Scalable and Modular Scheduling

Paul Feautrier

Paul.Feautrier@ens-lyon.fr.

ENS Lyon

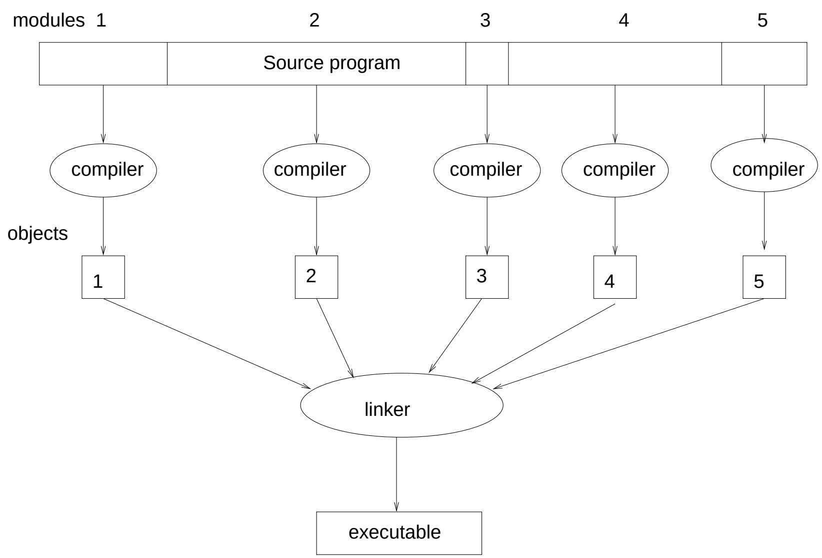
The Context

- Finding a schedule is a good way of finding parallelism in regular programs:
  - Operations (tasks) which are scheduled at the same time execute in parallel.
  - There are efficient algorithms for converting schedules into parallel programs (Quilleré, Bastoul – Cloog).
- A schedule is found by solving a linear program whose size increases roughly like $P^3 \times \ell^2$ where P is the number of statements and ℓ is the mean nesting level.

Scalability

- Since solving a LP of size n takes $O(n^3)$ (in practice), the method does not scale well.
- Observation: the constraint matrix is block sparse.
- The simplex cannot make use of sparsity: it has *fillup*.
- Find another solution algorithm.

Modularity



- Like in ordinary programs, one would like to do *separate scheduling*.
- Modules must be designed to minimize interferences.
- The compilation is necessarily incomplete. Where to stop?

Background

Program Model

A program is a way of specifying the set of tasks to be executed and the order in which they must be executed.

- Regular programs:
  - Arbitrary loop nests with affine parametric lower and upper bounds.
  - Affine array subscripts. Scalars are 0-dimensional arrays.
  - No tests, no function calls, no pointers.
- Each statement S has an iteration domain D_S which is deduced from its surrounding loops and which is a polyhedron. An iteration of S (i.e., a task) is written

$$\langle S, x \rangle, x \in D_S,$$

  where x is the *iteration vector*.

Dependences

- To each operation u one associate a schedule $\theta(u)$ which gives the start time of u. For practical and theoretical reasons, θ is chosen to be affine in the iteration vector of u.
- There is a dependence (or a precedence) from $\langle R, x\rangle$ to $\langle S, y\rangle$ iff:
  - $x \in D_R$ and $y \in D_S$.
  - $\langle R, x\rangle$ is executed before $\langle S, y\rangle$.
  - One of R and S or both modify some array A with the same subscripts:

    $$F_{AR}\,x = F_{AS}\,y,$$

    where F_{AR} and F_{AR} are subscript matrices (in homogeneous notations).
  - The conjunction of these constraints defines a dependence relation δ_{RS} which is again a polyhedron.

Scheduling Constraints

- The scheduling constraint expresses the fact that in case of a dependence, $\langle R, x\rangle$ must be executed before $\langle S, y\rangle$ in the parallel program:

$$\forall x, y : \langle R, x\rangle \; \delta_{RS} \; \langle S, y\rangle \Rightarrow \theta(R, x) + 1 \leq \theta(S, y).$$

- A similar constraint must be written for each pair of accesses to each array in the program.

The Farkas Algorithm

- Each scheduling constraint represents in fact $O(\text{Card } D_R \times \text{Card } D_S)$ linear constraints, which may be enormous or even infinite.
- Thanks to the fact that the schedules are affine, the quantifiers can be eliminated, giving a small number of constraints on the *coefficients* of x in the schedule $\theta(S, x)$. Elimination can be done either by the vertex method, or by making use of Farkas lemma.
- Let h_S be the coefficients of the schedule of S, and let $h = (h_{S_1}, \ldots, h_{S_n})^T$. The constraints can be written

$$M.h \geq b.$$

- Any solution is a valid schedule. One select a schedule with "good" properties (e.g. with the smallest coefficients).

Multidimensional Time, I

- If a program has an affine schedule, it can be executed in linear time with enough processors.
- This is not always possible, hence in some cases the scheduling constraints may be unfeasible.
- One has to use polynomial schedules, or, better, multidimensional schedules. θ is now a vector function, and $\langle R, x\rangle$ executes before $\langle S, y\rangle$ iff $\theta(R, x) \ll \theta(S, y)$ in lexicographic order.
- The dependence constraint becomes:

$$\forall x, y : \langle R, x\rangle \ \delta_{RS} \ \langle S, y\rangle \Rightarrow \theta(R, x) \ll \theta(S, y).$$

Multidimensional Time, II

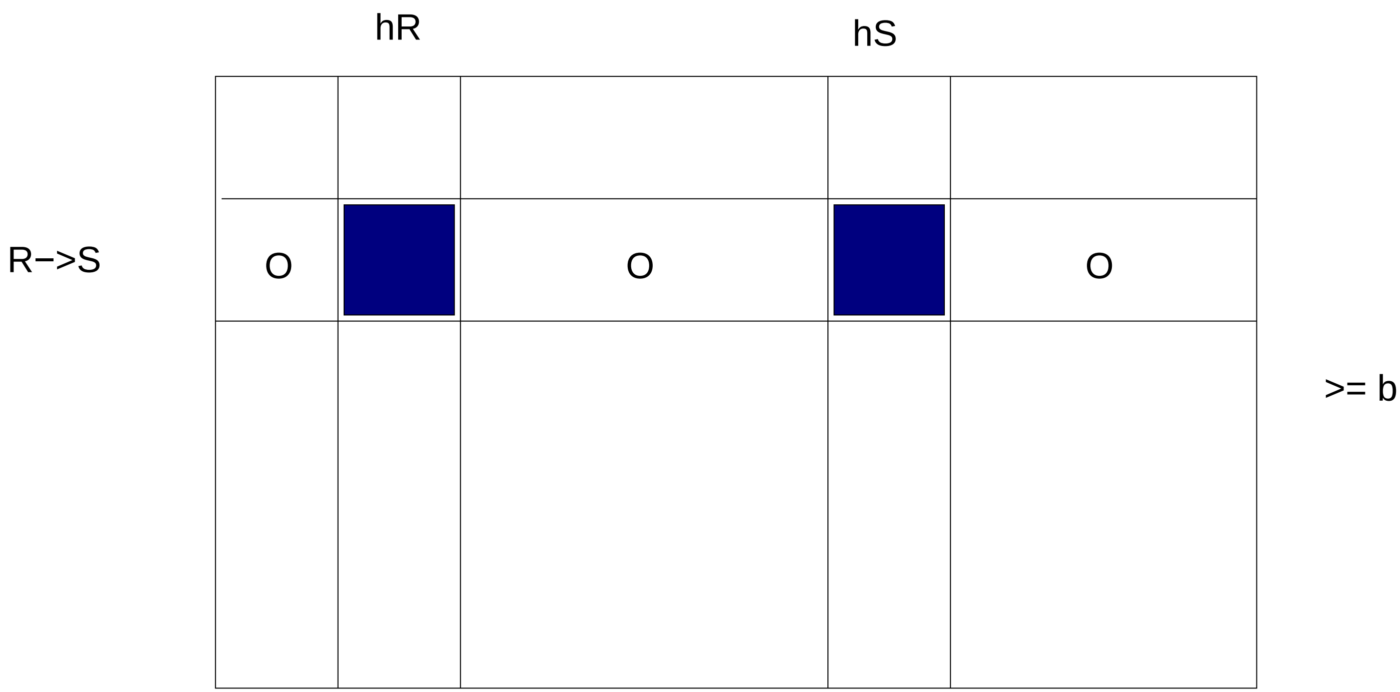
- The dependence constraint is rewritten

  $$\forall x, y : \langle R, x\rangle \ \delta_{RS} \ \langle S, y\rangle \Rightarrow \theta(R, x) + \epsilon_{RS} \leq \theta(S, y) \ 0 \leq \epsilon_{RS} \leq 1,$$

  and proceeds as before, selecting the solution which maximize $\sum \epsilon_{RS}$.
- A dependence with ϵ_{RS} is satisfied.
- If there are unsatisfied dependences, one solve a similar problem, ignoring the satisfied dependences, until all dependences are statisfied.
- One can prove that:
  - The algorithm terminates in no more than ℓ steps (ℓ the maximum nesting level);
  - The result is optimal in the asymptotic sense (F. Vivien).

Scalability

The Constraint Matrix is sparse

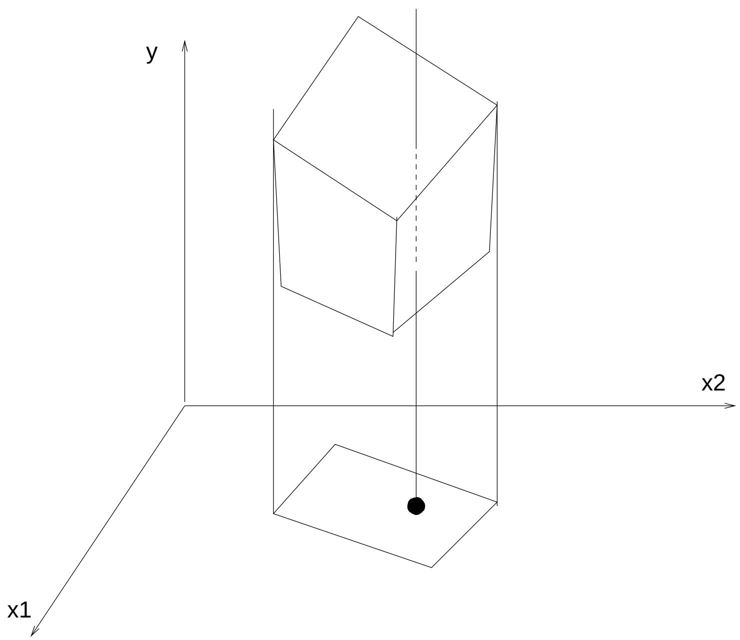


The constraint matrix is the incidence matrix of the dependence graph, if taken blockwise.

The Simplex has Fill-up

- In Gaussian elimination, on can control fill-up by proper selection of the pivot (see the work of Tarjan). The only constraint is that the pivot be non-zero.
- In the Simplex, in general, there is only one possible pivot:
  - The constant term of the pivot row must be negative.
  - The pivot must be positive.
  - The reduced pivot column must be lexicographically minimal.
- Hence, the Simplex cannot make use of the sparsity of the constraint matrix.

Projection Algorithms



- The projection of D along y is

$$P = \{x \mid \exists y : x.y \in D\}.$$

- If D is a polyhedron, so is P.
- There are many projection algorithms:
  - Fourier-Motzkin (superexponential, redundant, easy to program).
  - Pip (fast, redundant).
  - Chernikova (fast, no redundancy).
- There are backpropagation algorithms, which, given $x \in P$, find some y such that $x.y \in D$.

A Scalable Algorithm

- For each statement S:
  - Collect all the rows of M where h_S has a non-zero coefficient.
  - Eliminate h_S.
  - Remember the bounds for h_S.
- If the resulting system is trivially unfeasible ($-1 \geq 0$) stop.
- For each statement S in reverse order:
  - The bounds for h_S are constants.
  - Select a value within the bounds for h_S (e.g. the lower bound).
  - Substitute these values in all other bounds.

Choosing the Next Victim

- One can model the elimination process by a hypergraph on the statements of the program.
- There is a hyperlink on $\{R, S, T, \ldots\}$ if there is a row in M where $h_R, h_S, h_T, \ldots$ occur with non-zero coefficients.
- Initially, the hypergraph is the Dependence Graph.
- To simulate the elimination of S compute the new hyperlink $\cup_{S \in e} e - \{S\}$, add it to the hypergraph, remove all hyperlinks incident to S. This is an overestimate.
- Greedy heuristics: Select the S which generates a hyperlink of smallest size.
- There are many shortcuts.

Modularity

Modules: How and Why

- A module is a part of a program which can be *partially* compiled by itself. Traditionally, the result of partial compilation is called an *object*.
- When all modules have been compiled, another processor, the *linker* is needed to build the complete program.
- In sequential languages, a module is a function or a set of functions.
- *Systems* in ALPHA are similar to functions, with more restrictions on visibility.
- Modularity is obtained in ALPHA by surgery on the partial schedules. Some opportunities for parallelism are lost in the process.

Processes as Modules

- For parallelism, there is a more suitable kind of module: the *process*.
- A process is a toplevel object with local variables only.
- Processes communicate only throught channels.
- A channel is represented as an array which has one writer and possibly many readers. Reading is not destructive.
- Writing must have the *write once* property.
- The only constraint on reading is the *causality condition*.

Relations to KPNs

- The *send/receive* model can be simulated by introducing message counters to be used as subscripts to channel arrays.
- Message counters are induction variables. To fit in the polytope model, the induction must be solved and the result must be linear.
- The read-once and write-once conditions are automatically satisfied.
- Since reading is destructive, the system may be non-deterministic unless one enforce the *Kahn condition*: each channel must have only one reader and one writer.
- The present model is thus incomparable to the Kahn model. The bonus is that compile time analysis is possible.

Channel Clocks

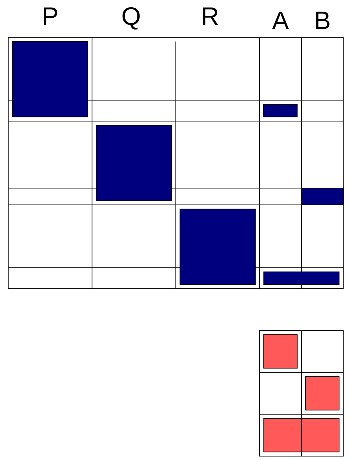
- Since output channels have the write once property, one can assign an availability date or *clock* to each cell of the channel: if x is a valid subscript for A, $A[x]$ is guaranteed to be available no later than $\theta(A, x)$.
- If $S : A[F_S x] := \cdots$ is a statement, then:

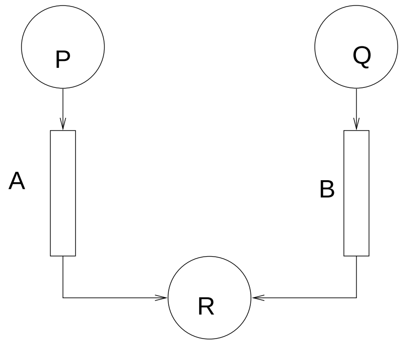
$$\theta(A, F_S x) \geq \theta(S, x) + 1.$$

- A statement $R : \cdots := \cdots A[F_R x] \cdots$ can read only available elements:

$$\theta(R, x) \geq \theta(A, F_R x).$$

The Constraint Matrix





- One can eliminate the local schedule of each process independently.
- The result is a relation between the clocks of its input and output channels (the input/ouput constraints).
- One can then interconnect the channels (i.e. identify variables in the channel clocks) and solve the global scheduling problem.
- Once the global schedule is known, one can find the local schedules by backpropagation.

Modularity as Incremental Compilation

Suppose one modifies one process. What are the consequences?

- One must redo the elimination for the modified process.
- One must solve again the global scheduling problem.
- One must redo backpropagation for all processes. This is a polynomial algorithm and there may be shortcuts.

Toward a Library Format

What is the content of a process object?

- The process statements, with their domains.
- The upper and lower bounds for the local schedules.
- The input/output constraints.

What happens for IP's, where the local schedules are fixed at implementation time? Under which conditions is the backpropagation phase stable (i.e., modifies only constant terms)?

The Multidimensional Case, I

- Let us consider the scheduling problem, before any elimination. It may not be feasible, for two reasons:
  - There is a deadlock in the system.
  - There is no affine schedule for complexity reasons.
- One can resort to the same trick as above: replace the unit delays by ϵ. After all eliminations, one get a system of constraints on the ϵ. There are three cases:
  - The all-ones solution is feasible: the system has an affine schedule.
  - The only feasible point is all zeroes: the system probably has a deadlock.
  - One can select a feasible point where some ϵ are non-zero (some dependences are satisfied). One must proceeds to compute the next component of the schedule, ignoring the satisfied dependences.
- How does this interfere with modularity?

The Multidimensional Case, II

- Modularity is preserved if all the ϵ associated to communication edges are 1. Multidimensional scheduling occurs only inside processes.
- One can prove that this is always possible if the communication graph is a DAG.
- But there are counterexamples in the general case.
- What can one do?
  - Forbid cycles in the communication graph, i.e. fuse strongly connected components in the CG, perhaps changing the semantics!
  - Waive modularity.

Conclusion: A Roadmap

- An implementation is under way.
- Quantify the compilation speed-up due to scalability.
- Explore the advantages of modularity: speed-up, reuse, process libraries.
- Investigate the problems of modular multidimensional schedules.
- Is there a way, when solving the global scheduling problem, to bound the size of the channel arrays?
- Is there a way of taking into account ressource constraints when solving the local scheduling problem?
- Code generation for processors (VLIW, SuperScalar, EPIC, DSP) is well understood (Chamsky, Quilleré, Bastoul) but is not modular. Is there a hope for a modular Cloog?
- Code generation for special purpose hardware (FPGA, ASIC).