User manual of the Alpha **Scheduler**
Version of July 10, 2006

Tanguy Risset and Patrice Quinton

July 10, 2006

Contents

This documentation[1] provides some information regarding the ALPHA schedulers. It is available in the file:
`$MMALPHA/doc/user/Scheduler_user_manual.pdf`

Section 1 describes the principles of the Farkas and the vertex schedulers. Section 2 presents the main utilization modes of the `schedule` function, while section 3 describes the options of this function. In section 4 the output format of the scheduler is detailed. Section 5 describes briefly the `Schedule.m` package. A short example is shown in section 6. Troubles are described in section 7.

1 Introduction

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

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. 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 research on systolic array synthesis and on automatic parallelization. Currently two techniques are implemented for the computation of a scheduling function, we will call them the *Farkas method* [Fea92b] (method used by default), and the *vertex method* [MQRS90].

In order to compute a schedule, the scheduler gathers all the constraints that the schedule must meet and builds a linear program (LP) which is then solved with a particular software. The two implemented methods proceed differently but both give a *mono-dimensional affine by variable* schedule. They can also provide a *multi-dimensional affine by variable* schedule by changing the options of the schedule (see section 2).

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

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

[1] It improves and replaces the previous scheduler manual that was called `docSched.tex`. Unfortunately, this documentation in far from being complete.

The vector $(\tau_A^i, \tau_A^j, \tau_A^N)$ is called the *scheduling vector*. For instance: $T_A(i,j) = 2i + j + 3$ is a mono-dimensional affine schedule for variable A; Its corresponding scheduling vector is $(2, 1)$.

One important subset of mono-dimensional affine schedules are *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 used for programs that do not admit a linear execution time (for instance, if the execution time is N^2). In this case, the schedule function is a vector of linear functions and the order that is defined by the schedule is the lexicographic order on the components (see [Fea92a] for details).

2 The MmAlpha `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 the `$result` MmAlpha variable, and to put the result in the global variable `$schedule` (see section 4 for the structure of the resulting schedule). Options allow several parameters to be changed (see section 3).

The possible forms of the use of the function `schedule` are:

- `schedule[]`: finds an affine by variable schedule for `$result` which minimizes the global execution time of the system and assigns this schedule to `$schedule`.
- `schedule[sys]`: finds an affine by variable schedule for the Alpha system `sys`; the schedule minimizes the global execution time. The resulting schedule is assigned to `$schedule`
- `schedule[option_1->value_1,...,option_n->value_n]`: finds a schedule for `$result` which respects the chosen options and assign it to `$schedule`
- `schedule[sys,option_1->value_1,...,option_n->value_n]` find a schedule for the Alpha system `sys` which respects the chosen options and assigns it to `$schedule`.

3 Options of the `schedule` function

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 have completely different implementations, hence their options are sometimes different. Choosing between the Farkas method and the vertex method is done with the `schedMethod` option.

3.1 Main options

schedMethod

This option[2] allows one to switch from the Farkas method to the vertex method. Its type is symbolic. Changing the value of this option greatly influences 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) selects the Farkas method.
- `schedMethod -> vertex` selects the vertex method.

One of the main differences between the two methods is that the Farkas method uses the `Pip` software to solve the LP while the vertex method uses the MATHEMATICA linear solver.

scheduleType

This option gives the type of schedule that `schedule` looks for. Its type is symbolic, and its possible values are:

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

The default option allows one to get any kind of affine schedule. Option `sameLinearPart` constraints the schedule to have all a common linear part. As a consequence, all variables must have the same dimension (not checked).

[2] As of July 10, 2006, this option does not work. To use the vertex method, run the `scd[]` command.

multidimensional

By default, the scheduler looks for a mono-dimensional schedule. If the `multidimensional`is set, the scheduler also looks for multi-dimensional schedules. This option is boolean, hence its possible values are:

- `multidimensional -> False` (default): the schedule will be mono-dimensional.
- `multiDimensional -> True`: the schedule will be multidimensional. If you set this options, be aware that it greatly influences 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 dimensions of the resulting schedule). Options `optimizationTime` is automatically set to multi.

optimizationType

This option sets the objective function chosen. Its type is symbolic, and its 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 values for this option are not compatible with multi-dimensional scheduling.

addConstraints

This option allows one to add some constraints to the generated LP. Adding constraints is very important if you want to control the resulting schedule.

The type of this option is a list of strings, each one of which represent a constraint. The type of constraints authorized are affine constraints on scheduling vectors. Constraints can have two forms: one can force the value of a scheduling vector or one can force coefficients of schedule to meet some constraints:

- Forcing a variable $A[i,j]$ to be scheduled at time $i+2j+2$ can be done by the adding `"TA[i,j]=i+2j+2"` to the constraints list.
- For more precise constraints, one can directly access each component of the schedule functions of each variable. For instance `TAD2` represents coefficient τ_A^j of the schedule of A and `CA` represents the constant coefficient α_A). With these names, one can add linear constraints on these coefficients 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.

These options can also be used during the search of a multi-dimensional schedule. In that case, its value is a list of lists of string, each list of strings corresponding to constraints imposed on 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

This option allows the user to specify precisely the execution time for each equation or for each dependence. By default, each equation is assumed to need exactly one cycle to be evaluated. Sometimes, one want to be more precise, especially when designing a circuit. For example, an equation of the form `A = B` imposes that `A` be scheduled one cycle after `B`.

The type of this option is a list of integers. It may be affected by the value of the `durationByEq` option.

For the Farkas method, the possible values of this option are:

- `durations ->` {} (default): each equation lasts one cycle.
- `duration ->` {`0,0,1,1,0,...0,3,1`} (list of integers):
  - If the `durationByEq` option is set to `True` (default in the Farkas method): integer number i indicates the duration of the equation

defining variable number i of the Alpha program. The variables are numbered in their order of declaration in the ALPHA system: input variables, then output variables, and finally local variables.

– If the `durationByEq` option is set to `False` (default in the vertex method), integer number i indicating the duration of dependence number i. The dependencies are numbered in the order given by the `dep` function: to get the list of dependences run the function `show[ dep[] ]`.

Note: as of July 10, 2006, I think that the option `durationByEq -> True` is only valid for the Farkas method and the option `durationByEq -> False` is only valid for the vertex method.

outputForm

This options allows a non standard output to be obtained. The standard schedule output is described in section 4. One can also obtain as a result the linear program to be solved in various formats, or the schedule polytope, i.e. the polytope which contains all the valid solutions in the ALPHA format. The possible values of `outputForm` are:

- `outputForm -> scheduleResult` (default): standard schedule output form (Alpha'ScheduleResult structure, see section 4).
- `outputForm -> lpSolve`: outputs in the format of the lp_solve software, the linear programming problem to solve in order to find the schedule.
- `outputForm -> lpMMA`: outputs the linear programming problem to solve in order top find the schedule in the format of the linear solver of Mathematica. Warning; as of July 10, 2006, this option is not implemented.
- `outputForm -> domain`: outputs the schedule polytope, i.e. the 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

Prints additional information and does 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

If set, the scheduler returns additional information, otherwise, it returns only the result. This option is boolean, its possible values are:

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

3.2 Advanced options

These options are here for an advanced use of the schedule function.

resolutionSoft

This function indicates which software will be used to solve the LP. Its type is symbolic, the possible values are:

- `resolutionSoft -> pip`: uses P. Feautrier's PIP software (only available for the Farkas method).
- `resolutionSoft -> mma`: uses the `ConstrainedMin` function of MATHEMATICA (only implemented in the vertex method).
- `resolutionSoft -> lpSolve`: uses 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`: minimizes 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 strings, the possible values are:

- `onlyVar -> all` (default value): schedules all the variables.
- `onlyVar -> {"a","B","c"}` (list of strings): schedules only the specified variables, Warning, if some variables are needed for the computation of the variables listed in the option, the function will try to find their execution dates in $schedule (This feature is only implemented in the Farkas method).

onlyDep

Indicates which dependences to schedule (used for multidimensional scheduling). Its type is a list of integers and its possible values are:

- `onlyDep -> all` (default value): schedules all the dependencies.
- `onlyDep -> {1,4,5}` (list of integers): schedules 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 the schedule takes into account calls to other subsystems.

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

In the Farkas method, the default value is not to consider subsystems, and if there are some calls, the method fails. Therefore, to schedule a structured system, one must run:

`schedule[ subSystem -> True ]`

or equivalently:

`structSched[]`

subSystemsSchedule

Indicates where are the schedules of the subsystems used in the system we schedule. Its type is a list of schedules (see the format in section 4.

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

Note that the scheduler automatically appends to `$scheduleLibrary` the schedules of all new systems that it schedules.

4 Format of `schedule`'s results

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 the second argument is the schedule itself.

The Last argument of `Alpha‘scheduleResult` is for internal use.

The syntax of this structure is described here.

```
<schedResult> =
  Alpha‘ScheduleResult[
    scheduleType_Integer, <sched3List>, objFunc_]

<sched3List> = { nameVar_String, 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[3].

The communication between MATHEMATICA and `pip` is done by files. These files are written in the directory indicated by the MATHEMATICA variable `$tmpDirectory`. Currently, all methods should work on Solaris, Windows and MacOS X plateforms.

6 Examples

Consider the program of Fig. 1. In the following, we give the result of the scheduler with different options.

```
system matmult : {M |M>1}
  (a,b : {i,j | 1<=i,j<=M} of real)
returns
  (c : {i,j | 1<=i,j<=M } of real);
var
  C : {i,j,k | 1<=i,j<=M; 0<=k<=M} of real;
let
  c[i,j] = C[i,j,M];
  C[i,j,k] = case
        {|k=0} : 0[];
        {|1<=k<=M} : C[i,j,k-1]+a[i,k]*b[k,j];
  esac;
tel;
```

Figure 1: Matrix multiplication

Default use

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

[3] Explain where...

```
In[65]:=
schedule[]
Checking options...
Dependence analysis...
Building LP...
LP: 92 variables, 80 Constraints
Writing file for PIP....
Solving the LP...
Total execution Time: 1+M
T_a{{i,j,M} = 0
T_b{i,j,M} = 0
T_c{i,j,M} = 1+M
T_C{i,j,k,M} = k

Out[65]=
scheduleResult[matmult,{{a,{i,j,M},sched[{0,0,
  0},0]},{b,{i,j,M},sched[{0,0,0},0]},{c,{i,j,M},sched[{0,0,1},1]},{
      C,{i,j,k,M},sched[{0,0,1,0},0]}},{1,1}]
```

The first lines indicates which computations are performed. Then the result is printed on the screen. Here, for instance the schedule is affine by variable. `c[i,j,k]` is computed at time `1+M`. The result (after `Out[65]`) is the corresponding MATHEMATICA structure assigned to `$schedule`.

Adding a constraint

Suppose that we want to impose that variable `C[i,j,k]` be schedule at time $2i + j + k + 7$ and that $\tau_a^i = \tau_b^i = \tau_c^i$. We have to add the two constraints: `"TC[i,j,k]=2i+j+k+7"`, `"TaD1==TbD2"` and `"TaD1==TcD2"`, hence the command is:

```
In[15]:= schedule[addConstraints->{"TA[i,j,k]=2i+j+k+7",
  "TaD1==TbD2", "TaD1==TcD1}]
```

The result is

```
T_a{i, j, N} = i
T_b{i, j, N} = i
T_c{i, j, N} = 8 + i + j + 2 M
T_C{i, j, k, N} = 7 + 2i + j +  k
```

Notice that in this schedule, `c` and `C` do not have the same τ^i coefficient. We may, if needed, fix this by adding the constraint `TcD1==TCD1`.

7 Troubles

No schedule is found

If no schedule is found, a message tells the user and there may be several reasons.

- No schedule exists (no way of solving this problem). You should be aware that finding a schedule for a system of recurrence equations is in the general case undecidable. Finding out an affine schedule is decidable, but an absence of affine schedule does not guarantee that there does not exist a schedule. Most often, however, the scheduler fails because there is an error in the ALPHA program which contains a dependence cycle: the only solution is to check the program (use the `analyze` command, then check your program "by hand").
- No schedule of the chosen type exists: try a multi-dimensional schedule, or try another type of schedule.
- There exists a schedule but the time is not bounded. In this case try with the option `objFunction` set to 1): this prevents the scheduler to try optimizing the total time of the algorithm.
- The program is not semantically correct: try the `analyze` function.

An awful error message

In general, when something goes wrong during the scheduling, the error is captured correctly. Sometimes, 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 the scheduler: MATHEMATICA tries then to open a file which was not closed. You can close these files (here `/tmp/mat.tmp`) by typing:

```
Close["/tmp/mat.tmp"]
```

In general it is not recommended 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*. Thèse de doctorat, Ifsic, Université de Rennes 1, December 1989.

[MQRS90] C. Mauras, P. Quinton, S. Rajopadhye, and Y. 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.