PhD Defense

Optimisation de code base sur des transformations source-a-source guides par des métriques issues de profilages

Youenn Lebras

Advisor : William Jalby
Co-supervisor : Andres S. Charif-Rubial

3 July 2019
Computer Architecture Evolution

Recent evolution: a new hope?

- The performance model shifted from high frequency single core processors to multitasking high-core-count parallel architectures
- Larger vector lengths (AVX-512) & automatic vectorization
- Specialized ports (i.e. FMA)
- New kind of memory (i.e. HBM, Optane)
With great evolution comes great difficulties

- Increasing number of different architectures
- Additional optimization challenges related to parallelism
- Performance issues are heavily tied to increased vector lengths and advanced memory hierarchy
- The optimization process remains key to maintain a reasonable performance level on modern micro-processor architecture

- Optimizing code has become an art
- Harder and harder to optimize and maintain manually
- Time consuming and error-prone
### Motivating Example

```fortran
INTEGER :: lt_var_j
INTEGER :: lt_var_i

IF (m .GT. 100 .AND. n .GT. 100) THEN
  DO lt_var_j = 1, m, 8
    DO j = lt_var_j, min(m,lt_var_j + 8 - 1)
      DO lt_var_i = 1, n, 8
        DO i = lt_var_i, min(n,lt_var_i + 8 - 1)
          res = res+a(j,i)*b(j,i)
        END DO
      END DO
    END DO
  END DO
END IF

ELSE IF (n .EQ. 4) THEN
  DO j = 1,m
    res = res+a(j,1)*b(j,1)
    res = res+a(j,2)*b(j,2)
    res = res+a(j,3)*b(j,3)
    res = res+a(j,4)*b(j,4)
  END DO
END IF

ELSE
  DO j = 1,m
    DO i = 1,n
      res = res+a(j,i)*b(j,i)
    END DO
  END DO
END IF
```
Goals

**Key idea:** Performance analysis tools (e.g. Scalasca, MAQAO, Tau) are pretty good at identifying some specific problems, we need to go further and fix automatically performance issues.

**Automatic Source-to-Source assistant: ASSIST**

- Source code transformation framework
- Transformation driven framework: ideally detect whether a transformation is beneficial or not
- Exploiting performance analysis tools metrics
- Open to user advice
- Keep a maintainable code
Compilers

Compiler task

- Transform a human readable file into a computer readable one
- Optimizing an application for a target architecture
  - Evaluating if a sequence of transformations is optimal
  - Predicting the behavior of a multi-core processor which has complex pipelines, multiple functional units, complex memory hierarchy, hardware data prefetching, etc
- Profile Guided Optimization / Feedback-Directed Optimization
Profile Guided Optimization (PGO)

3 steps

- Producing an instrumented binary
- Executing the binary in order to obtain a profile (feedback data)
- Using the obtained feedback data to produce a new version that is expected to be more efficient
Profile Guided Optimization (PGO)

What is done (Intel PGO)

- Value profiling of indirect and virtual function calls
- Intermediate language (IR) is annotated with edge frequencies and block counts to guide optimization decisions
- Grouping hot/cold functions
Compilers Limitations

Main Limitations

- Remain conservative (static performance cost model & heuristics)
- PGO lacks information gathered and transformations
- Black box
- Can ignore user directives
- Searching the best sequence of transformations remains too complex

”Never send a human to do a machine job”
-Agent Smith
Performance Analysis Tool

What are they?
Can be classified into two types:
- **Static**: estimate different issues and control the code quality
- **Dynamic**: find what happened during the execution

What are they for?
- Analyze & profile sequential/parallel codes
- Detect hotspots & performance issues / bottlenecks
- Provide hints on how to improve the code
A novel study of how and when well-known transformations allow to gain on real-world HPC applications using a novel FDO source-to-source approach

A novel semi-automatic and user controllable method with a system open to user advices

An FDO tool combining both dynamic and static analysis information to guide code optimization

A more flexible alternative to compilers PGO / FDO modes

A verification system to check if our transformations do not have a negative impact on performances
Outline

1. Background

2. ASSIST
   - MAQAO
   - Design & Implementation
   - Supported Transformations
   - How to Trigger Transformations
   - Assessing Transformation Verification

3. Issues & Limitations

4. Experiments

5. Conclusion
MAQAO Modules

- Static analyzer
  - CQA: Code Quality Analyzer
- Dynamic analyzer (using sampling & tracing)
  - LProf: Lightweight Profiler
  - VProf: Value Profiler
  - DECAN: DECremental ANalysis
- Global view
  - OneView
Overview of Tool Usage

Automatic Source-to-Source assISTant (ASSIST):

Diagram:
- Transformation Script
- (Annotated) Source File
- Parsing into AST
- AST Modifications
- Rewriting System
- Modified Source File
- Static Analyses
- Dynamic Analyses
- Optional
Automatic Source-to-Source asslSTant (ASSIST)

Technical Design

- Based on the Rose Compiler Project
- Support of Fortran 77, 90, 95, 2003 / C / C++03
- Same language at input and output
- Aiming at be easy to use with a simple user interface
- Targeting different kind of users
- Integrated as a MAQAO Module
Supported Transformations

Different types of transformations

**AST Modifier**
- Unroll
- Full Unroll
- Interchange
- Tile
- Strip Mine
- Loop/function Specialization

**Directive(s) insertion**
- Loop Count (LCT)

**Mix of both**
- Short Vectorization (SVT)
Zoom on LCT

Loop count Transformation - Type: Directives insertion

- Loop count knowledge enables to guide compiler optimizations choices
- Compilers cannot always guess the loop trip count at compile time
- Simplify
  - Control flow (less loop versions)
  - Choice of vectorization/unrolling
- Requires dynamic feedback (VPROF)
- Limitations
  - Loop bounds are dataset dependent
  - Only for Intel Compiler; unfortunately, other compilers do not offer such capability
Short Vectorization Transformation - Type: Mix AST modifier and directive insertion

- Compilers may refuse to vectorize a loop with too few iterations
- Performing a loop decomposition
- Increasing the vectorization ratio by:
  - Forcing the vectorization (SIMD Directive)
  - Avoiding dynamic or static loop peeling transformation (UNALIGNED Directive)
Zoom on SVT

(a) Before Short Vectorization

#pragma MAQAO SHORTVEC=AVX2
for i=0; i < 7; i++ do
    <Body>
end for

(b) After Short Vectorization

#pragma simd
#pragma vector unaligned
for i=0; i < 4; i++ do
    <Body>
end for
#pragma simd
#pragma vector unaligned
for i=4; i < 6; i++ do
    <Body>
end for
<Body>
Zoom on SVT

(a) Before Short Vectorization

```c
#pragma MAQAO SHORTVEC=AVX2
for i=0; i < 7; i++ do
    <Body>
end for
```

(b) After Short Vectorization

```c
#pragma simd
#pragma vector unaligned
for i=0; i < 4; i++ do
    <Body>
end for
#pragma simd
#pragma vector unaligned
for i=4; i < 6; i++ do
    <Body>
end for
<Body>
```

Loop decomposition
Residual
Transformation Example

(a) Source code

#pragma MAQAO UNROLL=4
for i=0; i < n; i++ do
    a[i] += b[i];
end for

(b) Corresponding AST
for i=0; i < n; i++ do
    a[i] += b[i];
end for

(a) Source code  (b) Corresponding AST
Supported Transformations

Transformation Example

for i=0; i < n; i++ do
  a[i] += b[i];
  a[i+1] += b[i+1];
end for

(a) Source code

(b) Corresponding AST
for i=0; i < n; i++ do
  a[i] += b[i];
  a[i+1] += b[i+1];
  a[i+2] += b[i+2];
end for

(a) Source code
(b) Corresponding AST
for i=0; i < n; i++ do
    a[i] += b[i];
    a[i+1] += b[i+1];
    a[i+2] += b[i+2];
    a[i+3] += b[i+3];
end for

(a) Source code

(b) Corresponding AST
for i=0; i < n; i++ do
    a[i] += b[i];
    a[i+1] += b[i+1];
    a[i+2] += b[i+2];
    a[i+3] += b[i+3];
end for

(a) Source code

(b) Corresponding AST
for i = 0; i < n; i += 4 do
   a[i] += b[i];
   a[i+1] += b[i+1];
   a[i+2] += b[i+2];
   a[i+3] += b[i+3];
end for

(a) Source code

(b) Corresponding AST
for $i=0; i < n; i += 4$ do
  $a[i] += b[i]$;
  $a[i+1] += b[i+1]$;
  $a[i+2] += b[i+2]$;
  $a[i+3] += b[i+3]$;
end for

(a) Source code

(b) Corresponding AST
for i=0; i<n-3; i+=4 do
    a[i] += b[i];
    a[i+1] += b[i+1];
    a[i+2] += b[i+2];
    a[i+3] += b[i+3];
end for

\( (a) \) Source code

\( (b) \) Corresponding AST
How to Trigger Transformations

3-ways

- Insert directives in sources
- Provide a transformation script
- Use OneView report
  - SVT $\Rightarrow$ CQA (vectorization ratio) + VPROF (iteration count)
  - Tiling $\Rightarrow$ DECAN (DL1)
  - Loop count $\Rightarrow$ VProf (Iteration count)

```
!DIR$ MAQAO UNROLL=4
!DIR$ MAQAO FULLUNROLL
!DIR$ MAQAO INTERCHANGE=1,2
!DIR$ MAQAO TILE=5
!DIR$ MAQAO SPECIALIZATION(a=5, b={1,10}, c<50)
```
How to Trigger Transformations

3-ways

- Insert directives in sources
- Provide a transformation script
- Use OneView report
  - SVT $\Rightarrow$ CQA (vectorization ratio) + VPROF (iteration count)
  - Tiling $\Rightarrow$ DECAN (DL1)
  - Loop count $\Rightarrow$ VProf (Iteration count)

```
loops = {
    { line = 26, transformation = {"TILE=5"} },
    { line = 34, transformation = {"UNROLL=8"} },
    { label = "LOOPLABEL1", transformation = {"INTERCHANGE=1,2"} }
}
```
How to Trigger Transformations

3-ways

- Insert directives in sources
- Provide a transformation script
- Use OneView report
  - SVT $\Rightarrow$ CQA (vectorization ratio) + VPROF (iteration count)
  - Tiling $\Rightarrow$ DECAN (DL1)
  - Loop count $\Rightarrow$ VProf (Iteration count)

```
loop_id    = 5,
lineStart = 5,
lineStop  = 7,
file      = "codelet.c",
ite_min   = 1000000,
ite_max   = 1000000,
ite_avg   = 1000000,
```

```
 r_l1_min = 2.361754348463,
 r_l1_max = 2.0752602660095,
 r_l1_med = 2.3972978247604,
 vecRatio = 12.5,
```
Assessing Transformation Verification

Process

- Step 1: Execute ONEVIEW on the Nth version.
- Step 2: Use analysis info to apply transformation on the $N^{th}$ version.
- Step 3: Compare global metrics and CQA, DECAN and VPROF metrics between $N^{th}$ and $N + 1^{th}$. 
### Assessing Transformation Verification

<table>
<thead>
<tr>
<th>Global Metrics</th>
<th>Before Transfo</th>
<th>After Transfo</th>
<th>Is Better?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Total Time (s):</strong></td>
<td>2.34</td>
<td>1</td>
<td>lower is better</td>
</tr>
<tr>
<td>Flow Complexity</td>
<td>1</td>
<td>1</td>
<td>higher is better</td>
</tr>
<tr>
<td>Array Access Efficiency (%)</td>
<td>71.95</td>
<td>54.66</td>
<td>lower is better</td>
</tr>
<tr>
<td>Speedup if clean</td>
<td>1</td>
<td>1.04</td>
<td>lower is better</td>
</tr>
<tr>
<td>Nb loops to get 80% if clean</td>
<td>1</td>
<td>1</td>
<td>lower is better</td>
</tr>
<tr>
<td>Speedup if FP Vectorised</td>
<td>1</td>
<td>1.47</td>
<td>lower is better</td>
</tr>
<tr>
<td>Nb loops to get 80% if FP vectorized</td>
<td>1</td>
<td>1</td>
<td>lower is better</td>
</tr>
<tr>
<td>Speedup if Fully Vectorised</td>
<td>1.66</td>
<td>1.87</td>
<td>lower is better</td>
</tr>
<tr>
<td>Nb loops to get 80% if Fully vectorized</td>
<td>3</td>
<td>2</td>
<td>lower is better</td>
</tr>
<tr>
<td>Speedup if data in L1 Cache</td>
<td>1</td>
<td>1.18</td>
<td>lower is better</td>
</tr>
<tr>
<td>Nb loops to get 80% if data in L1 Cache</td>
<td>0</td>
<td>2</td>
<td>lower is better</td>
</tr>
<tr>
<td>Compilation Options</td>
<td>OK</td>
<td>OK</td>
<td></td>
</tr>
<tr>
<td>Loop lines</td>
<td>314 - 314</td>
<td>881-884</td>
<td></td>
</tr>
<tr>
<td>Loop_id</td>
<td>74</td>
<td>246</td>
<td></td>
</tr>
<tr>
<td>Speedup if clean</td>
<td>1.00</td>
<td>1.06</td>
<td>lower is better</td>
</tr>
<tr>
<td>Speedup if fully vectorized</td>
<td>1.62</td>
<td>2.06</td>
<td>lower is better</td>
</tr>
<tr>
<td>Bottlenecks</td>
<td>p5</td>
<td>P0.01</td>
<td></td>
</tr>
<tr>
<td>Unroll confidence level</td>
<td>max</td>
<td>max</td>
<td></td>
</tr>
<tr>
<td>Cycles L1 if clean</td>
<td>13</td>
<td>8</td>
<td>higher is better</td>
</tr>
<tr>
<td>Cycles L1 if fully vec</td>
<td>8</td>
<td>2.06</td>
<td>higher is better</td>
</tr>
<tr>
<td>Vector-efficiency ratio all</td>
<td>85.71</td>
<td>42.06</td>
<td>higher is better</td>
</tr>
<tr>
<td>Vectorization ratio all</td>
<td>67.86</td>
<td>68.00</td>
<td>higher is better</td>
</tr>
<tr>
<td>FP op per cycle L1</td>
<td>2.46</td>
<td>3.76</td>
<td>higher is better</td>
</tr>
</tbody>
</table>
Outline

1. Background
2. ASSIST
3. Issues & Limitations
4. Experiments
5. Conclusion
Issues & Limitations

Analysis

- Debug information accuracy
- What information to collect while limiting the overhead
Issues & Limitations

Analysis

- Debug information accuracy
- What information to collect while limiting the overhead

Transformations

- Rose frontend/backend issues on Fortran/C++
- How to match the right transformation with collected metrics
- Compiler can ignore a transformation
- Directives are often compiler dependent
Issues & Limitations

Analysis
- Debug information accuracy
- What information to collect while limiting the overhead

Transformations
- Rose frontend/backend issues on Fortran/C++
- How to match the right transformation with collected metrics
- Compiler can ignore a transformation
- Directives are often compiler dependent

Verification
- Compare two different binaries (loop splitted/duplicated, disappeared, etc)
Outline

1 Background

2 ASSIST

3 Issues & Limitations

4 Experiments
   - Impact of the Loop Count
   - Impact of Specialization
   - Impact of Specialization with SVT
   - Impact of Specialization with Tiling

5 Conclusion
Experiments

Results have been obtained on a Skylake Server and are compiled with Intel 17.0.4 and compared to Intel PGO version 17.0.4 (IPGO)

Application Pool

- **Yales2 (F03)**: numerical simulator of turbulent reactive flows
- **AVBP (F95)**: parallel computational fluid dynamics code
- **ABINIT (F90)**: find the total energy charge density and the electronic structure of systems made of electrons and nuclei
- **POLARIS MD (F90)**: microscopic simulator for molecular systems
- **Convolution Neural Networks (C)**: objet recognition.
- **QmcPack (C++)**: computation of the real space quantum Monte-Carlo algorithms
Impact of the Loop Count

Comparision with IPGO and ASSIST LCT + IPGO.

**YALES2 - 3D CYLINDER**

- Speedup (higher is better)
- Number of processes (MPI)
- Speedup
- MPI %

**YALES2 - 1D COFFEE**

- Speedup (higher is better)
- Number of processes (MPI)
- Speedup
- MPI %
Impact of the Loop Count

<table>
<thead>
<tr>
<th></th>
<th>AVBP NASA</th>
<th>AVBP TPF</th>
<th>AVBP SIMPLE</th>
<th>Yales2 3D Cylinder</th>
<th>Yales2 1D COFFEE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of loops</td>
<td>149</td>
<td>173</td>
<td>158</td>
<td>162</td>
<td>122</td>
</tr>
</tbody>
</table>

Cumulated speedup for Yales2 - 3D Cylinder & AVBP - NASA, sorted by coverage.
Impact of Specialization

**SRC ORIG**

```c
for (img = 0; img < nImg; ++img) {
    for (ifm = 0; ifm < nIfm; ++ifm) {
        for (ofm = 0; ofm < nOfm; ++ofm) {
            for (oj = 0; oj < ofh; ++oj) {
                i = oj * stride_h - pad_h;
                for (oi = 0; oi < ofw; ++oi) {
                    ii = oi * stride_w - pad_w;
                    for (kj = 0; kj < kh; ++kj) {
                        if (ij+kj < 0 || ij+kj >= ifh) continue;
                        for (ki = 0; ki < kw; ++ki) {
                            if (ii+ki < 0 || ii+ki >= ifw) continue;
                            input_t[(img * input_img_stride) +
                                (ifm * input_ifm_stride) +
                                ((ij + kj) * ifwp) + (ii + ki)] +=
                            output[(img * output_img_stride) +
                                (ofm * output_ofm_stride) +
                                (oj * ofw) + oi] *
                            filter[(ofm * weight_ofm_stride) +
                                (ifm * weight_ifm_stride) +
                                (kj * kw) + ki];
                        }
                    }
                }
            }
        }
    }
```

**SRC SPE**

```c
if (kw == 1 && ifw == 28 && stride_w == 1 && pad_w == 0 &&
    ofw == 28 && kh == 1 && ifh == 28 && ofh == 28
    && stride_h == 1 && pad_h == 0 ) {
    for (img = 0; img < nImg; ++img) {
        for (ifm = 0; ifm < nIfm; ++ifm) {
            for (ofm = 0; ofm < nOfm; ++ofm) {
                for (oj = 0; oj < 28; ++oj) {
                    i = oj * 1 - 0;
                    for (oi = 0; oi < 28; ++oi) {
                        ii = oi * 1 - 0;
                        for (kj = 0; kj < 1; ++kj) {
                            for (ki = 0; ki < 1; ++ki) {
                                input_t[(img * input_img_stride) +
                                    (ifm * input_ifm_stride) +
                                    ((ij + kj) * ifwp) + (ii + ki)] +=
                                output[(img * output_img_stride) +
                                    (ofm * output_ofm_stride) +
                                    (oj * 28) + oi] *
                                filter[(ofm * weight_ofm_stride) +
                                    (ifm * weight_ifm_stride) +
                                    (kj * 1) + ki];
                            }
                        }
                    }
                }
            }
        }
```
Impact of Specialization

Convolution Neural Network - (1x1) and 3(x3) filters

- Speedup (higher is better)

ASSIST specialization
Impact of Specialization combined with SVT

```
... DO n=1, nel
  DO no = 1, nvert
    !DIR$ SIMD
    DO k = -nproduct + 1, 0
      zobj(no * nproduct + k, n) = z(ielob(no,n) * nproduct + k)
    END DO
  END DO
END DO
...```

<table>
<thead>
<tr>
<th>Name</th>
<th>Coverage (%)</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>gather_o_cpy</td>
<td>13.67</td>
<td>21.18</td>
</tr>
<tr>
<td>Loop 16183 - gather_o_cpy.f90:197-201 - AVBP_V7.0.1_orig</td>
<td>0.27</td>
<td>0.42</td>
</tr>
<tr>
<td>Loop 16182 - gather_o_cpy.f90:198-201 - AVBP_V7.0.1_orig</td>
<td>3.83</td>
<td>5.93</td>
</tr>
<tr>
<td>Loop 16181 - gather_o_cpy.f90:197-201 - AVBP_V7.0.1_orig</td>
<td>0.23</td>
<td>0.36</td>
</tr>
<tr>
<td>Loop 16180 - gather_o_cpy.f90:198-201 - AVBP_V7.0.1_orig</td>
<td>2.32</td>
<td>3.59</td>
</tr>
<tr>
<td>Loop 16178 - gather_o_cpy.f90:197-201 - AVBP_V7.0.1_orig</td>
<td>0.09</td>
<td>0.14</td>
</tr>
<tr>
<td>Loop 16177 - gather_o_cpy.f90:198-201 - AVBP_V7.0.1_orig</td>
<td>0.58</td>
<td>0.90</td>
</tr>
<tr>
<td>Loop 16194 - gather_o_cpy.f90:197-201 - AVBP_V7.0.1_orig</td>
<td>0.08</td>
<td>0.12</td>
</tr>
<tr>
<td>Loop 16193 - gather_o_cpy.f90:198-201 - AVBP_V7.0.1_orig</td>
<td>2.07</td>
<td>3.21</td>
</tr>
<tr>
<td>Loop 16192 - gather_o_cpy.f90:197-201 - AVBP_V7.0.1_orig</td>
<td>0.07</td>
<td>0.11</td>
</tr>
<tr>
<td>Loop 16191 - gather_o_cpy.f90:198-201 - AVBP_V7.0.1_orig</td>
<td>1.82</td>
<td>2.82</td>
</tr>
<tr>
<td>Loop 16214 - gather_o_cpy.f90:197-201 - AVBP_V7.0.1_orig</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Loop 16213 - gather_o_cpy.f90:198-201 - AVBP_V7.0.1_orig</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Loop 16212 - gather_o_cpy.f90:199-201 - AVBP_V7.0.1_orig</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Loop 16211 - gather_o_cpy.f90:198-201 - AVBP_V7.0.1_orig</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Loop 16210 - gather_o_cpy.f90:197-201 - AVBP_V7.0.1_orig</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Loop 16209 - gather_o_cpy.f90:198-201 - AVBP_V7.0.1_orig</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Loop 16208 - gather_o_cpy.f90:197-201 - AVBP_V7.0.1_orig</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Loop 16207 - gather_o_cpy.f90:198-201 - AVBP_V7.0.1_orig</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Loop 16206 - gather_o_cpy.f90:197-201 - AVBP_V7.0.1_orig</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Loop 16205 - gather_o_cpy.f90:198-201 - AVBP_V7.0.1_orig</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Loop 16204 - gather_o_cpy.f90:197-201 - AVBP_V7.0.1_orig</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Loop 16203 - gather_o_cpy.f90:198-201 - AVBP_V7.0.1_orig</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Loop 16202 - gather_o_cpy.f90:197-201 - AVBP_V7.0.1_orig</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Loop 16201 - gather_o_cpy.f90:198-201 - AVBP_V7.0.1_orig</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Loop 16199 - gather_o_cpy.f90:197-201 - AVBP_V7.0.1_orig</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Loop 16198 - gather_o_cpy.f90:198-201 - AVBP_V7.0.1_orig</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Loop 16197 - gather_o_cpy.f90:197-201 - AVBP_V7.0.1_orig</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Loop 16196 - gather_o_cpy.f90:198-201 - AVBP_V7.0.1_orig</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Loop 16195 - gather_o_cpy.f90:197-201 - AVBP_V7.0.1_orig</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Loop 16194 - gather_o_cpy.f90:198-201 - AVBP_V7.0.1_orig</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Loop 16193 - gather_o_cpy.f90:197-201 - AVBP_V7.0.1_orig</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Loop 16192 - gather_o_cpy.f90:198-201 - AVBP_V7.0.1_orig</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Loop 16191 - gather_o_cpy.f90:197-201 - AVBP_V7.0.1_orig</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Loop 16190 - gather_o_cpy.f90:198-201 - AVBP_V7.0.1_orig</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Loop 16189 - gather_o_cpy.f90:197-201 - AVBP_V7.0.1_orig</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>Loop 16188 - gather_o_cpy.f90:198-201 - AVBP_V7.0.1_orig</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>
Impact of Specialization Combined with SVT

Function specialization

<table>
<thead>
<tr>
<th>Name</th>
<th>Coverage (%)</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>gather_o_cpy_flat_2d_assist_rntVert4_nriProductMod42</td>
<td>7.55</td>
<td>12.21</td>
</tr>
<tr>
<td>Loop 9633 - gather_o_cpy.f90:314-318 - AVBP_V7.0.1_speF</td>
<td>0.36</td>
<td>0.58</td>
</tr>
<tr>
<td>Loop 9632 - gather_o_cpy.f90:315-318 - AVBP_V7.0.1_speF</td>
<td>2.07</td>
<td>3.23</td>
</tr>
<tr>
<td>Loop 9633 - gather_o_cpy.f90:317-318 - AVBP_V7.0.1_speF</td>
<td>3.79</td>
<td>6.13</td>
</tr>
<tr>
<td>Loop 9634 - gather_o_cpy.f90:317-318 - AVBP_V7.0.1_speF</td>
<td>1.37</td>
<td>2.21</td>
</tr>
</tbody>
</table>

Loop specialization

<table>
<thead>
<tr>
<th>Name</th>
<th>Coverage (%)</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>gather_o_cpy</td>
<td>13.91</td>
<td>21.31</td>
</tr>
<tr>
<td>Loop 9465 - gather_o_cpy.f90:165-168 - AVBP_V7.0.1_speE</td>
<td>2.21</td>
<td>3.30</td>
</tr>
<tr>
<td>Loop 9473 - gather_o_cpy.f90:223-227 - AVBP_V7.0.1_speF</td>
<td>0.29</td>
<td>0.44</td>
</tr>
<tr>
<td>Loop 9472 - gather_o_cpy.f90:224-227 - AVBP_V7.0.1_speE</td>
<td>3.87</td>
<td>5.03</td>
</tr>
<tr>
<td>Loop 9471 - gather_o_cpy.f90:214-218 - AVBP_V7.0.1_speE</td>
<td>0.21</td>
<td>0.32</td>
</tr>
<tr>
<td>Loop 9470 - gather_o_cpy.f90:215-218 - AVBP_V7.0.1_speE</td>
<td>2.55</td>
<td>3.91</td>
</tr>
<tr>
<td>Loop 9486 - gather_o_cpy.f90:223-227 - AVBP_V7.0.1_speF</td>
<td>0.09</td>
<td>0.14</td>
</tr>
<tr>
<td>Loop 9485</td>
<td>3.87</td>
<td>5.03</td>
</tr>
</tbody>
</table>

Best specialization + Short Vectorization Transformation

<table>
<thead>
<tr>
<th>Name</th>
<th>Coverage (%)</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>gather_o_cpy</td>
<td>14.81</td>
<td>20.51</td>
</tr>
<tr>
<td>Loop 16014 - gather_o_cpy.f90:226-236 - AVBP_V7.0.1_SVT</td>
<td>3.52</td>
<td>4.87</td>
</tr>
<tr>
<td>Loop 16013 - gather_o_cpy.f90:215-222 - AVBP_V7.0.1_SVT</td>
<td>3.09</td>
<td>4.28</td>
</tr>
<tr>
<td>Loop 16009 - gather_o_cpy.f90:165-168 - AVBP_V7.0.1_SVT</td>
<td>2.51</td>
<td>3.48</td>
</tr>
<tr>
<td>Loop 16021 - gather_o_cpy.f90:226-236 - AVBP_V7.0.1_SVT</td>
<td>2.46</td>
<td>3.41</td>
</tr>
<tr>
<td>Loop 16020 - gather_o_cpy.f90:252-265 - AVBP_V7.0.1_SVT</td>
<td>2.45</td>
<td>3.39</td>
</tr>
<tr>
<td>Loop 16011 - gather_o_cpy.f90:241-248 - AVBP_V7.0.1_SVT</td>
<td>0.66</td>
<td>0.91</td>
</tr>
</tbody>
</table>
Impact of Specialization Combined with SVT

The figure shows the impact of different specialization techniques combined with SVT. The x-axis represents various operations, and the y-axis represents speedup, with higher values indicating better performance.

- ASSIST LCT
- IPGO
- ASSIST function specialization only
- ASSIST Loop specialization only
- ASSIST SVT on best specialization

The graph demonstrates the speedup for each operation across different techniques, highlighting the benefits of combining specialization with SVT.
Impact of Specialization with SVT

Impact of Specialization Combined with SVT

AVBP - TPF

AVBP - NASA

AVBP - SIMPLE
### Impact of Specialization Combined with SVT

<table>
<thead>
<tr>
<th></th>
<th>Execution Time before trans (sec)</th>
<th>Execution Time after trans (sec)</th>
<th>Speedup</th>
<th>Coverage (orig version)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Polaris</td>
<td>73.32</td>
<td>70.26</td>
<td>1.04</td>
<td></td>
</tr>
<tr>
<td>loop 6909</td>
<td>4.27</td>
<td>3.14</td>
<td>1.36</td>
<td>5.72%</td>
</tr>
<tr>
<td>loop 6911</td>
<td>3.64</td>
<td>2.36</td>
<td>1.54</td>
<td>4.98%</td>
</tr>
</tbody>
</table>

**Table:** Execution time and speedups of ASSIST SVT compared with the original version on Polaris using the "test_1.0.5.18" test case.
Impact of Specialization Combined with Tiling

(a) Before specialization + tiling

(b) After specialization + tiling
Impact of Specialization Combined with Tiling

<table>
<thead>
<tr>
<th></th>
<th># lines of code</th>
<th>Execution time (sec)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>original version</td>
<td>716</td>
<td>2.55</td>
<td>1</td>
</tr>
<tr>
<td>assist version</td>
<td>1338</td>
<td>1.47</td>
<td>1.75</td>
</tr>
</tbody>
</table>
Other results: QMCPACK other transformations

<table>
<thead>
<tr>
<th></th>
<th>orig</th>
<th>fu</th>
<th>split</th>
<th>fume</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>58.11</td>
<td>55.4</td>
<td>53.81</td>
<td>51.21</td>
</tr>
<tr>
<td>loop 18800</td>
<td>16.75</td>
<td>16.59</td>
<td>16.2</td>
<td>13.6</td>
</tr>
<tr>
<td>loop 26027</td>
<td>16.07</td>
<td>12.42</td>
<td>12.27</td>
<td>11.9</td>
</tr>
<tr>
<td>loop 26026</td>
<td>3.22</td>
<td>2.19</td>
<td>2.27</td>
<td>2.1</td>
</tr>
<tr>
<td>loop 26028</td>
<td>3.24</td>
<td>2.21</td>
<td>2.09</td>
<td>2.05</td>
</tr>
<tr>
<td>loop 18501</td>
<td>1.49</td>
<td>1.51</td>
<td>1.23</td>
<td>1.18</td>
</tr>
<tr>
<td>loop 26800</td>
<td>1.3</td>
<td>1.01</td>
<td>1.03</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Execution time (sec) of multiple versions of QMCPACK (k64s64).

Orig: Original version;
fu: Full unroll version;
split: fu + split a loop to increase its vectorization ratio;
fume: split + full unroll the inbetween loop of a nest and merge unrolled body in the innermost;
Results Summary

By application and dataset

- **Yales2**
  - 3D Cylinder - 10% (LCT), 14% (LCT+IPGO)
  - 1D Coffee - 4% (LCT), 6% (LCT+IPGO)

- **AVBP**
  - SIMPLE - 1% (LCT), 12% (SVT)
  - NASA - 8% (LCT), 24% (SVT)
  - TPF - 3% (LCT), 9% (SVT)

- **POLARIS**
  - test.1.0.5.18 - 4% (SVT)

- **CNN**
  - all layers - 50%-550%

- **QMCPACK**
  - k64s64 - 5%(FU), 8%(SPLIT), 13%(FUME)
Outline

1. Background
2. ASSIST
3. Issues & Limitations
4. Experiments
5. Conclusion
Conclusion

Contributions

- Good gains on real-world applications
- Novel study of how and when well-known transformations work
- Novel semi-automatic & user controllable method with a system open to user advice and to all kinds of users
- An FDO tool which can use both static and dynamic analysis information to guide code optimization
- A flexible alternative to current compilers PGO/FDO modes.

Available on github

https://youelebr.github.io/ (maqao binary, assist sources, test suite and documentation)
Conclusion

Perspectives

- Extend MAQAO analysis with source information
- Add new transformations or extend existing ones (i.e. Specialization)
- Find more metrics and how to associate them to know when to trigger a transformation
- Multiple datasets
- Auto-tuning with iterative compilation using our verification system
- Drive transformation for energy consumption and/or memory
Conclusion

Any questions?
Backup Slides
What is CQA

CQA: Code Quality Analyzer

- Goal: Assist developers in improving code performance
- Evaluate the quality of the compiler generated code
- Returns hints and workarounds to improve quality
- Static analysis (no execution / allows cross-analysis)

Main Concepts

- Relies on simplified CPU model (execution pipeline, port throughput, L1 data access)
- Key performance levers for core level efficiency:
  - Vectorizing
  - Avoiding high latency instructions
  - Having the compiler generate an efficient code
  - Reorganizing memory layout
What is Lprof

LProf: Lightweight Profiler
- Goal: Lightweight localization of application hotspots
- Dynamic analysis sampling base
- Access to hardware counters for additional information
- Results at function and loop granularity

Strengths
- Non intrusive: No recompilation necessary
- Low overhead
- Agnostic with regard to parallel runtime
What is Vprof

Vprof: Value Profiler

- Dynamic analysis tracing based
- Targets loops & functions
- Detection of stable values
- Loop characterization through number of iterations
- Provides leads for code specialization
What is DECAN

DECAN: DECremental ANalysis

- Goal: modify the application to identify causes of bottlenecks and estimate associated ROI
- Transformations:
  - Remove or modify groups of instructions
  - Targets memory accesses or computation

Typical Transformations

- FP: Only Floating Point arithmetic instruction are preserved (load and store are removed)
- LS: Only load and stores are preserved (compute instructions are removed)
- DL1: memory references replaced with global variables ones (data now accessed from L1)
Configuration File

File = "test_userconfig1.f90"

Arch = { -- contains all informations
    All = { -- All will always be called
        loops = { --For loops of the file
            -- Describe transformation
            { line = 30, transformation = {"TILE=5"} }
        }
    },
    -- specific target architecture
    -- call only when the user ask for it
    x86 = {
        loops = {
            { line = 26, transformation = {"TILE=5"} },
            { line = 34, transformation = {"UNROLL=8"} },
            { label = "LOOPLABEL1", transformation = {"INTERCHANGE=1,2"} }
        }
    }
}
oneview_global_metrics = {
    total_time_s = 43.26,
    compilation_options = "OK. \n".."",
    flow_complexity = 1.00,
    array_efficiency = 37.67,
    speedup_if_clean = 1.00,
    nb_loop_80_if_clean = 1,
    speedup_if_fp_vect = 1.00,
    nb_loop_80_if_fp_vect = 1,
    speedup_if_fully_vect = 7.14,
    nb_loop_80_if_fully_vect = 1,
    speedup_if_11 = 2.14,
    nb_loop_80_if_11 = 1,
}

oneview_cleaning_report = {
    {
        loop_id = 5,
        lineStart = 5,
        lineStop = 7,
        file = "/home/tests/oneview_test/tuto/codelet.c",
        ite_min = 1000000,
        ite_max = 1000000,
        ite_avg = 1000000,
        r_11_min = 2.361754348463,
        r_11_max = 2.0752602660095,
        r_11_med = 2.3972978247604,
        vecRatio = 12.5,
    },
}
Other results: prefetcher behavior with parallelism

![Graph showing speed-up over all on prefetchers for different MSR configurations with 1, 2, and 4 processes.]
Other results: prefetcher behavior at function level