Guiding Polyhedral Schedulers for Vectorization through Constraints Generated from an SLP Algorithm

Tom Hammer Stéphane Genaud Vincent Loechner

Université de Strasbourg & Inria CAMUS team

IMPACT '26, Kraków

Outline

Table of Contents

Loop vectorization vs. SLP

```
for(i = 0; i < 4; i++){
  A[i] = B[i] + C[i]
}
```

No dependencies between iterations of the loop

```
A[0] = B[0] + C[0]
A[1] = B[1] + C[1]
A[2] = B[2] + C[2]
A[3] = B[3] + C[3]
```

Iterations can be grouped into a vector instruction

```
A[0:3] = B[0:3] + C[0:3]
```

Only instances of the same statement can be grouped together

SLP vectorization

All control flow removed from program

```
A[0] = B[0] + C[0]
A[1] = B[1] + C[1]
D[2] = B[2] + C[2]
D[3] = B[3] + C[3]
```

Isomorphic instructions can be grouped into vector instructions through packing

```
v_out = A[0:3] + B[0:3]
A[0:1] = v_out[0:1]
D[2:3] = v_out[2:3]
```

Optimizations may not always be represented by affine schedules

Polyhdedral schedulers

- Polyhedral representation
  - Iteration domains
  - Data accesses
  - Schedules
- Find a new order for the iterations of a program
  - Through composition of basic program transformations
  - By selecting a schedule in a space of legal schedules
- Additionnal constraints
  - Data locality, loop fusion
  - GPU execution

Vectorization in polyhedral schedulers

- Polyhedral schedulers rely on loop vectorization
  - Delegate vectorization to compilers
  - Sink a parallel loop to the innermost dimension
  - Generate vector intrinsics after stripmining the innermost loop
- Additional constraining of the scheduling process
  - Objective variables modeling stride 0/1 references and permutability, Kong et al.
  - Driving the transformation process with cost modeling of vectorization, Trifunovic et al.
  - Explorative constraint injection for GPU scheduling, Bastoul et al.

Motivating example

trisolv kernel from Polybench/C

```
for (i = 0; i < N; i++)
  x[i] = b[i];               //S1
  for (j = 0; j < i; j++)
    x[i] -= L[i][j] * x[j]     //S2
  x[i] = x[i] / L[i][i]       //S3
```

Code generated by Pluto

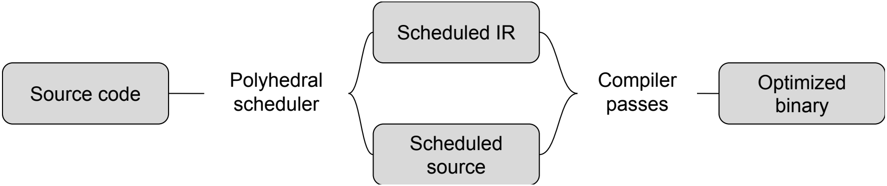
```
for (i = 0; i < N; i++)
  x[i] = b[i]
x[0] = x[0] / L[0][0]
for (i = 1; i < N ; i++)
  for (j = 0; j < i; j++)
    x[i] -= L[i][j] * x[j]     //S2
  x[i] = x[i] / L[i][i]
```

Transformation for vectorization

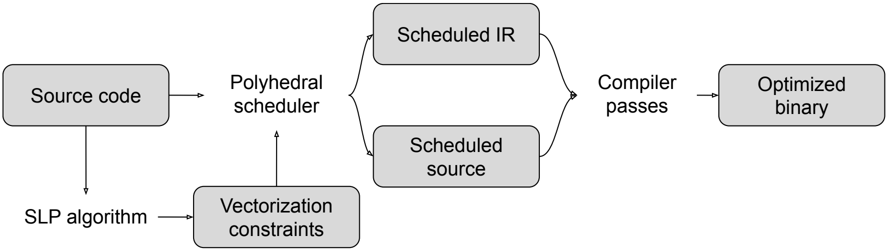
```
for (i=0; i < N; i++)
  x[i] = b[i]
for (i = 0; i < N-1 ; i++)
  x[i] = x[i] / L[i][i]
  for (j = i+1; j < N; j++)
    x[j] -= L[j][i] * x[i]     //S2
x[N-1] = x[N-1] / L[N-1][N-1]
```

Table of Contents

Classical polyhedral workflow



Proposed approach



Autovesk

- **Autovesk:** Automatic vectorized code generation from unstructured static kernels using graph transformations, Tayeb et al.
- Based on graphs of operations generated from C++ operator overloading
  - Stemming from loads
  - Leading to stores
  - No control flow
- Isomorphic nodes are fused into vector instructions
  - Additional Extract/Merge nodes for packing
  - Minimization of the total number of nodes
- Tested on small and simple kernels

General approach

- Execute an SLP algorithm (Autovesk)
  - Smaller problem sizes (N=8)
  - Annotate instructions with iterator information
- Generate configuration files for constraint injection
  - Select a dimension for vectorization from program traces
- Inject constraints into Pluto

Constraint injected for every statement

$$\theta_S(\vec{i_S}) = (c_1^S, c_2^S, c_3^S, \ldots, c_m^S)(\vec{i_S}) + c_0^S \qquad \vec{i_S} \in \mathbb{Z}$$

$$c_k^S = 0$$

Université de Strasbourg

Inria

Generating program traces

Trisolv kernel from Polybench/C

```
for (i = 0; i < N; i++)
  x[i] = b[i];                //S1
  for (j = 0; j < i; j++)
    x[i] -= L[i][j] * x[j]    //S2
  x[i] = x[i] / L[i][i]       //S3
```

Generating program traces

Trisolv kernel from Polybench/C

```
for (i = 0; i < N; i++)
  x[i] = b[i];                //S1
  for (j = 0; j < i; j++)
    x[i] -= L[i][j] * x[j]    //S2
  x[i] = x[i] / L[i][i]       //S3
```

- N = 4

Produced execution trace

```
x[0] = b[0];            //S1 0
x[0] = x[0] / L[0][0];  //S3 0
x[1] = b[1];            //S1 1
x[1] -= L[1][0] * x[0]; //S2 1 0
x[1] = x[1] / L[1][1];  //S3 1
x[2] = b[2];            //S1 2
x[2] -= L[2][0] * x[0]; //S2 2 0
x[2] -= L[2][1] * x[1]; //S2 2 1
x[2] = x[2] / L[2][2];  //S3 2
x[3] = b[3];            //S1 3
x[3] -= L[3][0] * x[0]; //S2 3 0
x[3] -= L[3][1] * x[1]; //S2 3 1
x[3] -= L[3][2] * x[2]; //S2 3 2
x[3] = x[3] / L[3][3];  //S3 3
```

Running Autovesk

Original trace

```
//S1 0
//S3 0
//S1 1
//S2 1 0
//S3 1
//S1 2
//S2 2 0
//S2 2 1
//S3 2
//S1 3
//S2 3 0
//S2 3 1
//S2 3 2
//S3 3
```

Running Autovesk

Original trace

```
//S1 0
//S3 0
//S1 1
//S2 1 0
//S3 1
//S1 2
//S2 2 0
//S2 2 1
//S3 2
//S1 3
//S2 3 0
//S2 3 1
//S2 3 2
//S3 3
```

Vectorized trace

```
// Vec node:
S1 0
S1 1
S1 2
S1 3
// End
S3 0
// Vec node:
S2 1 0
S2 2 0
S2 3 0
// End
S3 1
// Vec node:
S2 2 1
S2 3 1
// End
S3 2
S2 3 2
S3 3
```

Generating configuration files

- Count the occurrences of increments by one of each iterator within vector nodes
- Select the dimension with the max count for vectorization

```
// Vec node:
S1 0
S1 1
S1 2
S1 3
// End
S3 0
// Vec node:
S2 1 0
S2 2 0
S2 3 0
// End
S3 1
// Vec node:
S2 2 1
S2 3 1
// End
S3 2
S2 3 2
S3 3
```

S1: [+1]
S1: [+1]
S1: [+1]
S2: [+1, +0]
S2: [+1, +0]
S2: [+1, +0]

```
S1: [3]
S2: [3, 0]
S3: [0]
```

↓

```
S1
1
S2
1 0
S3
0
```

Constraint injection

- *S1[1]*: vectorized on i
  - $c_i = 0$ for 0 dimensions
- *S2[1, 0]*: vectorized on i
  - $c_i = 0$ for 1 dimension
- *S3* left unconstrained

Constraint injection

- *S1[1]*: vectorized on i
  - $c_i = 0$ for 0 dimensions
- *S2[1, 0]*: vectorized on i
  - $c_i = 0$ for 1 dimension
- *S3* left unconstrained

- Code generated by our modified Pluto algorithm

```
for (i=0; i < N; i++)
  x[i] = b[i]
for (i = 0; i < N-1 ; i++)
  x[i] = x[i] / L[i][i]
  for (j = i+1; j < N; j++)
    x[j] -= L[j][i] * x[i]     //S2
x[N-1] = x[N-1] / L[N-1][N-1]
```

Problematic cases

- In some cases, selected dimensions may not represent the traces generated

Seidel-2d kernel from Polybench/C

```
for (t = 0; t <= T_STEPS - 1; t++)
  for (i = 1; i <= N - 2; i++)
    for (j = 1; j <= N - 2; j++)
      A[i][j] = (A[i-1][j-1]+A[i-1][j]+
        A[i-1][j+1]+A[i][j-1]+A[i][j]+
        A[i][j+1]+A[i+1][j-1]+A[i+1][j]+
        A[i+1][j+1])/SCALAR_VAL(9.0);
```

Partial trace produced by Autovesk

```
// Vec node:
S1 0 4 1
S1 1 1 4
// End
S1 1 2 2
S1 2 1 1
S1 0 2 6
S1 0 3 4
// Vec node:
S1 0 4 2
S1 1 1 5
// End
```

Problematic cases

- In some cases, selected dimensions may not represent the traces generated

```
// Vec node:
S1 0 4 1
S1 1 1 4
// End
S1 1 2 2
S1 2 1 1
S1 0 2 6
S1 0 3 4
// Vec node:
S1 0 4 2
S1 1 1 5
// End
```

Generated configuration

```
S1
1 0 0
```

- It is impossible to vectorize all iterations on dimension t

Université de Strasbourg

Inria

Problematic cases

- In some cases, selected dimensions may not represent the traces generated

```
// Vec node:
S1 0 4 1
S1 1 1 4
// End
S1 1 2 2
S1 2 1 1
S1 0 2 6
S1 0 3 4
// Vec node:
S1 0 4 2
S1 1 1 5
// End
```

Generated configuration

```
S1
1 0 0
```

- It is impossible to vectorize all iterations on dimension t

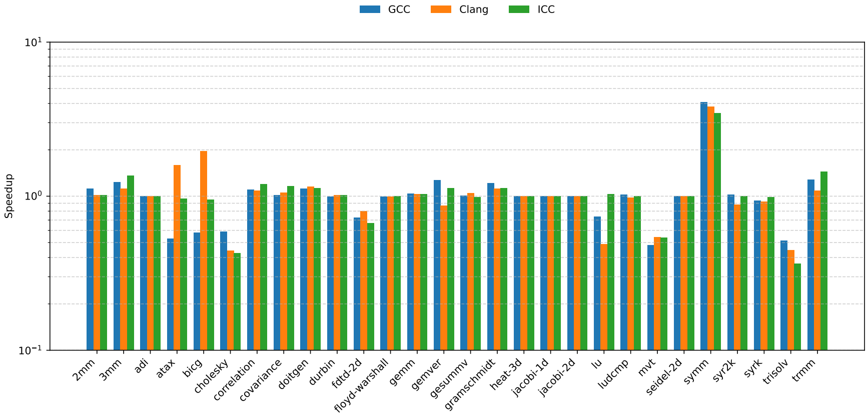
- In this case, we apply a constraint relaxation algorithm

Table of Contents

Experimental setup

- Intel 12th gen i7-12700H (6 P-Cores, 8 E-Cores)
  - 80KB L1, 1.25MB L2, 24MB L3 caches
  - AVX2 vector instructions
- Polybench/C benchmark suite
  - Custom problem sizes
  - Excluding *deriche* and *nussinov*
  - Parameters set to 8 in Autovesk
  - Double precision float
- GCC, Clang, ICC
- Profiled with perf-cpp

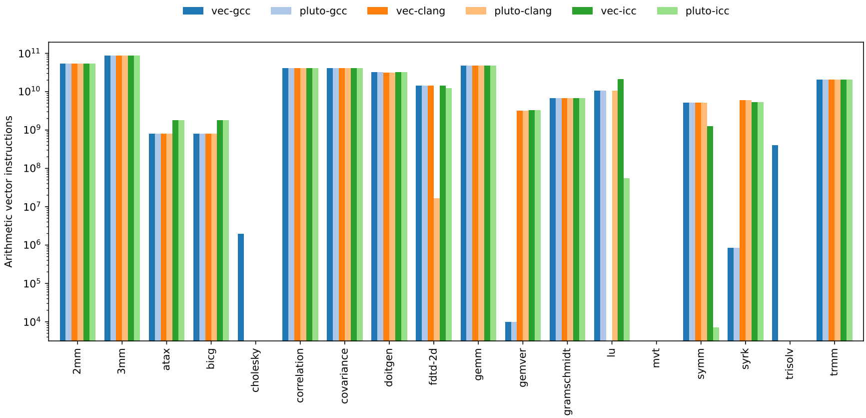
Speedup


GCC
Clang
ICC
Speedup
10^1
10^0
10^{-1}
2mm
3mm
adi
atax
bicg
cholesky
correlation
covariance
doitgen
durbin
fdtd-2d
floyd-warshall
gemm
gemver
gesummv
gramschmidt
heat-3d
jacobi-1d
jacobi-2d
lu
ludcmp
mvt
seidel-2d
symm
syr2k
syrk
trisolv
trmm

Benchmark breakdown

- 2 kernels could not be scheduled by either Pluto and our approach
  - *adi* and *ludcmp*
- 8 kernels produced the same schedule with both versions
  - *durbin, floyd-warshall, gesummv, heat-3d, jacobi-1d, jacobi-2d, seidel-2d, syr2k*
- 10 kernels yield a speedup and 8 yield a slowdown

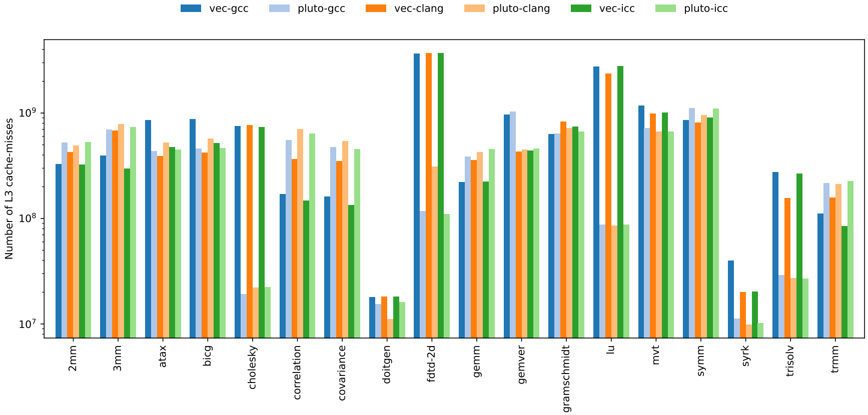
Vector instructions



Vector instructions

- Our constraints do not hinder vectorization in any cases
- GCC is able to generate vector instructions for 2 kernels where there were previously none
  - *cholesky* and *trisolv*
- We improved the number of vector instructions for 3 kernels
  - *lu*, *symm*, *fdtd-2d*
- Some other metrics might explain the speedups or slowdowns

Cache misses



Loads and stores

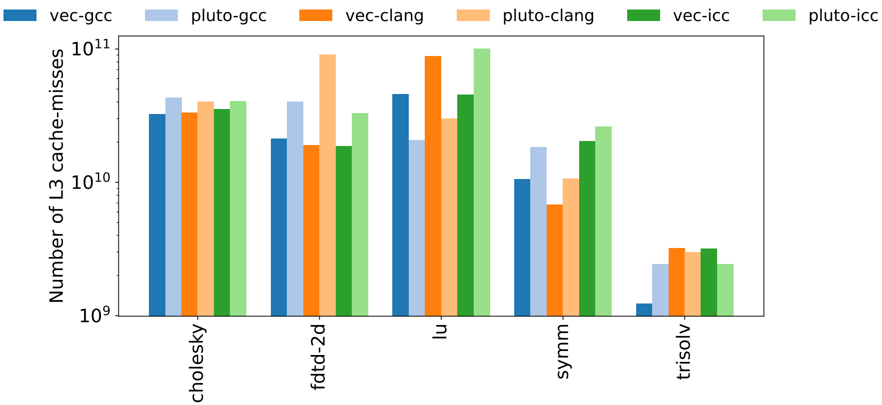


Table of Contents

Metric evaluation

- Our approach enables the generation of vector instructions on many kernels
- The number of vector instructions is not directly correlated with performance
  - Gains can be outweighed by losses in data locality
  - Cache misses above a certain threshold greatly impact execution time

Metric evaluation

- Our approach enables the generation of vector instructions on many kernels
- The number of vector instructions is not directly correlated with performance
  - Gains can be outweighed by losses in data locality
  - Cache misses above a certain threshold greatly impact execution time

```
for (i=0; i < N; i++)
  x[i] = b[i]
for (i = 0; i < N-1 ; i++)
  x[i] = x[i] / L[i][i]
  for (j = i+1; j < N; j++)
    x[j] -= L[j][i] * x[i]    //S2
x[N-1] = x[N-1] / L[N-1][N-1]
```

- Vectorization enabled on the j dimension
- The schedule reads from a different row of L at every iteration

Conclusion

- Improvements in the number of vector instructions
  - Data locality is not taken into account by our model
  - Gains in performance when the locality is preserved, losses otherwise
- In some cases, we get the same results as Pluto
  - Pluto does not enforce vectorization during the scheduling process
  - Pluto relies on post-processing the produced schedule for vectorization

Further work

- Implementation of finer grained constraints
  - Scalar dimensions for fusion/fission, and interleaving
  - Constraints on data locality and memory layout
- Testing on architectures presenting larger vector registers
  - AVX512
  - ARM SVE/SVE2