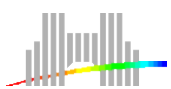
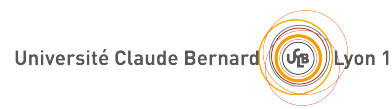
Communicating Regular Processes

Paul Feautrier

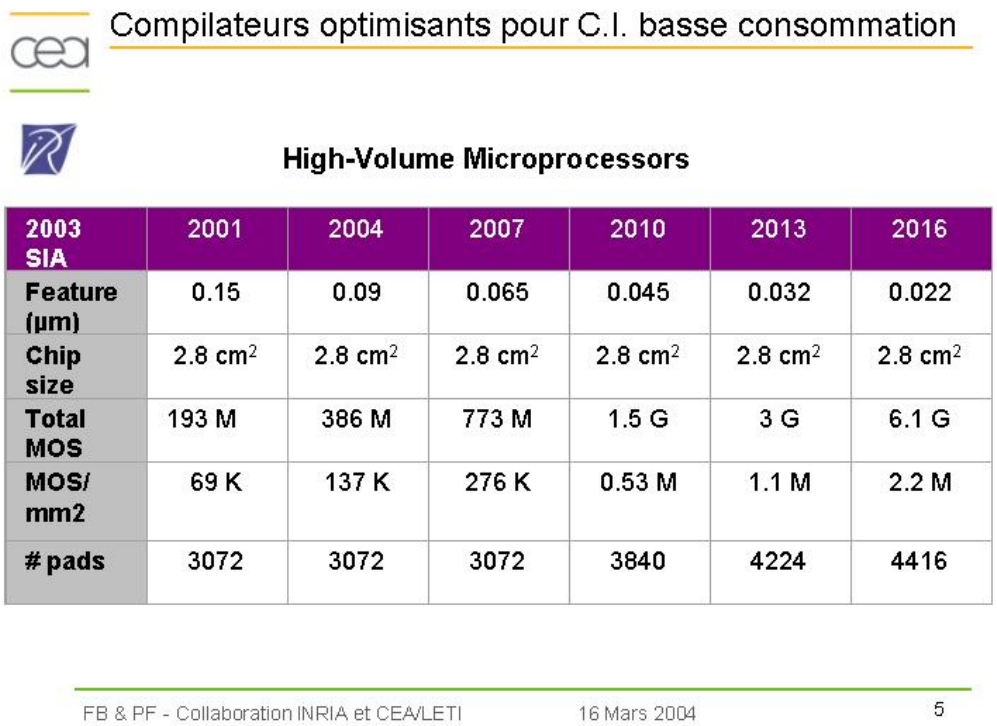
ENS de Lyon
Paul.Feautrier@ens-lyon.fr

17 juin 2009





The Challenge : Increased Parallelism



- Moore's law is slowing down, due to *fundamental* problems.
- On the other hand, there are only *practical* limitations to the chip size.
- More logic, more memory, more parallelism.

Kahn Process Networks Revisited

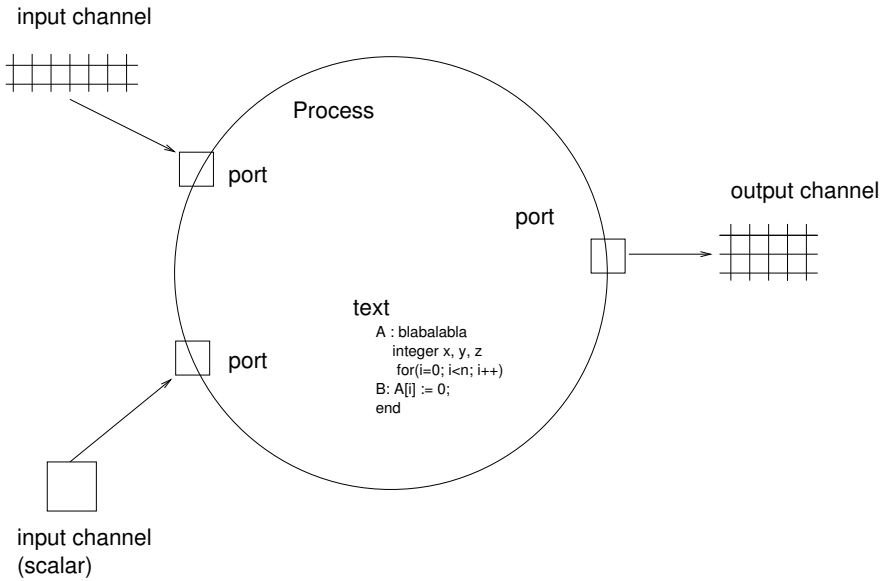
To analyze a Kahn Process Network (deadlock detection, scheduling, ...) one has to count the sends and receives on each channel and relate the operations with the same count :

```
send message n ===> receive message n
```

$$\theta(send, n) < \theta(receive, n)$$

but the message count may be a polynomial (or a more complex function) of the iteration vectors.

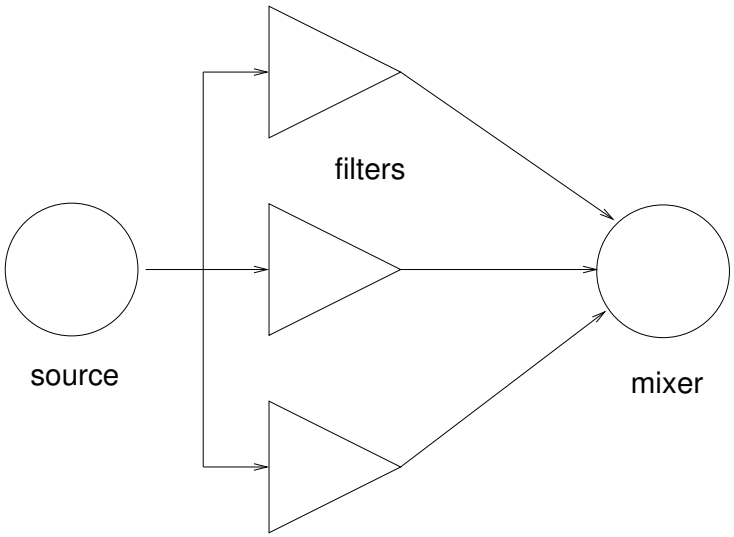
Communicating Regular Processes



CRP are similar to Kahn Process Networks but :

- Channels = unbounded shared arrays.
- Write once, read many. Implies determinism.
- A Write is non-blocking.
- A Read is blocking.
- Subscripts are affine.

The CRP language

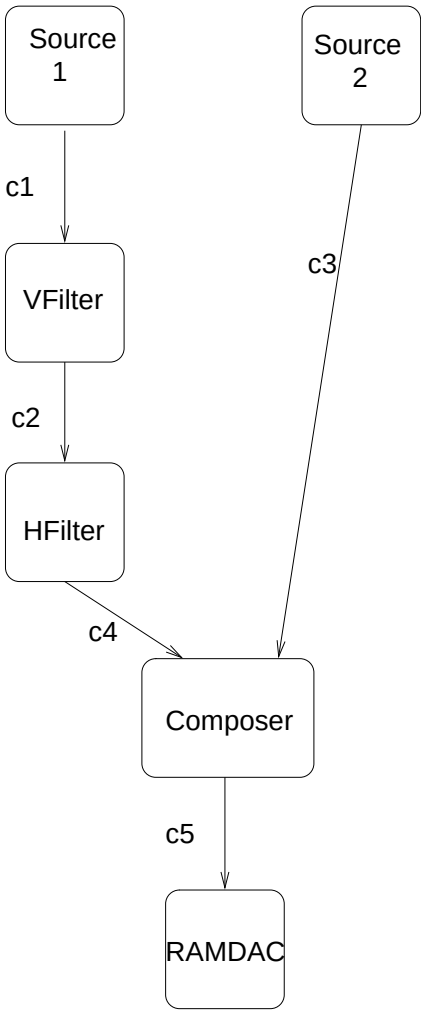


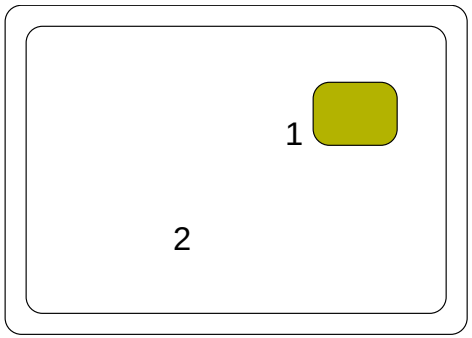

```
process filtre(inport float entree[],
               outport float sortie[],
               float poids[7]){
   int i, j;
   float s[7];

   for(i=0;;i++){
Z:   s[0] = 0.;
     for(j=1; j<7; j++)
MAC:   s[j] = s[j-1]
           + entree[i+j-1] * poids[j];
W:   sortie[i] = s[6];
   }
}
```

- Processes are executed in parallel and do not share variables.
- Communication and synchronization use ports and channels, implemented as shared write once / read many arrays (buffers).

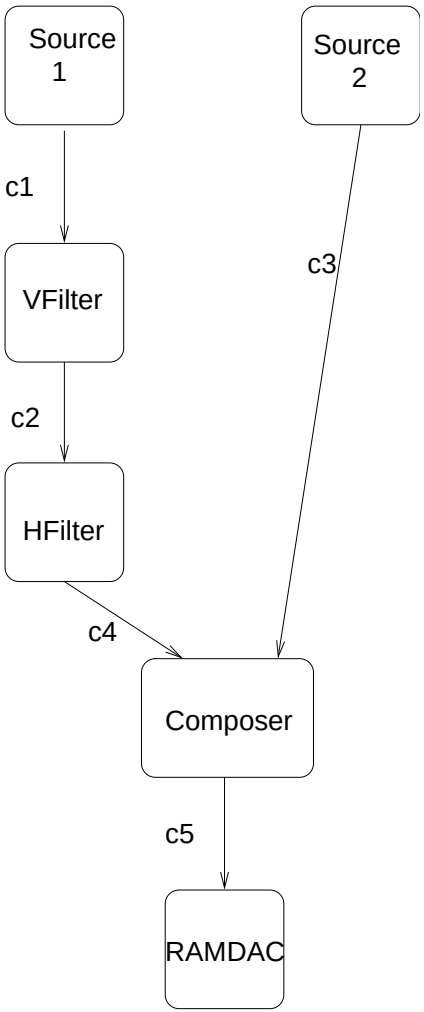
A Video Example, I





- Picture-in-picture
- Two video sources
- Source 1 is scaled down.
- Composer : for each screen pixel, select source 1 or 2.
- Ramdac : paint the screen.

A Video Example, II

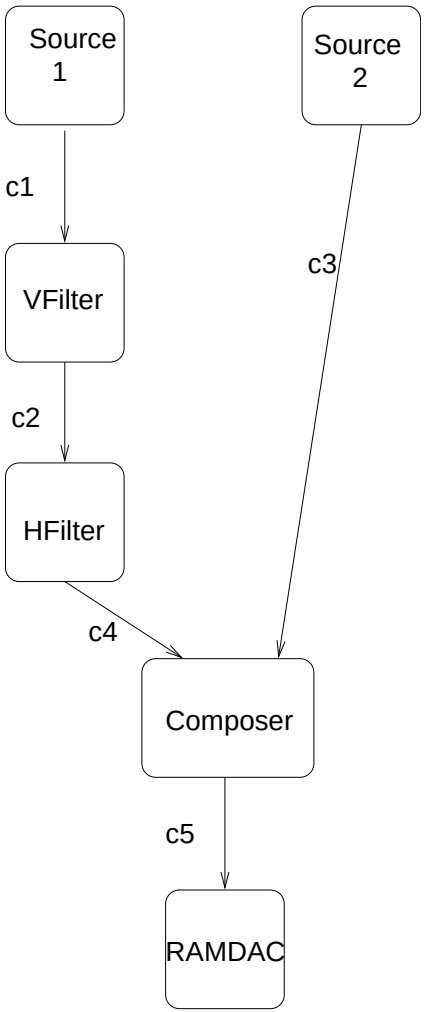


The HFilter

```
struct bigLine {
     char pixel[960];
     };
struct smallLine {
     char pixel[120];
     }
process HFilter(inport struct bigLine x[],
            outport struct smallLine y[]) {
 int i, j;

 for(i=0;;i++)
   for(j=0; j<120; j++)
     y[i].pixel[j] = x[i].pixel[8*j];
}
```

A Video Example, III

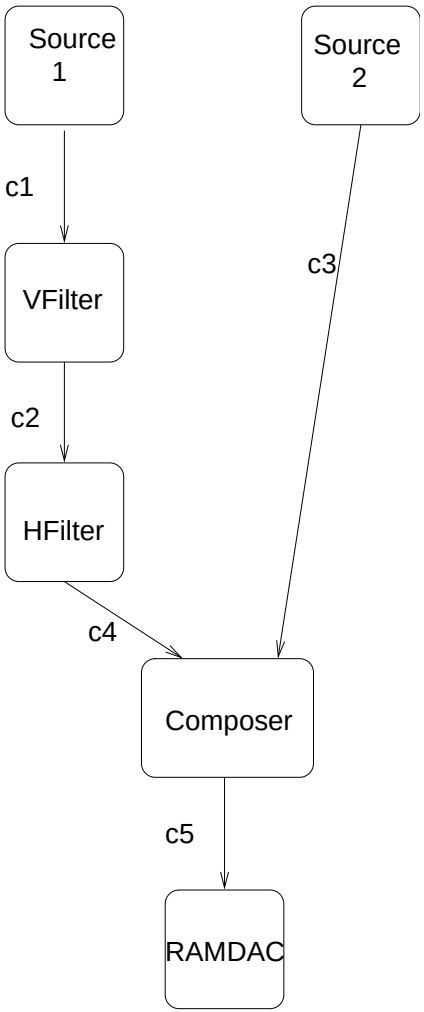


The Glue Code

```
void main(){
  channel struct bigLine c1[];
  channel struct bigLine c2[];
  channel struct bigLine c3[];
  channel struct smallLine c4[];
  channel struct bigLine c5[];

  source(c1, 1);
  source(c3, 2);
  VFilter(c1,c2);
  HFilter(c2, c4);
  composer(c3, c4, c5);
  ramdac(c5);
}
```

The Composer Code

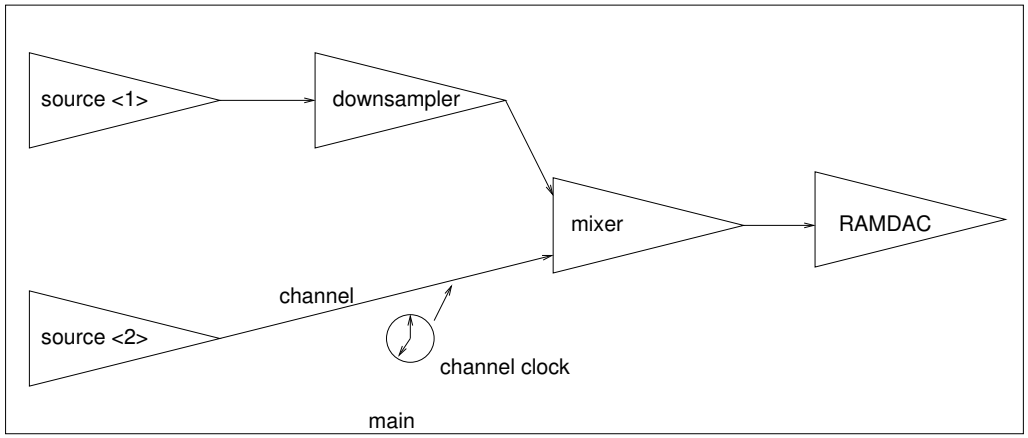



```
process composer(inport int s1[][960],
  inport int s2[][120], outport int d[][960]) {
  int i, l, c;
  for(i=0;;i++){
    for(l=0; l<90; l++)
      for(c=0; c<960; c++)
        d[960*i+l][c] = s1[960*i+l][c];
    for(l=90; l<180; l++){
      for(c=0; c<480; c++)
        y[960*i+l][c] = s1[960*i+l][c];
      for(c=480; c<600; c++)
        y[960*i+l][c] = s2[120*i+l-90][c-480];
      for(c=600; c<720; c++)
        y[960*i+l][c] = s1[960*i+l][c];
    }
    for(l=180; l<720; l++)
      for(c=0; c<960; c++)
        y[960*i+l][c] = s1[960*i+l][c];
  }
}
```

Modularity

- Scheduling is not scalable :
  - Number of dependences $\approx$ square of the size of the program ;
  - Simplex $\approx$ cube of the number of constraints.
- Modularity promote reuse.
- The trick : divide an application into *processes* with multidimensional *write-once read-many* channels (the analysis is simpler than for Kahn Process Netwoks).
- The application stays deterministics.

Modular Scheduling

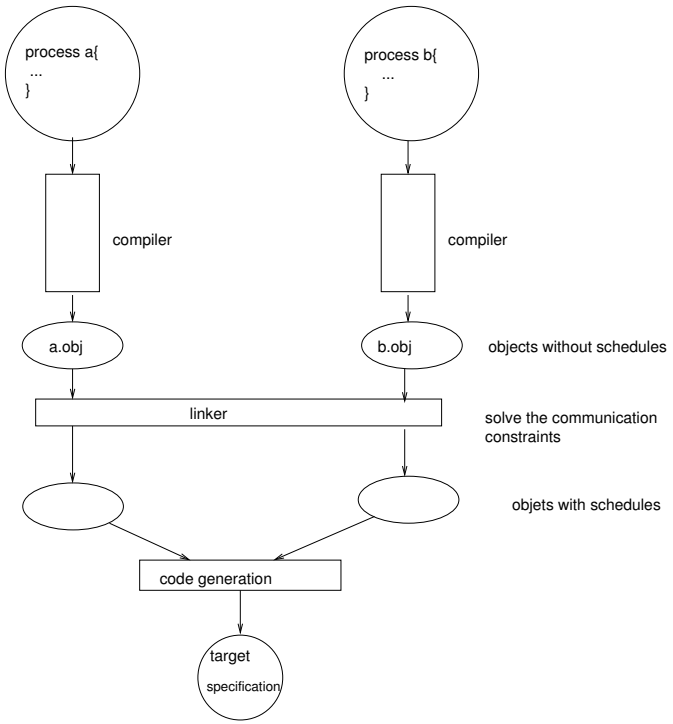


- Introduce channel clocks.
- Schedule `source`, `downsampler`, `mixer` `RAMDAC` independently, with the channel clocks as parameters.
- Schedule `main` (i.e., compute the channel clocks).
- Substitute the solution into the schedules for `downsampler` and `mixer`. The `source` processes are probably software and the `RAMDAC` is an IP.

Channel Clocks

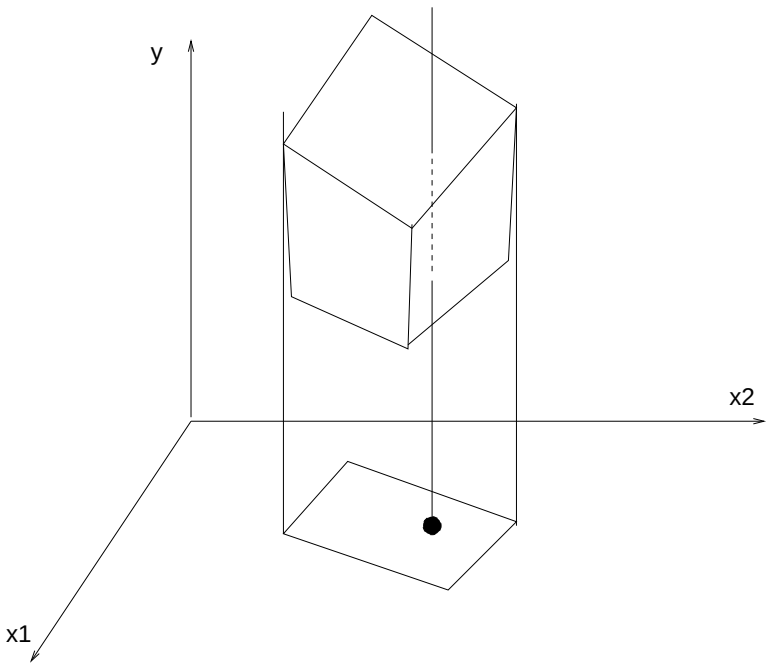
- The process schedules are not independant : they are linked by *communication dependences*.
- To restore a degree of independence, one introduces *channel clocks*. A being a channel, $\theta(A, x)$ is the date at which $A[x]$ is guaranteed to be available.
- A process schedule now depends only on the clocks of its incoming and outgoing ports.

Modular Scheduling



- Local scheduling for each process, communication schedules being kept as parameters
- Communication scheduling
- Back substitution into local schedules
- The complexity of scheduling becomes almost linear in the number of processes

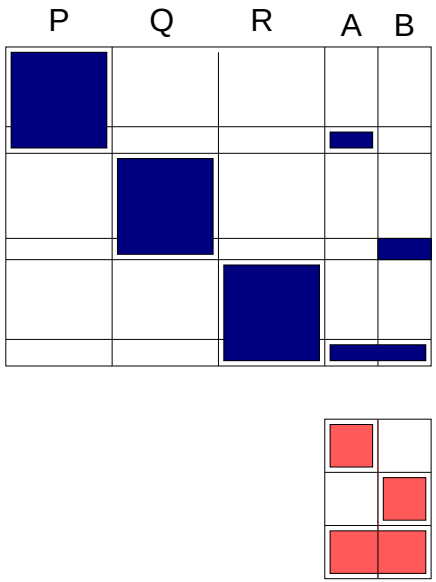
The Projection Method

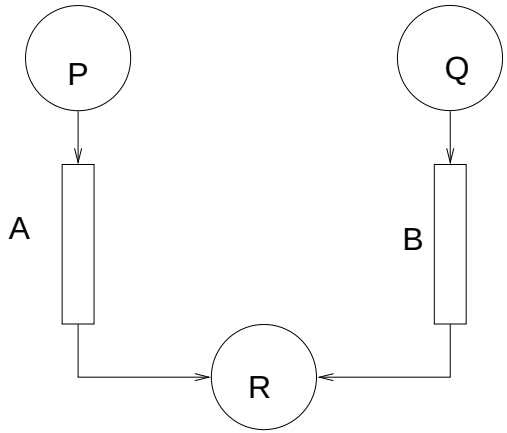


The trick : eliminate all inner schedules and get constraints involving only channel clocks.

- The projection of a polyhedron is a polyhedron.
- There are many projection algorithms :
  - Fourier-Motzkin (superexponential, redundant, easy to program).
  - Pip (fast, redundant).
  - Chernikova (fast, no redundancy).

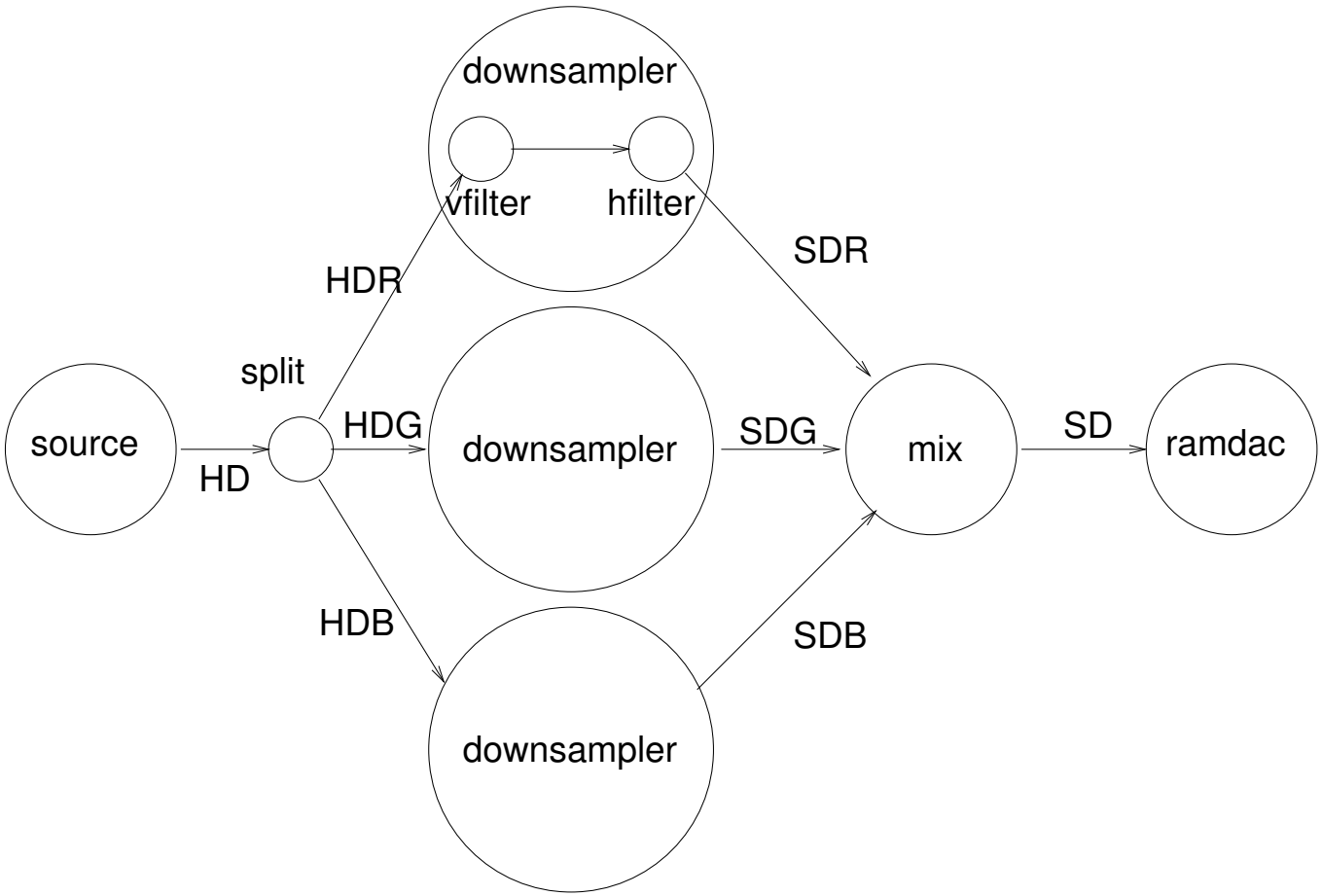
Modular Scheduling





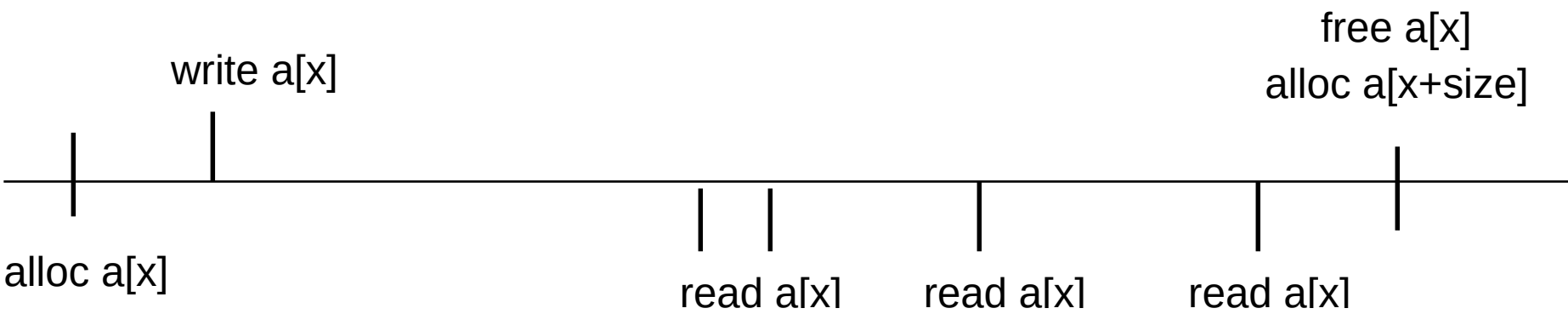
- One can eliminate the local schedule of each process independently.
- The result is a relation between the clocks of its input and output ports (the input/ouput constraints).
- One can then interconnect the channels (i.e. identify variables in the channel clocks) and solve the global scheduling problem.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

Structured Scheduling



Resource constraints, I/III

Buffer size



$$\begin{aligned} \theta(\text{write } a[x]) &\geq \theta(\text{alloc } a[x]), \\ \theta(\text{read } a[x]) &> \theta(\text{write } a[x]), \\ \theta(\text{free } a[x]) = \theta(\text{alloc } a[x + \text{size}]) &\geq \theta(\text{read } a[x]). \end{aligned}$$

Apply Farkas and solve.

Resource constraints II/III

Objective : bounded parallelism.
Exemple :

```
for(i=0; i<n; i++)
S:  a[i] = 0;
```

- Schedule $\theta(S, i) = 0$. Degree of parallelism n, too much.
- Schedule $\theta(S, i) = i \div 4$. Degree of parallelism 4, OK.

Method : Add a virtual dependence $\langle S, 0\rangle \longrightarrow \langle S, 4\rangle$.
Question : How to infer virtual dependences ?

Resource constraints, III/III

Architecture exploration

- Assumption : the iteration domain is *fat*.
- Bounded parallelism implies that the schedule is full dimensional.

$$\theta(S, t) = (T_s i + a_S) \div D_S,$$

  where T_S is of full row rank and D_S is a vector of integers.
- An estimate of the degree of parallelism is $X_S = |D_S|/|T_S|$, even if a fraction.
- Statements in the same scc of the dependence graph cannot be separated.
- Explore possible values for X_S under the constraints :

$$\sum_{S \in r, S \in H} X_S \leq N_r$$

  where H is an scc, r is a resource type with N_r instances.
- Question : how does one construct a schedule with a given determinant ?

Allocation functions

Rationale : it is useless to assign two dependent operations to different resources, since they are never executed at the same time :

$$u \ \delta \ v \Rightarrow \alpha(u) = \alpha(v). \tag{1}$$

- It is in general not possible to satisfy (1) everywhere with a non-zero α.
- The allocation function should at least respect dependences :

$$u \ \delta \ v \Rightarrow \alpha(u) \leq \alpha(v). \tag{2}$$

- It is then suitable as the first row of the schedule and can be tiled.
- Solves the matrix-vector multiply example.

Delay

- The delay (number of clock ticks) from a write to the last read is a bound on the number of writable memory cells ...
- ... and hence on the degree of parallelism.
- One can construct bounded delay schedules, but the delay is meaningful only in the one dimensional case.
- Solves the loop fusion and reversi examples.

Code Generation

Reconstructing a loop program from a schedule.

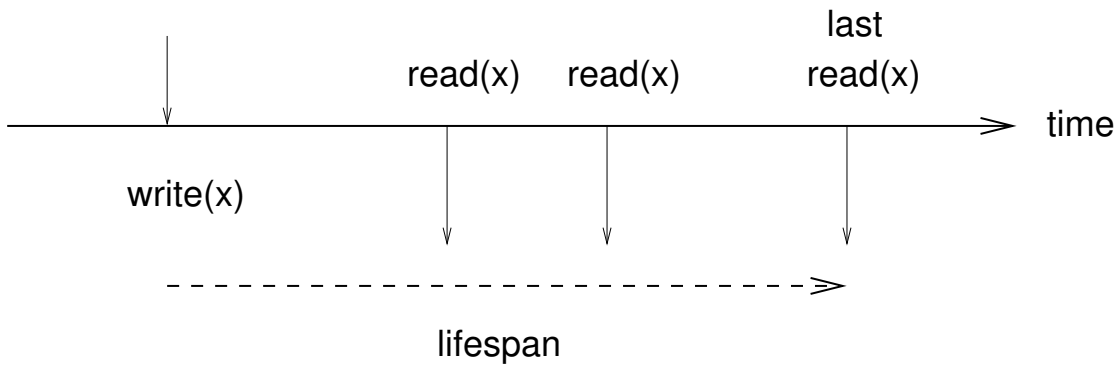
- Easy in theory. One just has to invert the schedule :

$$\begin{array}{rcl} \mathrm{do} & t = & 0, L \\ & \mathrm{doall} & \{u \mid \theta(u) = t\} \end{array}$$

- Difficult in practice. One must avoid complex control structures, which may offset the advantages of optimization or parallelization.
- There are good stand-alone implementations : [Pugh, Quilleré, Bastoul].

Memory Management

From a schedule, one can deduce the *lifespan* of each variable or value.



- Lifespans ⇒ Interference graph ⇒ Graph Coloring.
- Explicit graph coloring for registers and arrays as a whole, symbolic graph coloring (modulo allocation, A. Darte) to reduce the size of arrays.
- Unsolved problem : can one reverse the process, and build a schedule under memory size constraints ?

Target Code Generation

- A loop program is OK for embedded processors – it can be directly compiled for DSP or VLIW architectures.
- For an ASIC or FPGA, one has to construct the control FSM and the datapath, or generate directly an RTL specification.
- The RTL specification can be deduced from the loop program, but it might be better to construct it directly from the schedule.
- The description may have a varying degree of details, from plain RTL to CABA and more.

Conclusion

- A design can be divided in many processes, which can be reused elsewhere.
- Each process can be scheduled independently. The result is a set of constraints on its port clocks.
- A linker then solve the communication constraints, finalize the process constraints, and generate the object code.
- No recompilation is needed for unmodified processes.
- One can add constraints on the size of channel buffers.

Future Work

- An implementation is under way.

Future Work

- An implementation is under way.
- Explore the advantages of modularity : speed-up, reuse, process libraries.

Future Work

- An implementation is under way.
- Explore the advantages of modularity : speed-up, reuse, process libraries.
- Is there a way of taking into account ressource constraints when solving the local scheduling problem ?

Future Work

- An implementation is under way.
- Explore the advantages of modularity : speed-up, reuse, process libraries.
- Is there a way of taking into account ressource constraints when solving the local scheduling problem ?
- Code generation for special purpose hardware (FPGA, ASIC).

Restrictions à CRP

- L'analyseur syntaxique est incomplet.
  - Pas de structures.
  - Pas de tests ni d'expressions conditionelles.
  - Rien sur les pointeurs.
- L'ordonnanceur est incomplet. Outre les défauts ci-dessus :
  - Traitement imprécis des paramètres de structure.
  - Pas de gestion de la mémoire et des ressources.
  - Ordonnancements multi-dimensionnels et gestion des latences.

Tests

- Le traitement des tests est indispensable en synthèse.
- Instruction conditionnelle = test dans l'automate.
- Expression conditionnelle = test "cablé"
- Prendre en compte les dépendances de contrôle = "if conversion".
- Ne pas introduire de fausses dépendances entre branches opposées d'un test = ordre textuel partiel.

QUESTIONS ?