## AlphaZ Wrapup, Midterm Review, Pluto Start

**Announcements**

- Grades for HW5 should be sent out Saturday (if your HW5 was on time)
- Project proposal rewrite is due Sunday night

**Today**

- Alphabets examples
  - Forward substitution
- Some Alphabets syntax
- Scheduling and storage mapping in AlphaZ
- Parallelization of generated AlphaZ code
- Midterm review
- Getting started with Pluto



## Forward Substitution (Dense Matrix)

**Given an NxN lower triangular matrix with unit diagonals and a n-vector b solve for the vector x in** $Lx = b$

**How do we solve for x?**

$$x_i = b_i - \sum_{j=1}^{i-1} L_{i,j} x_j$$

## ForwardSolve.ab

```
affine ForwardSolve {N | N>1  }
  given float b {i | 1<=i<=N }; float L {i,j| 1<=(i,j)<=N };
  returns float x {i | 1<=i<=N };
through
  x[i] = case
    {|i==1} : b[i];
    {|i>1}  : b[i] - reduce(+,(i,j->i),{|1<=j<i} : L[i,j]*x[j]);
    esac;
.
```



## ForwardSolve.cs

```
prog = ReadAlphabets("ForwardSolve.ab");
system = "ForwardSolve";
outDir = "./forwardsolve-out/"+system;
CheckProgram(prog);
Show(prog);
AShow(prog);
PrintAST(prog);

NormalizeReduction(prog);
Show(prog);
setSpaceTimeMap(prog, system, "x,NR_x", "(i->i)");
setDimensionType(prog, system, "x,NR_x", 0, "S");
setStatementOrdering(prog, system, "NR_x", "x");

VerifyTargetMapping(prog, system, "MIN");

generateScheduledCode(prog, system, outDir);
generateWrapper(prog, system, outDir);
generateMakefile(prog, system, outDir);
```

## Some Alphabets Syntax

**Overall structure**

```
affine systemname <input parameter set constraints>
  given <input var domain list>;
  returns <output var domain>;
  using <temp var domain list>;
  through
    <system of affine recurrence equations>
.
```

**Input parameter set constraints**

```
{P, Q, R|P>1 && Q>1 && R>1}
```

**Variable domain examples**

```
float A {i,k | 0<=i<P && 0<=k<Q};
float C {i,j,k| 0<=i<P && 0<=j<R && k==Q+1};
```

**System of equations**

```
A[i,k] = C[i,k];
```



## Some Alphabets Syntax, cont…

**Restrict Expression**

```
{|1<=j<i} : L[i,j]*x[j]
{|i==1} : b[i]
{|t>0} : 1/3*(A[t,i-1] * A[t-1,i] * A[t-1,i+1])
```

**Case Expression**

```
case {|t==0} : X[i];
     {|t>0} : 1/3*(A[t,i-1] * A[t-1,i] * A[t-1,i+1]);
esac
```

**Dependence function and Expressions**

```
1/3*(A[t,i-1] * A[t-1,i] * A[t-1,i+1])
(((t,i->)@1 / (t,i->)@3) * (((t,i->t,i-1)@A * (t,i-
  >t-1,i)@A) * (t,i->t-1,i+1)@A))
```

**Reduction**

```
reduce(+,(i,j->i),{|1<=j<i} : L[i,j]*x[j])
```

## Key Concepts AlphaZ scripting

**Space-time mapping concept**

- Space-time mapping is same as the schedule mapping.
- The space-time adjective refers to some of the elements in the map indicating which processor (space) and some indicating time.
- In AlphaZ scripting there is also the concept of an element indicating statement order.

**Storage Mapping**

- Recurrence equations are analogous to array assignments where all of the arrays are fully expanded.
- This leads to single assignment, or each location is only written once.
- Efficient implementations share memory locations between writes.
- It is important to specify the storage mapping in AlphaZ.



## 1D Stencil Computation (AlphaZ example)

**1D Stencil Computation version 1**

```
// A[0,i] initialized to some values
for (i=0; i<N; i++) {
  A[0,i] = X[i];
}
for (t=1; t<=T; t++) {
  for (i=1; i<(N-1); i++) {
    A[t,i] = 1/3 * (A[t-1,i-1] + A[t-1,i] + A[t-1,i+1];
  }
}
```

**Translating to AlphaZ**

- Input parameters?
- Input array?
- Domain for A?
- One equation for all of A?

## Stencilv1.ab

```
affine stencilv1 {N,T | N>=3 && T>=1 }
  given float X {i | 0<=i<N };
  returns float R {t,i | t==T && 0<=i<N };
  using float A {t,i | 0<=t<=T && 0<=i<N };
through
  A[t,i] = case
    {|t==0} : X[i];
    {|t>0 && i==0} : X[i];
    {|t>0 && i==N-1} : X[i];
    {|t>0 && 0<i<N-1} : 1/3*(A[t-1,i-1] + A[t-1,i] + A
[t-1,i+1]);
    esac;
  R = A;
.
```



## Stencilv1.cs

```
prog = ReadAlphabets("Stencilv1.ab");
system = "stencilv1";
outDir = "./stencilv1-out/"+system;
CheckProgram(prog);
Show(prog);
AShow(prog);
PrintAST(prog);

setSpaceTimeMap(prog, system, "A", "(v,w->v,w,0)");
setSpaceTimeMap(prog, system, "R", "(v,w->v,w,1)");
setDimensionType(prog, system, "A,R", 0, "S");
setDimensionType(prog, system, "A,R", 1, "S");
setDimensionType(prog, system, "A,R", 2, "O");
#setMemoryMap(prog, system, "A", "A", "(v,w->v,w)", "2,0");
VerifyTargetMapping(prog, system, "MIN");

generateScheduledCode(prog, system, outDir);
generateWrapper(prog, system, outDir);
generateMakefile(prog, system, outDir);
```

## Tricky Bits

**Use the verification tools**

- CheckProgram(prog) should be done right after reading the Alphabets program.
- VerifyTargetMapping(prog, system, "MIN") should be done after all scheduling and storage mapping

**Dimensionality matching**

- Stencilv1.ab, made output R 2D with extent 1 in t dimension to match with 2D A variable.
- Stencilv2.ab, made output R 1D, but then have to access one of the A dimensions with a constant.

**Statement ordering**

- There is no default statement ordering.
- Need to specify it with the space-time mapping or with the setStatementOrdering() compiler script function.

**Reduction normalization**

- Tool needs reduction by itself on the right hand side.
- Can do this with NormalizeReduction(prog) but then use Show(prog) to determine any new variables introduced. All the variables will need a schedule.



## Onto Parallelism!

**Parallelizing code using AlphaZ**

- The exact flow dependences can be read from the system of recurrence equations.
  - <Create a dependence relation for an example>
  - <Create a dependence vector from the dependence relation>
- Apply the schedule mapping to the dependence vectors to determine the new dependence vectors.
- Any dimension in the new dependence vectors that do not carry a dependence can be made parallel.
- Currently have to insert the OpenMP pragma in the generated code, but the schedule verifier should be able to verify if parallelism is possible.

## Midterm Review I

**Understand and be able to compute**

- Speedup
- Efficiency
- Isoefficiency
- Big-O for a computation
- Matrix-matrix multiplication and matrix vector multiplication

**Analysis of application performance and transformation opportunities**

- Locate data reuse in a computation (think about how you might automate this using the omega calculator)
- Create a roofline model for a machine
- Compute the operational/arithmetic intensity for a computation
- Explain how one can use the roofline model to guide performance improvement of a computation
- Illustrate the parallelization versus storage tradeoff



## Midterm Review II

**Automating Program Transformation and Parallelization**

- Translate the computation/code to a model
  - Create an iteration space graphically and by specifying an omega calculator set
  - Compute all of the data dependences
    - anti, output, flow and memory or exact
    - Dependence relations, dependence vectors (distance and/or direction)
- Select a transformation/schedule
  - Specify a transformation as a unimodular matrix OR as an affine function
  - Show how to determine if a transformation is legal or not
- Transform the model and generate the resulting code
  - Know how to do this using the omega calculator and AlphaZ

## Midterm Review III

**Parallelization**

- Be able to parallelize a loop with OpenMP, you might need to do this for the midterm.
- When does superlinear speedup occur? Why does it indicate you are trying to trick the masses?
- How can we use dependence vectors to determine which loops can be parallelized? (loop carried dependence)

**Important Concepts and Terms**

- Loop transformations: permutation, skewing, reversal, fission, and fusion
- Why is dependence analysis NP-complete?
- Bernstein conditions
- Affine function
- Polyhedron
- Data locality
- System of recurrence equations



## Getting Started with Pluto

**Pluto has been installed on the CS linux machines**

**Try out some examples**

- git clone git://repo.or.cz/pluto.git will create a directory called pluto
- Test and examples directory contains example input files
- polycc test/seidel.c --tile –parallel

**Read some documentation**

- more pluto/doc/DOC.txt
- http://pluto-compiler.sourceforge.net/

## Next Time

**Reading**

- Review reading assignments up to this point
- Read the Pluto web page and DOC.txt

**Homework**

- Project proposal rewrite due Sunday 3/4/12 at midnight
- Study for the midterm
  - Practice doing problems
  - Ensure understanding of concepts covered in class, HWs, and on the quiz. Ask questions!!

**Lecture**

- More on Pluto
- Why is the polyhedral model so important?