The AlphaZ Automation Tool

Announcements

- HW5 is due Wednesday February 29th, tomorrow
- Project proposal rewrite is due Sunday night

Today

- Quiz 1 observations
- Project proposal suggestions
- HW4, setting up the data dependence problem
- Alphabets examples
  - Stencil version 1
  - Stencil version 2
  - Forward substitution
- Some Alphabets syntax
- Scheduling and storage mapping in AlphaZ
- Parallelization of generated AlphaZ code

Dwarf Interests

Brendan

- Sparse linear algebra
- "Problem hindering a lot of particle physics research"

Glenn

- Monte carlo
- Experience using Monte Carlo for financial calculations

Greg

- Dense linear algebra
- Research in "1-D and 3-D radiative transfer simulations for satellite remote sensing and modeling of planetary atmospheres"

Jared

- Spectral methods
- Experience with audio/vidio processing systems

Wenxiang

- Backtrack and branch and bound, because doing research in AI

Dwarf Interests cont…

Lixing

- Dense linear algebra
- Graphics in games

Matt

- N-body methods
- Galaxies and particles, wide range of applicability

Nirmal

- Spectral methods

Robert

- N-body problems
- Curious about O(N lgN) algorithms

Ryan

- Sparse linear algebra
- “Have the ability to encode themselves and all the other dwarfs”

Components in a grant/project proposal

Problem

- For this project you need to identify a performance problem within the code of interest.

Context (or why we should care)

- Who uses the benchmark or app?
- What impact will improving its performance have?

Approach or proposed work statement

- Grant Ex: "We propose to research and develop high-level implementation abstractions that leverage existing and new mechanisms for interfacing between algorithm and implementation specifications.”
- Project Ex: "Using a well-known benchmark, we plan to do a theoretical performance constraint analysis, parallelize and optimize it with the the aid of the chosen automation tool, and ultimately compare the experimental performance results to both the theoretical limits as well as hand-optimized versions."

Components in a grant/project proposal, cont …

Preliminary Results

- Preliminary performance analysis results on the benchmark/application
- Small example with the automation tool that is relevant to the performance problem
- Optional: Ideas for possible transformations would help make your schedule more concrete.

Research Plan and Project Timetable / Detailed Schedule

- Vague examples
  - "Summarize the experiments and write intermediate report."
  - "Hand parallelization complete"
  - "Deliver intermediate report"
  - "Develop a loop transformation and parallelization strategy for improving the performance of ..."

Research Plan/Detailed Schedule, cont …

- More concrete examples
  - "Experiment with parallelizing the computation loops. Generate equivalent parallelizations for Java and C. Compare results of each, with each kernel."
  - "Coding and testing for the manual OpenMP version."
  - "Parallelize the 2-class classifier in OpenMP."
- Concrete example
  - "Translate Python code to C code via Cython, check whether the correctness of translation, and complete c-WalLS.

Suggestions

- Describe specific graphs you plan to generate.
- If your application is memory bound, then indicate some possible data reuse in the application for which you will investigate ways to turn into data locality.
- What specific loops in your application are amenable to parallelization?
- How will you evaluate the output of the automatically generated and/or scheduled code?
- What other evaluation metrics will you use to evaluate the automation tool?

HW4

Let's go look at the key

Key concepts

- Setting up the data dependence problem
- Dependence vectors should be lexicographically non-negative

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);
```

1D Stencil Computation (version 2, AlphaZ example)

1D Stencil Computation, version 2

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

Translating to AlphaZ

- We need to expand the array, but still match the exact flow dependences.
- The output domain needs to match the output of this original loop.

Stencilv2.ab

```
affine stencilv2 {N,T | N>=3 && T>=1 }
  given float X {i | 0<=i<N };
  returns float R {i | 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} : 1/3*(A[t,i-1] * A[t-1,i] * A[t-1,i+1]);
    esac;
  R[i] = A[T,i];
.
```

Stencilv2.cs

```
prog = ReadAlphabets("Stencilv2.ab");
system = "stencilv2";
outDir = "./stencilv2-out/"+system;

CheckProgram(prog);

Show(prog);
AShow(prog);
PrintAST(prog);

setSpaceTimeMap(prog, system, "A", "(v,w->v,w)");
setSpaceTimeMap(prog, system, "R", "(w->T,w)");
setStatementOrdering(prog, system, "A", "R");
VerifyTargetMapping(prog, system, "MIN");

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


Forward Substitution (Dense Matrix)

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

$$b_i = \sum_{j=1}^{N} L_{i,j} xj$$

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>0}  : 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])
```

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.

Next Time

Reading

- AlphaZ wiki, AlphaZ LUD tutorial, and Alphabets grammar

Homework

- HW5 is due Wednesday 2/29/12
- Project proposal rewrite due Sunday 3/4/12 at midnight

Lecture

- Midterm review
- Using the PLUTO tool