About dataflow schedules

Patrice Quinton

April 9, 2007

1 Introduction

This short note describes some features that were added to the `VertexSchedule.m` package during July 2004, in order to allow data-flow systems to be scheduled. The theory of it is described in [**?**][1] and it is briefly recalled here.

This note should be read while executing the examples in the directory `WCDMA-Periodic`. Currently, this directory is not part of the distribution of MATHEMATICA but it soon will.

2 Dataflow systems

We call *elementary dataflow system* an ALPHA system where all symbols have a first index, say i, whose domain is $\{i|i \geq 0\}$. For example:

```
--
--   This system computes the complex multiplication
--   of two infinite complex flows of data
--
system complexMult
  (aRe, bRe, aIm, bIm : {i|0<=i} of integer) -- input signal
returns
  (sRe, sIm : {i|0<=i} of integer); -- output signal
let
  sRe[i] = aRe[i]*bRe[i] - aIm[i]*bIm[i];
  sIm[i] = aRe[i]*bIm[i] + aIm[i]*bRe[i];
tel;
```

[1] Asap 2004 paper.

is a dataflow system that describes a simple complex multiplication of two infinite flows of data.

The schedule of such a system is easy to obtain, using for example the following command

```
scd[optimizationType -> Null,
       addConstraints -> {TaReD1 == TaImD1 == TbReD1 ==
       TbImD1 == 1, AaIm == 1}]
```

Notice the options: the first one avoids trying to optimize for duration, and the second one forces the value of the time component along first dimensions of variables to be 1.

The schedule a such a system can be saved in a file using the

```
saveScheduleLibrary[ onlyMainSystem -> True]
```

command (see Section 4). The suffix of the created file is `scdlib`. The structure of this file is a list of schedules; a schedule is a structure with `scheduleResult` head; the fields of this structure are the name of the system, a list corresponding to the parameters, and the list of variable schedules; the schedule of a variable contains the variable name, the list of its indexes (strings) and the schedule, in form `sched[ ... ]`.

Here is the schedule of `complexMult`, as it appears in the `WCDMA-Periodic` directory (this file is protected):

```
{scheduleResult["complexMult", {}, {{"aIm", {"i"}, sched[{"$P"}, 0]},
   {"aRe", {"i"}, sched[{"$P"}, 0]}, {"bIm", {"i"}, sched[{"$P"}, 0]},
   {"bRe", {"i"}, sched[{"$P"}, 0]}}, {{"sIm", {"i"}, sched[{"$P"}, 1]},
   {"sRe", {"i"}, sched[{"$P"}, 1]}}]}
```

In this example, there is a special feature: the schedule of the variables contains a symbolic term `"$P"`, which means that the schedule is given up to a symbolic *period* factor $\$P$. This factor was set manually in the schedule file. It will allow a dataflow schedule to be computed later on.

3 Up-sampling and down-sampling

Two special systems are available: `oversampling` and `undersampling`[2]. The text of these systems is given in appendix A and B, respectively. The first system takes the up-sampling by a K factor of its input e and returns it in its output s (the input is repeated K times). The second one gives the down-sampling by a factor K of its input e and returns it in its output s.

[2]The correct name should be up and down sampling...

The schedule of the up-sampler, as given in the file `overSampling.scdlib` is:

```
{scheduleResult["overSampling", {"K"},
  {{"e", {"i", "K"}, sched[{"$P", 0}, 0]}},
  {{"s", {"j", "K"}, sched[{"$P"/"K", 0}, 0]}}]}
```

It is actually a *symbolic scheduling*, given by the MATHEMATICA expression `$P/K`. This means that if the input flow e is run at period $\$P$, then the output s is run at period $\$P/K$.

In a symmetric way, the schedule of the down sampler is (as found in file `underSampling.scdlib`:

```
{scheduleResult["underSampling", {"K"},
  {{"e", {"i", "K"}, sched[{"$P"/"K", 0}, 0]}},
  {{"s", {"j", "K"}, sched[{"$P", 0}, 0]}}]}
```

4 Saving a Schedule

To obtain the schedule of a subsystem, the simplest way is to schedule it (using either `scd` or `schedule`), then to save it using the command:

```
saveScheduleLibrary[  onlyMainSystem -> True]
```

This creates a file `name.scdlib` where `name` is the current system name. The schedule can be edited manually and changed in order to contain the periodic factor `"$P"`. It can be loaded in the `$scheduleLibrary` variable using

```
loadScheduleLibrary[ "systemName" ]
```

The `saveScheduleLibrary` command has an option `onlyMainSystem`, the default option of which is `False`.

5 Scheduling a dataflow system

To schedule a system which contains a set of subsystems, one uses the `scd` scheduler using the `periods` option

```
scd[ optimizationType -> Null, periods -> {p1, p2, ... }, ... ]
```

where `periods` is mapped to the list of (integer) periods of the use statements of the calling system.

The order of these variables is that given in the dependence table and it can be seen by printing the second part[3] of this table using the `dep` function:

```
ashow[ dep[][[2]] ]
```

will print the second part of the dependence table.

For the moment, computing the periods is left to the user, but the method is not difficult (see [**?**]): it amounts to solving a system of homogeneous equations of the form $p2 = Kp1$, where, for example, $p2$ is the period of a subsystem which is down sampled from the output of a down sampler (and symmetrically for up sampling).

In the current model, we assume that there exists only on level of hierarchy in the subsystems; we also assume that a given subsystem has a uniform period. These hypothesis could be relaxed, by assuming that a data-flow system has inputs and outputs with different dataflow periods.

6 A First Example: the WCDMA Emitter

In the example notebook, look at the WCDMA emitter example.

```
load["WCDMAemitter.alpha"];
```

loads the full program. Then

```
loadScheduleLibrary["ComplexMult"];
loadScheduleLibrary["OverSampling"];
loadScheduleLibrary["OVSF"];
loadScheduleLibrary["KASI"];
loadScheduleLibrary["Nyquist"];
loadScheduleLibrary["fir128_u"];
```

[3] The structure of the dependence table is as follows:

- `dtable[ list of depend, list of dependuse ]`,
- a `depend` is a structure containing a domain, the dependent var, the rhs var and a matrix that describes the dependence;
- a `dependuse` is a structure containing the subsystem name, the list of input name, the list of output names, the rank of the use in the program, the domain of its parameters, and the matrix of its parameters.

loads in `$scheduleLibrary` the schedules of the subsystems used in the emitter. The scheduler is called by the following command

```
scd[optimizationType -> Null, durations -> {0, 0, 1, 1, 1,
   1, 0, 0}, periods -> {16,
    1024, 4, 4, 4, 4, 4, 4, 4, 1, 1, 1},
      addConstraints -> {TscD1 == TscMirrD1 == TsdD1,
      TkascontrolCodeD2 ==
      TkascontrolCodeD3 == TkascontrolCodeD4 ==
      TkasdataCodeD2 == TkasdataCodeD3 == TkasdataCodeD4 ==
      TNyquistCodeD2 ==
      TNyquistCodeD3 == TNyquistCodeD4 ==
       TNyquistCodeD5 == TovsfControlCodeD2 ==
        TovsfControlCodeD3 == TovsfControlCodeD4 ==
         TovsfControlCodeD5 == TovsfDataCodeD2 ==
        TovsfDataCodeD3 == TovsfDataCodeD4 == 0,
        TsccontrolD2 == TsccontrolD3
        == TsccontrolD4 == TsccontrolD5 == TscdataD2 ==
        TscdataD3 == TscdataD4 ==
        TscdataD5 == 0, TscMirrD2 == TscMirrD3 ==
        TscMirrD5 == 0, TsdMirrD2 ==
         TsdMirrD3 == TsdMirrD5 == 0, TspcontrolD4 ==
         TspcontrolD2 ==
         TspcontrolD3 ==
        TspcontrolD5 == 0, TspdataD4 == 0},
        objFunction -> TscD1 + TscD2 +
        TscD3 + TscD4 + TscD5 + TsdD1 + TsdD2 +
        TsdD3 + TsdD4 + TsdD5]
```

This gives the following schedule. Here are some explanations about the options.

- The `optimizationType` option allows a schedule to be found even when the domain of the variables is infinite (the default option tries to optimize the total scheduling time, and this would fail). This is explained in the scheduler's manual.
- The `durations`option allows one to assign different integral durations to each dependence. The order of the integer in this list is given by the order of the dependences given by the `show[ dep[] ]` command.

`control[j]`	0
`controlMirr[j]`	$1024j$
`data[i]`	0
`dataMirr[i]`	i
`kascontrolCode[j]`	$4j$
`kasdataCode[i]`	$4i$
`NyquistCode[j]`	$j+3$
`ovsfControlCode[j]`	$4j$
`sc[j]`	$j+2+KN$
`sccontrol[i]`	$4i+2$
`scdata[i]`	$4i+2$
`scMirr[j]`	$j+2+KN$
`sd[i]`	$2+i+KN$
`sdMirr[i]`	$2+i+KN$
`spcontrol[j]`	$4j$
`spdata[i]`	$4i$
`ssccontrol[i]`	$i+2$
`sscdata[i]`	$i+2$
`sscontrol[i]`	$4i$
`ssdata[i]`	$4i$

Table 1: Schedules of the variables of the WCDMA emitter

- The `periods` option is new. It allows one to provide an integral periodic factor for each one of the subsystems. The order of the periods corresponds to the order of the subsystems as given in the dependence table, and as shown[4] by the `show[ dep[]]`. To know more about how to find the periods, see Section 8. Period values are assigned to the $\$P$ parameter of each subsystem schedule, in such a way that this schedule is adapted to the rate at which the system is able to run.
- Constraints to the schedule are given in the `addConstraints` option.
- Finally, the objective function is given in the `objFunction` parameter.

Another, more complex, example is given in the notebook.

7 How to Find a Schedule in Practice

When solving this example, I did not found immediately the right schedule parameters. Here are some hints:

- Try to schedule a system in an incremental fashion: comment out some subsystems until you find out a satisfying solution for some part of the system, then add progressively new subsystems. Indeed, when the scheduler fails, it is very difficult to find out which one of the constraints is not met and the reasons why it failed.
- Setting the durations, the additional constraints, and the objective function may be postponed until you find a solution to the whole systems with periods. Actually, the minimal parameters for the WCDMA emitter are:

```
scd[optimizationType -> Null,
  periods -> {16, 1024, 4, 4, 4, 4, 4, 4, 4, 1, 1, 1}]
```

This does not provide the optimal schedule

8 How to Find Periods

[4] A recent modification done on April 9, 2007.

A Up-sampling

```
--
--   This system oversamples an infinite integer
--   input signal e with an over sampling factor K
--
system overSampling: {K|1<=K}
 (e : {i|0<=i} of integer) -- input signal
returns
 (s : {j|0<=j} of integer); -- output signal
var
 -- indexj[j] has value j
 -- jmodk[j] has value j mod K
 indexj, jmodk : {j|0<=j} of integer;
 -- The trick... Build this infinite array...
 E: {i,j|0<=i; 0<=j} of integer;
let
  -- Definition of indexj
  indexj[j] =
   case
     {|j=0}: 0[];
     {|j>0}: indexj[j-1]+1[];
   esac;
  -- Definition of jmodk
  jmodk[j] =
  case
    {|j<K}: indexj[j];
    {|j>=K}: indexj[j-K];
  esac;
  -- Definition of E.
  E[i,j] =
   case
    {|j=0}: e[i];
    {|j>0}: if jmodk[j]=0[] then E[i+1,j-1] else E[i,j-1];
   esac;
  -- The result : take the first row of this infinite array
  s[j] = E[0,j];
tel;
```

B Down-sampling

```
system underSampling: {K|1<=K}
 (e : {i|0<=i} of integer) -- input signal
returns
 (s : {j|0<=j} of integer); -- output signal
var
 -- indexj[j] has value j
 -- jmodk[j] has value j mod K
 indexi : {i|0<=i} of integer;
 kindexj: {j|0<=j} of integer;
 kvalue: {k|0<=k<=K} of integer;
  -- The trick... Build this array...
 E: {i,j|0<=i; 0<=j} of integer;
let
  -- Definition of kvalue
  kvalue[k] =
    case
      {|k=0}: 0[];
      {|k>0}: kvalue[k-1]+1[];
    esac;
  -- Definition of indexi
  indexi[i] =
   case
     {|i=0}: 0[];
     {|i>0}: indexi[i-1]+1[];
   esac;
  -- Definition of kindexj
  -- This variable has value K times j
  kindexj[j] =
  case
    {|j=0}: 0[];
    {|j>0}: kindexj[j-1]+kvalue[K];
  esac;
  -- Definition of E.
  E[i,j] =
   case
    {|j=0}: e[i];
    {|j>0}: if indexi[i]<kindexj[j] then E[i+1,j] else E[i+1,j-1];
   esac;
```

```
  -- The result : take the first row of this
  s[j] = E[0,j];
tel;
```