TECHNISCHE UNIVERSITÄT DARMSTADT

Technische Universität Braunschweig

Embedded Systems & Applications

Abteilung E.I.S. VLSI

A Flexible Compute and Memory Infrastructure for High-Level Language to Hardware Compilation

Benjamin Thielmann (Tech. Univ. Braunschweig)
Hagen Gädke-Lütjens (Tech. Univ. Braunschweig)
Andreas Koch (Tech. Univ. Darmstadt)

2010-09-01

Content

- Introduction
- Modlib - a Modular Approach
- Performance Analysis
- Speculative Predicated Execution
- LMEM (Local Memory Infrastructure)
- Experimental Results
- Summary

Introduction

- Adaptive Computing System (ACS)
  - Central Processing Unit (CPU)
  - Reconfigurable Computation Unit (RCU)

- Characteristics
  - Compute-intense / frequent execution → Hardware (HW)
  - Less compute-intense / single execution / OS Calls → Software (SW)

- Application
  - High-performance computing
  - Low-power computing

- Programming Reconfigurable HW
  - HDL → optimum results @ high development costs
  - Synthesis from high-level languages → not mature / in development

Introduction

- COMRADE compiler
  - Covers most ANSI-C operators
  - Creates dynamically scheduled hardware
  - Efficient handling of control-intensive irregular code

- Implementation
  - Classic data flow model
  - Control flow predicating the execution of operators (optionally)
    - Activate Tokens (ATs) indicate the presence of data
    - Cancel Tokens (CTs) eliminate a miss-speculated computation

- Building blocks
  - Generic Library for Adaptive Computing Environments (GLACE)

GLACE

- GLACE characteristics
  - Hand-optimized operator library
  - Highly technology dependent
- Interface between compiler and module library
  - Request preplaced netlist
  - Obtain design parameters (area, throughput, latency, delay)
- Pros / Cons
  + Compact
  - Must be manually maintained
  - No pipelining
  - No unified interface (wrappers required)
- Hand-optimization worthwhile the effort?
  - Advanced synthesis tools

Modlib

- Wouldn't it be easier if we had primitives that...
  - ... provided a universal handshaking interface
  - … would be configurable to the application's and compiler's needs
  - ... acted mostly autonomously

- Modlib characteristics
  - Primitives in portable RTL descriptions
  - Extended operator support (ANSI-C)
  - Easily extendable (IP block insertion support)
  - Pipelining
  - Dynamic and static scheduling models
  - Speculative predicated execution (optionally)
  - Unified flexible parameterized interface

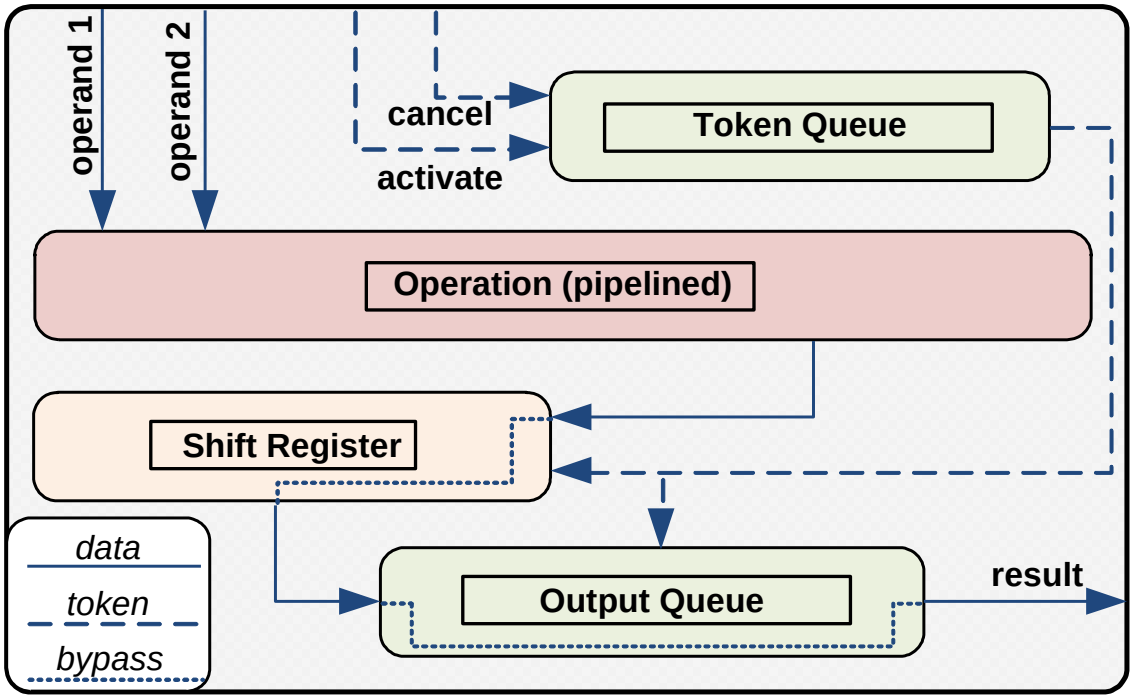
Internal View (Generic Operator)

- Shift Register is used for...
  - Balancing pipeline paths with unequal latency
  - Retiming registers to improve synthesis results

- Output Queue is used to ...
  - Decouple the execution from stalled data successors

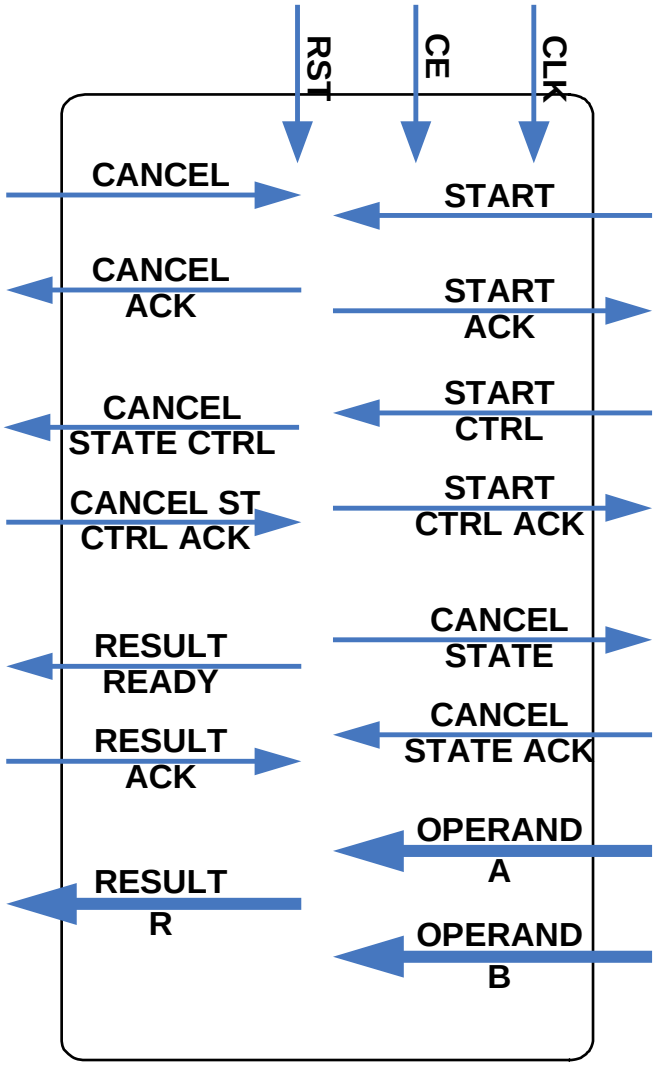
- Token Queue is used to ...
  - Reduce backpressure on the token delivering node

- Fully parameterized



Modlib Interface

- Actual Computation
  - Operands and result signal
- Dynamic Scheduling
  - Validation handshake for the operands
  - Validation handshake for the result
- Incoming Control Edge
  - Activate token intake via handshake
  - Cancel token intake via handshake
- Dynamic Cancel Tokens
  - Cancel forwarding handshake (for data)
  - Cancel forwarding handshake (for control)



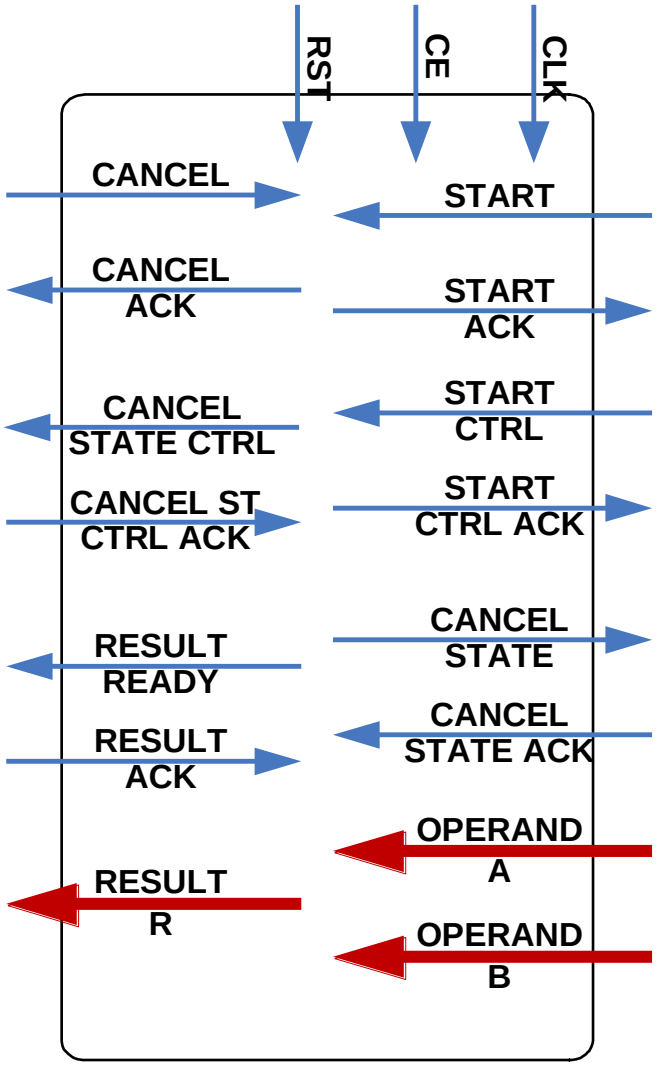
Modlib Interface

- Actual Computation
  - Operands and result signal

- Dynamic Scheduling
  - Validation handshake for the operands
  - Validation handshake for the result

- Incoming Control Edge
  - Activate token intake via handshake
  - Cancel token intake via handshake

- Dynamic Cancel Tokens
  - Cancel forwarding handshake (for data)
  - Cancel forwarding handshake (for control)



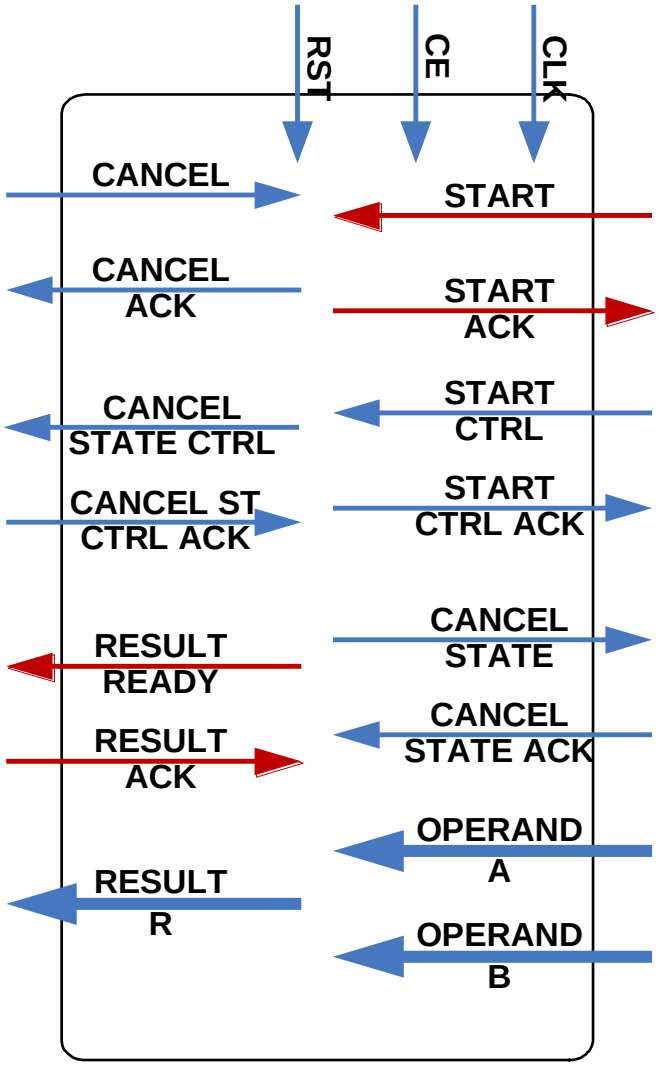
Modlib Interface

- Actual Computation
  - Operands and result signal

- Dynamic Scheduling
  - Validation handshake for the operands
  - Validation handshake for the result

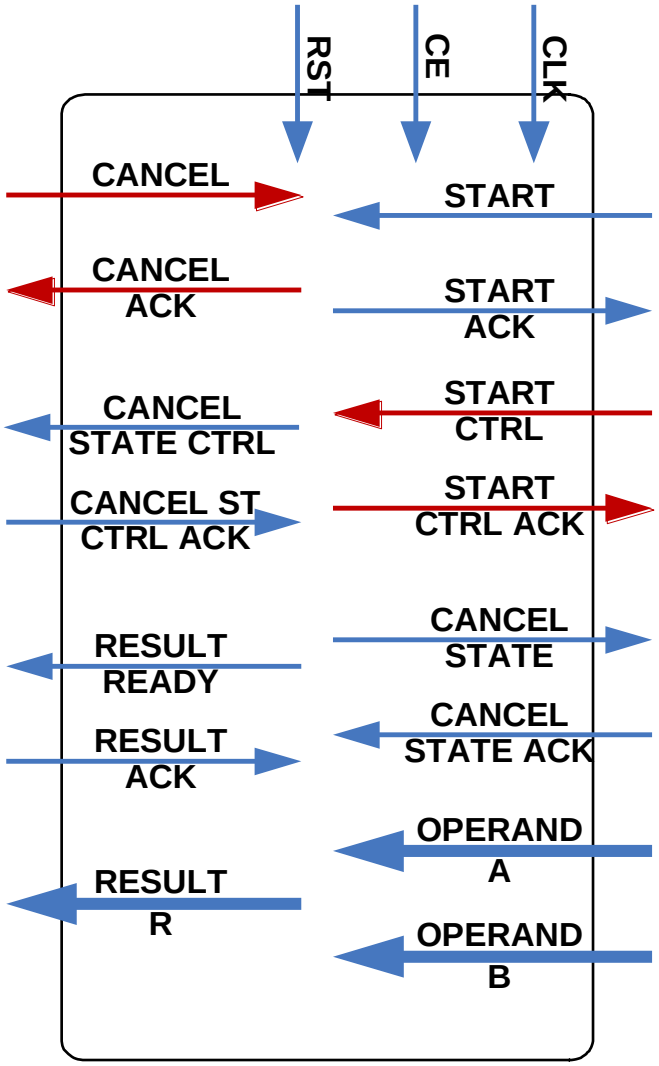
- Incoming Control Edge
  - Activate token intake via handshake
  - Cancel token intake via handshake

- Dynamic Cancel Tokens
  - Cancel forwarding handshake (for data)
  - Cancel forwarding handshake (for control)



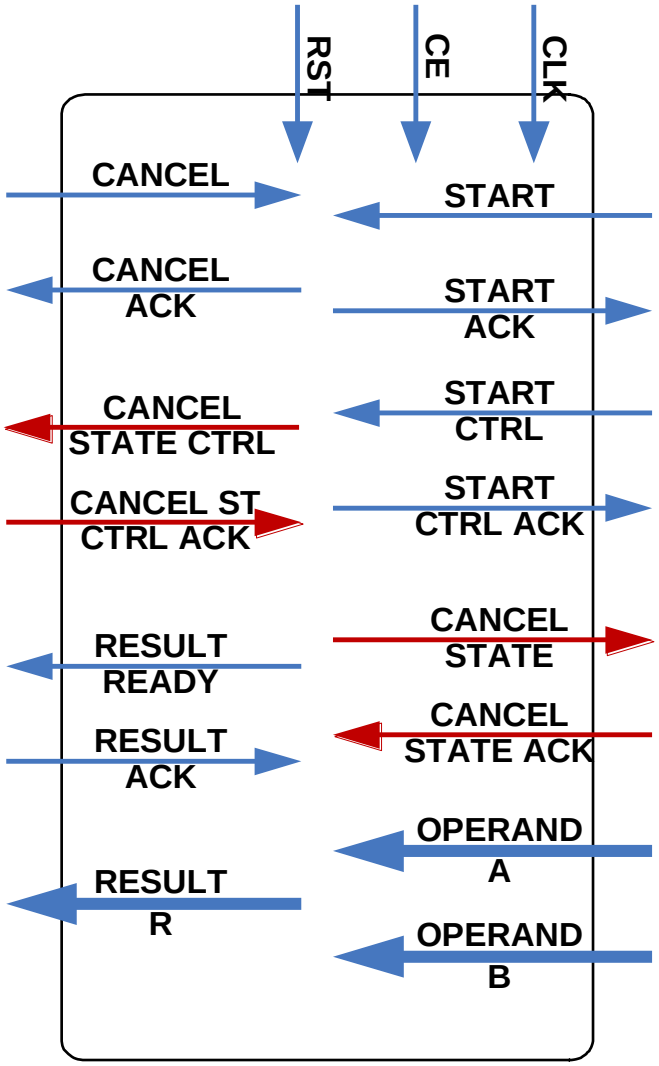
Modlib Interface

- Actual Computation
  - Operands and result signal
- Dynamic Scheduling
  - Validation handshake for the operands
  - Validation handshake for the result
- Incoming Control Edge
  - Activate token intake via handshake
  - Cancel token intake via handshake
- Dynamic Cancel Tokens
  - Cancel forwarding handshake (for data)
  - Cancel forwarding handshake (for control)



Modlib Interface

- Actual Computation
  - Operands and result signal
- Dynamic Scheduling
  - Validation handshake for the operands
  - Validation handshake for the result
- Incoming Control Edge
  - Activate token intake via handshake
  - Cancel token intake via handshake
- Dynamic Cancel Tokens
  - Cancel forwarding handshake (for data)
  - Cancel forwarding handshake (for control)

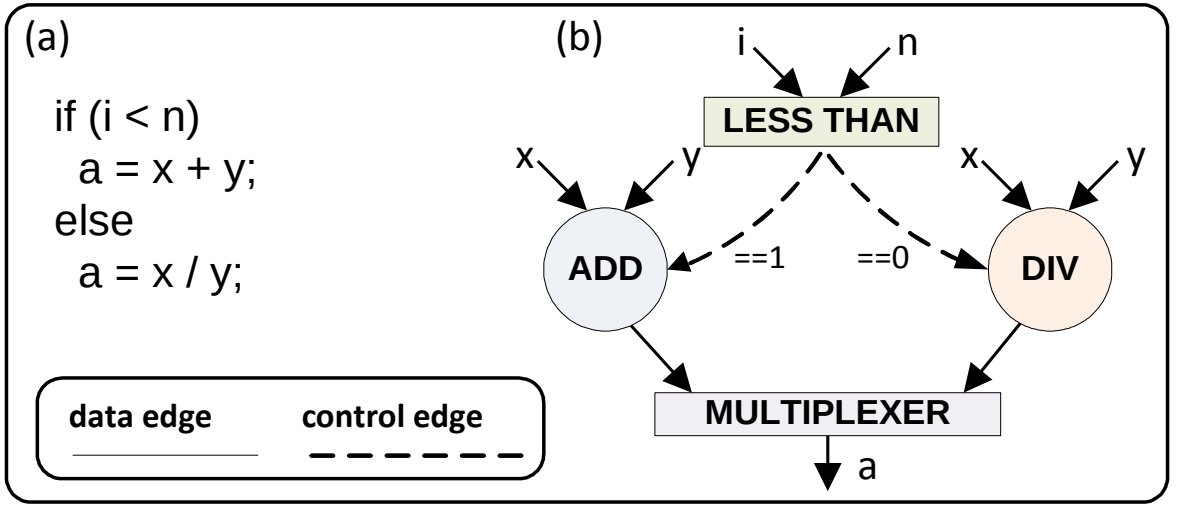


Speculative Predicated Execution

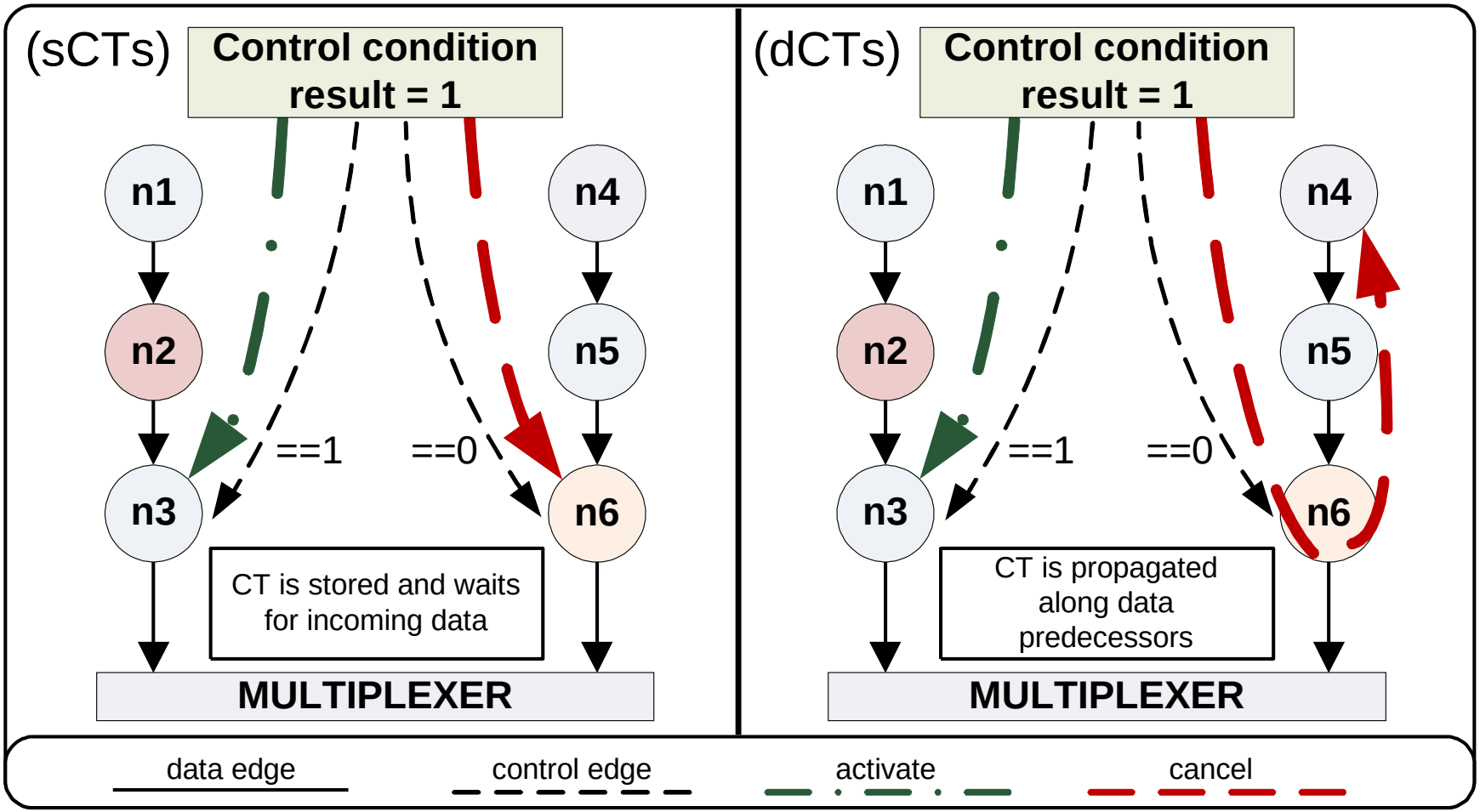
- Condition
  - Data Predecessors are ready (ATs available)
  - Control Condition is not resolved yet

- Control Flow Speculation
  - Execute all alternatives in parallel
  - Delete the results of the miss-speculated branches (by insertion of CTs)

- Advantages
  - Faster than pure lenient execution

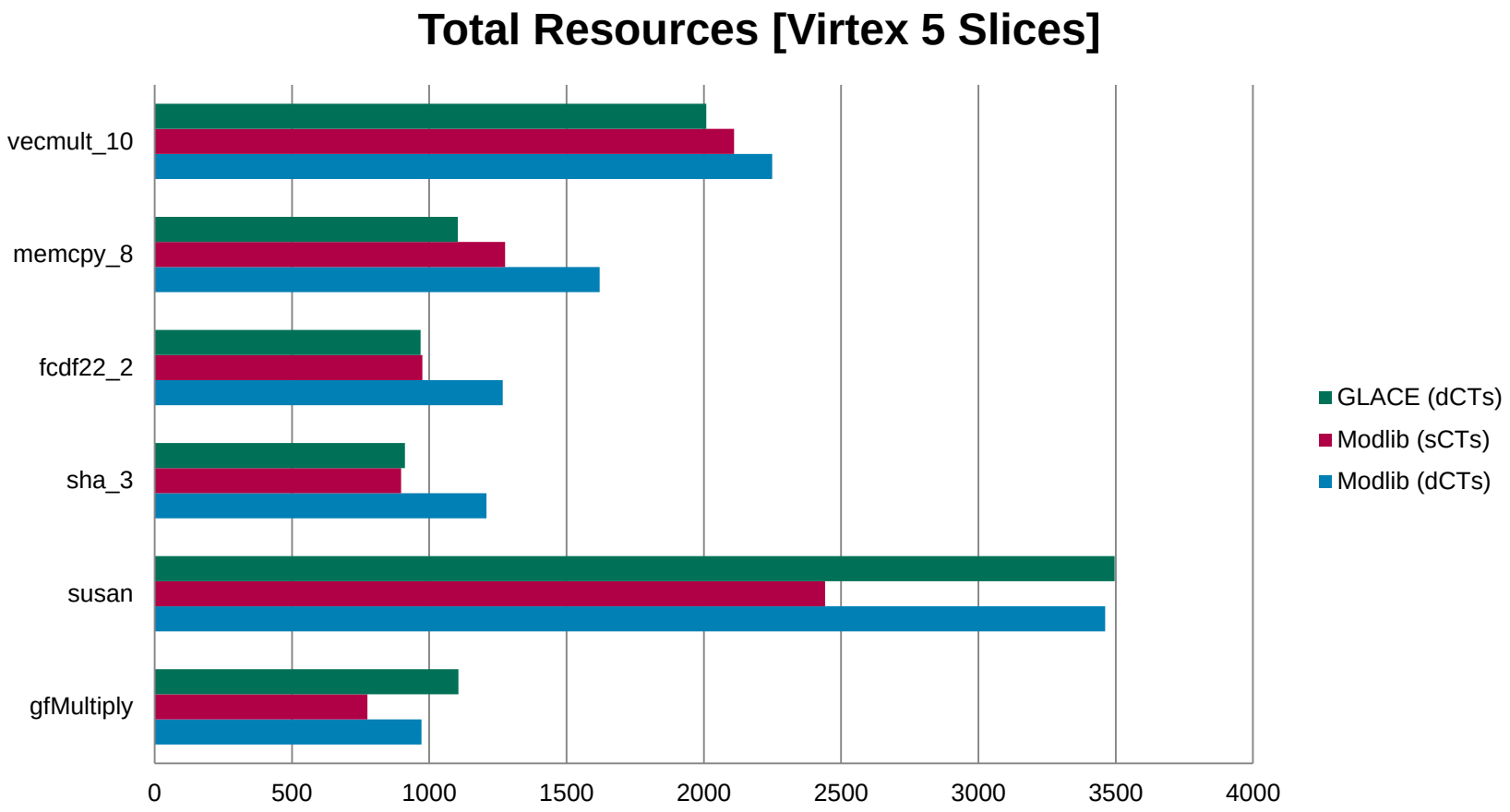


Static Cancel Tokens vs. Dynamic Cancel Tokens

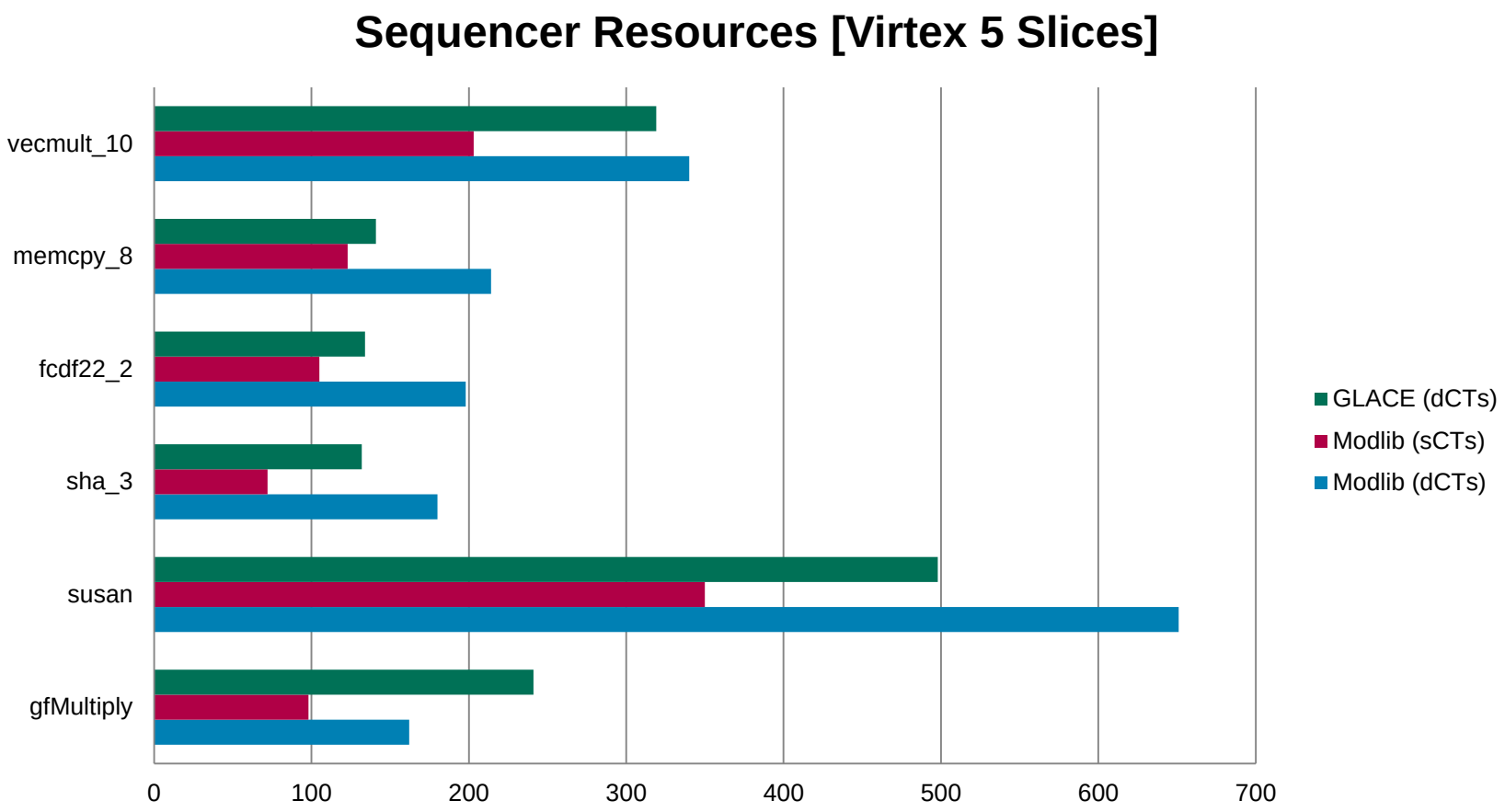


Performance Analysis

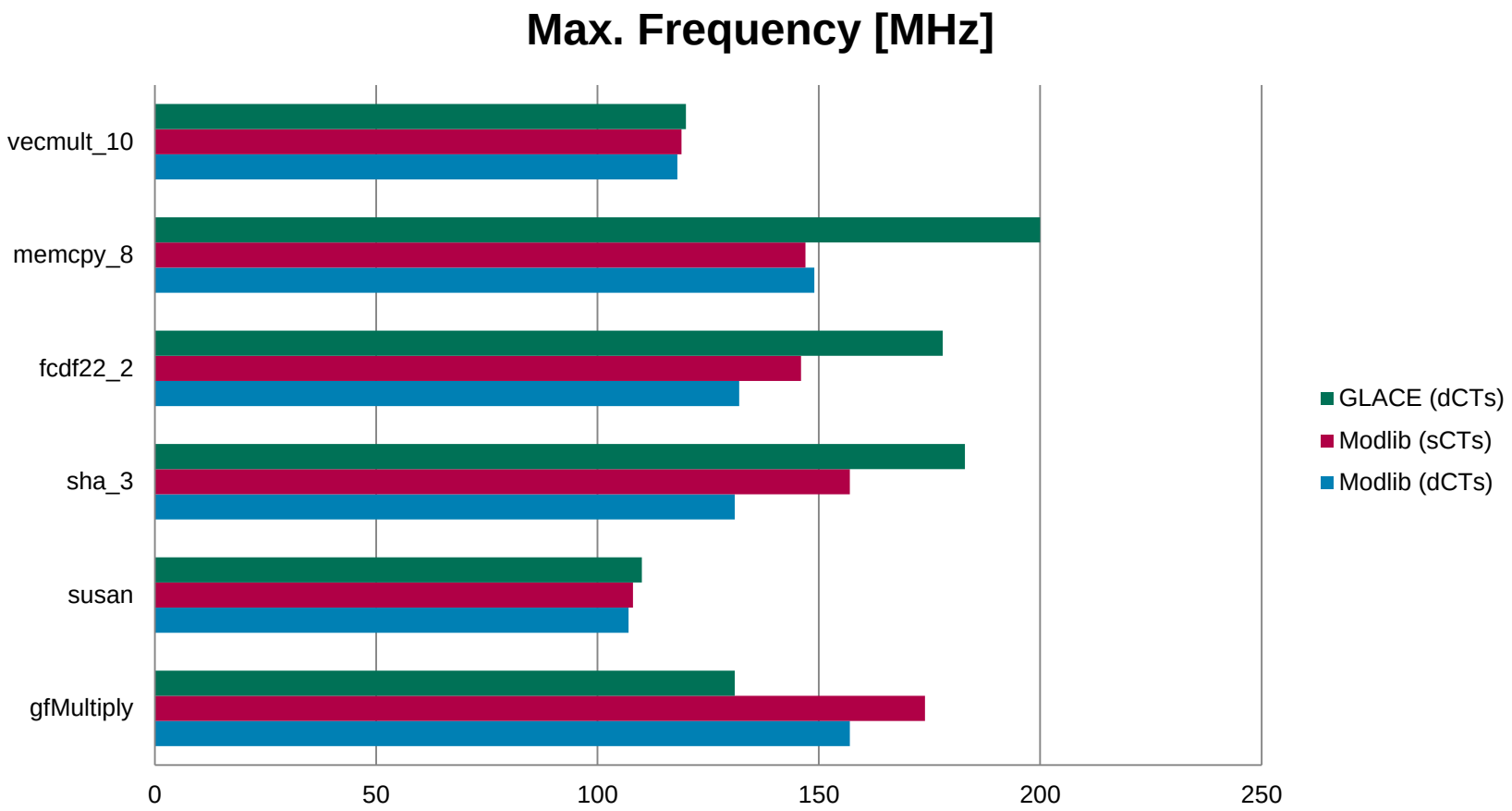
Modlib vs. GLACE I



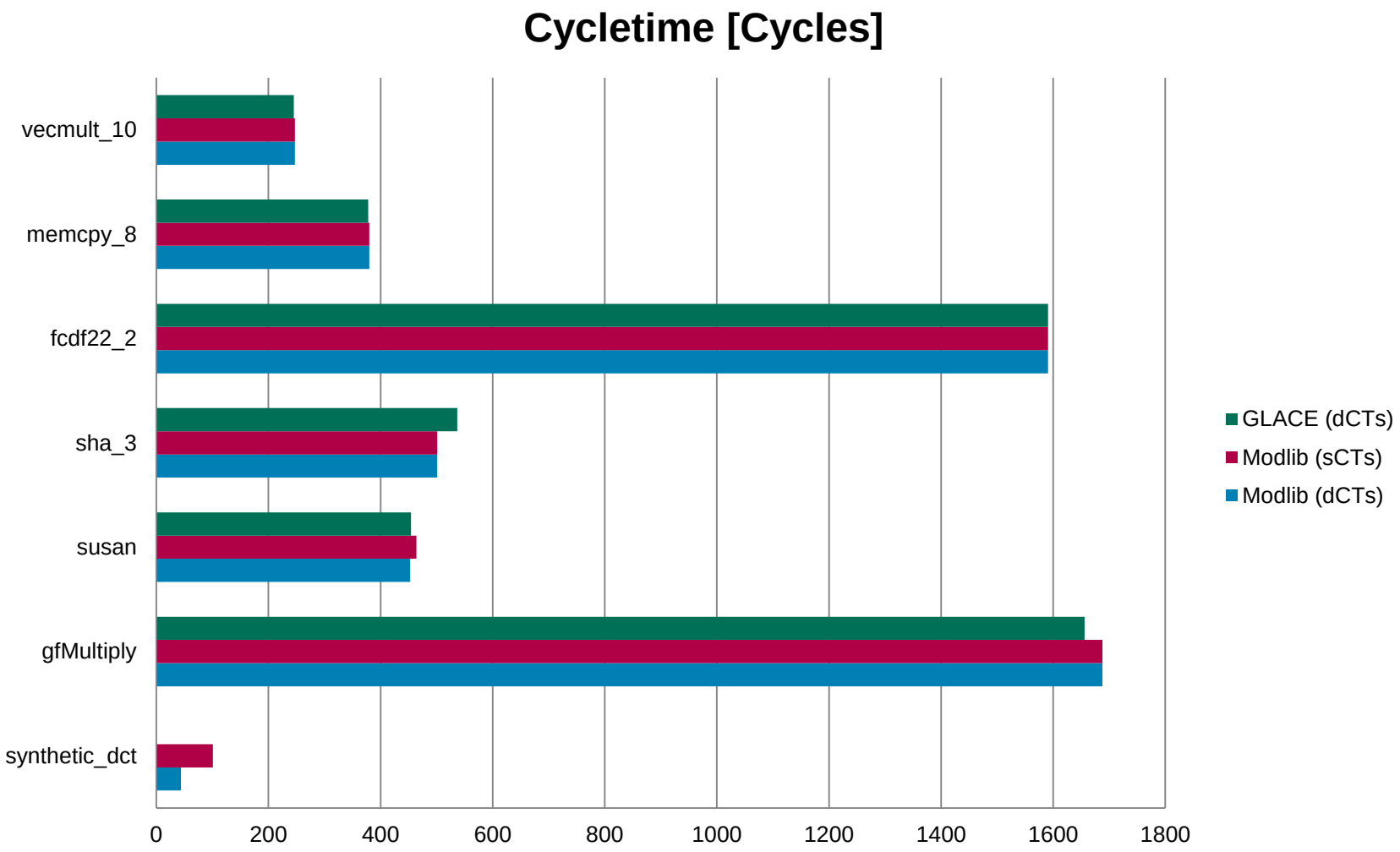
Modlib vs. GLACE II



Modlib vs. GLACE III



Modlib vs. GLACE IV



Example - Static CTs vs. Dynamic CTs

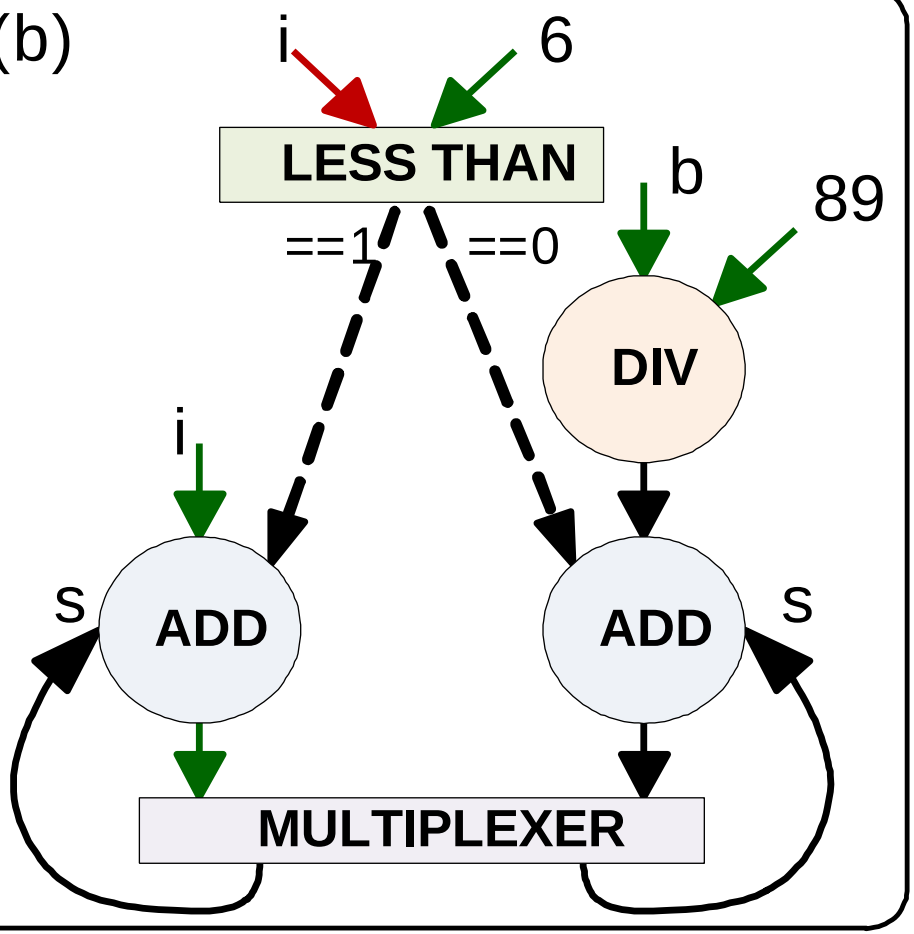
(a)

```
for (i=0; i<6 ; ++i)
  if ( i != 5 ) {
    s += i;
  } else {
    a = i + 10;
    b = a*90+1+i;
    c = b / 89;
    s += c;
  }
```

data edge ——————

control edge – – – – – –

(b)



Example - Static CTs vs. Dynamic CTs

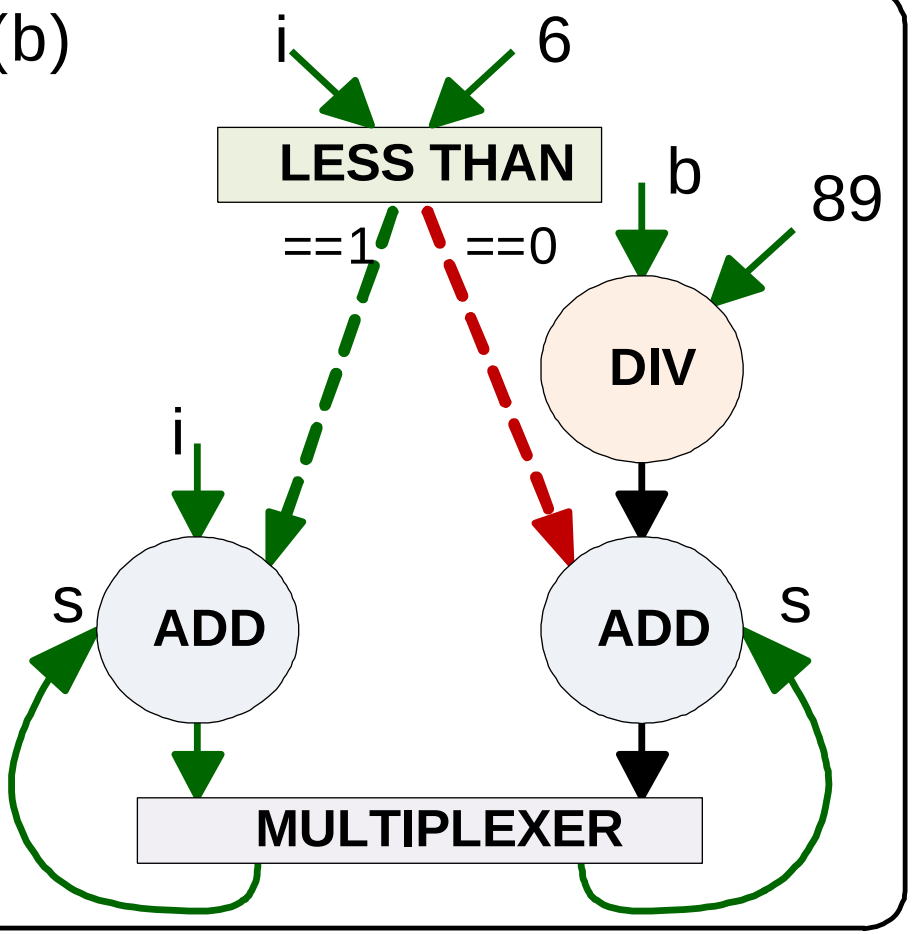
(a)

```
for (i=0; i<6 ; ++i)
  if ( i != 5 ) {
    s += i;
  } else {
    a = i + 10;
    b = a*90+1+i;
    c = b / 89;
    s += c;
  }
```

data edge ——————

control edge – – – – – –

(b)



Example - Static CTs vs. Dynamic CTs

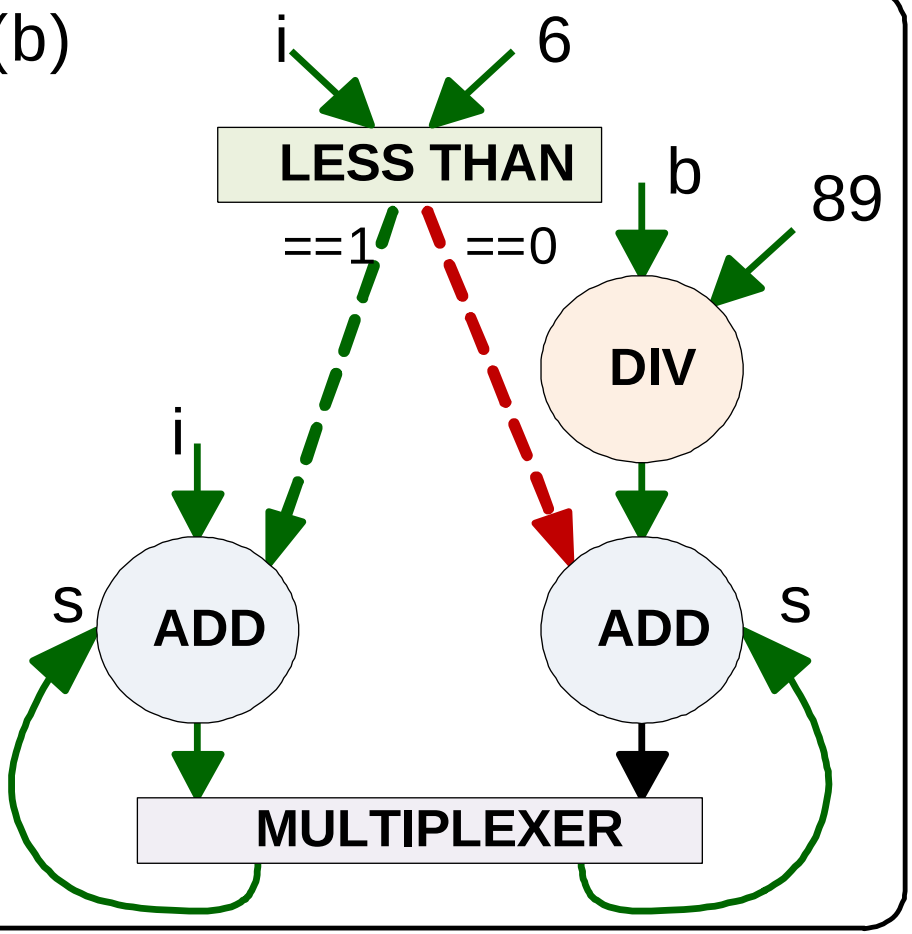
(a)

```
for (i=0; i<6 ; ++i)
  if ( i != 5 ) {
    s += i;
  } else {
    a = i + 10;
    b = a*90+1+i;
    c = b / 89;
    s += c;
  }
```

data edge ——————

control edge – – – – – –

(b)



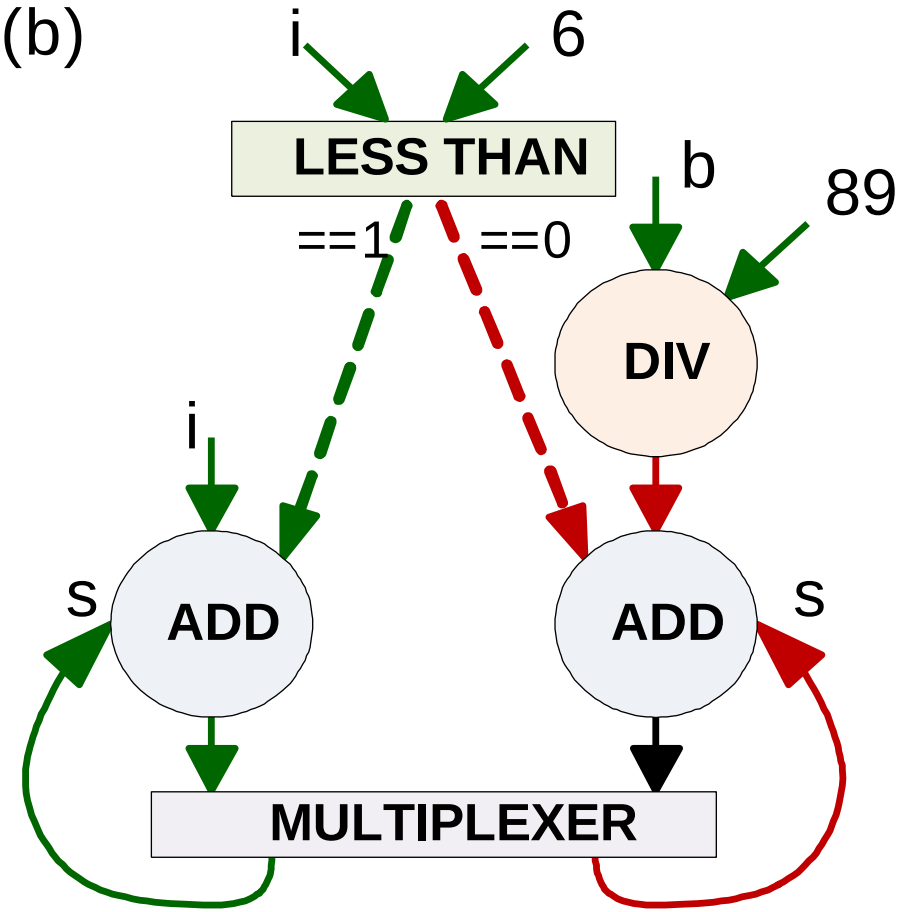
Example - Static CTs vs. Dynamic CTs

(a)

```
for (i=0; i<6 ; ++i)
  if ( i != 5 ) {
    s += i;
  } else {
    a = i + 10;
    b = a*90+1+i;
    c = b / 89;
    s += c;
  }
```

data edge ——————— control edge – – – – – –

(b)



Example - Static CTs vs. Dynamic CTs

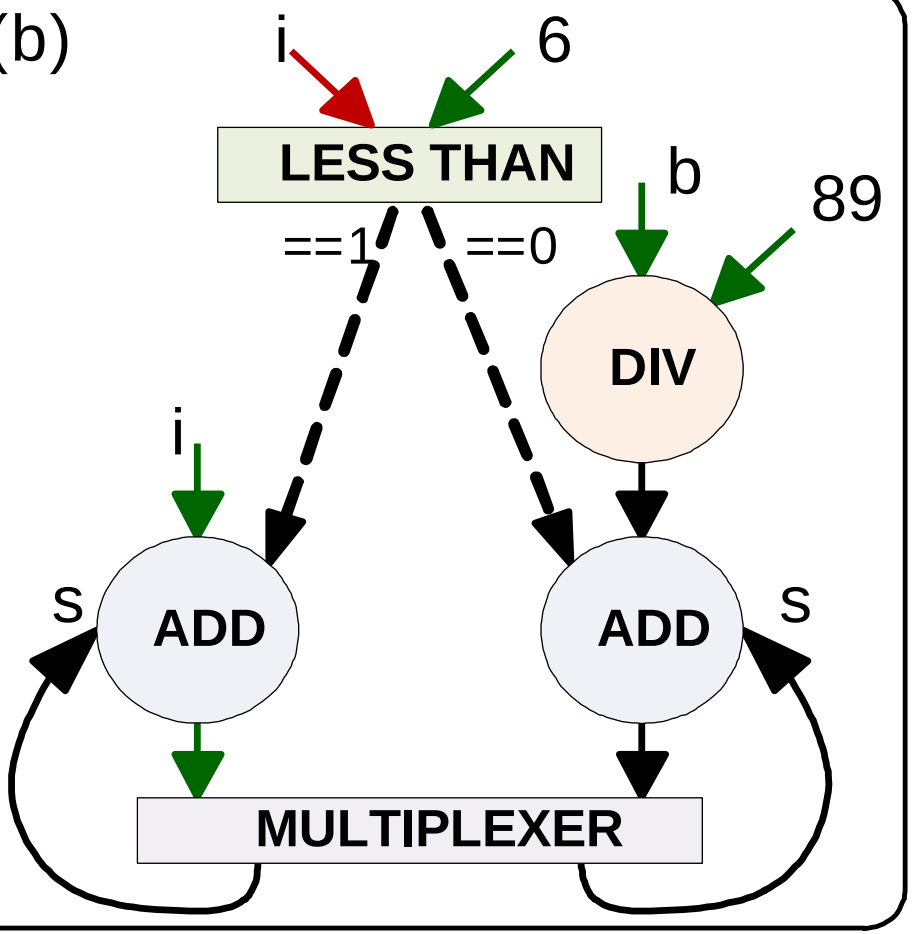
(a)

```
for (i=0; i<6 ; ++i)
  if ( i != 5 ) {
    s += i;
  } else {
    a = i + 10;
    b = a*90+1+i;
    c = b / 89;
    s += c;
  }
```

data edge ——————

control edge – – – – – –

(b)

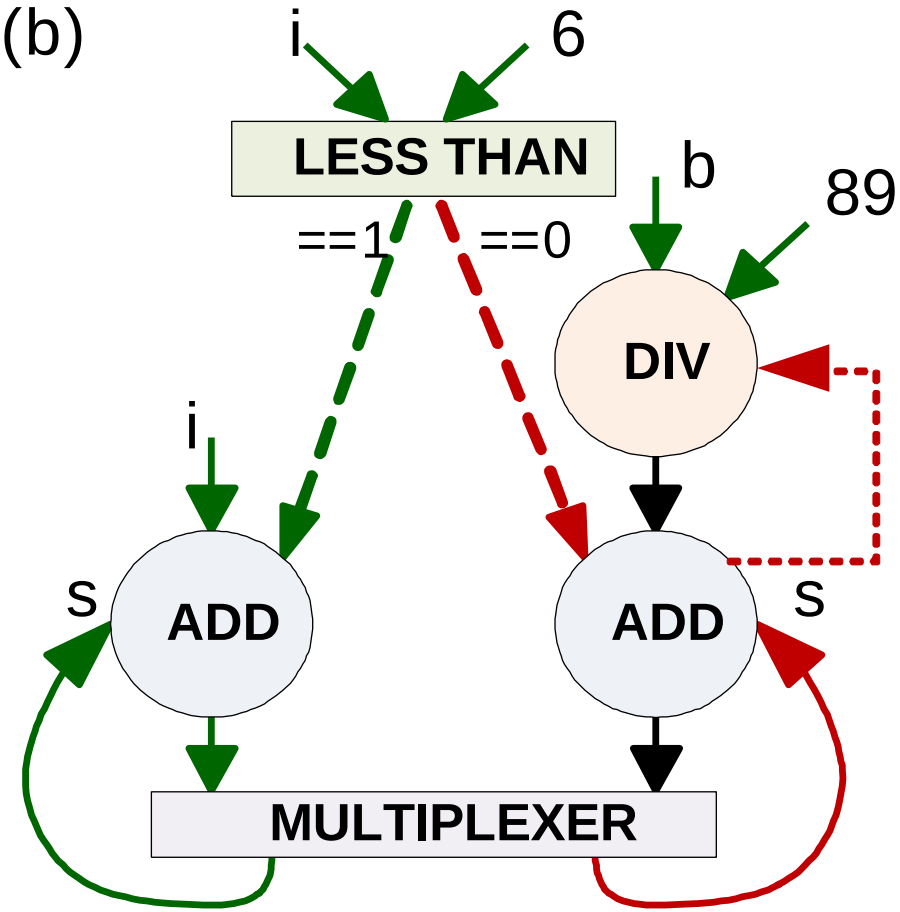


Example - Static CTs vs. Dynamic CTs

(a)

```
for (i=0; i<6 ; ++i)
  if ( i != 5 ){
    s += i;
  } else {
    a = i + 10;
    b = a*90+1+i;
    c = b / 89;
    s += c;
  }
```

(b)



data edge ——— control edge - - - - - -

Discussion

- Static CTs vs. Dynamic CTs
  - Static CTs mostly more efficient than Dynamic CTs
    - Less HW resources
    - Higher frequency
    - Choice for low complexity code
  - Dynamic CTs able to improve cycle efficiency
    - More HW resources
    - Lower frequency
    - For high complexity code (no need to wait for canceled branches)
    - Highly dependent on algorithm

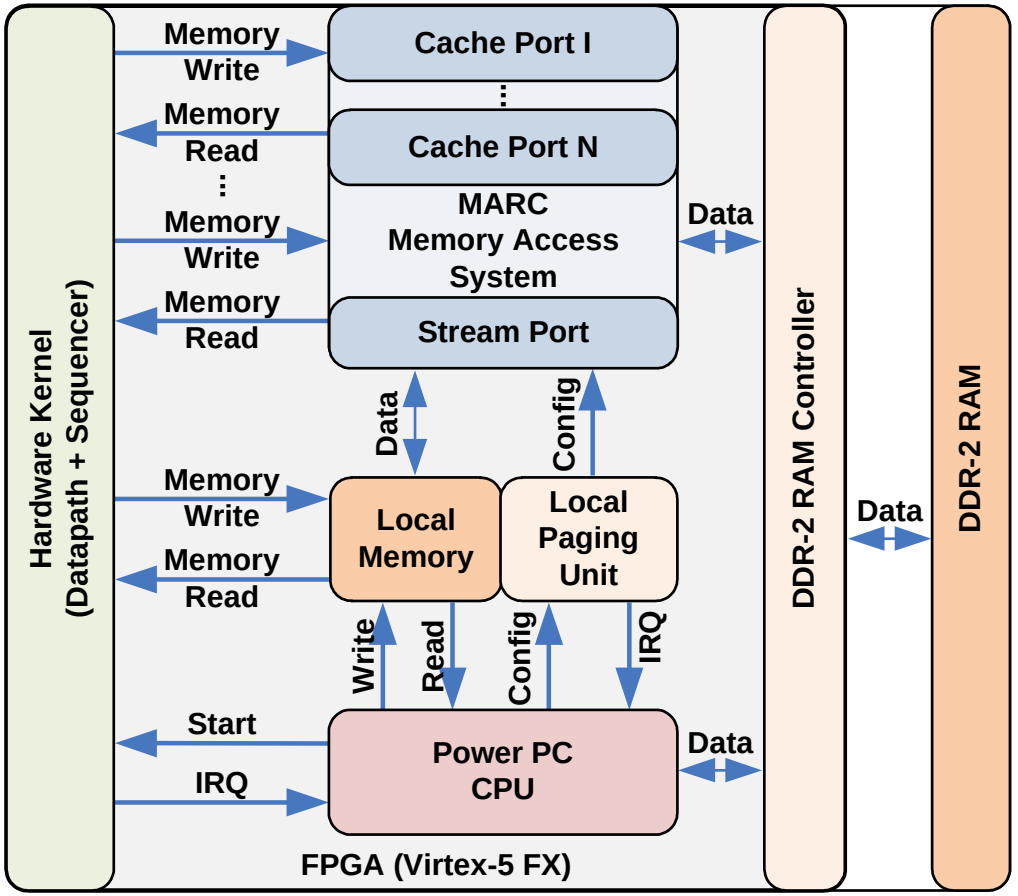
Discussion

- Modlib vs. GLACE
  - Modlib has slightly lower quality of result
  - Modlib is more flexible
  - Modlib is easier to maintain
  - Modlib supports pipelining
  - GLACE performance gain is not worthwhile the effort

- Performance optimization beyond arithmetic functions
  - Inevitable memory bottleneck
  - Take advantage of on chip memory resources

Local Memory Architecture (LMEM)

- Local Memory
  - BlockRAM implementation
  - Direct access via Power PC (PPC)
  - High bandwidth
  - Parallel access
- Local Paging Unit (LPU)
  - Configured via PPC
  - Initiates and supervises transfer
  - Notifies PPC on completion
- MARC Stream Port
  - Direct Memory Access (DMA)



Experimental Results

Benchmark LMEM / Pipelining

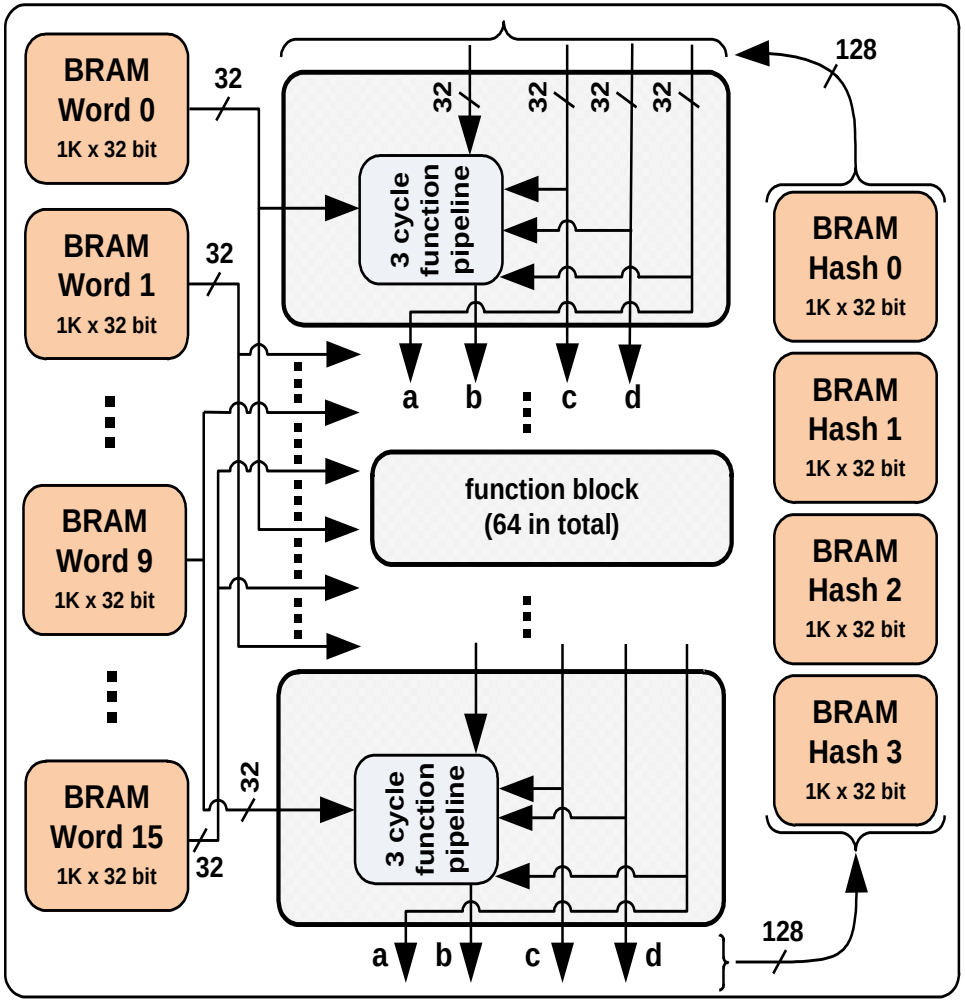
- MD5 (Message Digest Algorithm 5)
  - Cryptographic hash function
  - Computes a 128 bit hash

- Characteristics
  - Handles a message in 512 bit chunks
  - Each chunk processed in four rounds
  - Each round has 16 computation steps
  - Block chained algorithm

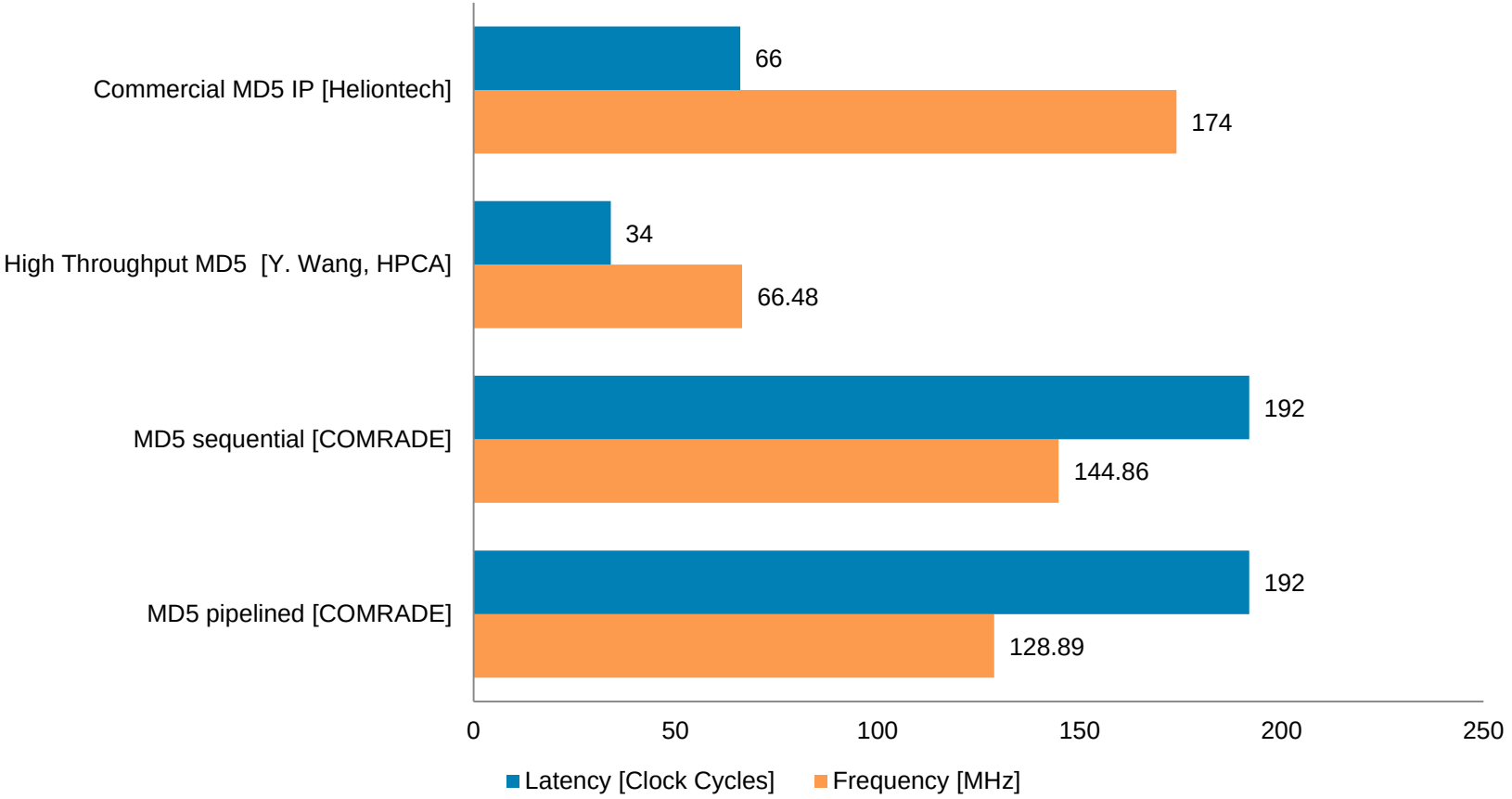
- Challenges
  - Difficult to pipeline → Data Parallelism (multiple messages)
  - Memory bandwidth → LMEM

MD5

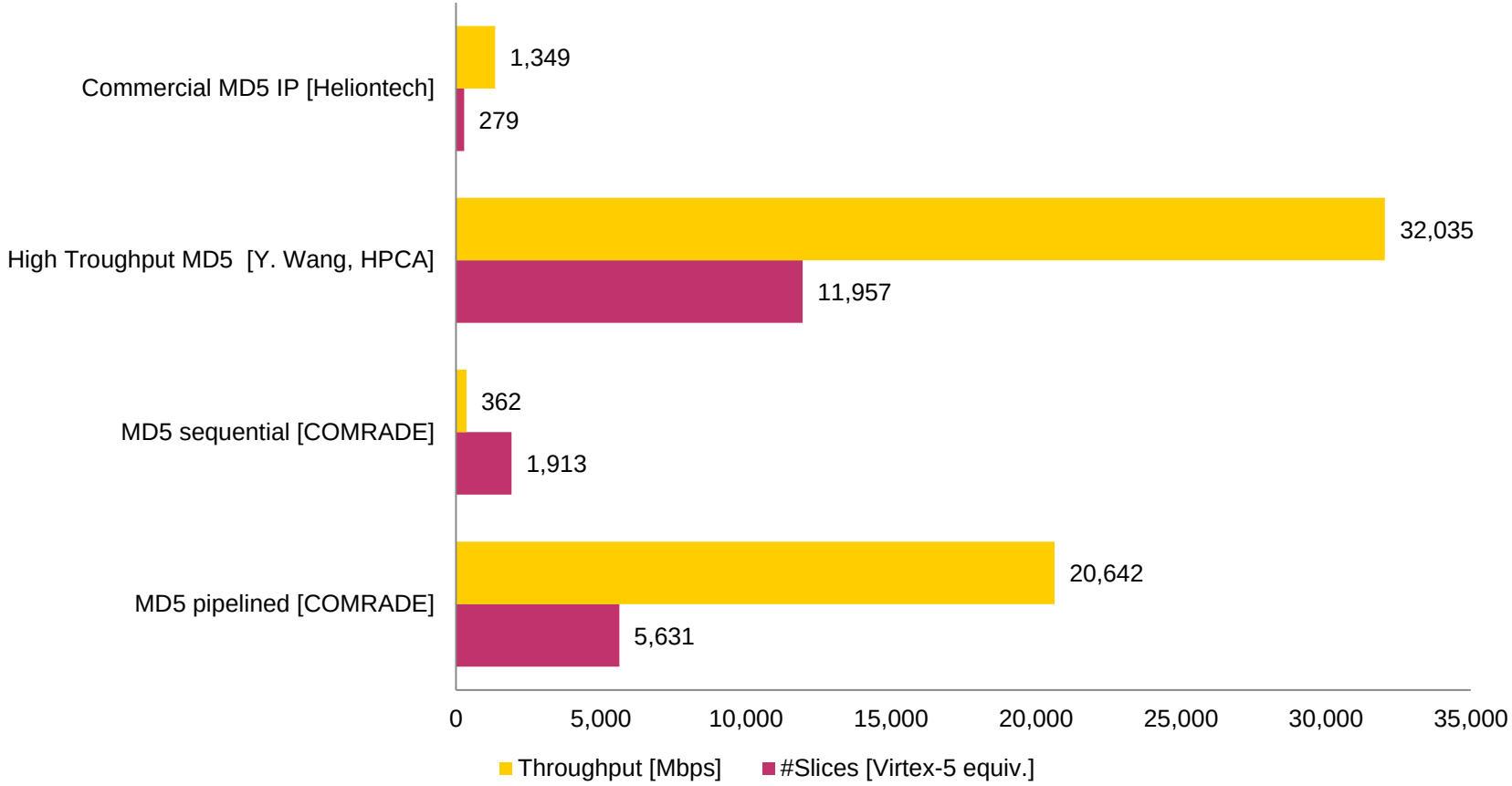
- Loop unrolling
  - 64 function blocks
  - Three cycles each
- LMEM insertion
  - Messages in BlockRAM
  - Hash in BlockRAM
- Output Queue configuration
  - Avoid pipeline stalls



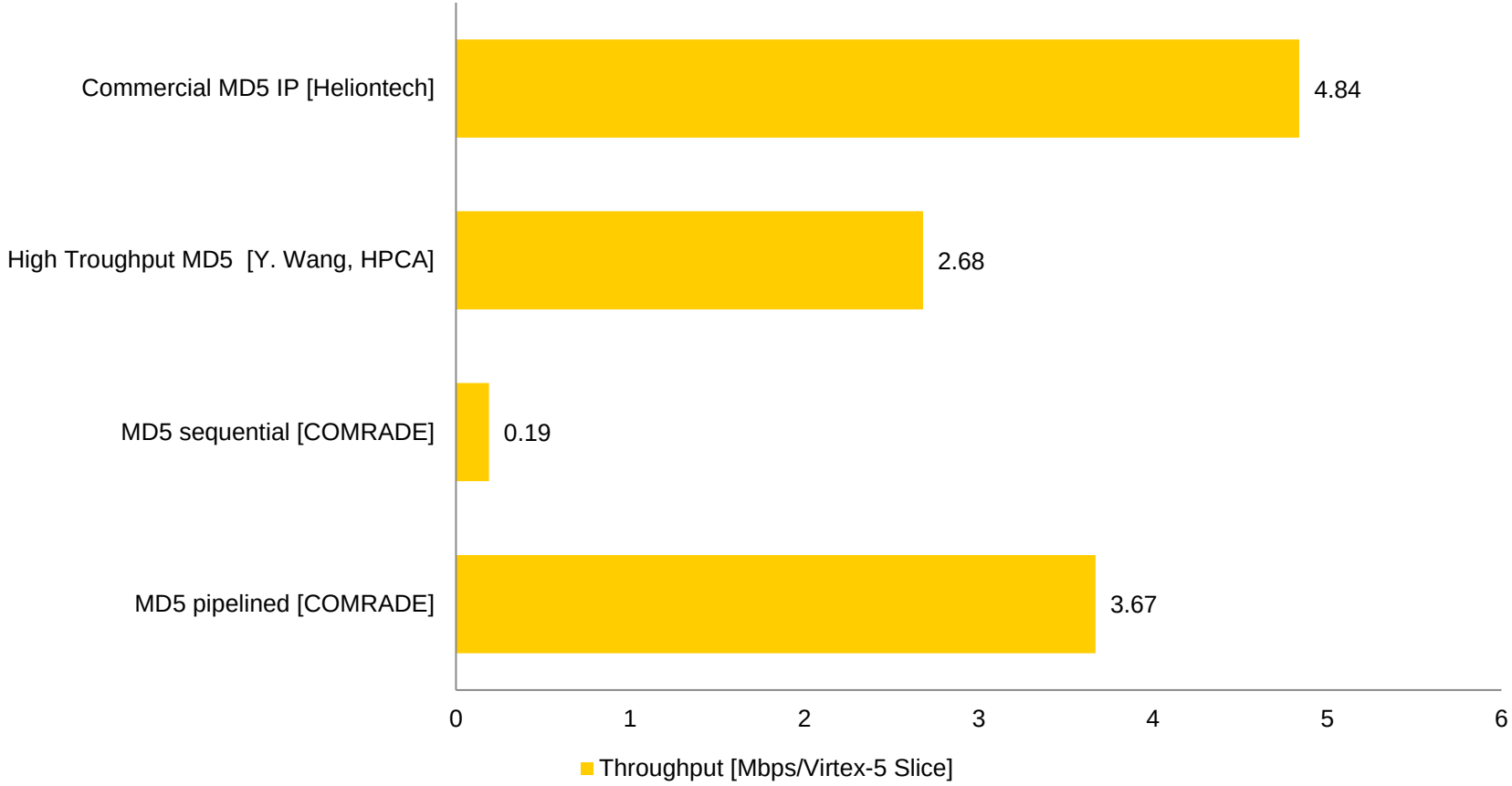
Latency vs. max. Frequency



Throughput and HW Resources



Throughput / HW Resource



MD5 Conclusion

- High Performance
  - Throughput comparable to highly optimized manual implementations
  - Full pipelining
  - LMEM access
- High Efficiency
  - Excellent performance per area
  - Efficient pipelining (queues)
- High Latency
  - Due to three cycle function pipeline
  - Improvement possible

Summary

- Modlib has advantages over GLACE
  - Similar quality of result
  - Enhanced application area
    - Target technology independent
    - Pipelining support
    - Support for different scheduling models
    - Support for different execution models
- Dynamic CTs can improve cycle count
  - Highly dependent on algorithm
  - More costly than static CTs
- LMEM
  - Integrates well in the architecture (lightweight)
  - Significantly speeds up bandwith intensive applications

Thank You

for Your Attention

The Modlib and its documentation is available
@
http://www.esa.cs.tu-darmstadt.de/modlib

Benchmarks Modlib vs. GLACE

- Vecmult 10
  - vector multiplication, unrolled x10 [Synthetic]
- Memcpy 8
  - array copy, unrolled x8 [Synthetic]
- Fcdf22 2
  - 2-D wavelet transform for image compression [Kumar, Pires]
- Sha 3
  - Secure Hash Algorithm [CHStone]
- Susan
  - Edge detection on gray scale image [MiBench]
- GfMultiply
  - Pegwit elliptic curve cryptography application.

Colors

R 190 G 30 B 60				
R 255 G 205 B 0	R 255 G 220 B 77	R 255 G 230 B 127	R 255 G 240 B 178	R 255 G 245 B 204
R 250 G 110 B 0	R 252 G 154 B 77	R 252 G 182 B 127	R 253 G 211 B 178	R 254 G 226 B 204
R 176 G 0 B 70	R 192 G 51 B 107	R 215 G 127 B 162	R 235 G 191 B 209	R 243 G 217 B 227
R 124 G 205 B 230	R 164 G 220 B 238	R 189 G 230 B 242	R 215 G 240 B 247	R 229 G 245 B 250
R 0 G 128 B 180	R 77 G 166 B 203	R 140 G 198 B 221	R 191 G 223 B 236	R 217 G 236 B 244
R 0 G 83 B 116	R 64 G 126 B 151	R 140 G 177 B 192	R 191 G 212 B 220	R 217 G 229 B 234

R 8 G 8 B 8	R 95 G 95 B 95	R 150 G 150 B 150	R 192 G 192 B 192	R 221 G 221 B 221
R 198 G 238 B 0	R 215 G 243 B 77	R 226 G 246 B 127	R 238 G 250 B 178	R 244 G 252 B 204
R 137 G 164 B 0	R 173 G 191 B 77	R 196 G 209 B 127	R 219 G 228 B 178	R 231 G 237 B 204
R 0 G 113 B 86	R 77 G 156 B 137	R 140 G 191 B 179	R 191 G 219 B 213	R 218 G 234 B 231
R 204 G 0 B 153	R 222 G 89 B 189	R 235 G 153 B 214	R 245 G 204 B 235	R 250 G 229 B 245
R 118 G 0 B 118	R 152 G 64 B 152	R 186 G 127 B 186	R 214 G 178 B 214	R 235 G 217 B 235
R 118 G 0 B 84	R 156 G 77 B 136	R 193 G 140 B 178	R 221 G 191 B 212	R 235 G 217 B 230