The programmer's view of a dynamically reconfigurable architecture

Luciano Lavagno

Politecnico di Torino

lavagno@polito.it

Joint work with:

Fabio Campi, Roberto Guerrieri, Andrea Lodi, Claudio Mucci, Mario Toma

Universita' di Bologna

Francesco Gregoretti, Alberto La Rosa, Mihai Lazarescu, Claudio Passerone

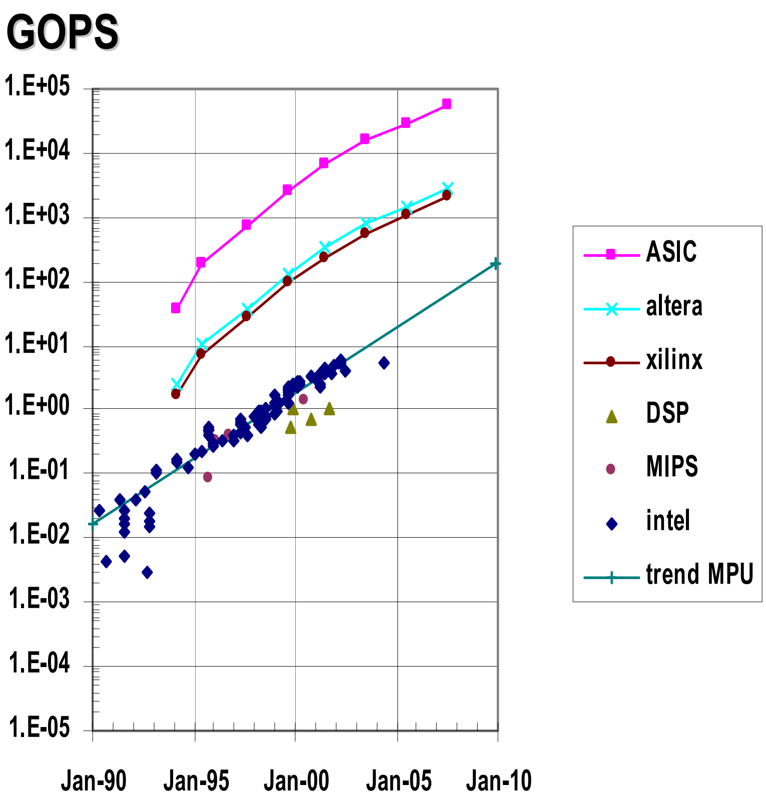
Politecnico di Torino

Outline

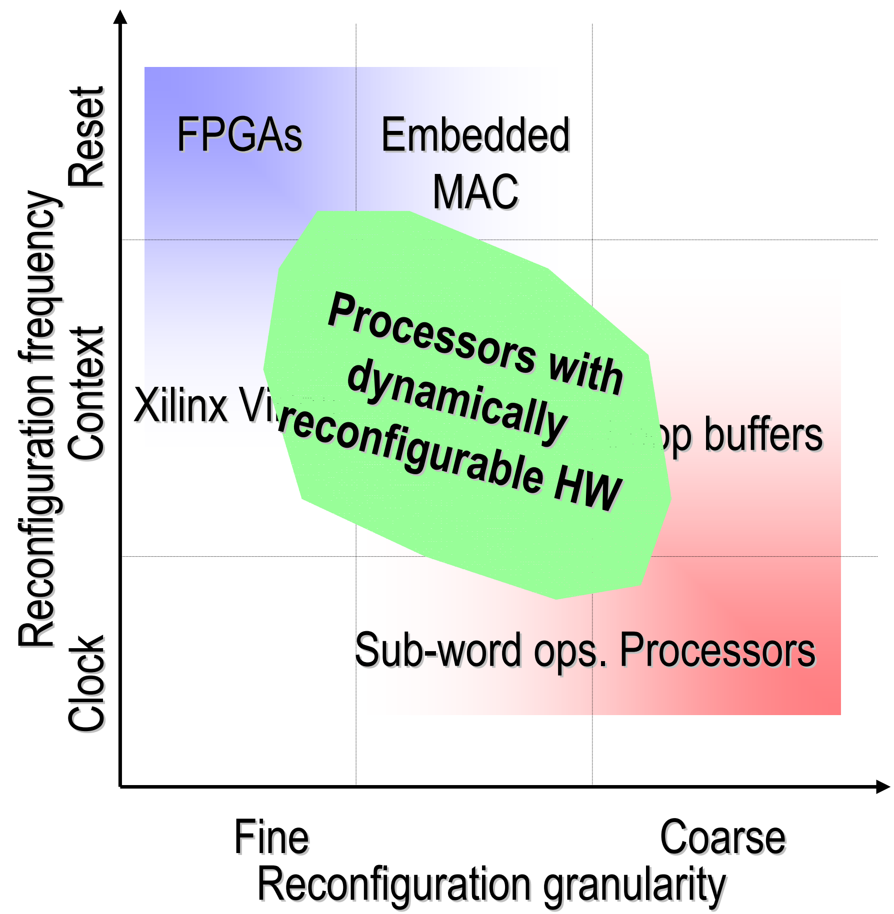
- Motivations
- The Target Reconfigurable Processor (XiRisc)
- Design Space Exploration
  - Design flow
  - Optimizations and limitations
- Turbo-decoder example
  - Memory optimizations
  - Dynamic instructions selection
  - Mapping
  - Experimental results
- Conclusions

Motivations

The reconfiguration landscape



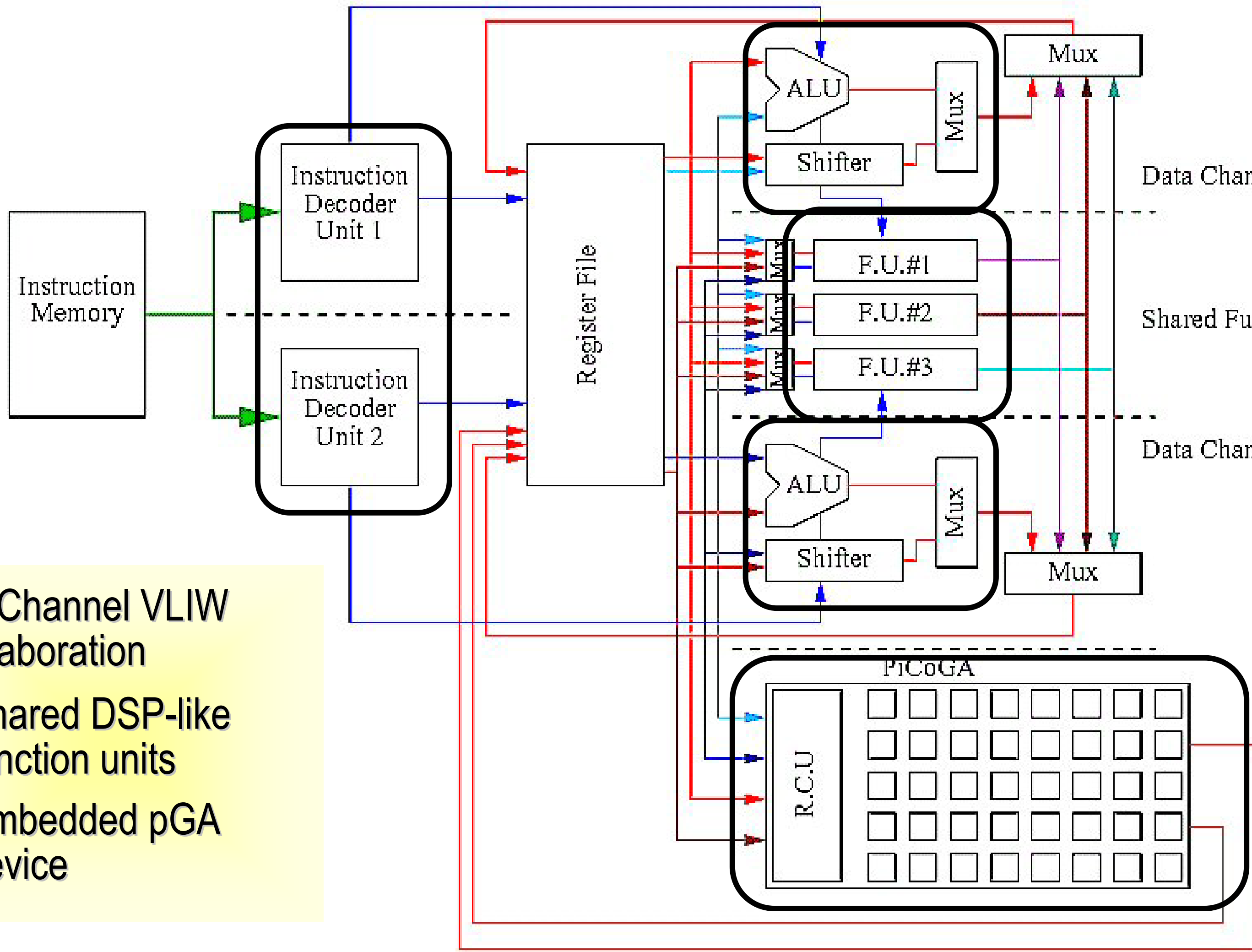
Source: Philips



Past work

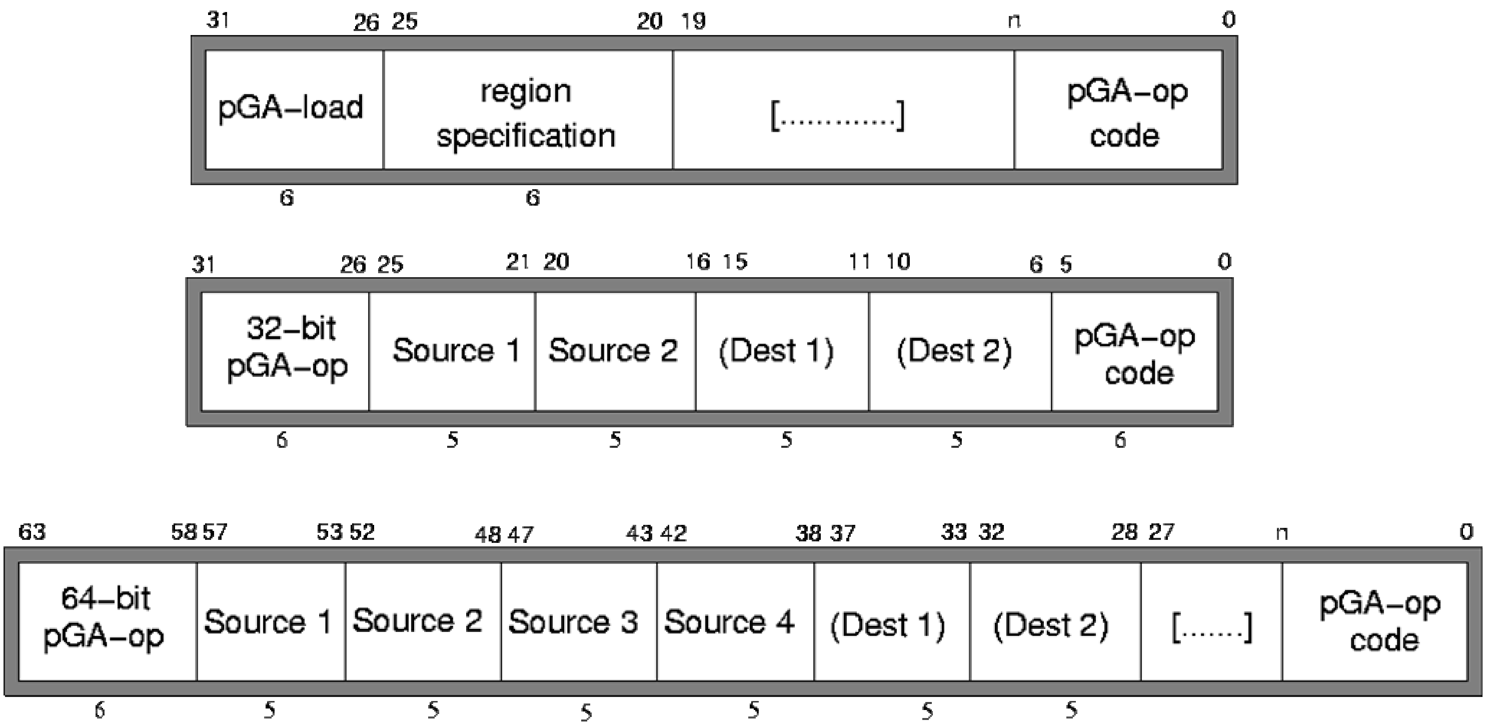
- Reconfigurable array as co-processor:
  - GARP (Callahan), Nimble compiler (Li)
- Reconfigurable array as functional unit:
  - Prisc (Razdan), Chimaera (Hauck), Concise (Kastrup)
- Key issues:
  - path to memory and I/O limitations (co-processor better)
  - ease of integration into ISA and compiler (FU better)
  - row-based architecture for good arithmetic op mapping
  - efficient HW synthesis onto non-standard architecture

The XiRisc Architecture

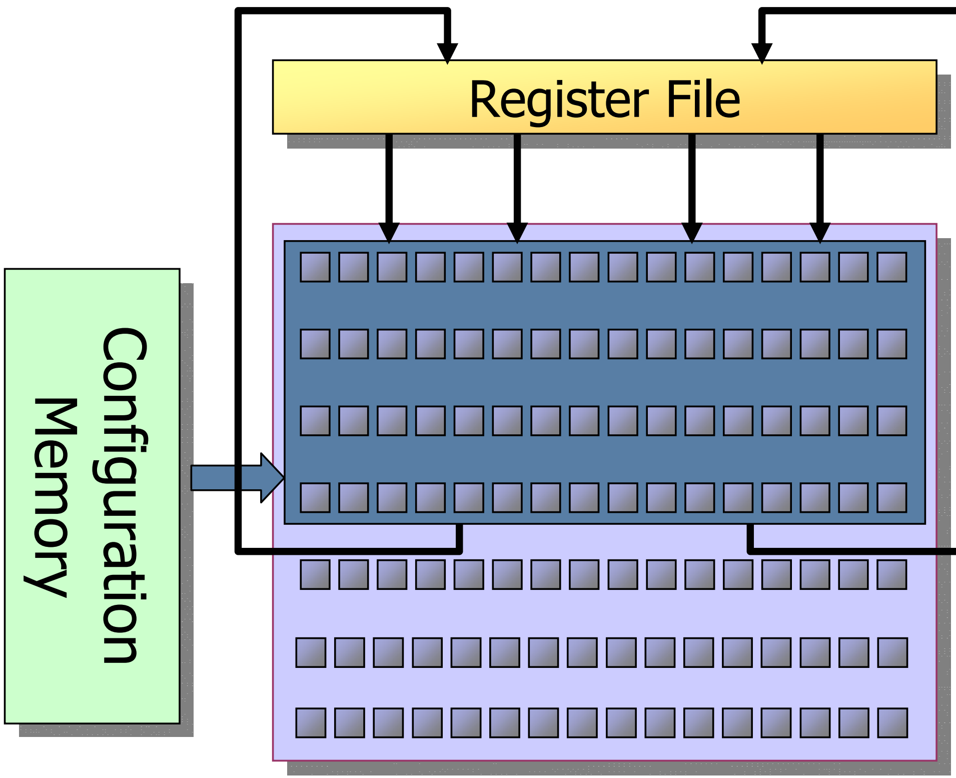


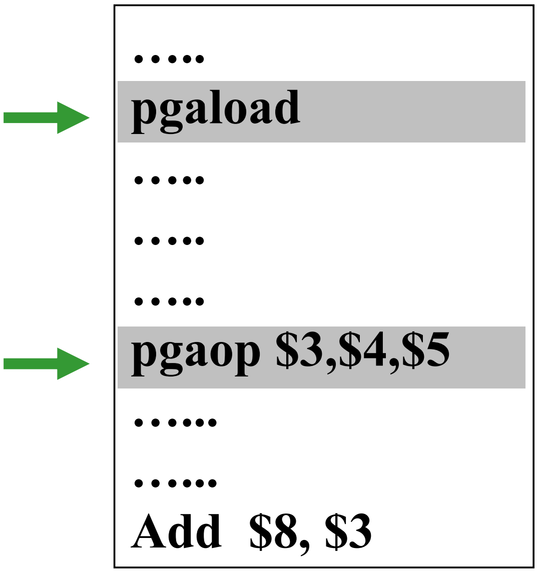
- 2-Channel VLIW Elaboration
- Shared DSP-like function units
- Embedded pGA device

Dynamic Instruction Set Extension

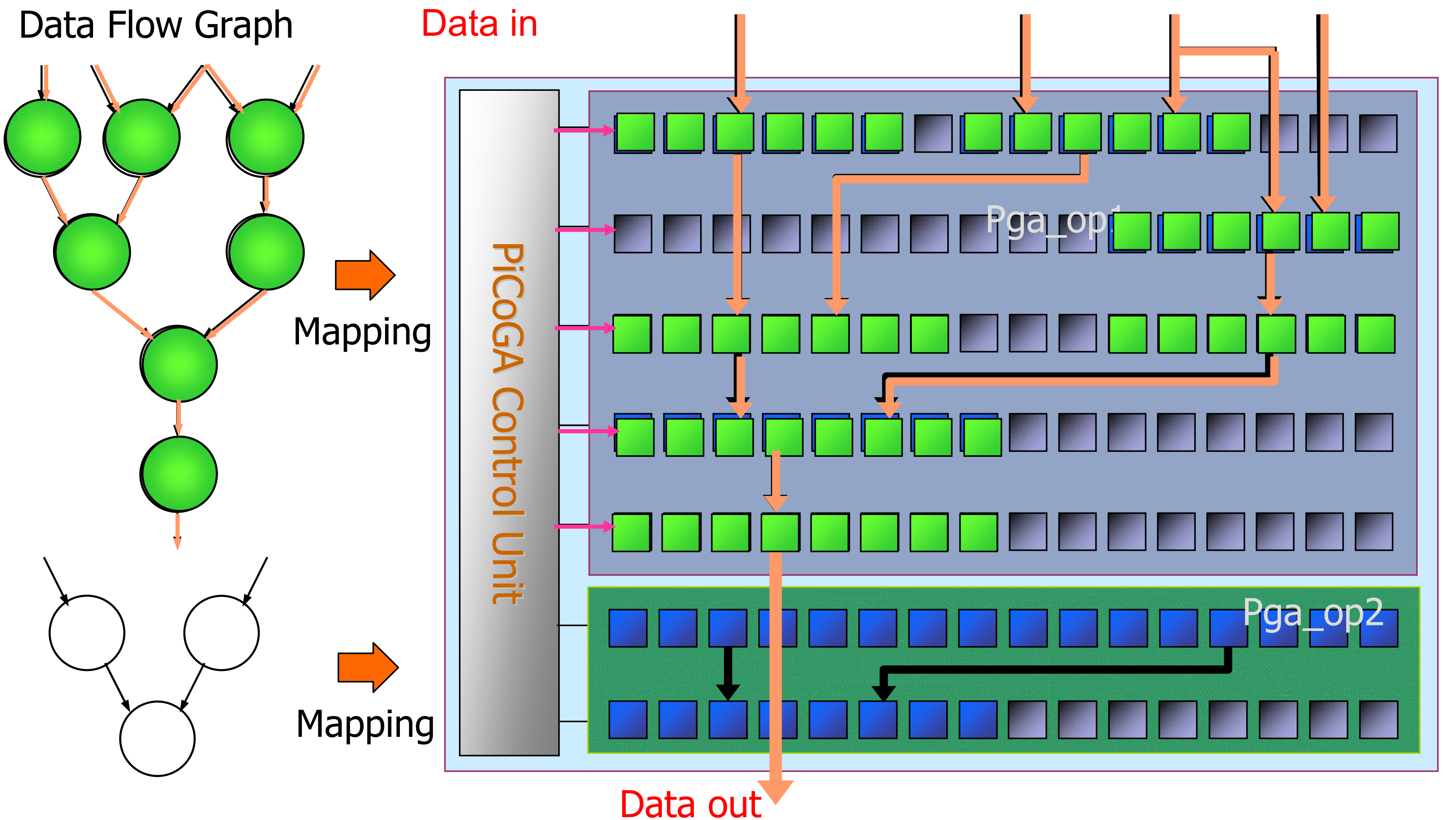


Dynamic Instruction Set Extension

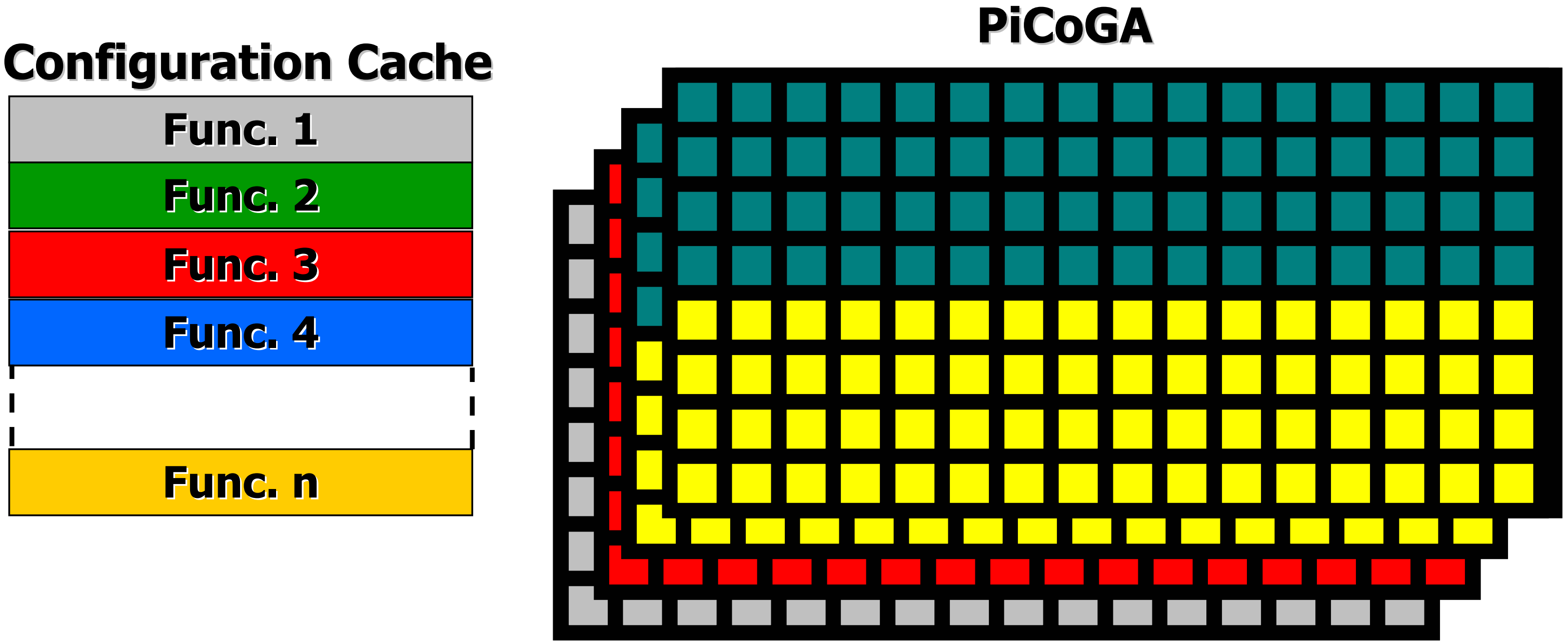




Computing on the PiCoGA



Multi-context Array

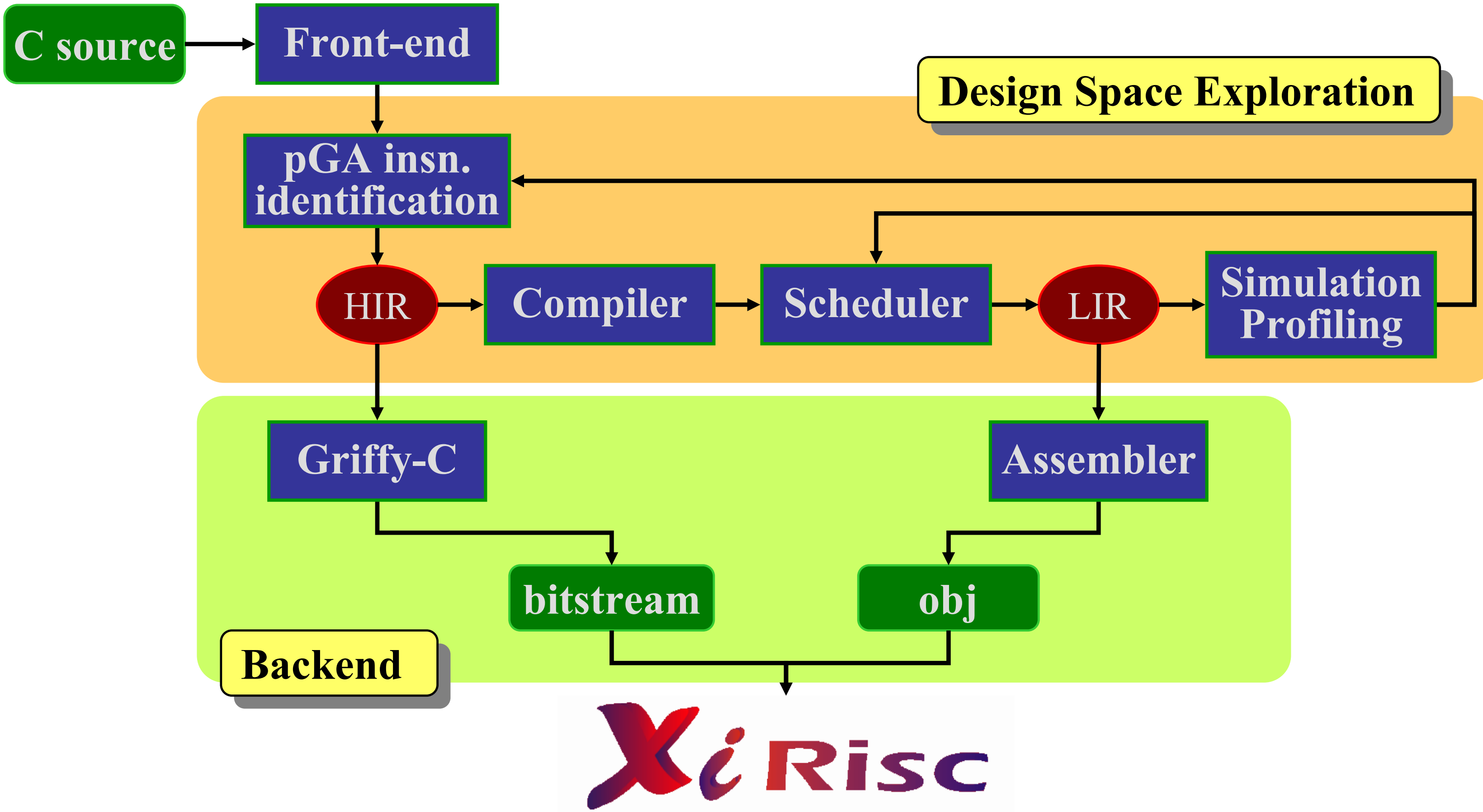


- Four configuration planes are available
- Plane switching takes one clock cycle
- While one plane is loading, others can work undisturbed

Design Space Exploration

- Software developer’s perspective:
  - Wants only the speed-up (cc -OR foo.c)
  - Does not want to see the architecture
- Reconfigurable processor compilers enable the transparent use of the reconfigurable instruction set via:
  - Pseudo-function calls (“intrinsics”)
  - Language extensions (pragmas)
- Design flow:
  - Identify compute intensive kernels
  - Group instructions into sets of user-defined pGA instructions
  - Use cost figures to compare costs and performance of different HW/SW partitions
  - Refine cost figures by manual or automatic synthesis

XiRisc Design Flow



Manual pGAop identification: example

```c
int bar (int a, int b) {
  int c;
  #pragma pgaop sa 0x12 5 1 2 c a b
  c = (a << 2) + b;
  #pragma end
  return c + a;
}

main() {
  i = bar(2,3);
  return;
}
```

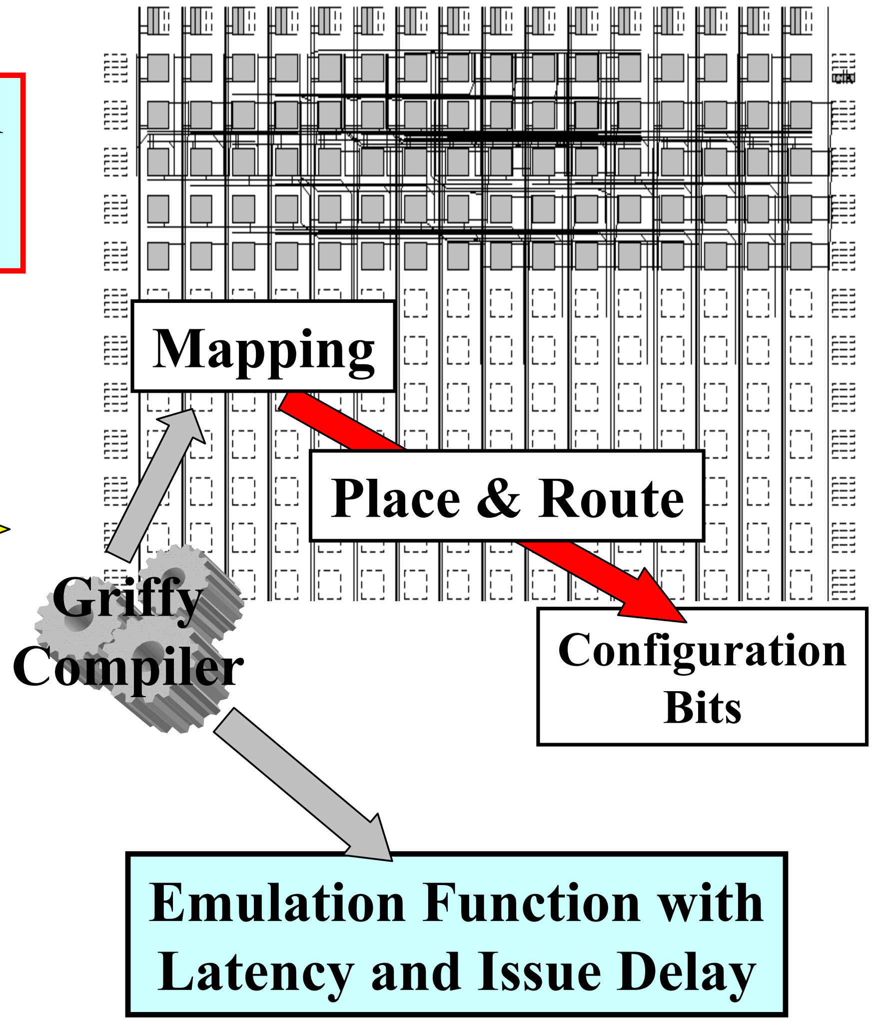
```c
int i;
int bar (int a, int b) {
  int c;
#if defined(PGA)
  asm ("pga5 0x12,%0,%1,%2":"=r"(c):"r"(a),"r"(b));
#else
  asm ("topga %1, %2, $0"::"r"(a),"r"(b));
  asm ("jal _sa");
  asm ("fmpga  %0, $0, $0": "=r"(c): );
#endif
  return c + a;
}
...
#if !defined(PGA)
void _sa () {
  int c,a,b;
  asm("move %0,$2;move %1,$3": "=r"(a),"=r"(b):"r"(c):
      "$2","$3","$4");
  c = (a << 2) + b;
  /* delay by 5 cycles */
  asm("move $2,%0; li $4,5": : "r"(c) : "$2","$3","$4"
);
}
#endif
```

Back-end

High-Level C Compiler

- DFG-based description
- Single Assignment
- Manual Dismantling

```
#pragma pga SAD4 1 2 out p1 p2
{

  unsigned char p10, p11, p12, p13;
  unsigned char cond0, cond1, cond2, cond3;

          ...

  #pragma attrib cond0, cond1, cond2, cond3 SIZE=1

          ...

  sub0a = p10 - p20;  sub0b = p20 - p10;
  cond0 = sub0a < 0;
  sub0  = cond0 ? sub0b : sub0a;

          ...

  out = acc1 + acc2;

}
#pragma end
```



Design Space Exploration

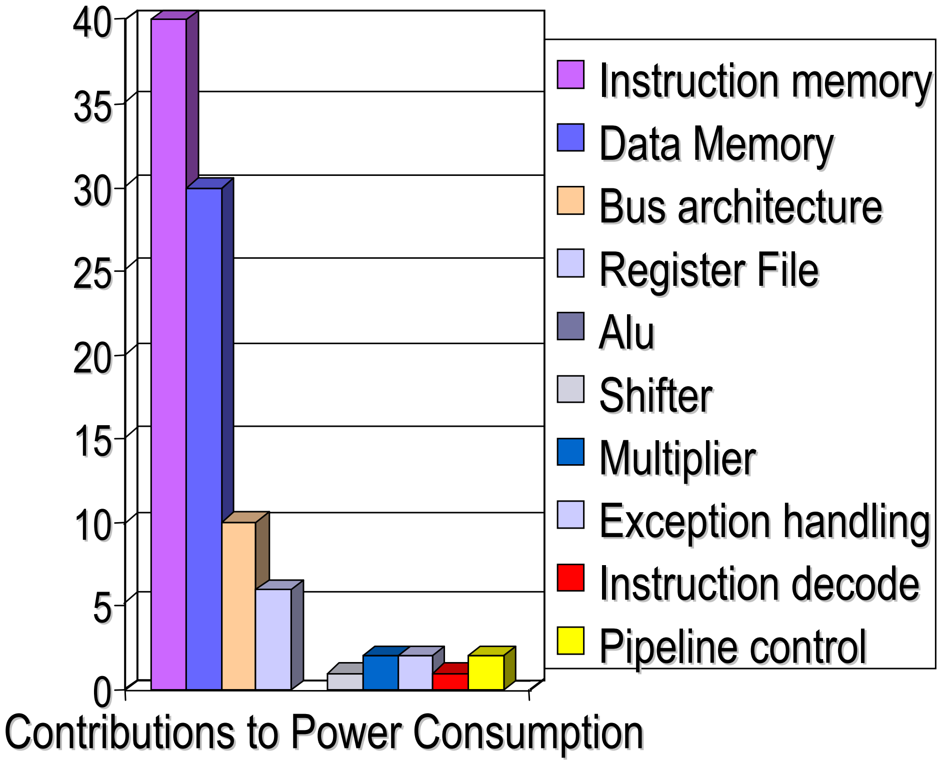
Optimizations for the Reconfigurable Array

Increase Performance

- Increase concurrency
- Minimize memory accesses
- Customize data-width
- Optimize data structures

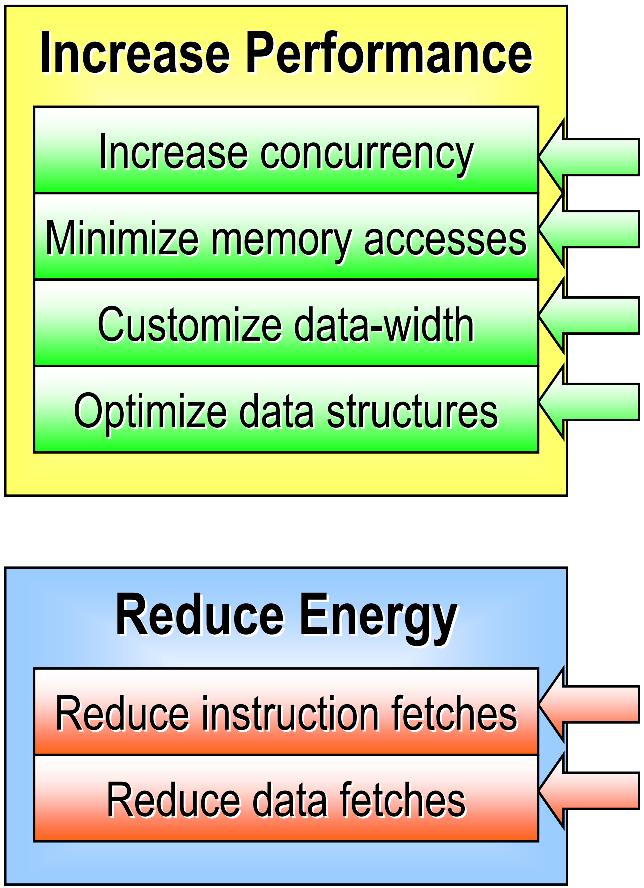
Reduce Energy

- Reduce instruction fetches
- Reduce data fetches



Design Space Exploration

Optimizations for the Reconfigurable Array



- Exploit concurrency
  - within the reconfigurable array
    - horizontally: operate on multiple data
    - vertically: pipelined implementation
  - with respect to the standard data-path
- Optimize data memory
  - internal storage reduces register spills
  - reordering and shifting are free
  - pack data into a single word (SIMD operation)
- Optimize instruction memory
  - reduced instruction fetches

Design Space Exploration
Limitations of the Reconfigurable Array

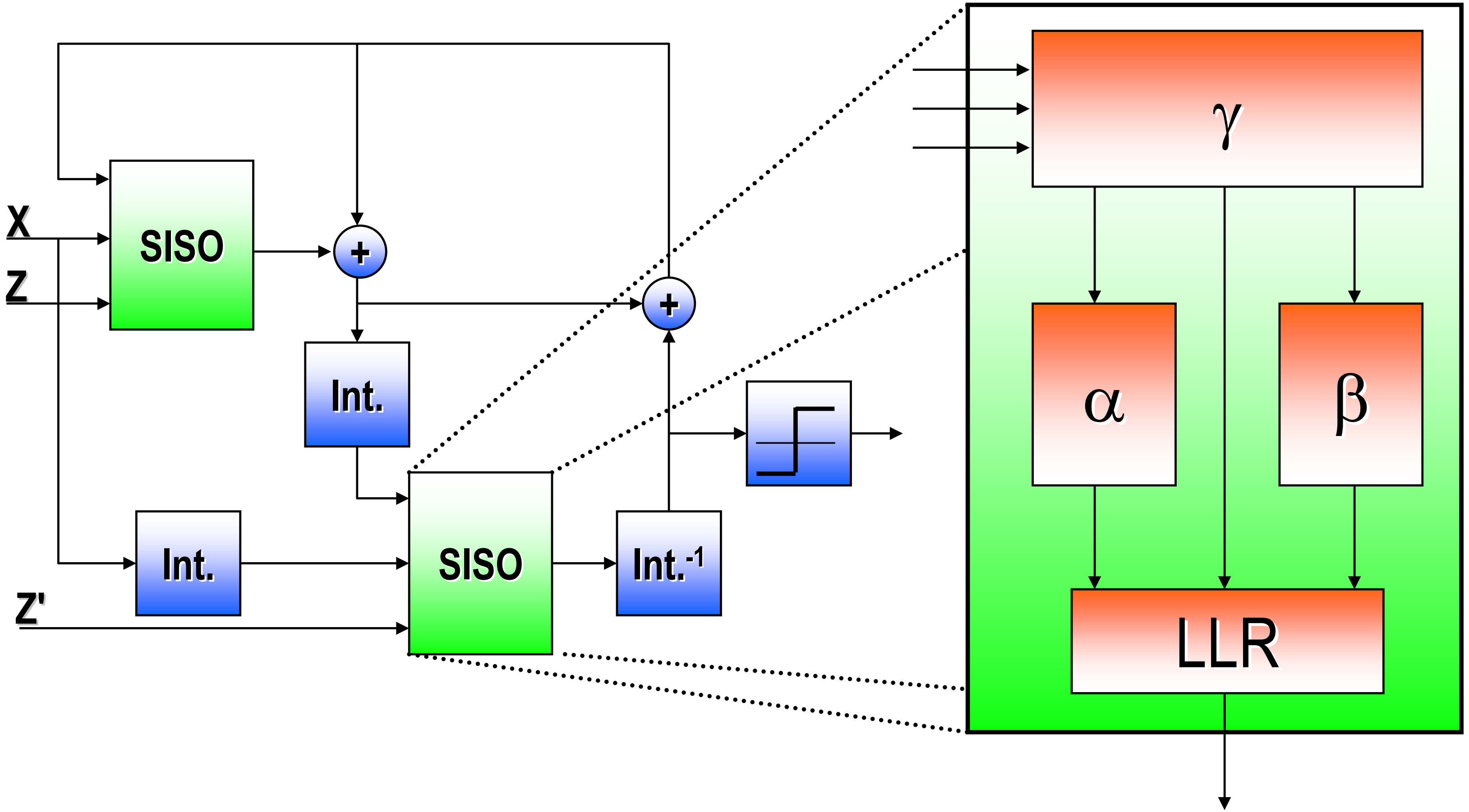
- No direct access to memory
  - processor memory access unit is a bottleneck
- Finite number of read/write register ports (operands)
  - 4 read, 2 write
- Finite chip area
- Number of custom instructions
- Reconfiguration time
  - 4 configuration caches
- Limited control flow
  - can implement data dependent loops and if-then-else

UMTS Turbo-decoder

- UMTS (3GPP) Turbo Code Specification:
  - 8 states trellis RSC $(1,15/13)_8$ Rate=1/3
  - variable frame size $40 \leq K \leq 5114$
- BCJR algorithm
  - Max log-Map + linear correction
  - 16 bit fixed point precision
- BPSK modulation along with AWGN channel

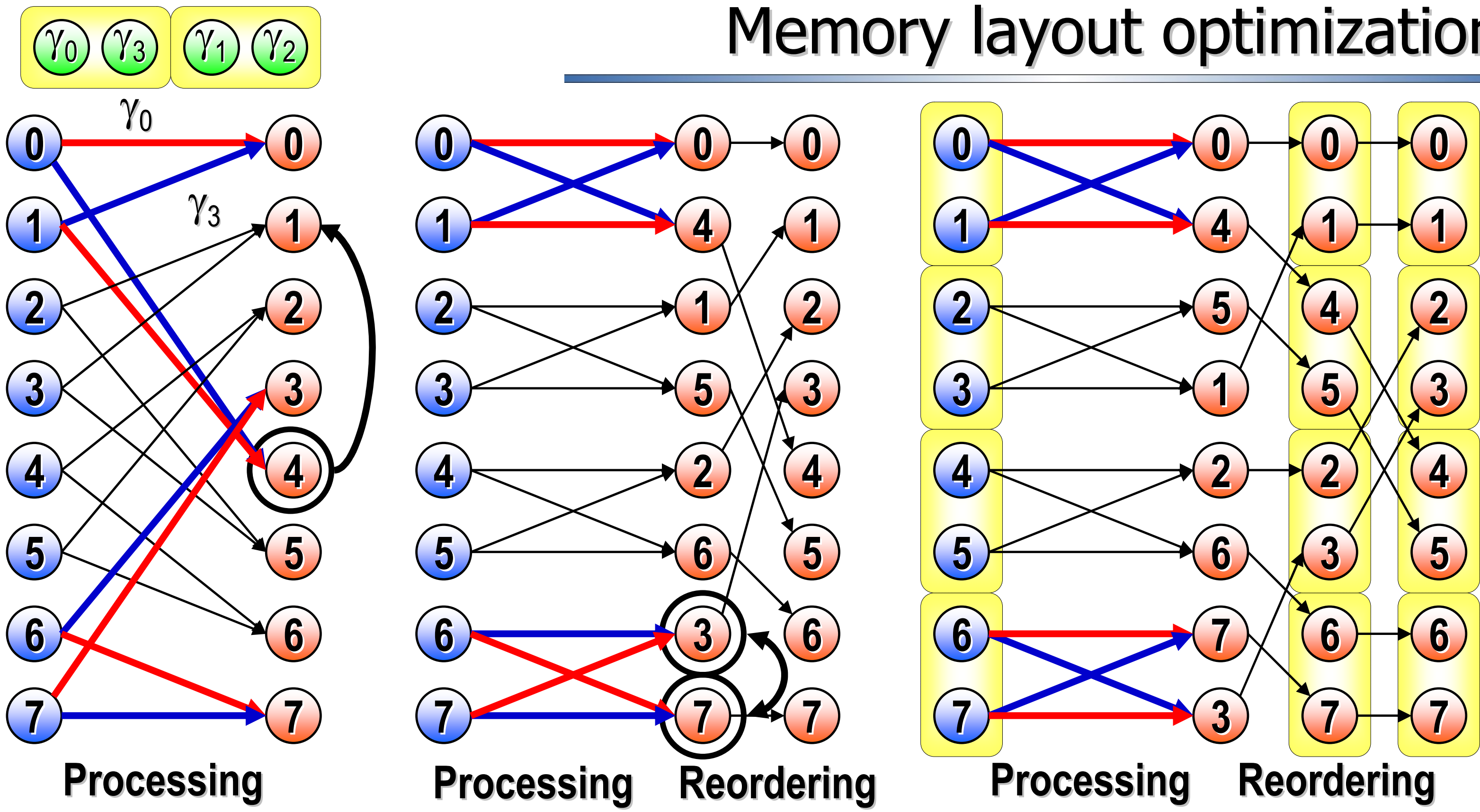
UMTS Turbo-decoder

Block diagram

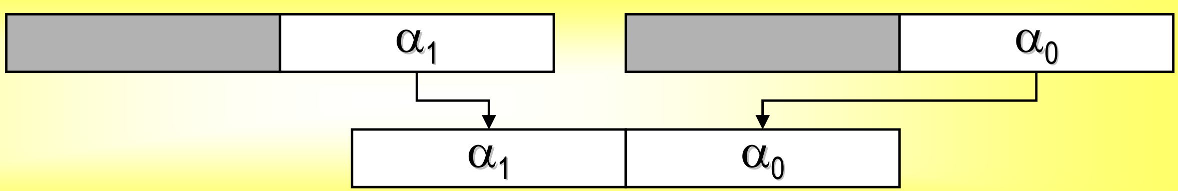


UMTS Turbo-decoder

Memory layout optimizations

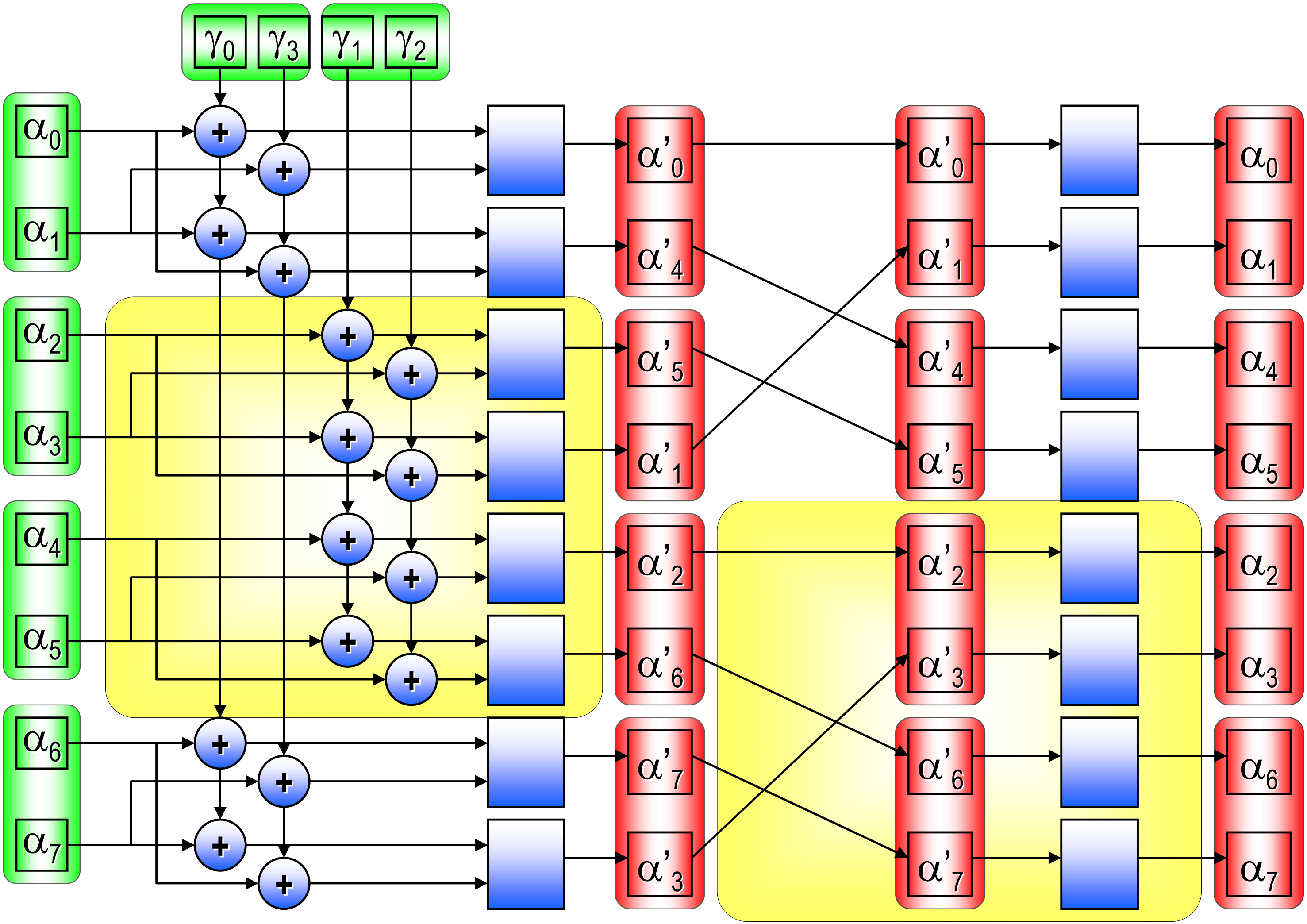


Data width and Packing



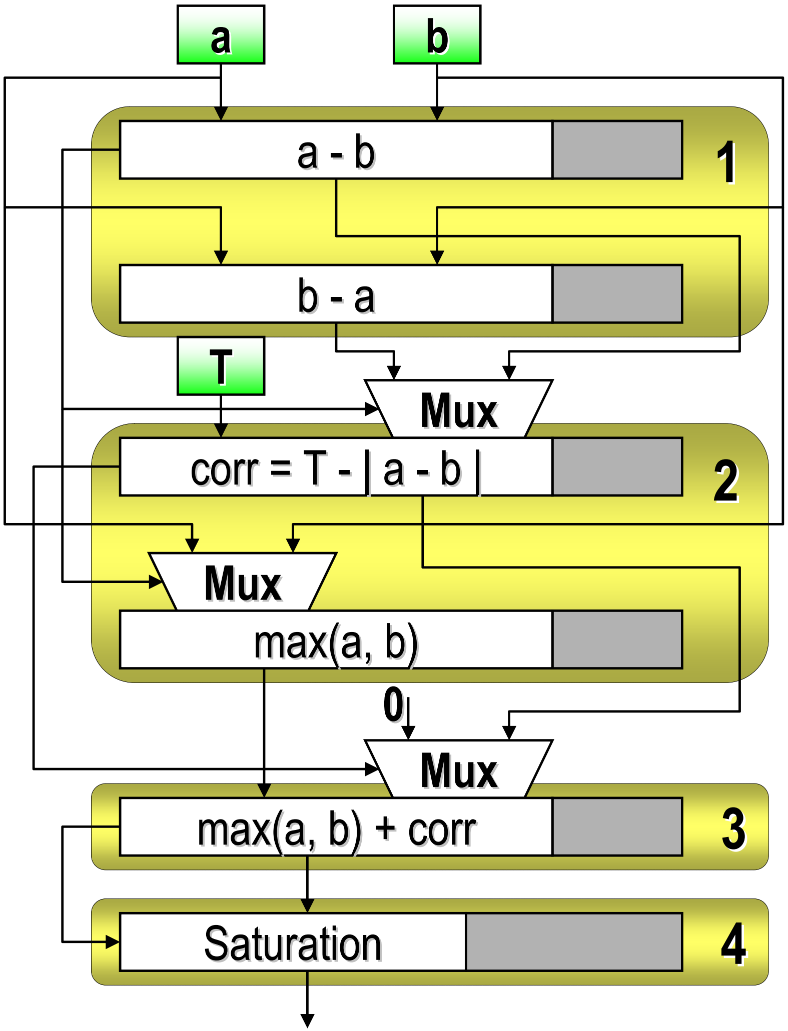
UMTS Turbo-decoder

pGA Instruction Selection



UMTS Turbo-decoder

Mapping pGA instructions



max*: $y = \ln(e^a + e^b)$

2 segment piece-wise linear approx.

if (abs(a-b) ≤ T) corr = (T - abs(a - b));
else corr = 0;
return (max(a, b) + corr);

- Speculative execution
- Two concurrent max*
  - input/output data packed
- Pipelined implementation
  - latency: 4 + 1 clock cycles
  - issue delay: 1 clock cycle

Experimental Results
UMTS Turbo Decoder K=40 single iteration

	Step	Speedup	Execution Cycles	Saved cycles
	Original	1x	177834	-
Profile	Gamma (1)	1.02x	173706	4128
	LLR (2)	1.83x	96913	80921
	Butterfly (3)	1.53x	115816	62018
	Reorder (4)	1.10x	161826	15972
Estimation	(1) + (2)	1.91x	92785	85049
	(1) + (2) +(3)	5.78x	30767	147067
	(1) + (2) + (3) + (4)	11.90x	14795	162907
Final	(1)+(2)+(3)+(4)	**11.73x**	**15157**	**162677**

Experimental Results: Speed-up and Energy Reduction

Algorithm	Energy reduction (vs. std. XiRisc)	Speed-up (vs. std. XiRisc)
DES encryption	89%	13.5x
Turbo decoder	75%	11.7x
Motion estimation	80%	10x
Median filter	60%	7.7x
CRC	49%	4.3x

As a comparison, Tensilica achieves

- 10X better speed-up (50-100X) on similar examples
- using 10X more gates (i.e. similar area)
- without the reconfiguration flexibility

Conclusions

- Reconfigurable computing dramatically speeds up highly data and control intensive applications
- Traditional software and hardware design flows do not support reconfigurable architectures
- Software oriented design flow:
  - fast exploration of HW and SW alternatives
  - no detailed knowledge of the underlined architecture
- Future work
  - automatic kernel extraction
  - fully automated path to implementation