Program Generation for Small Linear Algebra

Daniele G. Spampinato
Markus Püschel

Computer Science
ETH *zürich*

PhD May 2017

SPIRAL
www.spiral.net

Kalman filter

Predict

$x_k = Ax_{k-1} + Bu_k$

$P_k = AP_{k-1}A^T + Q$

Update

$x_k = x_k + P_kH^T(HP_kH^T + R)^{-1}(z_k - Hx_k)$

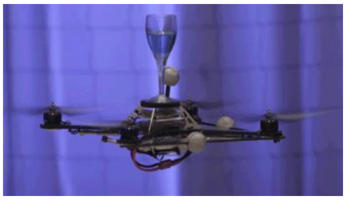
$P_k = P_k - P_kH^T(HP_kH^T + R)^{-1}HP_k$

→ **Fast code needed**

For example, commonly used in robotics
Could be 6, 11, 17, … states

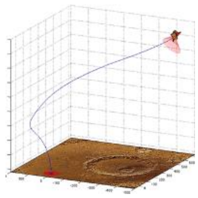
Linear algebra: Central to many domains

Control systems



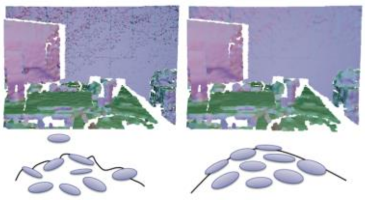
Source: flyingmachinearena.org

Optimization algorithms



Source: rain.aa.washington.edu

Computer graphics



Source: ETH CGL

Computer vision
Communication
Signal Processing
....

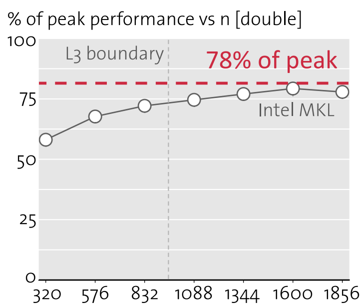
Library performance for DPOTRF

Intel MKL on Intel Core i7 CPU (AVX)

- The Cholesky decomposition

  $U^T U = S$

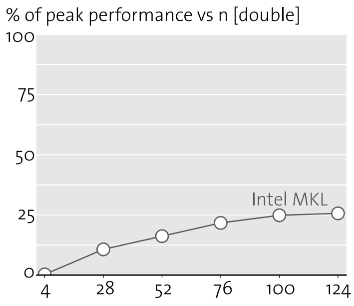
- Function DPOTRF in LAPACK

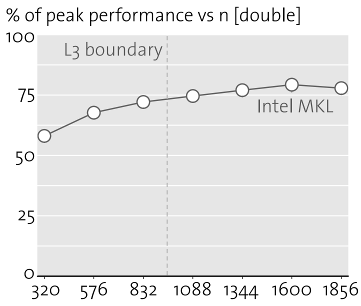


ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Library performance for DPOTRF: $U^T U = S$

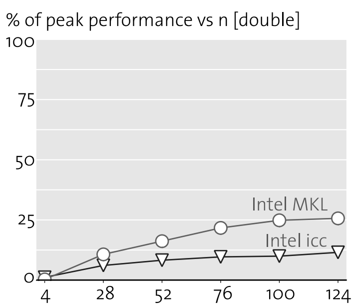
Intel MKL on Intel Core i7 CPU (AVX)

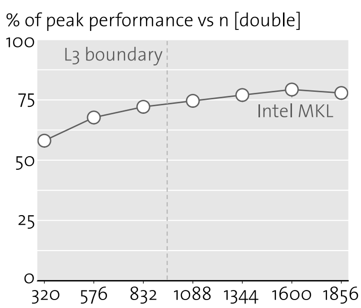




Library performance for DPOTRF: $U^T U = S$

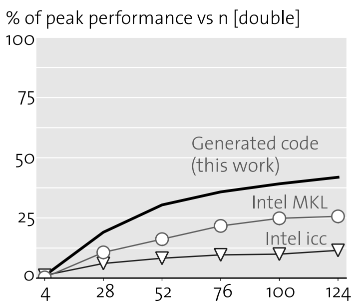
Intel MKL on Intel Core i7 CPU (AVX)

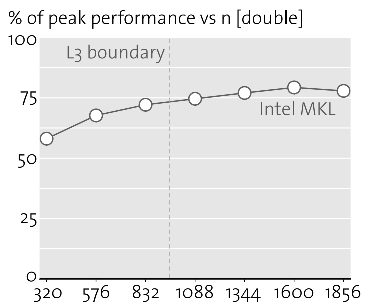




Library performance for DPOTRF: $U^T U = S$

Intel MKL on Intel Core i7 CPU (AVX)





Fast code = good algorithm
+ code style
+ locality
+ vectorization
(+ parallelization)

Example:
[LTE Viterbi Decoder]

Goal: Program Generation for *Small* Linear Algebra

Kalman filter

Predict

$x_k = Ax_{k-1} + Bu_k$

$P_k = AP_{k-1}A^T + Q$

Update

$x_k = x_k + P_kH^T(HP_kH^T + R)^{-1}(z_k - Hx_k)$

$P_k = P_k - P_kH^T(HP_kH^T + R)^{-1}HP_k$

```
void kf(double const * A, ...) {
  __m256d t0, …;

  a0 = _mm256_loadu_pd(A);
  a1 = _mm256_load_sd(A + 4);
  ...
  m0 = _mm256_mul_pd(a0, x0);
  ...
  h0 = _mm256_hadd_pd(m0, m1);
  ...
  p = _mm256_permute2f128_pd(...);
  b = _mm256_blend_pd(t6, t8);
  ...
  _mm256_storeu_pd(X, r0);
  ...

}
```

3

Classes of linear algebra computations

Kalman filter

Predict

$x_k = Ax_{k-1} + Bu_k$

$P_k = AP_{k-1}A^T + Q$

Update

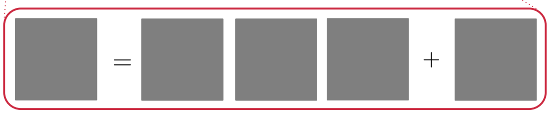
$x_k = x_k + P_kH^T(HP_kH^T + R)^{-1}(z_k - Hx_k)$

$P_k = P_k - P_kH^T(HP_kH^T + R)^{-1}HP_k$

Linear algebra computations

Basic linear algebra computations

BLACs



4

Classes of linear algebra computations

Kalman filter

Predict

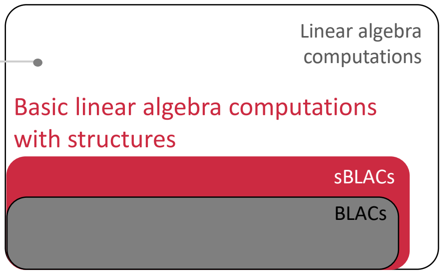
$x_k = Ax_{k-1} + Bu_k$

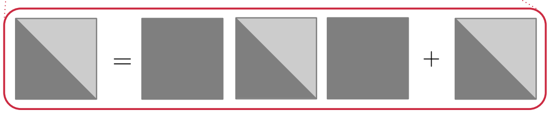
$P_k = AP_{k-1}A^T + Q$

Update

$x_k = x_k + P_kH^T(HP_kH^T + R)^{-1}(z_k - Hx_k)$

$P_k = P_k - P_kH^T(HP_kH^T + R)^{-1}HP_k$





Classes of linear algebra computations

Kalman filter

Predict

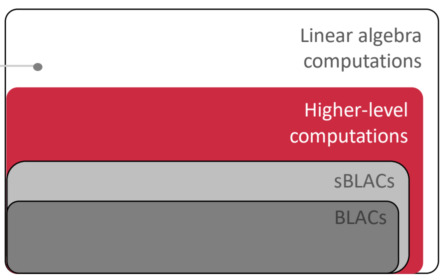
$x_k = Ax_{k-1} + Bu_k$

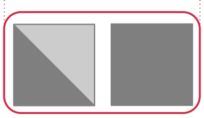
$P_k = AP_{k-1}A^T + Q$

Update

$x_k = x_k + P_kH^T(HP_kH^T + R)^{-1}(z_k - Hx_k)$

$P_k = P_k - P_kH^T(HP_kH^T + R)^{-1}HP_k$





Classes of linear algebra computations

Kalman filter

Predict

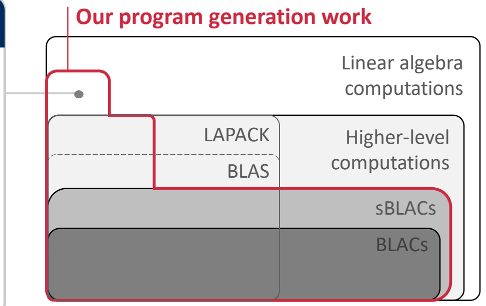
$$x_k = Ax_{k-1} + Bu_k$$

$$P_k = AP_{k-1}A^T + Q$$

Update

$$x_k = x_k + P_kH^T(HP_kH^T + R)^{-1}(z_k - Hx_k)$$

$$P_k = P_k - P_kH^T(HP_kH^T + R)^{-1}HP_k$$



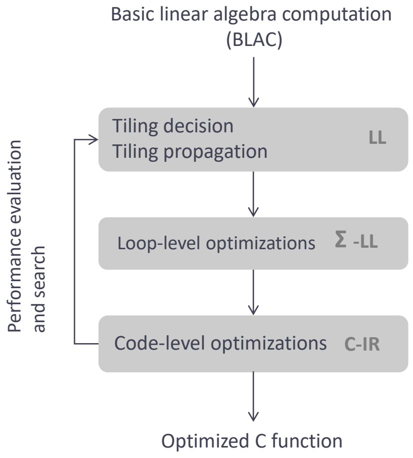
Part 1: BLACs

[CGO 2014,DATE 2015]

Matrices, vectors, scalars
Multiplication, addition, transposition

Example: $y = Ax + \alpha B^T(y + z)$

LGen: A basic linear algebra compiler



$y = Ax \leftarrow$ **A** is 2x3, **x** is 3x1

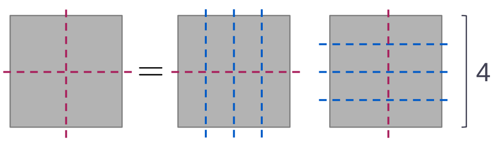
$[y]_{2,1} = [A]_{2,2}[x]_{2,1}$

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

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

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


Tiling in LL – targeting scalar code



$C = AB$

$[C = AB]_{r,c}$ ← First tiling decision (register level)

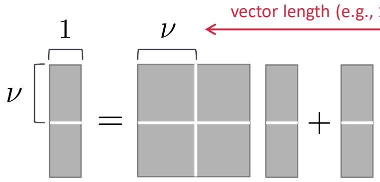
$[C = AB]_{2,2}$

$[C]_{2,2} = [AB]_{2,2}$

$[C]_{2,2} = [A]_{2,k}[B]_{k,2}$ —Choice of **k**→ $[C]_{2,2} = [A]_{2,1}[B]_{1,2}$

Vector code generation: General idea

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



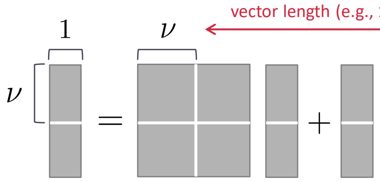
Goal: First level of tiling to express the computation in terms of ν-BLACs

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

9

Vector code generation: General idea

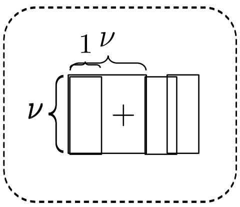
vector length (e.g., 1 for scalar float/double, 4 for SSE float)



Goal: First level of tiling to express the computation in terms of ν-BLACs

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

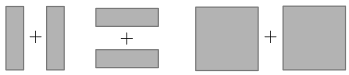
$$y = \sum_{i,j} [i]\left(A[i,j]x[j] + y[i]\right)$$



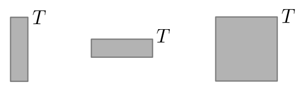
9

v-BLACs: Vectorization building blocks

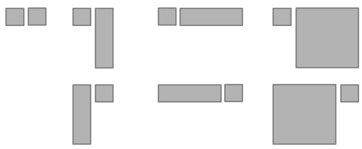
Addition (3 v-BLACs)



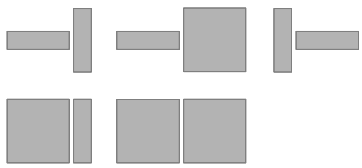
Transposition (3 v-BLACs)



Scalar Multiplication (7 v-BLACs)

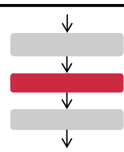


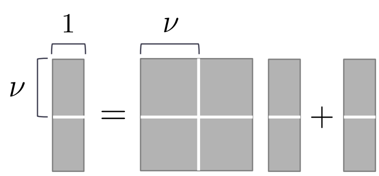
Matrix Multiplication (5 v-BLACs)



18 cases implemented once for every ISA

Vector code generation: General idea





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

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

Scatter Gathers

Σ -LL: Basics

Extension of Σ-SPL [Franchetti et al., PLDI 2005]

- Gathers: Extracting blocks

$$A = \begin{bmatrix} B & \\ & \end{bmatrix} \qquad B = A[0,0]^{4,4}_{2,2}$$

- Scatters: Expanding blocks

$$C = \begin{bmatrix} B & 0 \\ 0 & 0 \end{bmatrix} \qquad C = {}^{2,2}_{4,4}[0,0]B$$

Gather and scatter operators identify explicit data accesses

Σ-LL to C-IR

$$y = Ax + y$$
$$= \sum_i \sum_j [i]\,(A[i,j]x[j] + y[i])$$

GenC-IR(ISA=SSE2, P=double)

```
ForLoop ( i = 0; i < 4; i+=2 ) [
     ForLoop ( j = 0; j < 4; j+=2 ) [
          Ar0 = load(A[i,j], [0,1], hor)
          Ar1 = load(A[i+1,j], [0,1], hor)
          vx  = load(x[j], [0,1], ver)

          store(mmHaddPd(mmMulPd(Ar0, vx), mmMulPd(Ar1, vx)), ty, [0,1])

          vy  = load(y[j], [0,1], ver)

          store(mmAddPd(ty, vy), y[j], [0,1], ver)
     ]
]
```

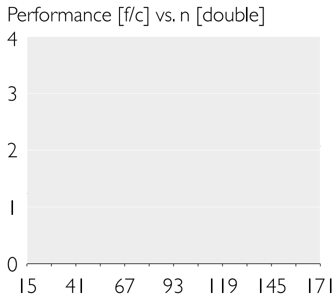
C-IR optimizations

- Loop unrolling
- Scalar replacement
- SSA normalization
- Alignment detection
- …

Plotting

Intel core i7 (Sandy Bridge), Linux 3.13

L1-D	L2	Vec. ISA	Th. Peak
32 kB	256 kB	AVX	8 f/c

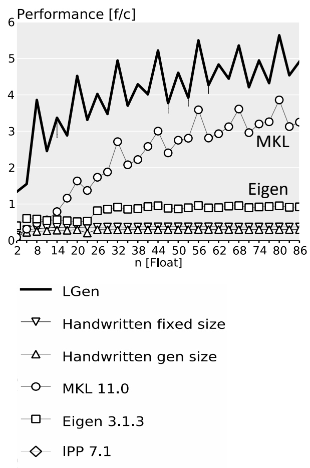


All experiments are executed in a warm-cache scenario

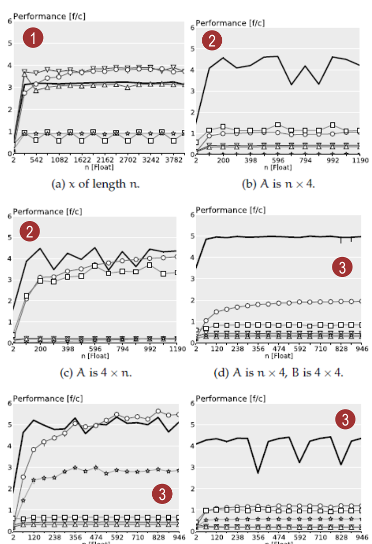
Intel Xeon X5680 (Westmere EP)

L1-D	32 kB
Vec. ISA	SSE 4
Th. Peak	8 f/c

$C = \alpha(A_0 + A_1)^T B + \beta C, \qquad A_0, A_1, B \in \mathbb{R}^{4\times n}$



BLAS 1-3 examples



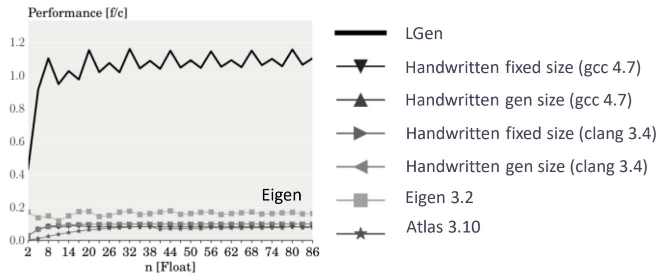
(a) x of length n.
(b) A is n × 4.
(c) A is 4 × n.
(d) A is n × 4, B is 4 × 4.
(e) A is 4 × 4, B is 4 × n.
(f) A is 4 × n, B is n × 4.

ARM Cortex A8

With N. Kyrtatas

L1-D	32 kB
Vec. ISA	Neon
Th. Peak	4 f/c

$$C = \alpha(A_0 + A_1)^T B + \beta C, \qquad A_0, A_1, B \in \mathbb{R}^{4\times n}$$



32

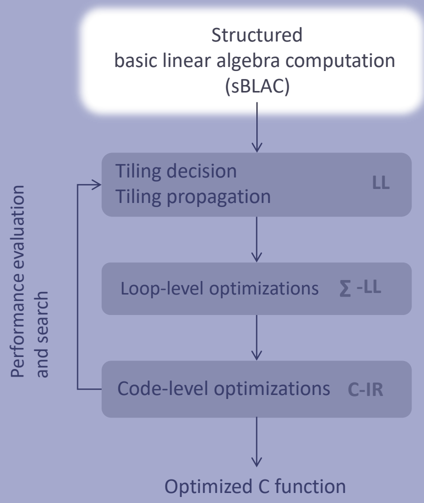
Part 2: sBLACs

[CGO 2016]

BLACs + Structured matrices

Example: $A = LU + S + xx^T$

LGen with Structure



$y = Ax \leftarrow$ **A** is 2x3, **x** is 3x1 and

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

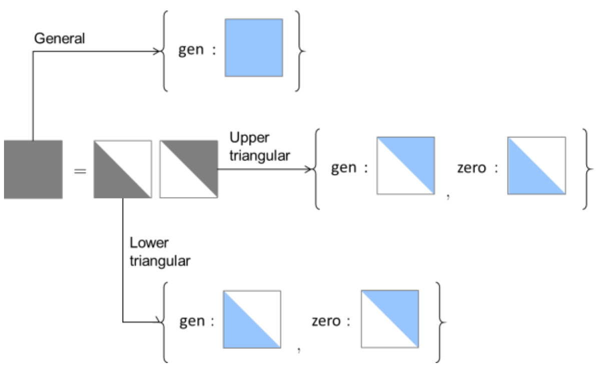
$$y = \sum_{i,j} [i] \left(A[i,j] x[j] \right)$$

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

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

7

Structured Matrix Representation

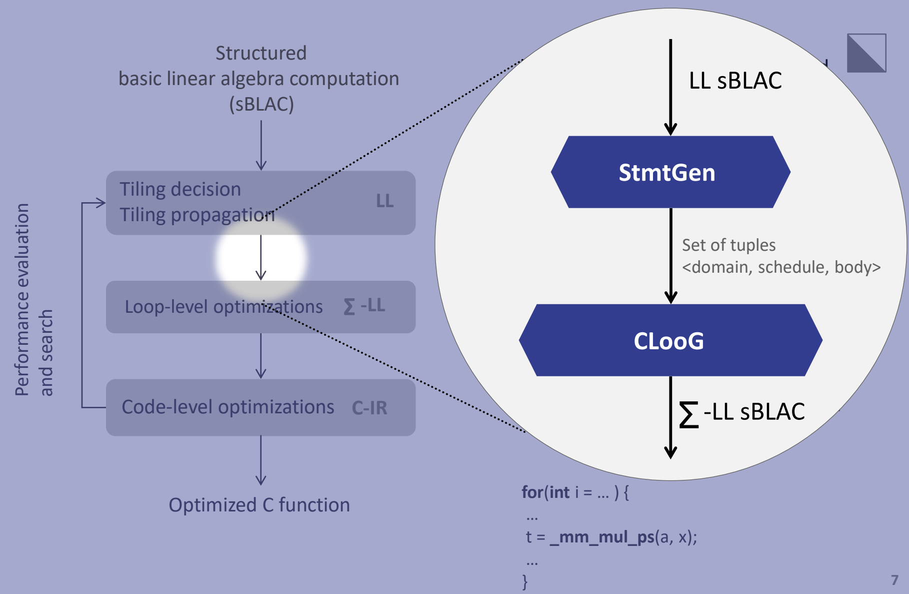


L: $\texttt{L.SInfo} = \begin{Bmatrix} \mathcal{G} : \{(i,j) | 0 \le i < 4 \wedge 0 \le j \le i\} \\ \mathcal{Z} : \{(i,j) | 0 \le i < 4 \wedge i < j < 4\} \end{Bmatrix}$ $\texttt{L.AInfo} = \left\{ \{(i,j) | 0 \le i < 4 \wedge 0 \le j \le i\} : ([i,j]^{4,4}_{1,1}, id) \right\}$

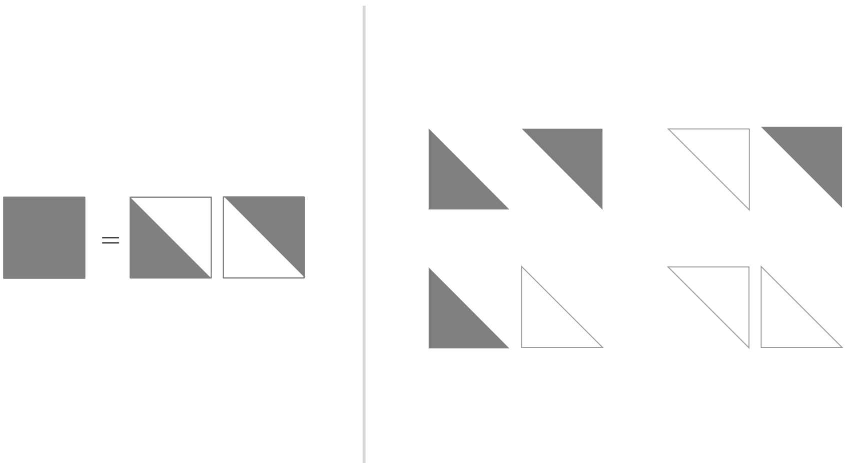
S: $\texttt{S.SInfo} = \{\mathcal{G} : \{(i,j) | 0 \le i, j < 4\}\}$ $\texttt{S.Ainfo} = \begin{Bmatrix} \{(i,j) | 0 \le i < 4, 0 \le j \le i\} : ([i,j]^{4,4}_{1,1}, id) \\ \{(i,j) | 0 \le i < 4, i < j < 4\} : ([j,i]^{4,4}_{1,1}, id) \end{Bmatrix}$

Verdoolaege, ISL: an integer set library for the polyhedral model [MS 2010]

LGen with Structure

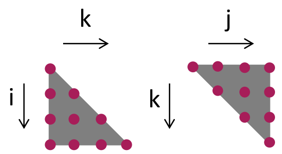


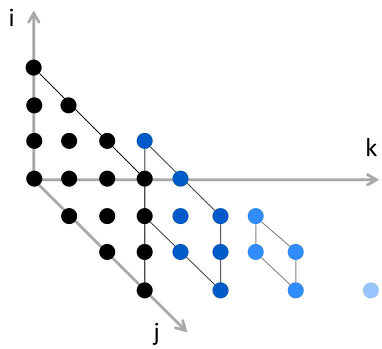
From LL to Σ-LL



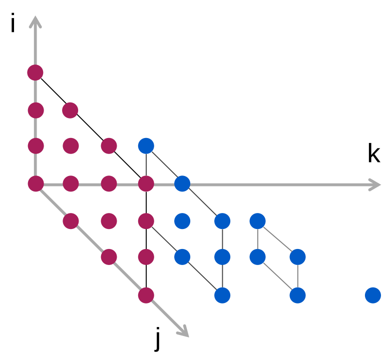
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

From LL to Σ-LL





From LL to Σ-LL



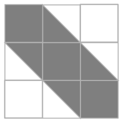
CLooG

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

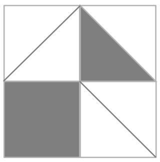
Loop order built based on known models (e.g., Goto model)

Extensibility

- Other important structures, e.g., banded matrices



- Or other combined structures

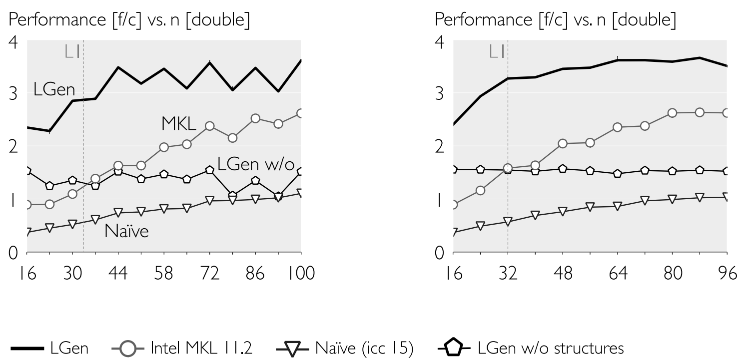


BLAS-like category

Intel core i7 (Sandy Bridge), Linux 3.13

L1-D	L2	Vec. ISA	Th. Peak
32 kB	256 kB	AVX	8 f/c

$$A = LU + S, \qquad L, U \in \mathbb{R}^{n \times n}$$



Part 3: Higher level linear algebra

[CGO 2018]

Cholesky factorization
LU factorization
Triangular solve
…

Collaboration:

Diego Fabregat-Traver Paolo Bientinesi

RWTH Aachen

Cholesky Factorization

Algorithm 2.13 The Cholesky decomposition.
$U^T U = P, \quad U \in \mathcal{U}_n$, and P is SPD.
U overwrites the upper half of P. Cost $\approx n^3/3$ flops.

Partition $P \rightarrow \left(\begin{array}{c|c} P_{TL} & P_{TR} \\ \hline P_{BL} & P_{BR} \end{array}\right)$
where P_{TL} **is** 0×0
while $\text{size}(P_{TL}) < \text{size}(P)$ **do**
Repartition

$$\left(\begin{array}{c|c} P_{TL} & P_{TR} \\ \hline P_{BL} & P_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} P_{0,0} & p_{0,1} & P_{0,2} \\ \hline p_{1,0}^T & \pi_{1,1} & p_{1,2}^T \\ \hline P_{2,0} & p_{2,1} & P_{2,2} \end{array}\right)$$

where $\pi_{1,1}$ **is** 1×1

$\pi_{1,1} := \pi_{1,1} - p_{0,1}^T p_{0,1}$
$\pi_{1,1} := \sqrt{\pi_{1,1}}$
$p_{1,2}^T := p_{1,2}^T - p_{0,1}^T P_{0,2}$
$p_{1,2}^T := (1/\pi_{1,1}) p_{1,2}^T$

Continue with

$$\left(\begin{array}{c|c} P_{TL} & P_{TR} \\ \hline P_{BL} & P_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} P_{0,0} & p_{0,1} & P_{0,2} \\ \hline p_{1,0}^T & \pi_{1,1} & p_{1,2}^T \\ \hline P_{2,0} & p_{2,1} & P_{2,2} \end{array}\right)$$

endwhile

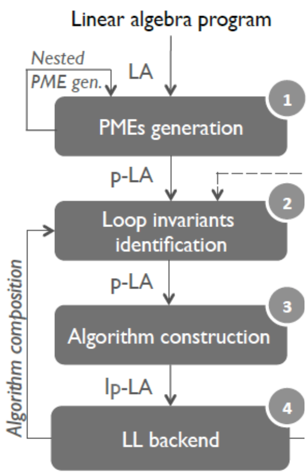
Algorithm synthesized by Cl1ck:

Fabregat & Bientinesi
[ICCSA 2011, CASC 2011]

Based on FLAME methodology
Bientinesi et al. [ACM TOMS 2005]
http://www.cs.utexas.edu/~flame

Requires availability of BLAS

How Cl1ck Works

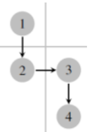


$X^T X = A$ X triangular, A symmetric pos def

$$\begin{pmatrix} X_{TL} & X_{TR} \\ 0 & X_{BR} \end{pmatrix} \text{ and } \begin{pmatrix} A_{TL} & A_{TR} \\ A_{BL} & A_{BR} \end{pmatrix}$$

$$\begin{pmatrix} X_{TL}^T X_{TL} = A_{TL} & X_{TL}^T X_{TR} = A_{TR} \\ * & X_{TR}^T X_{TR} + X_{BR}^T X_{BR} = A_{BR} \end{pmatrix}$$

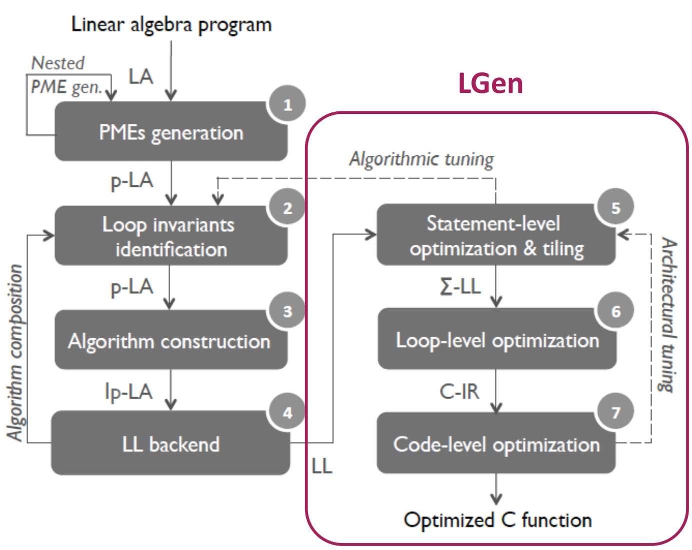
1. X_{TL} = CHOL(A_{TL})
2. X_{TR} = TRSM(X_{TL}^T, A_{TR})
3. T_1 = A_{BR} - X_{TR}^T * X_{TR}
4. X_{BR} = CHOL(T_1)



Loop invariant

```
program CHOL
  Matrix XTL <k, k, Output, UpperTriangular, Overwrites(ATL)>
  Matrix XTR <k, r, Output, Overwrites(ATR>
  Matrix ATL <k, k, Input, Symmetric, PositiveDefinite>
  Matrix ATR <k, r, Input>
  ( XTL = CHOL(ATL) | XTR = TRSM(XTL^T, ATR) )
end
```

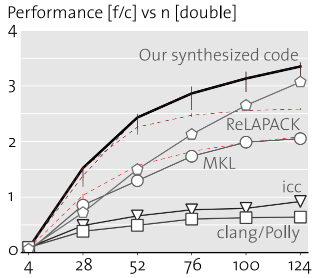
Connecting Cl1ck with LGen



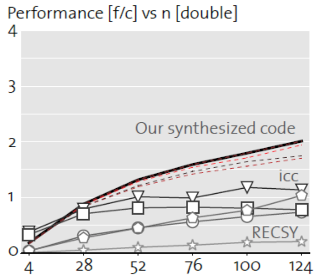
Intel core i7 (Sandy Bridge), Linux 3.13			
L1-D	L2	Vec. ISA	Th. Peak
32 kB	256 kB	AVX	8 f/c

Higher-level Computations

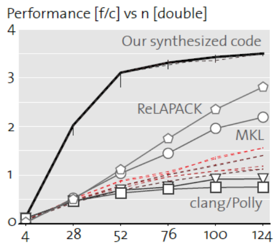
Cholesky $UU^T = S$



Sylvester $AX + XB = C$



- Our synthesized code
- Intel MKL 11.3.2
- ReLAPACK
- Intel icc 16
- clang 4/Polly 3.9
- RECSY '09



Triangular inverse
$X = L^{-1}$

Part 4: Linear algebra programs

[CGO 2018]

Kalman filter

Predict

$$x_k = Ax_{k-1} + Bu_k$$
$$P_k = AP_{k-1}A^T + Q$$

Update

$$x_k = x_k + P_k H^T (HP_k H^T + R)^{-1}(z_k - Hx_k)$$
$$P_k = P_k - P_k H^T (HP_k H^T + R)^{-1} HP_k$$

Grammar

⟨la-program⟩ ::= {⟨declaration⟩} {⟨statement⟩}

⟨declaration⟩ ::= 'Mat' ⟨id⟩ '(' ⟨size⟩ ',' ⟨size⟩ ')' '<' ⟨iotype⟩ { ',' ⟨property⟩ } [',' ⟨ow⟩] '>;'
| 'Vec' ⟨id⟩ ... | 'Sca' ⟨id⟩ ...

⟨iotype⟩ ::= 'In' | 'Out' | 'InOut'

⟨property⟩ ::= 'LoTri' | 'UpTri' | 'UpSym' | 'LoSym'
| 'PD' | 'NS' | 'UnitDiag'

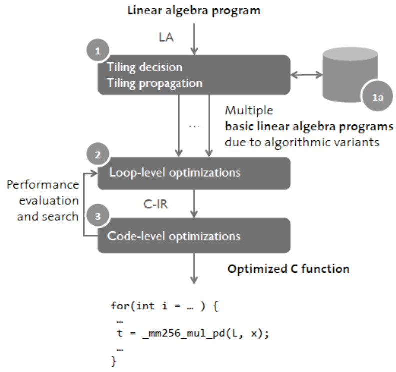
⟨ow⟩ ::= 'ow(' ⟨id⟩ ')'

⟨statement⟩ ::= ⟨for-loop⟩ | ⟨sBLAC⟩ | ⟨HLAC⟩ ';'

⟨for-loop⟩ ::= 'for (i = ...) {' {⟨statement⟩$_i$ } '}'

⟨sBLAC⟩ ::= ⟨id⟩ '=' ⟨expression⟩

⟨HLAC⟩ ::= ⟨expression⟩ '=' ⟨expression⟩
| ⟨id⟩ '=' '(' ⟨id⟩ ')$^{-1}$'



Intel core i7 (Sandy Bridge), Linux 3.13			
L1-D	L2	Vec. ISA	Th. Peak
32 kB	256 kB	AVX	8 f/c

Linear Algebra Computations

Kalman filter

Predict

$x_k = Ax_{k-1} + Bu_k$

$P_k = AP_{k-1}A^T + Q$

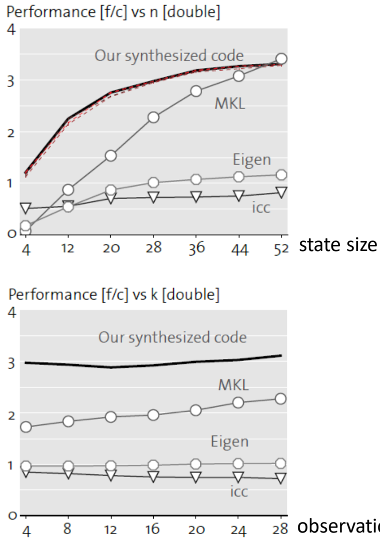
Update

$x_k = x_k + P_kH^T(HP_kH^T + R)^{-1}(z_k - Hx_k)$

$P_k = P_k - P_kH^T(HP_kH^T + R)^{-1}HP_k$

Input: $F, B, Q, H, R, P, u, x, z$
Output: P, x

$y = F * x + B * u$;
$Y = F * P * F^T + Q$;
$v_0 = z - H * y$;
$M_1 = H * Y$;
$M_2 = Y * H^T$;
$M_3 = M1 * H^T + R$;
$\underline{U}^T * \underline{U} = M_3$;
$U^T * \underline{v_1} = v_0$;
$U * \underline{v_2} = v_1$;
$U^T * \underline{M_4} = M_1$;
$U * \underline{M_5} = M_4$;
$x = y + M_2 * v_2$;
$P = Y - M_2 * M_5$;



Other Case Studies

Convex cone problem

Algorithm 1 L1-Analysis

$\theta_0 = 1 \quad \mathbf{v}_0^{(1)} = \mathbf{z}_0^{(1)} = 0 \quad \mathbf{v}_0^{(2)} = \mathbf{z}_0^{(2)} = 0$

for $k = 1 \to K$ **do**

$\quad \mathbf{y}_\mathbf{k}^{(1)} = (1-\theta_k)\mathbf{v}_\mathbf{k}^{(1)} + \theta_k \mathbf{z}_\mathbf{k}^{(1)}$

$\quad \mathbf{y}_\mathbf{k}^{(2)} = (1-\theta_k)\mathbf{v}_\mathbf{k}^{(2)} + \theta_k \mathbf{z}_\mathbf{k}^{(2)}$

$\quad \mathbf{x}_\mathbf{k} = \mathbf{x}_0 + \mu^{-1}(W^T \mathbf{y}_\mathbf{k}^{(1)} - A^T \mathbf{y}_\mathbf{k}^{(2)})$

$\quad \mathbf{z}_\mathbf{k+1}^{(1)} = \mathrm{Trunc}(\mathbf{y}_\mathbf{k}^{(1)} - \theta_\mathbf{k}^{-1}\mathbf{t}_\mathbf{k}^{(1)} \mathrm{W}\mathbf{x}_\mathbf{k}, \theta_\mathbf{k}^{-1}\mathbf{t}_\mathbf{k}^{(1)})$

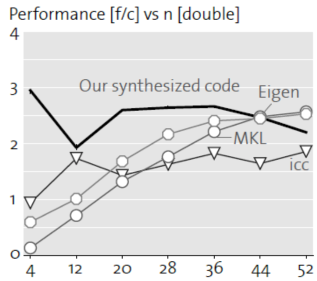
$\quad \mathbf{z}_\mathbf{k+1}^{(2)} = \mathrm{Shrk}(\mathbf{y}_\mathbf{k}^{(2)} - \theta_\mathbf{k}^{-1}\mathbf{t}_\mathbf{k}^{(2)}(\mathbf{y} - \mathrm{A}\mathbf{x}_\mathbf{k}), \theta_\mathbf{k}^{-1}\mathbf{t}_\mathbf{k}^{(2)}\varepsilon)$

$\quad \mathbf{v}_\mathbf{k+1}^{(1)} = (1-\theta_k)\mathbf{v}_\mathbf{k}^{(1)} + \theta_k \mathbf{z}_\mathbf{k+1}^{(1)}$

$\quad \mathbf{v}_\mathbf{k+1}^{(2)} = (1-\theta_k)\mathbf{v}_\mathbf{k}^{(2)} + \theta_k \mathbf{z}_\mathbf{k+1}^{(2)}$

$\quad \theta_{k+1} = \frac{1}{2}\left(1 + \sqrt{1 + \frac{4}{\theta_k^2}}\right)^{-1}$

end for



Gaussian process regression

input: X (inputs), $\mathbf{y}$ (targets), k (covariance function), σ_n^2 (
$\mathbf{x}$

$L := \mathrm{cholesky}(K + \sigma_n^2 I)$

$\boldsymbol{\alpha} := L^\top \backslash (L \backslash \mathbf{y})$ } predictive m

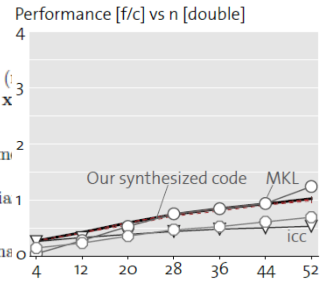
$\bar{f}_* := \mathbf{k}_*^\top \boldsymbol{\alpha}$

$\mathbf{v} := L \backslash \mathbf{k}_*$ } predictive varia

$\mathbb{V}[f_*] := k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{v}^\top \mathbf{v}$

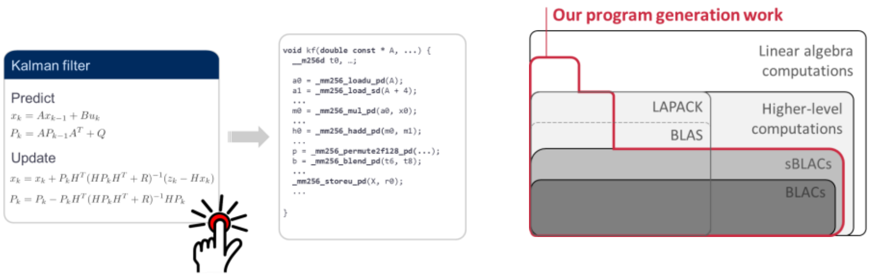
$\log p(\mathbf{y}|X) := -\frac{1}{2}\mathbf{y}^\top \boldsymbol{\alpha} - \sum_i \log L_{ii} - \frac{n}{2}\log 2\pi$

return: $\bar{f}_*$ (mean), $\mathbb{V}[f_*]$ (variance), $\log p(\mathbf{y}|X)$ (log margina



Conclusions

Program Generation for Small Linear Algebra



- Small linear algebra: An unsolved domain for performance
- Spiral-like approach to program generation
- Extensible (vector architectures, matrix structures)
- Good speedups
- Meanwhile there is more work on small linear algebra