Scaling to the End of Silicon: Performance Projections and Promising Paths

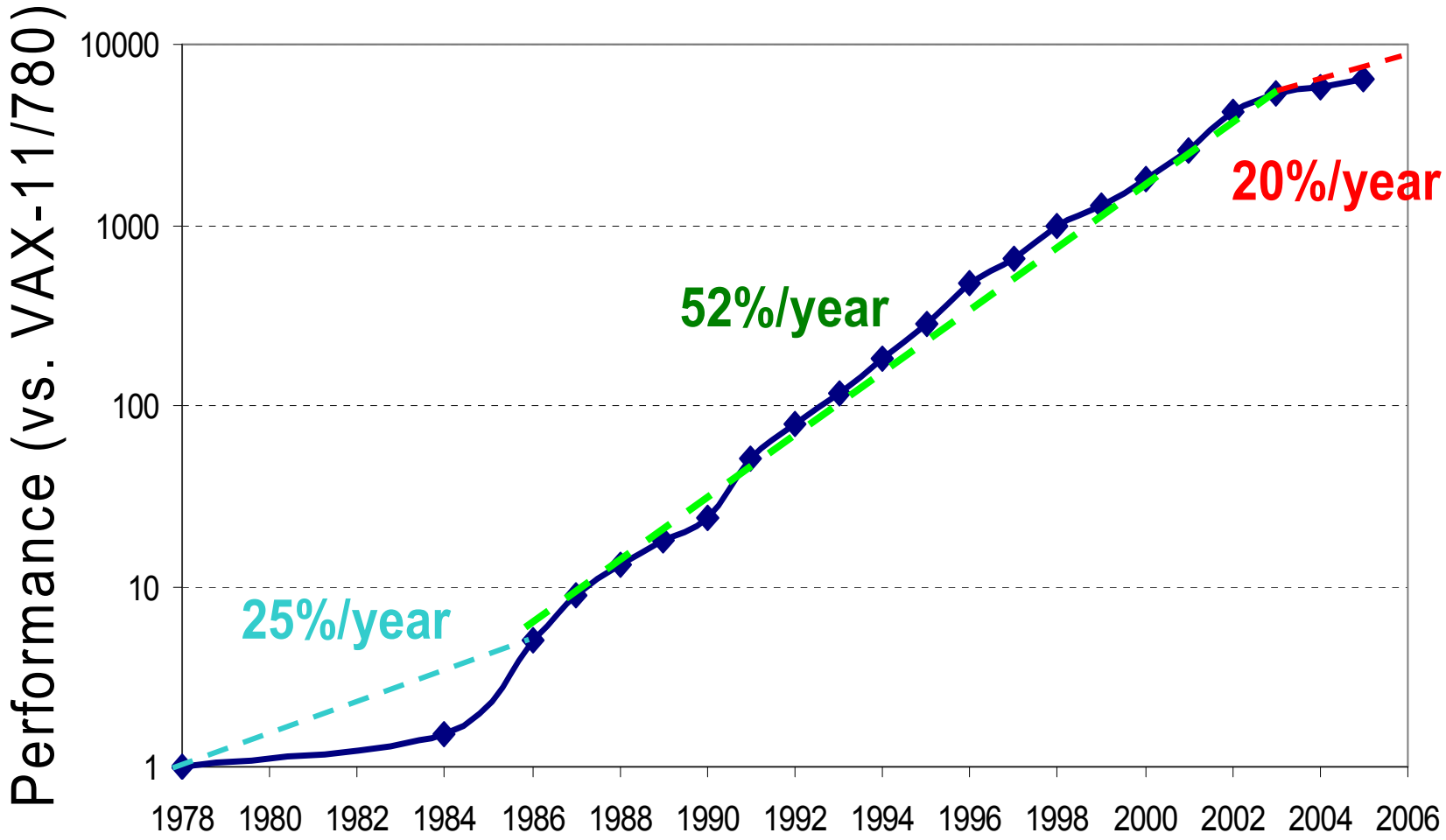
Frontiers of Extreme Computing
October 24, 2005

Doug Burger
The University of Texas at Austin

Where are HPC Systems Going?

- Scaling of uniprocessor performance has been historical driver
  - 50-55% per year for a significant period
  - Systems with a constant number of processors benefit
- Transistor scaling may continue to the end of the roadmap
  - However, system scaling must change considerably
  - The "last classical computer" will look very different from today's systems
- Outline of driving factors and views
  - Exploitation of concurrency - are more threads the only answer?
    - We are driving to a domain where tens to hundreds of thousands of processors are the sole answer for HPC systems
  - How will power affect system and architecture design?
  - How to provide the programmability, flexibility, efficiency, and performance future systems need?

Shift in Uniprocessor Performance

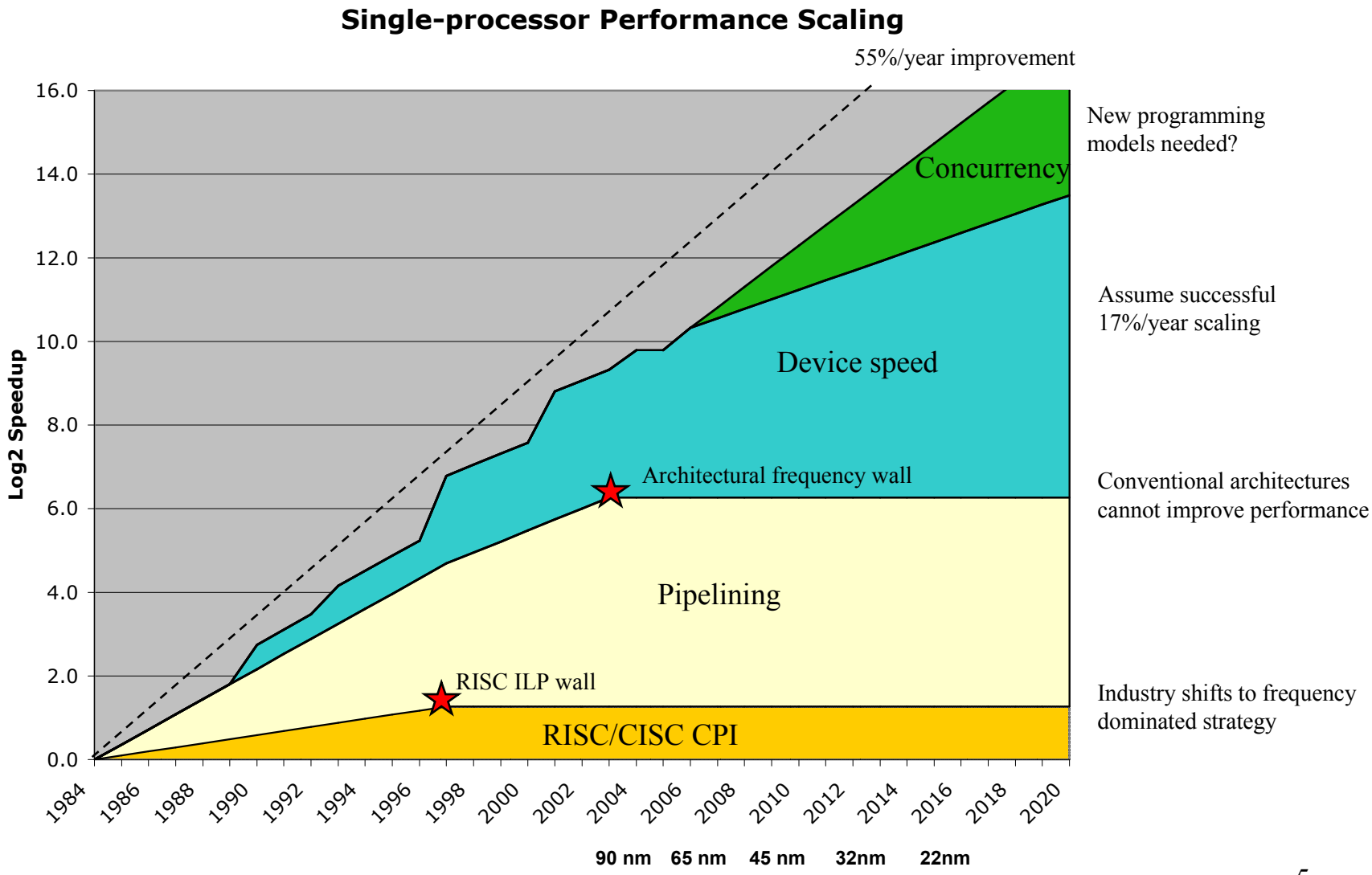


Slide by Dave Patterson

Historical Sources of Performance

- Four factors
  - Device speed (17%/year)
  - Pipelining (reduced FO4) - ~18%/year from 1990-2004
  - Improved CPI
  - Number of processors/chip - n/a
- Device speed will continue for some time
- Deeper pipelining is effectively finished
  - Due to both power and diminishing returns
  - Ends the era of 40%/year clock improvements
- CPI is actually increasing
  - Effect of deeper pipelines, slower memories
  - On-chip delays
  - Simpler cores due to power
- Number of processors/chip starting to grow
  - “Passing the buck” to the programmer
  - Have heard multiple takes on this from HPC researchers

Performance Scaling



Opportunity to End of Si Roadmap

- How much performance growth between now and 2020 per unit area of silicon?
  - 17% device scaling gives 10x performance boost
  - 50x increase in device count provides what level of performance?
  - Linear growth in performance: 500x performance boost
- What have we gotten historically?
  - 500x performance boost over that same period
  - However, a large fraction of that is increased frequency
  - Without that, historical boost would be 50X
  - The extra 10x needs to come from concurrency
- Opportunity
  - Many simpler processors per unit area provide more FLOP/transistor efficiency
  - May be efficiency issues (communication, load balancing)
  - May be programmability issues
- $64K question: how can we get that efficiency while circumventing the above problems?

Granularity versus Number of Processors

- Historically, designers opted for improved CPI over number of processors
- Shifting due to lack of CPI improvements (finite core issue widths)
  - What will be granularity of CMPs?
  - What will be power dissipation curves?
- Small number of heavyweight cores versus many lightweight cores?
- Problem with CMPs
  - Linear increase in per-transistor activity factors
  - If granularity is constant, number of processors scales with number of transistors
  - 32 lightweight processors today in 90nm become ~1000 at 17nm
  - Last-generation uniprocessor designs exploited lower transistor efficiency
  - Will bound the number of processors
- Interested in HPC researchers' thoughts on granularity issue
  - Key question: is the ideal architecture as many lightweight cores as possible, with frequency/device speed scaled down to make power dissipation tractable?

Scaling Power to Increase Concurrency

- Four strategies for reducing consumed power to increase per-chip concurrency exploitable
- 1) Adjust electrical parameters (supply/threshold voltages, high Vt devices, etc.)
  - Critical area, but more of a circuits and tools issue
- 2) Reduce active switching through tools (clock gating of unnecessary logic)
- 3) Reduce powered-up transistors
  - 5B transistors available
  - How many can be powered up at any time?
  - Answer depends on success at reducing leakage
  - Needs to be a different set, otherwise why build the transistors?
- 4) More efficient computational models

- Note: not factoring in the effects of unreliable devices/redundant computation!

Increasing Execution Efficiency

- Goal: reduce the energy consumed per operation
  - But must not hamstring performance
  - Example: eliminating prediction for branches
- Move work from the hardware to the compiler
  - Example 1: Superscalar -> VLIW
  - Example 2: Superscalar -> Explicit target architectures
- Microarchitectures that exploit less
  - Loop reuse in TRIPS
- This area will need to be a major focus of research

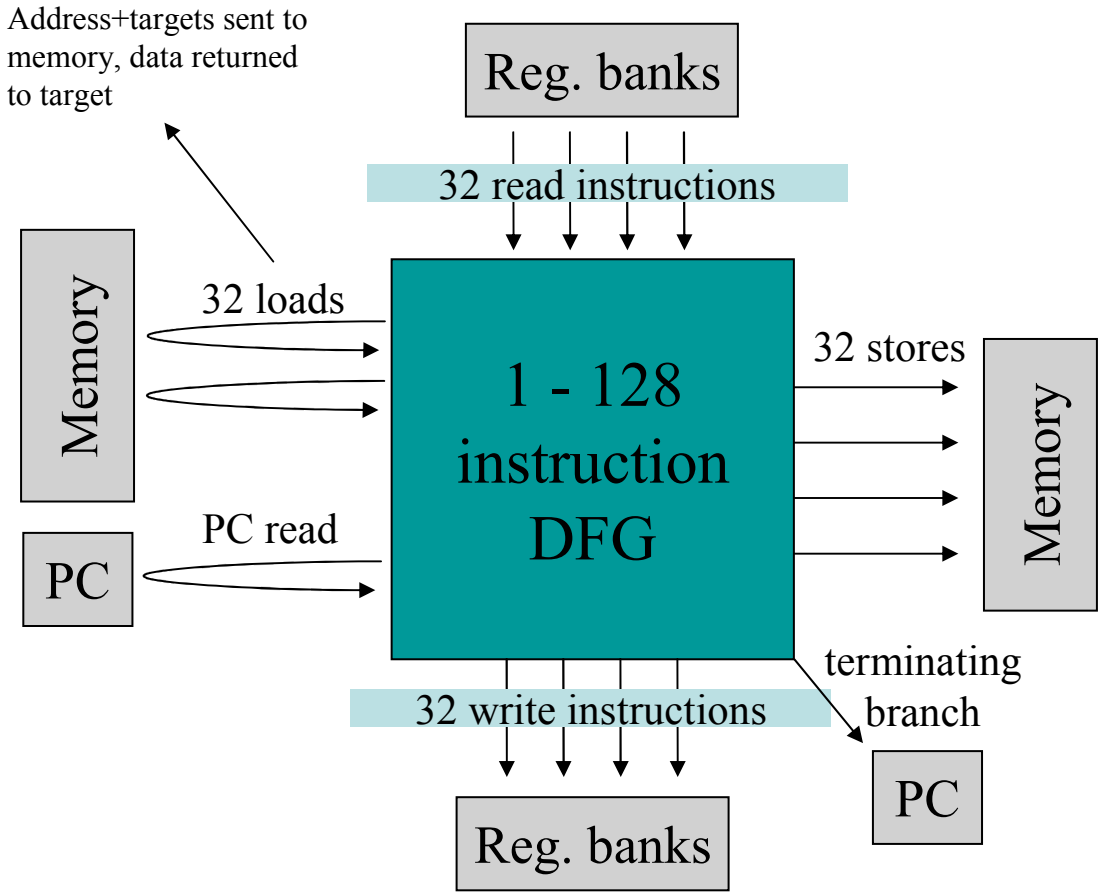
Example 1: Out-of-Order Overheads

- A day in the life of a RISC/CISC instruction
  - ISA does not support out-of-order execution
  - Fetch a small number of instructions
  - Scan them for branches, predict
  - Rename all of them, looking for dependences
  - Load them into an associative issue window
  - Issue them, hit large-ported register file
  - Write them back on a large, wide bypass network
  - Track lots of state for each instruction to support pipeline flushes
- BUT: performance from in-order architectures hurt badly by cache misses
  - Unless working set fits precisely in the cache
  - Take a bit hit in CPI, need that many more processors!

TRIPS Approach to Execution Efficiency

- EDGE (Explicit Data Graph Execution) architectures have two key features
  - Block-atomic execution
  - Direct instruction communication
- Form large blocks of instructions with no internal control flow transfer
  - We use hyperblocks with predication
  - Control flow transfers (branches) only happen on block boundaries
- Form dataflow graphs of instructions, map directly to 2-D substrate
  - Instructions communicate directly from ALU to ALU
  - Registers only read/written at begin/end of blocks
  - Static placement optimizations
    - Co-locate communicating instructions on same or nearby ALU
    - Place loads close to cache banks, etc.

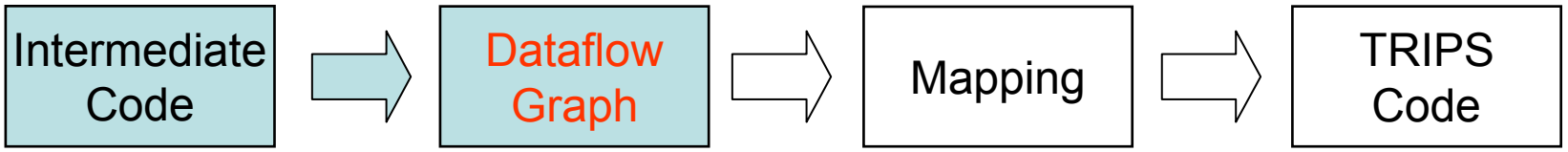
Architectural Structure of a TRIPS Block



Block characteristics:

- Fixed size:
  - 128 instructions max
  - L1 and core expands empty 32-inst chunks to NOPs
- Load/store IDs:
  - Maximum of 32 loads+stores may be emitted, but blocks can hold more than 32
- Registers:
  - 8 *read insts.* max to reg. bank (4 banks = max of 32)
  - 8 *write insts.* max to reg bank (4 banks = max of 32)
- Control flow:
  - Exactly one branch emitted
  - Blocks may hold more

Block Compilation

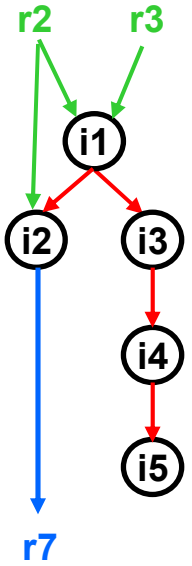


Intermediate Code

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

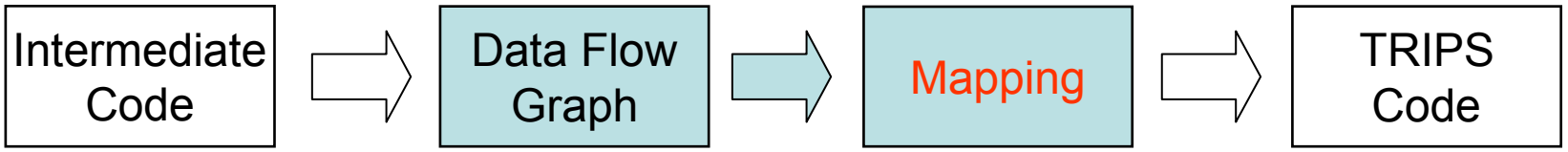
Compiler Transforms

Data flow graph

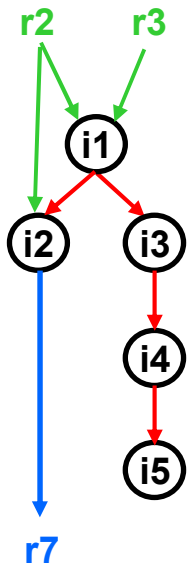


- Inputs (**r2, r3**)
- Temporaries (**r1, r4, r5**)
- Outputs (**r7**)

Block Mapping

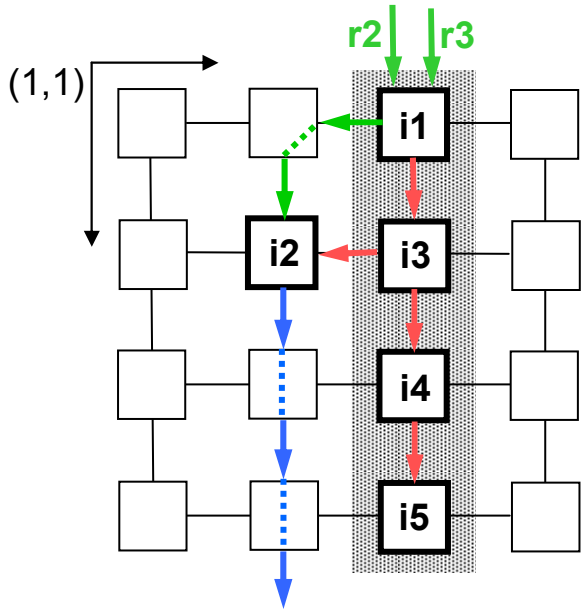


Data flow graph



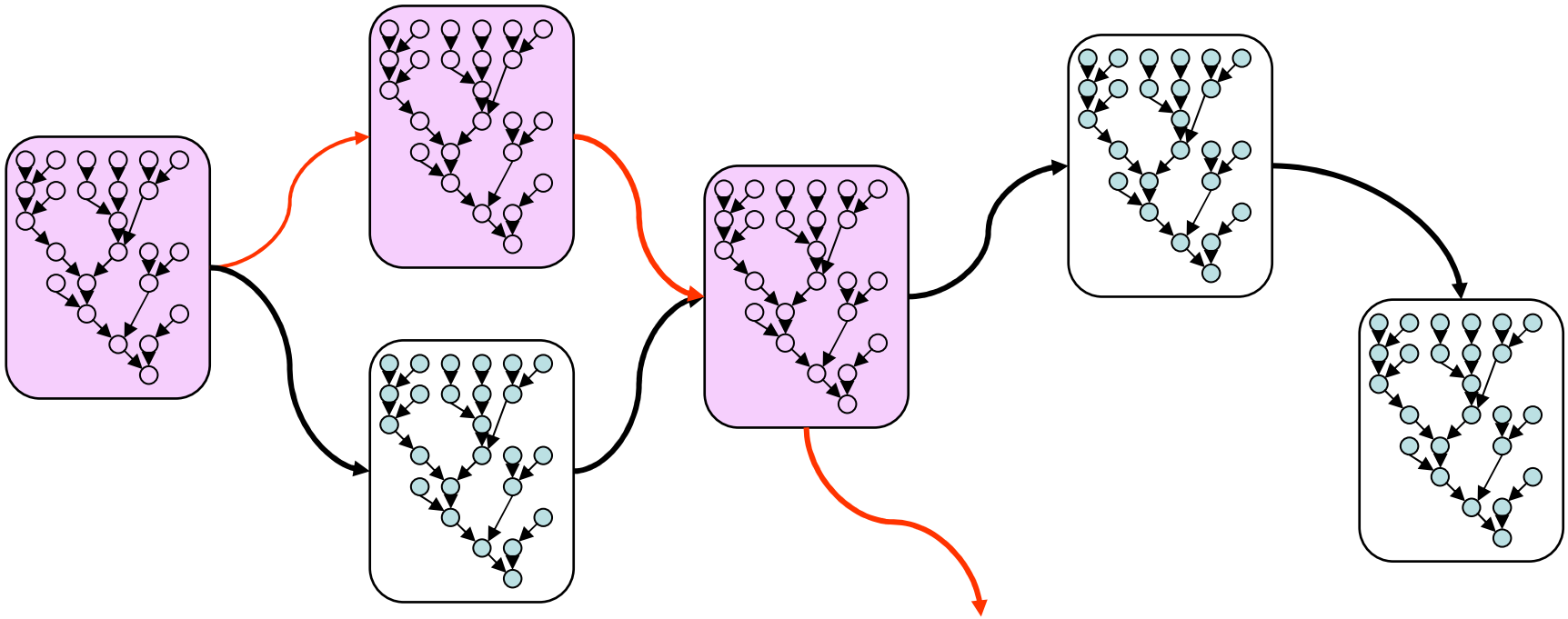
Scheduler

Mapping onto array

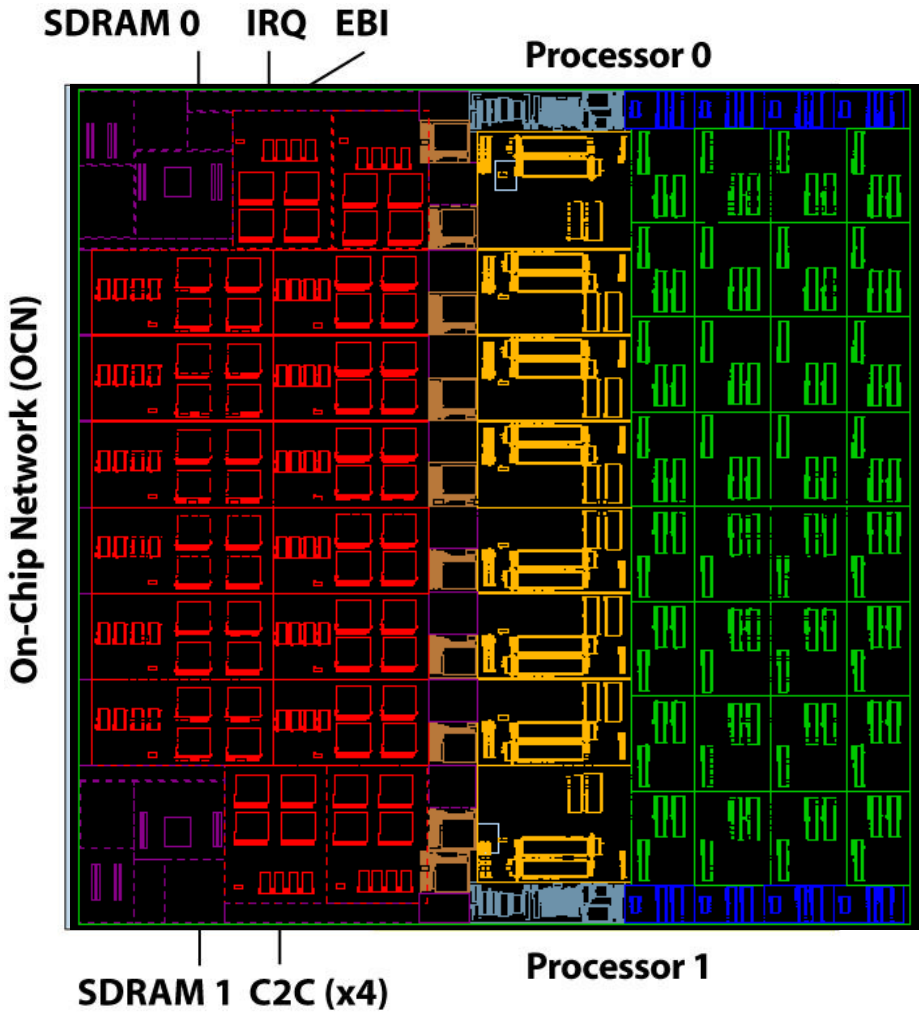


TRIPS Block Flow

- Compiler partitions program into “mini-graphs”
- Within each graph, instructions directly target others
- These mini-graphs execute like highly complex instructions
- Reduce per-instruction overheads, amortized over a block



Floorplan of First-cut Prototype

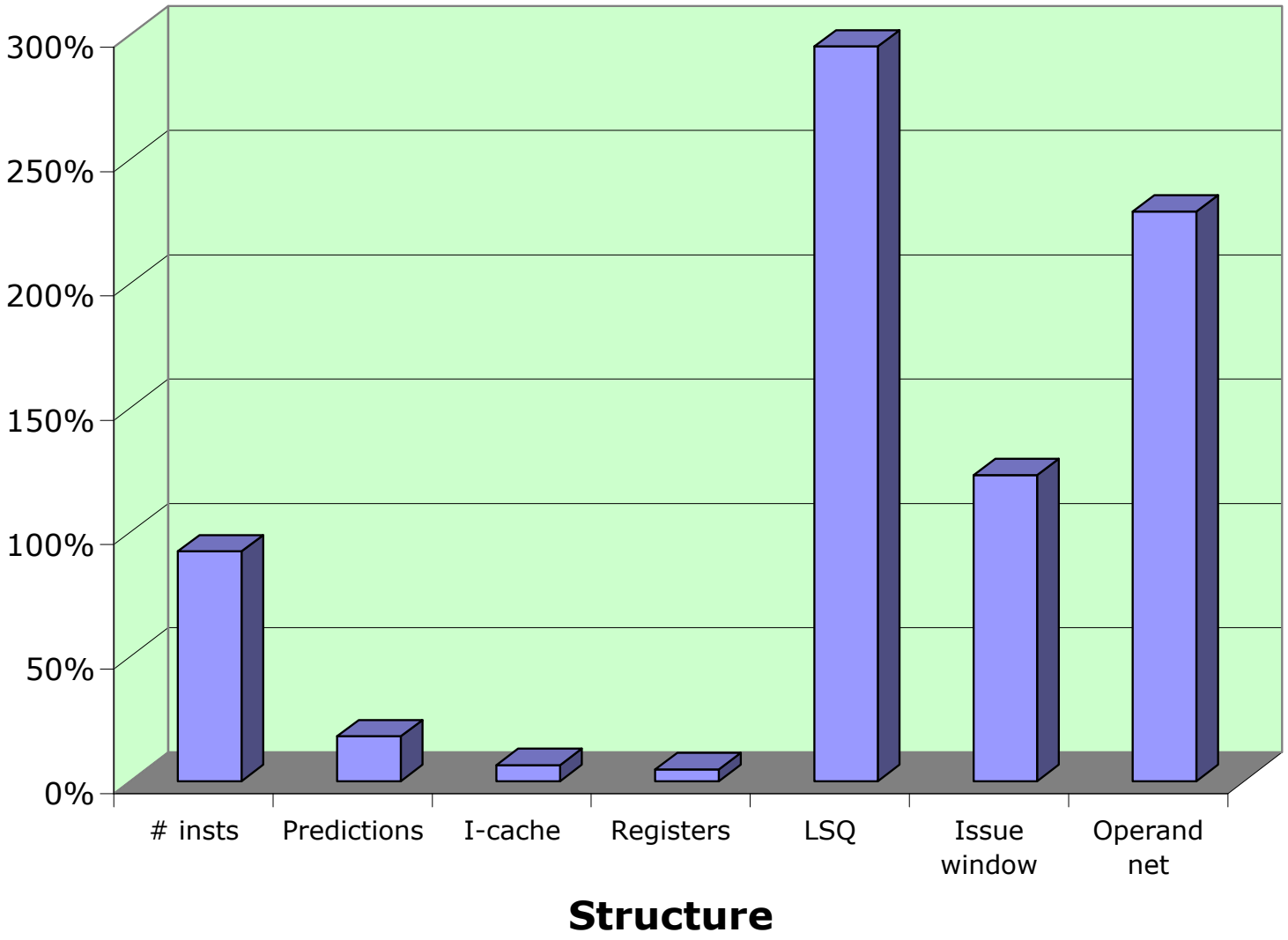


TRIPS Tiles and Interfaces

G: Processor control - dispatch, next block predictor, commit
R: Register file - 32 registers x 4 threads, register forwarding
I: Instruction cache - 16KB, 16-entry TLB variable-size pages
D: Data cache - 8KB, 64-entry load/store queue, 16-entry TLB
E: Execution unit - 128 reservation stations, integer/FP ALUs
M: Memory - 64KB, OCN router with 4 virtual channels
N: OCN network interface - OCN router, PA translation
DMA: Direct memory access controller
SDC: SDRAM controller
EBC: External bus controller (to PowerPC)
C2C: Chip-to-chip network links - to four neighbors

IRQ: Interrupt request - service request to PowerPC
EBI: External bus interfaces - command interface from PPC

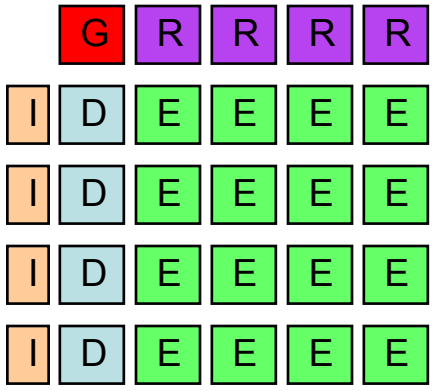
TRIPS/Alpha Activity Comparison

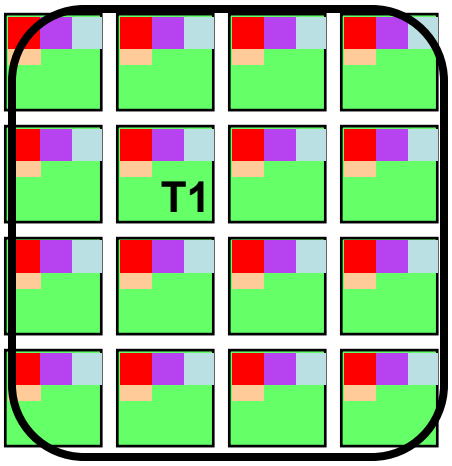


Example 2: Loop Reuse

- Simple microarchitectural extension: loop reuse
- If block that has been mapped to a set of reservation stations is the same as the next one to be executed, just refresh the valid bits
  - Signal can be piggybacked on block commit signal
  - Re-inject block inputs to re-run the block
- Implication for loops: fetch/decode eliminated while in the loop
  - Further gain in energy efficiency
- Loop must fit into the processor issue window
  - How to scale up the issue window for different loops?

Multigranular "Elastic" Threads

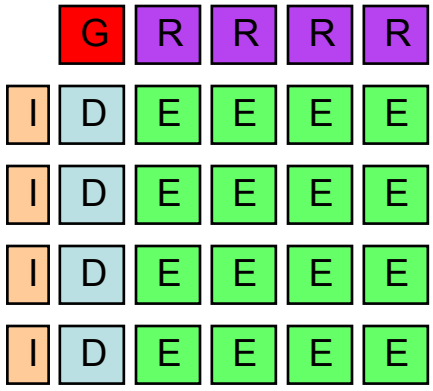


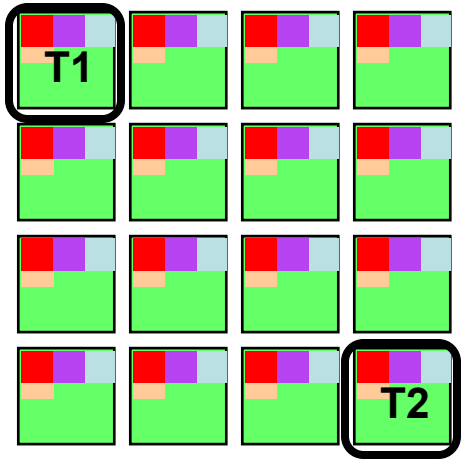


- Problems with TRIPS microarchitecture
  - Limited register/memory bandwidth
  - Number of tiles per core is fixed at design time
  - Multithreading is a hack to vary granularity
- Achievable by distributing all support tiles
  - Assume each tile can hold >= 1 block (128 insts.)
- Solutions being implemented to design challenges
  - Scalable cache capacity with number of tiles
  - Scalable memory bandwidth (at the processor interface)
- Does not address chip-level memory bandwidth

- **Config one: 1 thread, 16 blocks @ 8 insts/tile**
- **Config two: 2 threads, 1 block @ 128 insts/tile**
- **Config three: 6 threads, 1 thread on 8 tiles, 1 thread on 4 tiles, 4 threads on 1 tile each**

Multigranular "Elastic" Threads

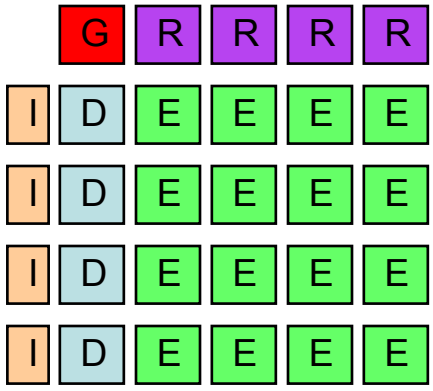


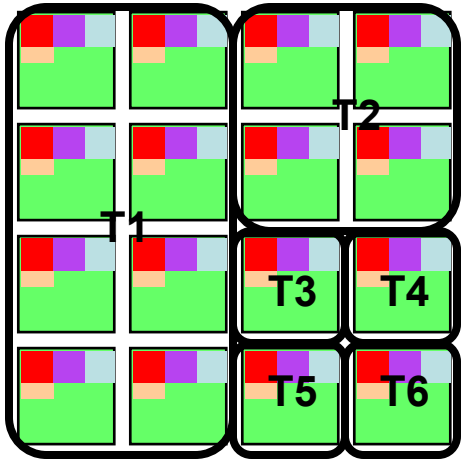


- Problems with TRIPS microarchitecture
  - Limited register/memory bandwidth
  - Number of tiles per core is fixed at design time
  - Multithreading is a hack to vary granularity
- Achievable by distributing all support tiles
  - Assume each tile can hold >= 1 block (128 insts.)
- Solutions being implemented to design challenges
  - Scalable cache capacity with number of tiles
  - Scalable memory bandwidth (at the processor interface)
- Does not address chip-level memory bandwidth

- **Config one: 1 thread, 16 blocks @ 8 insts/tile**
- **Config two: 2 threads, 1 block @ 128 insts/tile**
- **Config three: 6 threads, 1 thread on 8 tiles, 1 thread on 4 tiles, 4 threads on 1 tile each**

Multigranular "Elastic" Threads



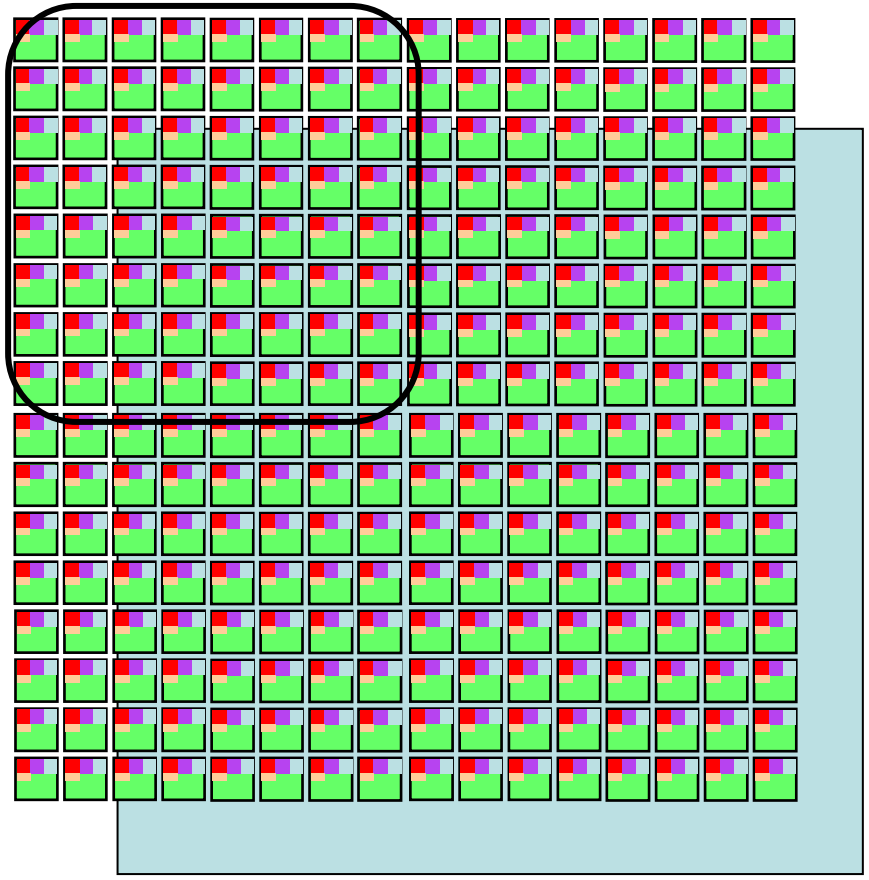


- Problems with TRIPS microarchitecture
  - Limited register/memory bandwidth
  - Number of tiles per core is fixed at design time
  - Multithreading is a hack to vary granularity
- Achievable by distributing all support tiles
  - Assume each tile can hold >= 1 block (128 insts.)
- Solutions being implemented to design challenges
  - Scalable cache capacity with number of tiles
  - Scalable memory bandwidth (at the processor interface)
- Does not address chip-level memory bandwidth

- **Config one: 1 thread, 16 blocks @ 8 insts/tile**
- **Config two: 2 threads, 1 block @ 128 insts/tile**
- **Config three: 6 threads, 1 thread on 8 tiles, 1 thread on 4 tiles, 4 threads on 1 tile each**

Looking forward

Map thread to PEs based on granularity, power, or cache working set



3-D integrated memory (stacked DRAM, MRAM, optical I/O)

- 2012-era EDGE CMP
  - 8GHz at reasonable clock rate
  - 2 TFlops peak
  - 256 PEs
  - 32K instruction window
- Flexible mapping of threads to Pes
  - 256 small processors
  - Or, small number of large processors
  - Embedded network
- Need high-speed BW
- Ongoing analysis
  - What will be power dissipation?
  - How well does this design compare to fixed-granularity CMPs?
  - Can we exploit direct core-to-core communication without killing the programmer?

Conclusions

- Potential for 2-3 orders of magnitude more performance from CMOS
- Significant uncertainties remain
  - How well will the devices scale?
  - What are application needs, how many different designs will they support?
- Concurrency will be key
- Must use existing silicon much more efficiently
  - How many significant changes will the installed base support?
  - New ISAs? New parallel programming models?
- Architecture community can use guidance on these questions