User manual of the Alpha **Scheduler**

Tanguy Risset

April 29, 1999

This documentation provides help for the user of the ALPHA scheduler. It is available in the file `$MMALPHA/doc/user/Scheduler_user_manual.ps`

1 Introduction

The scheduler is a package of MMALPHA that attempt to to find a schedule for given ALPHA program. An ALPHA program [Wil94, Mau89] has no sequential ordering indicated: any sequential or parallel ordering of computations is semantically valid as soon as it respect the data dependencies of the program.

The basic goal of the scheduler is to find a valid execution order that is *good* with respect to a particular criterion. The theoretical basis for the scheduling process is inherited from the systolic array synthesis researches and from the automatic parallelization researches. Currently two techniques are implemented for the computation of a schedule function, we will call them the *Farkas method* [Fea92b] (method used by default), and the *vertex method* [MQRS90]. Scheduling an ALPHA program consists in giving a computation date for each variable of the program. The time is considered as a discrete single clock.

For computing a schedule, we have to gather all the constraints that must verify the schedule in a linear programming problem (LP) and to solve it with a particular software. the two methods implemented proceed their computation differently but both give a *mono-dimensional affine by variable* schedule. They can also provide *multidimensional affine by variable* schedule by changing the options of schedule (see section 2).

A mono-dimensional affine by variable schedule assign to each computation: `A[i,j] = ...`, an execution date $T_A(i,j)$ which is given by an affine function of the indices (i,j) and the parameters (N):

$$T_A(i,j) = \tau_A^i i + \tau_A^j j + \tau_A^N N + \alpha_A \quad ,$$

$(\tau_A^i, \tau_A^j, \tau_A^N)$ is called the *scheduling vector*. for instance: $T_A(i,j) = 2i + j + 3$ is a mono-dimensional affine schedule for variable A, the corresponding scheduling vector is $(2,1)$. One important subset of such schedules are the *affine with constant linear part* schedules, where the value of the scheduling vector is the same for each local variable of the program (in this case, all variables must have the same number of indices):

Multidimensional schedules are necessary when program do not have linear execution time (for instance, if the execution time is N^2). In that case, the schedule function is a vector of linear function and the order is the lexicographic order on the components (see [Fea92a] for details).

2 The schedule function

The function to be called in order to schedule a program is: `schedule`. Its effects is to schedule the ALPHA program contained in `$result` by default, and to put the result in the global variable `$schedule` (see 4 for the structure of the resulting schedule). Options allow to set several parameters (see section 3). The possible forms of the use of the function `schedule` are:

- `schedule[]`
  finds an affine by variable schedule for `$result` which minimizes the global execution time and assign this schedule to `$schedule`
- `schedule[sys]`
  finds a schedule affine by variable for the ALPHA system `sys` which minimizes the global execution time and assign it to `$schedule`

- `schedule[option_1->value_1,...,option_n->value_n]`
  find a schedulee for `$result` which respects the chosen options and assign it to `$schedule`
- `schedule[sys,option_1->value_1,...,option_n->value_n]`
  find a schedulee for the ALPHA system `sys` which respects the chosen options and assign it to `$schedule`

3 Important options of schedule

These options have default values indicated hereafter. To change these values, put one of the corresponding rules as a parameter to the `schedule` function. the Farkas method and the vertex method rely on completely different implementations, hence, some options are only implemented in one of the method. Choosing between the Farkas method and the vertex method is done with the `schedMethod` option.

schedMethod

this option allow to switch from the Farkas method to the vertex method. Its type is symbolic. Changing the value of this option greatly influence the other options (for instance, it changes default values of some options: `durationByEq`,...), be sure to check the others options you use.

the possible values are:

- `schedMethod -> farkas` (default) select the Farkas method.
- `schedMethod -> vertex` select the vertex method.

One of the differences between the two methods is that the Farkas method uses the Pip software to solve the LP while the vertex method uses Mathematica linear solveur.

scheduleType

This option gives the type of schedule found. Its type is symbolic, the possible values are:

- `scheduleType -> affineByVar` (default) affine by variable scheduling;
- `scheduleType -> sameLinearPart` affine by variable scheduling with constant linear part. this option is often used for systolic designs.
- `scheduleType -> sameLinearPartExceptParam` affine by variable scheduling with constant linear part except for the parameters.

multidimensional

This indicates whether we look for a mono-dimensional schedule (default value) or a multidimensional schedule. Of course, searching for a multi-dimensional schedule may end up with a mono-dimensional schedule, but we cannot use all the options available (for instance, we cannot optimize the global time because multi-dimensional schedule tries to minimize the number of dimension of the schedule). The type of this option is boolean, hence the possible values are:

- `multidimensional -> False` the schedule will be mono-dimensional.
- `multiDimensional -> True` the schedule will be multidimensional. If you set this options, be aware that it greatly influence the value of other options: `addConstraints`, `durations` must be of type list of list (one list by schedule dimension, this means that you have to know the number of dimension of the resulting schedule). Options `optimizationTime` is automatically set to multi.

optimizationType

This option gives the objective function chosen. Its type is symbolic, the possible values are:

- `optimizationType -> time` (default) the total latency is minimized (In the Farkas method, this minimization always correspond to the lexicographic minimization of the coefficients of the global execution time which is an affine function of the parameters. In the vertex method the way this minimization is performed can be changed by the option `objFunction`).
- `optimizationType -> Null` no objective function (the coefficients of the scheduling vectors are minimized in a lexicographic order)
- `optimizationType -> delay` tries to minimize the delays on the dependencies (not implemented currently).
- `optimizationType -> multi` is for internal use, when a multidimensional scheduling is computed, just be aware that the three above value for this option are not compatible with multi-dimensional scheduling.

addConstraints

This option allows to add some constraints to the LP generated. Adding constraints is very important if you want to control the resulting schedule. Its type is a list of string, each string representing a constraint. The type of constraints authorized are affine constraints on scheduling vectors. There are two types of constraint added, one can force a scheduling vector value or simply set linear constraints on its components:

- forcing a variable $A[i,j]$ to be scheduled at time i+2j+2 can be done with the constraints: `"TA[i,j]=i+2j+2"`.
- for more precise constraint, one can directly access to each components of the schedule functions of each variable. For instance `TAD2` will represent the variable τ_A^j (and `CA` will represent the variable α_A). With these names, one can set linear constraints on these variable using operators `==` or `>=`. For instance, {`"TAD1 == 1","TAD2 == 2", "CA >= 2"` } is the same constraint as above except that the constant is allowed to be greater than two.

This options can also be used during a multidimensional scheduling process. In that case, its value is a list of list of string, each list of string corresponding to a dimension of the schedule. Example of use for mono-dimensional scheduling:

```
schedule[addConstraints->{"TA[i,j,N]=i+2j-2", "T1D2==2","T1D1+2T1D3>=1"}]
```

durations

Allow the user to define exactly the execution time for each equation or for each dependence. This is very useful for hardware design, for instance, some trivial equation like A=B should not last one top but zero because it is just a different name for the same signal. its type is list of integer. This options is affected by the value of the option `durationByEq`.

The possible values for the durations option are:

- `durations ->` {} each equation (if durationByEq is True) is 1 (resp. each dependence is 1 if durationByEq is False), whatever complex is the computation (default value).
- `duration ->` {`0,0,1,1,1,0,...0,3,1`} (List of integer),
  - If the durationByEq option is set to True (default in farkas method): integer number i indicates the duration of the equation defining variable number i of the Alpha program (the numbering of the variables is done in this order: starting with input variable, then output variable, and finally local variable, hence in declaration order).
  - If the durationByEq option is set to False (default in vertex method), integer number i indicating the duration of dependence number i.

outputForm

This options allow to have a non standard schedule output. the standard schedule output is described in section 4. You can also obtain as a result the LP to solve in various format or the schedule polytope, i.e. the polytope which contains all the valid solutions in Alpha'domain format. Its type is symbolic

- `outputForm -> scheduleResult` (default) standard schedule output form (Alpha'ScheduleResult structure, see section 4).
- `outputForm -> lpSolve` output the linear programming problem to solve in order top find the schedule in the format of the lp_solve software.
- `outputForm -> lpMMA` output the linear programming problem to solve in order top find the schedule in the format of the linear resolver of Mathematica.
- `outputForm -> domain` output the schedule polytope, i.e. Alpha'domain which is composed of all the constraints of the LP to solve. WARNING, this works only for SMALL programs (3 instructions), otherwise the domain is too big to be handled by polylib.

debug

Print more information and do not destroy the temporary files build for the interface with PIP. the type of this option is boolean, possible value are:

- `debug -> False` (default) debug mode not set;
- `debug -> True` debug mode set.

verbose

indicate not to print anything, just to return the result. its Type is boolean, possible value are:

- `verbose -> True` (default) normal printing;
- `verbose -> False` nothing is printed out.

3.1 Advanced options of schedule

These options are here for advanced users of the schedule function.

resolutionSoft

indicates which software is used for the resolution of the LP. its type is symbolic, the possible values are:

- `resolutionSoft->pip` use P. Feautrier's PIP software (only available for the Farkas method).
- `resolutionSoft->mma` use the ConstrainedMin function of Mathematica (only implemented in the vertex method).
- `resolutionSoft -> lpSolve` use the `lp_solve` software(not implemented yet)

objFunction

This option is used to indicate how the minimization of the objective function (which is usually a function of the parameters) is performed. its type is symbolic, the possible values are:

- `objFunction->lexicographic` minimize lexicographically the coefficient of this function in the order of the declaration of the corresponding parameters in the Alpha program (default in Fakas implementation).

- `objFunction->lexicographic["N","P","M"]` (not implemented anywhere)
- `objFunction->2"N" + "P"` minimize 2 time the coefficient of parameter "N" plus one time the coefficient of parameter "P" (only implemented in the vertex resolution).

onlyVar

indicates which variables to schedule (useful if you have a very long program and which to schedule only part of it. its type is a list of string, the possible values are:

- `onlyVar->all` (default value) schedule all the variables.
- `onlyVar->{"a","B","c"}` (list of string) schedule only the specified variables, Warning, if some variable are needed for the computation of the one indicated, the function will try to find their execution dates in $schedule (only implemented in the Farkas method)

onlyDep

indicates which variables to schedule (used for multidimensional scheduling). its type is a list of integer the possible values are:

- `onlyDep->all` (default value) schedule all the dependencies.
- `onlyDep->{1,4,5}` (list of integer) schedule only the specified dependencies, the number correspond to their position in the list of dependencies which is returned by the dep[] function (only implemented in the Farkas method)

subSystems

indicates whether or not he schedule takes into account the calls to other systems.

- `subSystems->False` (default value).
- `subSystems->True`.

subSystemsSchedule

indicates where are the schedules of the subsystems used in the system we schedule. its type is a list of schedule

- `subSystemsSchedule->$scheduleLibrary` (default value).
- `subSystemsSchedule->List[schedule..]`.

4 Result form

The result given by the function `schedule` has a special form. Two new head names are introduced: `Alpha‘scheduleResult` and `Alpha‘sched`. The outermost structure is a structure starting with the head `Alpha‘ScheduleResult`, where the first argument is the name of the system (string) and where second argument is the schedule itself. The Last argument of `Alpha‘scheduleResult` is for internal use. The syntax of the structure is described here.

```
<schedResult>=Alpha‘ScheduleResult[scheduleType_Integer,<sched3List>,objFunc_]
          {{ nameVar_String,
<sched3>=  indices_List,
           Alpha‘sched[tauVector_List,constCoef_Integer] }
```

example:

```
scheduleResult["test1",
   {{"a", {"i", "k"}, sched[{2, 2}, -4]},
    {"b", {"j"}, sched[{0}, 0]}, {"c", {"i"}, sched[{0}, 15]},
    {"A", {"i", "k"}, sched[{2, 2}, -3]},
    {"C", {"i", "k"}, sched[{2, 2}, -2]}}, {15}]
```

Other examples are given in section 6.

5 Technical settings

The packages is called `Schedule.m`, it uses packages `FarkasSchedule`, `VertexSchedule` and `scheduleTools.m`. The Farkas implementation uses the `pip` software (version D.1 [Fea88]) which must be present in the binary directory. the communication between mathematica and `pip` is done by files. These files are written in the directory indicated by $tmpDirectory. Currently, all method should work on solaris and Windows plateform.

6 Examples

Consider the program of figure 1, in the following, we give the result of the schedule with different options.

default use

If you type the following command (after having loaded the program of figure 1):
`schedule[]`
the output written on the screen session should look like:

```
In[14]:= schedule[]
Dependence analysis...
Building LP...
LP: 154      variables, 140 Constraints
Writing file for PIP....
Writing line 100
Solving the LP...
Version D.1

cross : 2714096, alloc : 1, compa : 0

n 1 u 107''' s 0'''
Shift coef: 0
Total execution Time: 1 + 2 N
T_a{i, j, N} = 0
T_b{i, j, N} = 0
T_c{i, j, N} = 1 + 2 N
T_B{i, j, k, N} = i
T_A{i, j, k, N} = j
T_C{i, j, k, N} = k + N


Out[10]= scheduleResult[1, {{a, {i, j, N}, sched[{0, 0, 0}, 0]},
   {b, {i, j, N}, sched[{0, 0, 0}, 0]}, {c, {i, j, N}, sched[{0, 0, 2}, 1]},
   {B, {i, j, k, N}, sched[{1, 0, 0, 0}, 0]},
   {A, {i, j, k, N}, sched[{0, 1, 0, 0}, 0]},
   {C, {i, j, k, N}, sched[{0, 0, 1, 1}, 0]}}, {2, 1}]
```

```
system  prodVect   : {N | N ≥ 2}
                  (a : {i,j | 1 ≤ i ≤ N;1 ≤ j ≤ N} of integer;
                   b : {i,j | 1 ≤ i ≤ N;1 ≤ j ≤ N} of integer)
          returns (c : {i,j | 1 ≤ i ≤ N;1 ≤ j ≤ N} of integer);
var
    B : {i,j,k | 1 ≤ i ≤ N;1 ≤ j ≤ N;1 ≤ k ≤ N} of integer;
    A : {i,j,k | 1 ≤ i ≤ N;1 ≤ j ≤ N;1 ≤ k ≤ N} of integer;
    C : {i,j,k | 1 ≤ i ≤ N;1 ≤ j ≤ N;0 ≤ k ≤ N} of integer ;
let

B[i,j,k] =
      case

          { | i = 1} : b[k,j];
          { | 2 ≤ i ≤ N} : B[i-1,j,k];

      esac;

A[i,j,k] =
      case

          { | j = 1} : a[i,k];
          { | 2 ≤ j ≤ N} : A[i,j-1,k];

      esac;

C[i,j,k] =
      case

          { | k = 0} : 0[];
          { | 1 ≤ k} : C[i,j,k-1] + A[i,j,k] * B[i,j,k];

      esac;

c[i,j] = C[i,j,N];

tel;
```

Figure 1: Uniform matrix matrix product

The first lines indicates which computations are performed. the lines starting by `Version` are output by Pip. Then the result is printed on the screen. Here, for instance the schedule is affine by variable. `B[i,j,k]` is computed at time `i`. The result (after `Out[11]`) is the corresponding Mathematica structure assigned to `$schedule`.

Adding a constraint

Suppose that we want to impose that the variable `A[i,j,k]` is schedule at time $i+j+2k+7$ and that $\tau_A^i = \tau_B^j$. We have to add the two constraints: `"TA[i,j,k]=i+j+2k+7"` and `"TAD1==TBD2"`, hence the command is:

```
In[15]:= schedule[addConstraints->{"TA[i,j,k]=i+j+2k+7","TAD1==TBD2"}]
```

The result is

```
T_a{i, j, N} = 0
T_b{i, j, N} = 0
T_c{i, j, N} = 9 + i + j + 2 N
T_B{i, j, k, N} = i + j
T_A{i, j, k, N} = 7 + i + j + 2 k
T_C{i, j, k, N} = 8 + i + j + 2 k
```

7 Problems

7.0.1 If no schedule is found

If no schedule is found, a message is output telling the user that there maybe several reasons for that:

- No schedule exists (no way of solving this problem).
- No schedule of the chosen type exists. In the case, you can try the old schedule which was more powerful (ask a ALPHAdeveloper).
- No affine one dimensional schedule exists (no way of solving this problem)
- There exists a schedule but the time is not bounded. In this case try with the option `objFunction` set to 1).
- The program is not Semantically correct, try analyze.

7.0.2 Awful error messages

In general, when something went wrong, the error is capture correctly. Sometime, the error may come with the following message `General::aofil: /tmp/mat.tmp already open as /tmp/mat.tmp.`
`OpenWrite::noopen: Cannot open /tmp/mat.tmp.`
`General::stream: $Failed is not a string, InputStream[ ], or OutputStream[ ].`
This may come from the fact that you have interrupted the previous execution of schedule. Thus Mathematica tries to open file which are not closed. You can close these files (here `/tmp/mat.tmp`) by typing: `Close["/tmp/mat.tmp"]`. In general it is very dangerous to interrupt the evaluation of the schedule function.

References

[Fea88] P. Feautrier. Parametric integer programming. *RAIRO Recherche Opérationnelle*, 22:243–268, September 1988.

[Fea92a] P. Feautrier. Some efficient solution to the affine scheduling problem, part II, multidimensional time. *Int. J. of Parallel Programming*, 21(6), December 1992.

[Fea92b] P. Feautrier. Some efficient solutions to the affine scheduling problem, part I, one dimensional time. *Int. J. of Parallel Programming*, 21(5):313–348, October 1992.

[Mau89] C. Mauras. *Alpha : un langage équationnel pour la conception et la programmation d'architectures parallèles synchrones*. PhD thesis, Université de Rennes 1, IFSIC, December 1989.

[MQRS90] Christophe Mauras, Patrice Quinton, Sanjay Rajopadhye, and Yannick Saouter. Scheduling affine parameterized recurrences by means of variable dependent timing functions. In S.Y Kung, Jr. E.E. Swartzlander, J.A.B. Fortes, and K.W. Przytula, editors, *Application Specific Array Processors*, pages 100–110. IEEE Computer Society Press, September 1990.

[Wil94] D. Wilde. The alpha language. Technical Report 827, IRISA, Rennes, France, Dec 1994.