PhD Defense

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

Youenn Lebras

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

UNIVERSITÉ DE VERSAILLES ST-QUENTIN-EN-YVELINES
université PARIS-SACLAY
Exascale computing research

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

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

. .

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

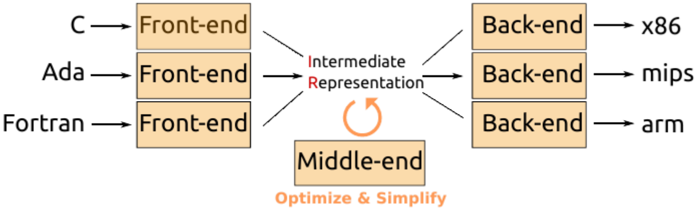
Goals

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



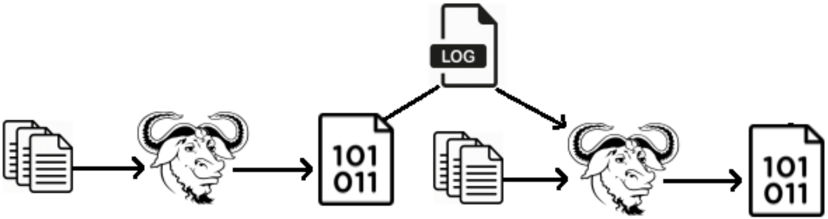
Compiler task

- 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 controle 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

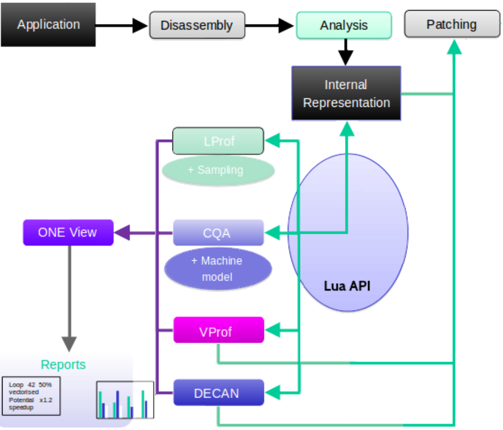
Contributions

- 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

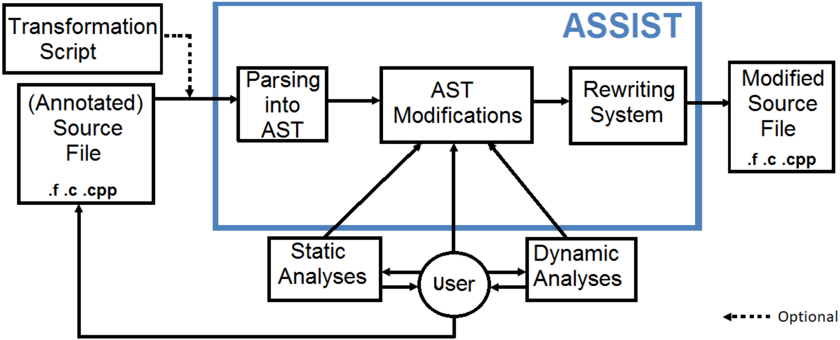


MAQAO Modules

- 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):



Automatic Source-to-Source assISTant (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

Zoom on SVT

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

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

(a) Before 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>
```

(b) After Short Vectorization

Zoom on SVT

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

(a) Before 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>
```

Loop decomposition

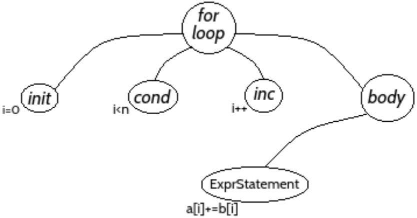
Residual

(b) After Short Vectorization

Transformation Example

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

(a) Source code



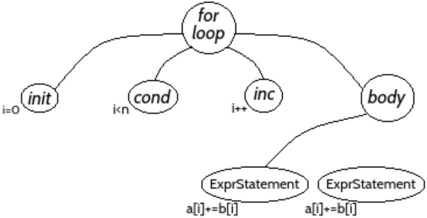
(b) Corresponding AST

Transformation Example

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

end for

(a) Source code



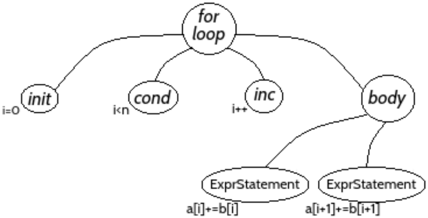
(b) Corresponding AST

Transformation Example

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

end for
```

(a) Source code



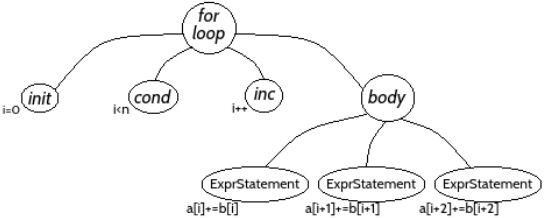
(b) Corresponding AST

Transformation Example

```
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

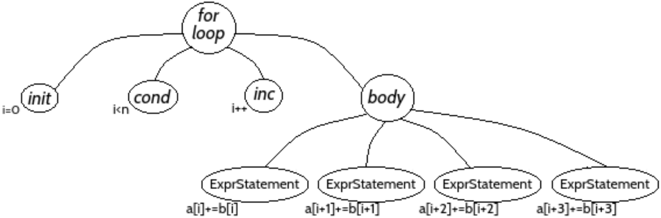


(b) Corresponding AST

Transformation Example

```
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

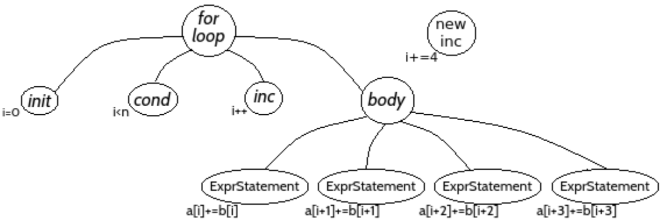


(b) Corresponding AST

Transformation Example

```
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

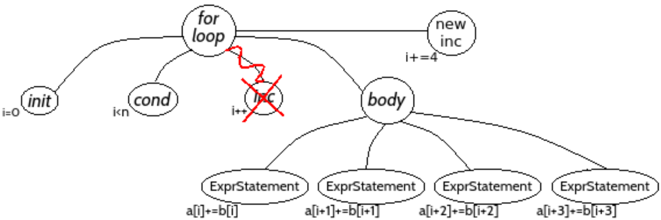


(b) Corresponding AST

Transformation Example

```
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

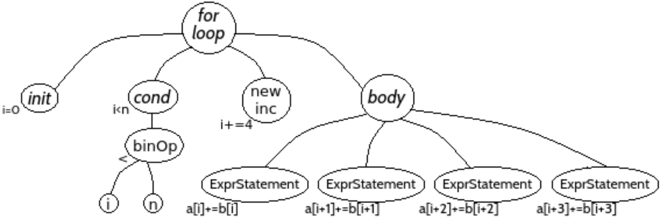


(b) Corresponding AST

Transformation Example

```
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

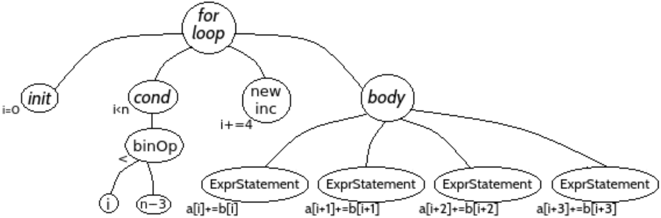


(b) Corresponding AST

Transformation Example

```
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 => CQA (vectorization ratio) + VPROF (iteration count)
  - Tiling => DECAN (DL1)
  - Loop count => 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 => CQA (vectorization ratio) + VPROF (iteration count)
  - Tiling => DECAN (DL1)
  - Loop count => 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 => CQA (vectorization ratio) + VPROF (iteration count)
  - Tiling => DECAN (DL1)
  - Loop count => 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

GLOBAL METRICS	BEFORE TRANSFO	AFTER TRANSFO	IS BETTER ?
Total Time (s) :	2.34	1	
Flow Complexity :	1	1	lower is better
Array Access Efficiency (%) :	71.95	54.66	higher is better
Speedup if clean :	1	1.04	lower is better
Nb loops to get 80% if clean :	1	1	lower is better
Speedup if FP Vectorised :	1	1.47	lower is better
Nb loops to get 80% if FP vectorized :	1	1	lower is better
Speedup if Fully Vectorised :	1.66	1.87	lower is better
Nb loops to get 80% if Fully vectorized :	3	2	lower is better
Speedup if data in L1 Cache :	1	1.18	lower is better
Nb loops to get 80% if data in L1 Cache :	0	2	lower is better
Compilation Options :	OK	OK	
Loop lines :	314 - 314	881-884	
Loop_id :	74	246	
Speedup if clean :	1.00	1.06	lower is better
Speedup if fully vectorized :	1.62	2.06	lower is better
Bottlenecks :	p5	P0,01	
Unroll confidence level :	max	max	
Cycles L1 if clean :	13	8.00	higher is better
Cycles L1 if fully vec :	8	2.06	higher is better
Vector-efficiency ratio all :	85.71	42.00	higher is better
Vectorization ratio all :	67.86	68.00	higher is better
FP op per cycle L1 :	2.46,	3.76	higher is better

Outline

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

Experiments

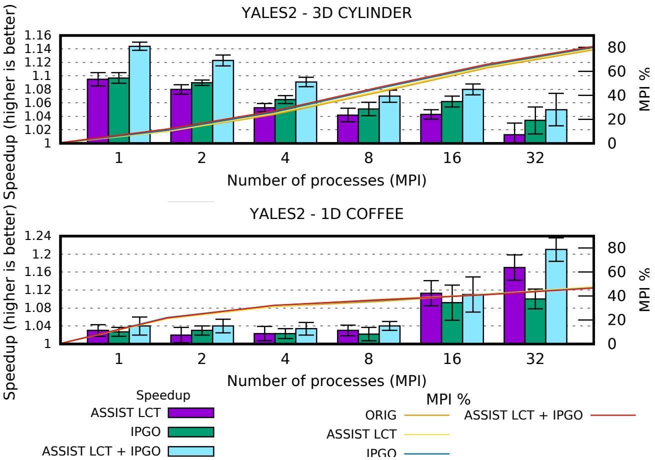
Results have been obtained on a Skylake Server and are compiled with Intel 17.0.4 and compared to Intel PGO version 17.0.4 (IPGO)

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

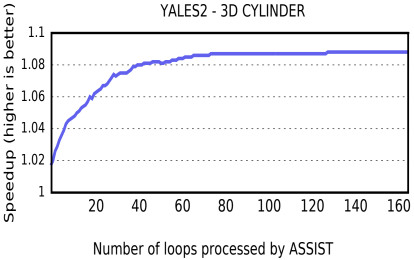
Impact of the Loop Count

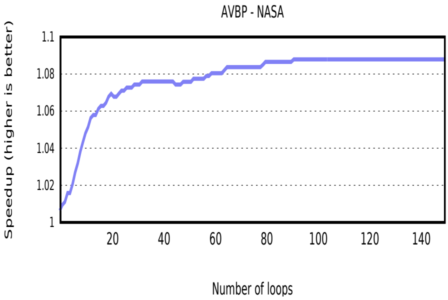
Comparision with IPGO and ASSIST LCT + IPGO.



Impact of the Loop Count

	AVBP NASA	AVBP TPF	AVBP SIMPLE	Yales2 3D Cylinder	Yales2 1D COFFEE
Number of loops	149	173	158	162	122





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

Impact of Specialization

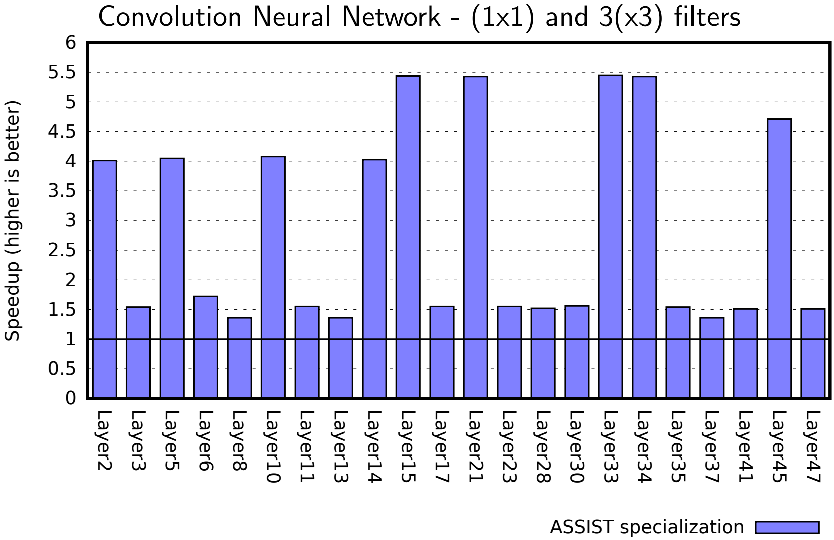
SRC ORIG

```
for (img = 0; img < nImg; ++img) {
  for (ifm = 0; ifm < nIfm; ++ifm) {
    for (ofm = 0; ofm < nOfm; ++ofm) {
      for (oj = 0; oj < ofh; ++oj) {
        ij = 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

```
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) {
          ij = 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];
              }
            }
          }
        }
      }
    }
  }
} else {
  <Original Body>
}
```

Impact of 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
  ...
```

Name	Coverage (%)	Time (s)
▼ gather_o_cpy	13.67	21.18
▼ Loop 16183 - gather_o_cpy.f90:197-201 - AVBP_V7.0.1_orig	0.27	0.42
○ Loop 16182 - gather_o_cpy.f90:198-201 - AVBP_V7.0.1_orig	3.83	5.93
▼ Loop 16181 - gather_o_cpy.f90:197-201 - AVBP_V7.0.1_orig	0.23	0.36
○ Loop 16180 - gather_o_cpy.f90:198-201 - AVBP_V7.0.1_orig	2.32	3.59
▼ Loop 16178 - gather_o_cpy.f90:197-201 - AVBP_V7.0.1_orig	0.09	0.14
○ Loop 16177 - gather_o_cpy.f90:198-201 - AVBP_V7.0.1_orig	0.58	0.9
▼ Loop 16194 - gather_o_cpy.f90:197-201 - AVBP_V7.0.1_orig	0.08	0.12
○ Loop 16193 - gather_o_cpy.f90:198-201 - AVBP_V7.0.1_orig	2.07	3.21
▼ Loop 16192 - gather_o_cpy.f90:197-201 - AVBP_V7.0.1_orig	0.07	0.11
○ Loop 16191 - gather_o_cpy.f90:198-201 - AVBP_V7.0.1_orig	1.82	2.82
▼ Loop 16214 - gather_o_cpy.f90:197-201 - AVBP_V7.0.1_orig	0	0
▼ Loop 16213 - gather_o_cpy.f90:198-201 - AVBP_V7.0.1_orig	0	0
○ Loop 16216 - gather_o_cpy.f90:200-201 - AVBP_V7.0.1_orig	0	0
○ Loop 16215 - gather_o_cpy.f90:200-201 - AVBP_V7.0.1_orig	0	0
○ Loop 16201 - gather_o_cpy.f90:197-201 - AVBP_V7.0.1_orig	0	0
○ Loop 16202 - gather_o_cpy.f90:197-201 - AVBP_V7.0.1_orig	0	0
▼ Loop 16218 - gather_o_cpy.f90:197-201 - AVBP_V7.0.1_orig	0	0
○ Loop 16217 - gather_o_cpy.f90:198-201 - AVBP_V7.0.1_orig	0	0
○ Loop 16179 - gather_o_cpy.f90:197-201 - AVBP_V7.0.1_orig	0	0
▼ Loop 16212 - gather_o_cpy.f90:197-201 - AVBP_V7.0.1_orig	0	0
○ Loop 16211 - gather_o_cpy.f90:198-201 - AVBP_V7.0.1_orig	0	0

Impact of Specialization Combined with SVT

Function specialization

	Name	Coverage (%)	Time (s)
	▼ gather_o_cpy_flat_2d_assist_nverte4_nproductmod42	7.55	12.21
1	▼ Loop 9633 - gather_o_cpy.f90:314-318 - AVBP_V7.0.1_speF	0.36	0.58
	▼ Loop 9632 - gather_o_cpy.f90:315-318 - AVBP_V7.0.1_speF	2	3.23
	○ Loop 9635 - gather_o_cpy.f90:317-318 - AVBP_V7.0.1_speF	3.79	6.13
	○ Loop 9634 - gather_o_cpy.f90:317-318 - AVBP_V7.0.1_speF	1.37	2.21
	▼ gather_o_cpy_flat_2d_assist_nverte4_nproductmod41	3.92	6.34
2	▼ Loop 9637 - gather_o_cpy.f90:288-292 - AVBP_V7.0.1_speF	0.27	0.44
	▼ Loop 9636 - gather_o_cpy.f90:289-292 - AVBP_V7.0.1_speF	1.51	2.44
	○ Loop 9639 - gather_o_cpy.f90:291-292 - AVBP_V7.0.1_speF	1.7	2.75
	○ Loop 9638 - gather_o_cpy.f90:291-292 - AVBP_V7.0.1_speF	0.43	0.7
	▼ gather_o_cpy	2.88	4.66
4	○ Loop 9567 - gather_o_cpy.f90:155-158 - AVBP_V7.0.1	2.17	3.51
5	▼ Loop 9569 - gather_o_cpy.f90:339-343 - AVBP_V7.0.1_speF	0.04	0.06
	○ Loop 9568 - gather_o_cpy.f90:340-343 - AVBP_V7.0.1_speF	0.58	0.94
	▼ gather_o_cpy_flat_2d_assist_nverte4	2.14	3.46
3	▼ Loop 9645 - gather_o_cpy.f90:235-239 - AVBP_V7.0.1_speF	0.07	0.11
	▼ Loop 9644 - gather_o_cpy.f90:236-239 - AVBP_V7.0.1_speF	0.39	0.63
	○ Loop 9647 - gather_o_cpy.f90:238-239 - AVBP_V7.0.1_speF	1.28	2.07
	○ Loop 9646 - gather_o_cpy.f90:238-239 - AVBP_V7.0.1_speF	0.4	0.65

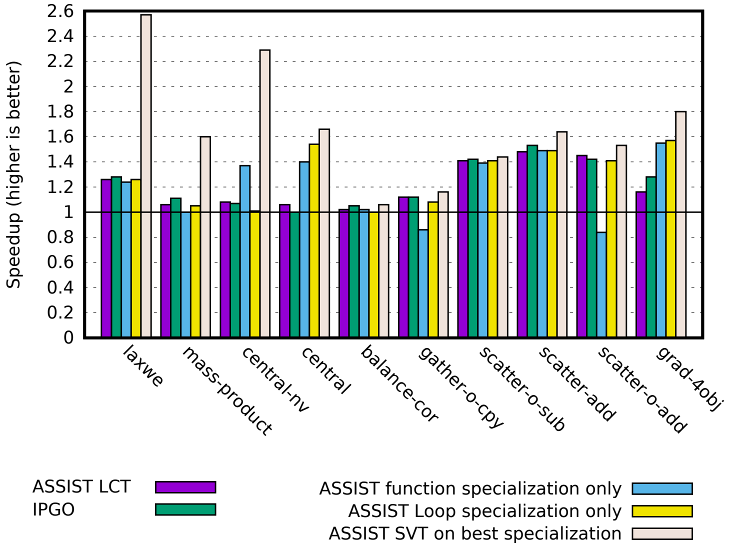
Loop specialization

	Name	Coverage (%)	Time (s)
	▼ gather_o_cpy	13.91	21.31
4	○ Loop 9465 - gather_o_cpy.f90:165-168 - AVBP_V7.0.1_speL	2.21	3.39
1	▼ Loop 9473 - gather_o_cpy.f90:223-227 - AVBP_V7.0.1_speL	0.29	0.44
	○ Loop 9472 - gather_o_cpy.f90:224-227 - AVBP_V7.0.1_speL	3.87	5.93
2	▼ Loop 9471 - gather_o_cpy.f90:214-218 - AVBP_V7.0.1_speL	0.21	0.32
	○ Loop 9470 - gather_o_cpy.f90:215-218 - AVBP_V7.0.1_speL	2.55	3.91
1	▼ Loop 9486 - gather_o_cpy.f90:223-227 - AVBP_V7.0.1_speL	0.09	0.14
	○ Loop 9485 - gather_o_cpy.f90:224-227 - AVBP_V7.0.1_speL	2.05	3.14
3	▼ Loop 9484 - gather_o_cpy.f90:241-245 - AVBP_V7.0.1_speL	0.08	0.12
	○ Loop 9483 - gather_o_cpy.f90:242-245 - AVBP_V7.0.1_speL	1.78	2.73
5	▼ Loop 9468 - gather_o_cpy.f90:232-236 - AVBP_V7.0.1_speL	0.07	0.11
	○ Loop 9467 - gather_o_cpy.f90:233-236 - AVBP_V7.0.1_speL	0.59	0.9

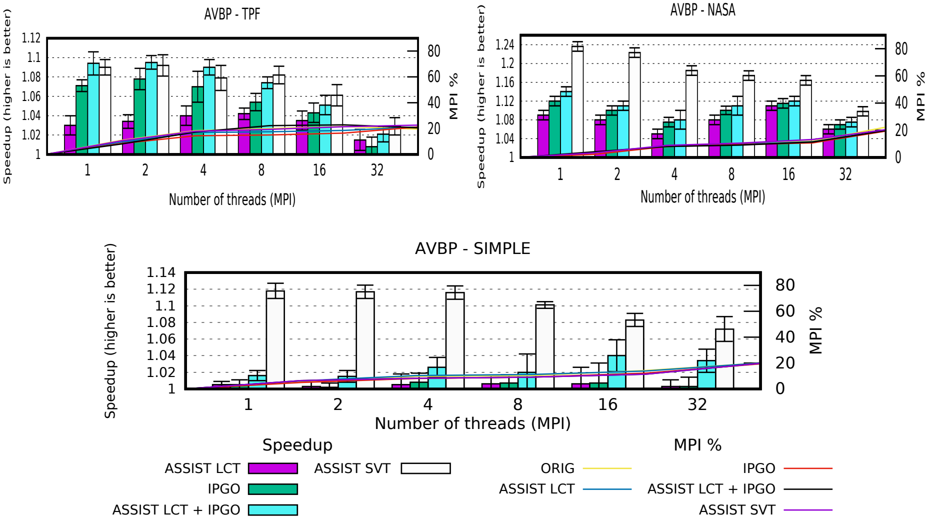
Best specialization + Short Vectorization Transformation

	Name	Coverage (%)	Time (s)
	▼ gather_o_cpy	14.81	20.51
1	○ Loop 16014 - gather_o_cpy.f90:226-236 - AVBP_V7.0.1_SVT	3.52	4.87
2	○ Loop 16013 - gather_o_cpy.f90:215-222 - AVBP_V7.0.1_SVT	3.09	4.28
4	○ Loop 16009 - gather_o_cpy.f90:165-168 - AVBP_V7.0.1_SVT	2.51	3.48
1	○ Loop 16021 - gather_o_cpy.f90:226-236 - AVBP_V7.0.1_SVT	2.46	3.41
3	○ Loop 16020 - gather_o_cpy.f90:252-265 - AVBP_V7.0.1_SVT	2.45	3.39
5	○ Loop 16011 - gather_o_cpy.f90:241-248 - AVBP_V7.0.1_SVT	0.66	0.91

Impact of Specialization Combined with SVT



Impact of Specialization Combined with SVT



Impact of Specialization Combined with SVT

	Execution Time before trans (sec)	Execution Time after trans (sec)	Speedup	Coverage (orig version)
Polaris	73.32	70.26	**1.04**	
loop 6909	4.27	3.14	**1.36**	5.72%
loop 6911	3.64	2.36	**1.54**	4.98%

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

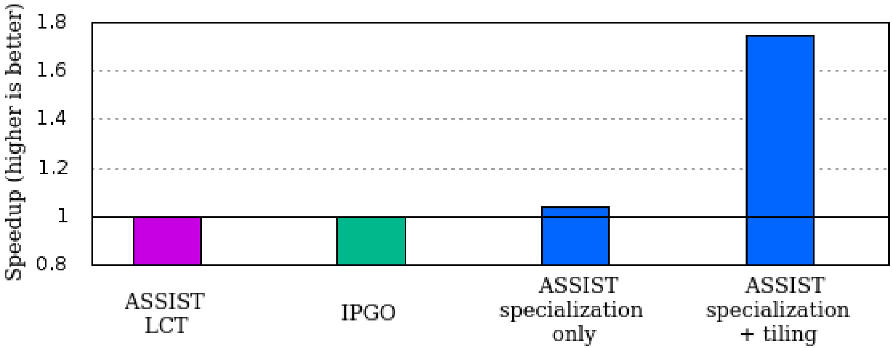
```
!DIR$ MAQAO SPECIALIZE(choice=1,paw_opt=3,cplex=2)
!DIR$ MAQAO SPECIALIZE(choice=1,paw_opt<3,cplex=2)
!DIR$ MAQAO SPECIALIZE(choice=1,paw_opt>3,cplex=2)
subroutine opernlb_ylm(choice,cplex,paw_opt,...)
 ...
 if(choice==1) then
  !DIR$ MAQAO IF_SPE_choicee1_TILE_INNER=8
  do ilmn=1, nlmn
   do k=1, npw
    z(k)=z(k)+ffnl(K,1,ilmn)*cplx(gxf(1,ilmn) &
         ,gxf(2,ilmn),kind=dp)
   end do
  end do
 end if
 ...
end subroutine
```

(a) Before specialization + tiling

```
SUBROUTINE opernlb_ylm(...)
 IF ((choice.EQ.1).AND.(paw_opt.EQ.3)/aND(cplex.EQ.2)) then
  CALL opernlb_ylm_ASSIST_choicee1_paw_opte3_cplexe2(...)
  RETURN
 END IF
 IF ((choice.EQ.1).AND.(paw_opt.LT.3)/aND(cplex.EQ.2)) then
  CALL opernlb_ylm_ASSIST_choicee1_paw_opte3_cplexe2(...)
  RETURN
 END IF
 IF ((choice.EQ.1).AND.(paw_opt.GT.3)/aND(cplex.EQ.2)) then
  CALL opernlb_ylm_ASSIST_choicee1_paw_opte3_cplexe2(...)
  RETURN
 END IF
 ...
END SUBROUTINE
...
SUBROUTINE opernlb_ylm_ASSIST_choicee1_paw_opte3_cplexe2(...)
 ...
 lt_bound_npw = (npw / 8) * 8
 DO lv_var_k = 1, lt_bound_npw, 8
   DO ilmn = 1, ilmn
     DO k = lt_var_k, lt_var_k + (8-1)
      z(k)=z(k)+ffnl(K,1,ilmn)*cplx(gxf(1,ilmn) &
          ,gxf(2,ilmn),kind=dp)
     END DO
   END DO
 END DO
 ...
END SUBROUTINE
SUBROUTINE opernlb_ylm_ASSIST_choicee1_paw_opti3_cplexe2(...)
 ...
END SUBROUTINE
SUBROUTINE opernlb_ylm_ASSIST_choicee1_paw_opts3_cplexe2(...)
 ...
END SUBROUTINE
```

(b) After specialization + tiling

Impact of Specialization Combined with Tiling



	# lines of code	Execution time (sec)	Speedup
original version	716	2.55	1
assist version	1338	1.47	1.75

Other results: QMCPACK other transformations

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

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

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 regar 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

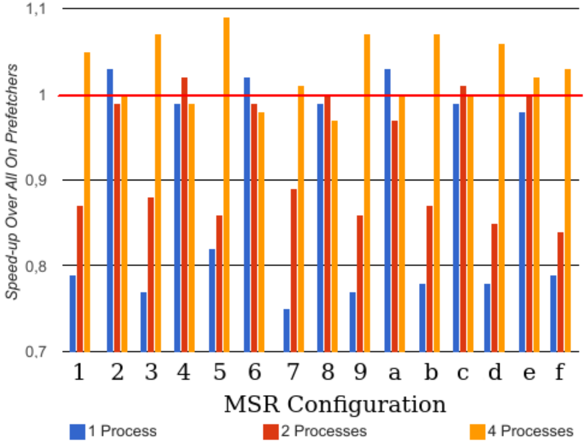
```
File = "test_userconfig1.f90"

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 Report

```
oneview_global_metrics = {
  total_time_s = 43.26,
  compilation_options = "OK. \n".."",
  flow_complexity = 1.00,
  array_efficieny = 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_l1 = 2.14,
  nb_loop_80_if_l1 = 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_l1_min   = 2.361754348463,
    r_l1_max   = 2.0752602660095,
    r_l1_med   = 2.3972978247604,
    vecRatio   = 12.5,
  },
}
```

Other results: prefetcher behavior with parallelism



Other results: prefetcher behavior at function level