**LGen**

A Basic Linear Algebra Compiler

Daniele Spampinato  
Markus Püschel

*Computer Science  
ETH Zürich*

---

**Linear algebra: Central to many domains**

**Control systems**

**Optimization algorithms**

**Computer Graphics**

*Quoting App performance section*

“The frame rate of 12 FPS, however, is not enough to handle sudden motions. In future work, optimisation of implementation would improve the frame rate and provide better user experience.”
Library performance for sgemm (C += AB)

Intel MKL on Intel Core i7 CPU (AVX)

91% of peak

A closer look at small problem sizes

Intel MKL on Intel Core i7 CPU (AVX)

28% of peak

Program Generation for Performance: LGen
Math/CS/HPC Workshop, St. Germain au Mont D’Or, May 2016
© Markus Püschel, Computer Science, ETH Zürich
Bridging the gap with DSLs

Kalman Filter

Predict

\[ X = A x_{k-1} + B u_{k-1} \]

\[ p = A P A^T + Q \]

Update

\[ K = p H^T (H p H^T + R)^{-1} \]

\[ z_k = x_k + K (z - H x_k) \]

\[ p x_k = (I - K H) p \]

Goal

```
void kf(double const * A, ...) {
  __m256d t0, ...;
  a0 = _mm256_loadu_pd(A);
  a1 = _mm256_load_sd(A + 4);
  ...
  h0 = _mm256_hadd_pd(m0, m1);
  ...
  p = _mm256_permute2f128_pd(...);
  b = _mm256_blend_pd(t6, t8);
  ...
  _mm256_storeu_pd(X, r0);
  ...
}
```
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

---

**LGen: A basic linear algebra compiler**

Basic linear algebra computation (BLAC) \[ \downarrow \]

\[ y = Ax \] \hspace{1cm} A is 2 x 3

\[ x \text{ is 3 x 1} \]
LGen: A basic linear algebra compiler

$y = Ax \leftarrow A = 2 \times 3$

$x = 3 \times 1$

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

Program Generation for Performance: LGen
Math/CS/HPC Workshop, St. Germain au Mont D’Or, May 2016
© Markus Püschel, Computer Science, ETH Zürich
**LGen: A basic linear algebra compiler**

```
y = Ax \leftarrow \begin{array}{c}
A \text{ is } 2 \times 3 \\
x \text{ is } 3 \times 1
\end{array}
```

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

\[
y = \sum_{i,j} [i] (A[i, j]x[j])
\]

\[
\text{Mov (mmMulPs A(0,0), x(0,0), t[0,0])}
\]

```
for(int i = ... ) {
  ...
  t = _mm_mul_ps(a, x);
  ...
}
```
Vector code generation: Basic Idea

$$\nu \{ \begin{array}{c} \downarrow \nu \\ 1 \end{array} = \nu \Bigg\{ \begin{array}{c} \downarrow \\ 1 \end{array} + \Bigg\}$$

**Goal:** express the computation in terms of $v$-BLACs

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

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

---

v-BLACs: Vectorization building blocks

**Addition** (3 v-BLACs)

$$\begin{array}{c} ++ \\
\end{array}$$

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

$$\begin{array}{c} \underbrace{\vdots} \ldots \underbrace{\vdots} \\
\end{array}$$

**Transposition** (3 v-BLACs)

$$\begin{array}{c} \uparrow^T \\
\end{array}$$

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

$$\begin{array}{c} \underbrace{\vdots} \ldots \underbrace{\vdots} \\
\end{array}$$

*18 cases implemented once for every ISA*
The importance of structures

Kalman Filter
Predict
\[ x_k = Ax_{k-1} +Bu_k \]
\[ P_k = AP_{k-1}A^T + Q \]
Update
\[ K = P_kH^T(HPH^T + R)^{-1} \]
\[ x_k = x_k + K(z_k - Hx_k) \]
\[ P_k = (I-KH)P_k \]
Extending LGen with Structures

Code generation including structures

Structured basic linear algebra computation (sBLAC)

- Tiling decision
- Tiling propagation

LL

- Loop-level optimizations
- Code-level optimizations C-IR

Optimized C function

\[ y = Ax \]

\[ [y]_{2,1} = \sum_{i,j} [i] (A[i, j]x[j]) \]

C-IR

\[ \text{for}(\text{int} i = \ldots) \{ \]
\[ \quad \text{t} = \_\_\_\_\_\_\text{mul}_\text{ps}(a, x); \]
\[ \} \]

A is 2x3 and x is 3x1

A is 2x3 and x is 3x1
Structured matrices representation

Tiling decision
Tiling propagation

General

Upper triangular

Lower triangular

Performance evaluation and search

Structured basic linear algebra computation (sBLAC)

Optimized C function

for(int i = ... ) {
    _...
    t = _mm_mul_ps(a, x);
    _...
}
From LL to Σ-LL

From LL to Σ-LL
From LL to Σ-LL

\[ C[i, j] = A[i, k]B[k, j] \]

\[ C = \sum_{i=0}^{3} \sum_{j=0}^{3} \left( A[i, 0]B[0, j] \right) \]
\[ + \sum_{k=1}^{3} \sum_{i=k}^{3} \sum_{j=k}^{3} \left( A[i, k]B[k, j] \right) \]

CLOoG

Scattering relation built based on optimization models (e.g., Goto model)
Representing structured tiled matrices

From LL to Σ-LL

\[ C = [0, 0] (A[0, 0] B[0, 0]) + [0, 2] (A[0, 0] B[0, 2]) + [2, 0] (A[2, 0] B[0, 0]) + [2, 2] (A[2, 0] B[0, 2] + A[2, 2] B[2, 2]) \]
Optimized C function

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

Structured basic linear algebra computation (sBLAC)

- Tiling decision
- Loop-level optimizations
- Code-level optimizations C-IR

Optimized C function

\[ y = Ax \]

\[ y = \sum_{i,j} [i] (A[i, j]x[j]) \]

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

**Code generation including structures**

Performance evaluation and search
Experimental settings

Intel Core i7 (Sandy Bridge)

Ubuntu 14.04 with Linux 3.13

Double computations with AVX

Warm cache scenario (32 kB L1 D-cache, 256 kB L2 cache)

Four competitors:

- LGen w/ and w/o structures support
- Intel MKL 11.2
- Naive code (non optimized double/triple loop code)

All kernels compiled with icc 15 w/ flags:

-03 -xHost -fargument-noalias -fno-alias

Plotting
Plotting
BLAS category: dsyrk

\[ S_u = AA^T + S_u, \quad A \in \mathbb{R}^{n \times 4} \]

BLAS-like category

\[ A = LU + S_l, \quad L, U \in \mathbb{R}^{n \times n} \]
More general structures & future work

Approach extensibility

Other important structures, e.g., banded matrices

Or other combined structures
Support for higher-level functionalities

Algorithm: \( A := LU_{BLK, VAR}(A) \)

Partition \( A \rightarrow \begin{pmatrix} \text{ATL} & \text{AFR} \\ \text{ABL} & \text{ABR} \end{pmatrix} \)

where \( \text{ATL} \) is \( 0 \times 0 \)

while \( m(\text{ATL}) < m(A) \) do

Determine block size \( b \)

Repartition

\[ \begin{pmatrix} \text{ATL} & \text{AFR} \\ \text{ABL} & \text{ABR} \end{pmatrix} \rightarrow \begin{pmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{pmatrix} \]

where \( A_{11} = b \times b \)

\[ A_{01} := U_{01} = L_{01}^{-1} A_{01} \]

\[ A_{10} := L_{10} = A_{10} L_{00}^{-1} \]

\[ A_{11} := \text{LU}(A_{11} - L_{10} U_{02}) \]

Continue with

\[ \begin{pmatrix} \text{ATL} & \text{AFR} \\ \text{ABL} & \text{ABR} \end{pmatrix} \rightarrow \begin{pmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{pmatrix} \]

d endwhile

Source: FLAME Project - http://www.cs.utexas.edu/~flame

Connecting with Cl1ck (Fabregat-Traver, Bientinesi)

Kalman Filter

\[ X = A_{n-1} + R_{n-1} \]

\[ T = H^T \alpha \]

\[ K = \text{def}(H \beta + R) \]

\[ X_{n} = X_{n-1} + K_{n} \]

\[ P_{n} = (I - K_{n} \alpha) P_{n-1} \]

\[ \text{void lu(...) { } \_m256d t0, ...; \]

\[ ... \]

Conclusion
Our Approach

Kalman Filter

\[
X_{k} = A X_{k-1} + B u_{k-1} \\
\hat{\theta} = \alpha P^{-1} \hat{\theta} \\
X_{k} = P X_{k} [H X_{k} + R]^{-1} \\
\hat{X}_{k} = X_{k} K_{k} (H X_{k} + R) \\
\hat{\theta}_{k} = \hat{\theta}_{k} - K_{k} H \hat{\theta}_{k} \\
\]

void kernel(...) {
    __m256d t0, ...
    ...}

spiral.net/software/lgen.html