AlphaZ & the Polyhedral Equational Model

Tomofumi Yuki, Sanjay Rajopadhye
Polyhedral Compilation

- The Polyhedral Model
  - Established approach for automatic parallelization
  - Based on mathematical formalism
- Many tools and compilers:
  - PIPS, PLuTo, MMAlpha, Par4All, RStream, XLF/XLC GRAPHITE (gcc), Polly (LLVM), …
  - and AlphaZ
Design Space (a subset)

- Space-Time + Tiling: schedule + parallel loops
  - Primary focus of existing tools

- Memory Allocation
  - Most tools do not modify the original allocation
  - Complex interaction with space time

- Higher-level Optimizations
  - Detection/parallelization of reductions & scans
  - Simplifying Reductions (complexity reduction)
  - Equational Programming
AlphaZ

Tool for Exploration
- Provides a collection of analyses, transformations, and code generators

Unique Features
- Memory Allocation
- Reductions

Can be used as a push-button system (e.g., Parallelization à la PLuTo is possible) but not our current focus
- [caveat: a push button MPI code generator is now available]
Two Examples

- adi.c from PolyBench
  - Re-considering memory allocation allows the program to be fully tiled
  - Outperforms PLuTo that only tiles inner loops

- LU Decomposition (illustration)
  - Deriving an equational program from first principles
Focus on Memory

- Tiling requires more memory
- e.g., Smith-Waterman dependence
ADI-like Computation

- Updates 2D grid with outer time loop
- PLuTo only tiles inner two dimensions
  - Due to a memory based dependence
  - With an extra scalar, all three dimensions can be tiled
- PolyBench implementation has a bug
  - It does not correctly implement ADI
  - All dimensions of a correct ADI program cannot be tiled
adi.c: Original Allocation

```c
for (t=0; t < tsteps; t++) {
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            X[i][j] = foo(X[i][j], X[i][j-1], ...)
...
    for (i = 0; i < n; i++)
        for (j = n-1; j >= 1; j--)
            X[i][j] = bar(X[i][j], X[i][j-1], ...)
...}
```

- Not tilable because of the reverse loop
- Memory based dependence: \((i,j \rightarrow i,j+1)\)
- Requires all dependences to be non-negative
for (j = 0; j < n; j++)
S1: \( X[i][j] = \text{foo}(X[i][j], X[i][j-1], \ldots) \)
...

for (j = n-1; j >= 1; j--)
S2: \( X[i][j] = \text{bar}(X[i][j], X[i][j-1], \ldots) \)
...
adi.c: With Extra Memory

Once the two loops are fused:

- Value of $X$ only needs to be preserved for one iteration of $j$

  ```c
  for (j = 0; j < n; j++)
  S1:     X[i][j] = foo(X[i][j], X[i][j-1], …)
  ...
  
  for (j = 1; j < n; j++)
  S2:     X'[i][j] = bar(X[i][j], X[i][j-1], …)
  ...
  ```

We don’t need a full array $X'$, just a scalar
PLuTo does not scale because the outer loop is not tiled
Moral

- War is too serious a matter to entrust to military men.
  - Georges Clemenceau, early 20th century French PM

- Memory is too serious to entrust to programmers
Equational Programming: $E=mc^2$
LU Decomposition (derivation)

\[
A_{i,j} = \begin{cases} 
  \sum_{k=1}^{\min(i,j)} L_{i,k} U_{k,j} & i \leq j \\
  \sum_{k=1}^{\min(i,j)} L_{i,k} U_{k,j} & i > j
\end{cases}
\]

\[
A_{i,j} = \sum_{k=1}^{i-1} U_{i,j,k} U_{k,j} + \sum_{k=1}^{j-1} L_{i,k} U_{k,j}
\]
LU Decomposition (derivation)

\[ U_{i,j} = A_{i,j} - \sum_{k=1}^{i-1} L_{i,k} U_{k,j} \]

\[ L_{i,j} = \left( A_{i,j} - \sum_{k=1}^{j-1} L_{i,k} U_{k,j} \right) / U_{j,j} \]
affine LUD \( \{N \mid 1 < N\} \)
input
float A \( \{i, j \mid 0 < (i, j) \leq N\} \)
output
float L \( \{i, j \mid 0 < j < i \leq N\} \)
float U \( \{i, j \mid 0 < i \leq j \leq N\} \)
let
L[i, j] = A[i, j] - \text{reduce}(+, [k] L[i, k] \times U[k, j])
U[i, j] = (A[i, j] - \text{reduce}(+, [k] L[i, k] \times U[k, j])) / U[j, j]
AlphaZ System Overview

Target Mapping:
- Specifies schedule, memory allocation, etc.

Polyhedral Representation

Target Mapping

Transformations

Analyses

Code Gen

C

Alpha

C+OpenMP

C+CUDA

C+MPI
Human-in-the-Loop

- Automatic parallelization—“holy grail” goal
  - Current automatic tools are restrictive
    - A strategy that works well is “hard-coded”
    - difficult to pass domain specific knowledge

- Human-in-the-Loop
  - Provide full control to the user
    - Help finding new “good” strategies
    - Guide the transformation with domain specific knowledge
Conclusions

- There are more strategies worth exploring
  - some may currently be difficult to automate

- Two examples
  - adi.c: memory
  - Deriving LU decompostition (first principles)

- AlphaZ: Tool for trying out new ideas: see
  - https://www.cs.colostate.edu/AlphaZ/wiki
  - http://www.cs.colostate.edu/TechReports
    - 12-101 [AlphaZ details] & others
Acknowledgements

- AlphaZ Developers/Users
  - Members of MÉLANGE at CSU
  - Members of CAIRN at IRISA, Rennes
  - Dave Wonnacott and students, Haverford University
Key: Simplifying Reductions

- Simplifying Reductions [POPL 2006]
  - Finds “hidden scans” in reductions
  - Rare case: compiler can reduce complexity
- Main idea:
  \[
  X[i] = \sum_{k=0}^{i} A[i]
  \]
  \[O(n^2)\]
  becomes
  \[
  X[i] = \begin{cases} 
  i = 0 : A[i] \\
  i > 0 : X[i - 1] + A[i]
  \end{cases}
  \]
  \[O(n)\]