Schedulers

Thanks to ISL, we AlphaZ now have two schedulers. Multi-dimensional farkas scheduler by Paul Feautrier, and PLuTo scheduling by Uday Bondhugula. For details of the schedulers, refer to the corresponding articles.

```
@article{feautrier1992some,
  title={Some efficient solutions to the affine scheduling problem. part II. multidimensional time},
  author={Feautrier, P.},
  journal={International Journal of Parallel Programming},
  volume={21},
  number={6},
  pages={389--420},
  year={1992},
  publisher={Springer}
}
```

```
@article{bondhugula2008practical,
  title={A practical automatic polyhedral parallelizer and locality optimizer},
  author={Bondhugula, U. and Hartono, A. and Ramanujam, J. and Sadayappan, P.},
  journal={ACM SIGPLAN Notices},
  volume={43},
  number={6},
  pages={101--113},
  year={2008},
  publisher={ACM}
}
```

Scheduler Commands

Schedulers take PRDG (Polyhedral Reduced Dependence Graph) as inputs and returns affine functions for each node in the PRDG. Following commands are related to scheduling.

- BuildPRDG(Program, String, int) : Builds an instance of PRDG for the specified system. Second argument is system name. Third argument is an integer value treated as a boolean value, when true the input variables are excluded from the PRDG. It is usually preferred to use this option so that input variables are not scheduled.
- FarkasMDScheduler(PRDG) : Takes a PRDG and returns List<ScheduledStatement>; uses Farkas scheduling.
- PlutoScheduler(PRDG) : Takes a PRDG and returns List<ScheduledStatement>; uses PLuTo scheduling.
- setSchedule(Program, String, List<ScheduledStatement>) : Applies the schedules computed to the TargetMapping.

Scheduling Examples

Lets take a simple program, matrix multiply, as an example. Matrix multiplication has a 3D iteration space, and the only dependence is along the k-axis for accumulating.

```
affine matrix_product {P, Q, R|P>1 && Q>1 && R>1}
       given  float A {i,k| 0<=i<P && 0<=k<Q};
              float B {k,j| 0<=k<Q && 0<=j<R};
     returns  float C {i,j,k| 0<=i<P && 0<=j<R && k==Q+1};
using // Using an accumulator locally
   float temp_C {i,j,k|0<=i<P && 0<=j<R && 0<=k<=Q};
through
   temp_C[i,j,k] = case
               // For the computation of the value of temp_C at (i,j,k),
               // we need the value of temp_C at (i,j,k-1), A at (i,k) and B
at (k,j)
               {|k>0}  : temp_C[i,j,k-1] + A[i,k-1]*B[k-1,j];
               {|k==0} : 0; // Initialization of the accumulator
            esac;
   C[i,j,k]      = temp_C[i,j,k-1];
.
```

Farkas Scheduler

The following scripts uses FarkasMD scheduling to find schedules for the above program.

```
# Load Program and store as 'prog'
prog = ReadAlphabets("../../testcases/matrix_product/matrix_product.ab");
# Define a variable 'system' to store the system name
system = "matrix_product";

prdg = BuildPRDG(prog, system, 1);
schedules = FarkasMDScheduler(prdg);
setSchedule(prog, system, schedules);
listSpaceTimeMaps(prog, system);
```

The output should look like the following:

```
C (i,j,k->k,j,i)
temp_C (i,j,k->k,j,i)
===SpaceTime Maps===
SpaceTimeMap: C (i,j,k->k,j,i) [SEQUENTIAL, SEQUENTIAL, SEQUENTIAL]
SpaceTimeMap: temp_C (i,j,k->k,j,i) [SEQUENTIAL, SEQUENTIAL, SEQUENTIAL]
```

Since Farkas scheduler tries to satisfy as much dependencies as possible in the outer dimensions, the k-axis has became the outer most dimension in the resulting schedule.

PLuTo Scheduler

The following is a similar script for matrix multiply, but uses PLuTo scheduling.

```
# Load Program and store as 'prog'
prog = ReadAlphabets("../../testcases/matrix_product/matrix_product.ab");
# Define a variable 'system' to store the system name
system = "matrix_product";

prdg = BuildPRDG(prog, system, 1);
schedules = PlutoScheduler(prdg);
setSchedule(prog, system, schedules);
listSpaceTimeMaps(prog, system);
```

The output should look like the following:

```
C (i,j,k->i,j,k)
temp_C (i,j,k->i,j,k)
===SpaceTime Maps===
SpaceTimeMap: C (i,j,k->i,j,k) [SEQUENTIAL, SEQUENTIAL, SEQUENTIAL]
SpaceTimeMap: temp_C (i,j,k->i,j,k) [SEQUENTIAL, SEQUENTIAL, SEQUENTIAL]
```

Note that now the k-axis is the innermost dimension of the resulting schedules. Because PLuTo scheduler have a different cost function, the resulting schedule can be different.

PLuTo Scheduler for Jacobi 1D Stencil

Here is another example to illustrate the difference between Farkas and PLuTo scheduling. The alphabets program below is a Jacobi-style stencil computation for 1D data. Because of the dependence `(t,i->t,i+1)`, this program is not tilable with the identity schedule.

```
affine jacobi_1d {TSTEPS,N | TSTEPS > 2 && N > 5}
given
    double A {i|0<=i<N};
returns
    double B {t,i|0<=i<N && t==TSTEPS};
using
    double temp_B {t,i|0<=i<N && 0<=t<TSTEPS};
through

    temp_B[t,i] = case
                        {|t == 0} : A[i];
                        {|t > 0 && 1<=i<N-1}  : (temp_B[t-1,i-1] +
temp_B[t-1,i] + temp_B[t-1,i+1])*0.33333;
                        {|t > 0 && 0==i } || {|t > 0 && i==N-1 } :
temp_B[t-1,i];
                    esac;
```

```
    B[t,i] = temp_B[t-1,i];
.
```

PLuTo Scheduling, which tries to find tilable shcedules, gives the following. Note that the second dimension of the schedules are shifted by `t`. This is an instance of “skewing”, and this schedule is tilable.

```
temp_B (t,i->t,i+t)
B (t,i->t,i+t)
===SpaceTime Maps===
SpaceTimeMap: temp_B (t,i->t,i+t) [SEQUENTIAL, SEQUENTIAL]
SpaceTimeMap: B (t,i->t,i+t) [SEQUENTIAL, SEQUENTIAL]
```

Farkas Scheduling does not try to produce tilable schedule, and thus gives the following schedule.

```
temp_B (t,i->t,i)
B (t,i->t,i)
===SpaceTime Maps===
SpaceTimeMap: temp_B (t,i->t,i) [SEQUENTIAL, SEQUENTIAL]
SpaceTimeMap: B (t,i->t,i) [SEQUENTIAL, SEQUENTIAL]
```