Automated High-Level Loop Fusion for FLAME Algorithms

Krzysztof A. Drewniak

Carnegie Mellon University

June TODO, 2018

High-level loop fusion

- Problems often are a series of subproblems
- Combining subalgorithms often helps performance
- Goal: find all the fused algorithms for a problem
- Compilers know too many details - need a high level approach

FLAME algorithms, loop invariants

- FLAME = Formal Linear Algebra Methods Eenvironments
- Provably correct algorithms from spec
- Algorithms $\Leftrightarrow$ loop invariants
- We know how to:
  - Autogenerate algorithm/code from loop invariant
  - Autogenerate all possible loop invariants
  - Identify when fusion is possible (in theory)

What we add

- Autogenerate all sets of fusable loop invariants
- Input is *partitioned matrix expression* — indicates needed computations
- Can be used to generate code

Section 2

FLAME

Goal

Want to compute

$$\widetilde{A} = \mathcal{F}(\hat{(}A), \underbrace{\ldots}_{O})$$

$\hat{A}$ and $\widetilde{A}$ share memory (A).
Initially, $A = \hat{A}$.
At termination, $A = \widetilde{A}$.

$$\widetilde{A} = CHOL(\hat{A})$$

Algorithm structure

partition $A \rightarrow \left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline A_{BL} & A_{BR} \end{array}\right)$

where $\dim(A_{TL}) = 0 \times 0$

do until $\dim(A_{TL}) = n \times n$

repartition $\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c||c|c} A_{00} & a_{01} & A_{02} \\ \hline\hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{02} & a_{21} & A_{22} \end{array}\right)$

$\vdots$] loop body

continue with $\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c||c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline\hline A_{20} & a_{21} & A_{22} \end{array}\right)$

enddo

Algoriithm example

partition $A \rightarrow \left(\begin{array}{c||c} A_{TL} & * \\ \hline\hline A_{BL} & A_{BR} \end{array}\right)$

where $\dim(A_{TL}) = 0 \times 0$

do until $\dim(A_{TL}) = n \times n$

repartition $\left(\begin{array}{c||c} A_{TL} & * \\ \hline\hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c||c|c} A_{00} & * & * \\ \hline\hline a_{10}^T & \alpha_{11} & * \\ \hline A_{02} & a_{21} & A_{22} \end{array}\right)$

$\alpha_{11} := \sqrt{\alpha_{11}}$

$a_{21} := a_{21} / \alpha_{11}$

$A_{22} := A_{22} - a_{21} a_{21}^T$

continue with $\left(\begin{array}{c||c} A_{TL} & * \\ \hline\hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c||c} A_{00} & * & * \\ \hline a_{10}^T & \alpha_{11} & * \\ \hline\hline A_{20} & a_{21} & A_{22} \end{array}\right)$

enddo

Partitioned Matrix Expressions

- Take A (and maybe other stuff), split it into regions.
- Lines between regions move during algorithm

$$\left(\begin{array}{c||c} \widetilde{A}_{TL} = \mathcal{F}_{TL}(\hat{A}, \ldots) & \widetilde{A}_{TR} = \mathcal{F}_{TR}(\hat{A}, \ldots) \\ \hline\hline \widetilde{A}_{BL} = \mathcal{F}_{BL}(\hat{A}, \ldots) & \widetilde{A}_{BR} = \mathcal{F}_{BR}(\hat{A}, \ldots) \end{array}\right)$$

$$\left(\begin{array}{c||c} \widetilde{A}_{TL} = CHOL(\hat{A}_{TL}) & * \\ \hline\hline \widetilde{A}_{BL} = \hat{A}_{BL}\widetilde{A}_{TL}^{-T} & \widetilde{A}_{BR} = CHOL(\hat{A}_{BR} - \widetilde{A}_{BL}\widetilde{A}_{BL}^{T}) \end{array}\right)$$

Loop invariants

- Find f_R and f'_R so $\mathcal{F}_R(\hat{A}) = f'(f(\hat{A}))$.
- f_R is loop invariant for R, f'_R is remainder
- Invariant for algorithm is an invariant per region
- Completely determine algorithm

This is a loop invariant

Starting from Cholesky's PME:

$$\left(\begin{array}{c||c} \widetilde{A}_{TL} = CHOL(\hat{A}_{TL}) & * \\ \hline\hline \widetilde{A}_{BL} = \hat{A}_{BL}\widetilde{A}_{TL}^{-T} & \widetilde{A}_{BR} = CHOL(\hat{A}_{BR} - \widetilde{A}_{BL}\widetilde{A}_{BL}^{T}) \end{array}\right)$$

We obtain

$$\left(\begin{array}{c||c} A_{TL} = CHOL(\hat{A}_{TL}) & * \\ \hline\hline A_{BL} = \hat{A}_{BL}\widetilde{A}_{TL}^{-T} & A_{BR} = \hat{A}_{BR} - \widetilde{A}_{BL}\widetilde{A}_{BL}^{T} \end{array}\right)$$

As are these

$$\left(\begin{array}{c||c} A_{TL} = CHOL(\hat{A}_{TL}) & * \\ \hline\hline A_{BL} = \hat{A}_{BL}\widetilde{A}_{TL}^{-T} & A_{BR} = \hat{A}_{BR} \end{array}\right)$$

$$\left(\begin{array}{c||c} A_{TL} = CHOL(\hat{A}_{TL}) & * \\ \hline\hline A_{BL} = \hat{A}_{BL} & A_{BR} = \hat{A}_{BR} \end{array}\right)$$

But not these

$$\left(\begin{array}{c||c} A_{TL} = CHOL(\hat{A}_{TL}) & * \\ \hline\hline A_{BL} = \hat{A}_{BL}\widetilde{A}_{TL}^{-T} & A_{BR} = CHOL(\hat{A}_{BR} - \widetilde{A}_{BL}\widetilde{A}_{BL}^{T}) \end{array}\right)$$

$$\left(\begin{array}{c||c} A_{TL} = \hat{A}_{TL} & * \\ \hline\hline A_{BL} = \hat{A}_{BL} & A_{BR} = \hat{A}_{BR} \end{array}\right)$$

Or this

$$\left(\begin{array}{c||c} A_{TL} = \hat{A}_{TL} & * \\ \hline\hline A_{BL} = \hat{A}_{BL}\widetilde{A}_{TL}^{-T} & A_{BR} = \hat{A}_{BR} \end{array}\right)$$

Tasks

- We need to specify split points

$$\left(\begin{array}{c||c} \widetilde{A}_{TL} :=_O CHOL(\hat{A}_{TL}) & * \\ \hline\hline \widetilde{A}_{BL} := \hat{A}_{BL}\widetilde{A}_{TL}^{-T} & \begin{array}{c} A_{BR,0} := \hat{A}_{BR} - \widetilde{A}_{BL}\widetilde{A}_{BL}^{T}; \\ \widetilde{A}_{BR} :=_O CHOL(A_{BR,0}) \end{array} \end{array}\right)$$

$$\left(\begin{array}{c||c} \widetilde{A}_{TL} :=_O \hat{A}_{TL}^{-1} & * \\ \hline\hline \begin{array}{c} A_{BL,(0,a)} := (\hat{A}_{BL} \vee A_{BL,(0,b)}) \cdot \widetilde{A}_{TL}; \\ A_{BL,(0,b)} := -\hat{A}_{BR}^{-1} \cdot (\hat{A}_{BL} \vee A_{BL,(0,a)}) \end{array} & \widetilde{A}_{BR} :=_O \hat{A}_{BR}^{-1} \end{array}\right)$$

More abstractly

The code translates tasks to

$$\left(\begin{array}{c||c} A_{TL,\top} :=_O \{A_{TL,\bot}\} & * \\ \hline\hline A_{BL,\top} := \{A_{BL,\bot}, A_{TL,\top}\} & \begin{array}{c} A_{BR,0} := \{A_{BR,\bot}\, A_{BL,\top}; \\ A_{BR,\top} :=_O \{A_{BR,0}\} \end{array} \end{array}\right)$$

$$\left(\begin{array}{c||c} A_{TL,\top} :=_O \{A_{TL,\bot} & * \\ \hline\hline \begin{array}{c} A_{BL,(0,a)} := \{A_{BL,\bot} \vee A_{BL,(0,b)}, A_{TL,\top}\}; \\ A_{BL,(0,b)} := \{A_{BR,\bot}, A_{BL,\bot} \vee A_{BL,(0,a)}\} \end{array} & A_{BR,\top} :=_O \{A_{BR,\bot}\} \end{array}\right)$$

Dependencies, v2

- $A_{R,\sigma}$ is before (can compute) $A_{R',\sigma'}$ if:
  - $R \neq R'$ (different regions) or
  - $$\begin{array}{ccccccccc} & & (0,x) & \rightarrow & (1,x) & \rightarrow & \cdots & & \\ & & \updownarrow & & \updownarrow & \rightarrow & \updownarrow & & \\ \perp & \rightarrow & 0 & \rightarrow & 1 & \rightarrow & \cdots & \rightarrow & \top \\ & & \updownarrow & & \updownarrow & \rightarrow & \updownarrow & & \\ & & (0,y) & \rightarrow & (1,y) & \rightarrow & \cdots & & \end{array}$$
- If anything from an or is before, all of it is

Dependency validity

- Invariant/remainder split has valid dependencies if:
  - All past task inputs before all future task outputs
  - All past task outputs not after all future task inputs

Finding all invariants

1. Pick a past/future split for each region
2. Check if the loop can make progress
3. Check for dependency validity

Section 3

Loop fusion

States of a region

Fully computed All tasks in the invarient

Uncomputed All tasks in the remainder

Partially computed Everything else

The fusion problem

$$\left.\begin{aligned}
\widetilde{A}^0 \quad & = \mathcal{F}^0(\hat{A}^0) \\
\widetilde{A}^1 \quad & = \mathcal{F}^1(\hat{A}^1) \\
& \vdots \\
\widetilde{A}^{n-1} \quad & = \mathcal{F}^{n-1}(\hat{A}^{n-1})
\end{aligned}\right\} \widetilde{A}^{n-1} = \mathcal{F}(\hat{A}^0)$$

where $\hat{A}^{i+1} = \widetilde{A}^i$

Fusion conditions

$$\hat{A}_{\mathbf{R}}^{i+1} = \widetilde{A}_{\mathbf{R}}^{i} \textbf{ if needed}$$

- $\mathcal{F}^i$'s invariant needs $R \Rightarrow \mathcal{F}_R^{j<i}$ fully computed
- $\mathcal{F}^i$'s remainder needs $R \Rightarrow \mathcal{F}_R^{j>i}$ uncomputed

An example: Cholesky + lower-triangular solve

Cholesky algorithms.

$$\left(\begin{array}{c||c} L_{TL} = CHOL(\hat{L}_{TL}) & * \\ \hline\hline L_{BL} = \hat{L}_{BL} & L_{BR} = \hat{L}_{BR} \end{array}\right)$$

$$\left(\begin{array}{c||c} L_{TL} = CHOL(\hat{L}_{TL}) & * \\ \hline\hline L_{BL} = \hat{L}_{BL}\widetilde{L}_{TL}^{-T} & L_{BR} = \hat{L}_{BR} \end{array}\right)$$

$$\left(\begin{array}{c||c} L_{TL} = CHOL(\hat{L}_{TL}) & * \\ \hline\hline L_{BL} = \hat{L}_{BL}\widetilde{L}_{TL}^{-T} & L_{BR} = \hat{L}_{BR} - \widetilde{L}_{BL}\widetilde{L}_{BL}^{T} \end{array}\right)$$

Lower triangular solve algorithm

$$\left(\begin{array}{c} B_T = L_{TL} \setminus \hat{B}_T \\ \hline\hline B_B = \hat{B}_B \end{array}\right)$$

$$\left(\begin{array}{c} B_T = L_{TL} \setminus \hat{B}_T \\ \hline\hline B_B = \hat{B}_B - L_{BL}\widetilde{B}_T \end{array}\right)$$

5 fused algorithms. (All combinations fuse except one.)

An example: Cholesky + lower-triangular solve

Cholesky algorithms.

$$\cancel{\left(\begin{array}{c||c} L_{TL} = CHOL(\hat{L}_{TL}) & * \\ \hline\hline L_{BL} = \hat{L}_{BL} & L_{BR} = \hat{L}_{BR} \end{array}\right)}$$

$$\left(\begin{array}{c||c} L_{TL} = CHOL(\hat{L}_{TL}) & * \\ \hline\hline L_{BL} = \hat{L}_{BL}\widetilde{L}_{TL}^{-T} & L_{BR} = \hat{L}_{BR} \end{array}\right)$$

$$\left(\begin{array}{c||c} L_{TL} = CHOL(\hat{L}_{TL}) & * \\ \hline\hline L_{BL} = \hat{L}_{BL}\widetilde{L}_{TL}^{-T} & L_{BR} = \hat{L}_{BR} - \widetilde{L}_{BL}\widetilde{L}_{BL}^{T} \end{array}\right)$$

Lower triangular solve algorithr

$$\left(\begin{array}{c} B_T = L_{TL} \setminus \hat{B}_T \\ \hline\hline B_B = \hat{B}_B \end{array}\right)$$

$$\cancel{\left(\begin{array}{c} B_T = L_{TL} \setminus \hat{B}_T \\ \hline\hline B_B = \hat{B}_B - L_{BL}\widetilde{B}_T \end{array}\right)}$$

5 fused algorithms. (All combinations fuse except one.)

We can't go further

Consider:

$$L := CHOL(L) \quad \left(\begin{array}{c||c} \widetilde{A}_{TL} :=_O CHOL(\hat{A}_{TL}) & * \\ \hline\hline \widetilde{A}_{BL} := \hat{A}_{BL}\widetilde{A}_{TL}^{-T} & \begin{array}{c} A_{BR,0} := \hat{A}_{BR} - \widetilde{A}_{BL}\widetilde{A}_{BL}^T; \\ \widetilde{A}_{BR} :=_O CHOL(A_{BR,0}) \end{array} \end{array}\right)$$

$$T := L^{-1}B \quad \left(\begin{array}{c} \widetilde{T}_T :=_O TRSV(\hat{L}_{TL}, B_T) \\ \hline\hline T_{B,0} := \hat{T}_B - \hat{L}_{BL}\widetilde{T}_T \\ \widetilde{T}_B :=_O TRSV(\hat{L}_{BR}, T_{B,0}) \end{array}\right)$$

$$X := L^{-T}B \quad \left(\begin{array}{c} X_{T,0} := \hat{X}_T - \hat{L}_{BL}^T\widetilde{X}_B \\ \widetilde{X}_T :=_O TRSV(\hat{L}_{BR}, X_{T,0}) \\ \hline\hline X_B := TRSV(\hat{L}_{BR}, \hat{T}_B) \end{array}\right)$$

- No fused algorithm (we checked)
- Top to bottom vs. bottom to top

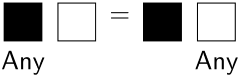
Strips

- Strip: sequence of region R from each loop
- Potentially fusable strip has:
  - Some number of fully computed regions, then
  - Optionally, one partially computed region, then
  - Uncomputed regions

but not

Finding fusable loops

- Search through potentially fusable strips



  -

- Enforce fusion constraints throughout
- Check all fusable strip-sets to see if each loop has an invariant

Last computed, first uncomputed

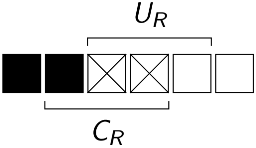
- Track constraints on last computed region C_R (and first uncomptued U_R)
- Initially, $-1 \leq C_R, U_R \leq n$ (maybe nothing/everything is computed/uncomputed)
- Past read in loop i: $C_R \geq i - 1$
- Future read in loop i: $U_R \leq i + 1$
- When strip is made, set C_R and U_R, add more constraints
- On failure, backtrack

Multiple matrices

- Some operations have multiple outputs
- (Ex. $y = Lx; L = L^{-1}$)
- All strips must be same length — add empty regions
- De-dup check from before works

Multiple matrices

- Last computed or first uncomputed can be followed by empty
- If so, bound $\{C, U\}_R$ to include the empty regions
- Needed to make constraints work



Comes from task

- For things like $LU = A$, tasks write multiple regions
- To prevent duplicates, use $U_R \leftarrow L_R$ (comes from)
- If L_R computed, U_R is computed, otherwise not

Another important example

- Graph problem $C = (AM + (AM)^T) - MM$, where A and M are symmetric
- $C := (AM + (AM)^T); C := C - MM$ has 56 fused algorithms
- However, $C = A$ or $C = M$ gives 0 algorithms
  - Dependencies: $TL \leftrightarrow TR$ and $TR \leftrightarrow BR$
  - Overwriting one quadrant requires computing everything
  - **TODO figure**

TODO do an experiment

Conclusions

- We can automatically find fusable loop invariants
- This is often helpful
- This analysis needs to be at this level

Acknowledgments

- Tze Meng for doing all the theory

Future work

- Probably not — maybe codegen