Transformation Verifier

Tomofumi Yuki, Sanjay Rajopadhye

April 25, 2009

Equations to Parallel Programs

Certain subsets of programs can be represented as system of equations (Polyhedral Model)

We are working on a system (AlphaZ) that exploits benefits of this model

Missing pieces for efficient parallel execution of equations:

- Schedule
- Processor Allocation
- Memory Mapping

Tool to verify the above is useful for both manual and automated exploration of efficient programs

Overview of the Entire Flow

1. Specify TPMSpec
   - Currently manual
   - Limited scheduler is available
2. Verify TPMSpec
3. Apply a set of transformations to reflect the TPMSpec
   - Code generator generates loops
   - We want each axis to be aligned with T or P
4. Generate code

TPMSpec : Time/Processor/Memory Specification
This is only a subset of AlphaZ

Background

Alphabets: Language to specify computations as equations

- Affine Dependencies
- Only the computation itself is specified

Affine Functions (Ax+c) are used in many places:

- Dependencies
- Schedule
- Processor Allocation
- Memory Mapping

An Example

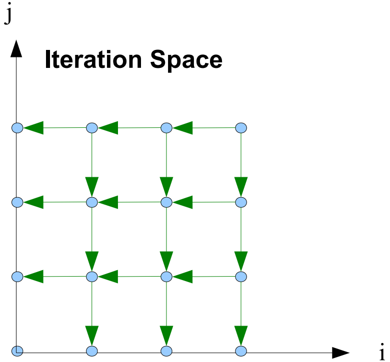
C:

```
for (t=0; t < T; t++)
  for (i=1; i < N; i++)
    for (j=1; j < N; j++)
      A[i,j] = f(A'[i-1,j], A'[i,j-1]);
```

Alphabets:

Domain(A) = {t,i,j | 0<=t<T, 0<=i,j<N}

A[t,i,j] = {i>0 || j>0} : f(A[t-1,i-1,j], A[t-1,i,j-1]);
{i=0 || j=0} : boundary values



An Example : Implicit Specification

C:

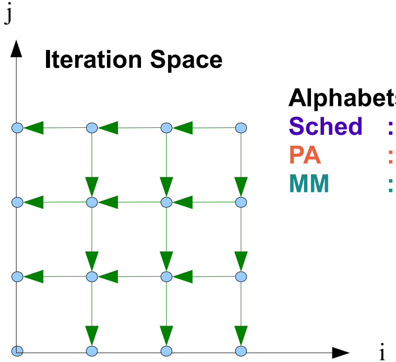
```
for (t=0; t < T; t++)
  for (i=1; i < N; i++)
    for (j=1; j < N; j++)
      A[i,j] = f(A'[i-1,j], A'[i,j-1]);
```

Alphabets:

```
Domain(A) = {t,i,j | 0<=t<T, 0<=i,j<N}
A[t,i,j] = {i>0 || j>0} : f(A[t-1,i-1,j], A[t-1,i,j-1]);
           {i=0 || j=0} : boundary values
```

C:

Sched	**: t,i,j**
PA	**: 0**
MM	**: i,j**



Alphabets:

Sched	**: not given**
PA	**: not given**
MM	**: not given**

An Example : 2D Parallelization

C:

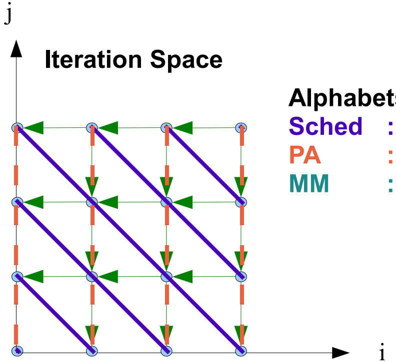
```
for (t=0; t < T; t++)
  for (i=1; i < N; i++)
    for (j=1; j < N; j++)
      A[i,j] = f(A'[i-1,j], A'[i,j-1]);
```

Alphabets:

```
Domain(A) = {t,i,j | 0<=t<T, 0<=i,j<N}
A[t,i,j] = {i>0 || j>0} : f(A[t-1,i-1,j], A[t-1,i,j-1]);
           {i=0 || j=0} : boundary values
```

C:

Sched : t,i,j
PA : 0
MM : i,j



Alphabets:

Sched : t,i+j
PA : i
MM : i,j

An Example : 2D Parallelization

C:

```
for (t=0; t < T; t++)
  for (i=1; i < N; i++)
    for (j=1; j < N; j++)
      A[i,j] = f(A'[i-1,j], A'[i,j-1]);
```

Alphabets:

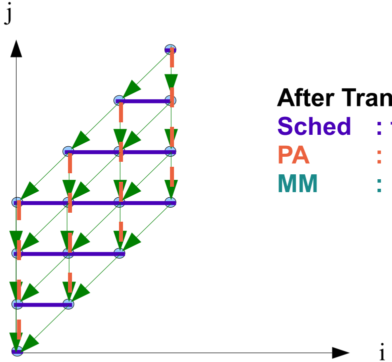
Domain(A) = {t,i,j | 0<=t<T, 0<=i,j<N}

A[t,i,j] = {i>0 || j>0} : f(A[t-1,i-1,j], A[t-1,i,j-1]);

{i=0 || j=0} : boundary values

Transformation:

Sched	**: t,i,j->t,i,i+j**
PA	**: t,i,j->t,i,j**
MM	**: t,i,j->t,i,j**



After Transformation:

Sched	**: t,j**
PA	**: i**
MM	**: i,j**

The Verifier

Given a program and TPMSpec for each variable:

1. Generate RDG
2. Verify Schedule
3. Verify Processor Allocation
4. Verify Memory Mapping

RDG : Reduced Dependence Graph
concise representation of variables and dependencies of program

Legality of Schedule

ϕ_x : Scheduling function of x
D_x : Domain of x
I : Dependence function

A[a] = ... B[I(a)] ...

Positivity:
$\forall_a \in D_A : \phi_A(a) \geq 0$
$\forall_b \in D_B : \phi_B(b) \geq 0$

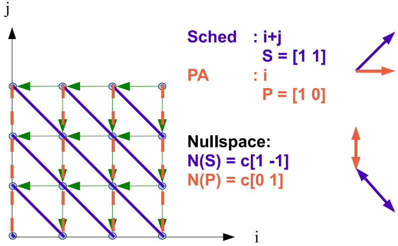
Respecting Dependence:
$\phi_A(a) \geq \phi_B(I(a)) + delay$

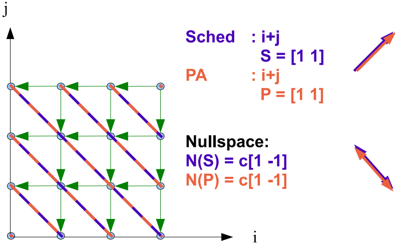
Originally formulated in the context of finding a schedule by Paul Feautrier (1992)

Legality of Processor Allocation

$Sx + s$: Scheduling function
$Px + p$: PA function

Processor allocation is legal when:
$N(S) \wedge N(P) = 0$ ... intersection of nullspaces is only at 0





Legality of Memory Mapping

μ_x : Memory mapping function of x

A[a] = ... B[I(a)] ...

First find how long a variable must stay live:
$required\_lifetime = \max_{\forall_a \in D_A}(\phi_A(a) - \phi_B(I(a)))$

w that satisfies the following:
$\mu_B(w) = 0$ ... writes to the same location
$\phi_B(w) > 0$ ... later in time
must satisfy below for the memory allocation to be legal:
$\phi_B(b + w) - \phi_B(b) \geq required\_lifetime$ ... variable B
(writes after the required_lifetime has passed)

Limitations and Future Work

Limitations:

- Restriction on input programs (no reductions)
- Performance
  - 7 sec with the example program
  - 5 minutes with a real application (18 nodes, 256 edges)

Future Work:

- Automated exploration of TPMSpec
- Code generators