# A scalable wide-issue clustered VLIW with a reconfigurable interconnect

Osvaldo Colavin – Davide Rizzo

STMicroelectronics – San Diego

# Overview

- Motivations
- Context
- Contributions
- Architecture
  - Runtime reconfigurable inter-cluster bus
  - Software pipelining architectural support
- Compilation
- Scheduling example
- Experiments
- Conclusions

# Motivations

- Reduce the recourse to hardwired accelerators in embedded systems
  - Provide a programmable fabric with significant parallelism (10s of operations in parallel)
- Keep it simple
  - The design envelope for embedded systems is incredibly tight
- Compiler driven architecture
  - Programmable in C (no exotic language, no RTL)
  - Essential for programmer's productivity

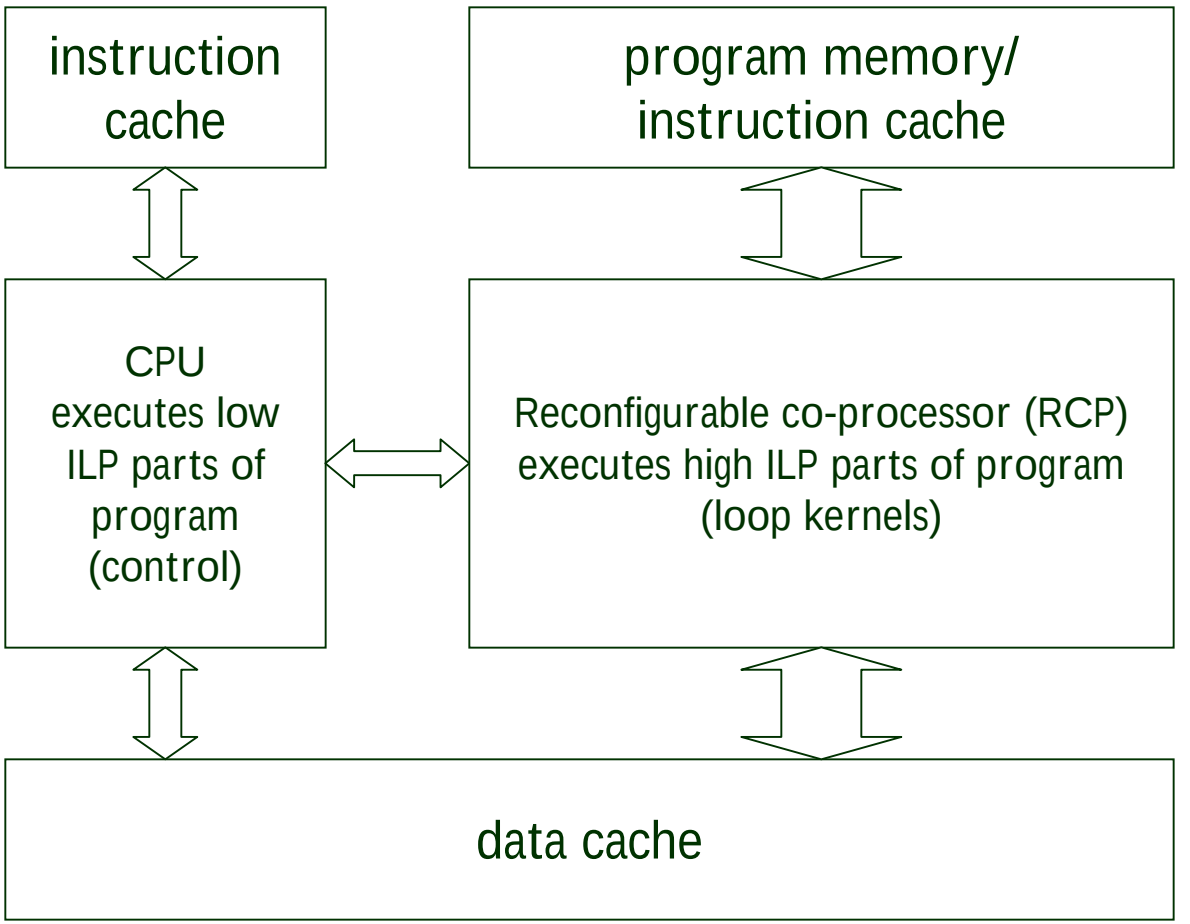
# Context

- Clustered VLIW (Equator, TI C6x, ST200, ST100)
  - \+ Familiar programming model; mature compiler technology
  - – Limited computing fabric scalability due to inter-cluster communication overhead
- Reconfigurable architectures (Chameleon, QuickSilver, MorphoTech, MathStar, PACT; GARP, Rapid, PipeRench)
  - \+ Lots of parallelism
  - – Poor programming model (reconfiguration, low-level abstraction)
  - – Scalability limited by global resources (usually interconnect)
  - – Cycle time known after P&R
  - – Poor area/power merit figure

# Contributions of this work

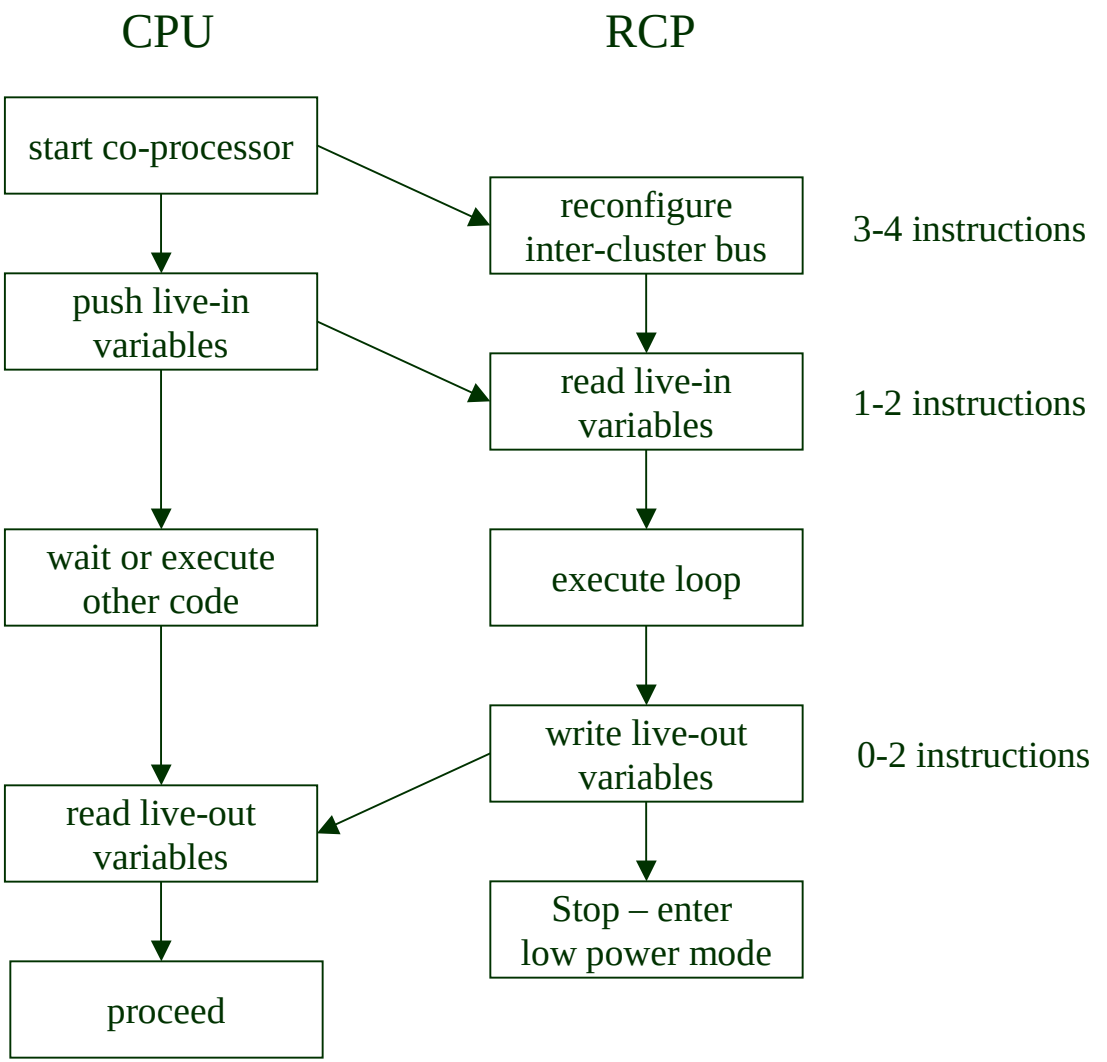
- Truly scalable computing fabric
  - clock cycle time is independent of the # of clusters
  - Inter-cluster transfer latency is constant
  - Area is a linear function of number of clusters
- Compiler friendly runtime reconfigurable architecture
  - Adapted version of modulo scheduling for clustered VLIW
  - Original forms of predication and register file architecture for software pipelining support
- Limited reconfigurable architecture side-effects
  - Programmed like a VLIW, not like a FPGA
  - Reconfiguration takes a few cycles (<10)

# General purpose CPU coupled to a wide issue width co-processor



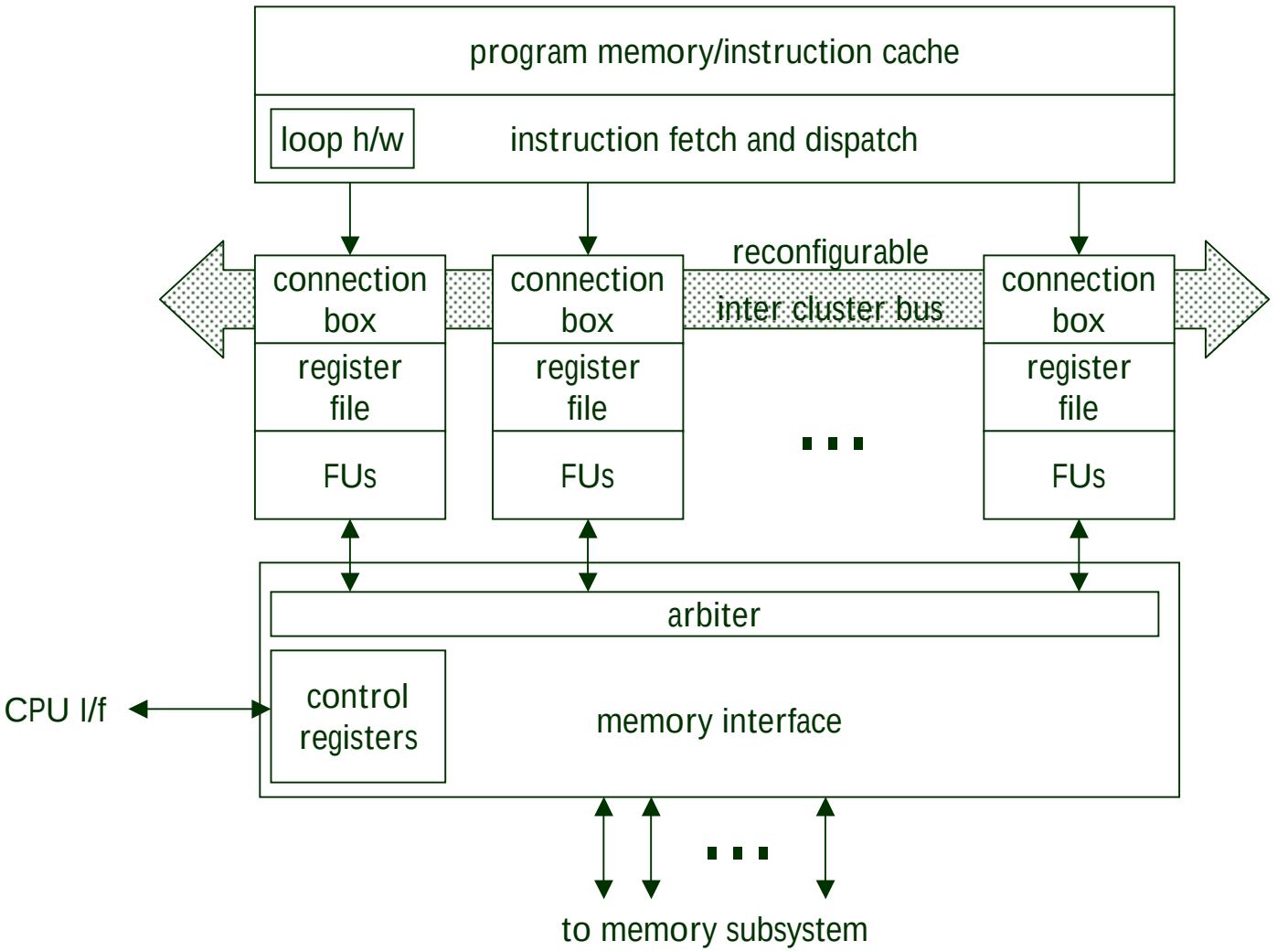

- Compiler identifies loops to be executed on RCP
- Programmer's view is that of a single machine

# Execution model




- Relatively little RCP overhead
- Reconfiguration requires only few instructions

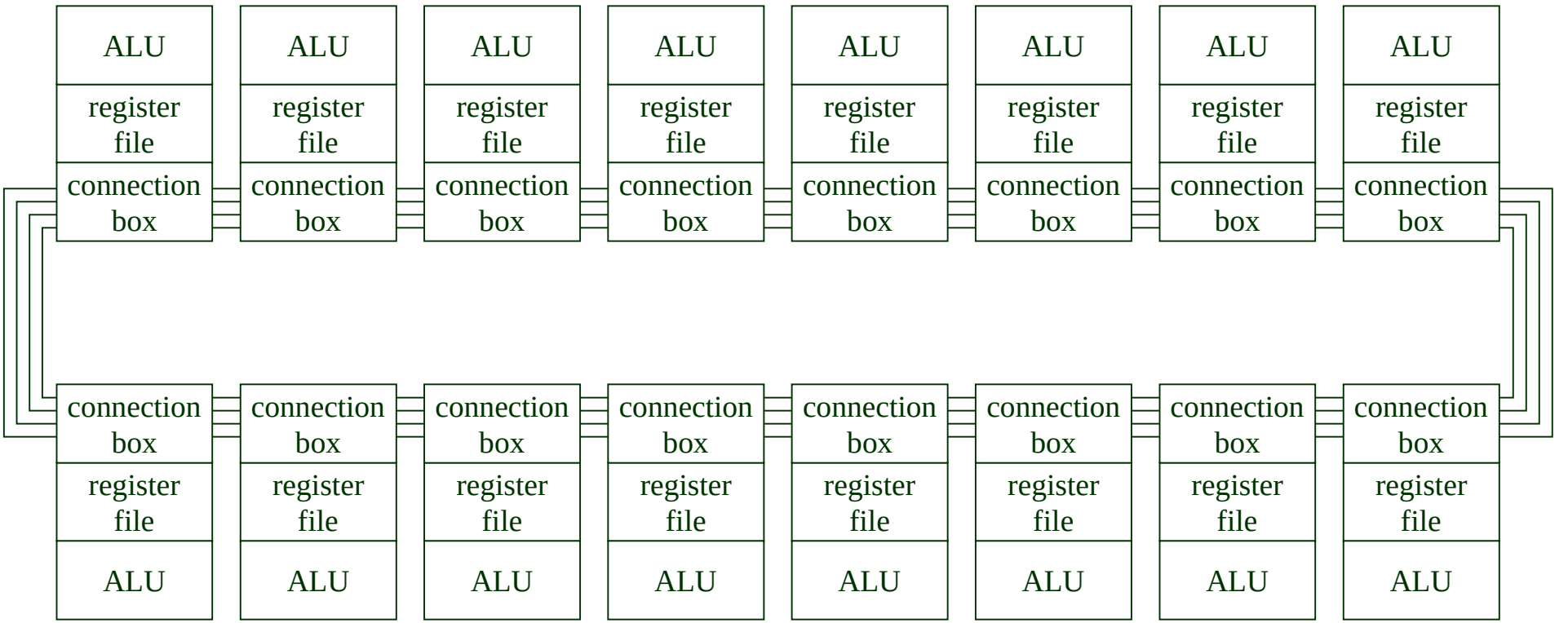
# Runtime reconfigurable co-processor



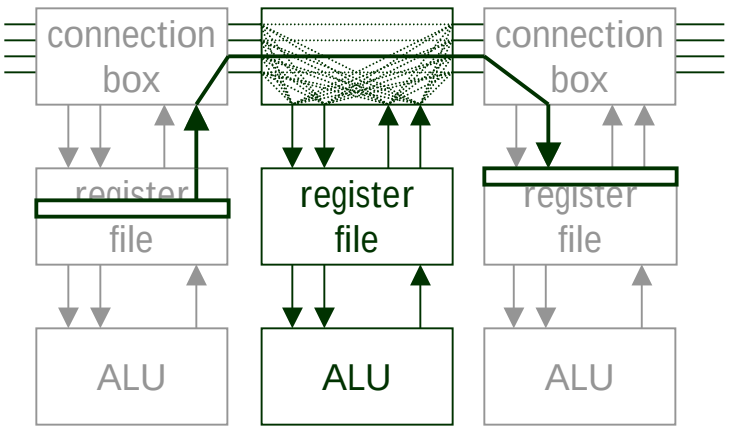
# RCP parameters considered

- 1issue/cluster, 16 registers/cluster
- Each register file has
  - 4 (2R/2W) ports dedicated to inter-cluster bus connections
  - 3 (2R/1W) ports for locally executed instruction
- Inter-cluster bus
  - Has a ring topology
  - Is composed of 4 segmented buses

# Example: 16-cluster RCP



# Connection box

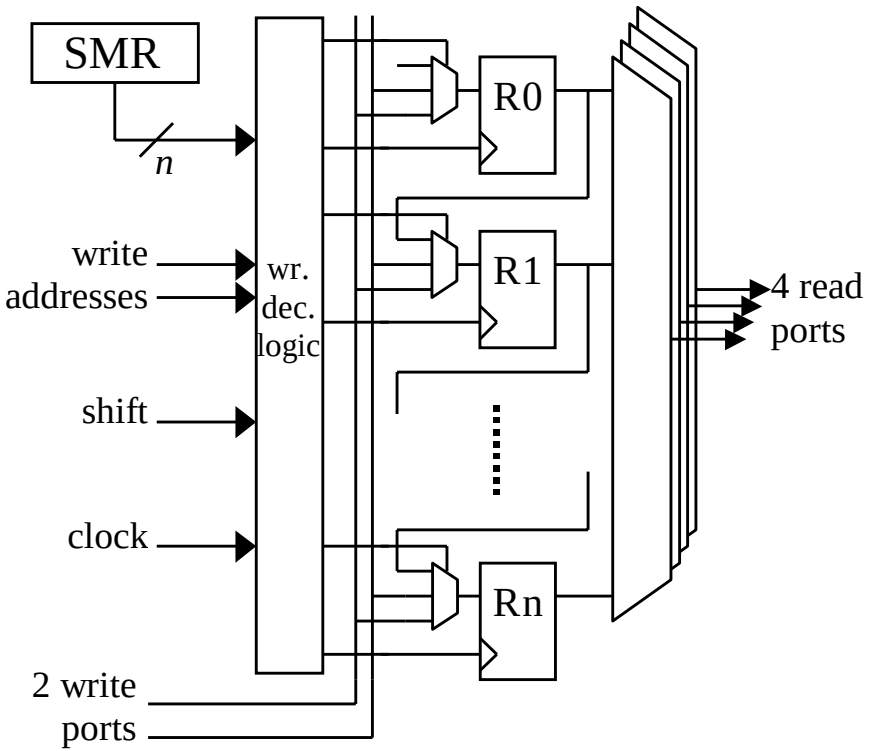



- Reconfiguration instructions connect register file ports to interconnect resources
- They establish connections between specific registers inside register files which remain until next reconfiguration
- The span of a connection is limited by the cycle time; it is the compiler's responsibility to respect this constraint

# Architectural support for software pipelining

- Renaming Register File
  - Reduces code size by obviating the need to unroll the loop kernel (modulo variable expansion)
- Predicated execution
  - Increases the number of loops candidate for software pipelining by converting control-flow to data-flow
  - Reduces code size of software pipelined loops by obviating the need of prolog and epilog code

# Shifting register file




- A register file in which queues of consecutive registers can be defined
- The Shift Mask Register is the architected register defining the queues
- A queue shifts when a particular signal is asserted (shift)
- In the RCP, the destination of an inter-cluster transfer is typically the head of a queue

# Operand predicates

- Add a predicate bit to each register
- A result's predicate is the logical AND of its source predicates
  - Intuitively, it is a dataflow semantic
- Formally,

```
op dst src1 src2
```

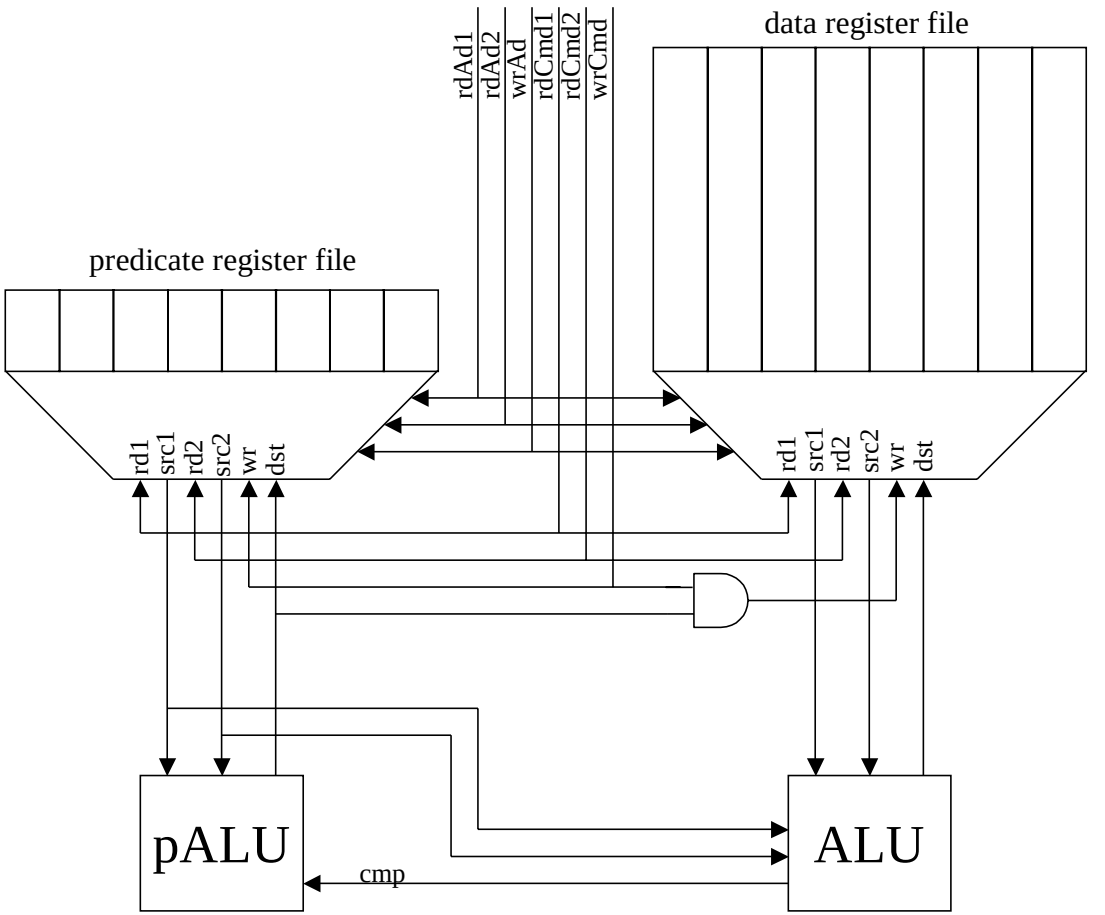
is equivalent to

```
dst.p = scr1.p && src2.p
dst = src1 op src2
```

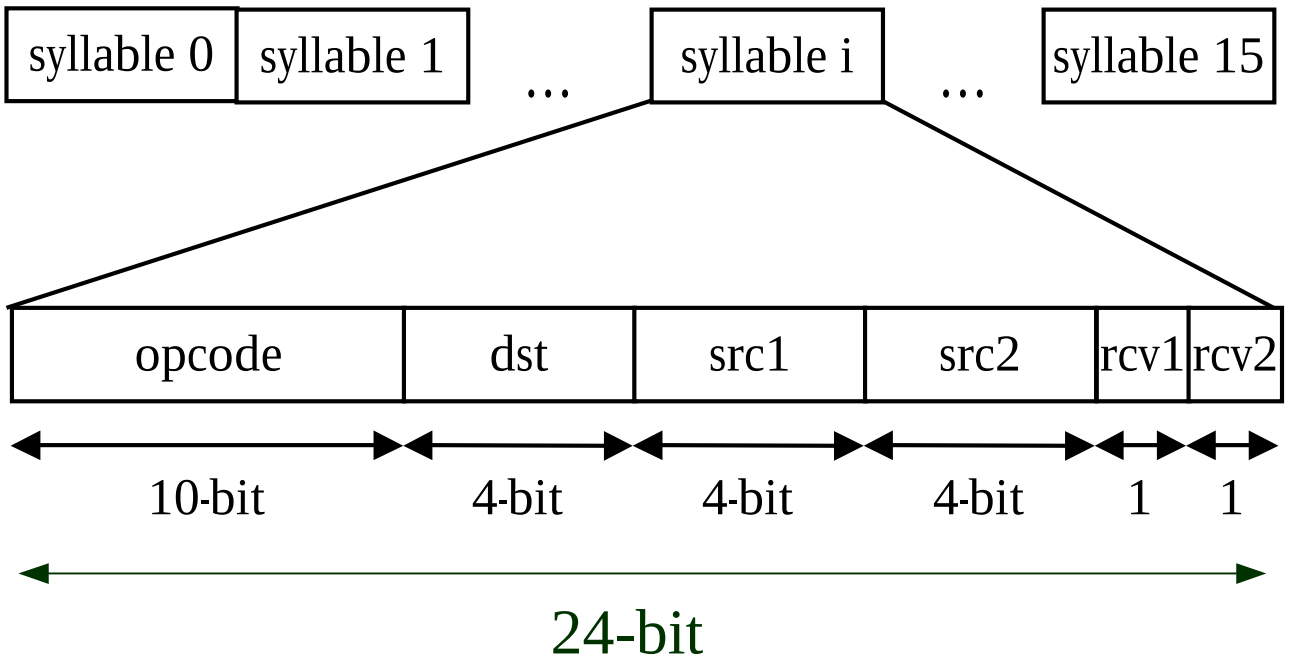
- memory operations are predicated

| | |
|---|---|
| `ld dst src`<br>is equivalent to<br>`dst.p = scr.p`<br>`dst = [src] if dst.p` | `st src1 src2`<br>is equivalent to<br>`[src1] = src2 if src1.p && src2.p` |

# Architectural support for operand predication



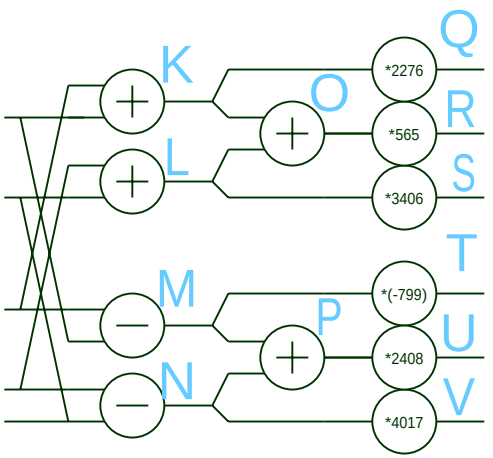
# Instruction format

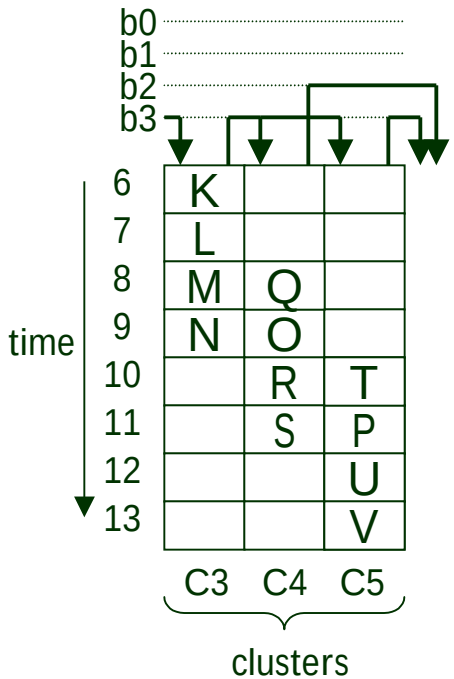


# Compiling for the RCP

- Software pipelining of inner loops
  - Use if-conversion to increase number of candidates
- Adapted version of modulo scheduling for clustered VLIW
  - Inter-cluster communication resources are statically allocated by the compiler
  - When a schedule fails because of limited inter-cluster communication resources, increase II
- Can be built on top of existing VLIW compiler infrastructure

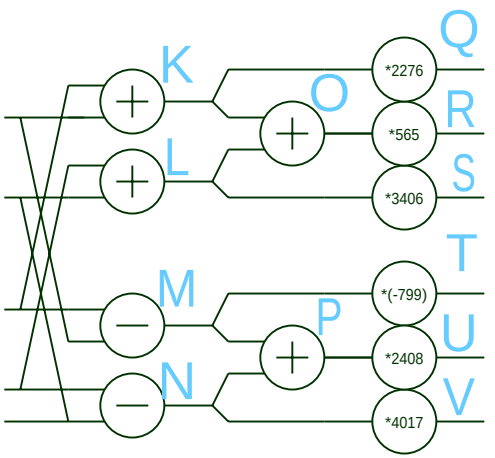
# Scheduling example using a portion of the DCT data flow graph – II=4

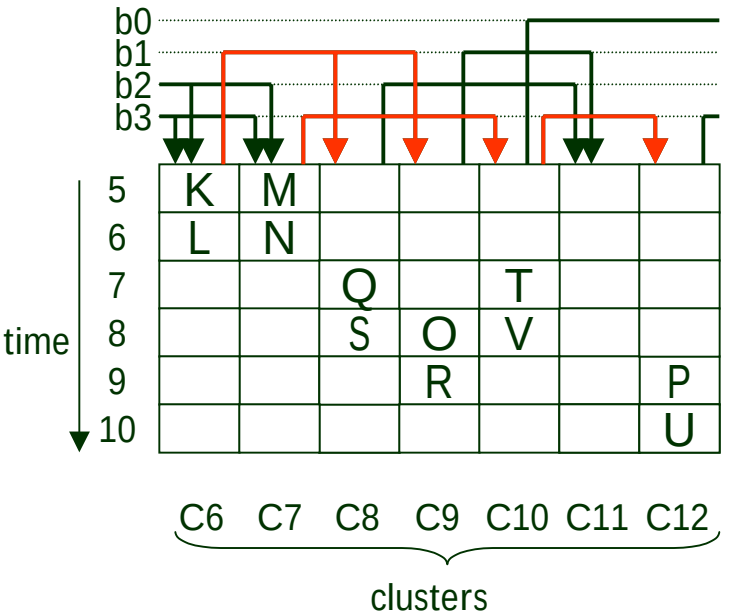


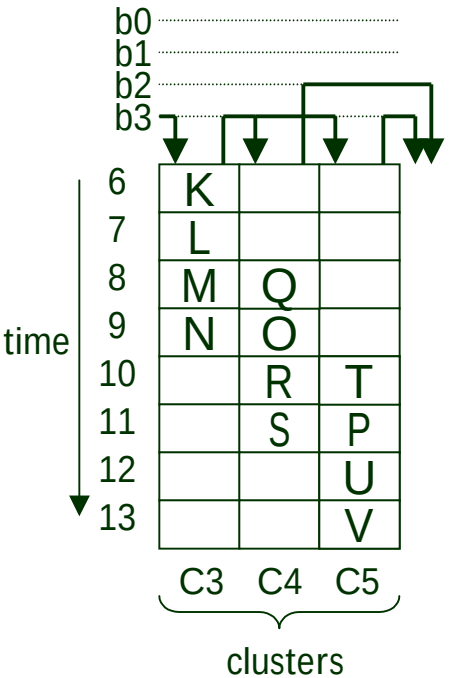




- modulo scheduling with II=4
- inter-cluster latency is 1 cycle
- interconnect resources are exposed to and allocated by the compiler
- example shows time multiplexing of same bus (b3) to copy data from C3 to C4 (cycles 7,8) and then from C3 to C5 (cycles 9,10)

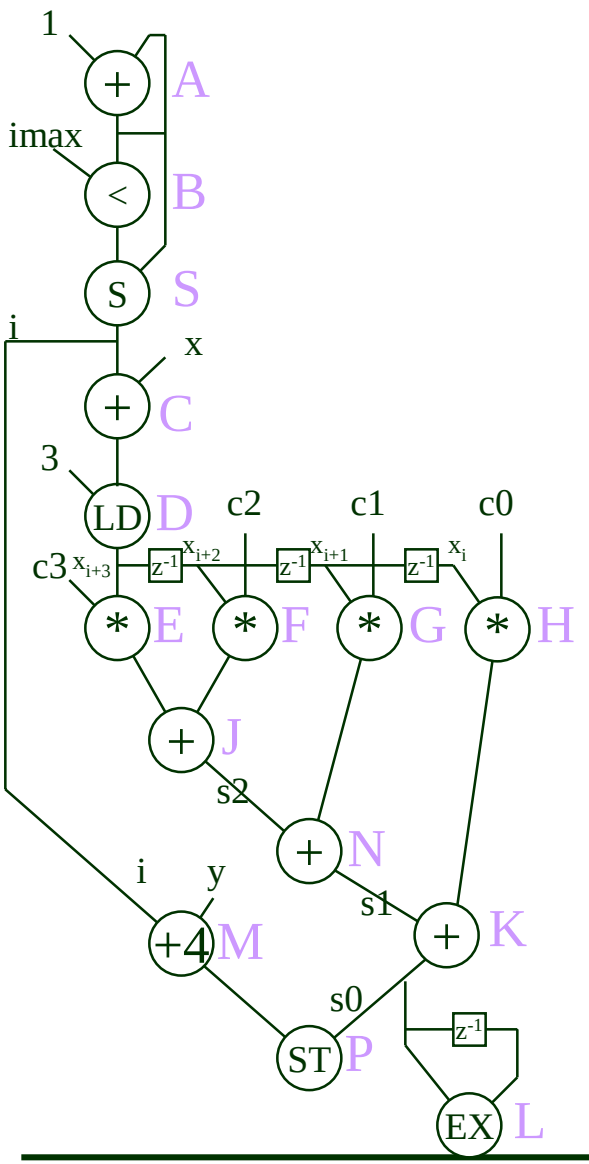
# Scheduling example using a portion of the DCT data flow graph – II=2

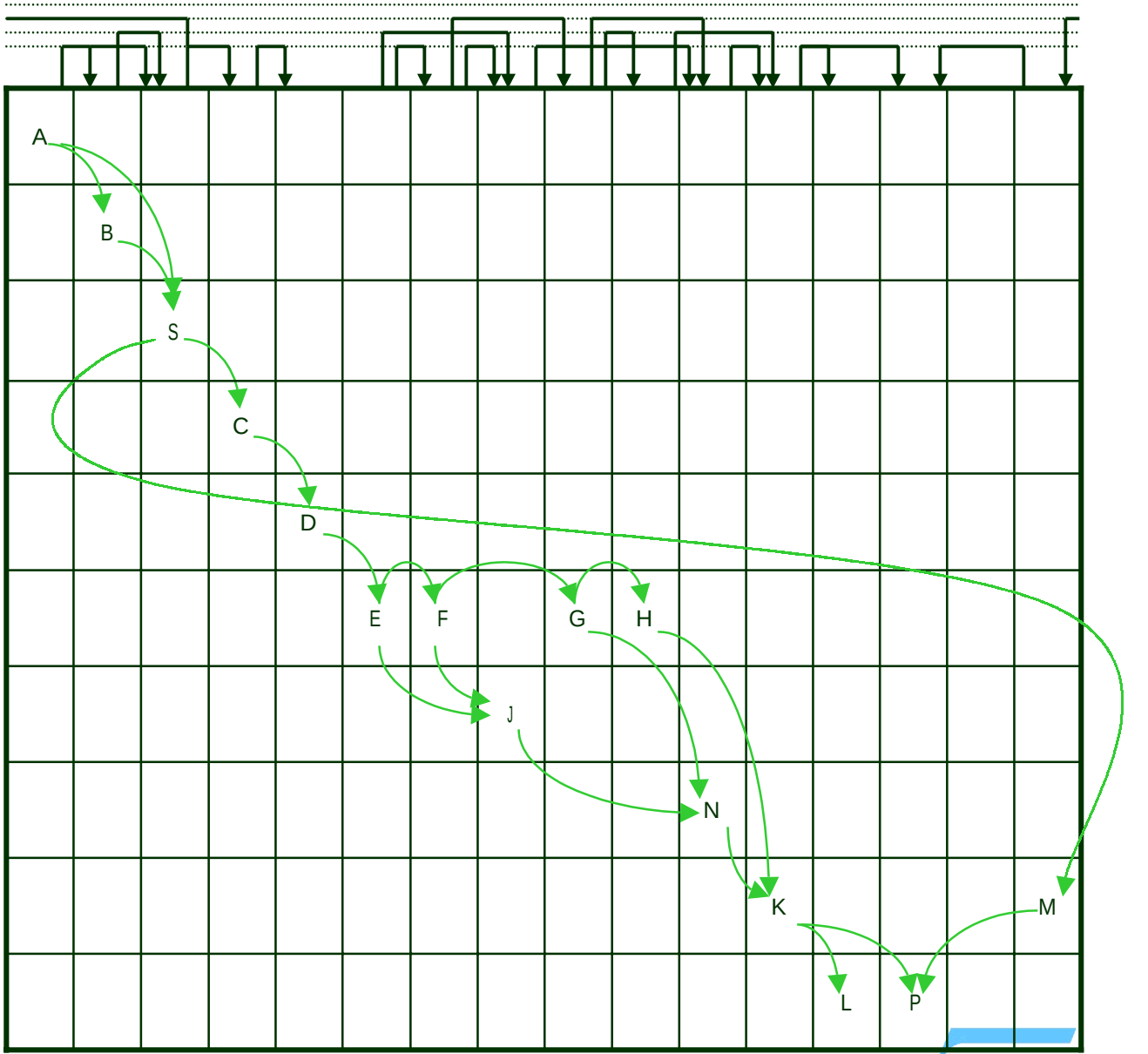


# 4-tap FIR scheduling example – II=1



# 4-tap FIR example loop kernel


- Only one VLIW instruction for the kernel
- ILP=15
- The loop is effectively vectorized

# Experimental setup

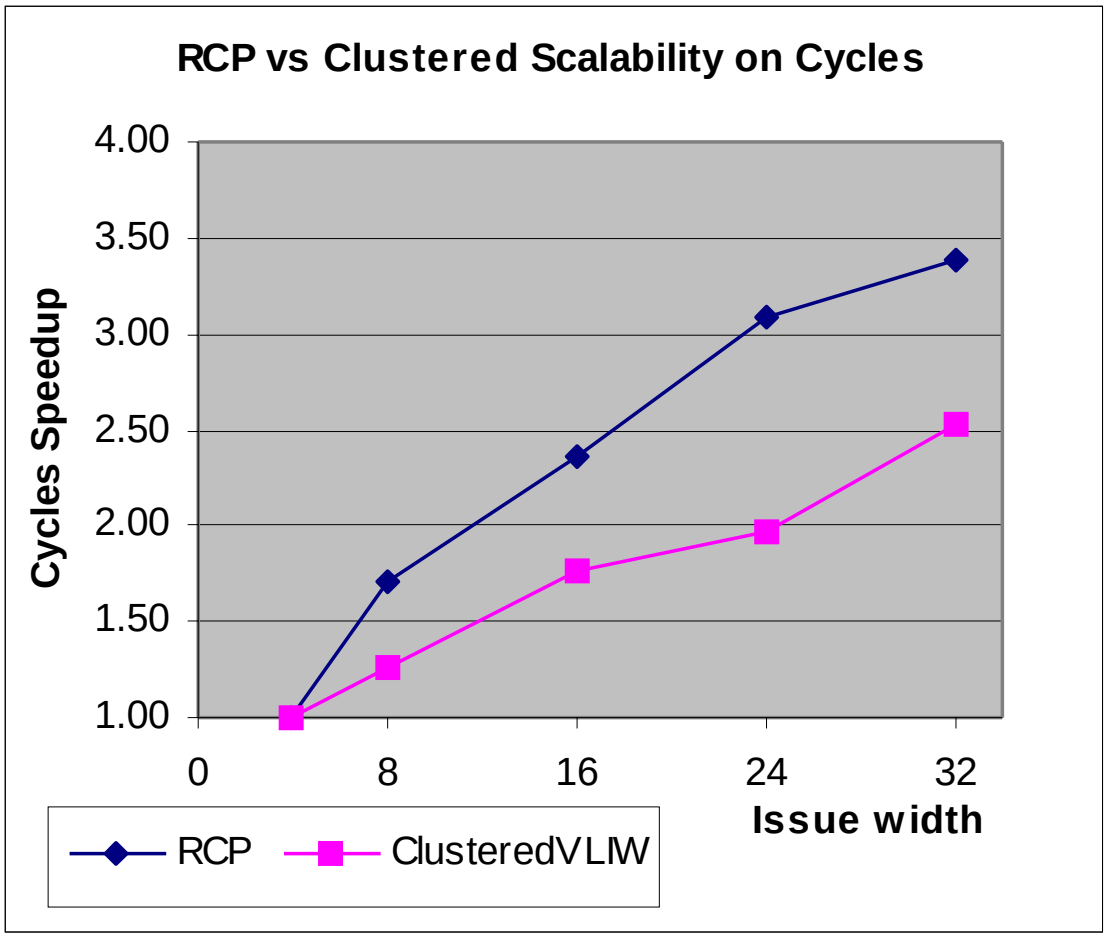
- Compare RCP scalability to traditional clustered VLIW scalability
- Clusters are identical
- Limited to one kernel: IDCT
  - Hand-coded for RCP
  - Compiled but loop fully unrolled for VLIW
- Memory variable separated
  - Memory bandwidth scaled with issue width
  - Large cache essentially eliminates misses

# Machines Compared

| label | # clusters | issue/cluster | issue width |
|---|---|---|---|
| VLIW1_4 | 1 | 4 | 4 |
| VLIW2_4 | 2 | 4 | 8 |
| VLIW4_4 | 4 | 4 | 16 |
| VLIW4_6 | 4 | 6 | 24 |
| VLIW4_8 | 4 | 8 | 32 |

| label | # clusters | issue/cluster | issue width |
|---|---|---|---|
| RCP4 | 1 | 4 | 4 |
| RCP8 | 2 | 4 | 8 |
| RCP16 | 4 | 4 | 16 |
| RCP24 | 4 | 6 | 24 |
| RCP32 | 4 | 8 | 32 |

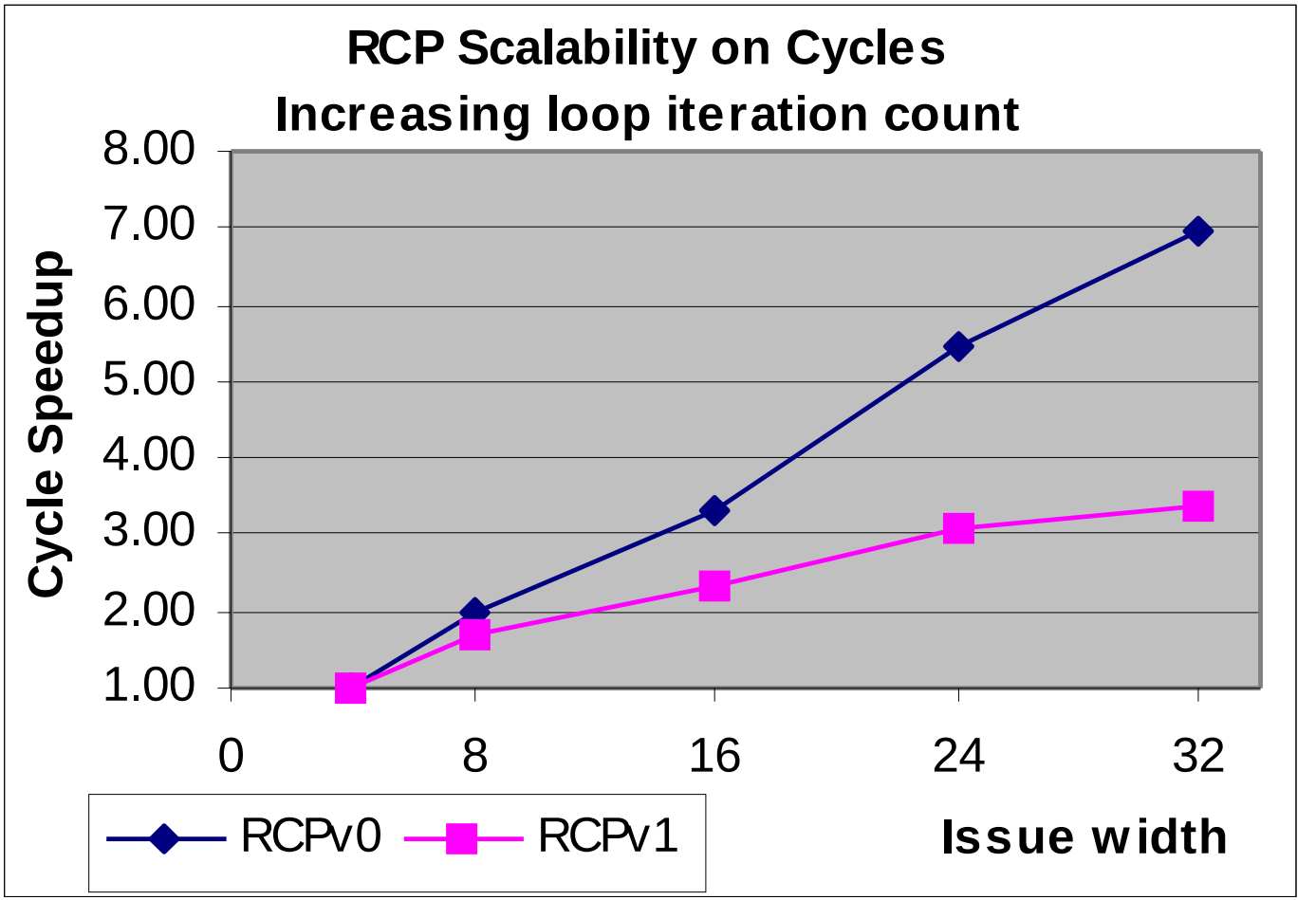
# Results (1)




Code size % dedicated to inter-cluster communications

| Total Issues | RCP Cost (%) | VLIW Cost (%) |
|---|---|---|
| 4 | 8.3% | 0 |
| 8 | 8.3% | 26.10% |
| 16 | 8.3% | 32.68% |
| 24 | 8.3% | 31.62% |
| 32 | 8.3% | 19.96% |

# Results – increasing iteration count



# Conclusions and future work

- We have described
  - a scalable clustered architecture using a limited connectivity inter-cluster bus scheme
  - inter-cluster transfer code overhead can be constant using a combination of runtime reconfiguration and register file architecture
  - a way to compile for such an architecture using a slightly modified version of modulo scheduling
- On going work
  - a compiler based on an existing VLIW compiler infrastructure
  - microarchitectural evaluation
  - memory interface

# Thank you!