Cl1ck + LGen: FLAME for small scale linear algebra

Diego Fabregat-Traver

with Daniele Spampinato, Markus Püschel and Paolo Bientinesi

HPAC, RWTH Aachen
fabregat@aices.rwth-aachen.de

BLIS Retreat, September 19th, 2016
University of Texas at Austin

Motivation

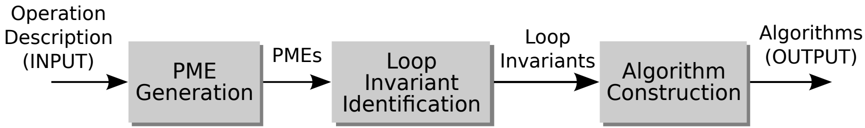
- Dense linear algebra software stack:
  - Optimized for *large enough* matrices
  - Rigid interface
- Applications may require:
  - High-performance at a smaller scale
  - More flexibility in terms of interface
  - Optimization across a sequence of operations

Goal

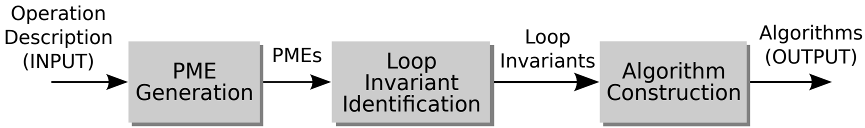
- Code generator for small scale linear algebra
- Combination of two code generators: CL1CK and LGen
- LGen: BLAS-like operations, code generation
- CL1CK: LAPACK-like, algorithmic aspects
- Scope: linear systems, factorizations, RECSY-like

Outline

Cl1ck


Operation
Description
(INPUT)
PME
Generation
PMEs
Loop
Invariant
Identification
Loop
Invariants
Algorithm
Construction
Algorithms
(OUTPUT)

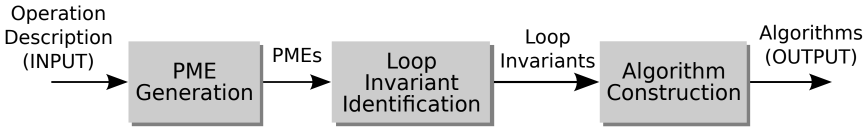
CL1CK



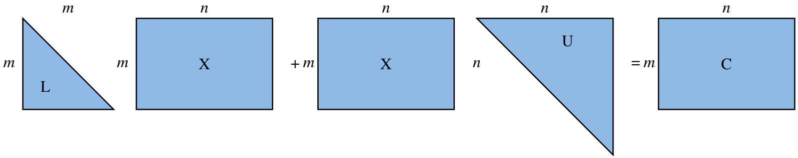
Example: Triangular Continuous-time Sylvester Equation

$$X := \Omega(L, U, C) \equiv \begin{cases} \textit{Equation} : LX + XU = C \\ \textit{Properties} : \texttt{Matrix}(L, U, C, X) \wedge \texttt{Input}(L, U, C) \wedge \texttt{Output}(X) \wedge \\ \qquad \texttt{LowerTriangular}(L) \wedge \texttt{UpperTriangular}(U) \end{cases}$$

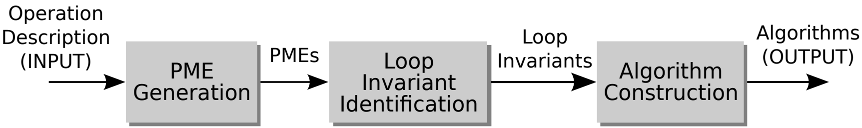
Cl1ck


Operation
Description
(INPUT)
PME
Generation
PMEs
Loop
Invariant
Identification
Loop
Invariants
Algorithm
Construction
Algorithms
(OUTPUT)

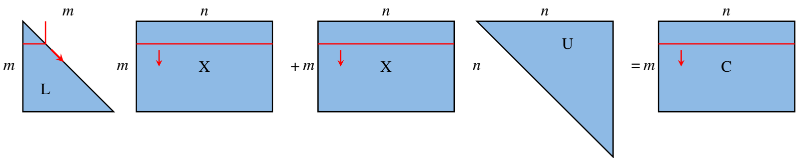
$$LX + XU = C$$


m
m
L
n
m
X
n
+ m
X
n
n
U
n
= m
C

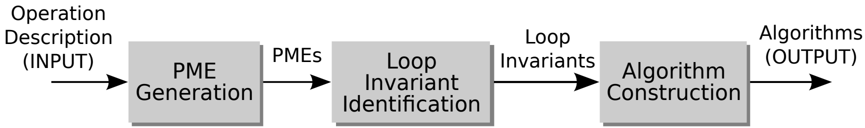
Cl1ck


Operation Description (INPUT)
PME Generation
PMEs
Loop Invariant Identification
Loop Invariants
Algorithm Construction
Algorithms (OUTPUT)

$$LX + XU = C$$


m
m
L
n
m
X
n
+ m
X
n
n
U
n
= m
C

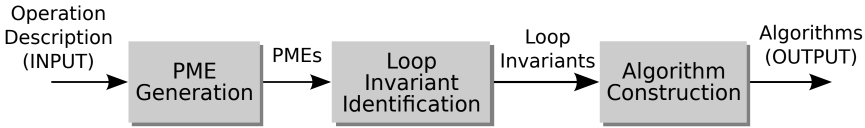
CL1CK



$$LX + XU = C$$

$$\left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right) \times \left(\frac{X_T}{X_B}\right) + \left(\frac{X_T}{X_B}\right) \times (U) = \left(\frac{C_T}{C_B}\right)$$

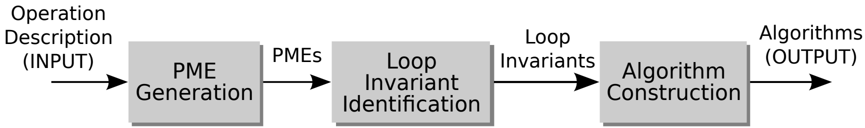
Cl1ck



$$LX + XU = C$$

$$\left(\begin{array}{c} L_{TL}X_T + X_T U = C_T \\ \hline L_{BL}X_T + L_{BR}X_B + X_B U = C_B \end{array}\right)$$

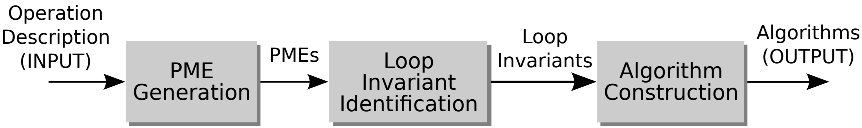
Cl1ck



$$LX + XU = C$$

PME: $$\left(\frac{X_T = \Omega(L_{TL}, U, C_T)}{X_B = \Omega(L_{BR}, U, C_B - L_{BL}X_T)} \right)$$

Cl1ck



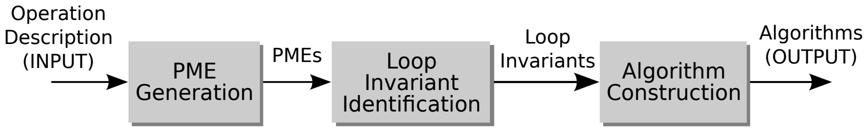
$$LX + XU = C$$

PME: $$\left(\frac{X_T = \Omega(L_{TL}, U, C_T)}{X_B = \Omega(L_{BR}, U, C_B - L_{BL}X_T)} \right)$$

$$\Downarrow$$

2 loop invariants $\longrightarrow$ 2 algorithms

Cl1ck



$$LX + XU = C$$

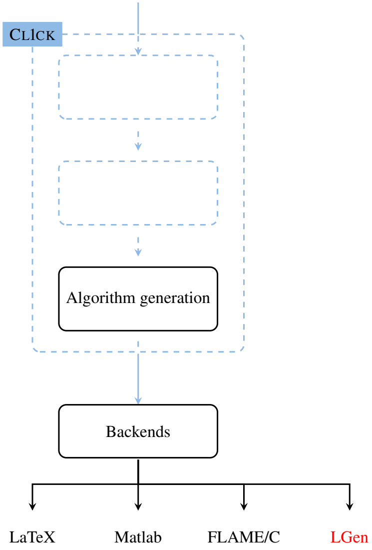
```
for( it = 0; it < m; it += mb )
{
```

$$\boxed{X_1} = \boxed{C_1} - \boxed{L_{10}} * \boxed{C_0}$$

$$\boxed{X_1} = sylv(\boxed{L_{11}}, \boxed{U}, \boxed{C_1})$$

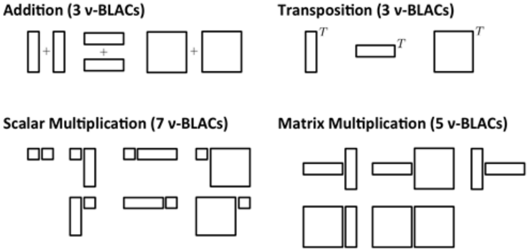
```
}
```

Cl1ck


CL1CK
Algorithm generation
Backends
LaTeX
Matlab
FLAME/C
LGen

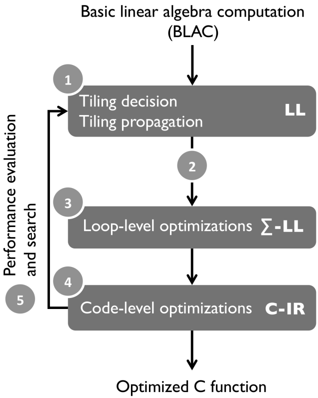
LGen

- Designed after Spiral
- Matrix expression to SIMD-vectorized C function
- Building blocs: ν-blacs



LGen

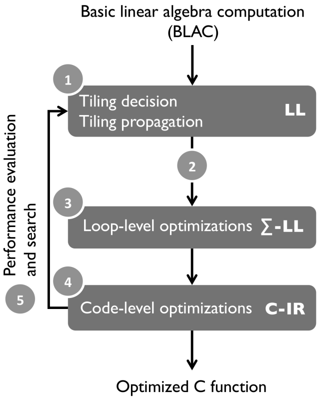
Tiling



Basic linear algebra computation
(BLAC)
1
Tiling decision
Tiling propagation
LL
2
3
Loop-level optimizations ∑-LL
4
Code-level optimizations C-IR
5
Performance evaluation
and search
Optimized C function

LGen

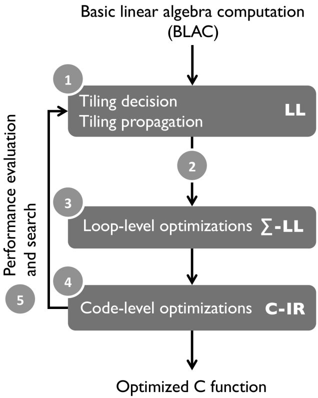
Tiling



- Tiling decision:
  $[C]_{2,2} = [A]_{2,2}[B]_{2,2} + [C]_{2,2}$
- Modeling + empirical search
- Hierarchical tiling
- For SIMD code, last level
  $\in (\nu, \nu), (\nu, 1), (1, \nu)$

LGen

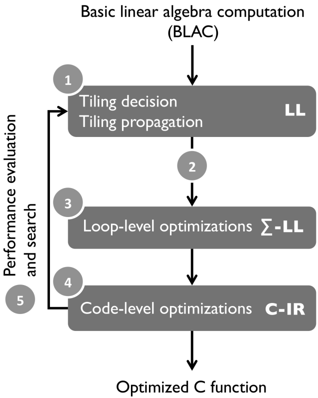
Loop-level optimizations



- $C = \Sigma_{i,j,k=0,2}[i,j](A[i,k]B[k,j] + C[i,j])$
- Algebraic manipulations:
  - Loop ordering
  - Loop fusion
  - Reduce reads/writes

LGen

C-IR optimizations

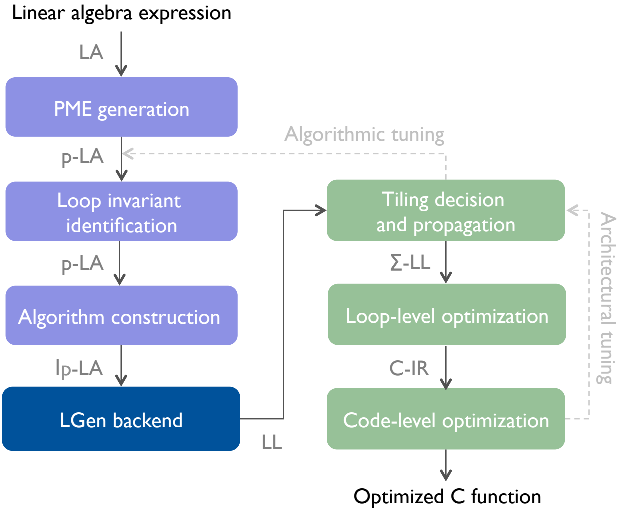


- Loop unrolling
- Scalar replacement
- Mapping onto ν-blacs

Outline

1. Cl1ck and LGen
2. Interfacing Cl1ck and LGen
3. Preliminary results
4. Conclusions

Cl1ck + LGen



Linear algebra expression
LA
PME generation
p-LA
Algorithmic tuning
Loop invariant identification
p-LA
Algorithm construction
lp-LA
LGen backend
LL
Tiling decision and propagation
Σ-LL
Loop-level optimization
C-IR
Code-level optimization
Architectural tuning
Optimized C function

CL1CK + LGen

Interface

- Example: $LX + XU = C$

- declaration.ll

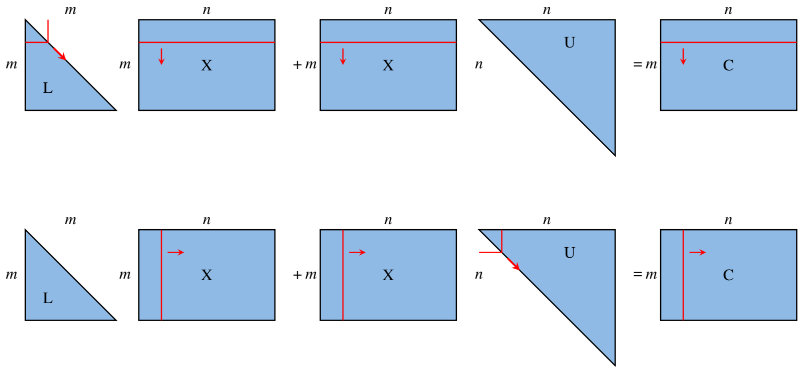
```
C: matrix<@m, @n, inout>;
L: triangular<@m, l, in>;
U: triangular<@n, u, in>;

C = sylv(@m, @n; L, U, C);
```

CL1CK + LGen

Interface

- One or more algorithms per dimension



Cl1ck + LGen

Interface

- var-m1.ll

```
For[ it; 0; m; mb ]
{
  % X_1 = C_1 - L_10 * X_0
  out0[h(mb,m,it), h(n,n,0)] = op2[h(mb,m,it), h(n,n,0)] - ...;
  % X_1 = sylv( L_11, U, X_1 )
  out0[h(mb,m,it), h(n,n,0)] = sylv(mb, n; ...);
};
```

Cl1ck + LGen

Interface

- var-n1.ll

```
For[ it; 0; n; nb ]
{
  % X_1 = C_1 - X_0 * U_01
  out0[h(m,m,0), h(nb,n,it)] = op2[h(m,m,0), h(nb,n,it)] - ...;
  % X_1 = sylv( L, U_11, X_1 )
  out0[h(m,m,0), h(nb,n,it)] = sylv(m, nb; ...);
};
```

Cl1ck + LGen

Interface

- Apply m1, then n1, ...

```
For[ it; 0; m; mb ]
{
  % matrix product
  For[ it2; 0; n; nb ]
  {
    % matrix product
    % sylv(...)
  };
};
```

Cl1ck + LGen

Interface

- Apply m1, then n1, ...

```
For[ it; 0; m; mb ]
{
  % matrix product
  For[ it2; 0; n; nb ]
  {
    % matrix product
    % sylv(...)
  };
};
```

- scalar-case.ll

```
out0 = op2/(op0 + op1);
```

Outline

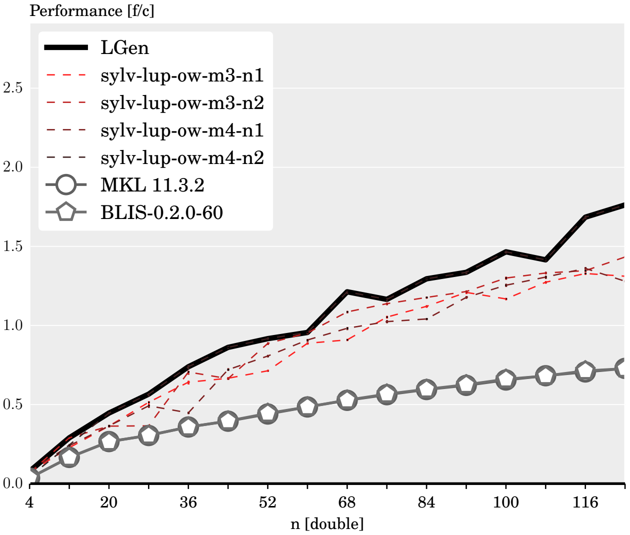
Preliminary results

Experimental setup

- Intel Sandy Bridge (AVX)
- 32KB L1-D cache, 256KB L2
- Single-core experiments
- Double precision
- Architecture peak performance: 8 f/c

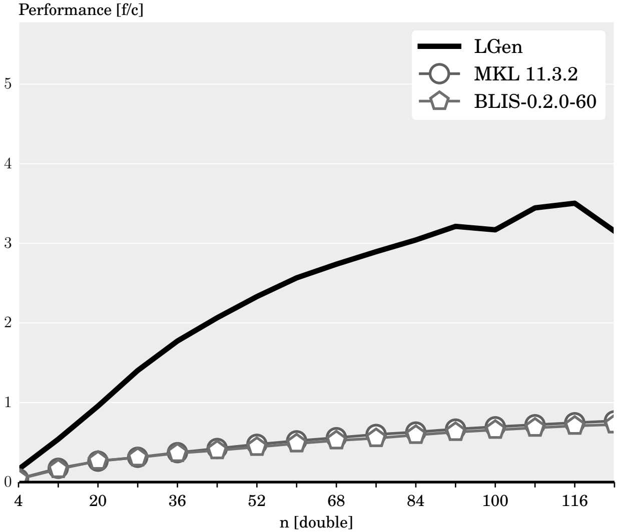
Preliminary results

Sylvester: $LX + XU = C$



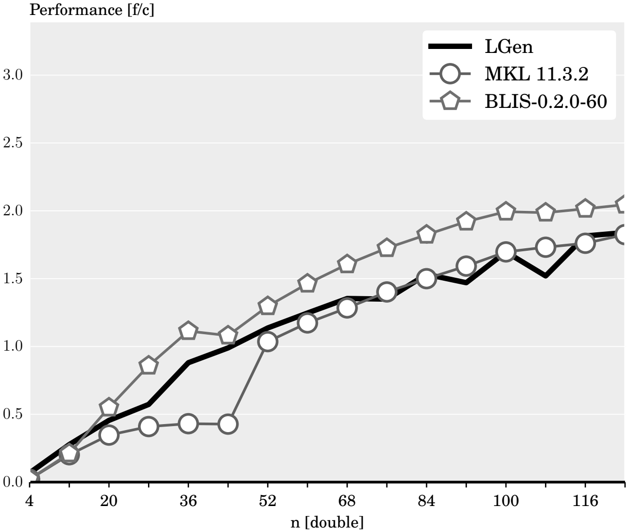
Preliminary results

Lyapunov: $LX + XL^T = C$



Preliminary results

Cholesky: $LL^T = A$



Conclusions

- Goal: High-performance for small problems
- Combination of CL1CK + LGen
- CL1CK: Algorithms
- LGen: Mapping onto building blocks and low level optimizations
- Interface: Algorithms partitioning in each dimention + base case
- Recursive application of partitionings

Conclusions

- Goal: High-performance for small problems
- Combination of CL1CK + LGen
- CL1CK: Algorithms
- LGen: Mapping onto building blocks and low level optimizations
- Interface: Algorithms partitioning in each dimention + base case
- Recursive application of partitionings

- Performance for small size, warm cache, competitive or outperforms BLIS/MKL.
- Better performance expected with further optimizations

Thank you for your attention!

CL1CK

- Knowledge-Based Automatic Generation of Partitioned Matrix Expressions.
  Diego Fabregat-Traver and Paolo Bientinesi. CASC 2011.
- Automatic Generation of Loop-Invariants for Matrix Operations.
  Diego Fabregat-Traver and Paolo Bientinesi. ICCSA 2011.

LGen

- A Basic Linear Algebra Compiler.
  Daniele G. Spampinato and Markus Püschel. CGO 2014.
- A Basic Linear Algebra Compiler for Structured Matrices.
  Daniele G. Spampinato and Markus Püschel. CGO 2016.
- http://www.spiral.net/software/lgen.html

DFG

Financial support from DFG through Grant BI 1533/2-1 is gratefully acknowledged