TRIPS: Extending the Range of Programmable Processors

Stephen W. Keckler
Doug Burger and Chuck Moore

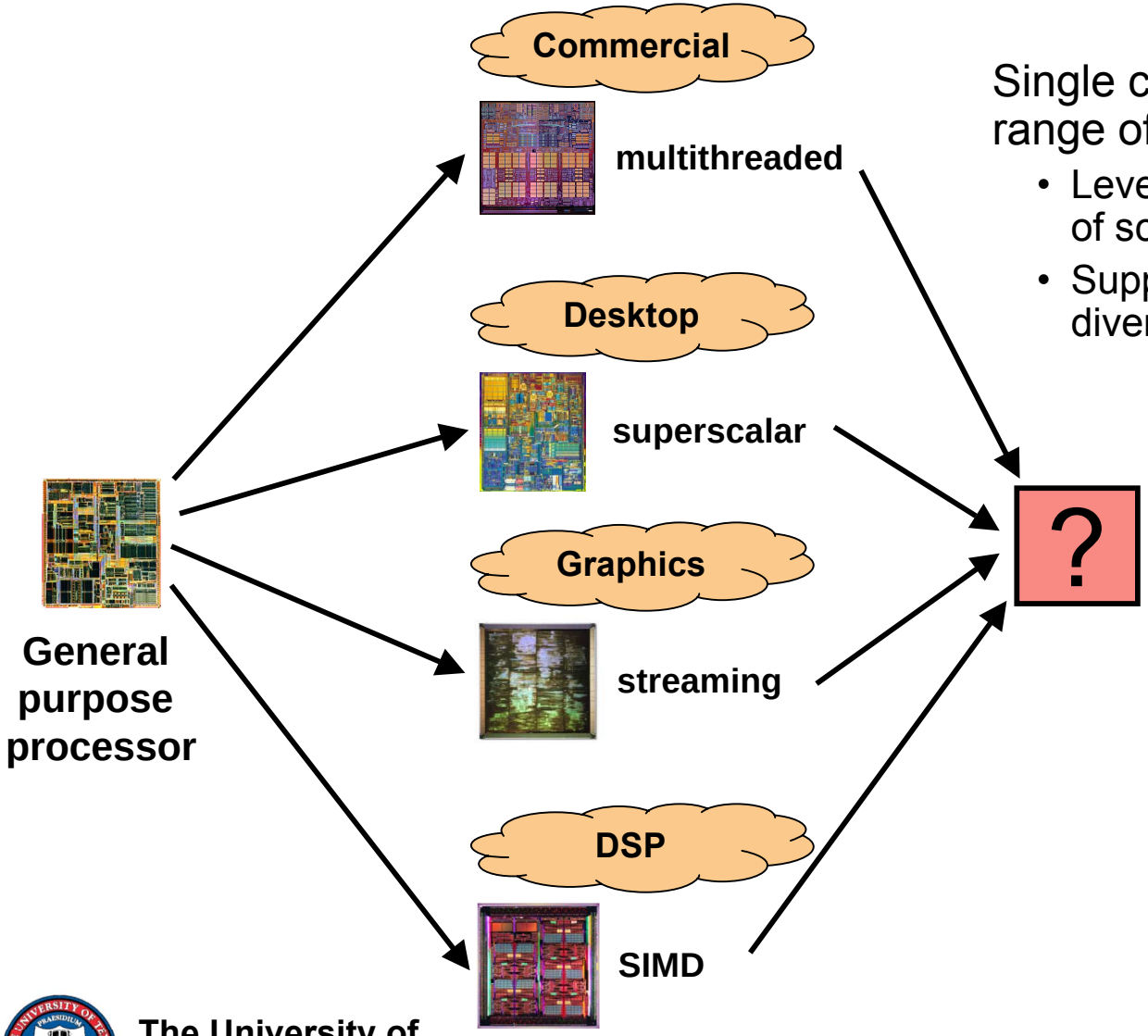
Computer Architecture and Technology Laboratory
Department of Computer Sciences
The University of Texas at Austin
www.cs.utexas.edu/users/cart

The University of Texas at Austin

Outline

- Application diversity and performance scalability
- Limitations of conventional architectures
- TRIPS – a new architecture family
- Conclusions and future work

Application Range Decreasing in GP Processors

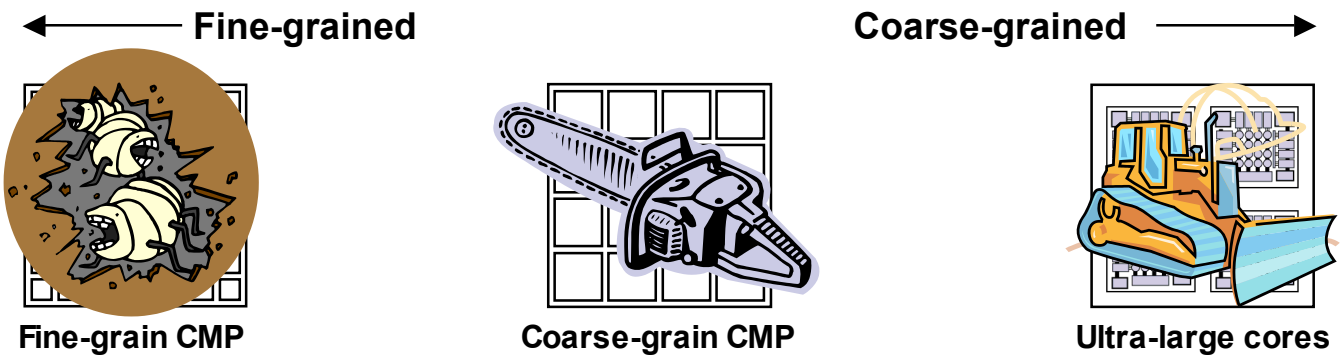


Single chip with greater range of capabilities

- Leverage larger economies of scale
- Support intra-application diversity

The University of Texas at Austin

Increasing Performance Scalability Range



- Termites and chainsaws
  - Good performance when tasks are abundant
  - But: burden of communication, synchronization, load balancing
- Multiprocessor performance starts with a powerful uniprocessor
  - Bulldozers serve a broader range of applications
    - Better to build ten 10x processors than two hundred 1x processors
  - Sub-divide into chainsaws/termites when necessary for finer granularity
    - SMT is a contemporary example

Scaling Superscalar Processors?

- Looking back in time
  - Enormous gains in frequency
    - **1998**: 500MHz → **2002**: 3000MHz
    - Equal contributions from pipelining and technology
  - IPC basically unchanged
    - **1998**: ~1 IPC → **2002**: ~1 IPC
    - Microarchitecture innovations just overcome losses due to pipelining

- Looking forward
  - Faster clock rates? → *deeper pipelines (toward < 10 FO4)*
    - Key latencies increase … IPC decreases
    - Power overheads increase superlinearly
    - After next (and final) FO4 jump, frequency growth limited to technology only
  - Higher IPC? → i.e. w*ide issue (16) and large window (512+)*
    - Complexity grows *quadratically,* but gain is *logarithmic*
      - Bypass broadcast, renaming, instruction scheduling
    - Wire delay limits size/speed of monolithic structures
    - Achieving higher IPC is problematic in conventional architectures

What is Going Wrong?

1. Superscalar microarchitecture: *scalability is limited*
   - Relies on large, centralized structures that want to grow larger
   - Partitioning is a slippery slope: *complexity, IPC loss…*

2. Architecture: *conventional binary interface is outdated!*
   - Linear sequence of instructions
   - Defined for simple, single-issue machines

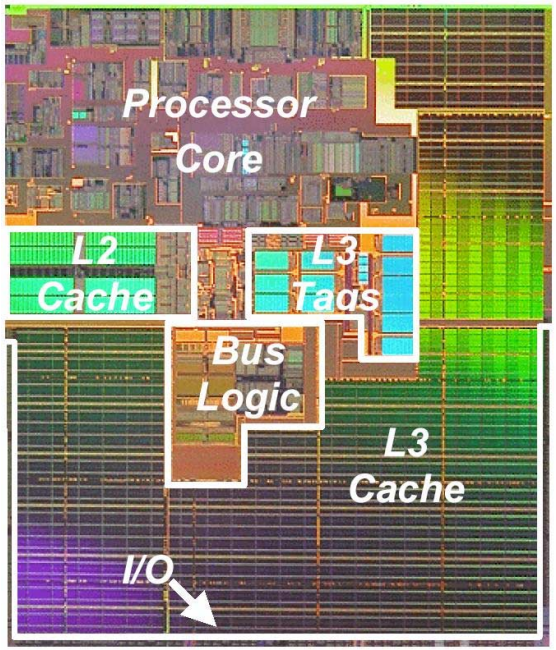
   - Not natural for compiler
     - Compiler forced to map control/data flow graphs into linear sequence
     - Lots of useful information gets thrown away

   - Not natural for instruction parallel machines
     - Instruction relationships scattered throughout linear sequence
     - Hardware must dynamically rediscover control/data flow graphs
     - N^2 problem → large, centralized structures

Explicitly Parallel Architectures (VLIW)

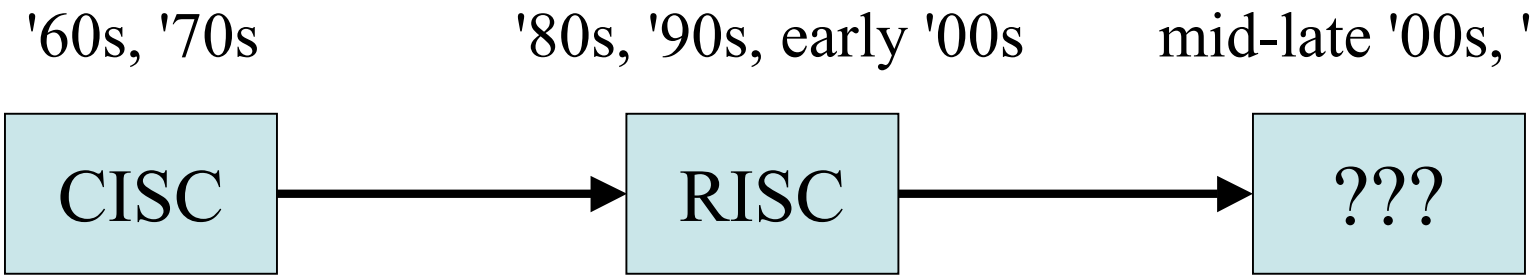
- Architecture can be clean
  - Hardware does not reconstruct dataflow graphs
  - Simple in-order issue semantics
  - Opportunity for higher arithmetic density
  - Opportunity for power reduction
    - Shift scheduling work to compiler

- But – scalability issues
  - Common register file
  - Full broadcast result bypass
  - In-order issue not without complexity
    - ALAT, register stack engine
  - Future transition to OOO?
    - Faces challenges discovered in superscalar
    - Not without becoming a new architecture



IPF – Itanium 2 (Intel, DAC 2003)
1.5GHz, 6-issue

Architecture Generations Driven by Technology

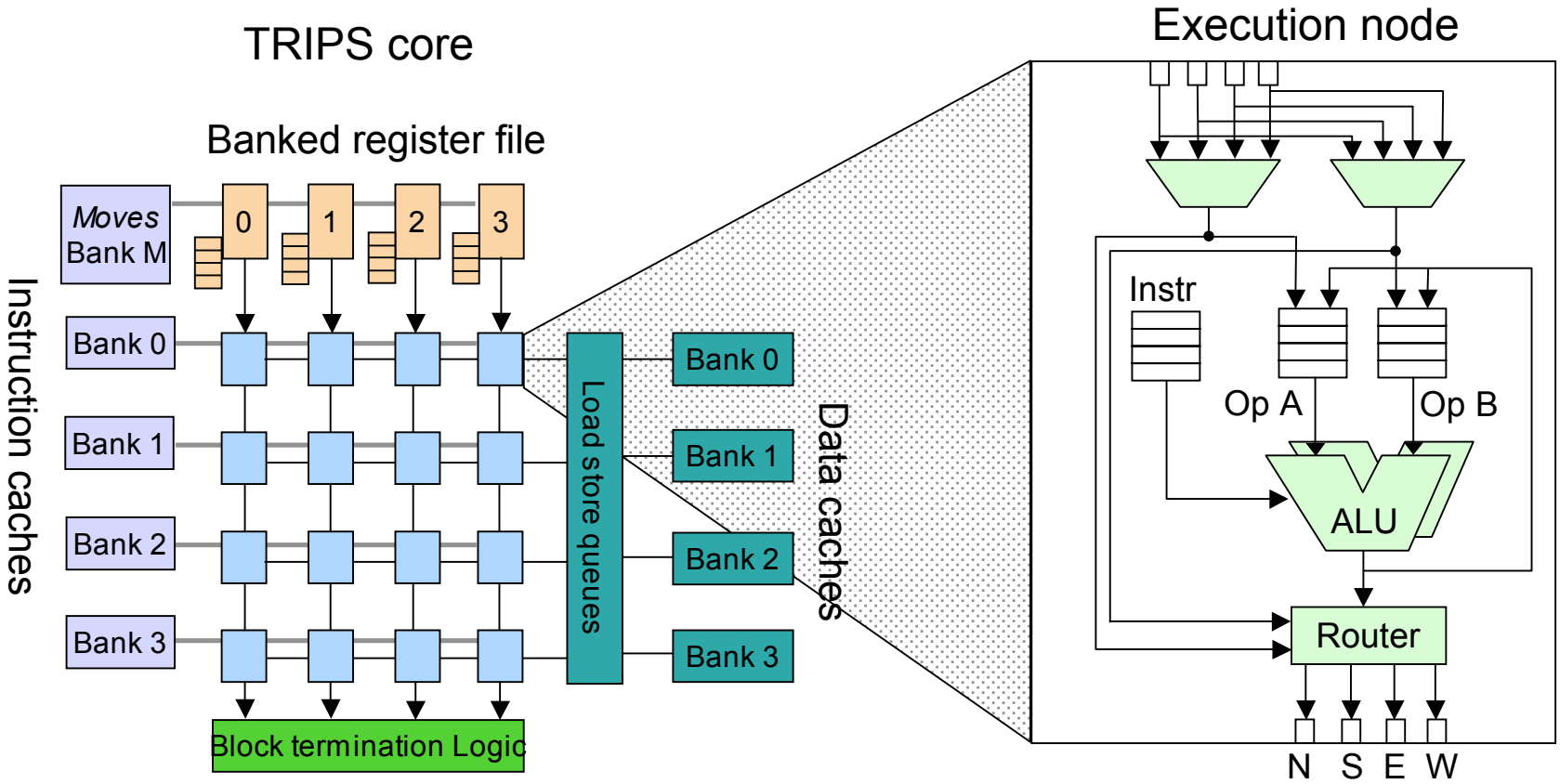


Transistor limited	Chip-area limited	Communication limited
Complex instructions	Simple instructions	Reduce overheads of single insts.
Dense encodings	Optimized for pipelining	Efficient out-of-order processing
Few instructions in flight	Tens of instructions in flight	Hundreds to thousands in flight
Simple compilers	Compiler instruction scheduling	Compiler managed communication
Pipelining difficult	Wide-issue difficult	???

TRIPS – A New Execution Model

- Compiler structures program into sequence of *hyperblocks*
  - Atomic unit of fetch / schedule / execute / commit
- Blocks specify *explicit instruction placement* in the ALU array
  - Critical path placed to minimize communication delays
  - Less critical instructions placed in remaining positions
- Instructions specify consumers as *explicit targets*
  - Communication cast into instruction encoding → ***no HW dependence analysis***
  - Point-to-point results forwarding → ***no associative issue queues***
    → ***no global bypass network***
  - In-array storage expands register space → ***no register renaming***
  - Only block outputs written back to register file → ***fewer RF ports needed***
- Dynamic instruction issue
  - ALU array forms large distributed window with independent issue control
  - Instructions execute in original *dataflow-order*

TRIPS Processor Overview



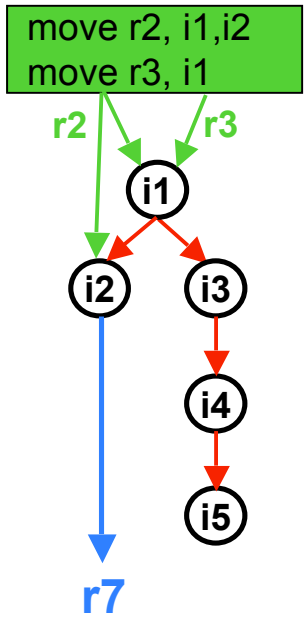
Block Compilation

Intermediate Code

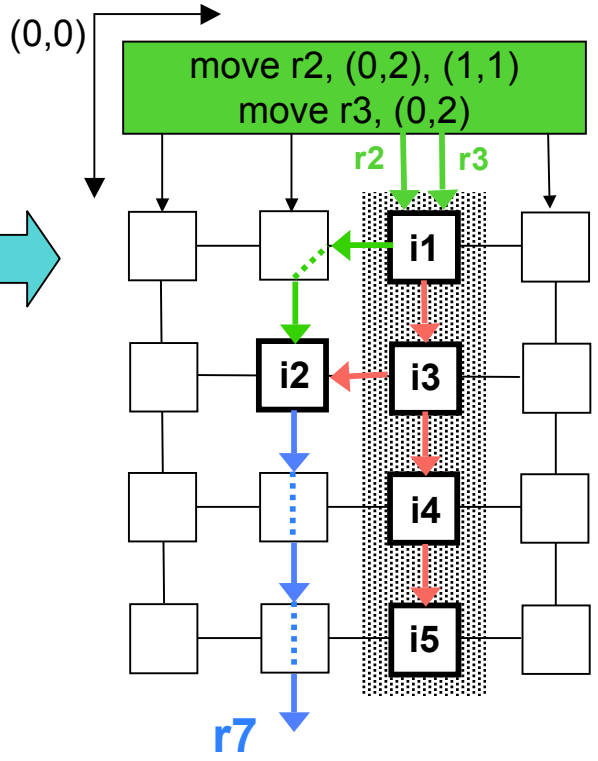
i1) add r1, r2, r3
i2) add r7, r2, r1
i3) ld r4, (r1)
i4) add r5, r4, 1
i5) beqz r5, 0xdeac

Inputs (r2, r3)
Temporaries (r1, r4, r5)
Outputs (r7)

Data flow graph



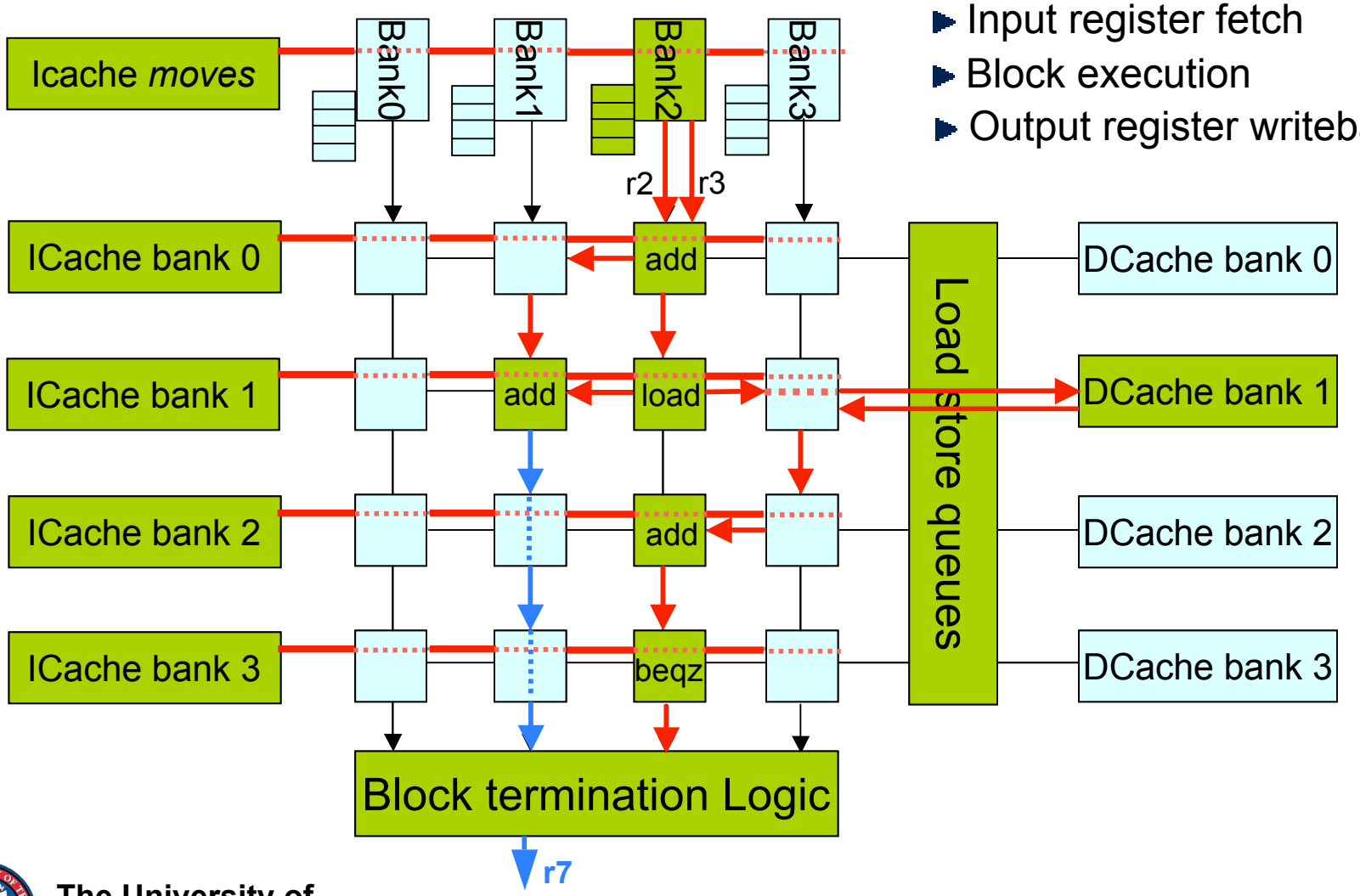
Mapping onto TRIPS



First, place ***critical path*** to minimize communication delays
Then place less critical paths to *maximize ILP*

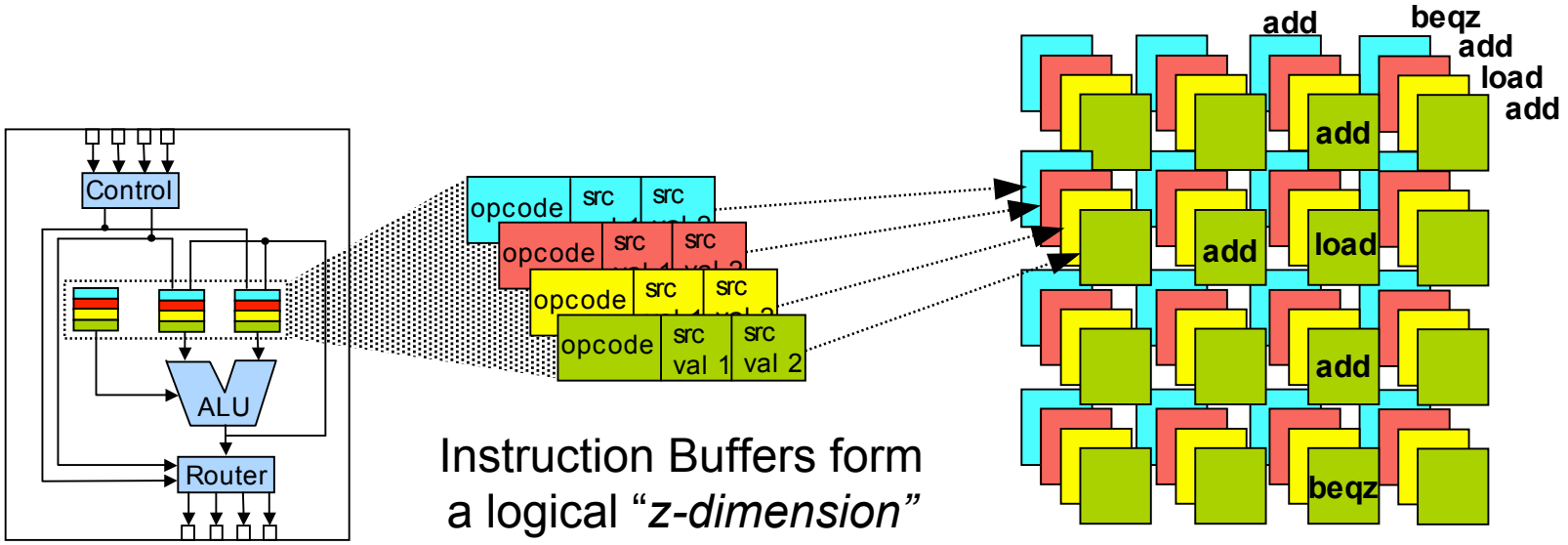
Block Execution

- Instruction distribution
- Input register fetch
- Block execution
- Output register writeback



Instruction Buffers: *Frames*

- Instruction Buffers add *depth* and define *frames*
  - 2D array of execution units; 3D scheduling of instructions
  - Allows very large blocks to be mapped onto a TRIPS processor
  - Result addresses explicitly specified in 3-dimensions (x,y,z)
  - Instructions execute in dataflow order, regardless of frame

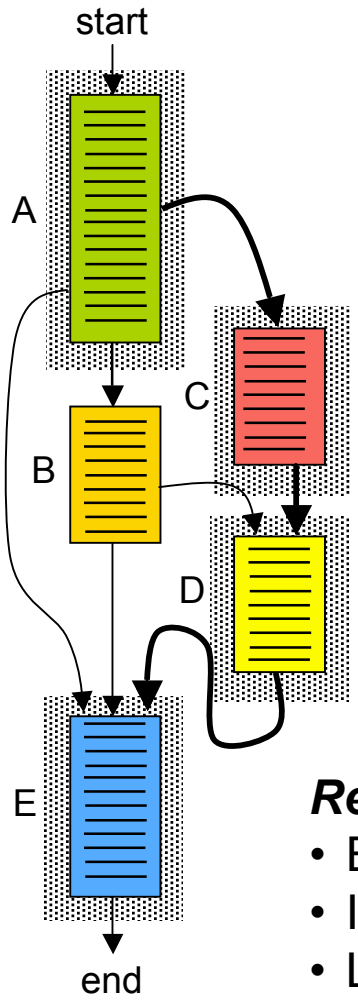


Execution Node

Instruction Buffers form a logical "*z-dimension*" in each node

4 logical *frames* each with 16 instruction slots

Using *Frames* for Speculation and ILP



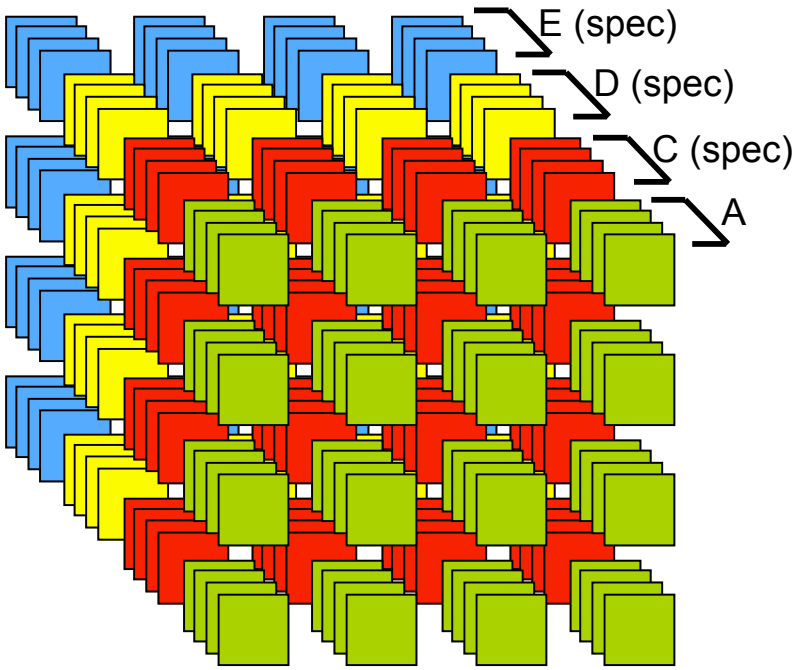
Map A onto array
Start executing A

Predict C is next block
Speculatively execute C

Predict is D is after C
Speculatively execute D

Predict is E is after D
Speculatively execute E

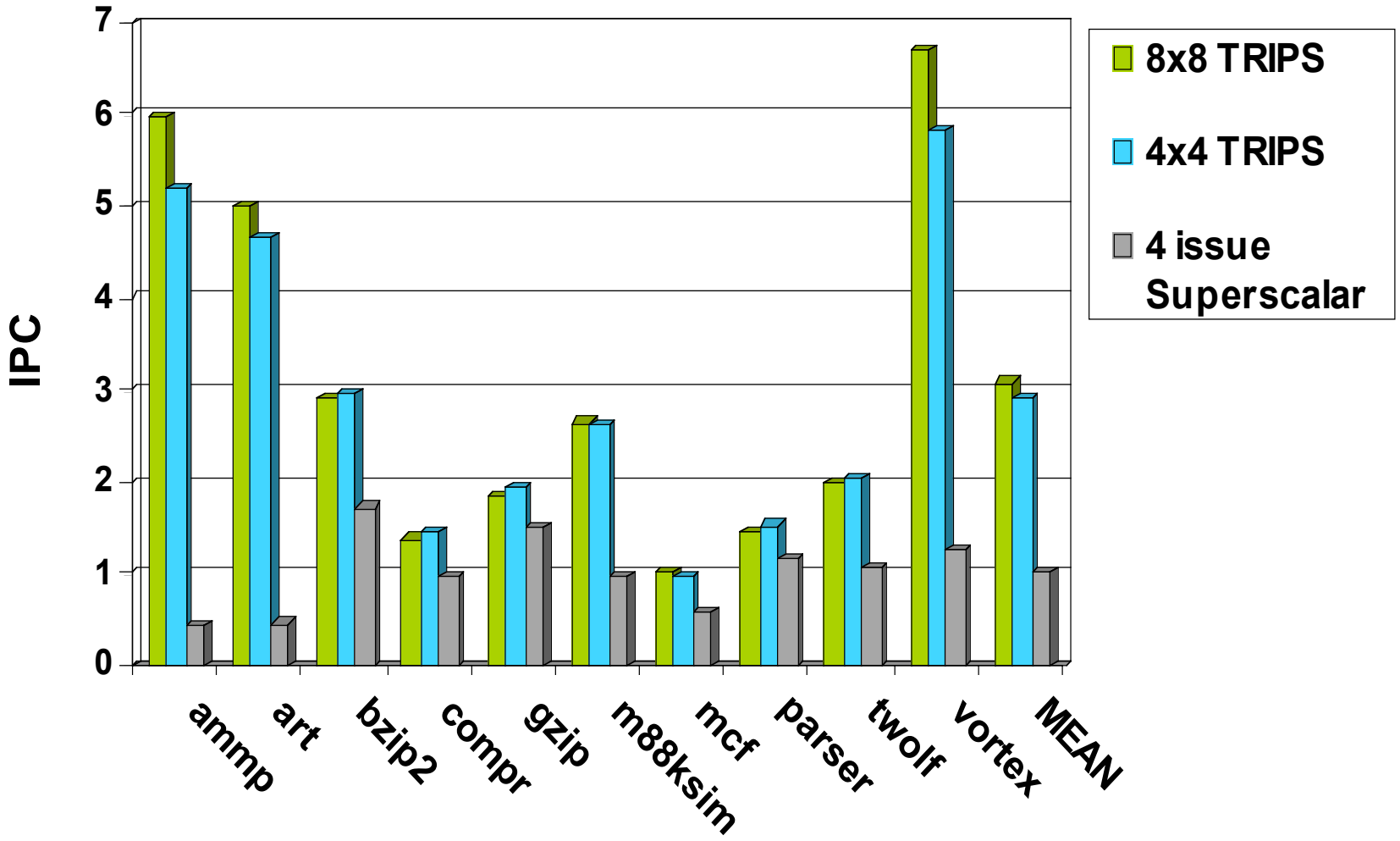
16 total frames (4 sets of 4)



Result:

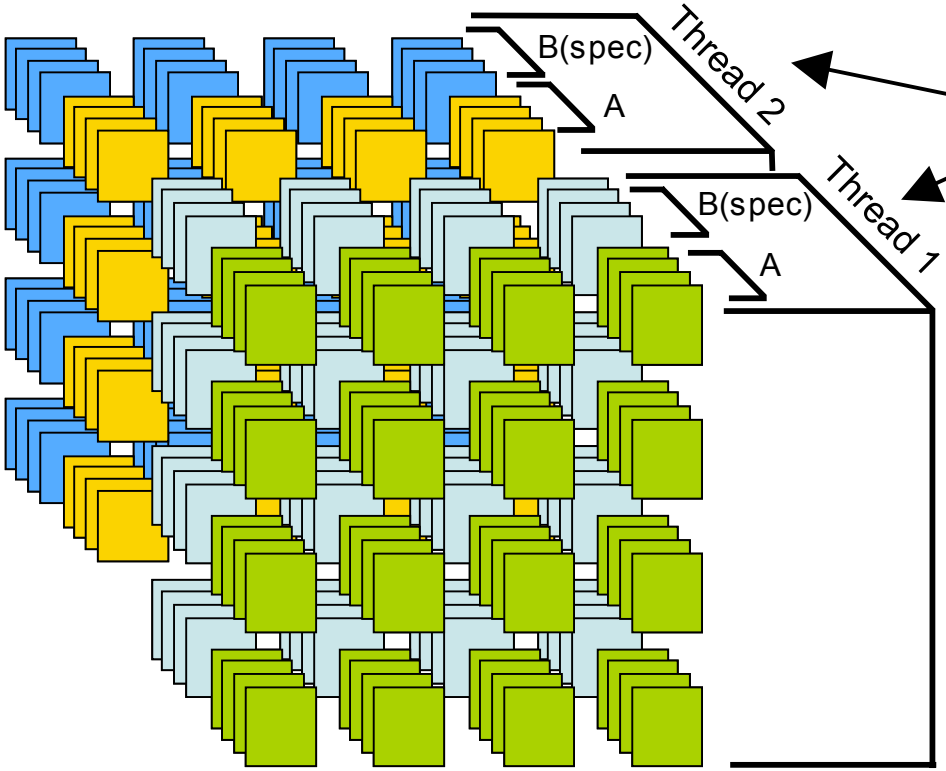
- Enormous effective instruction window for extracting ILP
- Increased utilization of execution units *(accuracy counts!)*
- Latency tolerance for interconnect delays and load instructions

Results – TRIPS Instructions per Cycle



The University of Texas at Austin

Using *frames* for TLP



Divide frame space among threads

Each can be further divided to enable some degree of speculation

Shown: *2 threads, each with 1 speculative block*

Alternate configuration might provide 4 threads

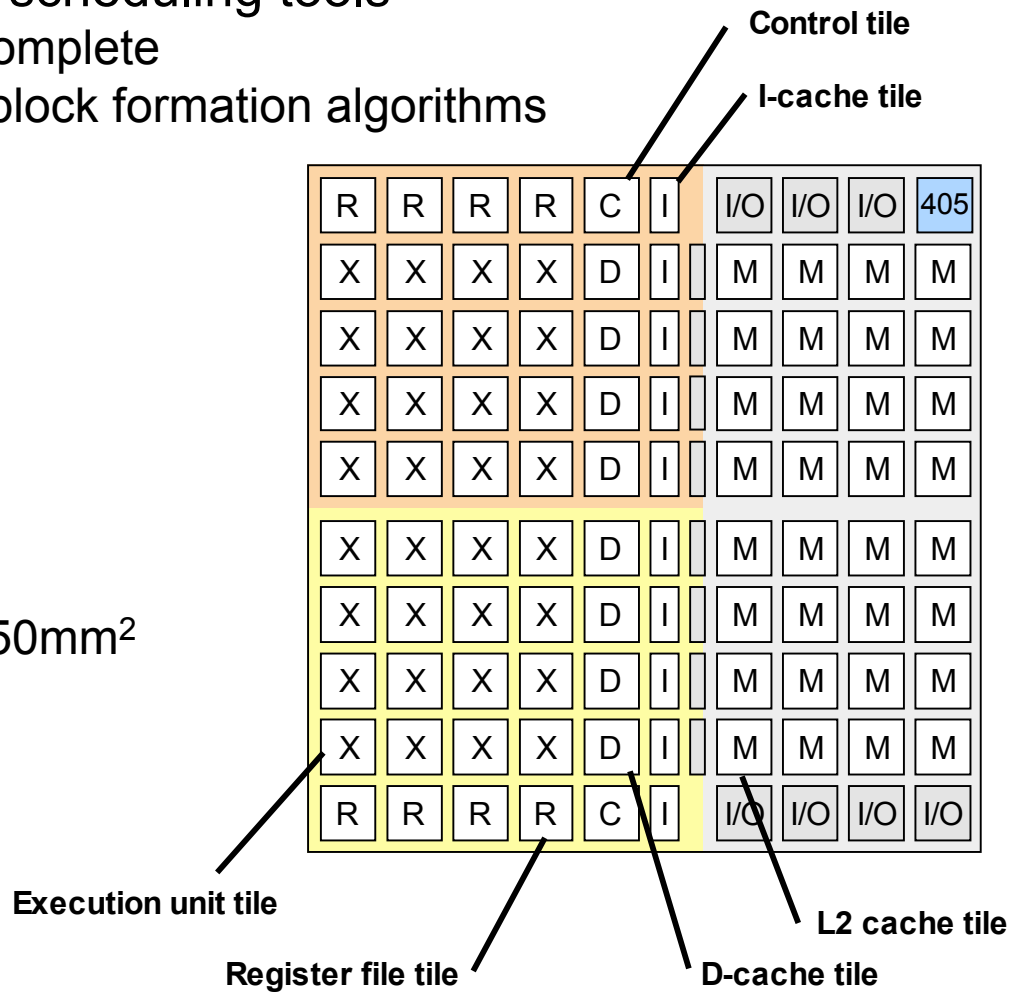
Result:

- Simultaneous Multithreading (SMT) for TRIPS processors
- *Polymorphism:* *Use same resources in different ways for different workloads ("T-morph")*

Current Status

- Architecture studies complete
- Enhancing compilation and scheduling tools
  - 3D instruction scheduler complete
  - Currently improving hyperblock formation algorithms

- TRIPS chip prototype
  - 2 4x4 TRIPS cores
    - 16-way issue cores
    - 1K instruction window
    - Up to 4 threads/core
  - NUCA L2 cache
  - Tiled architecture
  - ASIC process, 130nm, ~350mm^2
  - 1000+ signal I/O, 500MHz
  - 12 person design team
  - Q1 2005 tape-out



The University of Texas at Austin

Observations and Challenges

- Compatibility – TRIPS has a different binary interface
  - Variety of solutions in marketplace (IPF, Transmeta, etc.)
- Undersized blocks waste i-cache capacity and bandwidth
  - Code compression techniques may prove promising
- Compiler obligations – hyperblock formation, predication
  - But – scheduling burden diminishes relative to VLIW
- Exceptions – block precise, not instruction precise
  - Previous machines have supported *imprecise* exceptions

Conclusions

- Multiprocessor performance starts with a powerful uniprocessor
  - Contemporary architectures have limited scalability
- Technology trends indicate that it is time for a new architecture
  - Pipeline limitations, global wire delay, inefficient binary interface
- TRIPS represents a promising technology direction
  - Wire delay constraints: at microarchitecture *and* architecture
  - Eliminates difficult centralized structures dominating today's designs
  - Architectural partitioning encourages regularity and re-use
  - Enhanced information flow between compiler and hardware
  - Dataflow substrate also suitable for threaded and data-parallel computing
  - Power efficiency: no power-hungry structures, dataflow sub-graph execution