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

1. Click and LGen
2. Interfacing Click and LGen
3. Preliminary results
4. Conclusions
Example: Triangular Continuous-time Sylvester Equation

\[
X := \Omega(L, U, C) \equiv \begin{cases}
\text{Equation:} \\
LX + XU = C
\end{cases}
\]

\[
\begin{align*}
\text{Properties:} \\
\text{Matrix (} & L, U, C, X) & \land \text{Input (} & L, U, C) & \land \text{Output (} & X) & \land \\
\text{LowerTriangular (} & L) & \land \text{UpperTriangular (} & U) & 
\end{align*}
\]
Example: Triangular Continuous-time Sylvester Equation

\[
X := \Omega(L, U, C) \equiv \left\{ \begin{array}{l}
\text{Equation}: LX + XU = C \\
\text{Properties}: \text{Matrix}(L, U, C, X) \land \text{Input}(L, U, C) \land \text{Output}(X) \land \\
\quad \text{LowerTriangular}(L) \land \text{UpperTriangular}(U)
\end{array} \right.
\]
Operation Description (INPUT)

PME Generation

PMEs

Loop Invariant Identification

Loop Invariants

Algorithm Construction

Algorithms (OUTPUT)

$LX + XU = C$
Operation Description (INPUT) → PME Generation → PMEs → Loop Invariant Identification → Loop Invariants → Algorithm Construction → Algorithms (OUTPUT)

\[ LX + XU = C \]
<table>
<thead>
<tr>
<th>Operation Description (INPUT)</th>
<th>Description</th>
<th>PME Generation</th>
<th>Loop Invariant Identification</th>
<th>Algorithm Construction</th>
<th>Algorithms (OUTPUT)</th>
</tr>
</thead>
</table>

\[
LX + XU = C
\]

\[
\begin{pmatrix} L_{TL} & 0 \\ L_{BL} & L_{BR} \end{pmatrix} \times \left( \begin{pmatrix} X_T \\ X_B \end{pmatrix} \right) + \left( \begin{pmatrix} X_T \\ X_B \end{pmatrix} \right) \times (U) = \left( \begin{pmatrix} C_T \\ C_B \end{pmatrix} \right)
\]
\[ LX + XU = C \]

\[
\begin{align*}
L_{TL}X_T + X_TU &= C_T \\
L_{BL}X_T + L_{BR}X_B + X_BU &= C_B
\end{align*}
\]
\[ LX + XU = C \]

PME:

\[
\begin{align*}
X_T &= \Omega(L_{TL}, U, C_T) \\
X_B &= \Omega(L_{BR}, U, C_B - L_{BL}X_T)
\end{align*}
\]
\[ LX + XU = C \]

PME: 

\[
\begin{align*}
X_T &= \Omega(L_{TL}, U, C_T) \\
X_B &= \Omega(L_{BR}, U, C_B - L_{BL}X_T)
\end{align*}
\]

\[ \downarrow \]

2 loop invariants \(\longrightarrow\) 2 algorithms
$LX + XU = C$

```c
for( it = 0; it < m; it += mb )
{
    \[ X_1 = C_1 - L_{10} \times C_0 \]
    \[ X_1 = \text{sylv}(L_{11}, U, C_1) \]
}
```
LGen

- Designed after Spiral
- Matrix expression to SIMD-vectorized C function
- Building blocs: $\nu$-blacs
Tiling decision: \[ C_{2 \times 2} = [A_{2 \times 2}, B_{2 \times 2}] + [C_{2 \times 2}] \]

Modeling + empirical search

Hierarchical tiling

For SIMD code, last level \( \in (\nu, \nu), (\nu, 1), (1, \nu) \)

---

Basic linear algebra computation (BLAC)

1. Tiling decision
2. Tiling propagation
3. Loop-level optimizations \( \Sigma - LL \)
4. Code-level optimizations \( C - IR \)
5. Performance evaluation and search

Optimized C function
Tiling decision:
\[[C]_{2,2} = [A]_{2,2}[B]_{2,2} + [C]_{2,2}\]

- Modeling + empirical search
- Hierarchical tiling
- For SIMD code, last level \(\in (\nu, \nu), (\nu, 1), (1, \nu)\)
Loop-level optimizations

Basic linear algebra computation (BLAC)

1. Tiling decision
2. Tiling propagation
3. Loop-level optimizations
4. Code-level optimizations
5. Optimized C function

- $C = \sum_{i,j,k=0,2}[i,j](A[i,k]B[k,j] + C[i,j])$

- Algebraic manipulations:
  - Loop ordering
  - Loop fusion
  - Reduce reads/writes
C-IR optimizations

- Loop unrolling
- Scalar replacement
- Mapping onto \( \nu \)-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

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

Optimized C function

Algorithmic tuning

Architectural tuning

PME generation

Loop invariant identification

Algorithm construction

LGen backend

Optimized C function

Diego Fabregat | Cl1ck + LGen: FLAME for small scale linear algebra 18 / 30
Example: $LX + XU = C$

declaration.ll

```plaintext
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
`var-m1.ll`

```c
For[ it; 0; m; mb ]
{
    % X_1 = C_1 - L_10 \ast 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; ...);
};
```
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; ...);
};
**Apply m1, then n1, ...**

```plaintext
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

1. Cl1ck and LGen
2. Interfacing Cl1ck and LGen
3. Preliminary results
4. Conclusions
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: $LX + XU = C$

Performance [f/c]

- LGen
- sylv-lup-ow-m3-n1
- sylv-lup-ow-m3-n2
- sylv-lup-ow-m4-n1
- sylv-lup-ow-m4-n2
- MKL 11.3.2
- BLIS-0.2.0-60

Diego Fabregat | Cl1ck + LGen: FLAME for small scale linear algebra
Preliminary results

Lyapunov: \( LX + XL^T = C \)
Preliminary results

Cholesky: $LL^T = A$

Performance [f/c]

- LGen
- MKL 11.3.2
- BLIS-0.2.0-60

Diego Fabregat | Cl1ck + LGen: FLAME for small scale linear algebra
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 dimension + 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 dimension + 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.

  *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](http://www.spiral.net/software/lgen.html)

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