Reuse Analysis via Affine Factorization

Ryan Job
Advisor: Prof. Sanjay Rajopadhye
Motivation

• Consider the 3D expression here.
• Each plane reads the same value from an input.
  – The blue plane reads the same value of $A$
  – The red plane reads the same value of $B$.
• Their intersection produces the same results in $Y$.
• We have a 2D computation in 3D space.
• How do we automatically detect and exploit this?

\[ Y[i, j, k] = A[i + k] + B[i + j + k] \]
Outline

• Background
• Affine Factorization Algorithm
• Automating Reduction Simplification
Background
Affine Maps and Matrix Notation

• Affine maps apply a linear transformation and a translation to a domain.
  – $y = Ax + b$

• We use an augmented matrix notation:
  – Augment the input ($x$) with a constant 1.
  – Merge the transformation ($A$) with the translation ($b$).
  – $y = [A|b] \cdot \begin{bmatrix} x \\ 1 \end{bmatrix}$
Hermite Normal Form (HNF)

- HNF is analogous to reduced row echelon form (RREF), but for integer spaces.
- Given a set of vectors, HNF finds a basis which spans them.
- We use HNF to decompose a matrix into two:
  - \( M \): input vectors, written as rows of a matrix.
  - \( H \): the basis of the input vectors.
  - \( U^{-1} \): a transformation from the basis to the input.
  - \( M = U^{-1} \cdot H \)
Affine Factorization Algorithm
Algorithm Overview

- We use HNF to “factorize” a set of affine maps with a common domain.
- Find the smallest subspace of distinct values for the computation.
  - We call this the “intermediate space”.
- Rewrite the original maps as the composition of two:
  - $H$ maps the domain to the intermediate space.
  - Then, subsets of $U^{-1}$ map to the desired ranges.
- This use case is mathematically simple, but we could not find it in use.
  - Neither in the polyhedral community nor the wider compilation community.
Algorithm Details

- Write the affine maps as augmented matrices.
- Concatenate the matrices on top of each other: $M$.
- HNF is used to rewrite the maps with $H$ and $U^{-1}$.
  - Each map uses $H$ as-is, and a subset of $U^{-1}$.
- Since $H$ is common to all rewrites, it can be factored out.
  - Introduce a new variable of only the unique values ($U^{-1}$).
  - Map the full output to these values ($H$).

---

**Algorithm 1** Algorithm for factorizing affine maps

**Input:** A list matrices $M_i$ representing affine maps, all with the same $D$-dimensional domain.

**Output:** A common right factor $H$ and left factors $Q_i$.

1: procedure FACTORIZEMAPS($M_0 \ldots M_n$)
2:   $M \leftarrow$ CONCATENATE($M_0 \ldots M_n$)
3:   $H, U \leftarrow$ HERMITE_NORMAL_FORM($M$)
4:   $Q \leftarrow$ MATRIX_INVERSE($U$)
5: for $r = \text{ROWS}(H) -1 \ldots 0$ do
6:   if IS_ROW_OF_ZEROS($H, r$) then
7:     $H \leftarrow$ DROP_ROW($H, r$)
8:     $Q \leftarrow$ DROP_COL($Q, r$)
9:   end if
10: end for
11: start $\leftarrow$ 0
12: for $i = 0 \ldots n$ do
13:   end $\leftarrow$ start + ROWS($M_i$)
14:   $Q_i \leftarrow$ GET_ROWS($Q, \text{start}, \text{end}$)
15:   start $\leftarrow$ end
16: end for
17: return $H, Q_0 \ldots Q_n$
18: end procedure
Automating Reduction
Simplification
Alpha & AlphaZ

- Alpha is a declarative, equational language for the polyhedral model.
- Reductions are modeled as a collection of inputs combined with an operator.
- AlphaZ is a system for optimizing Alpha equations and generating C code.
- We are focusing on the “Simplifying Reductions” optimization.
  - Exploits reused values to lower the asymptotic time complexity of the computation.
- Currently, it requires human input to indicate how values are reused.
  - Given this information, the reduction can be automatically rewritten.
Automatic Reduction Simplification

- We apply affine factorization to the affine maps which index input variables.
- If the space of unique values is lower dimension than the result:
  - Values are reused throughout the computation.
  - The basis, $H$, will have a non-trivial null space.
- Vectors in this null space indicate how values are reused.
- Any such vector is enough information to automate Simplifying Reductions.
Current Status

• Developed a proof-of-concept for affine factorization.
  – Publicly available on GitHub (link in the paper).
  – Presented as a Jupyter notebook using the islpy library.

• Incorporating the algorithm into AlphaZ.
  – Goal: automate the Simplifying Reductions optimization.
Additional Uses

• Found a use case for memory layout transformations in FPGA accelerators.
  – Relates to work by Corentin Ferry, being presented later today.
• Investigating applications to algorithm-based fault tolerance.
  – Relates to work by Louis Narmour, presented at IMPACT last year.
• We hope to hear from you about more use cases!
Thank you!