SAC: SIMD Architecture Compiler

Serge Guelton

Télécom Bretagne

25 octobre 2010

What is SAC ?

> Introduction

A multimedia instruction generator for pips

- Looks for Superword Level Parallelism ($\neq$ vector loops)
- Generic in the nature of the instruction set and the register width
- Implemented in pyps w/ legacy tpips include file
- Inherited from several Télécom Bretagne Trainees :)

A generic mutlimedia instruction set

- Types
  - v4sf
  - v2df
  - a4si . . .
- Memory Instructions
  - SIMD_LOAD_V4SF(vec,arr)
  - SIMD_STORE_V4SF(vec,arr) . . .
- Arithmetic Instructions
  - SIMD_ADD_PS(vout,vin0,vin1)
  - SIMD_MUL_PS(vout,vin0,vin1)
  - SIMD_DIV_PD(vout,vin0,vin1) . . .

Follow the source, Luke

- Reference implementation

Plain C, sequential, parsed by PIPS.

→ used in interprocedural analysis

- Target specific implementation

Plain C, uses Compiler intrinsic

Included as last compilation step

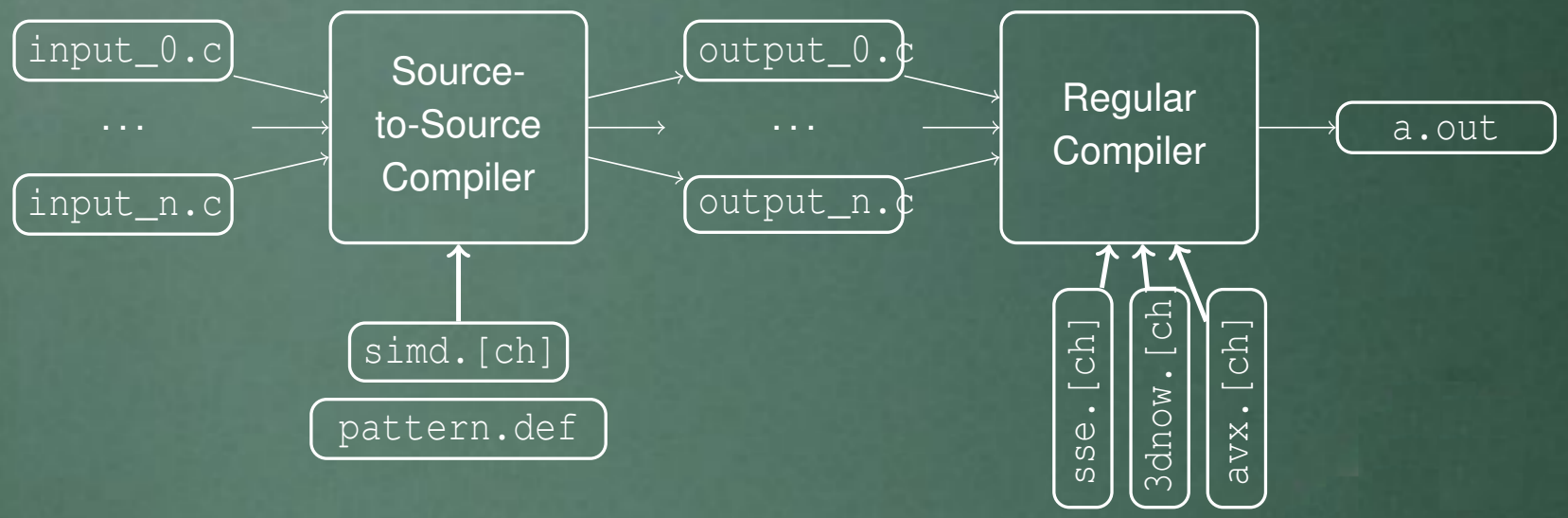
```
void SIMD_ADD_V4SF(float out[4],
                   float in0[4],
                   float in1[4]) {
  for(size_t i=0;i<4;i++)
    out[i]=in0[i]+in1[i];
}

#include <xmmintrin.h>
#define SIMD_ADD_V4SF(out,in0,in1) \
(out) = _mm_add_ps((in0),(in1))
```

A Parametric Pattern Matcher

- patterns.def
  - binds names to IR expressions
  - needs \PipsPropRef{SAC_SIMD_REGISTER_WIDTH} expressions

```
sub : = REFERENCE - REFERENCE REFERENCE ;
sub : = REFERENCE - REFERENCE CONSTANT ;
sub : = REFERENCE - CONSTANT REFERENCE ;
sub : = REFERENCE + UNARY_MINUS REFERENCE CONSTANT ;
sub : = REFERENCE + CONSTANT UNARY_MINUS REFERENCE ;
sub : = REFERENCE + REFERENCE UNARY_MINUS REFERENCE
    ;
```

The Big Picture



Additional Transformations

- if conversion

Execute both branches and rely on masked assignment

- auto unrolling

Unrolls all inner loops by the appropriate step

- reduction parallelization

performs partial reductions to make //me appear

- scalar renaming

remove false dependences

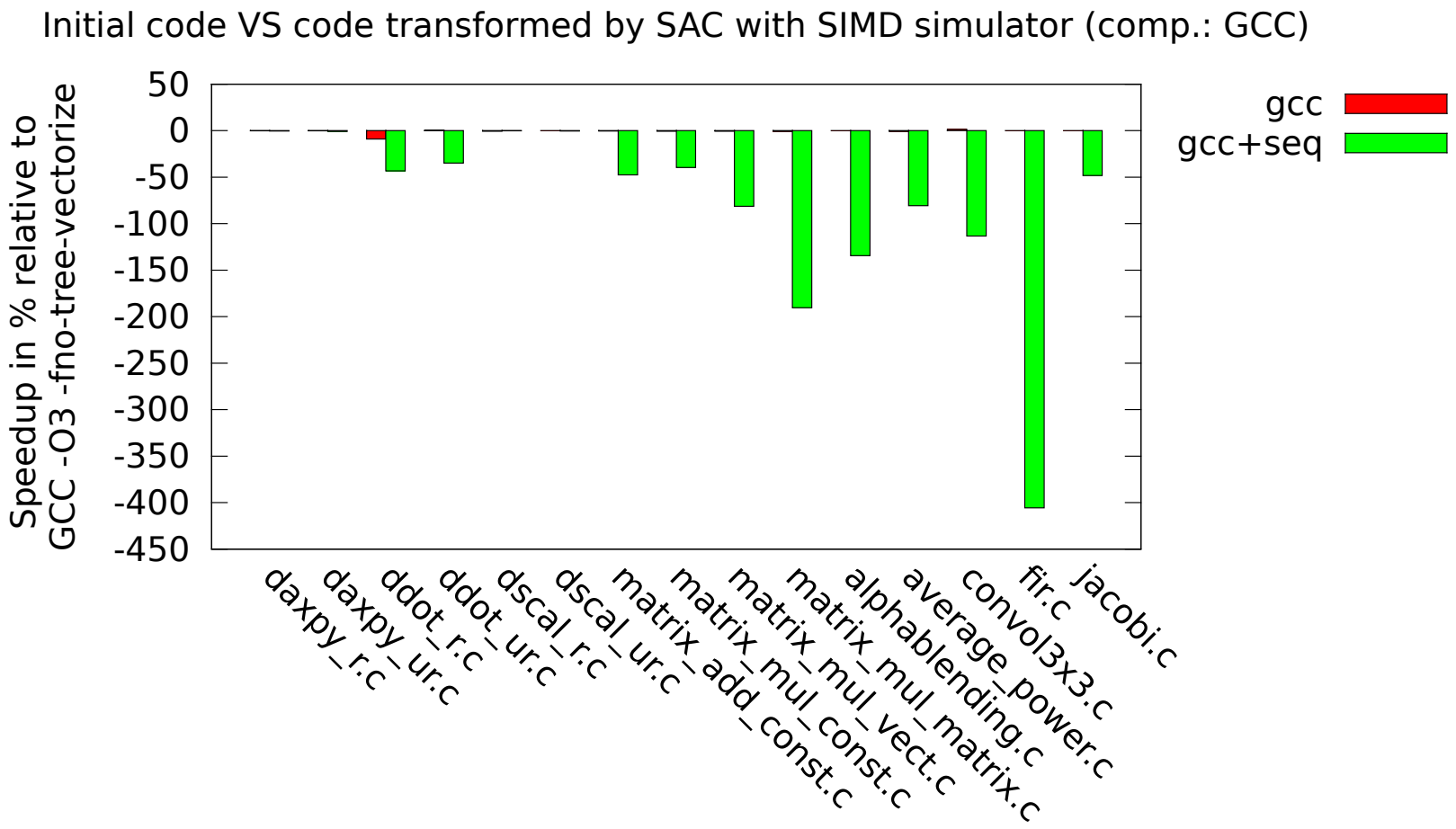
- redundant load store elimination

optimize load and stores to/from vector registers

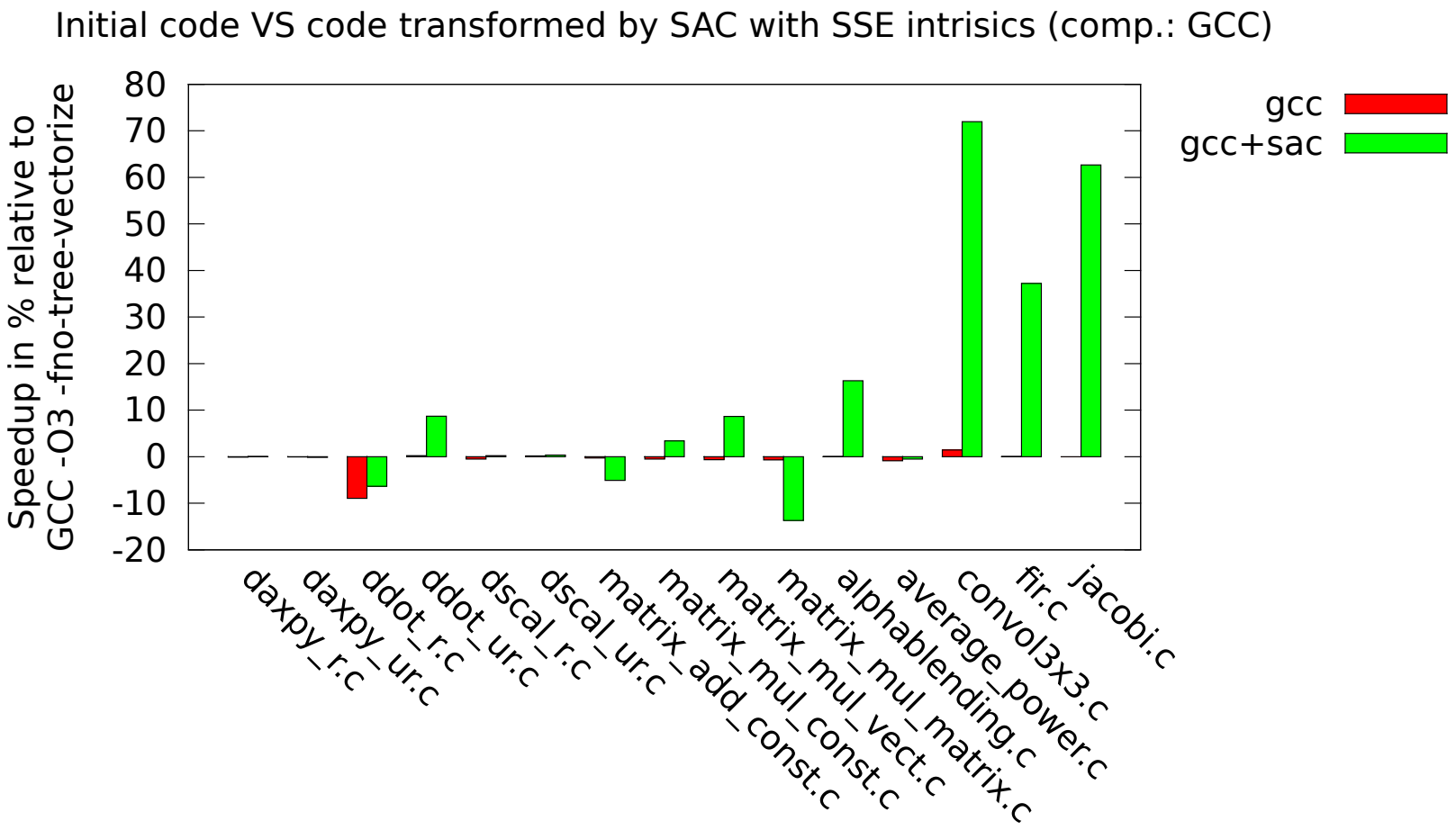
Exempli Gratia : scalar product

Add your demo here

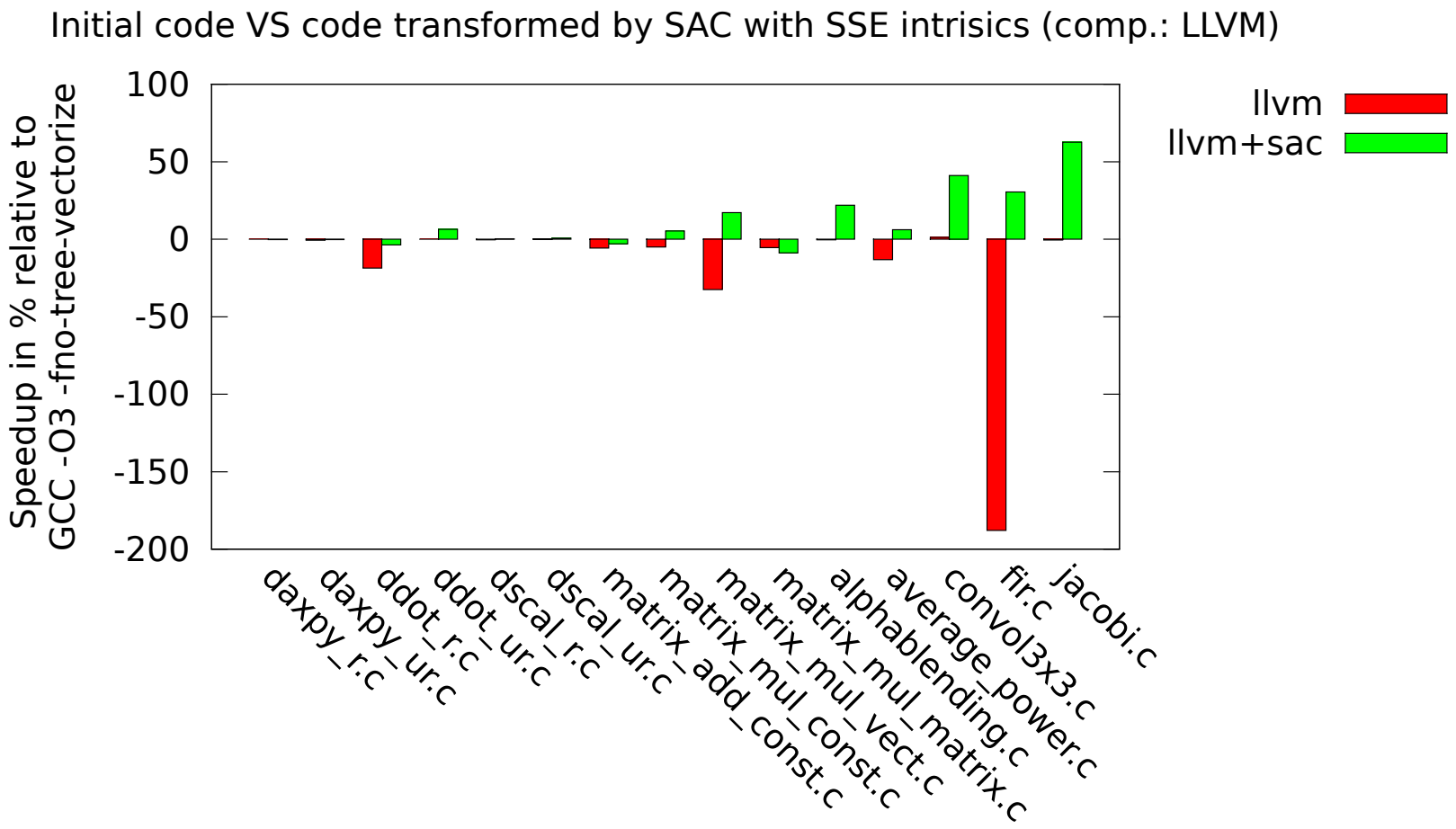
Benchmarks


Initial code VS code transformed by SAC with SIMD simulator (comp.: GCC)
Speedup in % relative to GCC -O3 -fno-tree-vectorize
50
0
-50
-100
-150
-200
-250
-300
-350
-400
-450
gcc
gcc+seq
daxpy_r.c
daxpy_ur.c
ddot_r.c
ddot_ur.c
dscal_r.c
dscal_ur.c
matrix_add_const.c
matrix_mul_const.c
matrix_mul_vect.c
matrix_mul_matrix.c
alphablending.c
average_power.c
convol3x3.c
fir.c
jacobi.c

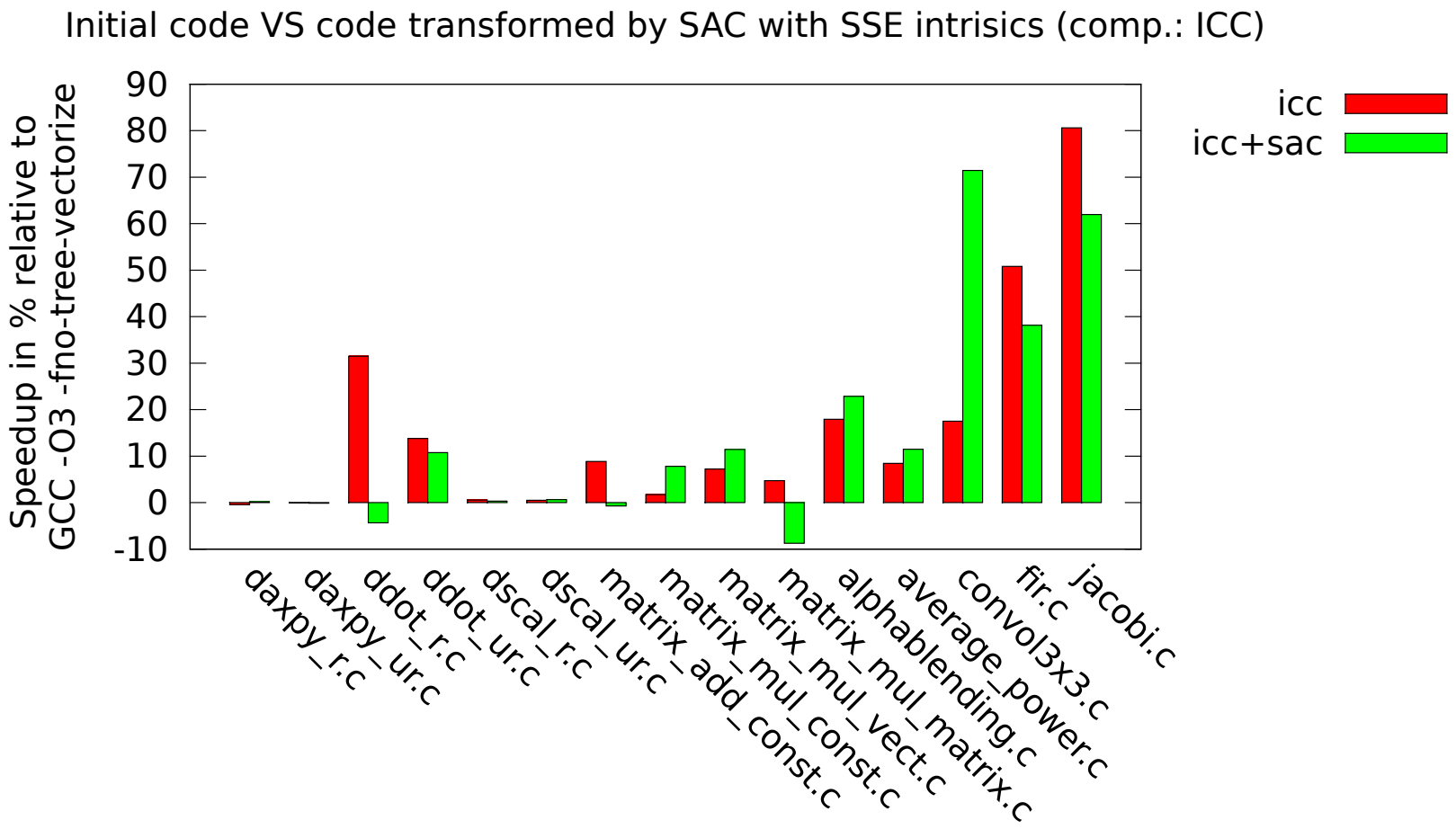
Benchmarks


Initial code VS code transformed by SAC with SSE intrisics (comp.: GCC)
Speedup in % relative to GCC -O3 -fno-tree-vectorize
80
70
60
50
40
30
20
10
0
-10
-20
gcc
gcc+sac
daxpy_r.c
daxpy_ur.c
ddot_r.c
ddot_ur.c
dscal_r.c
dscal_ur.c
matrix_add_const.c
matrix_mul_const.c
matrix_mul_vect.c
matrix_mul_matrix.c
alphablending.c
average_power.c
convol3x3.c
fir.c
jacobi.c

Benchmarks


Initial code VS code transformed by SAC with SSE intrisics (comp.: LLVM)
Speedup in % relative to GCC -O3 -fno-tree-vectorize
100
50
0
-50
-100
-150
-200
llvm
llvm+sac
daxpy_r.c
daxpy_ur.c
ddot_r.c
ddot_ur.c
dscal_r.c
dscal_ur.c
matrix_add_const.c
matrix_mul_const.c
matrix_mul_vect.c
matrix_mul_matrix.c
alphablending.c
average_power.c
convol3x3.c
fir.c
jacobi.c

Benchmarks


Initial code VS code transformed by SAC with SSE intrisics (comp.: ICC)
Speedup in % relative to GCC -O3 -fno-tree-vectorize
90
80
70
60
50
40
30
20
10
0
-10
icc
icc+sac
daxpy_r.c
daxpy_ur.c
ddot_r.c
ddot_ur.c
dscal_r.c
dscal_ur.c
matrix_add_const.c
matrix_mul_const.c
matrix_mul_vect.c
matrix_mul_matrix.c
alphablending.c
average_power.c
convol3x3.c
fir.c
jacobi.c

Future Works

- implement a shuffle operator
  - → vector register state is already here ☺
- support more instruction set
  - → ARM's NEON and Intel's AVX on going ☺
- understand matrix product performance ☹