Preliminary Study in Guiding Kernel SIMDization through Performance Modeling and Code Mockup Generation

Christopher Haine

Inria Runtime Team

January 14, 2014

Motivation (1)

SIMDization is necessary to obtain high performance.

- Explicit vectorization is time consuming.
- Use of compilers and tools is needed.

Compilers propose automatic vectorization.

- Quality of vectorized codes?
- A lot of source codes are too complicated for a compiler to vectorize.
- Hints about what caused vectorization to fail are hard to understand.

Examples with GCC 4.6.3:

```
tsc.c:1986: note: not vectorized, possible dependence between
            data-refs b[D.18123_4] and b[i_46] (->s1213)
tsc.c:121: note: not vectorized: complicated access pattern.
tsc.c:5476: note: not vectorized: relevant stmt not supported:
            *D.16720_10 = D.16721_11;
```

Motivation (2)

We proposed in a recent paper the automatic generation of vectorization hints

- Aumage, O., Barthou, D., Haine, C., Meunier, T.: *Detecting SIMDization Opportunities through Static-Dynamic Dependence Analysis.* Workshop on Productivity and Performance (2013)

Implemented in MAQAO software, a performance tuning tool.

However, as we will show, hints themselves are not sufficient.

- Hints are hints. They may be wrong.
- Even if they are correct, would it be worth to apply suggested transformations?

How to better guide the user through his vectorization process?

- Performance gain estimation is precious information.

Outline

Outline

MAQAO Overview

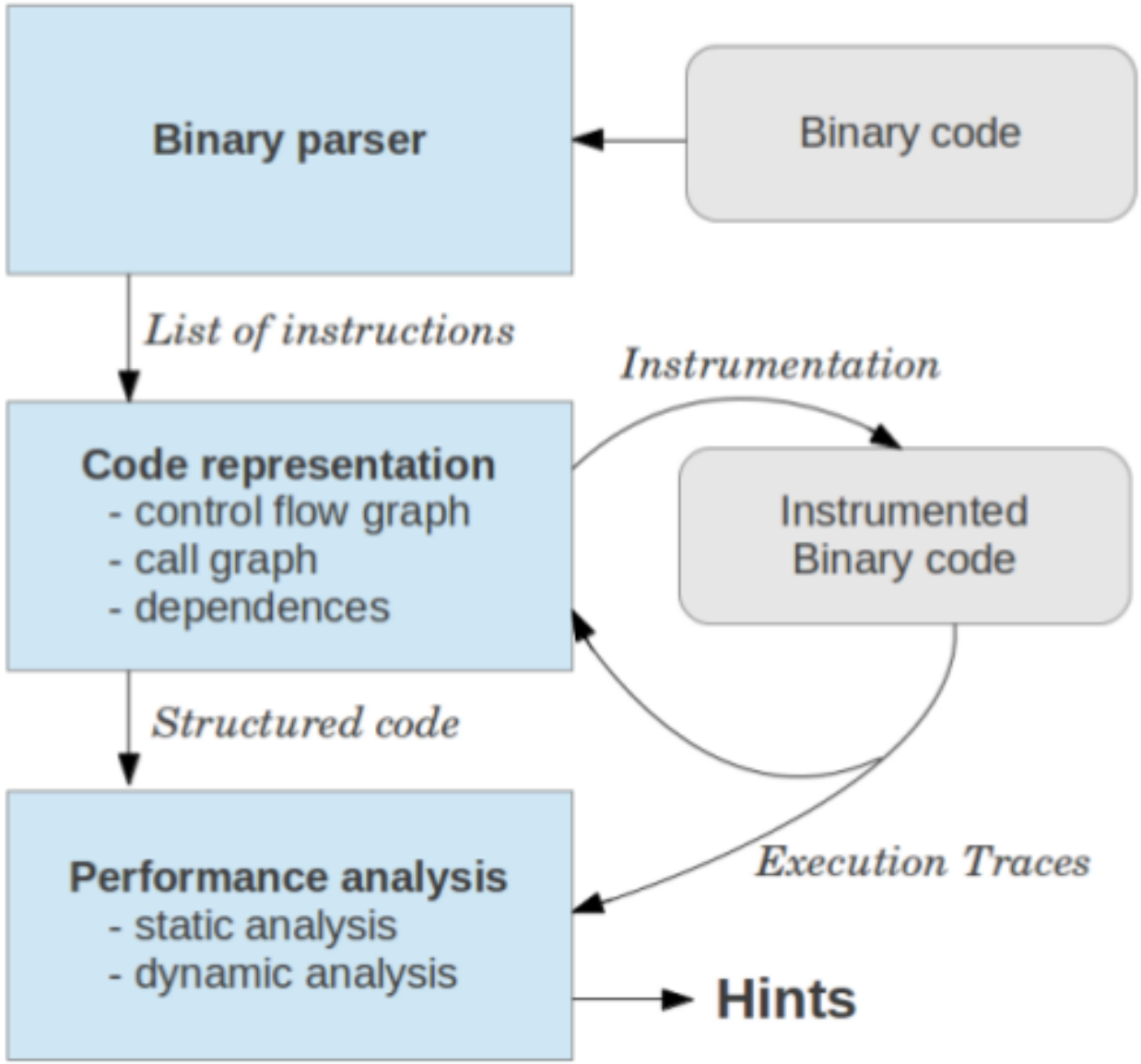

Figure : MAQAO Architecture

Dependence Graph

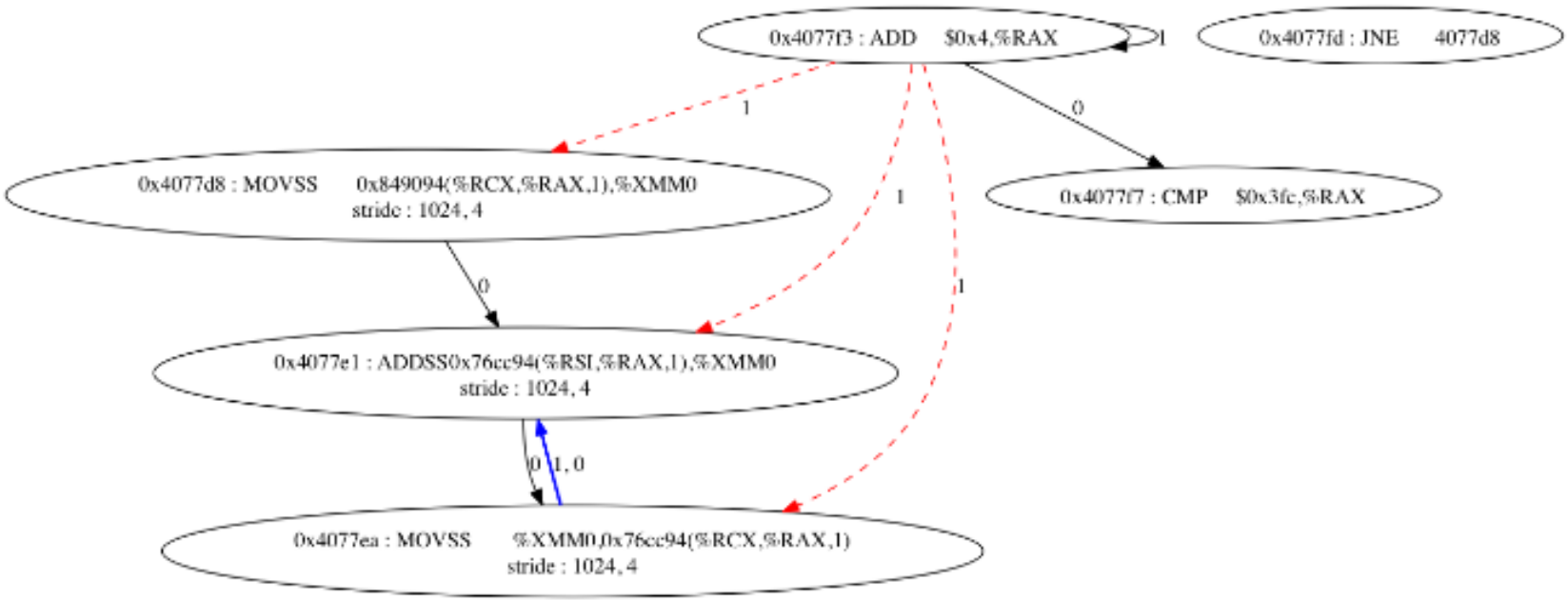

Figure : Dependence Graph of s2233 TSVC Benchmarks

Dependence Graph and Vectorization

The graph is said vectorizable if one of the three conditions applies to the computational part of the graph:

- There is no cycle.
- There is a cycle, with a cumulative weight greater than the width of SIMD vectors.
- There is a cycle, with a cumulative weight smaller than the width of SIMD vectors, and the instructions of the cycle all are of one of the following types: `add`, `mul`, `max`, `min`. The cycle corresponds to a reduction.

A code with a vectorizable dependence graph may require transformations in order to be SIMDizable.

Vectorization Hints

Hints for code transformations required for SIMDization are based on dependence graph, strides and control flow graph.

- Data alignment

  first address offset is not a multiple of the vector
- Loop transformations, loop interchange

  all accesses within loop have a large innermost loop stride
- Loop transformations, loop reversal

  strides in the wrong way, or negative strides
- Idiom Recognition

  simple pattern matching on dependence graph (dot product, mem copy)
- ...

Test Machine Specifications

All conducted experiments are based on ARMv7 architecture.

- ST-Ericsson Snowball board (ARM Cortex-A9)
  - 32-bit out-of-order processor @ 800Mhz
- NEON Advanced SIMD
  - Single precision floating-point arithmetic
  - up to 128-bit registers

Outline

Performance Model

First approach to predict performances

- Goal is to state an upper bound of performance
- The bound represents the underlying architecture limit

We model pipelines usage through a capacity model

- Intimate knowledge of the architecture is needed
- We rely on microbenchmarks to obtain accurate information

Microbenchmarks

Microbenchmarks are small portions of code

- They allow measurements of particular specifications
- This approach brings more accurate results than stated in documentations

Two main pieces of information are needed to build the performance model

- Number of pipelines
  - How many pipelines for a given instructions?
  - Is a given pipeline shared among several instructions?
- Cycles Per Instruction (CPI)
  - Number of cycles required to issue an instruction

Capacity Model (1)

- During the kernel execution, what determines performance the most is the busiest pipeline
- Accordingly, the model is expressed as follows:

$$t = max_{\pi}(\sum_{i=1}^{nins_{\pi}} CPI_i), \forall \pi \in \{pipelines\} \quad (1)$$

- Where t represents minimum time in cycles necessary for the execution of a loop kernel per iteration

Capacity Model (2)

- Reductions are important code patterns in scientific applications
  - Reductions are chains of dependences
  - Instructions latency is considered, as next instruction requires previous instruction result
- Model can be extended to include this constraint

$$t = max(max_{\pi}(\sum_{i=1}^{nins_{\pi}} CPI_i), max_{\chi}(\sum_{i=1}^{nins_{\chi}} latency_i + \rho_{CPI})),$$
$$\forall \pi \in \{pipelines\}, \forall \chi \in \{chains\} \quad (2)$$

Outline

Code Mockups

More reliable performance prediction of transformed loop kernel than performance models

- Code Mockups are designed to tend to the vectorized loop kernel behavior
  - Goal is to achieve similar performance to give reliable output to the user
  - Code Mockups are semantically diverging from basis loop kernel
- Two different classes of Code Mockups
  - Code Mockups of basis loop kernel
    - Naive vectorization
    - Other minor modifications e.g. unrolling, unroll&jam...
  - Code Mockups of restructured loop kernel
    - Major transformations e.g. data restructuration, loop interchange...

Example #1 - Alignment Issues

Example of Code Mockup application on TSVC Benchmark

- Study of s111 function
- MAQAO reported an alignment issue
  - Explain alignment issue...

```
for (int i = 1; i < LEN; i += 2) {
        a[i] = a[i - 1] + b[i];
}
```

Figure : Loop Kernel of s111 Function in TSVC Benchmark

Example #1 - Alignment Issues

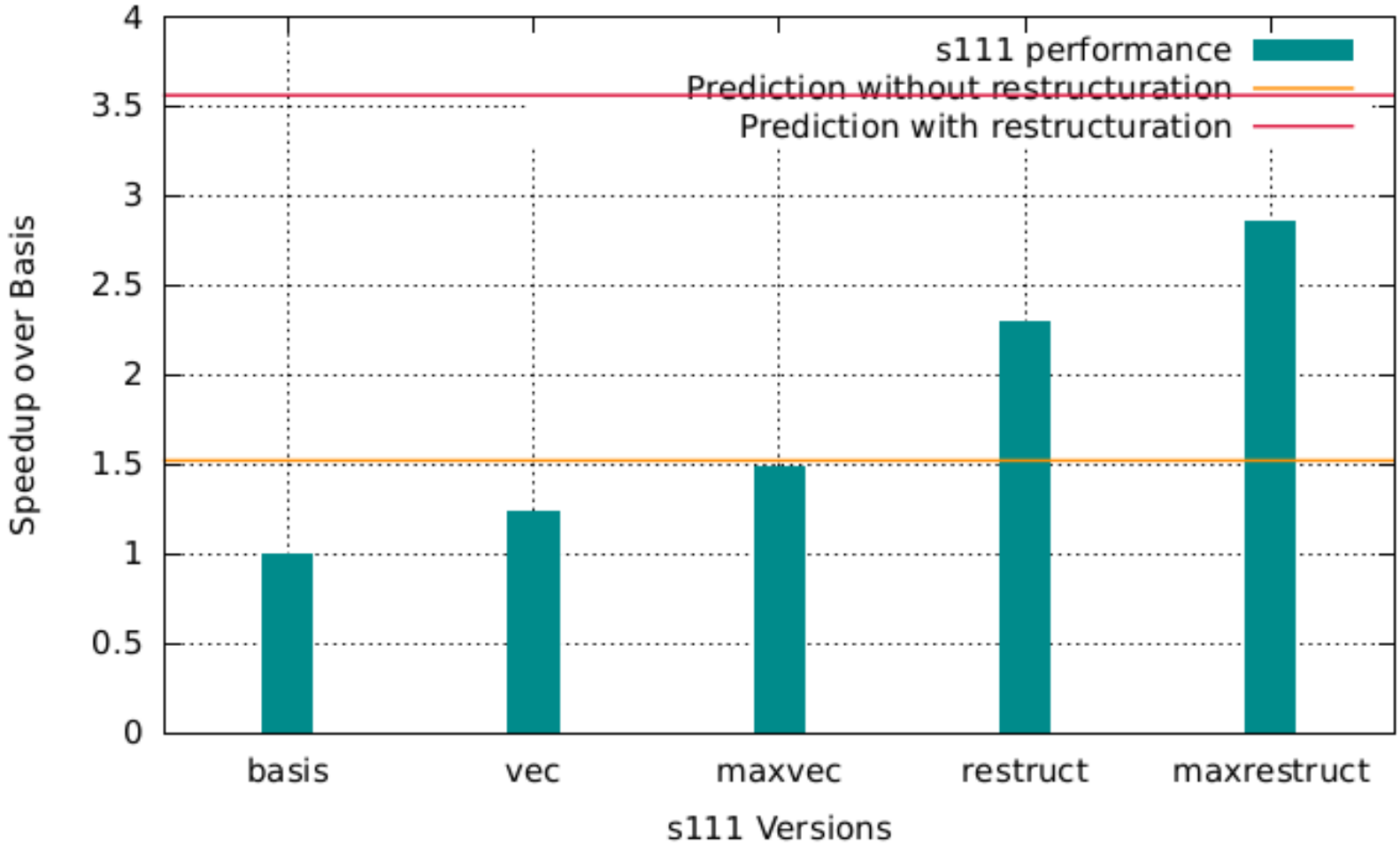

Figure : Output of s111 Function in TSVC Benchmark

Example #2 - Loop Interchange or Data Transpose

Other example of Code Mockup application on TSVC Benchmark

- Study of s1115 function
- MAQAO reported Loop interchange/Data Transpose issue
  - Explain...

```
for (int i = 0; i < LEN2; i++) {
    for (int j = 0; j < LEN2; j++) {
        aa[i][j] = aa[i][j]*cc[j][i] + bb[i][j];
    }
}
```

Figure : Loop Kernel of s1115 Function in TSVC Benchmark

Example #2 - Loop Interchange or Data Transpose

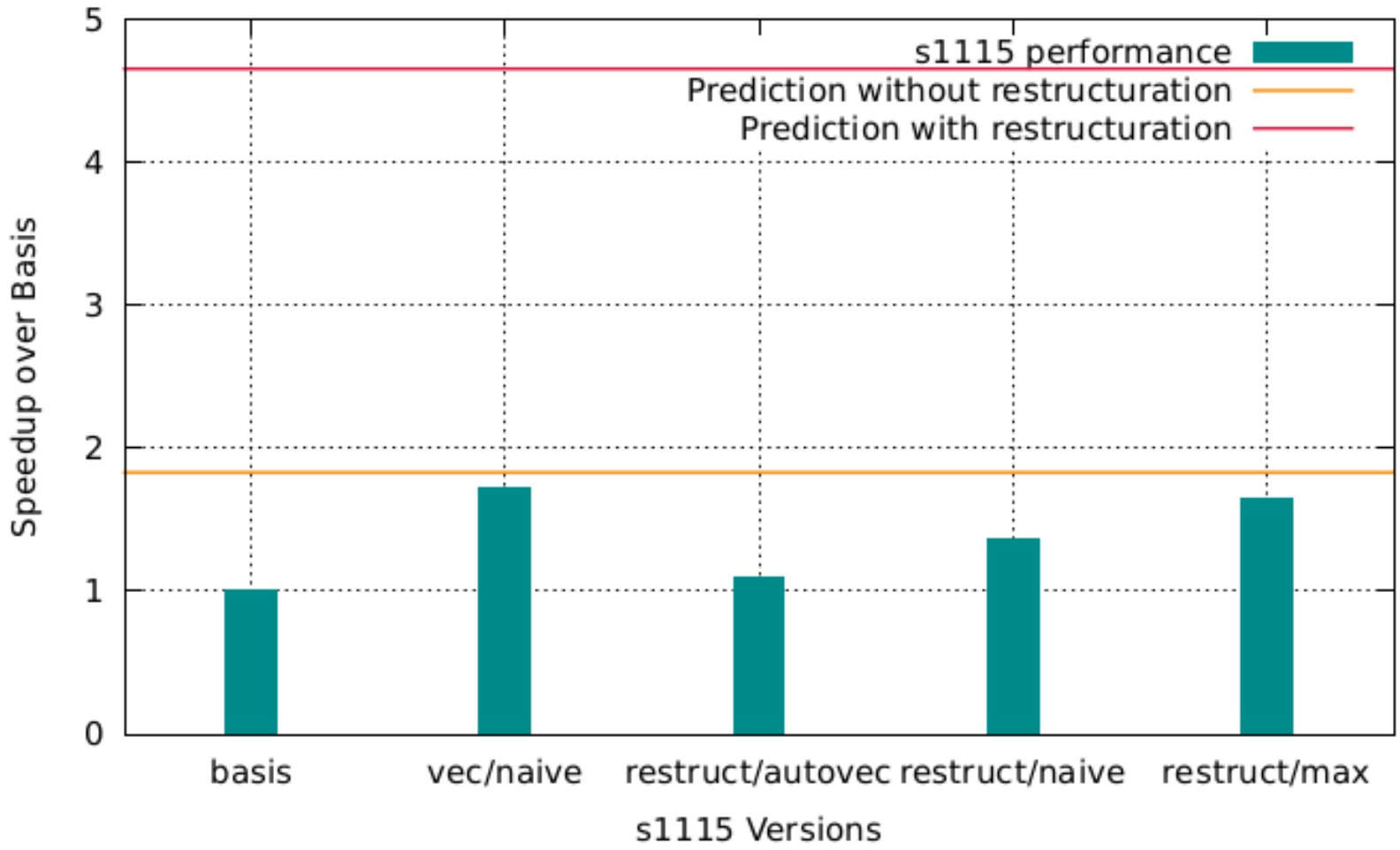

Figure : Output of s1115 Function in TSVC Benchmark

Outline

Case Study: Multigrid Kernel (1)

We use a modified version of the multigrid kernel part of the NPB 3.3 series of NAS benchmarks

```
for (k=1; k< N-1; k++) {
    for (j=1; j< N-1; j++) {
        for (i=0; i< N; i++) {
            u1[i] = u1[i] * u1[i];
            u2[i] = u2[i] * u2[i];
        }
        for (i=1; i< N-1; i++) {
            r[k][j][i] = u1[i-1] + u1[i+1]
                + u2[i-1] + u2[i] + u2[i+1];
        }
    }
}
```

Figure : Sample C Code of Modified Multigrid Benchmark

Case Study: Multigrid Kernel (2)

```
  > some base addresses not aligned on
'word size * vector width' bytes: data restructuration
needed for vectorization compatibility
  > some non contiguous strides:
loop/data restructuration needed for vectorization
```

Figure : MAQAO Report Excerpt on Multigrid Benchmark

Case Study: Multigrid Kernel (3)

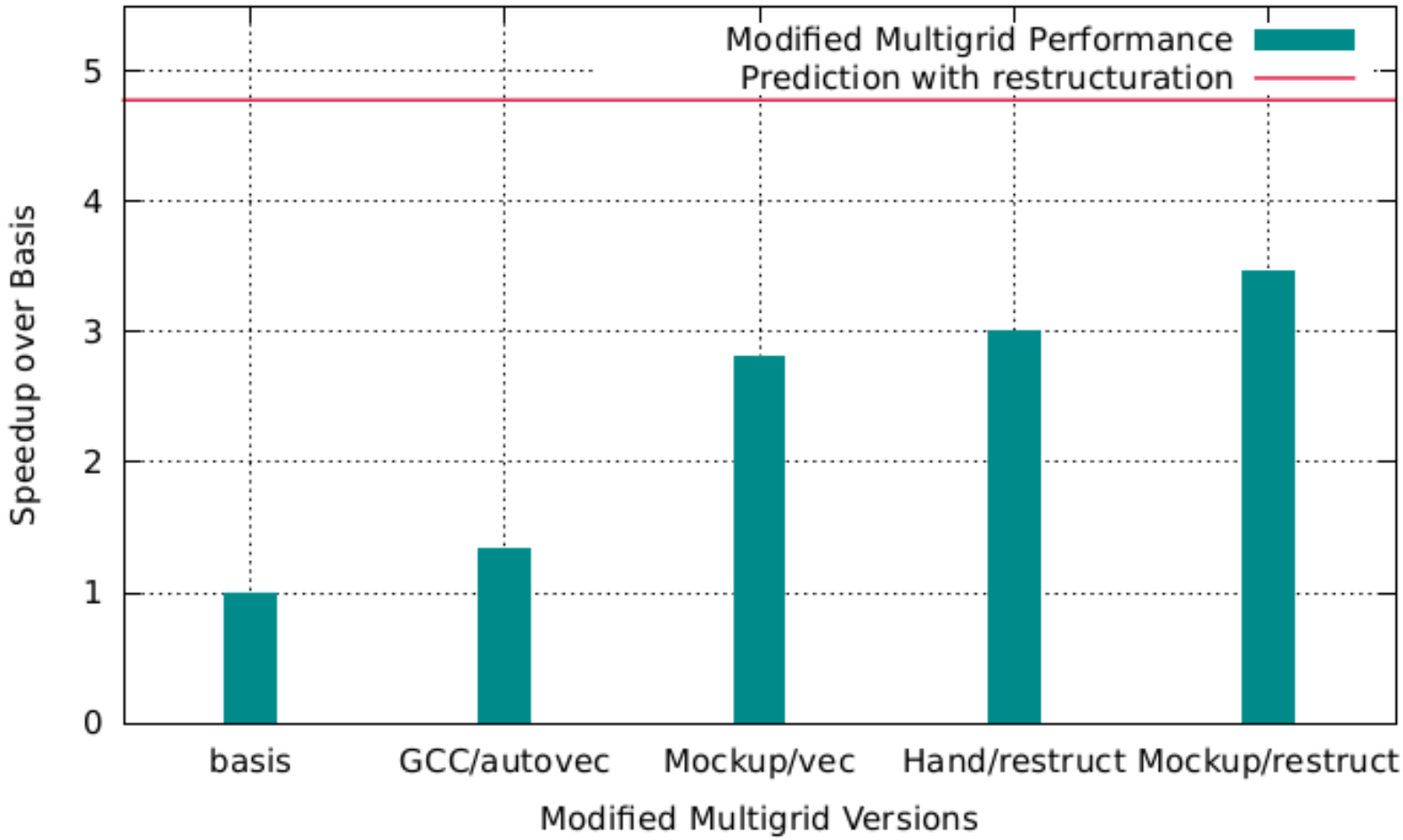

Figure : Modified Multigrid Performance

Outline

Related Works

This work is based on:

- Aumage, O., Barthou, D., Haine, C., Meunier, T.: *Detecting SIMDization Opportunities through Static-Dynamic Dependence Analysis.* Workshop on Productivity and Performance (2013)

Related works on vectorization:

- Videau, B., Marangozova-Martin, V., Genovese, L.: *Optimizing 3D Convolutions for Wavelet Transforms on CPUs with SSE Units and GPUs.* Euro-Par (2013)
- Kong, Martin and Veras, Richard and Stock, Kevin and Franchetti, Franz and Pouchet, Louis-Noël and Sadayappan, P.: *When polyhedral transformations meet SIMD code generation.* PLDI (2013)

Related Software:

- Intel Vtune
- PIN tool

Conclusion

Unsufficient/unefficient compilers autovectorization.

- We propose guided vectorization through user hints
- We backup these hints with performance estimations

We presented two different approaches for estimating the performance of vectorized loop kernels.

- Performance Model
  - Predicts an upper bound of performance
- Code Mockups Generation
  - Approach vectorized loop kernel behavior
  - We showed on a typical simulation code that this approach is relevant

Future Works:

- More Mockups (rescheduling, etc.)
- Evaluate Code Mockups Generation approach on GPU architectures
- Possibility of including MAQAO in a compilation chain