Automated High-Level Loop Fusion for FLAME Algorithms

Krzysztof A. Drewniak

Carnegie Mellon University

June 13, 2018

Loop fusion

```
while (...) {
   A
}
while (...) {
   B
}
```

$\longrightarrow$

```
while (...) {
    A;
    B
}
```

- Helpful for performance
- Often not possible

FLAME-like loops

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

Why high-level loop fusion?

Can we fuse this Cholesky algorithm

$$\begin{aligned} \lambda_{11} &:= \sqrt{\lambda_{11}} \\ l_{21} &:= l_{21}/\lambda_{11} \\ L_{22} &:= l_{21} l_{21}^T \end{aligned}$$

with this lower-triangular solve algorithm

$$\begin{aligned} b_{10} &:= (l_{10}^T B_{00})/\lambda_{11} \\ \beta_{11} &:= \beta_{11}/\lambda_{11}? \end{aligned}$$

- Hard to tell
- Compiler won't do it
- Need to look at higher level — loop invariants

Loop invariants

- Invariant says what the regions contain before & after each iteration
- In terms of $\hat{A}_R$ (initial value) & $\widetilde{A}_R$ (final value)
- For example:

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

and

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

- Fusion analysis much easier here
- Algorithm $\leftrightarrow$ loop invariant

What we add

- Known: how to find all possible loop invariants/algorithms for a problem
- Our work: finding all collections of *fusable* invariants

Section 2

Theory

Partitioned Matrix Expressions

- Show all computations needed in a region
- Take operation, split matrix into regions, solve for function
- Cross out parts to get loop invariants

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

$$\left(\begin{array}{c} \widetilde{B}_{T} = L_{TL} \setminus \hat{B}_{T} \\ \hline\hline \widetilde{B}_{B} = L_{BR} \setminus (\hat{B}_{B} - L_{BL}\widetilde{B}_{T}) \end{array}\right)$$

Forming loop invariants

- Cross out parts to get loop invariants
- Crossed-out parts go to *remainder*

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

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

Forming loop invariants

- Cross out parts to get loop invariants
- Crossed-out parts go to *remainder*

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

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

Forming loop invariants

- Cross out parts to get loop invariants
- Crossed-out parts go to *remainder*

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

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

Forming loop invariants

- Cross out parts to get loop invariants
- Crossed-out parts go to *remainder*

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

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

Forming loop invariants

- Cross out parts to get loop invariants
- Crossed-out parts go to *remainder*

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

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

States of regions

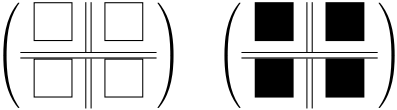
Fully computed Nothing crossed off/remainder is identity ■

Uncomputed Everything crossed off/invariant is identity □

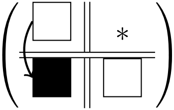
Partially computed Neither of the above

Not all splits work

- Can't remove everything/nothing
  - Can't remove every/no instance of underlying operation
- If you cross off $\hat{A}_R$, can't write to it
- If you don't cross off $\widetilde{A}_R$, must fully compute it



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



Fusion

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

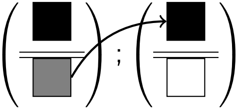
where $\hat{A}^{i+1} = \widetilde{A}^i$

Conditions for fusion

- Invariant reads $A_R^i \Rightarrow A_R^{i-1}$ fully computed
- Remainder reads $A_R^i \Rightarrow$ all later A_R^j uncomputed

Corollary:

but not



not good

Cholesky + lower-triangular solve

Cholesky invariants.

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

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

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

Six cases to check (3×2).

Lower triangular solve algorithms

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

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

Cholesky + solve: easy cases

Cholesky invariants.

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

or

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

Lower triangular solve algorithms

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

and

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

- Greediest algorithm needs L_{TL} and L_{BL}
- Both these Cholesky algorithms fully compute them

Cholesky + solve, remaining cases

Cholesky invariants.

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

Lower triangular solve algorithms

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

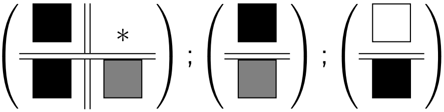
and

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

- Can't fuse with second solve algorithm (L_{BL} unavailable)
- So, five fusable algorithms

Cholesky + lower solve + upper solve

- Can't add $L^T \setminus B$
- We'd need L_{BR}^T, which is never fully computed
- Would also need to write on B_B
- Doesn't work even with temporary variables



Section 3

Implementation

Tasks

- Need to show software where partial computations can happen
- Pull suboperations that overwrite region into own names
- $:=_O$ is operation we want to do

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

$$\left(\begin{array}{c} \widetilde{B}_T :=_O L_{TL} \setminus \hat{B}_T \\ \hline\hline \begin{array}{c} B_{B,0} := \hat{B}_B - L_{BL}\widetilde{B}_T; \\ \widetilde{B}_B :=_O L_{BR} \setminus B_{B,0} \end{array} \end{array}\right)$$

Working in either order

- $\widetilde{A} = \hat{A} - B - C$ can be:
  - $A_0 := \hat{A} - B; \widetilde{A} := A_0 - C$ or
  - $A_0 := \hat{A} - C; \widetilde{A} := A_0 - B$
- Having all four creates duplication - complicates analysis
- Add new temporary type, $A_{R,(n,x)}$
- Now we can write

$$\{A_{(0,a)} := (\hat{A} \vee A_{(0,b)}) - B;$$
$$A_{(0,b)} := (\hat{A} \vee A_{(0,a)}) - C\}$$

Dependencies, v2

- Name $\hat{A}_R$ as $A_{R,\perp}$ and $\widetilde{A}$ as $A_{R,\top}$
- $A_{R,\sigma}$ is before (can compute) $A_{R',\sigma'}$ if:
  - $R \neq R'$ (different regions) or
  - σ can reach σ' on

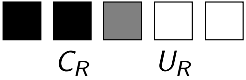
$$
\begin{array}{ccccccccc}
 & & (0,x) & \rightarrow & (1,x) & \rightarrow & \cdots & & \\
 & & \updownarrow & & \updownarrow & \rightarrow & \updownarrow & & \\
\perp & \rightarrow & 0 & \rightarrow & 1 & \rightarrow & \cdots & \rightarrow & \top \\
 & & \updownarrow & & \updownarrow & \rightarrow & \updownarrow & & \\
 & & (0,y) & \rightarrow & (1,y) & \rightarrow & \cdots & &
\end{array}
$$

- If anything from an or is before, all of it is

Finding all loop invariants

- For each invariant/remainder split:
  - Operation task ($:=_O$) in invariant and remainder (in different regions)
  - All inputs to invariant tasks must be before all remainder task outputs (no using data you don't have)
  - All invariant task outputs must not be after all remainder task inputs (no overwriting data you'll need)

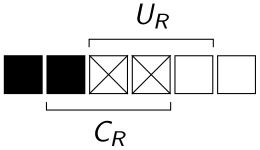
Fusion works differently



- While searching, add constraints to C_Rs and U_Rs
- A_R read in invariant $i \Rightarrow C_R \geq i - 1$
- A_R read in remainder $i \Rightarrow U_R \leq i + 1$
- Translates conditions from earlier
- If constraints fail, unwind search

Multiple matrices

- Need to add empty regions so all strips are same length
- Slight change to constraint system



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

Section 4

Demo

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, $A := (AM + (AM)^T); A := A - MM$ has no algorithms

Demo time

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