A Basic Linear Algebra Compiler

Daniele Spampinato
Markus Püschel

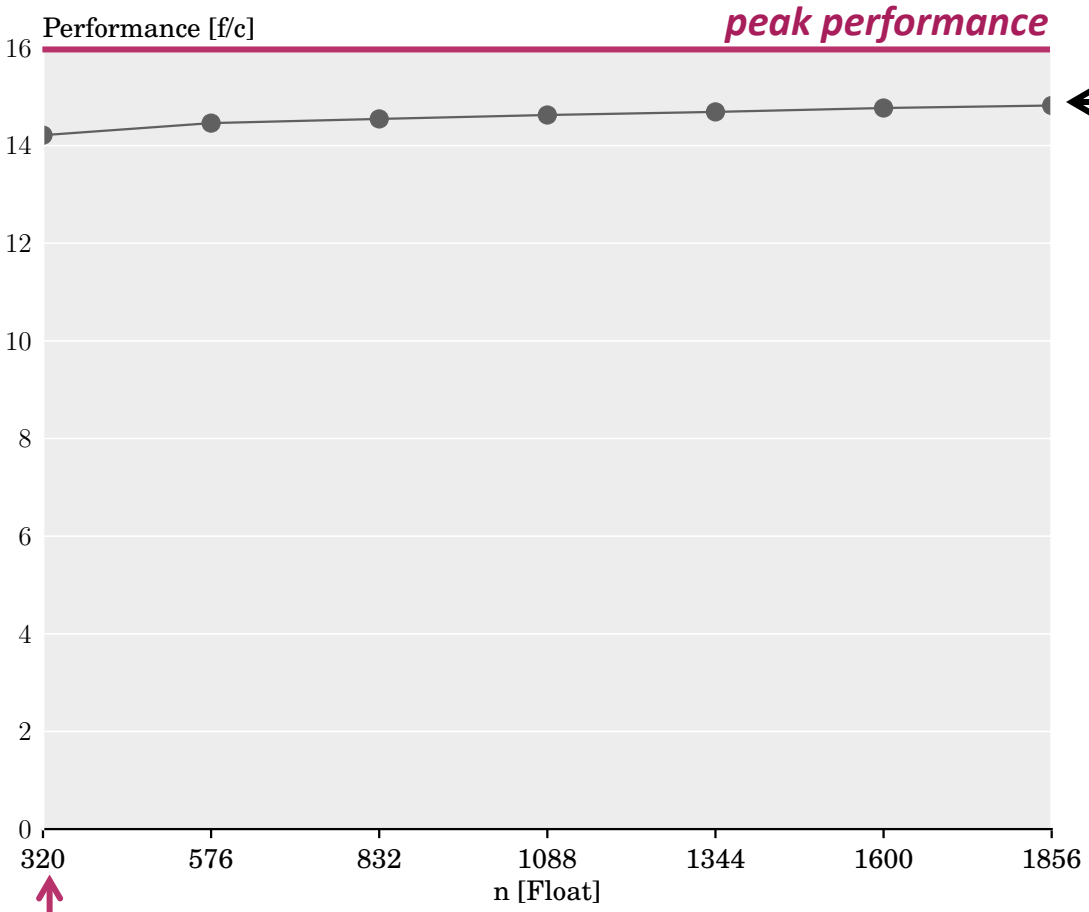
Department of Computer Science
ETH Zürich, Switzerland

ETH zürich

SPIRAL
www.spiral.net

Library performance for sgemm (C += AB)

Intel MKL 10.3 on Intel Core i7-2600 CPU @ 3.40GHz

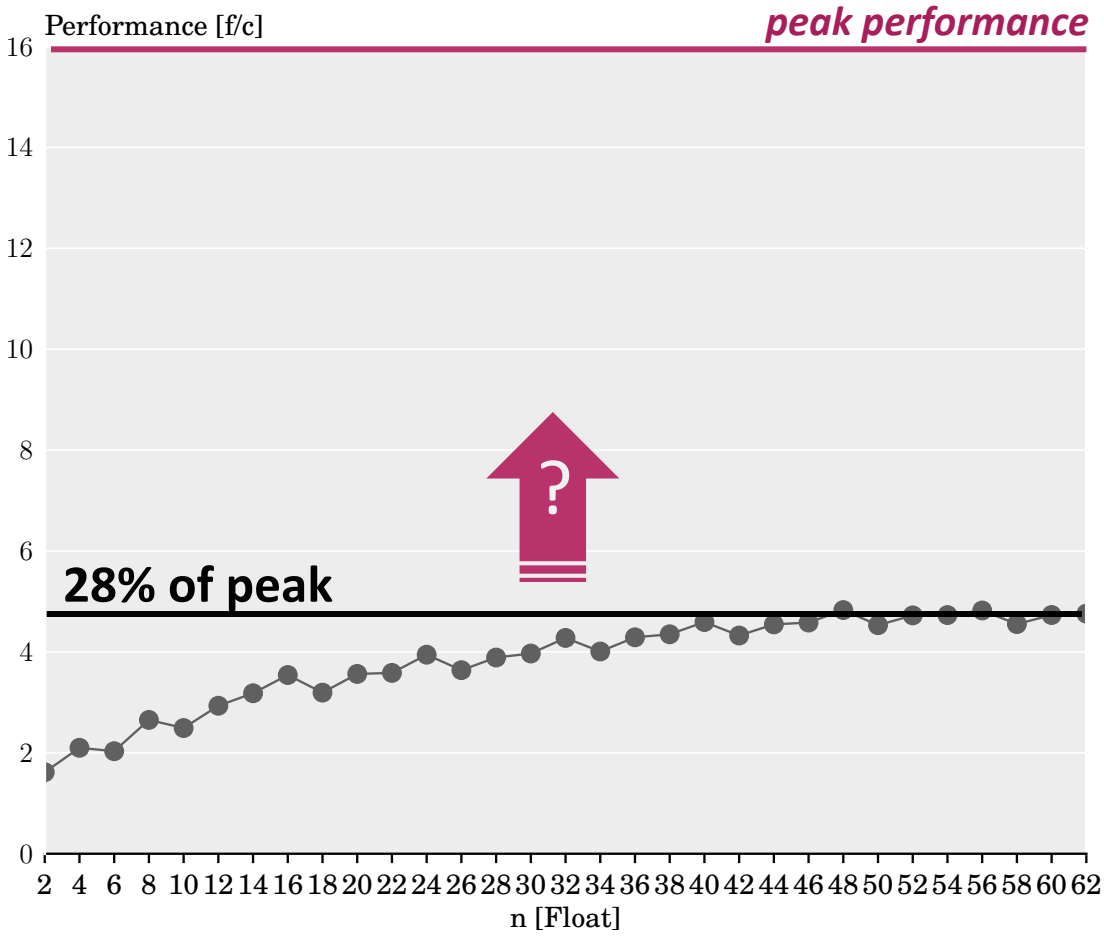


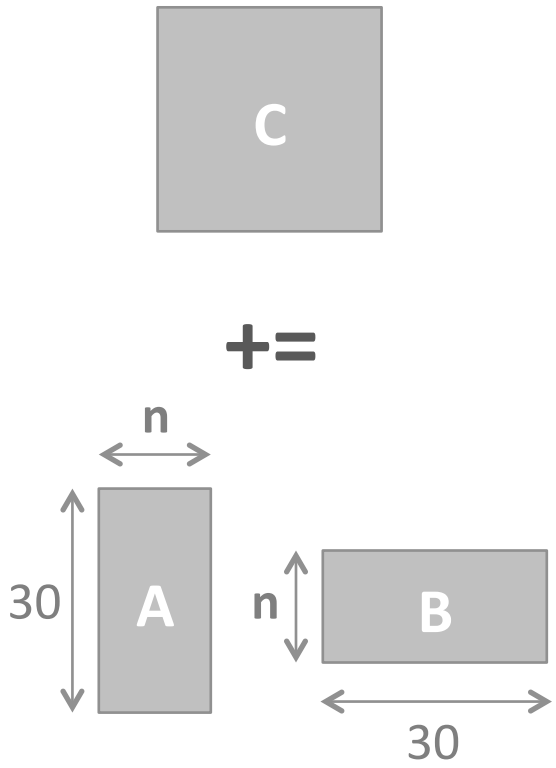
91% of peak

What happens for smaller M,K, and N?

A closer look at small problem sizes

Intel MKL 10.3 on Intel Core i7-2600 CPU @ 3.40GHz





Are small problems so important?

- **Required by many performance-critical applications:**
  - Optimization algorithms
  - Kalman filters
  - Geometric transformations
  - Real-time localization and mapping
- **Often for specific input sizes**
- **Do not necessarily comply with standard interface (e.g., BLAS)**
- **Of special interest for a variety of embedded systems**
  - Reduced HW and SW resources

Basic Linear Algebra Computations (BLACs)

Examples:

$$y = Ax$$

$$C = \alpha AB^T + \beta C$$

$$\gamma = x^T(A + B)y + \delta$$

Composed of:

- Scalars, vectors, and matrices
- Operators:
  - Addition
  - Scalar multiplication
  - Matrix multiplication
  - Transposition

All input and output vectors and matrices have a fixed size

Our Goal: From (Any) BLAC to fast code

$$y = Ax$$

A is 2x3
x is 3x1

LGen

Design similar to Spiral (www.spiral.net)

```
void f(double const * A, double const * x, double * y) {
  double t0, …;

  t0 = x[0];
  t1 = x[1];
  ...
  t9 = t3 * t0;
  t10 = t6 * t0;
  t11 = t4 * t1;
  t12 = t9 + t11;
  ...
  y[0] = t16;
  y[1] = t18;
}
```

Our Goal: From (Any) BLAC to fast code

$$y = Ax$$

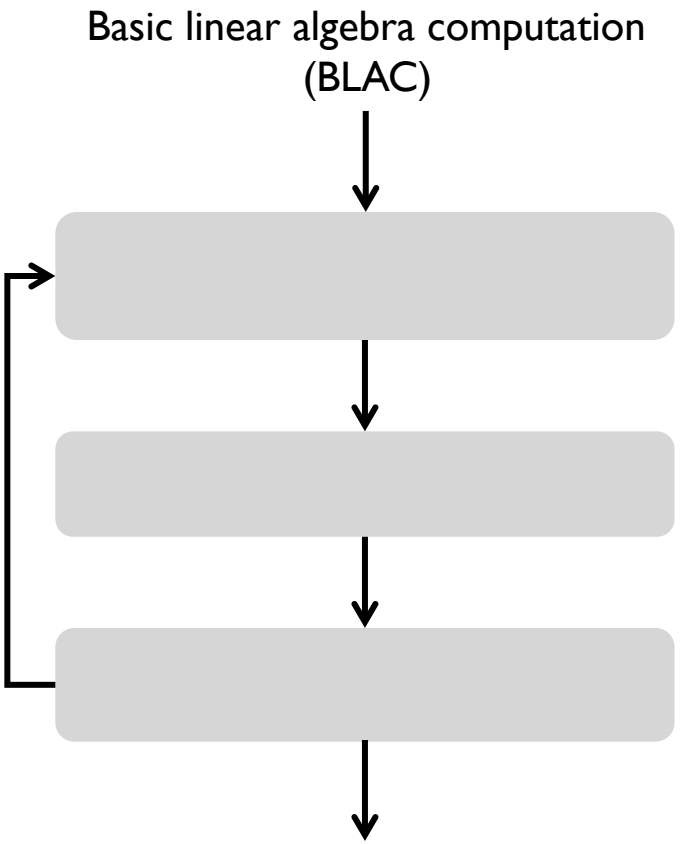
A is 2x3
x is 3x1

LGen

Design similar to Spiral (www.spiral.net)

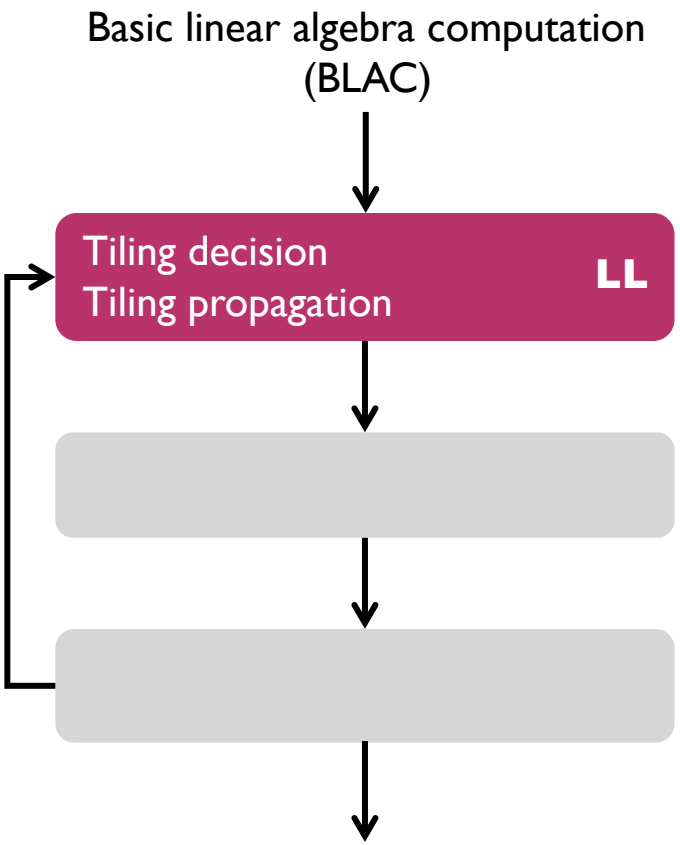
```
void f(double const * A, double const * x, double * y) {
  __m128d t0, …;

  t0 = _mm_loadu_pd(A);
  t1 = _mm_load_sd(A + 2);
  ...
  t6 = _mm_hadd_pd(_mm_mul_pd(t0, t4), _mm_mul_pd(t2, t4));
  t7 = _mm_shuffle_pd(t1, t3, 0);
  t8 = _mm_mul_pd(t7, _mm_shuffle_pd(t5, t5, 0));
  t9 = _mm_add_pd(t6, t8);

  _mm_storeu_pd(y, t9);
}
```

Architecture of LGen



$$y = Ax$$

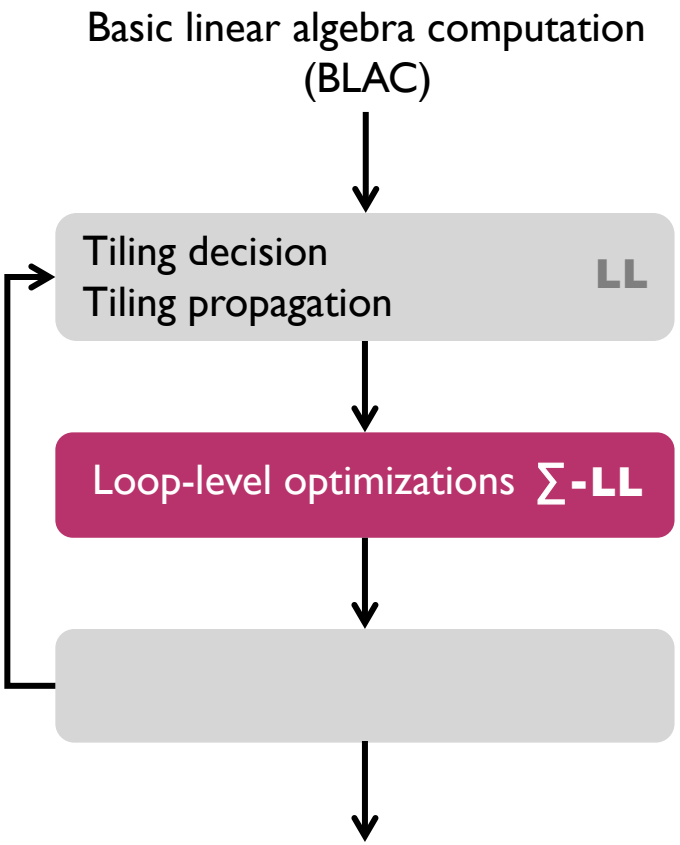
Architecture of LGen



$$y = Ax$$

$$[y = Ax]_{2,1}$$

Architecture of LGen

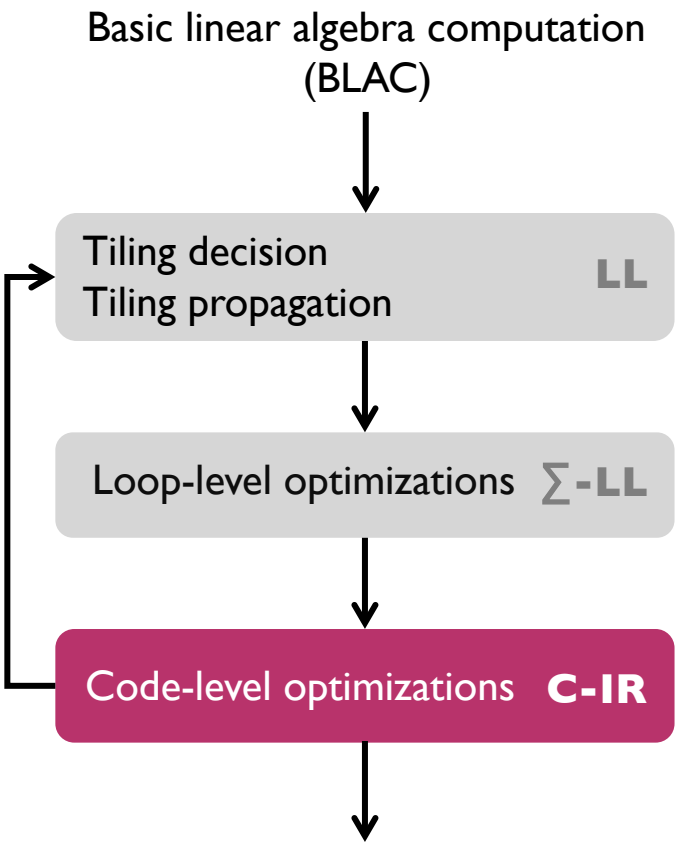


$$y = Ax$$

$$[y = Ax]_{2,1}$$

$$y = \sum_{i,j} S_i(G_i \cdots)$$

Architecture of LGen



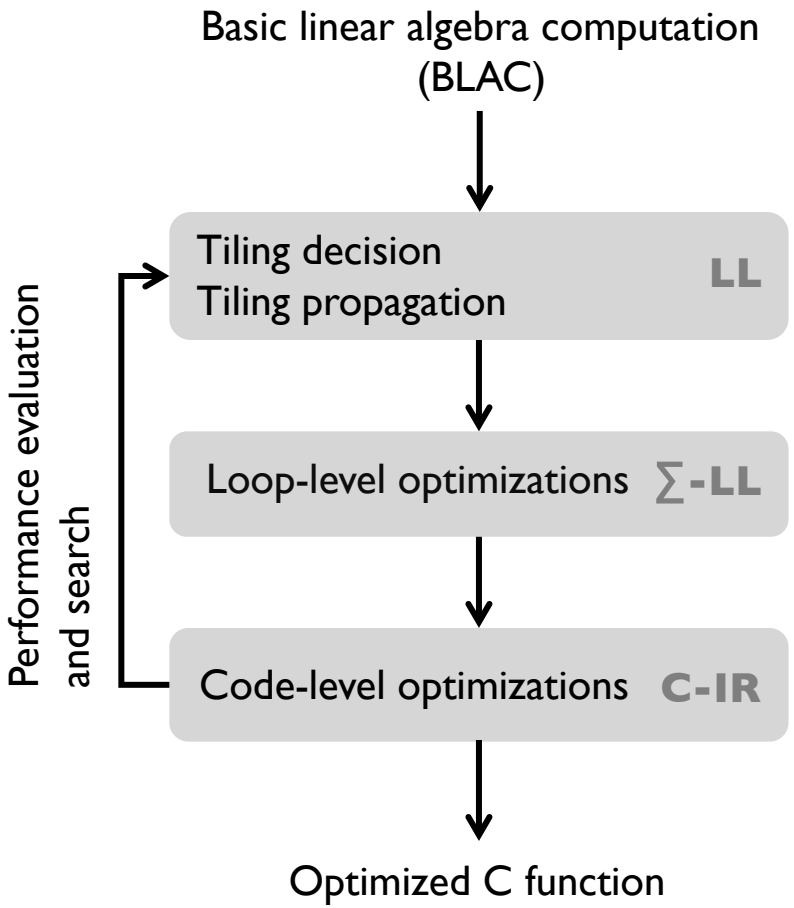
$$y = Ax$$

$$[y = Ax]_{2,1}$$

$$y = \sum_{i,j} S_i (G_i \cdots)$$

...
Mov (mmMulPs A[0,0], x[0,0]), t[0,0]
...

Architecture of LGen



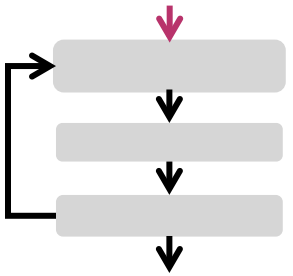
$$y = Ax$$

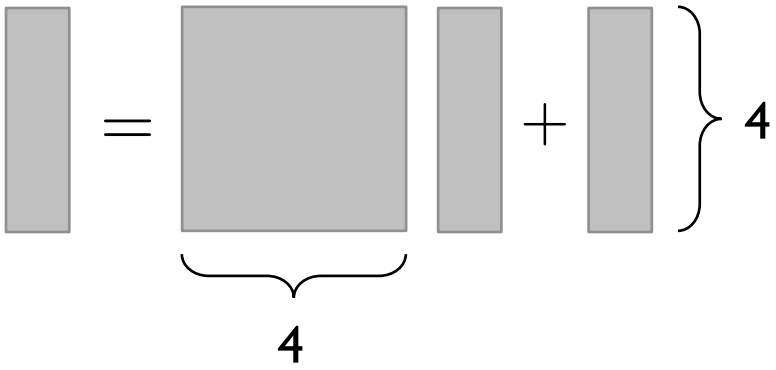
$$[y = Ax]_{2,1}$$

$$y = \sum_{i,j} S_i (G_i \cdots)$$

```
...
Mov (mmMulPs A[0,0], x[0,0]), t[0,0]
...

for(int i = … ) {
 …
 t = _mm_mul_ps(a, x);
 …
}
```

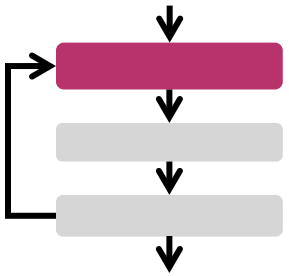
Scalar code generation





$$y = Ax + y$$

Tiling in LL



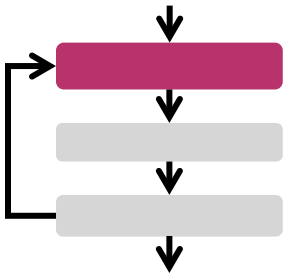
$$[y = Ax + y]_{r,c}$$

Tiling decision for equation

$$r = 2, c = 1$$

Initial task: Deriving tiling decision for operands

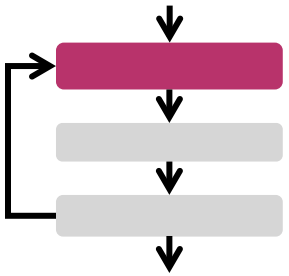
Tiling in LL



$$[y = Ax + y]_{2,1}$$

$$[y]_{2,1} = [Ax + y]_{2,1}$$

Tiling in LL

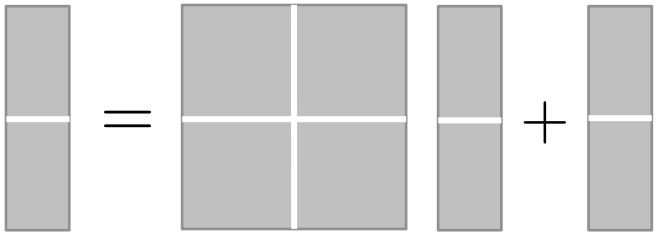


$$[y = Ax + y]_{2,1}$$

$$[y]_{2,1} = [Ax + y]_{2,1}$$

$$[y]_{2,1} = [Ax]_{2,1} + [y]_{2,1}$$

Tiling in LL



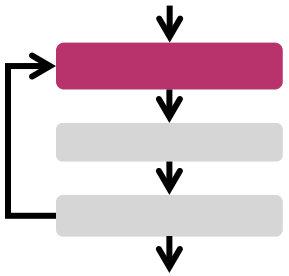
$$[y = Ax + y]_{2,1}$$

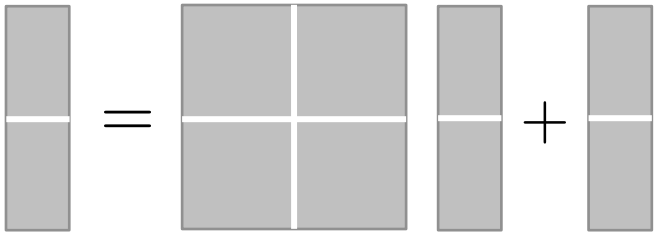
$$[y]_{2,1} = [Ax + y]_{2,1}$$

$$[y]_{2,1} = [Ax]_{2,1} + [y]_{2,1}$$

$$[y]_{2,1} = [A]_{2,k}[x]_{k,1} + [y]_{2,1}$$

Tiling in LL





$$[y = Ax + y]_{2,1}$$

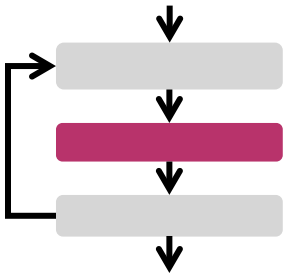
$$[y]_{2,1} = [Ax + y]_{2,1}$$

$$[y]_{2,1} = [Ax]_{2,1} + [y]_{2,1}$$

$$[y]_{2,1} = [A]_{2,2}[x]_{2,1} + [y]_{2,1}$$

Σ-LL: Basics

Extension of Σ-SPL (Franchetti et al., PLDI 2005)



- **Gathers:** $G_L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$ $G_R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$
- **Scatters:** $S_L = G_R$ $S_R = G_L$
- **Extracting a block**

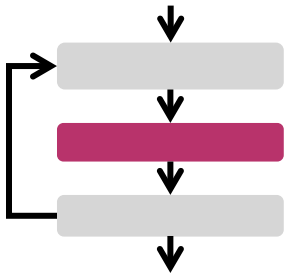
$$A = \begin{bmatrix} B & \\ & \end{bmatrix} \qquad B = A(0:1, 0:1) = G_L A G_R$$

- **Expanding a block**

$$C = \begin{bmatrix} B & 0 \\ 0 & 0 \end{bmatrix} \qquad C = S_L B S_R$$

Gathers and scatters allow for an explicit identification of data accesses

Σ-LL: Some properties



$$\alpha = G_i^{1,2} G_j^{2,4} x$$

$$= G(h_{j,1}^{2\to 4} \circ h_{i,1}^{1\to 2})x = G_{i+j}^{1,4} x$$

$$\alpha = G_i \sum_{j=0}^{3} S_j \beta_j$$

$$= S_i^T \sum_{j=0}^{3} S_j \beta_j = \beta_i$$

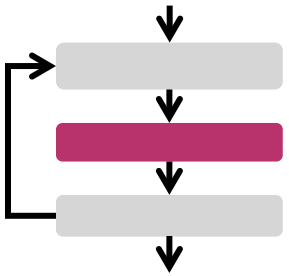
```
for ( k = 0; k < 2; k++ ) t[k] = x[j+k];
a = t[i];
```

```
a = x[i+j];
```

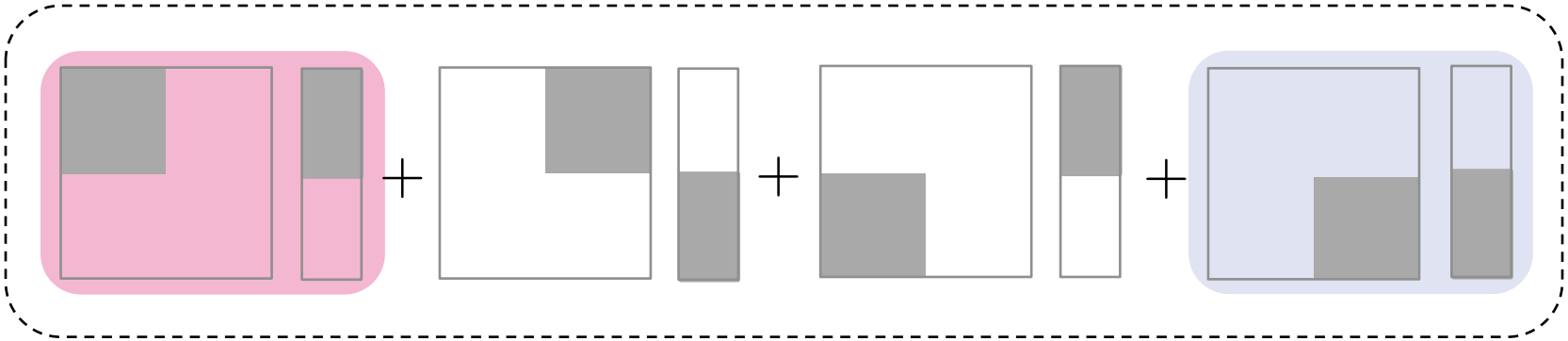
```
for ( j = 0; j < 4; j++ ) t[j] = b[j];
a = t[i];
```

```
a = b[i];
```

LL to Σ-LL



$$[y]_{2,1} = [A]_{2,2}[x]_{2,1} + [y]_{2,1}$$

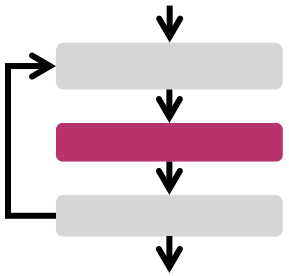


$$S_0\,(G_0AG_0)\,S_0 \cdot S_0\,(G_0x) + \cdots + S_2\,(G_2AG_2)\,S_2 \cdot S_2\,(G_2x)$$

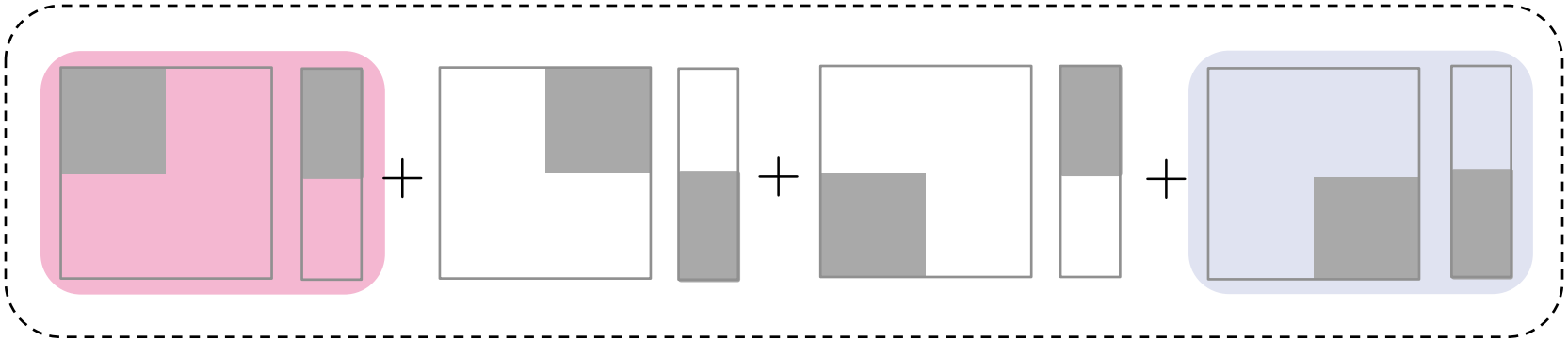
$$= \sum_{\iota=0,2}^{3} \sum_{j=0,2}^{3} S_\iota\,(G_\iota AG_j)\,(G_j x)$$

$$= \sum_{\iota=0,2}^{3} \sum_{j=0,2}^{3} S_\iota \sum_{\iota'=0}^{1} \sum_{j'=0}^{1} S_{\iota'}\,(G_{\iota'}G_\iota AG_jG_{j'})\,(G_{j'}G_j x)$$

LL to Σ-LL



$$[y]_{2,1} = [A]_{2,2}[x]_{2,1} + [y]_{2,1}$$

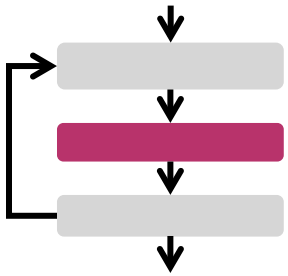


$$S_0\,(G_0 A G_0)\,S_0 \cdot S_0\,(G_0 x) + \cdots + S_2\,(G_2 A G_2)\,S_2 \cdot S_2\,(G_2 x)$$

$$= \sum_{\iota=0,2}^{3} \sum_{j=0,2}^{3} S_\iota\,(G_\iota A G_j)\,(G_j x)$$

$$= \sum_{\iota,j,\iota',j'} S_{\iota+\iota'}\,(G_{\iota+\iota'} A G_{j+j'})\,(G_{j+j'} x)$$

LL to Σ-LL



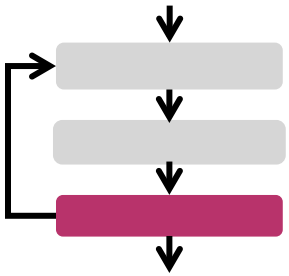
$$[y]_{2,1} = [A]_{2,2}[x]_{2,1} + [y]_{2,1}$$

$$\begin{cases} t = \sum_{\iota,j,\iota',j'} S_{\iota+\iota'} \left(G_{\iota+\iota'} A G_{j+j'}\right) \left(G_{j+j'} x\right) \\ y = \sum_{i,i'} S_{i+i'} \left(G_{i+i'} t + G_{i+i'} y\right) \end{cases}$$

$$G_{i+i'} \sum_{\iota,j,\iota',j'} S_{\iota+\iota'} \left(\cdots\right) = \left(G_{i+i'} A G_{j+j'}\right) \left(G_{j+j'} x\right)$$

$$y = \sum_{i,i',j,j'} S_{i+i'} \left[\left(G_{i+i'} A G_{j+j'}\right) \left(G_{j+j'} x\right) + G_{i+i'} y\right]$$

Σ-LL to C-IR



$$y = \sum_{i,i',j,j'} S_{i+i'} \left[\left(G_{i+i'} A G_{j+j'} \right) \left(G_{j+j'} x \right) + G_{i+i'} y \right]$$

```
...
Mov (Mul A[0,0], x[0,0]), t[0,0]
Mov (Mul A[0,1], x[1,0]), t[1,0]
Mov (Mul A[0,2], x[2,0]), t[2,0]
...
```

Loop unrolling

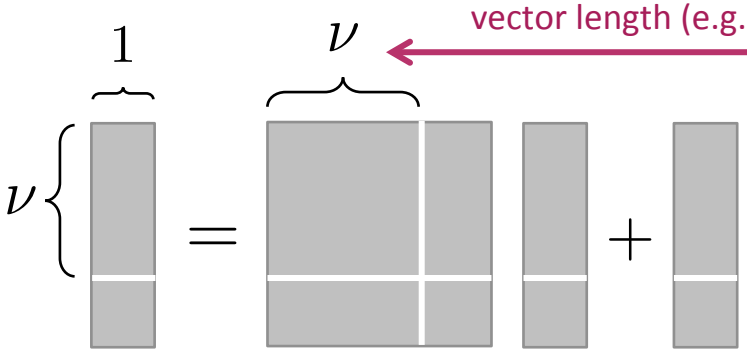
Scalar replacement

SSA normalization

Peephole optimizations

Vector code generation: Basic Idea

vector length (e.g., 1 for scalar float/double, 4 for SSE float)

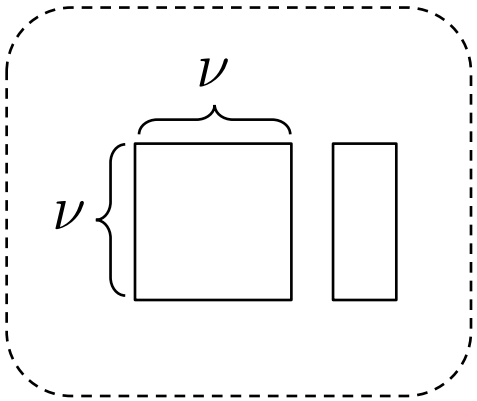


$$[y = Ax + y]_{r,c}$$

$$(r, c) \in \{(1, \nu), (\nu, 1), (\nu, \nu)\}$$

$$[y]_{\nu,1} = [A]_{\nu,\nu}[x]_{\nu,1} + [y]_{\nu,1}$$

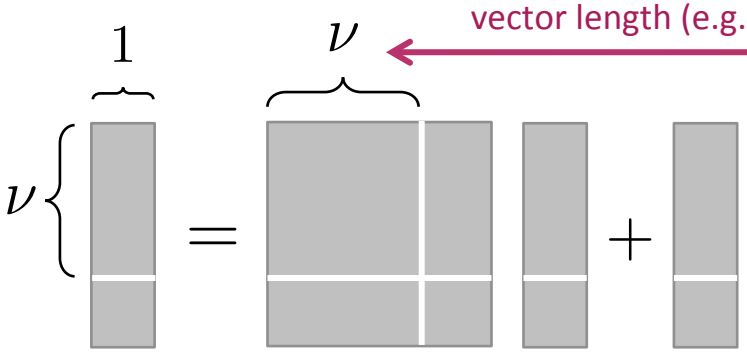
$$y = \sum_{i,j} S_i^{\nu,4} \left[\left(G_i^{\nu,4} A G_j^{\nu,4} \right) \left(G_j^{\nu,4} x \right) + G_i^{\nu,4} y \right]$$



Computation expressed in terms of v-BLACs

Vector code generation: Basic Idea

vector length (e.g., 1 for scalar float/double, 4 for SSE float)

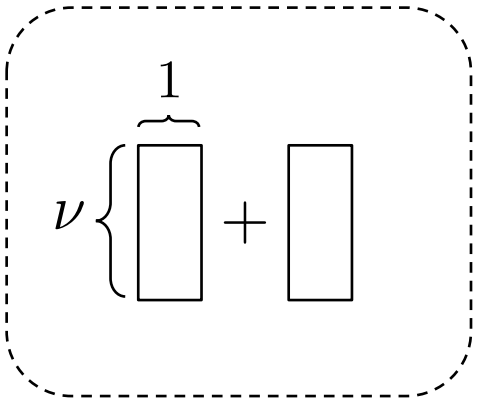


$$[y = Ax + y]_{r,c}$$

$$(r, c) \in \{(1, \nu), (\nu, 1), (\nu, \nu)\}$$

$$[y]_{\nu,1} = [A]_{\nu,\nu}[x]_{\nu,1} + [y]_{\nu,1}$$

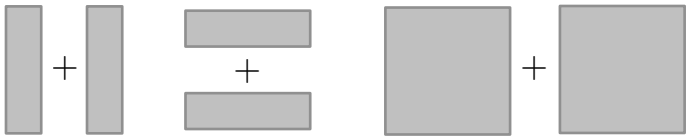
$$y = \sum_{i,j} S_i^{\nu,4} \left[\left(G_i^{\nu,4} A G_j^{\nu,4} \right) \left(G_j^{\nu,4} x \right) + G_i^{\nu,4} y \right]$$



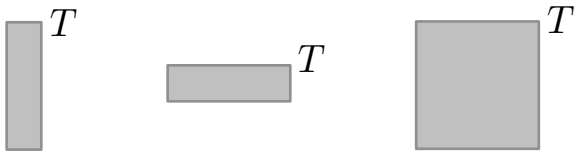
Computation expressed in terms of v-BLACs

ν-BLACs

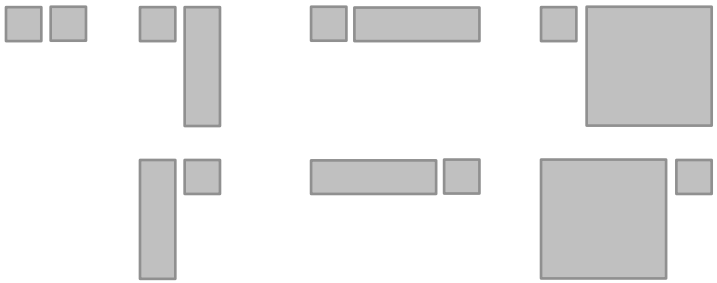
- **Addition (3 ν-BLACs)**



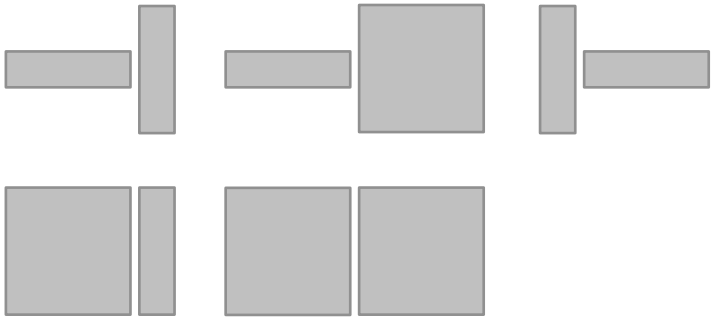
- **Transposition (3 ν-BLACs)**



- **Scalar Multiplication (7 ν-BLACs)**

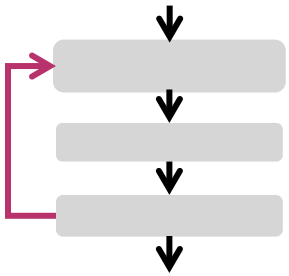


- **Matrix Multiplication (5 ν-BLACs)**

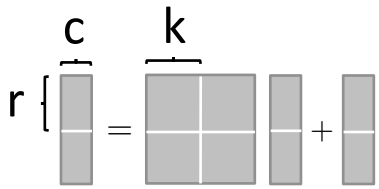


18 cases implemented once for every ISA

Performance evaluation & search



- **Search on tiling strategies**

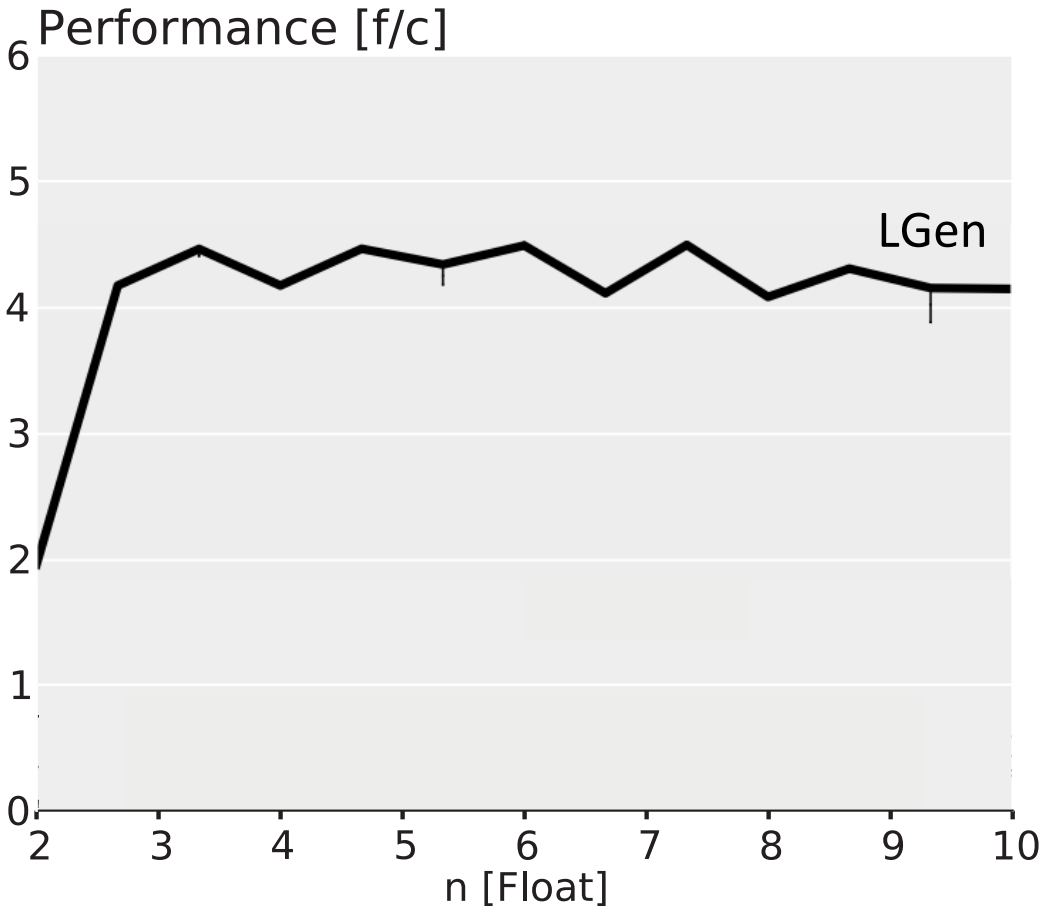


- **Other degrees of freedom:** currently model
- **Current search methods:**
  - exhaustive search
  - random search (in the following: 10 samples)

Experiments

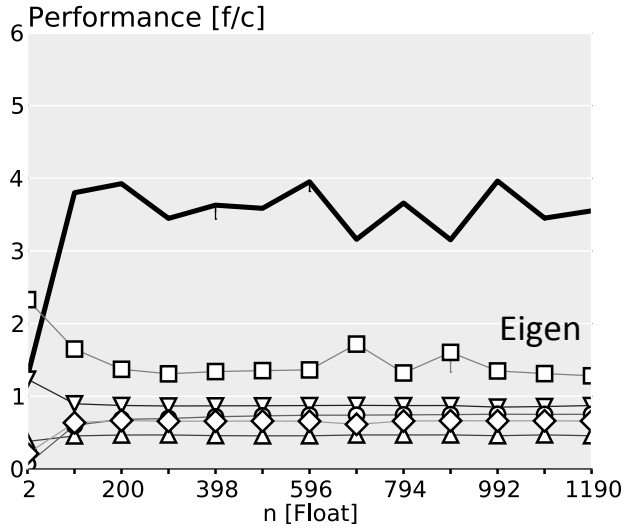
- **Hardware details**
  - Intel Xeon X5680 (Westmere EP) @ 3.3 GHz
  - 32 kB L1 D-cache
  - SSE 4.2 (theoretical peak 8 flops/cycle)
  - Intel's SpeedStep and Turbo Boost disabled
- **Software details**
  - RHEL Server 6 – kernel v. 2.6.32
  - icc v. 13.1 with flags: `-O3 -xHost -fargument-noalias -fno-alias -ip –ipo`
- **Comparisons**
  - Handwritten naïve code: Fixed and general size
  - Libraries: Intel MKL v. 11, Intel IPP v. 7.1
  - Generators: Eigen v.3.1.3, BTO v.1.3

Plotting

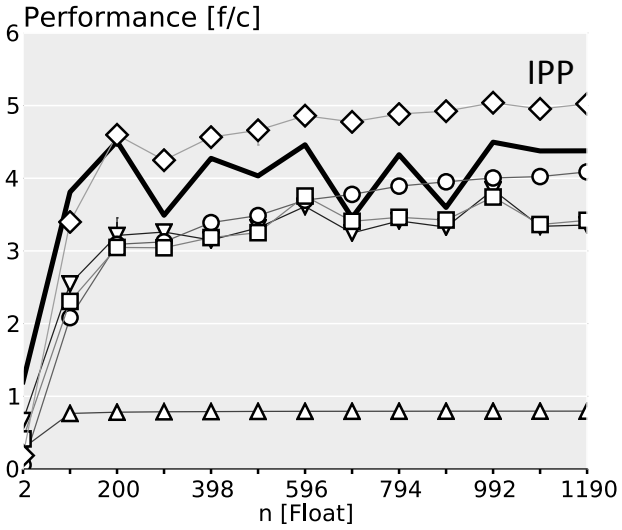


Case 1: Simple BLACs

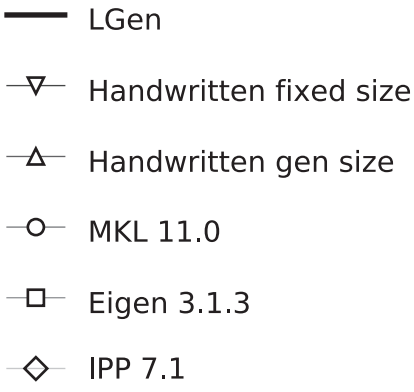
$$y = Ax$$



$$A \in \mathbb{R}^{n \times 4}$$

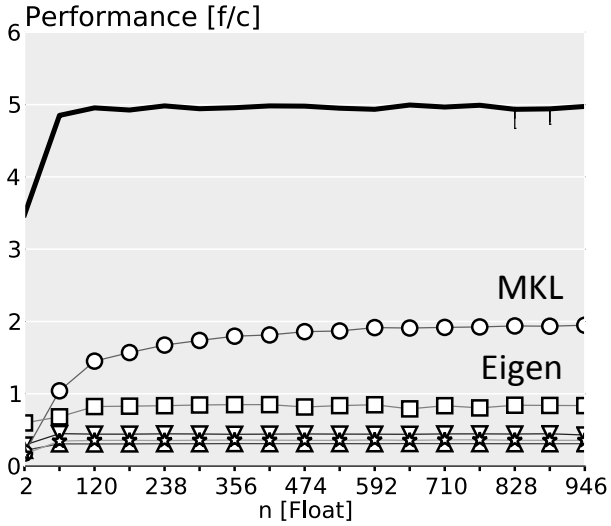


$$A \in \mathbb{R}^{4 \times n}$$



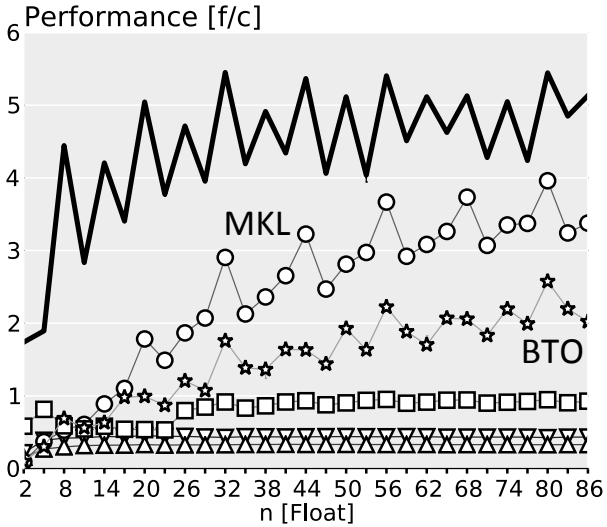
Case 2: BLACs closely matching BLAS

$$C = \alpha AB + \beta C$$



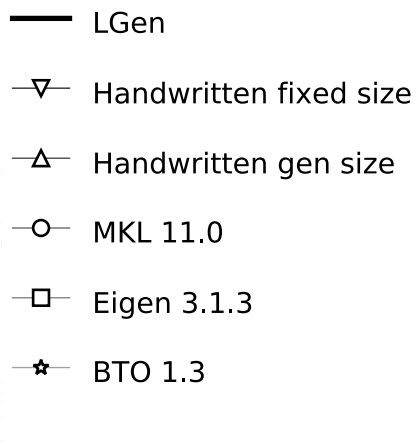
$$A \in \mathbb{R}^{n \times 4}$$

$$B \in \mathbb{R}^{4 \times 4}$$



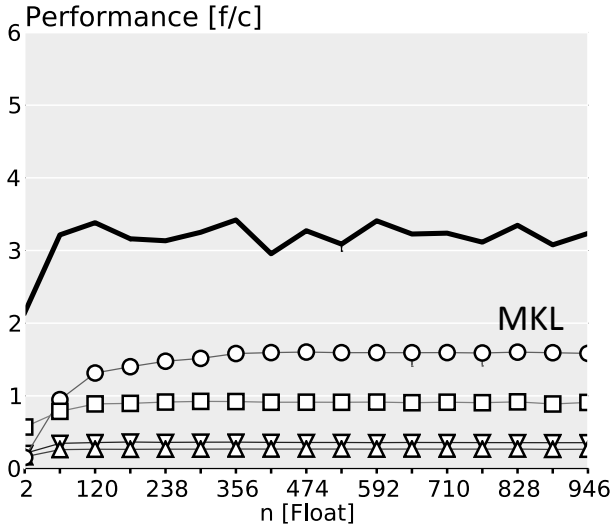
$$A \in \mathbb{R}^{n \times 4}$$

$$B \in \mathbb{R}^{4 \times n}$$



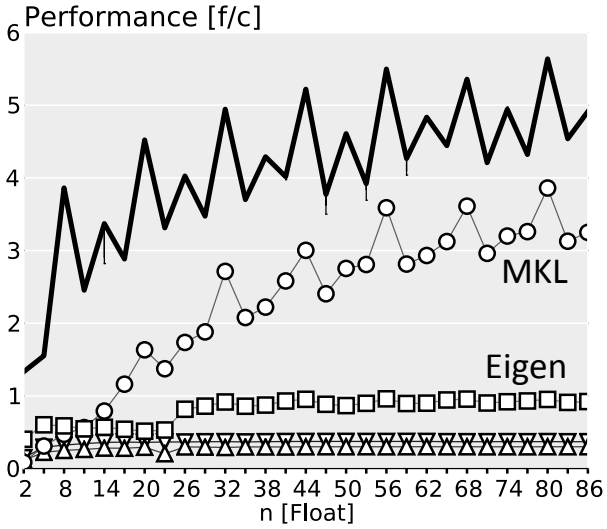
Case 3: More than one BLAS call

$$C = \alpha(A_0 + A_1)^T B + \beta C$$



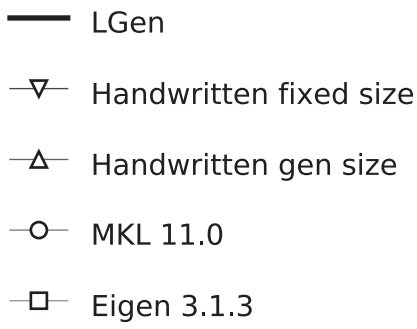
$$A_0 \in \mathbb{R}^{4 \times n}$$

$$B \in \mathbb{R}^{4 \times 4}$$

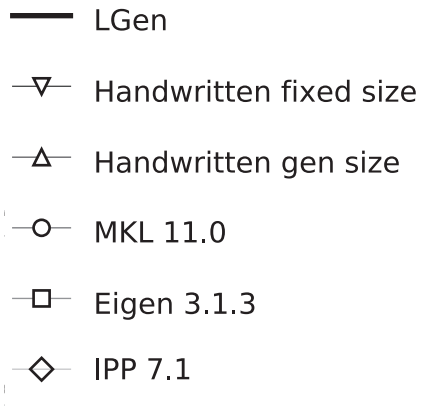


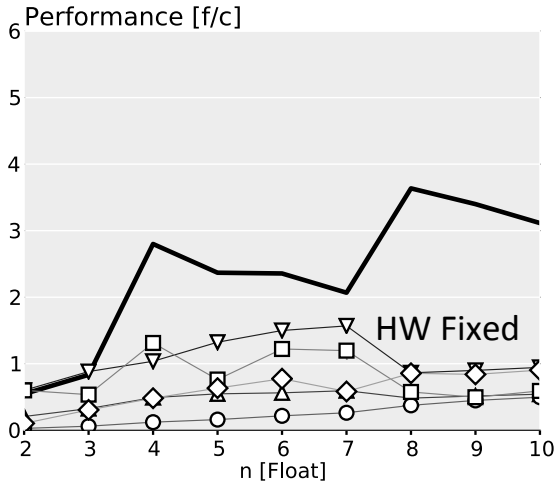
$$A_0 \in \mathbb{R}^{4 \times n}$$

$$B \in \mathbb{R}^{4 \times n}$$

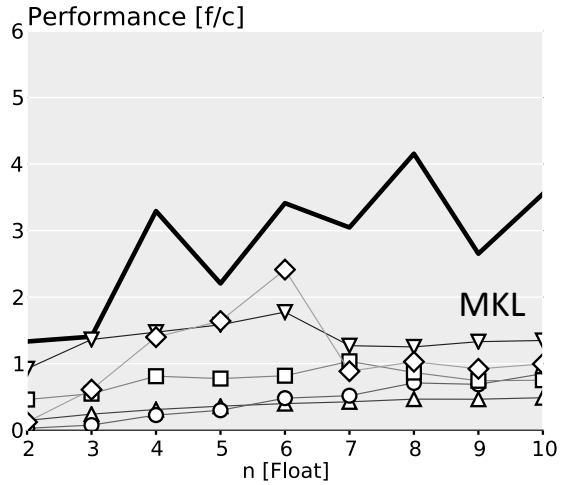


Case 4: Micro BLACs

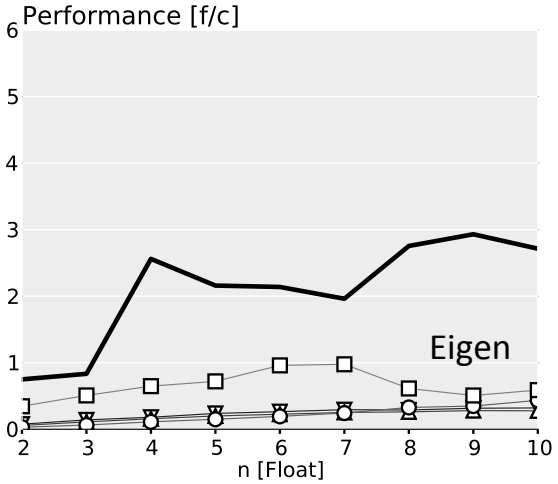




$$y = Ax$$



$$C = AB$$



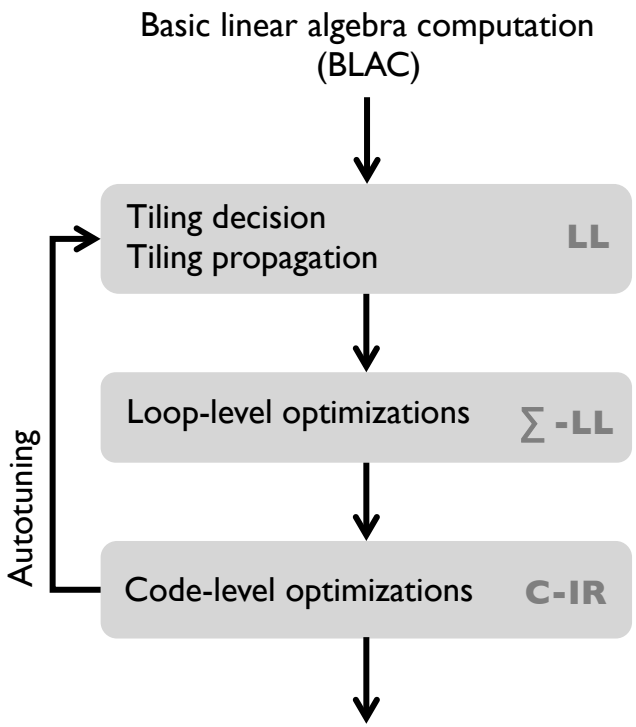
$$\alpha = x^T Ay$$

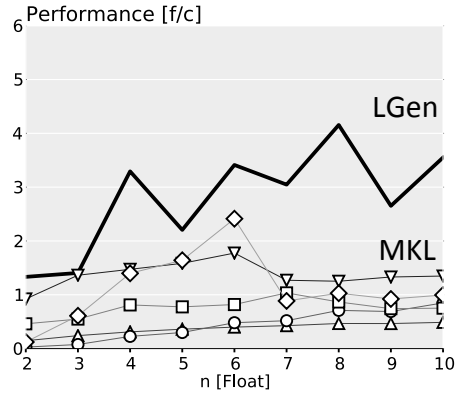
Conclusion

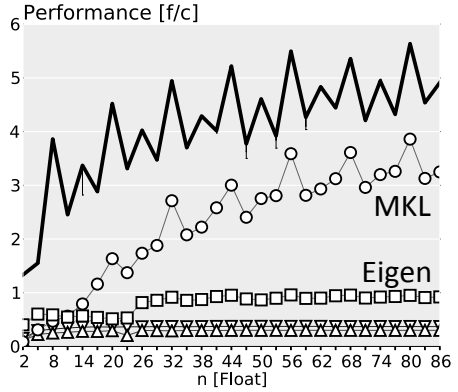
A is 2x3
x is 3x1
$\rightarrow y = Ax$

LGen

```
void f(double …) {
  double t0, …;

  t0 = _mm_loadu_pd(A);
  t1 = _mm_load_sd(A + 2);
  ...
  t6 = _mm_hadd_pd(...)

  _mm_storeu_pd(y, t9);

}
```







Future work: Higher-level functionalities, general size code, better search/model

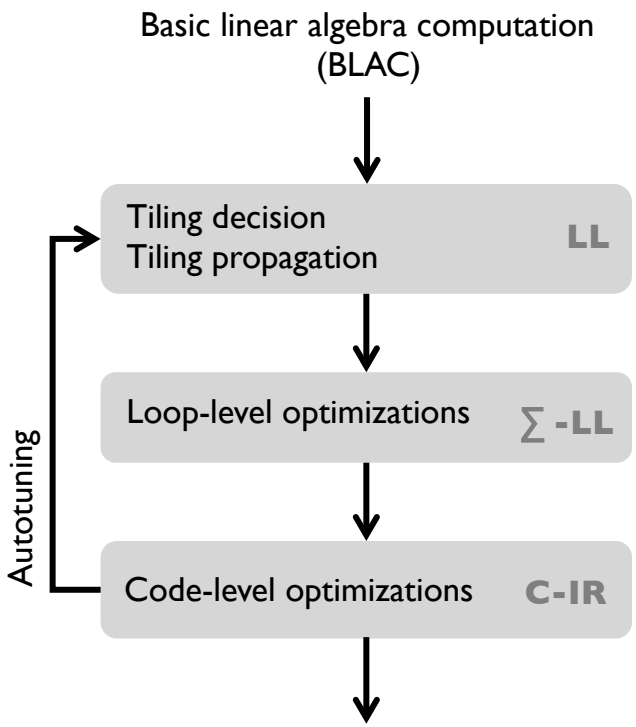
Conclusion

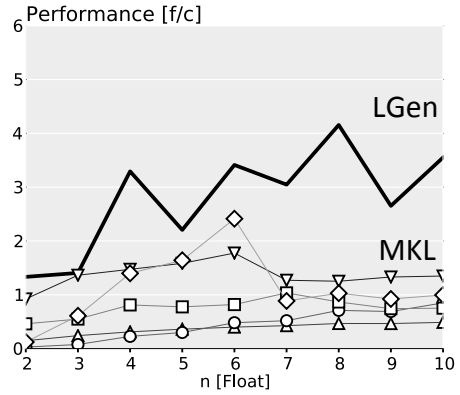
A is 2x3
x is 3x1

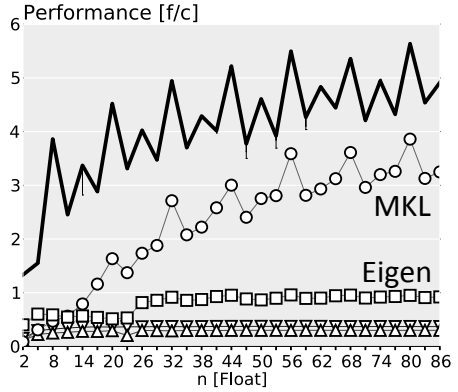
$$y = Ax$$

LGen

```
void f(double …) {
  double t0, …;

  t0 = _mm_loadu_pd(A);
  t1 = _mm_load_sd(A + 2);
  ...
  t6 = _mm_hadd_pd(...)

  _mm_storeu_pd(y, t9);

}
```



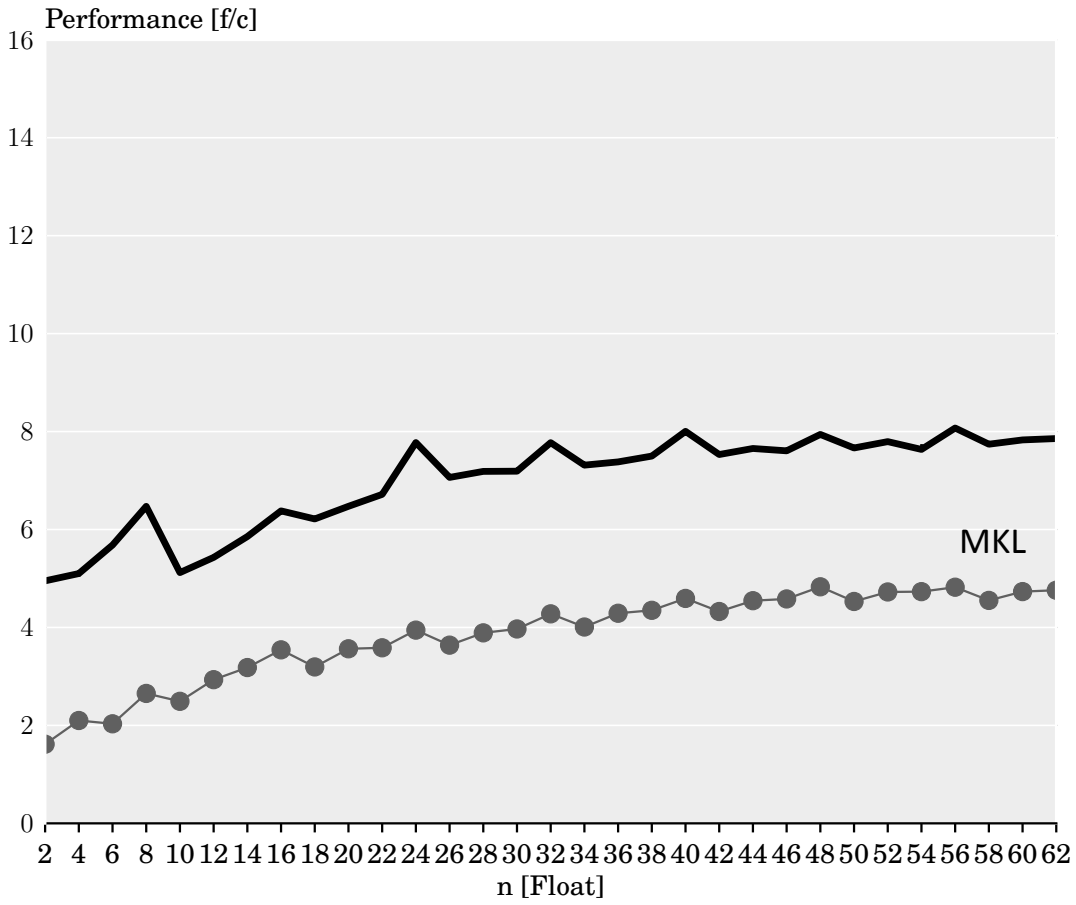


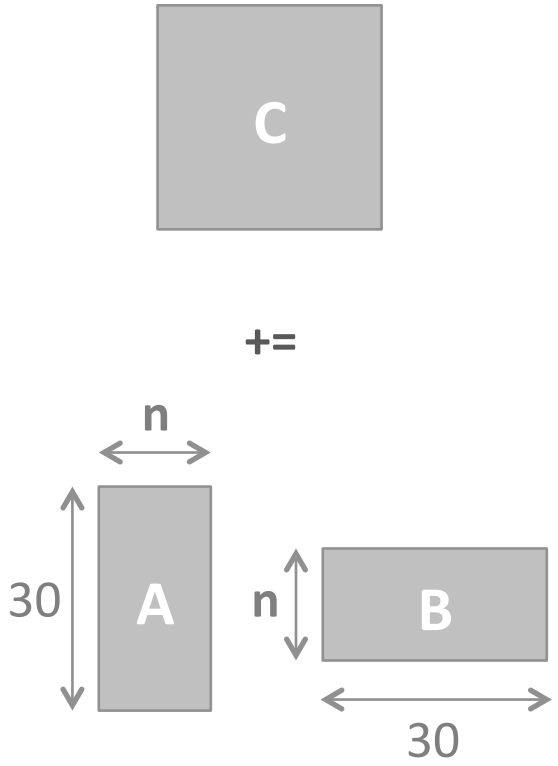


More details: http://spiral.net/software/lgen.html

Sgemm on Sandy Bridge

Intel Core i7-2600 CPU @ 3.40GHz

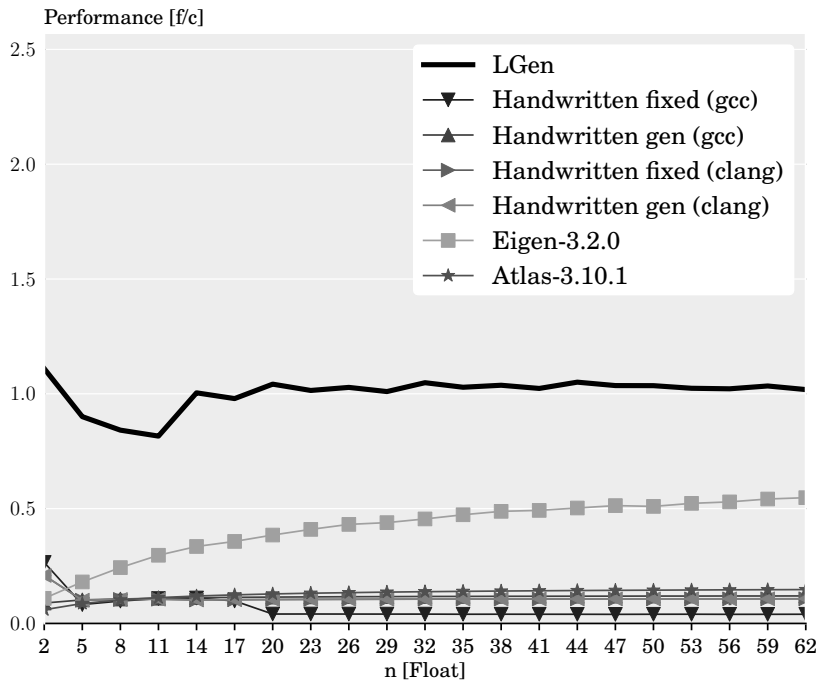




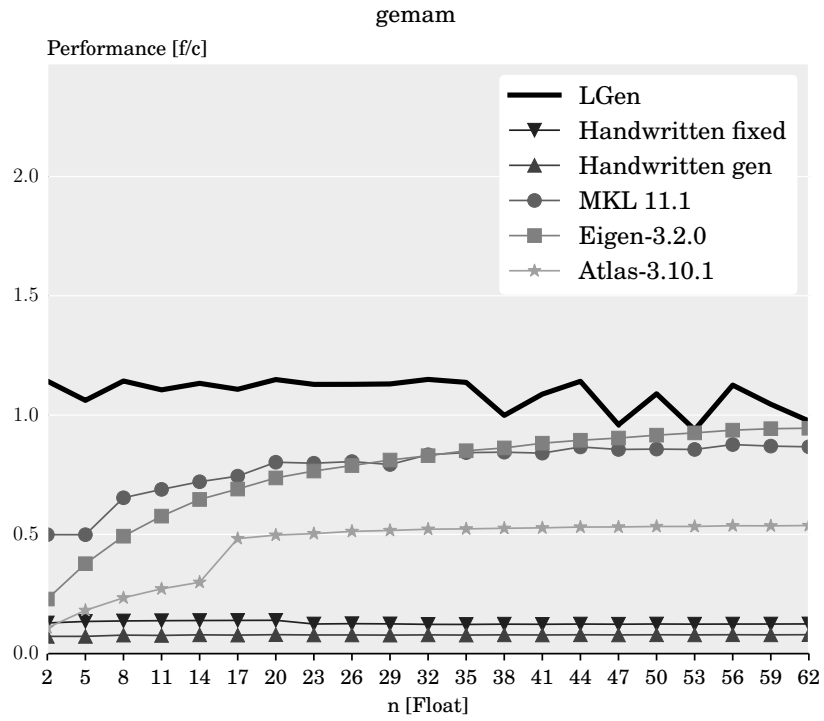
On Embedded Architectures

In collaboration with Nikos Kyrtatas

$$C = \alpha(A_0 + A_1)^T B + \beta C$$



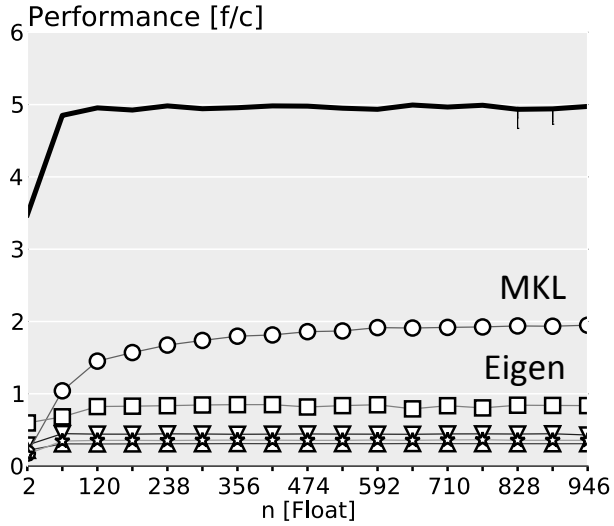
ARM Cortex-A8 @ 1 GHz



Intel Atom D2550 @ 1.86 GHz

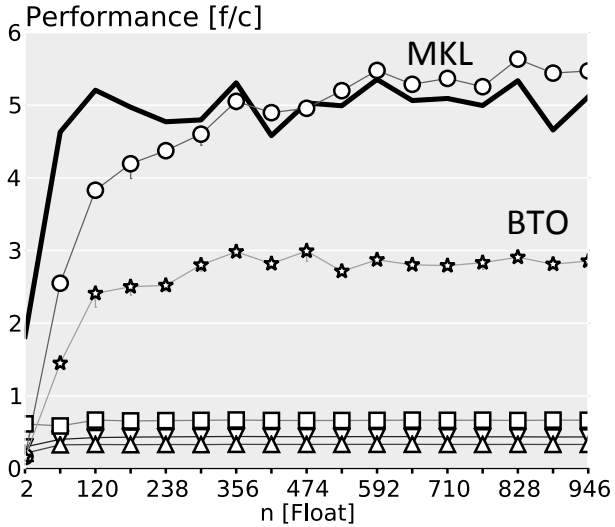
Case 2: BLACs that closely match BLAS

$$C = \alpha AB + \beta C$$



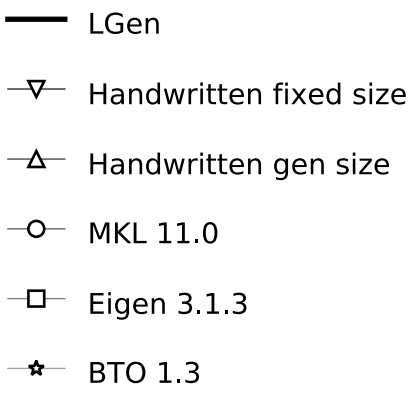
$$A \in \mathbb{R}^{n \times 4}$$

$$B \in \mathbb{R}^{4 \times 4}$$



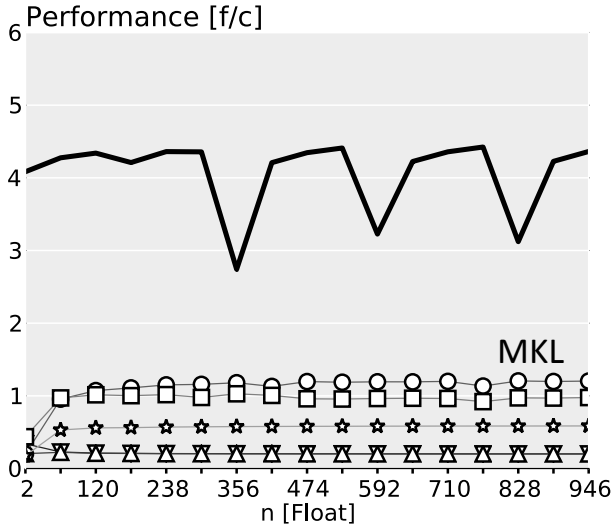
$$A \in \mathbb{R}^{4 \times 4}$$

$$B \in \mathbb{R}^{4 \times n}$$



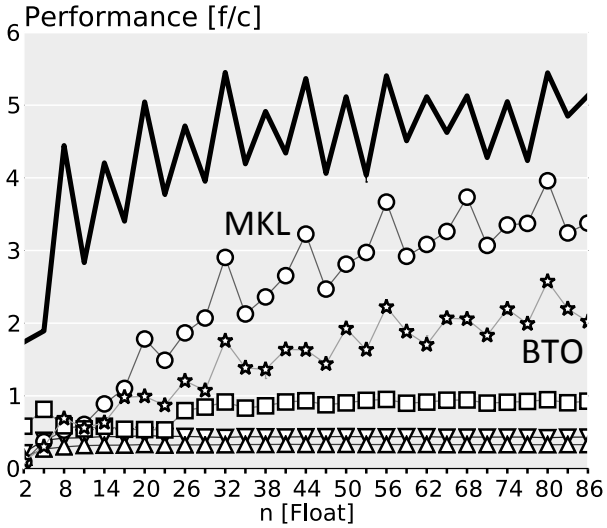
Case 2: BLACs that closely match BLAS

$$C = \alpha AB + \beta C$$



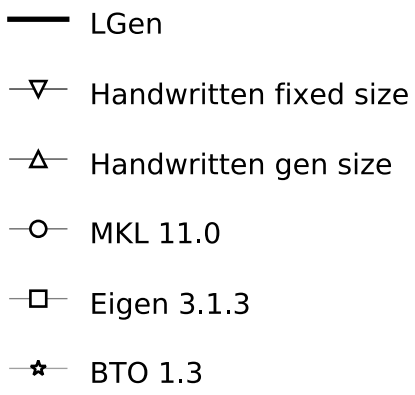
$$A \in \mathbb{R}^{4 \times n}$$

$$B \in \mathbb{R}^{n \times 4}$$



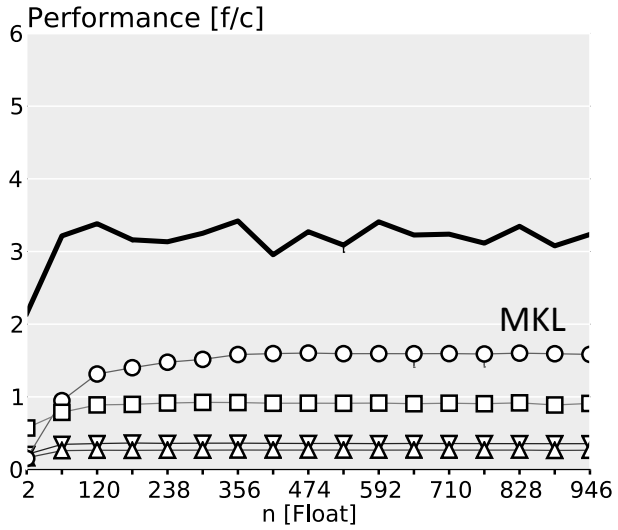
$$A \in \mathbb{R}^{n \times 4}$$

$$B \in \mathbb{R}^{4 \times n}$$



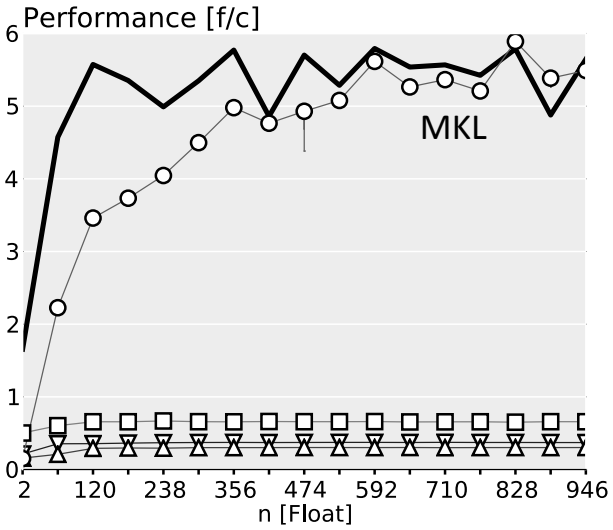
Case 3: More than one BLAS call

$$C = \alpha(A_0 + A_1)^T B + \beta C$$



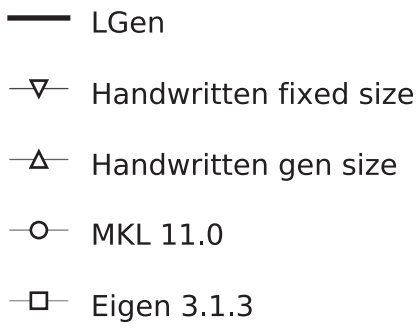
$$A_0 \in \mathbb{R}^{4 \times n}$$

$$B \in \mathbb{R}^{4 \times 4}$$



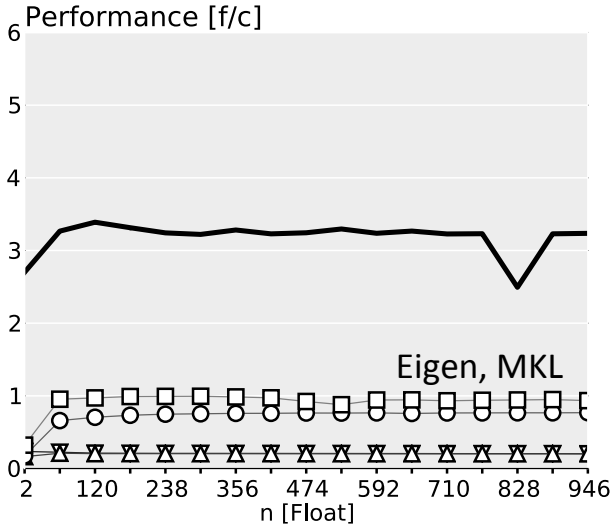
$$A_0 \in \mathbb{R}^{4 \times 4}$$

$$B \in \mathbb{R}^{4 \times n}$$



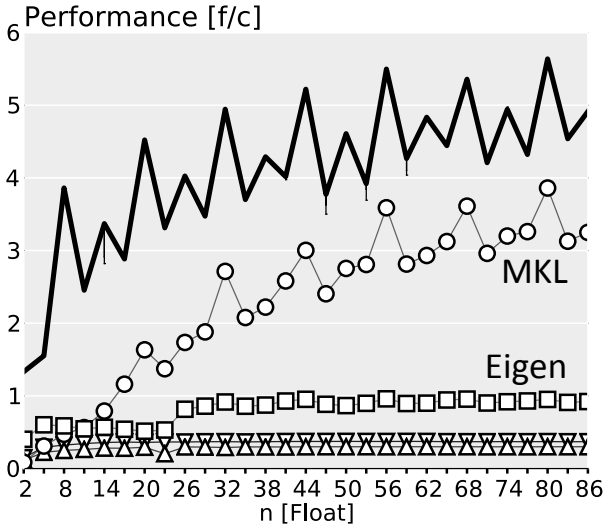
Case 3: More than one BLAS call

$$C = \alpha(A_0 + A_1)^T B + \beta C$$



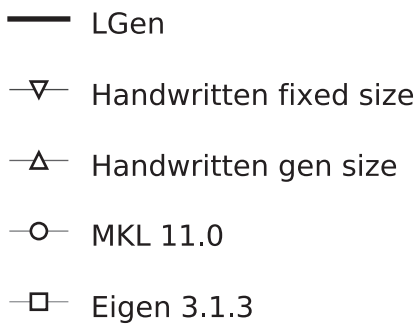
$$A_0 \in \mathbb{R}^{n \times 4}$$

$$B \in \mathbb{R}^{n \times 4}$$



$$A_0 \in \mathbb{R}^{4 \times n}$$

$$B \in \mathbb{R}^{4 \times n}$$



Case 4 + forcing SIMDization

$$y = Ax$$