The Polyhedral Model

Patrice Quinton

ENS Rennes

MVDE@SEI Summer School 2014

Outline

Context

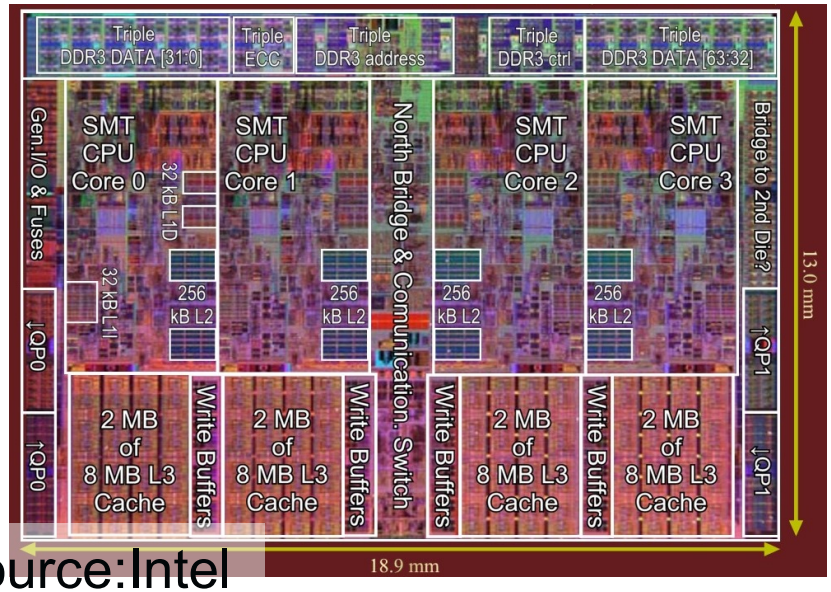
Embedded Systems

- Embedded systems are everywhere...
- Performance needed (speed, area, power consumption)
- Powerful methods to fulfill the "design gap"
- Parallelism makes things more complex

Outline

Multi-core Processor Architectures

- Nehalem : Intel Core i7
  - Four processor core + shared L3 cache with coherency



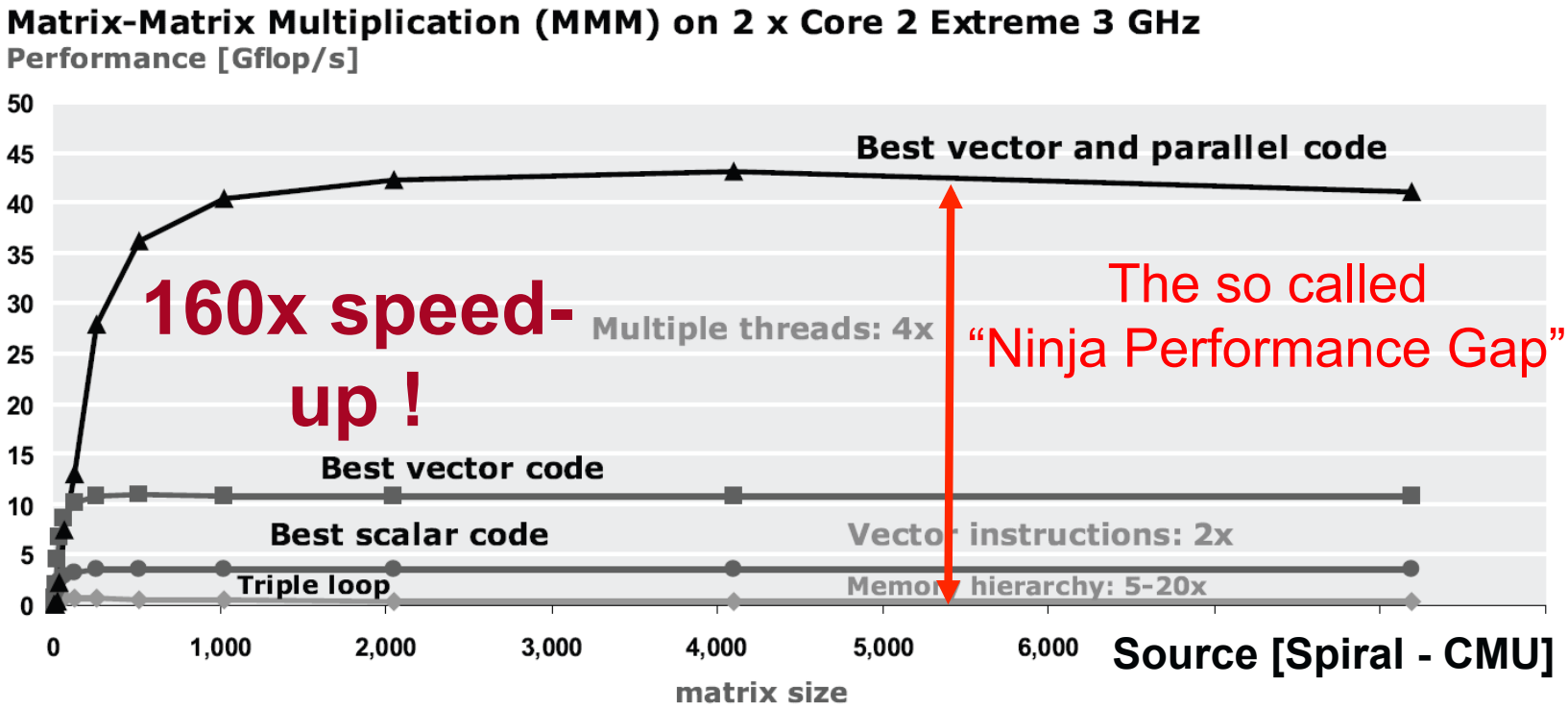
Source:Intel

- Simultaneous Multithreading (2 threads/ core)
- SIMD instruction set with 128 bits registers (SSE4)

- Main programming model is thread level parallelism
  - Using openMP, pthreads, …
  - SIMD is handled by the compiler back-end

Program Optimizations & Performance

- Impact of optimizations on performance

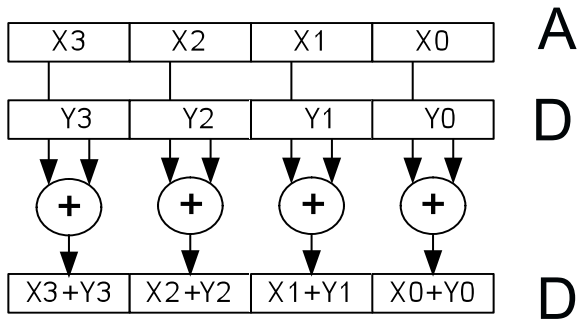


- Origin of improvements
  - Parallelism (thread x SIMD): 8x - Memory optimization: 5x-20x !

SIMD Short-Width Vector Instructions

- Expose vector level parallelism in the ISA
  - Initially for regular (8bits, 16bits data) multimedia kernels
  - Extended to support floating point (Intel SSE, AVX)
  - Very challenging for compilers !
- Example from SSE : **`ADDPS xmm1, xmm2/m128`**
  - **`m128`** : 16 bytes aligned memory location,
  - **`xmm0-7`** : 128 bit SSE registers
- Operation

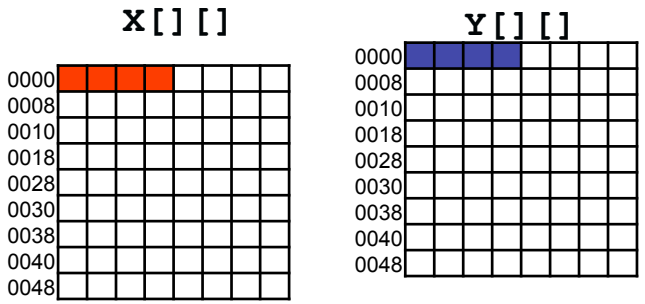
```
D[31-0]   :=D[31-0]   +A[31-0];
D[63-32]  :=D[63-32]  +A[63-32];
D[95-64]  :=D[95-64]  +A[95-64];
D[127-96]:=D[127-96]+A[127-96];
```



SIMD Instructions : Layout Constraints

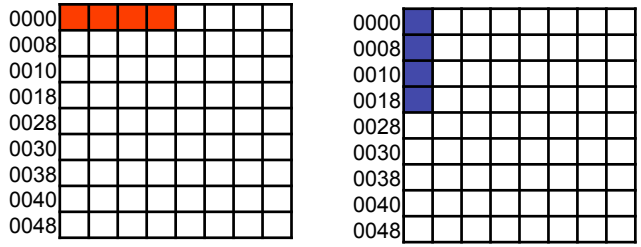
- SIMD memory access = only contiguous data in memory
  - Unaligned accesses (64/128 bits) are not supported or cause performance penalties

```
for(i=0;i<8;i++) {
  for(j=0;j<8;j++) {
    X[8*i+j]+=A[j]*Y[8*i+j];
  }
}
```



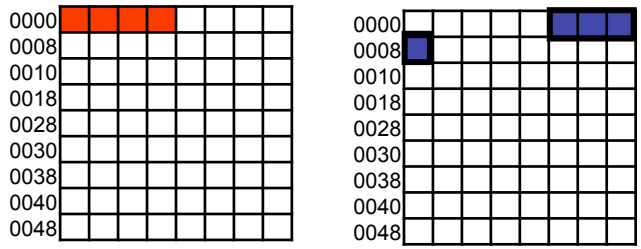
Efficient SIMD vectorization

```
for(i=0;i<8;i++) {
  for(j=0;j<8;j++) {
    X[8*i+j]+=A[j]*Y[8*j+i];
  }
}
```



No SIMD because of the Y[j][i] **non contiguous** access pattern

```
for(i=0;i<8;i++) {
  for(j=0;j<8;j++) {
    X[8*i+j]+=A[j]*Y[8*i+j+5];
  }
}
```

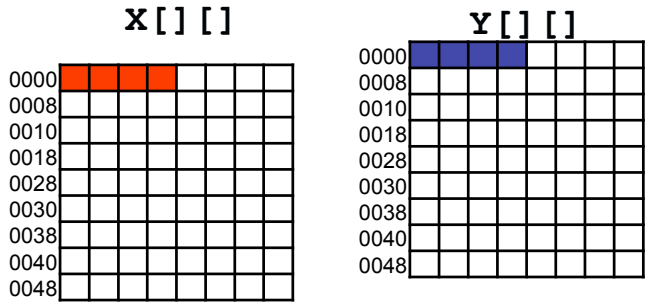


Inefficient vectorization (unaligned access)

SIMD Instructions : Layout Constraints

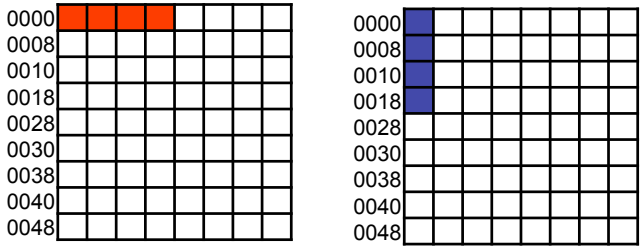
- SIMD memory access = only contiguous data in memory
  - Unaligned accesses (64/128 bits) are not supported or cause performance penalties

```
for(i=0;i<8;i++) {
  for(j=0;j<8;j++) {
    X[8*i+j]+=A[j]*Y[8*i+j];
  }
}
```



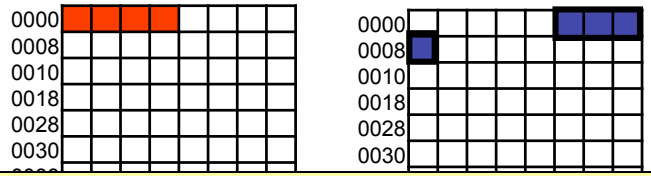
Efficient SIMD vectorization

```
for(i=0;i<8;i++) {
  for(j=0;j<8;j++) {
    X[8*i+j]+=A[j]*Y[8*j+i];
  }
}
```



No SIMD because of the Y[j][i] **non contiguous** access pattern

```
for(i=0;i<8;i++) {
  for(j=0;j<8;j++) {
    X[8*i+j]+=A[j]*Y[8*i+j+5];
  }
}
```



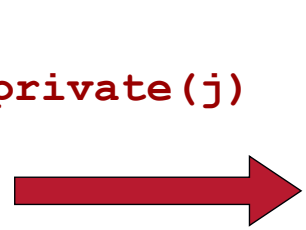
Inefficient vectorization (unaligned access)

How to transform loops (and possibly data organization) to enable efficient SIMD vectorization ?

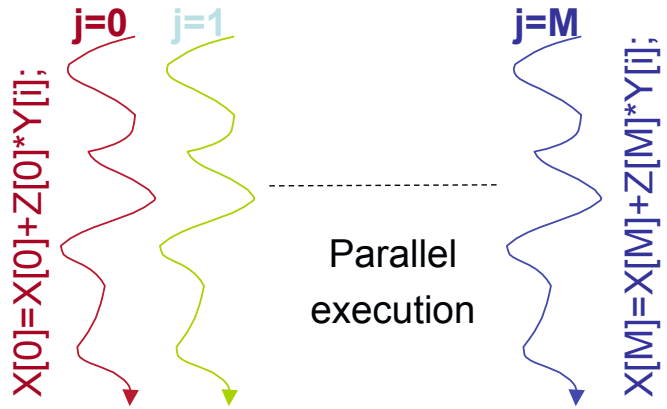
Thread Level Parallelism

- OpenMP = simple way to expose thread level parallelism
  - Through coMPIler directives in the user source code (#pragma)
  - Targeted toward shared memory machine models
- Example : **`#pragma omp parallel for`**
  - Every j iteration can be executed by its own thread.
  - Threads synchronize at the end of the loop.

```
for(i=0;i<N;i++) {
#pragma omp parallel for private(j)
  for(j=0;j<M;j++) {
    X[j]=X[j]+Z[j]*Y[i];
  }
}
```



```
for(i=0;i<N;i++) {
```

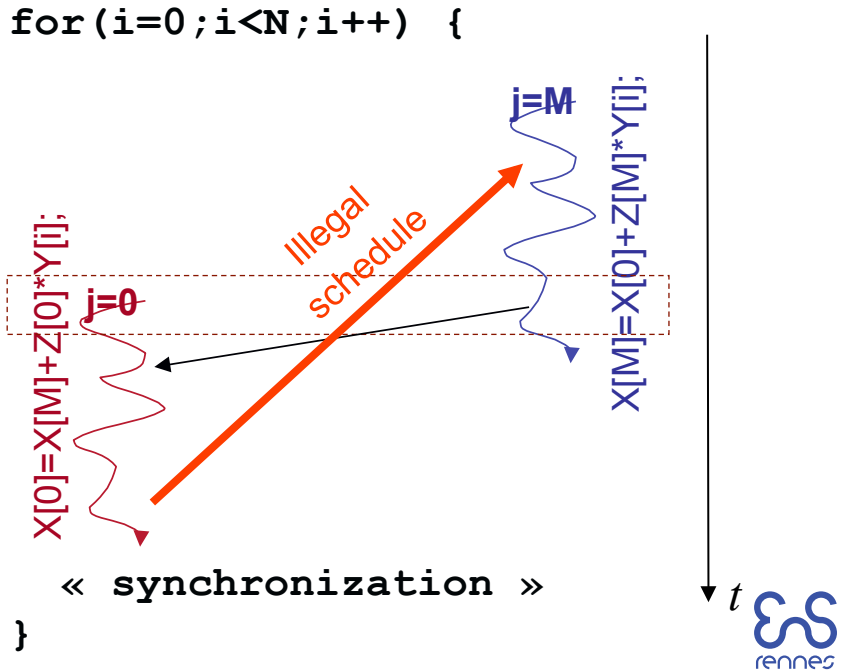



```
  « sync »
}
```

Data Race Issues in TLP

- The relative execution order of threads is not known
  - Dynamically determined by the OS scheduler
- The program execution may exhibit “data races”
  - When thread *x* reads a memory cell written by thread *y*
  - *Read can happen before write (or the other way round)*

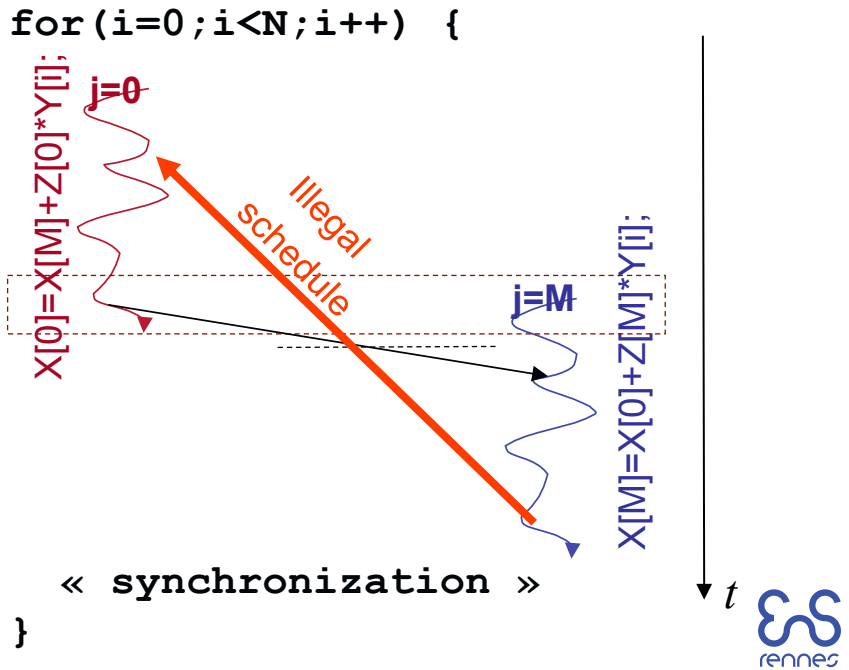
```
for(i=0;i<N;i++) {
#pragma omp parallel for private(j)
  for(j=0;j<M;j++) {
    X[j]=X[M-j]+Z[j]*Y[i];
  }
}
```



Data Race Issues in TLP

- The relative execution order of threads is not known
  - Dynamically determined by the OS scheduler
- The program execution may exhibit “data races”
  - When thread *x* reads a memory cell written by thread *y*
  - *Read can happen before write (or the other way round)*

```
for(i=0;i<N;i++) {
#pragma omp parallel for private(j)
  for(j=0;j<M;j++) {
    X[j]=X[M-j]+Z[j]*Y[i];
  }
}
```



Synchronization cost in TLP

- The runtime forks threads and wait till their completion
  - This has obvious performance overhead.

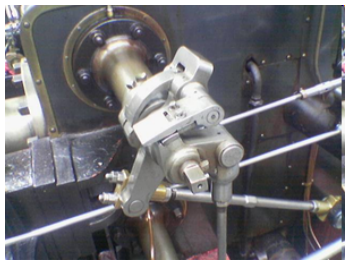
```
for(i=0;i<N;i++) {
#pragma omp parallel for private(j)
  for(j=0;j<4;j++) {
    X[j]=X[j]+Z[j]*Y[i];
  }
}
```

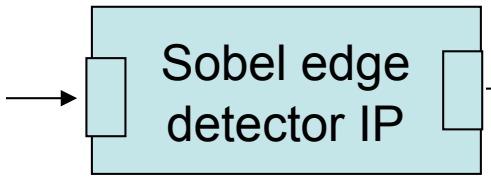
The thread parallel version is very likely to be slower than the sequential one

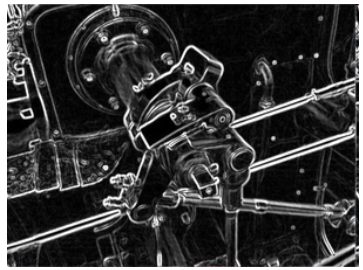
- Need to expose « coarser grain » parallelism.
  - Minimize the frequency of synchronization operations
  - Partition the computations in ***large*** independent “chunks”.
  - Pay attention to memory hierarchy (spatial/temporal locality)

High level synthesis

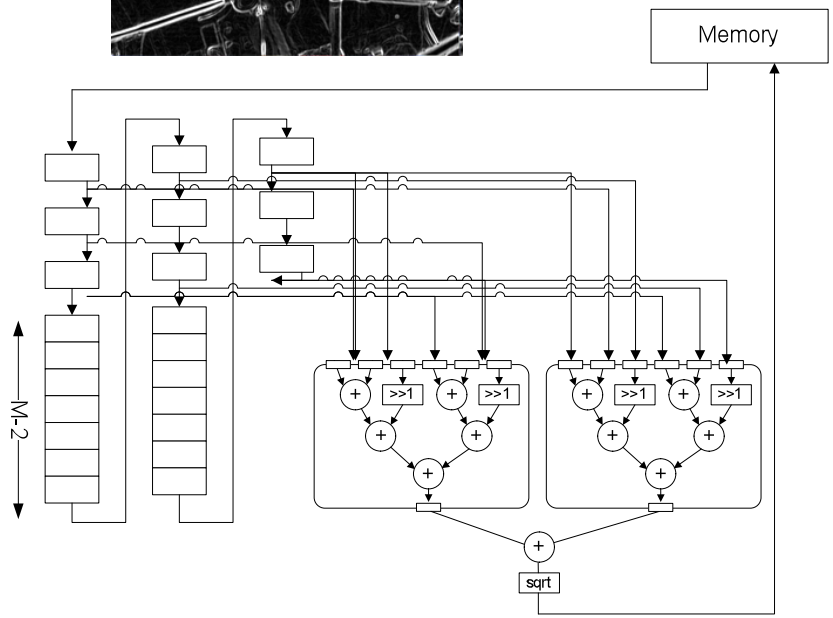
- Generating custom hardware from C/C++
  - HLS tools help boosting designers productivity by up to 5x-10x !





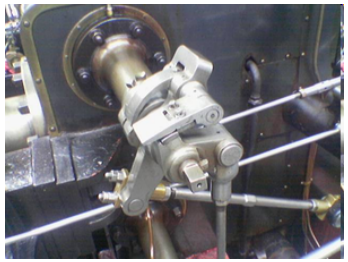


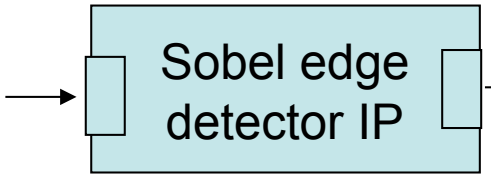
```
void image(char in[M][N], char out[M][N]) {
  for(int i=1;i<N-1;i++) {
    for(int j=0;j<M;j++) {
S0:     Gx=in[i][j+1]+2*in[i-1][j+1]+in[i+1][j+1]+
           in[i][j-1]+2*in[i-1][j-1]+in[i+1][j-1];
S1:     Gy=in[i+1][j-1]+2*in[i+1][j]+in[i+1][j-1]+
           in[i-1][j-1]+2*in[i-1][j]+in[i-1][j-1];
S2:     out[i][j]= sqrt(Gx*Gy);
    }
  }
}
```

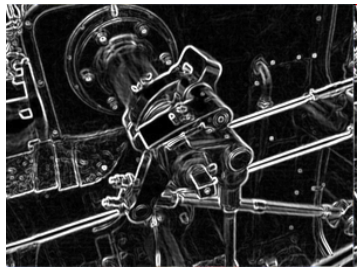


High level synthesis

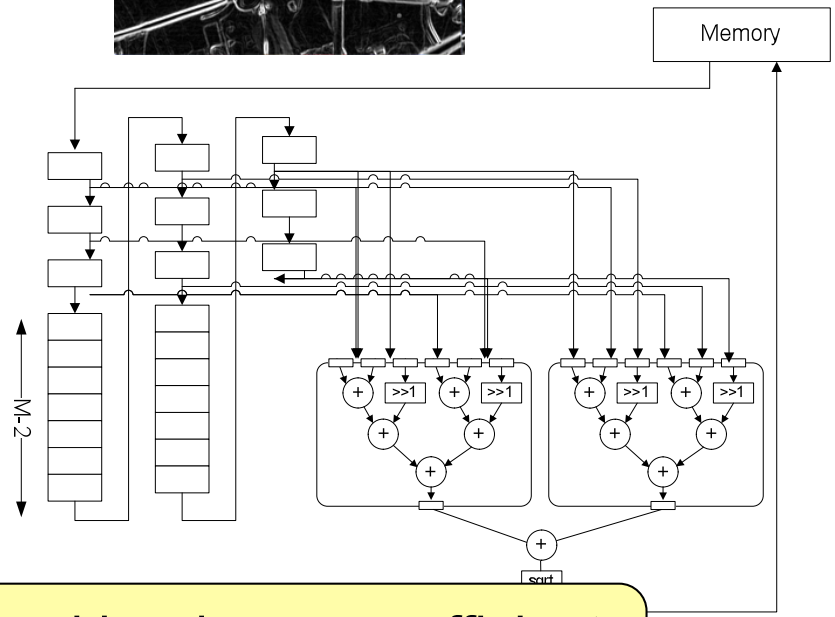
- Generating custom hardware from C/C++
  - HLS tools help boosting designers productivity by up to 5x-10x !







```
void image(char in[M][N], char out[M][N]) {
  for(int i=1;i<N-1;i++) {
    for(int j=0;j<M;j++) {
S0:    Gx=in[i][j+1]+2*in[i-1][j+1]+in[i+1][j+1]+
          in[i][j-1]+2*in[i-1][j-1]+in[i+1][j-1];
S1:    Gy=in[i+1][j-1]+2*in[i+1][j]+in[i+1][j-1]+
          in[i-1][j-1]+2*in[i-1][j]+in[i-1][j-1];
S2:    out[i][j]= sqrt(Gx*Gy);
    }
  }
}
```



How to make automatically synthesized hardware as efficient as manually designed circuits ?

What is the Polyhedral Model ?

- Representation of "regular computations" (loops)
- Basic idea : loops are functions on polyhedral set of points
- Instead of analyzing each instruction separately, consider groups of instructions related to polyhedral domains
- Use powerful mathematical techniques known on polyhedra

Organization of this Talk

- Introduction to the Model through Examples
- Representing Loops with the Alpha Functional Language
- Modularity and its Applications
- Data-flow Systems
- Conclusion

Outline

Matrix Multiplication

The Problem

c, a et b, order $n = 3$ square matrices.

$$c(i,j) = \sum_{k=1}^{3} a(i,k) \times b(k,j),$$

Loop Nest

Matrix Multiplication as an Imperative Program

```
for i := 1 to n do
    for j  := 1 to n do
        c[i,j] :=0 ;
        for k  := 1 to n do
            c[i,j] := c[i,j] + a[i,k] * b[k,j] ;
        endo
    endo
endo
```

Loop Nest

Matrix Multiplication as a Single-Assignment Program

```
for i := 1 to n do
    for j  := 1 to n do
        C[i,j,0] :=0 ;
        for k  := 1 to n do
            C[i,j,k] := C[i,j,k-1] + a[i,k] * b[k,j] ;
        endo
    endo
endo
```

Functional Expression

Matrix Multiplication as Recurrence Equations

$$\begin{aligned} 1 \leq i,j \leq n, k = 0 &\rightarrow C(i,j,k) = 0 \\ 1 \leq i,j \leq n, 1 \leq k \leq n &\rightarrow C(i,j,k-1) + a(i,k) \times b(k,j) \\ 1 \leq i,j \leq n &\rightarrow c(i,j) = C(i,j,n). \end{aligned}$$

Polyhedra everywhere !

$$\{i,j,k \mid 1 \leq i \leq n \wedge 1 \leq j \leq n \wedge 1 \leq k \leq n\}$$

Static Control Loop Nest

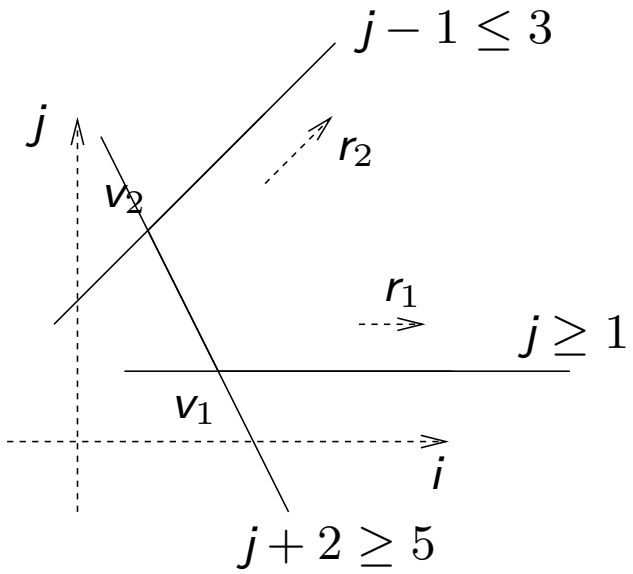
Definition

- Loop Bounds are affine functions of indices (i, j, k)
- Array References are affine functions of indices (i, j, k)
- One also may use parameters (n)

Why Consider Static Control Loop Nests ?

- Program Analysis : one can say "something" about program's behaviour
- Compiling : analysis of cache behaviour for example
- Program parallelization : where is the potential parallelism ?

A Convex Polyhedron



$$P = \{(i,j) \mid j \geq 1, j+2i \geq 5, j-i \leq 3\}$$

Dual Representation of Polyhedra

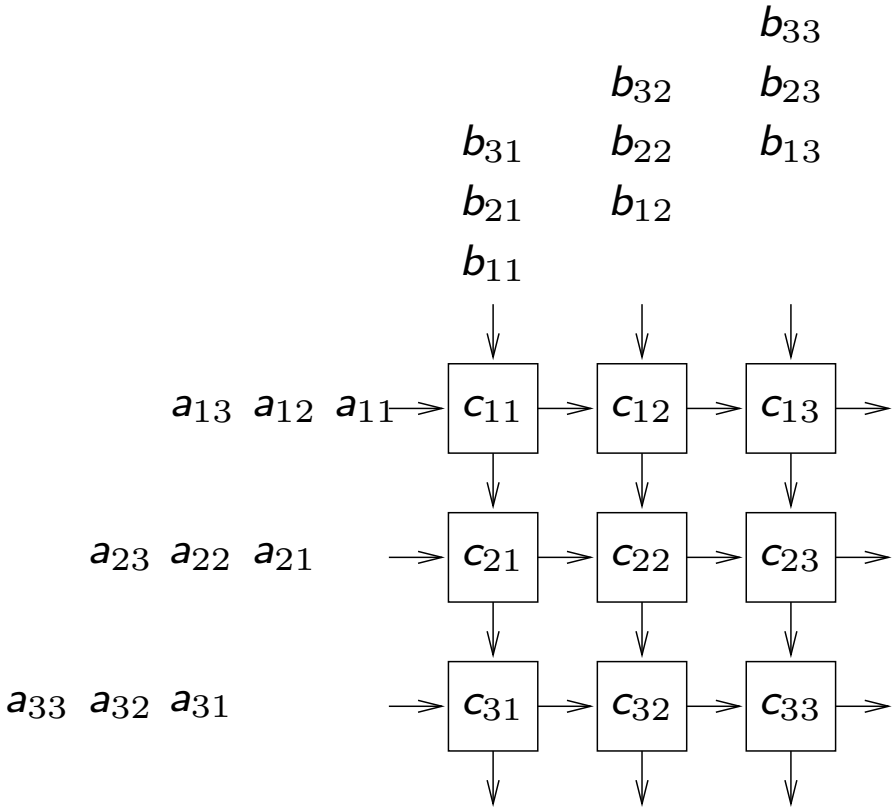
v_1, v_2 : vertices ; r_1, r_2 : extremal rays.

$$\begin{aligned} P &= \{(i,j) \mid j \geq 1, j+2i \geq 5, j-i \leq 3\} \\ &= \{x \mid x = \sum_{0 \leq a \leq 1} (av_1 + (1-a)v_2) + \sum_{b,c \geq 0} (br_1 + cr_2)\} \end{aligned}$$

A Few Properties of Polyhedra

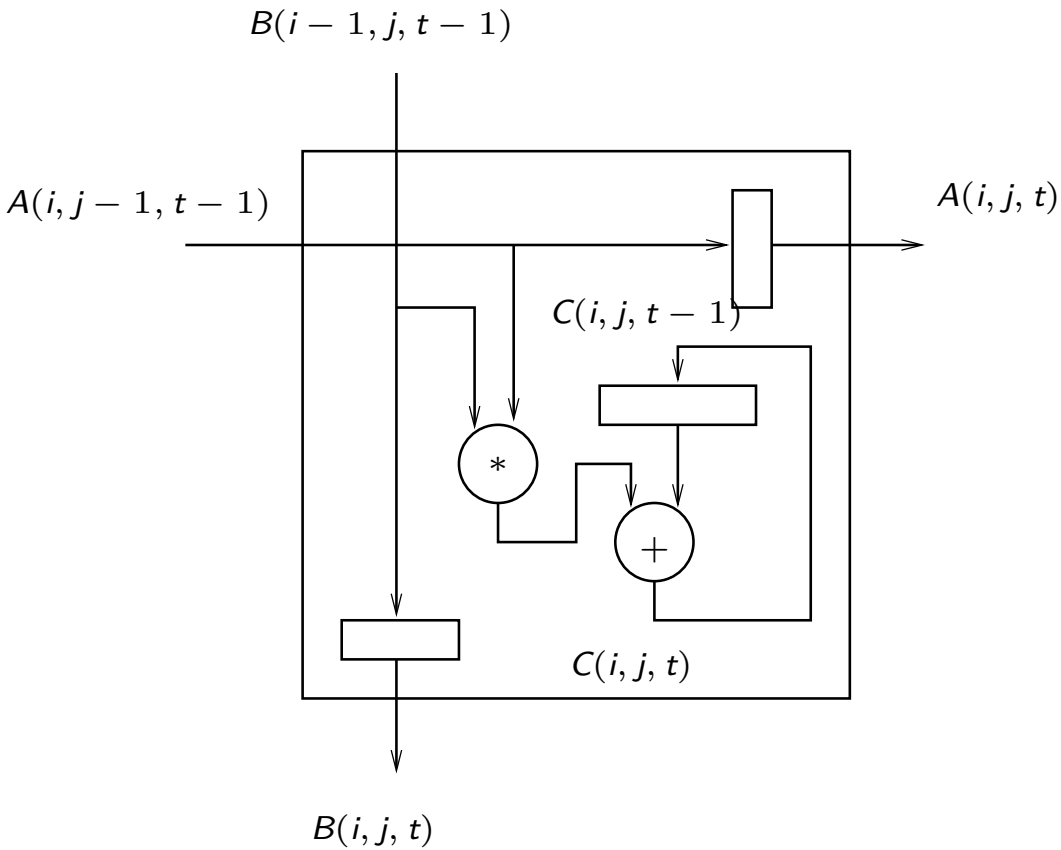
- Stable by intersection, image and pre-image by an affine function
- Dual representation allows many algorithms to be implemented
- There is a strong link between polyhedra and combinatorial optimization methods (simplex algorithm, for example)
- Libraries are available (Polylib, Parma etc.)
- But... some algorithms have exponential complexity

Parallel Matrix Multiplication



Describing a Parallel Architecture by Means of Recurrence Equations

One Processor of Matrix Multiplication

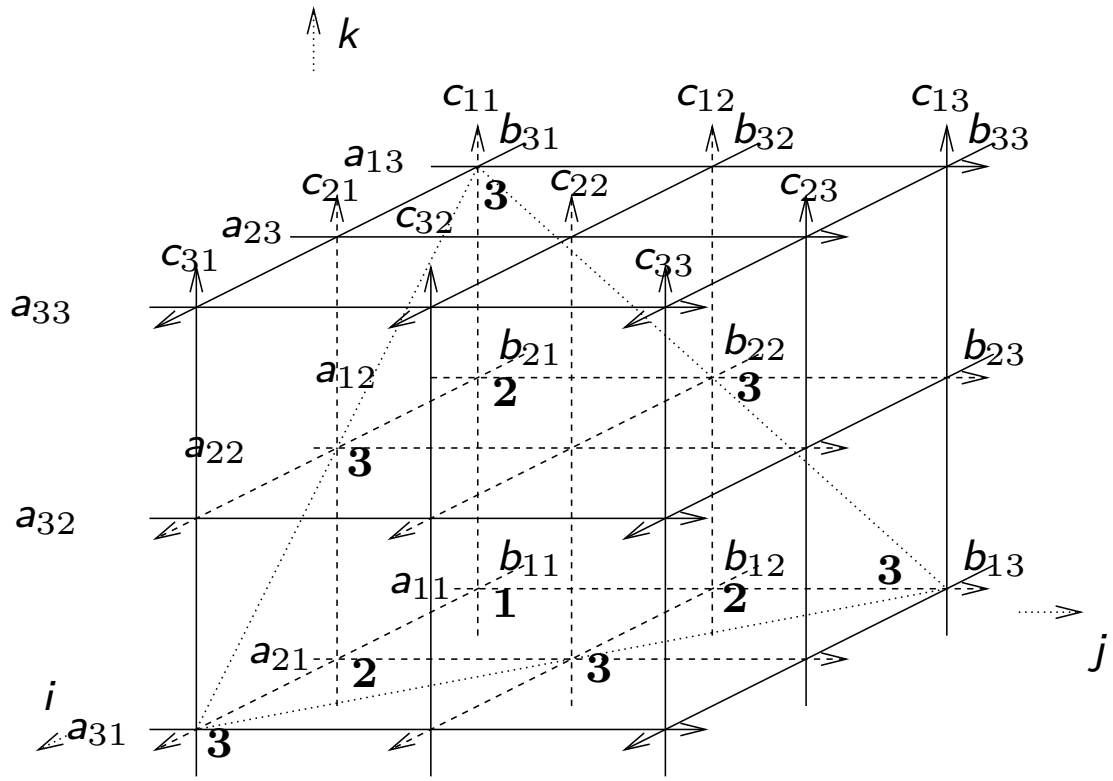


Describing a Parallel Architecture by Means of Recurrence Equations

Corresponding Recurrence Equations

$$
\begin{array}{lcl}
1 \leq i,j \leq 3, t \geq 1 & \rightarrow & C(i,j,t) = C(i,j,t-1) + \\
 & & A(i,j-1,t-1) \times B(i-1,j,t-1) \\
1 \leq i,j \leq 3, t \geq 1 & \rightarrow & A(i,j,t) = A(i,j-1,t-1) \\
1 \leq i,j \leq 3, t \geq 1 & \rightarrow & B(i,j,t) = B(i-1,j,t-1) \quad .
\end{array}
$$

From Algorithm to Architecture



Uniform Recurrence Equations

Equations

$$
\begin{array}{rcl}
1 \leq i,j,k \leq 3 & \rightarrow & C(i,j,k) = C(i,j,k-1) + A(i,j,k) \times B(i,j,k) \\
1 \leq i,j,k \leq 3 & \rightarrow & A(i,j,k) = A(i,j-1,k) \\
1 \leq i,j,k \leq 3 & \rightarrow & B(i,j,k) = B(i-1,j,k)
\end{array}
$$

Initial Conditions and Definition of Results

$$
\begin{array}{rcl}
1 \leq i,j \leq 3 \wedge k = 0 & \rightarrow & C(i,j,k) = 0 \\
1 \leq i,k \leq 3 \wedge j = 0 & \rightarrow & A(i,j,k) = a(i,k) \\
1 \leq j,k \leq 3 \wedge i = 0 & \rightarrow & B(i,j,k) = b(k,j) \\
1 \leq i,j \leq 3 & \rightarrow & c(i,j) = C(i,j,3).
\end{array}
$$

Spatio-temporal Transformations

Scheduling

Compute calculation (i, j, k) at time $t(i, j, k) = i + j + k - 2$

Allocation

Compute calculation (i, j, k) on processor (i, j)

Apply a change of basis to equations

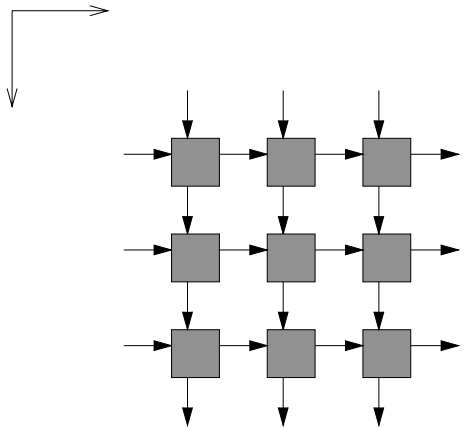
$$(i, j, k) \rightarrow (t, x, y) = (i + j + k - 2, i, j)$$

Matrix Multiplication (version 1)

Transformation

Scheduling : $t(i,j,k) = i+j+k$
Allocation Direction : k axis.
Change of Basis : $(i,j,k) \rightarrow (t,x,y) = (i+j+k, i, j)$

Architecture



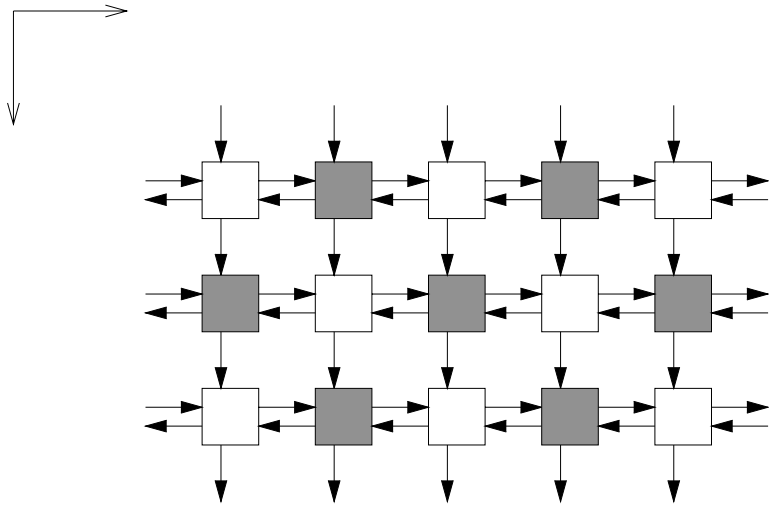
Matrix Multiplication (version 2)

Transformation

Allocation Direction : vector $s_2 = (1, 1, 0)^t$ (line $i = j,\ k = 0$)

Change of Basis : $(i, j, k) \rightarrow (t, x, y) = (i + j + k, j - i, k)$

Architecture



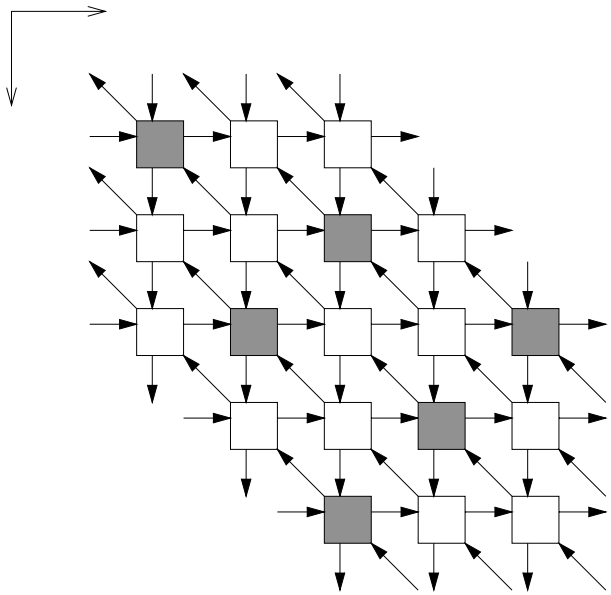
Matrix Multiplication (version 3)

Transformation

Allocation direction : vector $s_2 = (1,1,1)^t$ (line $i = j = k$)

Change of Basis : $(i,j,k) \rightarrow (t,x,y) = (i+j+k, j-i, k-i)$

Architecture



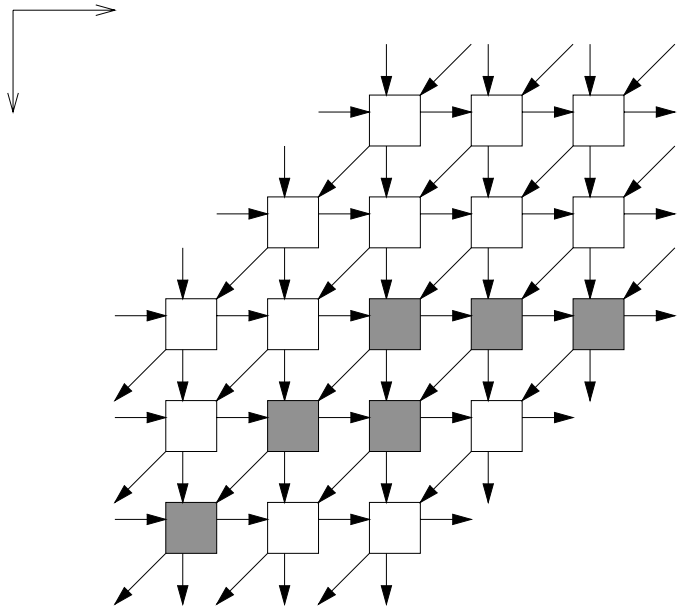
Matrix Multiplication (version 4)

Transformation

Allocation direction : vector $s_2 = (1, 1, -1)^t$ (line $i = j = -k$)

$(i, j, k) \rightarrow (t, x, y) = (i + j + k, j - i, k + i)$

Architecture



Partial Conclusion

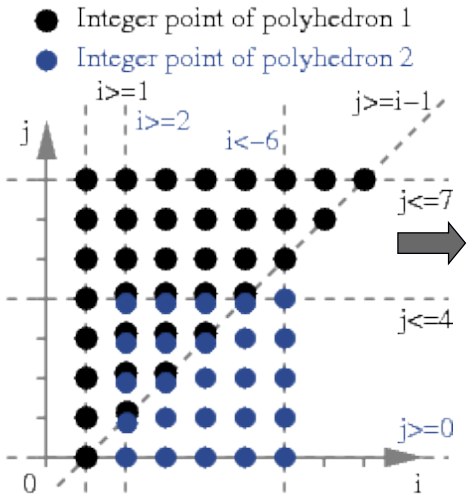
- Two flavors : analysis of imperative programs, or expression using functional languages (recurrence equations)
- Two different goals : efficient execution of (parallel) code, or synthesis of efficient hardware

$\Rightarrow$ My point of view (in this lecture) : functional language, and synthesis of hardware

- But remember that :
  - going from loops to recurrence equations can be done automatically,
  - translating recurrence equations to loops can be done efficiently.

Code Generation (Courtesy of Derrien)

- We must regenerate code for the transformed index set !
  - Sequence of loops which scans the transformed domains
  - Known as the polyhedron scanning problem
- Example : ClooG code generator [1]




```
for (i=1;i<=8;i++) {
  for (j=i-1;j<=7;j++)
    S1(i,j);
  if ((i>=2)&&(i<=6)) {
    for (j=0;j<=4;j++)
      S2(i,j);
  }
}
```

optimized for code size

optimized for control

```
for (j=0;j<=7;j++) S1(1,j);
for (i=2;i<=5;i++) {
  for (j=0;j<=i-2;j++) S2(i,j)
  for (j=i-1;j<=4;j++) {
    S1(i,j);
    S2(i,j) ;
  }
  for (j=5;j<=7;j++) S1(i,j);
}
for (j=0;j<=4;j++) S2(6,j) ;
for (j=5;j<=7;j++) S1(6,j);
for (i=7;i<=8;i++)
  for (j=i-1;j<=7;j++)
S1(i,j);
```

[1] Cedric Bastoul, Code Generation in the Polyhedral Model Is Easier Than You Think. In Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques, 2004

Outline

Alpha, a polyhedral language

- A fonctional language
- An intermediate language for loop transformations
- Allows both algorithm and architecture representation
- Makes a large use of polyhedra's properties

Matrix Multiplication in Alpha

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

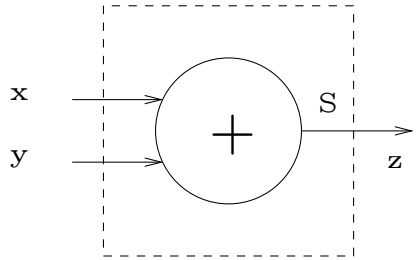
The True Nature of an Alpha Program

$C =$
case
$\{i, j, k | k = 0\} : 0.(i, j, k \rightarrow);$
$\{i, j, k | 1 \leq k \leq M\} : C.(i, j, k \rightarrow i, j, k - 1) + a.(i, j, k \rightarrow i, k) * b.(i, j, k \rightarrow k, j);$
esac ;

C is a function. It is defined by means of a functional expression using operators (+, *), transformations by affine functions, and operations on polyhedra.

Pointwise Operator

Hardware Device

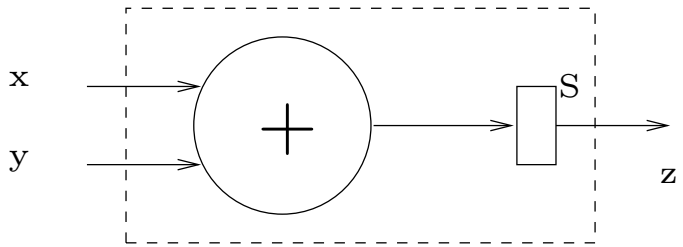


Alpha Program

```
system adder (x,y: {t | 1 <= t} of integer)          -- 1
returns (z: {t | 1 <= t} of integer);                -- 2
var S: {t | 1 <= t} of integer;                      -- 3
let                                                  -- 4
  S = x + y;                                         -- 5
  z = S;                                             -- 6
tel;                                                 -- 7
```

Dependence Function

Hardware Device

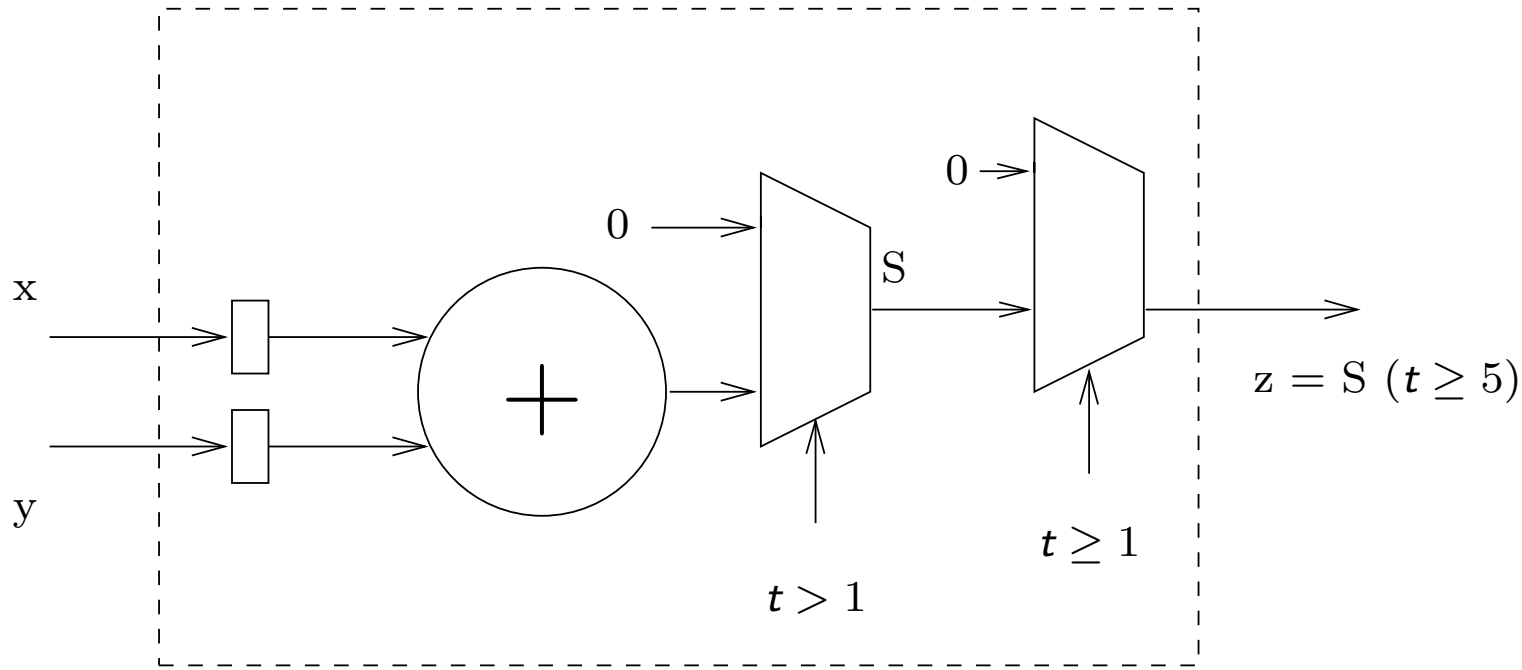


Alpha Program

```
system adderDelayed (x,y: {t|1<=t} of integer)    -- 1
returns (z: {t|2<=t} of integer);                 -- 2
var S: {t|2<=t} of integer;                       -- 3
let
  S = (x + y).(t->t-1);                           -- 4
  z = S;                                          -- 5
tel;
```

Case and restriction

Hardware Device



Case and restriction

Alpha Program (Array Form)

```
system adderMultiplexer (x,y: {t|1<=t} of integer)
returns (z: {t|5<=t} of integer);                    -- 1
var S: {t|1<=t} of integer;
let
 S[t] = case                                         -- 2
     {|t=1}: 0[];                                    -- 3
     {|t>1}: x[t-1] + y[t-1];                        -- 4
     esac;                                           -- 5
 z[t] = {|t>=5}: S[t];                               -- 6
tel;
```

Elements of Alpha

- Variable `x` : A function over a polyhedral domain $D(\texttt{x})$.
- Pointwise operation : `exp1 + exp2` is defined over $D(\texttt{exp1}) \cap D(\texttt{exp2})$.
- Restriction : `d :exp` is defined over $\texttt{d} \cap D(\texttt{exp})$.
- Case : `case exp1 ; exp2 esac` is defined over $D(\texttt{exp1}) \cup D(\texttt{exp2})$.
- Dependence : `exp.f` is defined over $\texttt{f}^{-1}(D(\texttt{exp}))$

A Few Properties

- One may substitute a variable by its definition anywhere (property of functional languages)
- Dependence distributes over operators.

$$(\texttt{exp1} + \texttt{exp2}).\texttt{f} = \texttt{exp1.f} + \texttt{exp2.f}$$

- Composition of dependences. $(\texttt{exp.f}).\texttt{g} = \texttt{exp1.}(\texttt{g.f})$
- Dependence distributes over case.
  `(case exp1 ; exp2 esac).f = case exp1.f ; exp2.f esac.`
- etc.

All these operations are valid for unions of convex polyhedra.

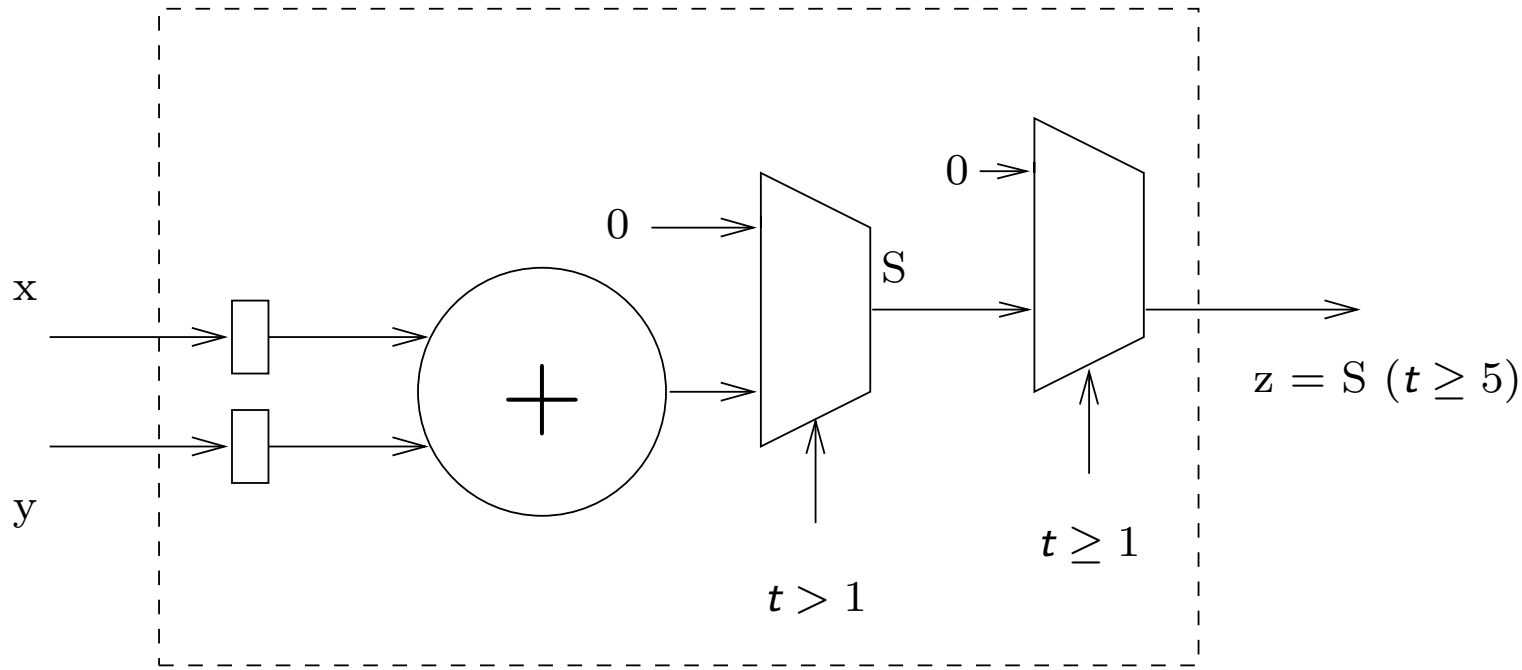
Normalization

Any expression can be rewritten as a case – restriction – dependence expression, by applying properties of the language.
Thus, any Alpha program can be simplified in its *normal form*.

```
case
   v1.f + v2.g ;
   ...
   v3.h + v4.k
esac
```

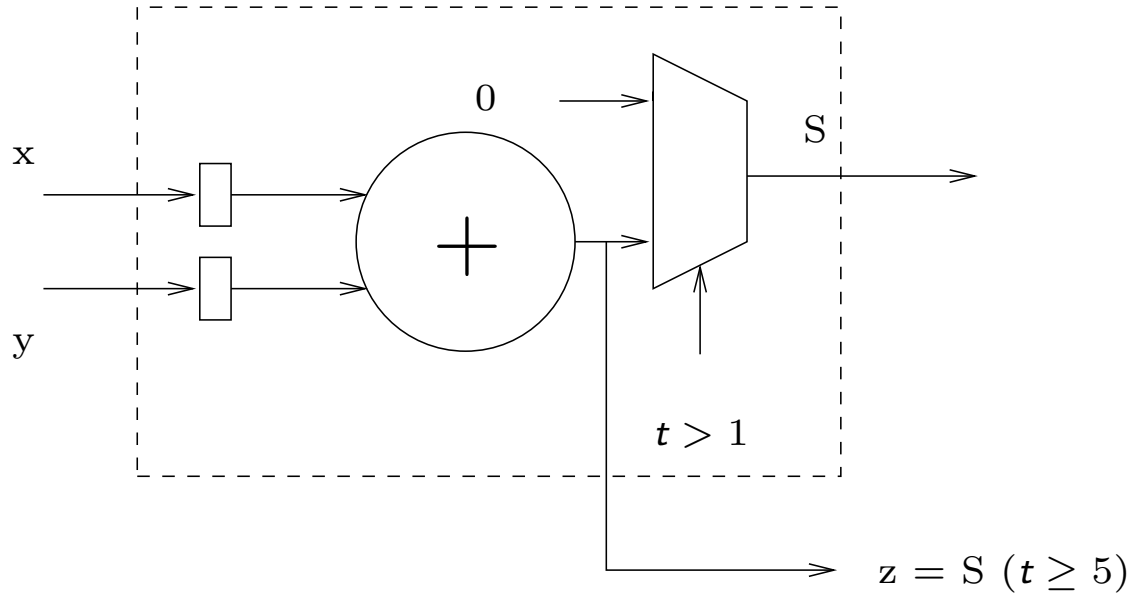
A Simple Application

Initial Architecture



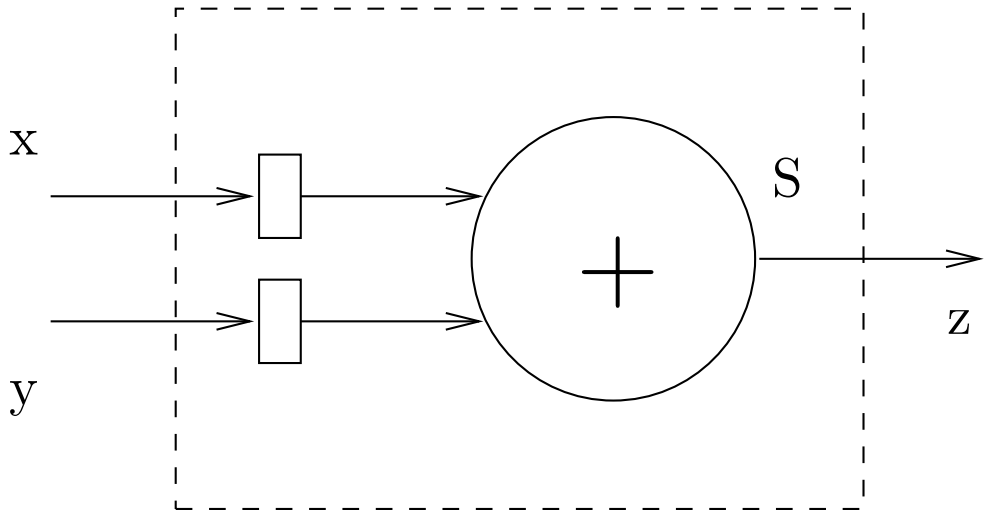
A Simple Application

Substitute and Normalize



A Simple Application

Architecture after simplification



Change of Basis

Let T be a linear, unimodular transformation of the space, and let T^{-1} be its inverse function. To apply change of basis T to some local variable V consists in :

- replace the declaration domain of V by its image under T,
- replace all occurences of V in the program by its image under T,
- replace the definition $V = exp$ of V by $V = exp.T^{-1}$.

This transformations preserves the behaviour of the program.

Exemple

system matmult : $\{M \mid M > 1\}$
  $(a, b : \{i, j \mid 1 \leq i, j \leq M\}$ **of real** ;
returns
  $(c : \{i, j \mid 1 \leq i, j \leq M\}$ **of real** ;
var C : $T(\{i, j, k \mid 1 \leq i, j \leq M\})$ of real ;
 let
$C = ($
  case
    $\{i, j, k | k = 0\} : 0.(i, j, k \rightarrow)$;
    $\{i, j, k | 1 \leq k \leq M\} : (C.T).(i, j, k \rightarrow i, j, k - 1)$
        $+a.(i, j, k \rightarrow i, k) * b.(i, j, k \rightarrow k, j)$;
  esac$).T^{-1}$;
  $c = (C.T).(i, j \rightarrow i, j, M)$;
tel ;

MMAlpha : an environment for Alpha

- Mathematica + C environment
- From Alpha program to VHDL code
- Uses Alpha both for representing algorithm and architecture
- Synthesis as a rewriting process

Steps of MMAlpha

Step 1

Parse and analyze

Step 2

Schedule and apply schedule (change of basis)

Step 3

Generate structured hardware (AlpHard)

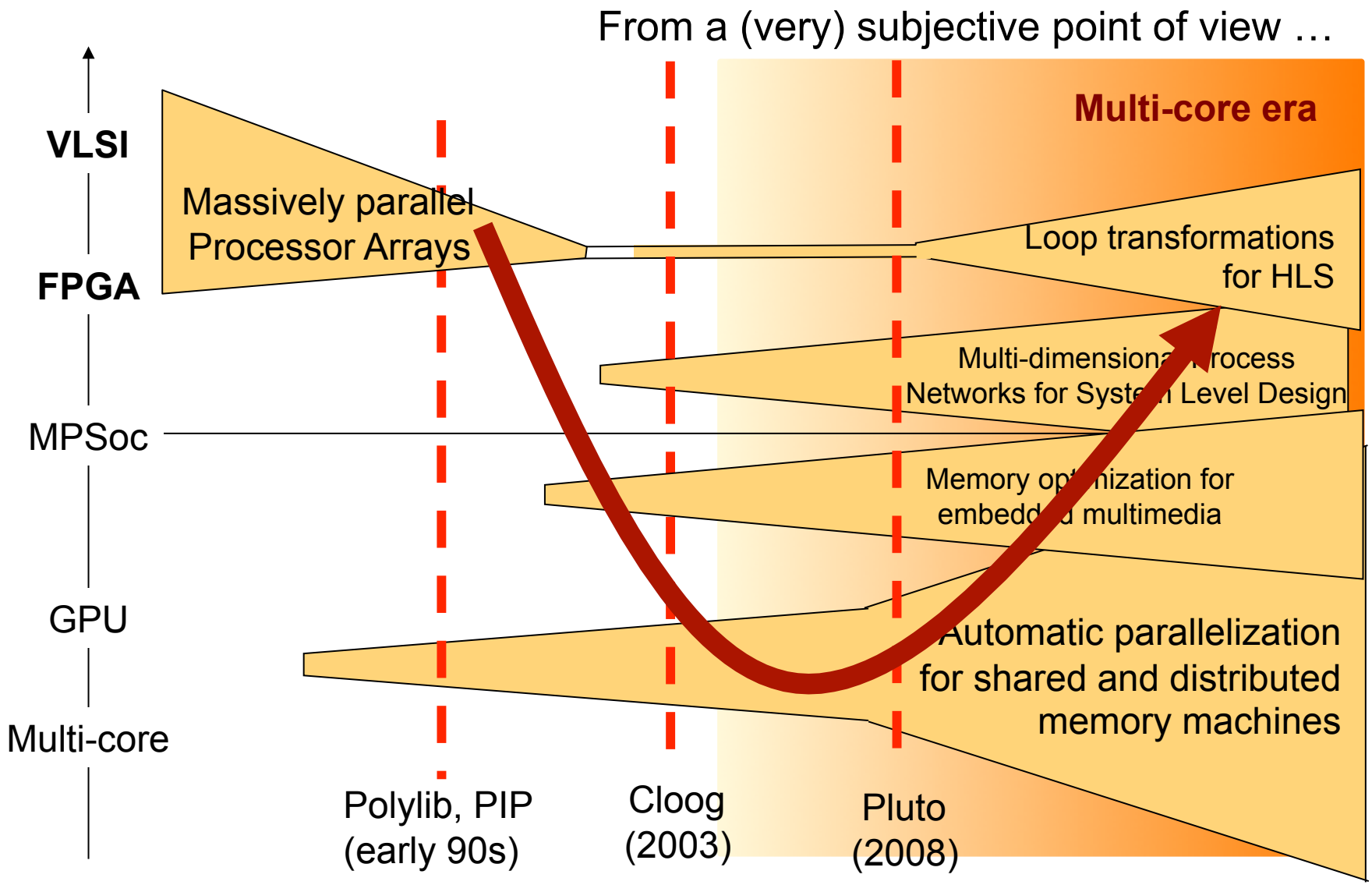
Step 4

Generate VHDL (AlpHard)

A Brief History

- First research on recurrence equations : 1967 Karp, Miller et Winograd
- Loop parallelization : 1973 Lamport
- Program parallelization : 1975 – 1985 Kuck (Univ. Illinois)
- Systolic architectures : 1980 – 1990 Moldovan, Quinton, Rajopadahye
- Loop analysis and scheduling : 1980 – ... Feautrier, Darte (ENS Lyon)
- Polylib library : Le Verge, Wilde (1990)
- Alpha and MMAlpha : Mauras (1989), Risset, Dupont, Cachera etc.
- Parallel circuit generation : 1990 – ... (Pico, Compaan, MMAlpha)

Another Version of the History



Outline

Why ?

- Simplifying analysis of systems : modular scheduling
- Abstracting components
  - Libraries of components (or IPs)
  - Complex systems
- Modelling hardware components
  - AlpHard
  - Data-flow systems
- Going beyond the limitations of the model
  - Recursive Alpha
  - Tiling

Outline of this Part

- Presentation of main principles (matrix multiplication)
- A second example related to hardware
- Kalman filtering
- Structured scheduling
- Subsystems as hardware components : the Smith-Waterman algorithm
- Data-flow systems

How ? Matrix multiplication in ALPHA (Mauras, 1989)

system matmult : $\{M \mid M > 1\}$ // Parameters
    (a,b : $\{i,j \mid 1 \leq i,j \leq M\}$ **of integer**) // Input matrices
returns (c : $\{i,j \mid 1 \leq i,j \leq M\}$ **of integer**) ; // Output matrix
var
    C : $\{i,j,k \mid 1 \leq i,j \leq M; 0 \leq k \leq M\}$ **of integer** ;
let // Equations
    c[i,j] = C[i,j,M] ; // Define result
    C[i,j,k] = **case**
        $\{\mid k = 0\}$: 0 ;
        $\{\mid 1 \leq k \leq M\}$: C[i,j,k-1]+a[i,k]*b[k,j] ;
    esac ; // Define accumulation, in single assignment form
tel ;

Idea

- ALPHA variables are functions from a polyhedral domain to some type value ($\mathbf{integer}, \mathbf{real}, \mathbf{boolean}$)
- An ALPHA program P is a function from its input variables to its output variables
- Mapping an ALPHA program P (subsystem) to the points of a polyhedral domain D will return a meaningful ALPHA program, with new added dimensions : **Map** (P, D)
- Syntax : **use** D $P[$ $parameters$ $]$ $(inputs)$ **returns** $(outputs)$

Structured matrix multiplication

system dotprod : $\{N \mid 1 \leq N\}$
(a : $\{k \mid 1 \leq k \leq N\}$ **of integer** ;
b : $\{k \mid 1 \leq k \leq N\}$ **of integer**)
returns (c : **integer**) ;
let
c = **reduce**(+, ($k \rightarrow$), a[k] * b[k]) ; // $\sum_k$ a[k] $\times$ b[k]
tel ;

Structured matrix multiplication

system matmult :$\{M \mid M > 1\}$
(a : $\{i,j \mid 1 \leq i \leq M; 1 \leq j \leq M\}$ **of integer** ;
b : $\{i,j \mid 1 \leq i \leq M; 1 \leq j \leq M\}$ **of integer**)
returns (c : $\{i,j \mid 1 \leq i \leq M; 1 \leq j \leq M\}$ **of integer**) ;
var
A : $\{k,i,j \mid 1 \leq k \leq M; 1 \leq i \leq M; 1 \leq j \leq M\}$ **of integer** ;
B : $\{k,i,j \mid 1 \leq k \leq M; 1 \leq i \leq M; 1 \leq j \leq M\}$ **of integer** ;
let
$A[k,i,j] = a[i,k]$;
$B[k,i,j] = b[k,j]$;
use $\{i,j \mid 1 \leq i \leq M; 1 \leq j \leq M\}$ dotprod$[M]$ (A, B) **returns** (c) ;
tel ;

Another example related to Hardware

```
system FullAdder(A, B, Cin : boolean)
returns (X, Cout : boolean);
let
    X = A xor B xor Cin;
    Cout = (A and B) or (A and Cin) or (B and Cin);
tel;
```

Binary adder of size W

system Plus : $\{W \mid W > 1\}$ (A,B : $\{b \mid 0 \leq b \leq W\}$ **of boolean**)
returns (S : $\{b \mid 0 \leq b \leq W\}$ **of boolean**) ;
var Cin, Cout, X : $\{b \mid 0 \leq b < W\}$ **of boolean** ;
let
  Cin[b] = **case**
    $\{ \mid b = 0\}$: **false**[] ;
    $\{ \mid b > 0\}$: Cout[$b-1$] ;
  esac ;
  S[b] = **case**
    $\{ \mid b < W\}$: X[b] ;
    $\{ \mid b = W\}$: Cout[$W-1$] ;
  esac ;
  use $\{b \mid 0 \leq b < W\}$ FullAdder (A,B,Cin) **returns** (X, Cout) ;
tel ;

Complexity in the expression : Kalman filtering

$$\hat{\mathbf{x}}_{k+1|k} = \Phi\hat{\mathbf{x}}_{k|k} \quad (1)$$

$$\mathbf{P}_{k+1|k} = \Phi\mathbf{P}_{k|k}\Phi^T + \mathbf{b}\beta\mathbf{b}^\mathbf{T}, \mathbf{P_{0/0}} = \mathbf{I} \quad (2)$$

$$r_{k+1} = \mathbf{h}_k\mathbf{P}_{k+1|k}\mathbf{h}_k^T + 1 \quad (3)$$

$$\mathbf{K}_{k+1} = \mathbf{P}_{k+1|k}\mathbf{h}_{k+1}^T r_{k+1}^{-1} \quad (4)$$

$$\hat{y}_{k+1} = \mathbf{h}_{k+1}\hat{\mathbf{x}}_{k+1|k} \quad (5)$$

$$\delta_{y_{k+1}} = \tilde{y}_{k+1} - \hat{y}_{k+1} \quad (6)$$

$$\hat{\mathbf{x}}_{k+1|k+1} = \hat{\mathbf{x}}_{k+1|k} + \mathbf{K}_{k+1}\delta_{y_{k+1}} \quad (7)$$

$$\mathbf{P}_{k+1|k+1} = (\mathbf{I} - \mathbf{K}_{k+1}\mathbf{h}_{k+1})\mathbf{P}_{k+1|k} \quad (8)$$

In Alpha

...

```
use matvect[M] (phi, xchapz) returns (xint);
use dotprod[M] ( xint, H) returns (Ychap);
I[] = yb[] - Ychap[];
use matmult[M] (P, phit) returns (V1);
use matmult[M] (phi, V1) returns (PP1);
PP = PP1 + bbt;
use matvect[M] (PP, H) returns (V2);
use dotprod[M] (Hz, V2) returns (Veint);
use matvect[M] (PP, Hz) returns (V3);
invVe[] = 1 / (Veint[]+1[]);
K[m] = V3[m] * invVe[];
xchap[m] = xint[m] + K[m] * I[];
Pint[m,i] = Ip[m,i] - K[m] * H[i];
use matmult[M] (Pint, PP) returns (Pkk);
```

...

Scheduling (a short summary)

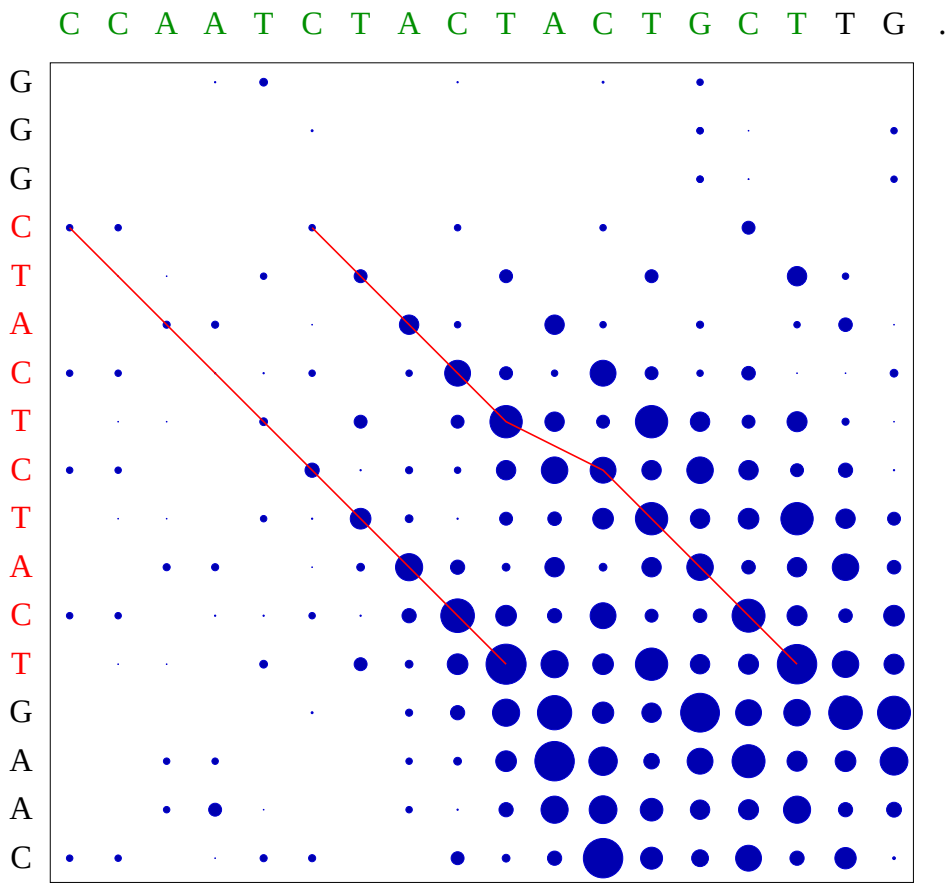
- Scheduling is assigning time values τ such that dependences are satisfied $(x = a + b \Rightarrow \tau(x) > \tau(a) \wedge \tau(x) > \tau(b)$
- For *polyhedral loops* or *polyhedral equations*, finding out τ amounts to solving a linear programming problem (see Feautrier's work)
- Scheduling does not scale well. Simplex is (roughly) cubic in the number of linear constraints, and dependencies are quadratic in the size of the program

$\Rightarrow$: What can we do ? Divide and (try to) conquer...

Complexity of the analysis : Matrix Multiplication

- Scheduling subsystems : inline, then schedule
- Hierarchical scheduling :
  - Instead of inlining then scheduling, apply "bottom-up" scheduling (Risset et al.)
  - Or, summarize the potential schedules of a component using a polyhedron (Feautrier)
- Example :
  - Matrix multiplication, after inlining : 289 constraints, 43 variables, 2.64s on this laptop using Mathematica's solver
  - Dot product : 35 constraints, 13 variables, 0.01s
  - Matrix multiplication, structured scheduling : 147 constraints, 28 variables, 0.13s

Genetic sequence alignment : the Smith-Waterman Algorithm



First application : generation of hardware

In Alpha

system sequence :$\{X, Y \mid 3 \leq X \leq Y-1\}$
(QS : $\{i \mid 1 \leq i \leq X\}$ **of integer** ; // Query sequence, size X
DB : $\{j \mid 1 \leq j \leq Y\}$ **of integer**) // Data base sequence, size Y
returns (res : $\{j \mid 1 \leq j \leq Y\}$ **of integer**) ; // Score of best match
var
M : $\{i, j \mid 0 \leq i \leq X; 0 \leq j \leq Y\}$ **of integer** ;
MatchQ : $\{i, j \mid 1 \leq i \leq X; 1 \leq j \leq Y\}$ **of integer** ;
let
M$[i, j]$ =
case
$\{\mid i = 0\} : 0$;
$\{\mid 1 \leq i; j = 0\} : 0$;
$\{\mid 1 \leq i; 1 \leq j\}$: max (0, max(M$[i, j-1] - 8$, M$[i-1, j-1]$ + MatchQ$[i, j]$)) ;
esac ;
MatchQ$[i, j]$ = **if** (QS$[i]$ = DB$[j]$) **then** 15 **else** -12 ;
res$[j]$ = M$[X, j]$;

tel ;

Generating Structured Hardware using MMALPHA

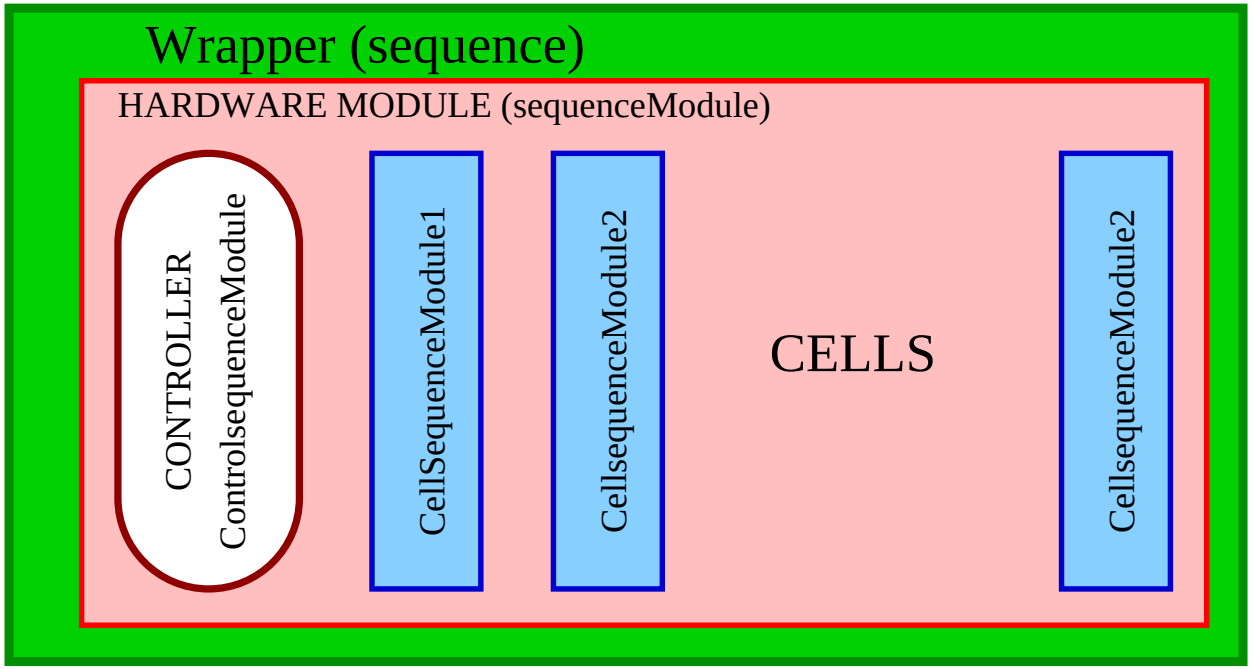
- Schedule, place
- Rewrite equations at register transfer level (subset of ALPHA called ALPHA_0), using *space-time transformation* (also called *change of basis*)
- Back-end : translate the new description into VHDL

$\Rightarrow$: necessary to separate the various components (controller, processing elements, hierarchy, etc.), to get efficient VHDL

$\Rightarrow$: use subsystems !

The Smith-Waterman Architecture

AlpHard : a subset of ALPHA to represent hardware



Alpha Wrapper

system sequence :$\{X, Y | 3 \leq X \leq Y-1\}$
(QS : $\{i \mid 1 \leq i \leq X\}$ **of integer** ;
DB : $\{j \mid 1 \leq j \leq Y\}$ **of integer**)
returns (res : $\{j \mid 1 \leq j \leq Y\}$ **of integer**) ; // Same interface
var // Mirror variables, with time and space
DBXMirr1 : $\{t, p \mid 1 \leq t \leq Y; p = 0\}$ **of integer** ;
TSep3 : $\{t, p \mid t = 1; 1 \leq p \leq X\}$ **of integer** ;
M : $\{t, p \mid 1 \leq t \leq Y+1; 0 \leq p \leq X\}$ **of integer** ;
let
// Mirror equations, allow test program to be generated automatically
DBXMirr1$[t, p]$ = DB$[t]$;
TSep3$[t, p]$ = QS$[t+p-1]$;
res$[j]$ = M$[j+1, X]$;
// Instanciate the Hardware module
use sequenceModule$[X, Y]$ (DBXMirr1, TSep3) **returns** (M) ;
tel ;

ALPHA description of the controller

system ControlsequenceModule : $\{X, Y \mid 3 \leq X \leq Y-1\}$ ( )
returns (pipeCQS59Xctl1 : $\{t \mid t = 1\} \mid \{t \mid 2 \leq t \leq Y\}$ **of boolean** ;
MXctl2 : $\{t \mid t = 1\} \mid \{t \mid 2 \leq t \leq Y+1\}$ **of boolean**) ;
let
pipeCQS59Xctl1[t] =
case
$\{\mid t = 1\}$: **true**[] ;
$\{\mid 2 \leq t\}$: **false**[] ;
esac ;
MXctl2[t] =
case
$\{\mid t = 1\}$: **true**[] ;
$\{\mid 2 \leq t\}$: **false**[] ;
esac ;
tel ;

ALPHA description of Hardware Module

system sequenceModule : $\{X, Y \mid 3 \leq X \leq Y-1\}$
(DBXMirr1In : $\{t, p \mid 1 \leq t \leq Y; p = 0\}$ of integer ;
TSep3In : $\{t, p \mid t = 1; 1 \leq p \leq X\}$ of integer)
returns$\{\}$ (MOut : $\{t, p \mid 1 \leq t \leq Y+1; 0 \leq p \leq X\}$ of integer) ;
var
pipeCQS59Xctl1 : $\{t \mid 1 \leq t \leq Y\}$ **of boolean** ; ...
let ...
// Instanciate the controller
use ControlsequenceModule[X, Y] () **returns** (pipeCQS59Xctl1, MXctl2) ;
// Instanciate first cell
use $\{p \mid p = 0\}$ cellsequenceModule1[p, X, Y] (DBXMirr1)
returns (M1, pipeCDB611) ;
– Instanciate other cells
use $\{p \mid 1 \leq p \leq X\}$ cellsequenceModule2[p, X, Y] (MReg2Xloc,
pipeCDB61Reg4, MXctl2XIn, pipeCQS59Xctl1XIn, TSep3)
returns (M2, pipeCDB612) ;
tel ;

ALPHA description of a Cell

```
system cellsequenceModule2 :{p,X,Y | 1<=p<=X; 3<=X<=Y-1} ...
returns             (M : {t | 1<=t<=Y+1} of integer; ...
var ... let ...
  MReg2[t] = MReg2Xloc[t-1];                -- Register
  pipeCQS59Reg5[t] = pipeCQS59[t-1];        -- Register
  MatchQReg3[t] = MatchQ[t-1];              -- Register
  MReg1[t] = Mloc1[t-1];                    -- Register
  pipeCQS59[t] =                            -- Mux
      case
        { | t=1} : if (pipeCQS59Xctl1XIn[t]) then TSep3[t] else 0[];
        { | 2<=t} : if (pipeCQS59Xctl1XIn[t]) then 0[] else pipeCQS59Reg5[t];
      esac;
  TSep4[t] = MReg1[t] - 8[];                -- Subtracter
  TSep5[t] = MReg2[t] + MatchQReg3[t];      -- Adder
  TSep6[t] = max(TSep4[t], TSep5[t]);       -- Max
  TSep7[t] = max(0[], TSep6[t]);            -- Max
  Mloc1[t] =                                -- Mux
      case
        { | t=1} : if (MXctl2XIn[t]) then 0[] else 0[];
        { | 2<=t} : if (MXctl2XIn[t]) then 0[] else TSep7[t];
      esac;
  TSep1[t] = pipeCQS59[t] = pipeCDB61Reg4[t];  -- Comparator
  TSep2[t] = -12[];
  MatchQ[t] = if (TSep1[t]) then 15[] else TSep2[t];  -- Mux
tel;
```

Conclusion on modularity

Summary

- Modularity can be effectively introduced in the polyhedral model
- ... to model hardware
- ... to tackle complexity
- ... to hide difficulties.

Related work on modularity (non exhaustive)

- R. Triolet et al. (1986) and more recently, B. Creusillet et al. (1996)
- B. Dupont de Dinechin (Thesis, 1997)
- P. Quinton and T. Risset (Structured Scheduling, 2001)
- P. Feautrier (Scalable and Structured scheduling, 2006)
- P. Quinton et al (Data-flow Systems, 2004, 2012)
- Teams API, Cosi, R2D2, Cairn at IRISA : MMALPHA, Gecos and all these sorts of things

Outline

Motivation and Context

Context

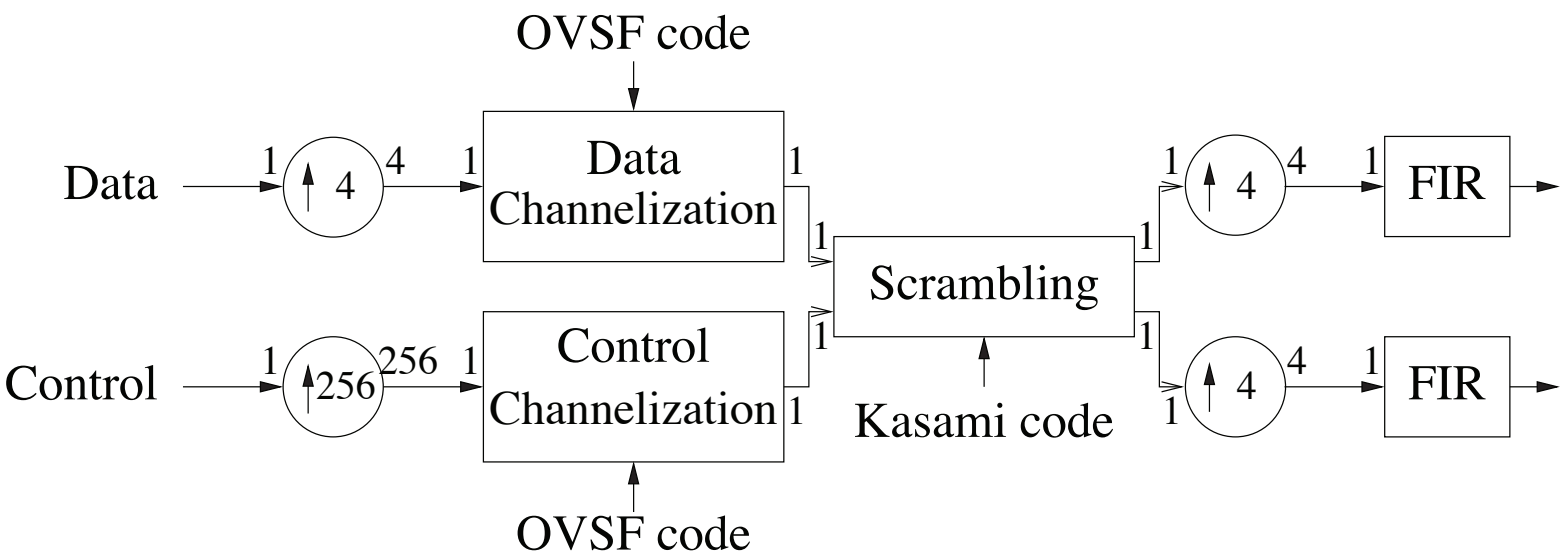
- Assembling components to build a system : flexibility, time models
- Multi-core and GPU : model parallelism
- Combine components with different frequencies

Contribution

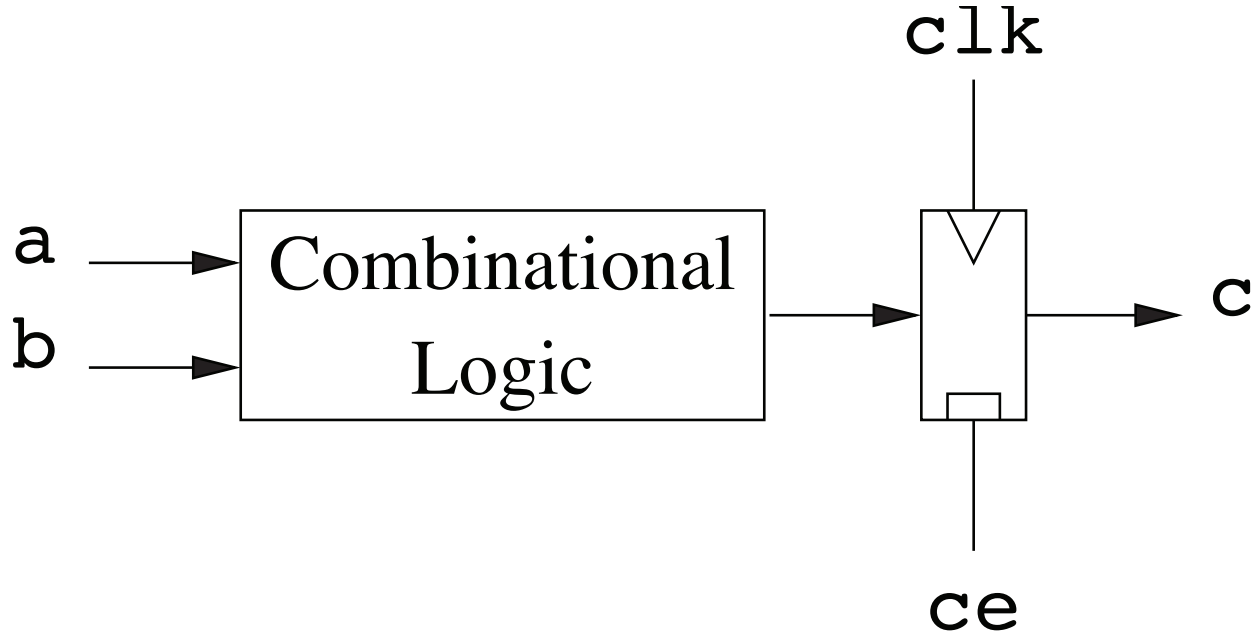
- Present a simple model of time
- Explain how it can be implemented in Hardware
- Show how this model of time can be use to schedule multidimensional data-flow systems
- Present implementation results

Example

Simplified WCDMA emitter

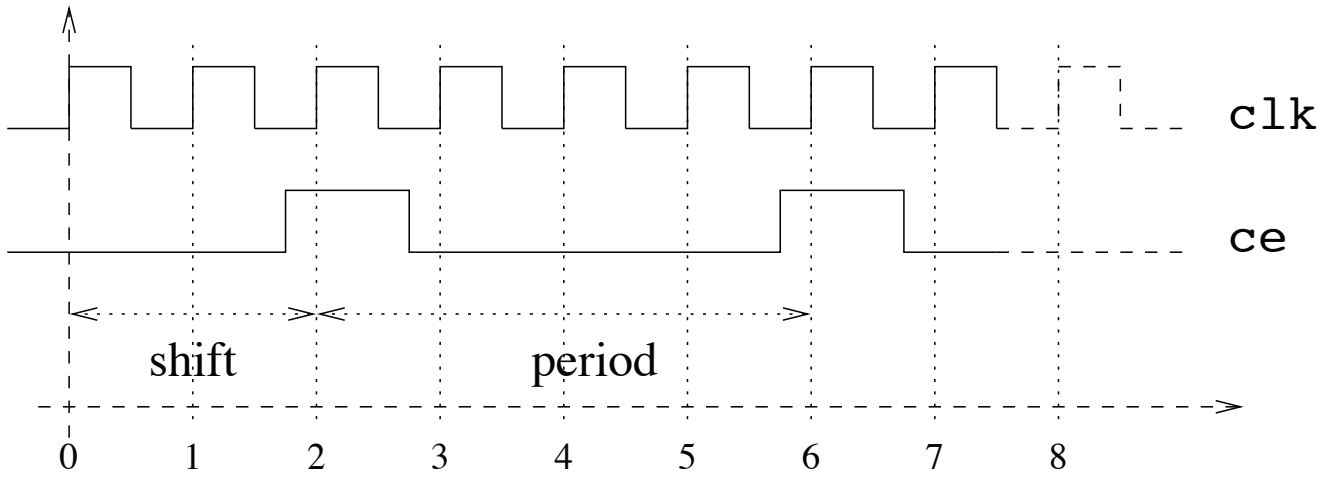


Combinational component



Clocking scheme

Example

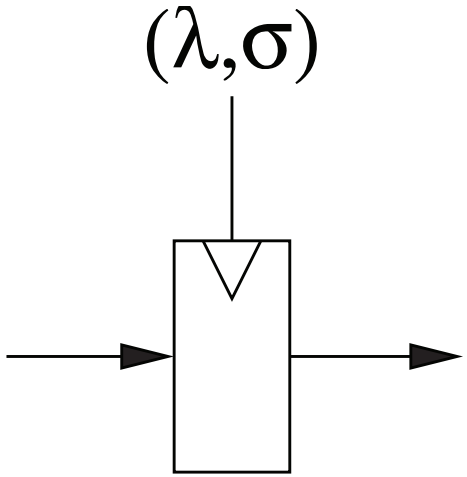


Shift and period

Clock enable allows initial shift and period to be controlled

Clocking scheme

Modeling registers and clocks



- λ is the period
- σ is the shift

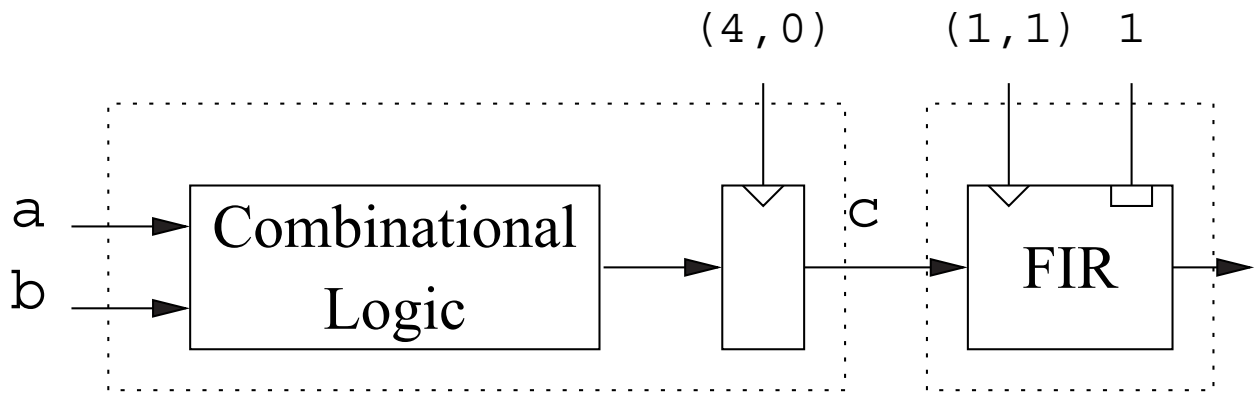
Controllers

Model of a controller

- All registers equipped with clock enable signal
- Reset signal to define initialization time
- Controller generates clock enable signals for internal components
- Controller generates also reset signals for internal components

Assembling components

Integrating components with different periods



Model of Time

- Synchronous time (based on single master clock)
- Affine periodic time
- Parallelism supported
- Components are *time delayable*
- Components are *slowable*
- Time is *stretchable*

Scheduling

The problem :

Find out the λ's and the σ's for all components

Constraints :

- Hierarchical scheduling method
- Cover the case of parallel components

The ALPHA language, as a support for expression

A declarative language

- Based on the *polyhedral model*
- Expressions are functions on polyhedral domains
- Allows data parallelism to be expressed
- Supports space-time transformations
- Hierarchical descriptions
- Implemented in the MMALPHA environment

ALPHA description of WCDMA system

```
system emitter : {KD, KC, KN, KK | 1<=KD; 1<=KC; 1<=KK; 3<=KN}
-- data, and control signal
         (data : {i|0<=i} of integer[S,2]; control : {j|0<=j} of integer[S,2])
returns
-- output data and control
         (sd: {i|KN<=i} of integer[S,32]; sc: {j|KN<=j} of integer[S,32]);
var
        dataMirr, dataMirrReg : {i|0<=i} of integer[S,2];
         ...
let
        dataMirr = data;
        use registerFile[1] (dataMirr) returns (dataMirrReg);
        ...
        use overSampling[4] (dataMirrReg) returns (ssdata);
         ...
        spdata[i] = ovsfDataCode[i] * ssdata[i];
        spcontrol[j] = ovsfControlCode[j] * sscontrol[j];
        ...
        use fir[KN] (sscdata, firCoeffs)  returns (sdMirr);
        use registerFile[1] (sdMirr) returns (sdMirrReg);
        ...
        sd = sdMirrReg;
         ...
tel;
```

Elements of the language

System declaration

```
system emitter : {KD, KC, KN, KK | 1<=KD ; 1<=KC ; 1<=KK ;
         3<=KN}
```

Inputs

```
         (data : {i|0<=i} of integer[S,2] ;
          control : {j|0<=j} of integer[S,2])
```

Outputs

```
returns (sd : {i|KN<=i} of integer[S,32] ;
         sc : {j|KN<=j} of integer[S,32]) ;)
```

Elements of the language

Local declarations

```
var dataMirr, dataMirrReg : i|0<=i of integer[S,2] ;
```

Instanciate subsystem

```
use registerFile[1] (dataMirr) returns (dataMirrReg) ;
use fir[KN] (sscdata, firCoeffs) returns (sdMirr) ;
```

Over sampler

```
use overSampling[4] (dataMirrReg) returns (ssdata) ;
```

Multi-dimensional structures

Declaration :

```
streamOfArrays :
 {i, n, m | 0 <= i ; 1 <= n, m <= 10}
 of integer ;
```

Meaning :

a stream of 10×10 matrices of integers

Multi-dimensional structures

Declaration :

```
streamOfArrays :
 {i, n, m | 0 <= i ; 1 <= n, m <= 10}
 of integer ;
```

Meaning :

a stream of 10×10 matrices of integers

Scheduling (principles)

System

A system S with stream input I and stream output O

Data-flow schedule

Schedule of the input : $T_{\mathbf{I}}(i, ...) = i + \alpha_{\mathbf{I}}$
Schedule of the output : $T_{\mathtt{O}}(i, ...) = i + \alpha_{\mathtt{O}}$

λ-slow schedule

for $\lambda > 0$, $\lambda T_{\mathbf{I}}$ and $\lambda T_{\mathtt{O}}$ are also valid schedules

Solving the schedule

Given :

- Components together with their schedule pattern
- An ALPHA system that connect components through over- and under samplers.

Find out the periods

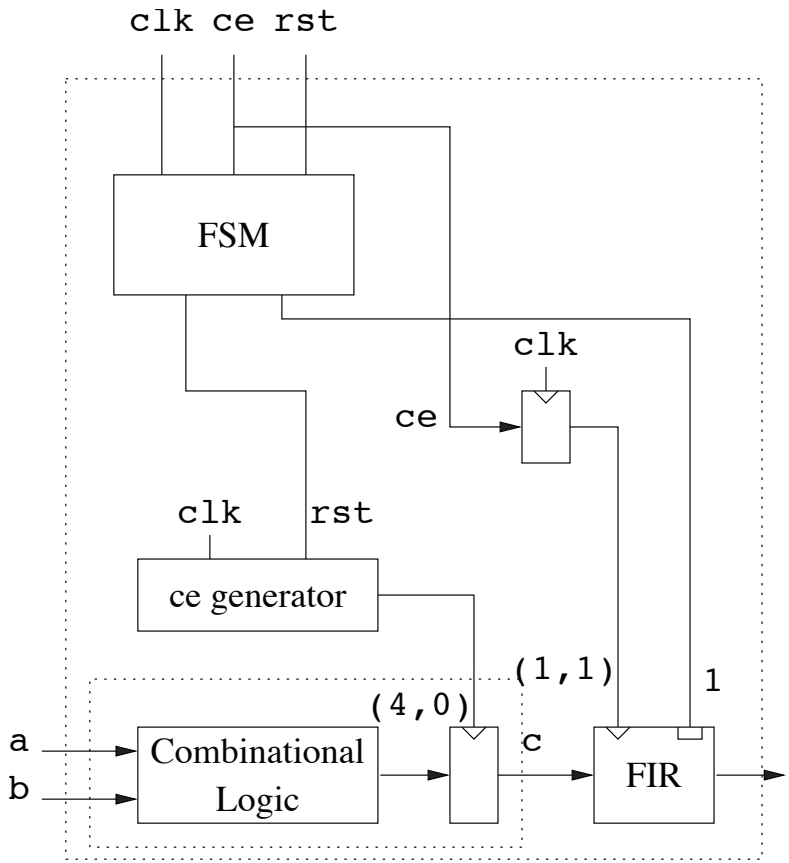
Solve a system of homogeneous equations (equivalent to *balance equations* in synchronous data-flow systems).

Solve the schedule

Report periods in schedule patterns of components, and solve the global schedule using hierarchical polyhedral schedule method.

Architecture Generation

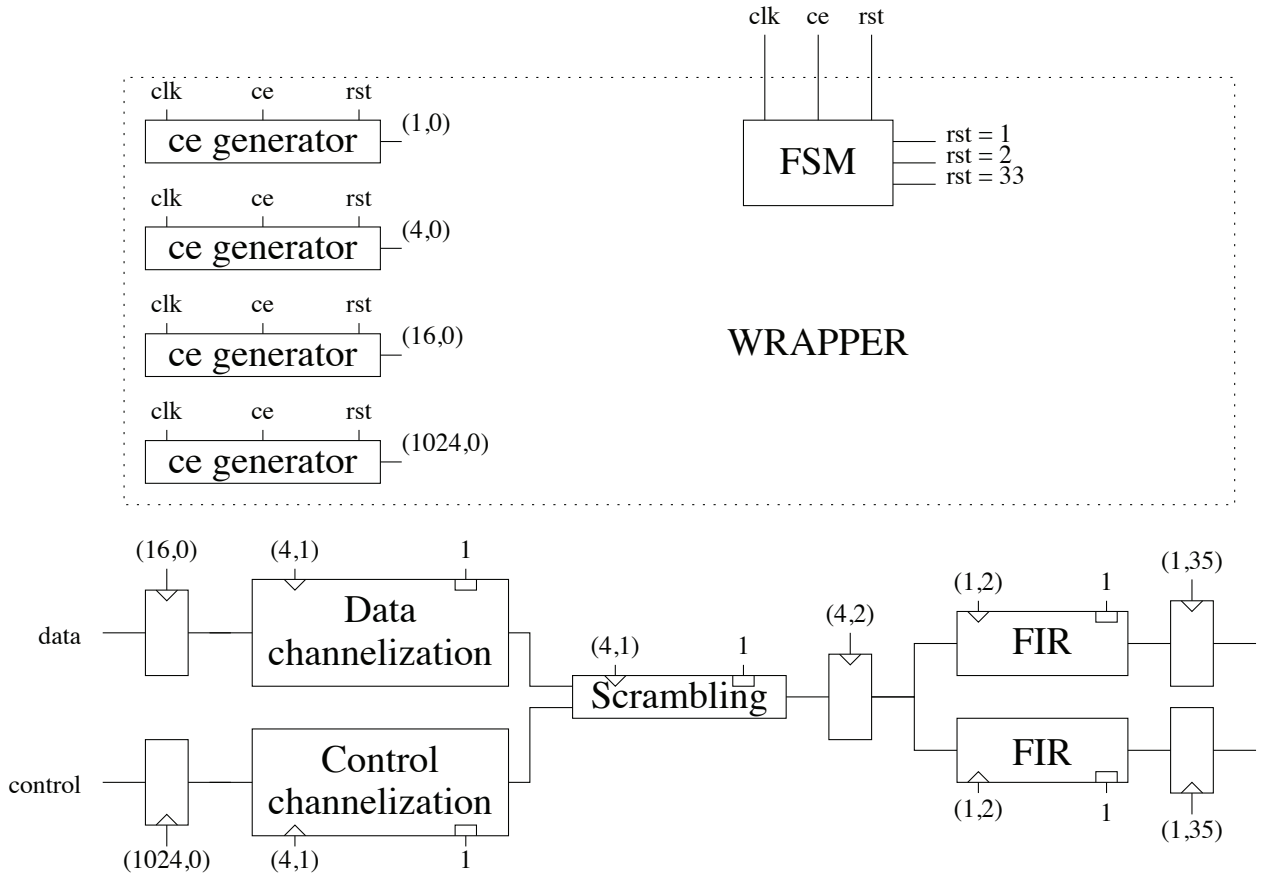
Wrapper architecture



Implementation

- System described using MMALPHA
- Fir filter synthesized using MMALPHA and abstracted by its scheduling
- Scheduling done automatically using the polyhedral method
- Wrapper, clock enable generators etc. generated automatically
- VHDL code generated, tested, and synthesized

Implementation



Results

Component	#Slices	#MAC	#Latch	Frequency (MHz)
FSM	24	0	34	-
Logical clock 4	2	0	2	-
Logical clock 16	10	0	13	-
Logical clock 1024	10	0	13	-
Wrapper (total)	41	0	53	415,6 MHz
Emitter (total)	1962	68	7528	129,5 MHz

Related work

- Multidimensional Synchronous Dataflow
- Latency Insensitive Design
- Stream Languages

Outline

The Future

- Few direct applications of this research as the background is pretty complex and algorithms are not totally mastered
- Need to hide such methods in compilers. See GeCoS for example
  http ://gecos.gforge.inria.fr/doku.php, or Pluto
  http ://pluto-compiler.sourceforge.net/ or AlphaZ
  http ://www.cs.colostate.edu/AlphaZ/wiki
- Resource management needed
- Interest increases with the advent of multicore chips

Acknowledgements

Thanks to

- Anne-Marie Chana (data-flow systems)
- Alain Darte (for his slide on the history)
- Steven Derrien (for his slides and hours of common work)
- Tanguy Risset and Florent de Dinechin (structured scheduling)
- Sanjay Rajopadhye (for nice discussions in Colorado)

Thank you for your attention !