reference manual for the Alpha **Scheduler**

Tanguy Risset

April 9, 1997

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

1 Introduction

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 parallization researches. Our implementation makes use of the procedure defined in [Fea92]. Rouglhy speaking, the time is considered as a discrete single clock. the idea of the scheduling process is to gather all the constraints that must verify the schedule in a linear programming problem (LP) and to solve it with a particular software.

This document is not intended to be a comprehensive explanation of the implementation, it is an introduction to the overall architecture of the package `NewSchedule.m` which is quite big (about 3000 lignes). The developper should carefully read this before getting into the code. The source code of `NewSchedule.m` is commented in such a way that the present document document should be enough to understand everything.

2 theoretical basis

The schedule is based on the method proposed in [Fea92] and uses the farkas lemma. We give her the form of this lemma that we use for ensuring constraints

theorem 1 (Affine form of Farkas lemma) *Let $\mathcal{D}$ be a polyhedron defined by the inequalities $Ax+b \geq 0$, Then the affine form ϕ is non-negative on $\mathcal{D}$ if and only if there exists $\lambda_0, \Lambda \geq 0$ such that $\phi = \Lambda \begin{pmatrix} A & b \end{pmatrix} + \lambda_0$*

Proof: See [Sch86] .

It is easy to see how we will make use of this lemma. To ensure a linear constraint (which can always be written as the fact that a linear function is positive on a domain), we will try to find Λ, λ_0 such that the above equality is verified (we transformed a $\forall x \in ...$ in a $\exists y \geq 0 s.t....$). The method proposed in [Fea92] distinguish three type of constraints that a scheduling function T must ensure.

- **Positivity constraints**: for each variable A of the program, for any point (x, N) in the domain of A, we must have $T_A(x, N) \geq 0$.
- **Dependency constraints**: for any dependence $A[x, N] = F(\ldots, B[I(x, N)], \ldots)$, for each point (x, N) where this dependence occurs, we must have: $T_A(x, N) - T_B(I(x, N)) - \delta \geq 0$ (here δ is a delai associated to the computation of A.
- **Objective function constraints**: These constraints depend on the type of objective function choosen. If we chose to minimiz e the overall execution time, we introduced a function $f(N)$ depending only upon the parameters of the program, which will be greater than any time scheduled (i.e. for each variable A, for any point (x, N) in the domains of A, we must have $f(N) - T_A(x, N) \geq 0$).

As index functions are affine functions in ALPHA, all these constraints are of the required form: ensure that an affine function is positive on a particular domain, thus we can use the Farkas lemma. For each constraints, we will use the formulae of theorem 1 and obtain a system of equalities that will be solve with mppip.

- Positivity constraints: if a variable A is defined on the domain $D_A = \{x \mid Cx + b \geq 0\}$ (x has n components and C has k rows), the scheduling function $T_A(x) = Tx + t$ is positive on D_A if and only if the system:
$$\begin{cases} (T\ t) - \Lambda.(C\ b) - \lambda_0 = 0 \\ \lambda_i \geq 0, 0 \leq i \leq k \end{cases}$$
has a solution (where Λ is a k-vectors whose i^{th} component is λ_i). Hence we get $n+1$ equation:
  - for each dimension i, if $C_{.i}$ is the i^{th} column of C: $\tau_i - \Lambda.C_{.i} = 0$
  - $t - \Lambda.b - \lambda_0 = 0$

- dependency constraints: $A[x]$ depends upon $B[I(x)]$ where $I(x) = Dx + d$ on the domain $D_{AB} = \{x \mid Cx + b \geq 0\}$ (x has n components and C has k rows) the scheduling functions $T_A(x) = T_A x + t_A$ and $T_B(x) = T_B x + t_B$ respect the dependency if an only if the system:
$$\begin{cases} (T_A\ t_A) - (T_B\ t_B).(D\ d) - \delta_{BA} - M.(C\ b) - \mu_0 = 0 \\ \mu_i \geq 0, 0 \leq i \leq k \end{cases}$$
has a solution (where M is a k-vectors whose i^{th} component is μ_i, δ_{BA} is the delai necessary between the production of B and the production of A). Hence we get $n+1$ equation:
  - for each dimension i, $\tau_{A,i} - (T_B\ t_B).D_{.i} - M.C_{.i} = 0$
  - $t_A - (T_B\ t_B).d - \delta_{BA} - M.b - \mu_0 = 0$

- Objective function constraints: if a variable A is defined on the domain D_A with $D_A = \{(x, N) \mid A \begin{pmatrix} x \\ N \end{pmatrix} + b \geq 0\}$ (x has n components, N is the parameter vectors has m component and A has k rows), the scheduling function $T_A(x) = T_A \begin{pmatrix} x \\ N \end{pmatrix} + t_A$ lower than the objective function $\Phi.N + \phi_0$ on D_A if and only if the system:
$$\begin{cases} (0\ \Phi\ \phi_0) - (T_A\ t_A) - R.(A\ b) - \rho_0 = 0 \\ \rho_i \geq 0, 0 \leq i \leq k \end{cases}$$
has a solution (where R is a k-vectors whose i^{th} component is ρ_i). Hence we get $n+m+1$ equation:
  - for each i, $1 \leq i \leq n$: $-\tau_{A,i} - R.A_{.i} = 0$
  - for each i, $n+1 \leq i \leq n+m$: $\phi_{i-n} - \tau_{A,i} - R.A_{.i} = 0$
  - $\phi_0 - t_A - R.b - \rho_0 = 0$

3 Algorithm of the schedule function

The basic structure of the fonction schedule is the following:

1. Perform various analysis (get the list of variable, domains for each variables, dependencies, duration for each instruction ...)
2. Build the list of constraints of the linear program (three types of constraints: constraints ensuring positivity for timing function, constraints ensuring dependancies, constraints for the objective function).
3. Write the LP in a unix file in `Pip`. Solve the LP with the software `mppip`, and get the result.
4. Print the result and get the final output form.

4 Step 1: various analysis

- `addAllParameterDomain`. The syntax of parameter in Alpha allow some context constraints that are implicit in all domains of the ALPHAprogram. If we want to apply Farkas lemma, we need all the constraints of the domain, hence, we add *by hand* these constraints to all domains of the program.

- `domains of variables`. We have choosen to distinguish three types of variables: input variables, local variables and Output variables. For each variable, the domain is convexise, i.e. we take the convex hull of the real domain of the variable (this allow to simplify the resulting LP, but imply a less powerfull scheduler).
- `dependencies`. The dependence analysis is first done by the `dep[]` function. Then, the list of dependancies are classified according to the type of dependancy (6 type of dependency, in this order: Local variable to Output variable, Input to Output, Output to Local, Input to Local, Local to Local, Output to Output).
- `makeNumInstr`. This function numbers all variable. The number of a variable is its order of declaration in the Alpha program. This function build a list of couple: {`name_String,number_Integer`} that will be used in the rest of the program to make the correspondance between the name of a variable and its number.
- `makeRefline`. At this point of the program, we are able to know exactly how many variables will have the linear program generated (how many λ's, ρ's, etc...). As we will see below, we will first build a constraint matrix in which each row has the same structure. This structure is indicated by the `refline` constructed here. `refline` contains 8 sublist. The first one indicates how many coefficient there are in the objective function Φ. The second one indicates, for each variable A, the dimension of the domain of A (hence the number of τ_A's). The third one indicates how many variables will be dedicated to the constant part of the scheduling functions (the α's). The fourth indicates, for each dependancy, how many farkas coefficient (mu) will be introduced by the equation needed to ensure this dependancy. The fifth indicates, for each variable, how many farkas coefficient (λ) will be necessary to ensure the positivity condition for this variable, The sixth indicates, for each variable, how many farkas coefficient (ρ) will be necessary for the objective function constraints. the 7^{th} and 8^{th} elements are set to {`1`} and will be explained later.

  As we have classified the variables in three categories and the dependancies in 6 categories, elements $2, 3, 5, 6$ of refline are divised in three sublist and element 4 of refline is divised in 6 sublist. The precise structure of `refline` is clearly commented in the `NewSchedule.m` package. For the moment, we just need to know that this variable will be used in the rest of the program for getting information on the program (for instance: how many input variable, how many λ's related to variable A etc.).

5 Step 2: building the list of constraint

the list of constraints is built by the function `makeTableEq`. As we have explained in section 2, there are three types of constraints. The positivity constraints are set for the Input, Output and Local variables, but we impose the objective function constraint only on the output variables. the dependency constraints are built for the six type of dependencies. The constraints are the elements of the `structTableEq`. Each constraint is a list which have the following structure:

```
{ PhiCoeffs_IntList,
{ TauLocal_IntListList,TauOutput_IntListList,TauInput_IntListList},
{ AlphaLocal_IntList,AlphaOutput_IntList,AlphaInput_IntList},
{ LambdaLocToOut_IntListList,  LambdaInToOut_IntListList,
  LambdaOutToLoc_IntListList,  LambdaInToLoc_IntListList,
  LambdaLocToLoc_IntListList,  LambdaOutToOut_IntListList},
{ MuLoc_IntListList,  MuOut_IntListList,  MuIn_IntListList },
{ RhoLoc_IntListList,  RhoOut_IntListList,  RhoIn_IntListList },
{ CstCoeff_Int },
{ EqOrIneq_Int} }
```

In the above structure, `_IntList` indicates that the element considered is a list of integer (and `_IntListList`, a list of list of integer). In a constraints `c1` of this type, `PhiCoeffs` is the list of factors of the coefficients

of the objective function Φ in `c1`. One element of `TauLocal` correspond to a local variable (say A) and gives the factor (in c1) of the components of the scheduling vectors of A. One element of `LambdaLocToOut` corresponds to a particular dependency from a local variable to an output variable and gives the coefficients (in c1) of the farkas coefficient λ introduced for this dependency, and so on ....

For instance, if A is the first local variable of the Alpha program which has only one parameter (the domain of A has dimension 3), and we want to impose the constraint: $\phi_1 - 2\tau_{A,1} = 0$, this constraint we be stored in one row as:

```
{ {1,0},{{{-2,0,0},{0.... (* structured list of  0 in the remaining *) ,{0}}
```

and the real constraint can be obtained by $expr1 = (1\ 0).\begin{pmatrix}\phi_1\\ \phi_0\end{pmatrix} + (-2\ 0\ 0).\begin{pmatrix}\tau_{A,1}\\ \tau_{A,2}\\ \tau_{A,3}\end{pmatrix} + \ldots = 0.$

In `NewSchedule.m` such structured constraints are always assigned to variables which are called `line0` or `line1` (like the lines of the constraint matrix). Usually, `line0` consists of a structured constraints corresponding to 0=0 (build from the informations contained in `refline`), then it is assigned to `line1`, and the components of `line1` are iteratively modified to obtain a valid constraint as indicated in section 2. The 7^th element of line1 is a list of one single integer which is the constant that appear in the constraint. The 8^th element of `line1` is a list of one single integer which indicates if the constraints is and equality (value 0) or an inequality (value 1: $expr1 \geq 0$)

6 Step 3: Solving the LP

`fichierPip` writes the constraints of `structTableEq` matrix in the mppip format in a file, then calls `mpppip` to solve the LP and read the solution.

`fichierPip` first calls `matPip` which flatten the output of each line of `structTablEq`. At this point, the order of the coefficients of the τ's is inverted in order to minimize first the coefficients of the parameters. `matPip` outputs the matrix `matrice` Then, `writefile` writes the matrix in a file (this part is quite long for big files). During this phase we add a single parameter to the LP which is specific to `Pip` behaviour. As we want to allow some variables to be negative (the τ's for instance), we have to add a *big parameter* G and replace each occurrence of a τ by $\tau - G$. This manipulation is explained in [?], This should be transparent and the coefficient of the big parameter in the solution is not important.

After the matrix is written, we perform some unix manipulations on the file to obtain the good mppip input format. Then, we run mppip [Fea92] and the result is read by `readPipResult`,

7 Step 4: final output form

`reorderSchedResult` place back the coefficients of the parameters at the end of the scheduling vectors. `getFinalOutputForm` pretty prints the result to the screen and set the output format to the following form:

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

References

[Fea92] 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.

[Sch86] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley and Sons, New York, 1986.