ASSIST: Using performance analysis tools for driving Feedback Directed Optimizations

Youenn Lebras (PhD Thesis)
Advisor William Jalby,
Co-supervisor: Andres S. Charif-Rubial
UVSQ/ECR

Hardware trends and consequences

- The performance model shifted from high frequency single core processors to multitasking high-core-count parallel architectures
- Larger vector lengths (AVX512)
- Specialized units (FMA, …)
- New memory technology (HBM, Optane)

CONSEQUENCES

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

- Optimizing code has become an art
- Codes are harder and harder to optimize and maintain manually
- Optimization is Time consuming and error-prone

Optimizing compilers

- ✚ Transparent for the user (no effort required) and code unmodified
- ✚ Can be improved through user inserted directives
- ▬ Remains conservative (static performance cost models & heuristics)
- ▬ Limited search space for optimizations (compilation time)
- ▬ Black Box : can ignore user directives ☺

An interesting alternative: Profile Guided Optimizations/Feedback Directed Optimizations

THREE STEPS PROCESS:

- 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

FDO/PGO

- ✚ Gets dynamic info on code behavior (stop shooting in the dark)
- ✚ Can implement well targeted optimizations
- ▬ Needs a first pass run or use continuous compilation (AutoFDO Google)
- ▬ Depends upon (often limited) info gathered during the profiling phase
- ▬ Data dependent

An interesting example: 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

ASSIST GOALS

Key idea: Performance analysis tools (e.g. Scalasca, MAQAO, Tau, Vtune, HPCToolkit, …) are pretty good at identifying some specific problems, but users do not want issues but solutions. We need to go further and try to fix automatically performance issues (at least some easy ones).

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 (interacts with the user)
- Keeps a maintainable code

MAQAO components provide two types of analysis

- Static: simple performance model and quantitative code quality assesment
- Dynamic: precise estimate of CPU versus memory bound information, accurate analysis of memory hierarchy (DL1 variant in which all of the data access are forced to be L1)

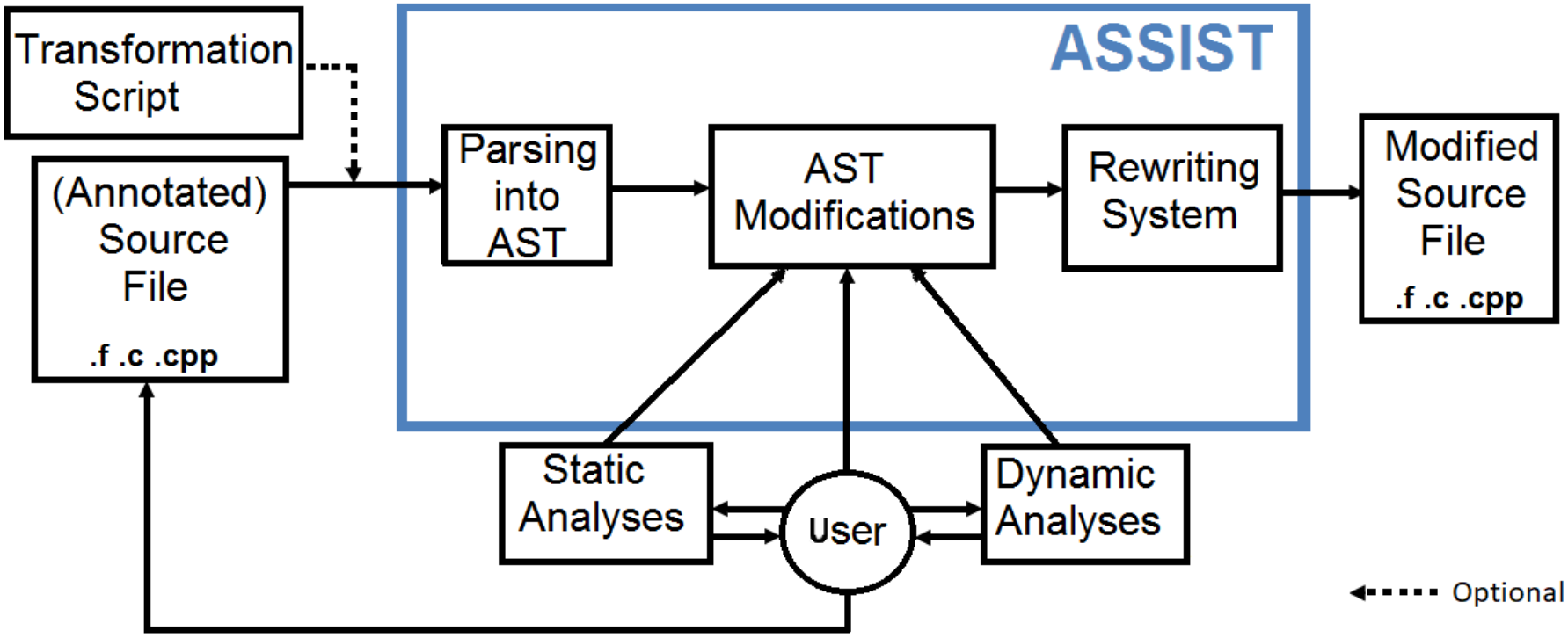
ONE VIEW (performance aggregator) provides analysis of code optimization opportunities

- Vectorization: Full and Partial
- Code quality
- CPU bound versus memory bound
- Blocking and array restructuring

Overview of Tool Usage

Automatic Source-to-source assISTant (ASSIST).

Staic and dynamic analysis are provided by MAQAO/ONE VIEW



- 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

Directive(s) Insertion

- Loop Count (LCT)
- Forcing Vectorization

AST Modifier (very classic transformations)

- Unroll
- Full Unroll
- Interchange
- Tile
- Strip Mine
- Loop/function Specialization

Combination of both

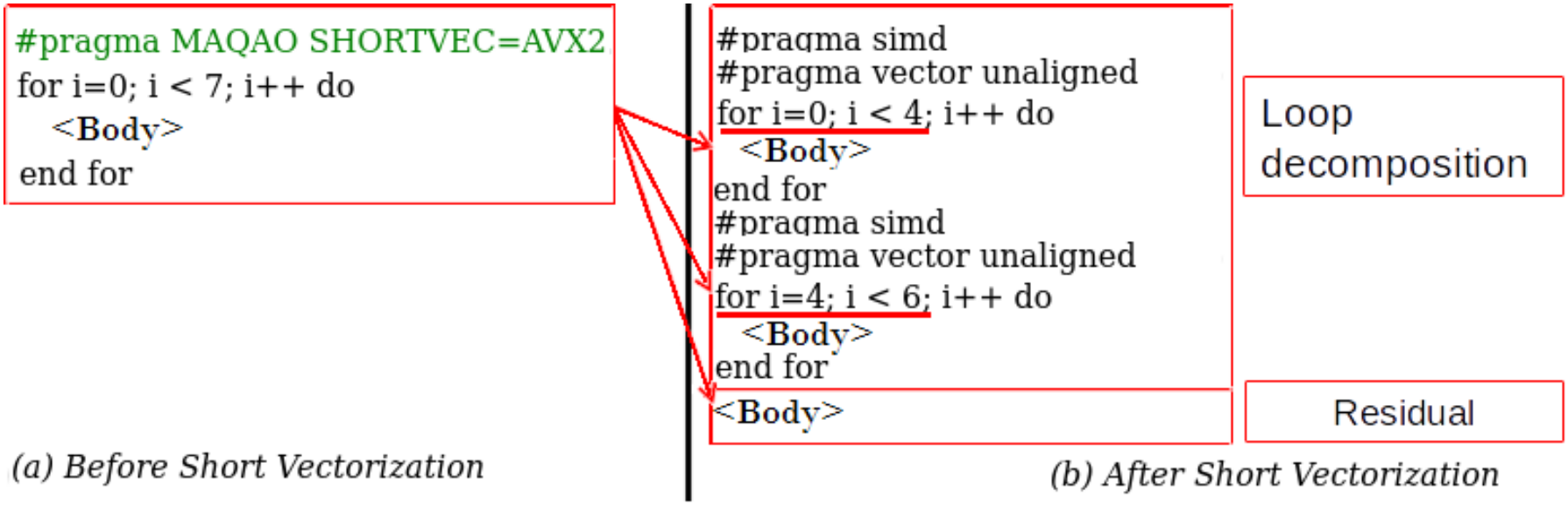
- Short Vectorization (SVT)

- 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: Mixed 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
#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
(a) Before Short Vectorization
(b) After Short Vectorization

Zoom on SPECIALIZATION

Spezialization is performed either at the function level or the loop level. Specialization proceeds in 3 steps

- ASSIST/ROSE identifies in the source code key integer variables: loop bounds, stride, involved in conditions, array index
- MAQAO/VPROF, at execution, profiles values of these variables and identifies the interesting ones with biased distributions: constant across all execution, very few values, a single very frequent value.
- ASSIST will then generate a specialised version of the function/loop

How to Enable Transformations

Two main approaches

- Under user full responsibility: insert directly directives in source code
- Use MAQAO report + User guidance(examples below)
  - CQA Vectorization Gain => Vectorization Directives
  - CQA (vectorization ratio) + VProf (iteration count) => SVT
  - DECAN (DL1) => Tiling
  - VProf (iteration count) => LCT

Additional approach: provide a transformation script, specifying transformations to be applied on a per source line number.

Assessing Transformation Verification

FIRST VERSION: STATIC ANALYSIS based on MAQAO/CQA

- Step 1: Perform static analysis using CQA on the target loop **BEFORE** transformation
- Step 2: Perform static analysis using CQA on the target loop **AFTER** transformation
- Step 3: Compare and decide.

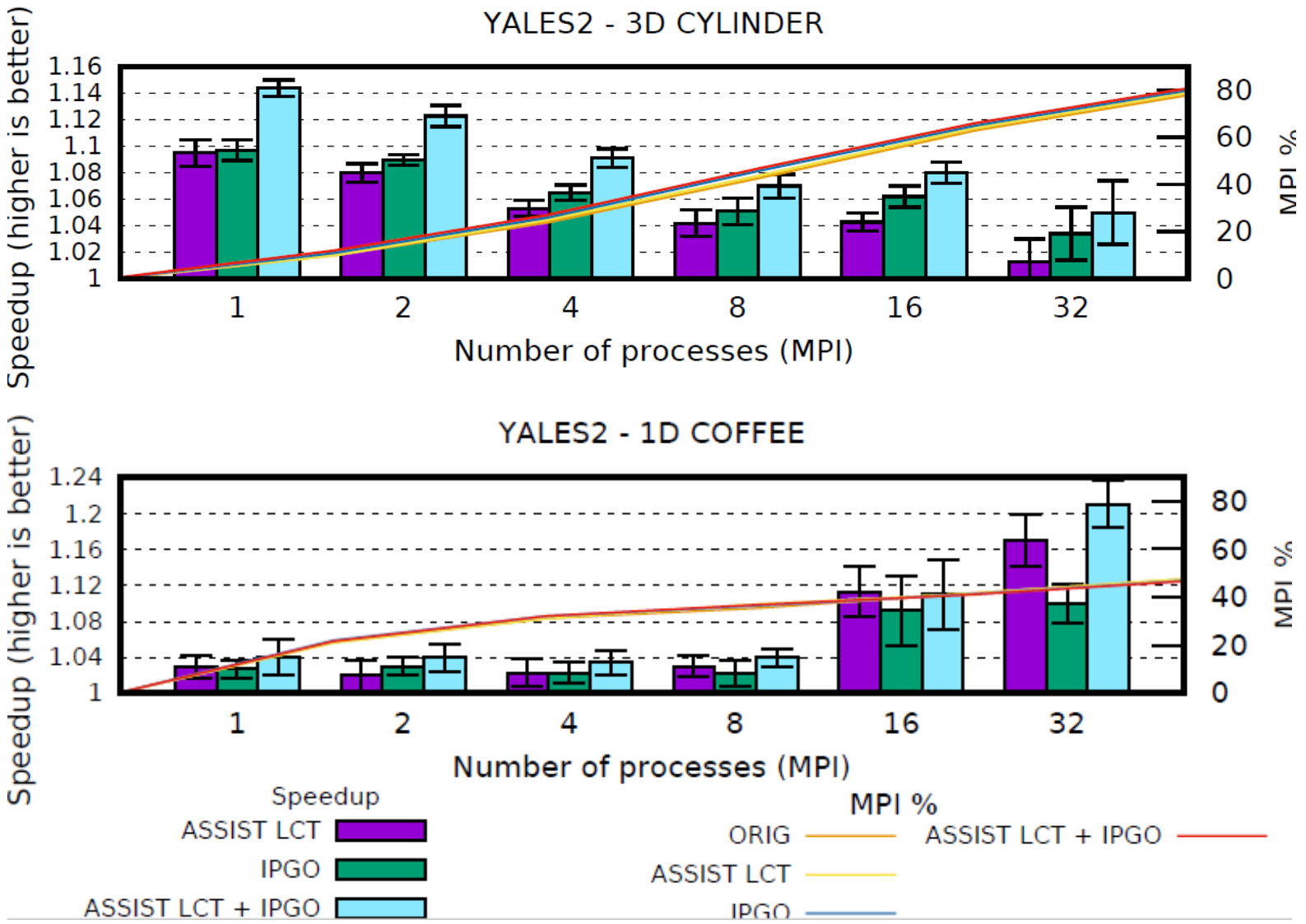
Results have been obtained on a Skylake Server and are compiled with Intel 17.0.4 and compared with Intel PGO version 17.0.4 (IPGO)

Application Pool

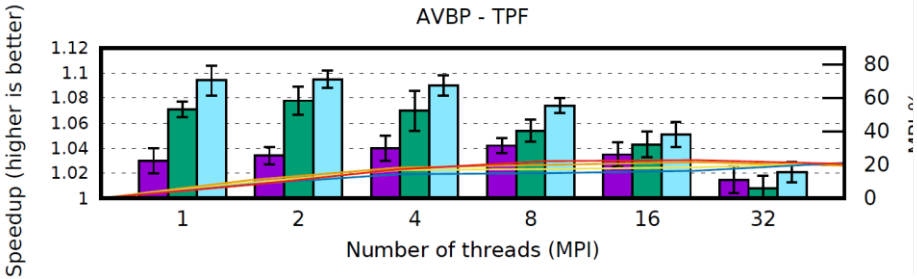
- **Yales2 (F03):** numerical simulation of turbulent reactive flows
- **AVBP (F95):** parallel computational fluid dynamics code
- **ABINIT (F90):** find the total energy charge density and the electronic structure of systems made of electrons and nuclei
- **POLARIS MD (F90):** microscopic simulator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

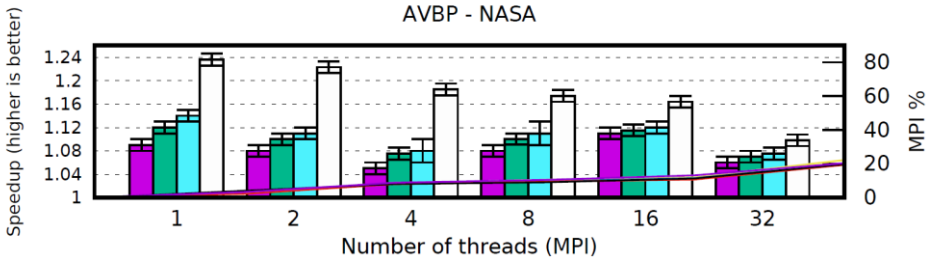
Impact of the Loop Count

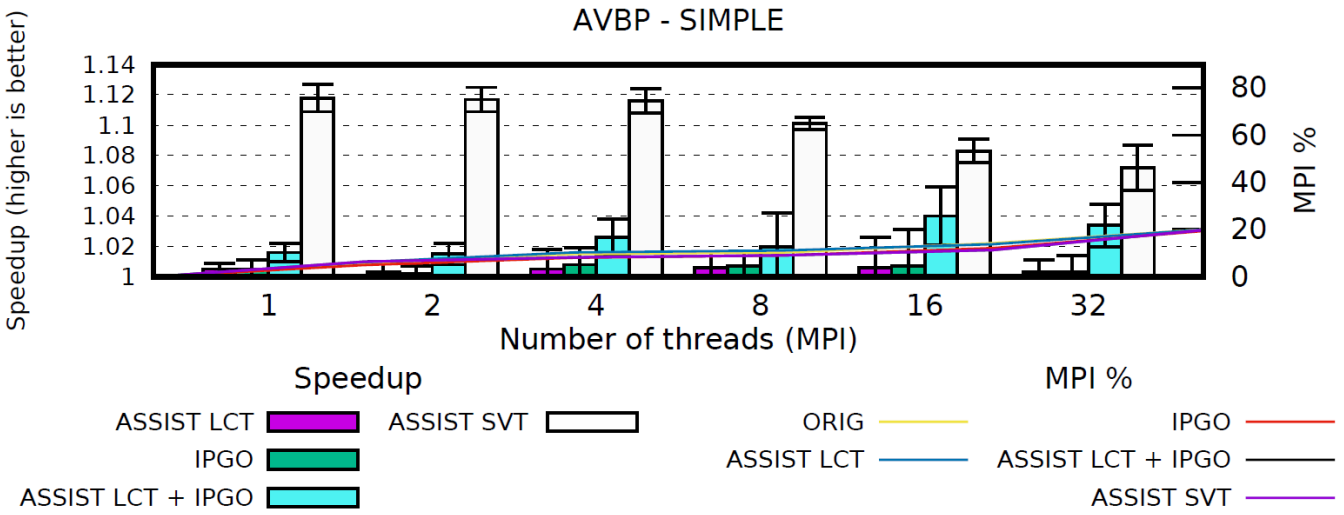
Comparison with IPGO and ASSIST LCT+IPGO



Impact of Specialization Combined with SVT

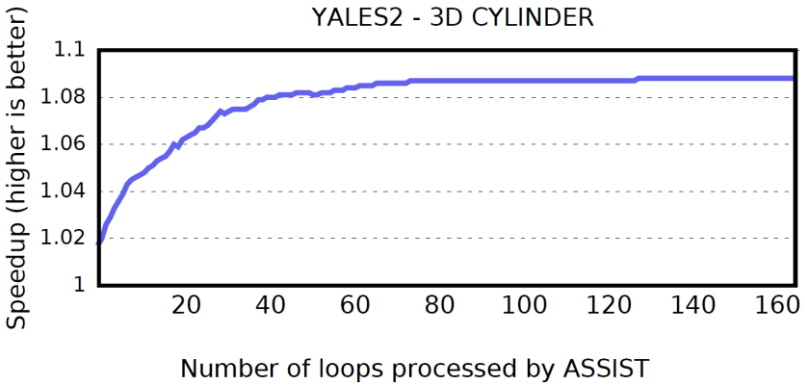


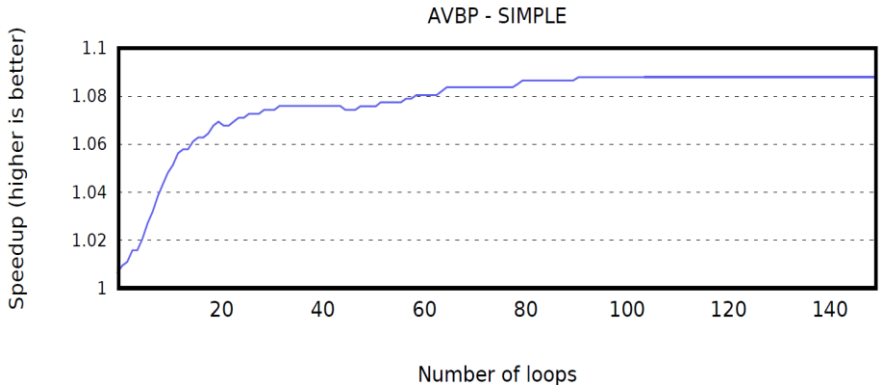




Number of loops processed

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



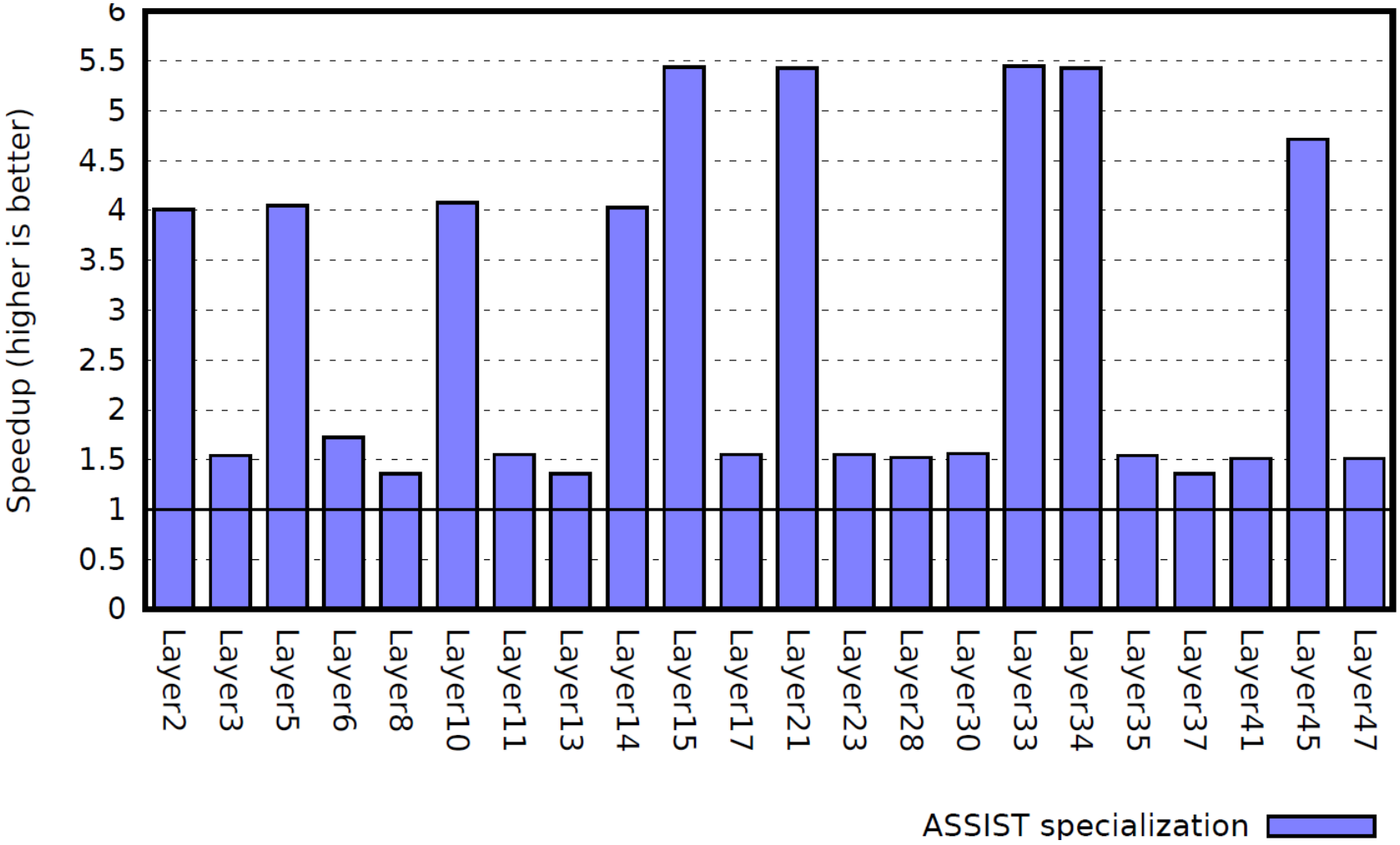


CNN: Impact of Specialization

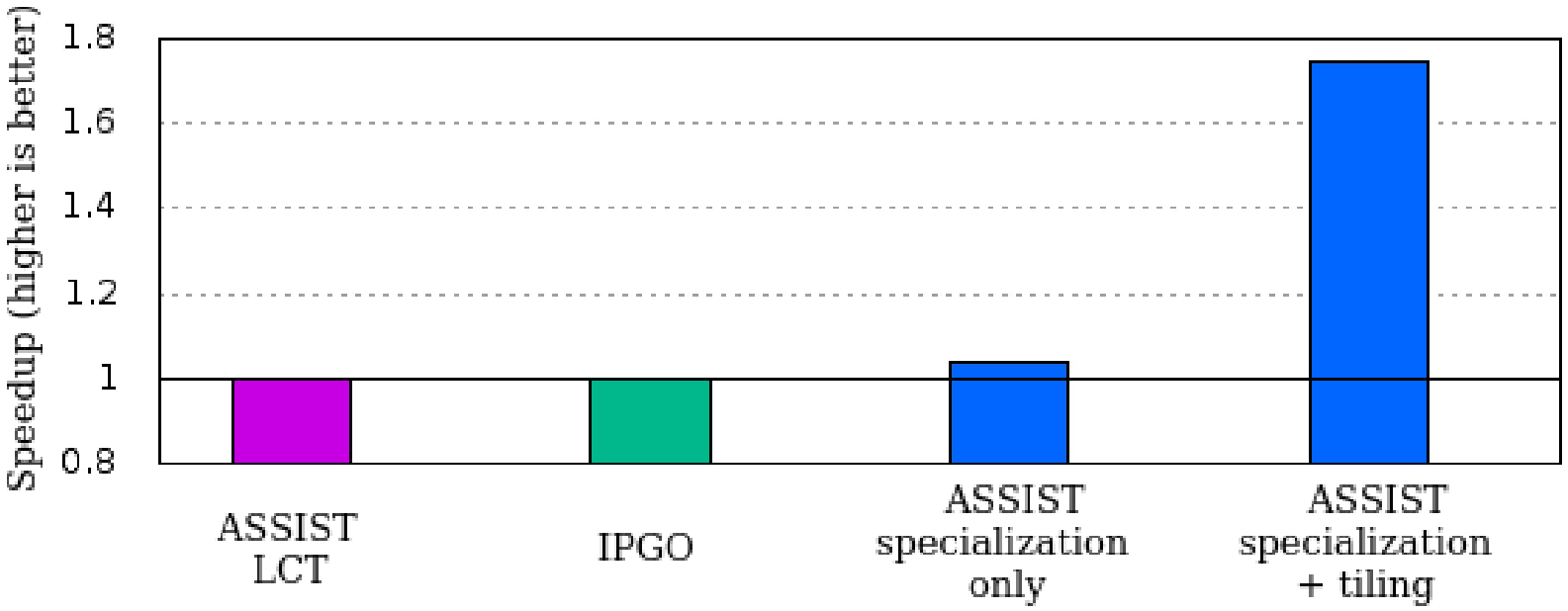
```
for (img = 0; img < nImg; ++img) {
  for (ifm = 0; ifm < nIfm; ++ifm) {
    for (ofm = 0; ofm < nOfm; ++ofm) {
      for (oj = 0; oj < ofh; ++oj) {
        ij = oj * stride_h - pad_h;
        for (oi = 0; oi < ofw; ++oi) {
          ii = oi * stride_w - pad_w;
          for (kj = 0; kj < kh; ++kj) {
            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];
            }
          }
        }
      }
    }
  }
}
```

```
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>
}
```

CNN: Impact of Specialization



Abinit: 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

- By application and dataset
  - Yales2
    - 3D Cylinder – 10% (LCT), 14% (LCT+IPGO)
    - 1D COFFEEE – 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%

Issues & Limitations

- **Analysis**
  - Debug information accuracy
  - What information to collect while limiting the overhead
- **Transformation**
  - Rose frontend/backend 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 split/duplicated, disappeared, etc)

Conclusion

Contributions

- Good gains on real-world applications
- New study of how and when well-known transformations work (such as LCT)
- New semi-automatic & user controllable method
- 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)**

Perspectives

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