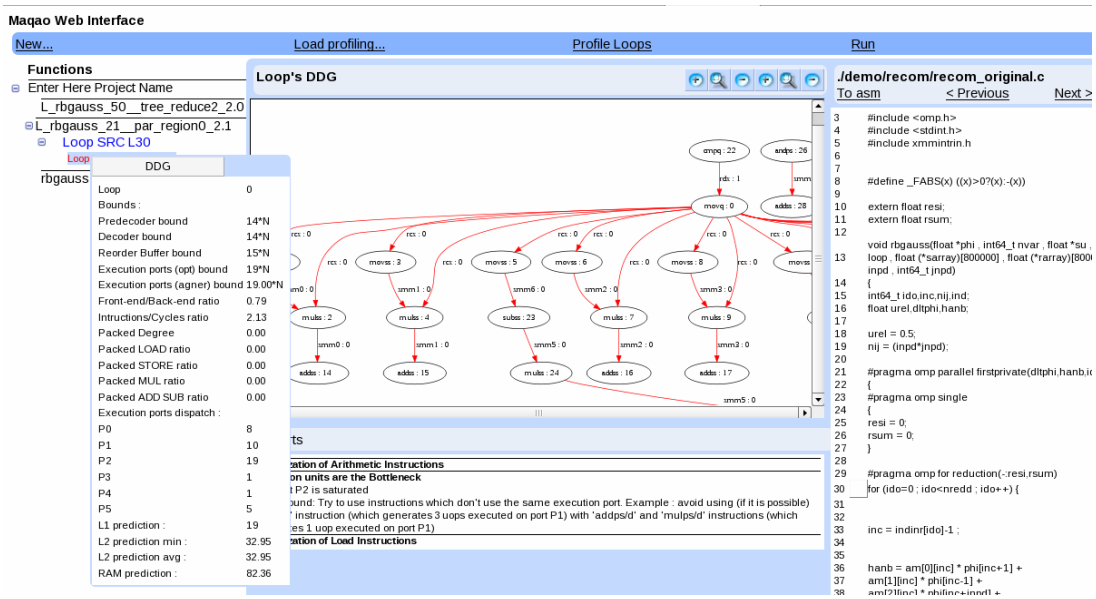
Methodology (semi-automatic)



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)

Static analysis with MAQAO



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 acess patterns
  - Optimisation strategies: data reshaping, add software prefetch instructions
- DECAN is tedious: automation at binary level in future

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,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
```

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)

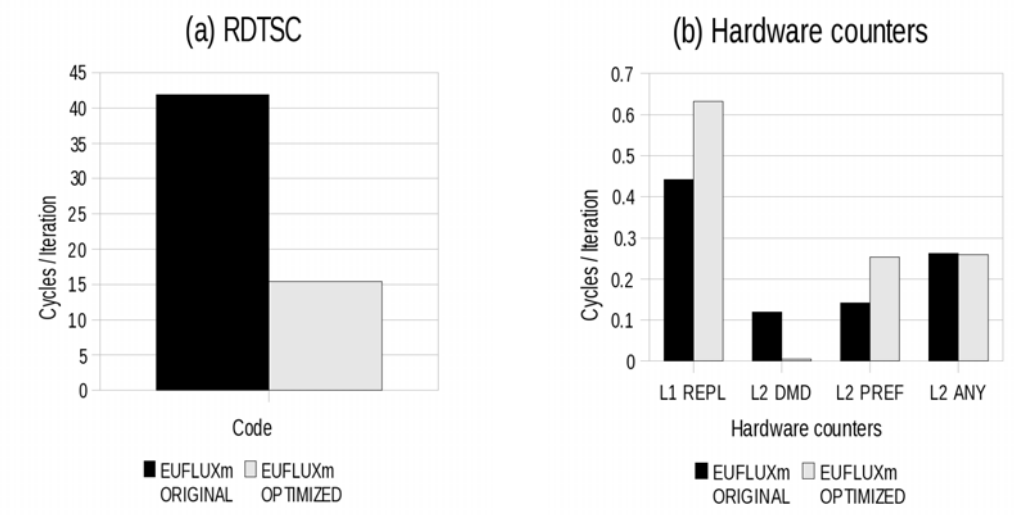
Example: ITRSOL by Dassault

- 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

Example: ITRSOL by Dassault

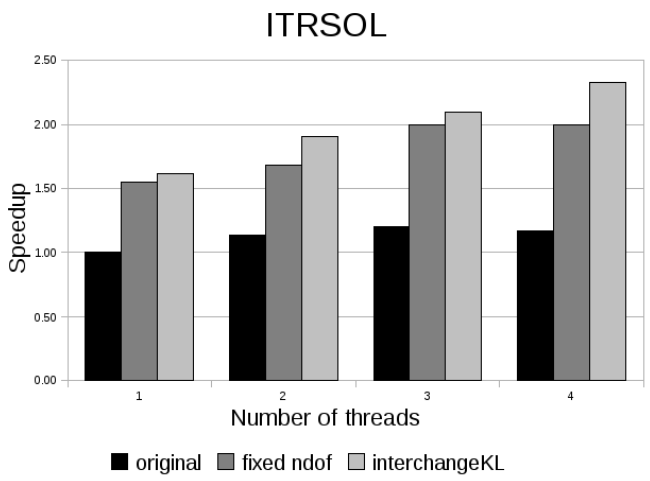
- Experimental platform:
  - Compute node with 4 Xeon X7350 (Tigerton), quad-core 2.93 GHz
  - 2 4MB L2 caches shared between 2 cores
  - 32kB L1 data cache (private to each core)
  - 48 GB RAM
  - Intel compilers v 10.1

Example: ITRSOL by Dassault



Unicore results

Example: ITRSOL by Dassault



Multicore results

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.

Backup