Static Placement, Dynamic Issue (SPDI) Scheduling for EDGE Architectures

Calvin Lin

Ramadass Nagarajan, Sundeep Kushwaha, Doug Burger, Kathryn S. McKinley, Stephen W. Keckler

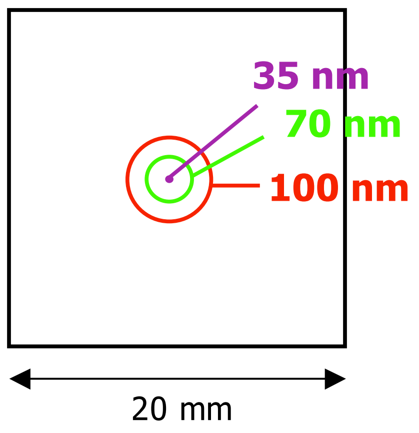
Department of Computer Sciences

The University of Texas at Austin

October 1, 2004

Architecture and Technology Trends

- Increasing wire delays limit sizes of monolithic structures [Agarwal, ISCA'00]
  - Need aggressive partitioning



- Clock rate growths show diminishing returns
  [Hrishikesh, ISCA'02] [Sprangle, ISCA'02]
  - Deeper pipelines approaching optimal limits
  - Need to improve instruction throughput (IPC)

- Conventional architectures and their schedulers are not equipped to deal with these trends

The Problem with Conventional Approaches

- VLIW approach
  - Relies completely on compiler to schedule code
  - + Eliminates need for dynamic dependence check hardware
  - + Good match for partitioning
  - + Can minimize communication latencies on critical paths
  - – Poor tolerance to unpredictable dynamic latencies
    - – These latencies continue to grow

- Superscalar approach
  - Hardware dynamically schedules code
  - + Can tolerate dynamic latencies
  - – Quadratic complexity of dependence check hardware
  - – Not a good match for partitioning
    - – Difficult to make good placement decisions
    - – ISA does not allow software to help with instruction placement

Dissecting the Problem

- Scheduling is a two-part problem
  - Placement: *Where an instruction executes*
  - Issue: *When an instruction executes*

- VLIW represents one extreme
  - Static Placement and Static Issue (SPSI)
  - + Static Placement works well for partitioned architectures
  - – Static Issue causes problems with unknown latencies

- Superscalars represent another extreme
  - Dynamic Placement and Dynamic Issue (DPDI)
  - + Dynamic Issue tolerates unknown latencies
  - – Dynamic Placement is difficult in the face of partitioning

Our Solution: EDGE Architectures

- EDGE: Explicit Dataflow Graph Execution
  - Supports Static Placement and Dynamic Issue (SPDI)
  - Renegotiates the compiler/hardware binary interface

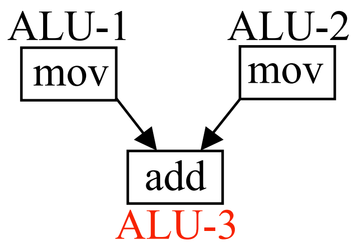
- An EDGE ISA explicitly encodes the dataflow graph specifying *targets*

RISC

```
i1:  movi r1, #10
i2:  movi r2, #20
i3:  add r3, r2, r1
```

EDGE

```
ALU-1:  movi #10, ALU-3
ALU-2:  movi #20, ALU-3
ALU-3:  add ALU-4
```



- Static Placement
  - Explicit DFG simplifies hardware → ***no HW dependency analysis!***
  - Results are forwarded directly through point-to-point network → ***no associative issue queues!*** → ***no global bypass network!***

- Dynamic Instruction Issue
  - Instructions execute in original *dataflow-order*

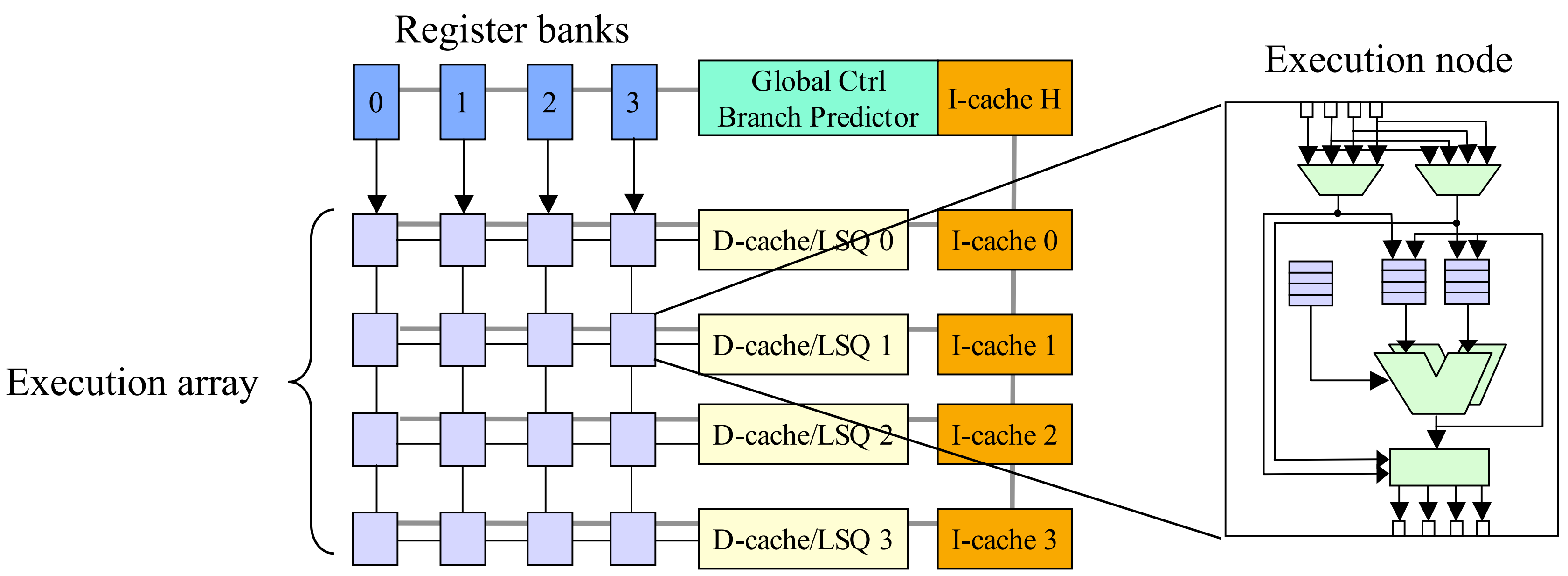
Static Placement and Dynamic Issue (SPDI)

- Combines strengths of static and dynamic schedulers
  - Static Placement (SP)
  - Dynamic Issue (DI)
- Benefits for the static scheduler
  - Precise timing information not required
  - Can convey placement information to the hardware
- Benefits for the dynamic scheduler
  - No associative tag match
  - Tolerates dynamic latencies
- Scheduling Goals
  - Spread parallelism among numerous execution resources
  - Minimize on-chip communication latencies

Outline

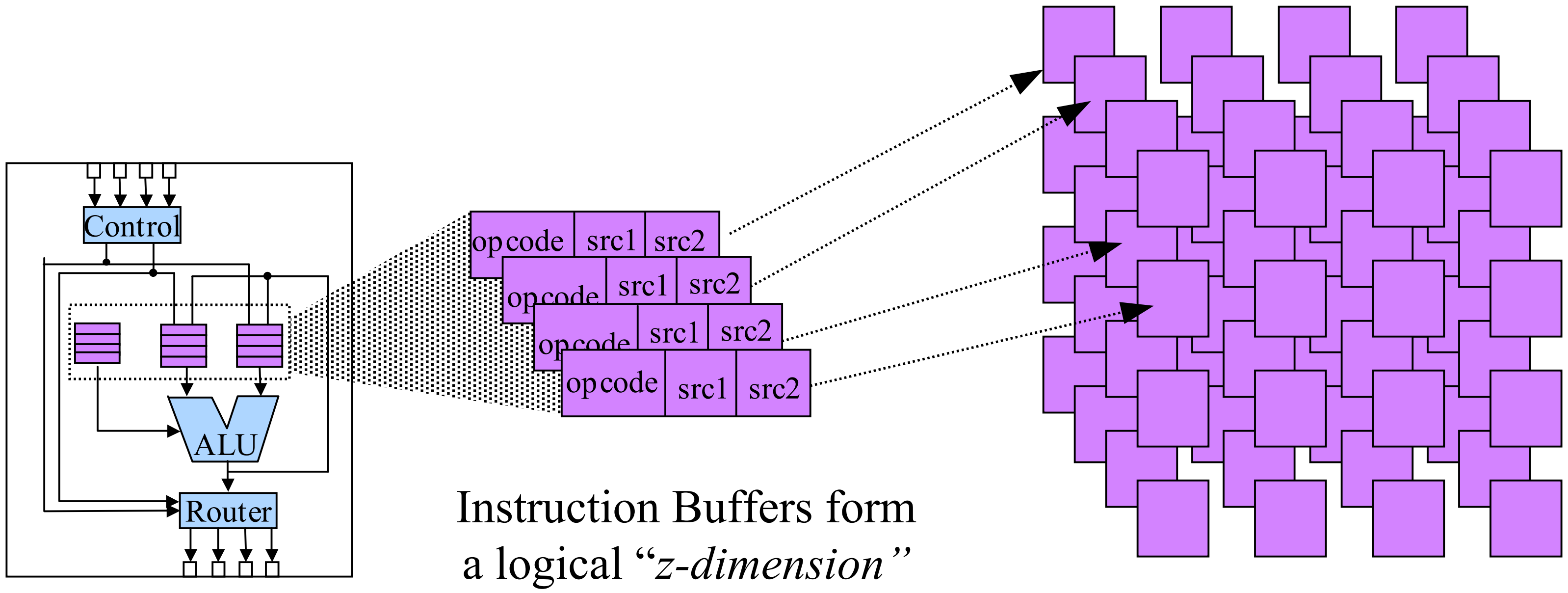
- Architectual Overview
  - Execution substrate
  - Scheduling problem
- SPDI scheduling algorithm
  - Locality optimizations
  - Contention optimizations
- Experimental results
- Conclusions

TRIPS Architecture



- Topology and latency of interconnect exposed to the static scheduler
- Reduced register pressure

The Scheduling Problem



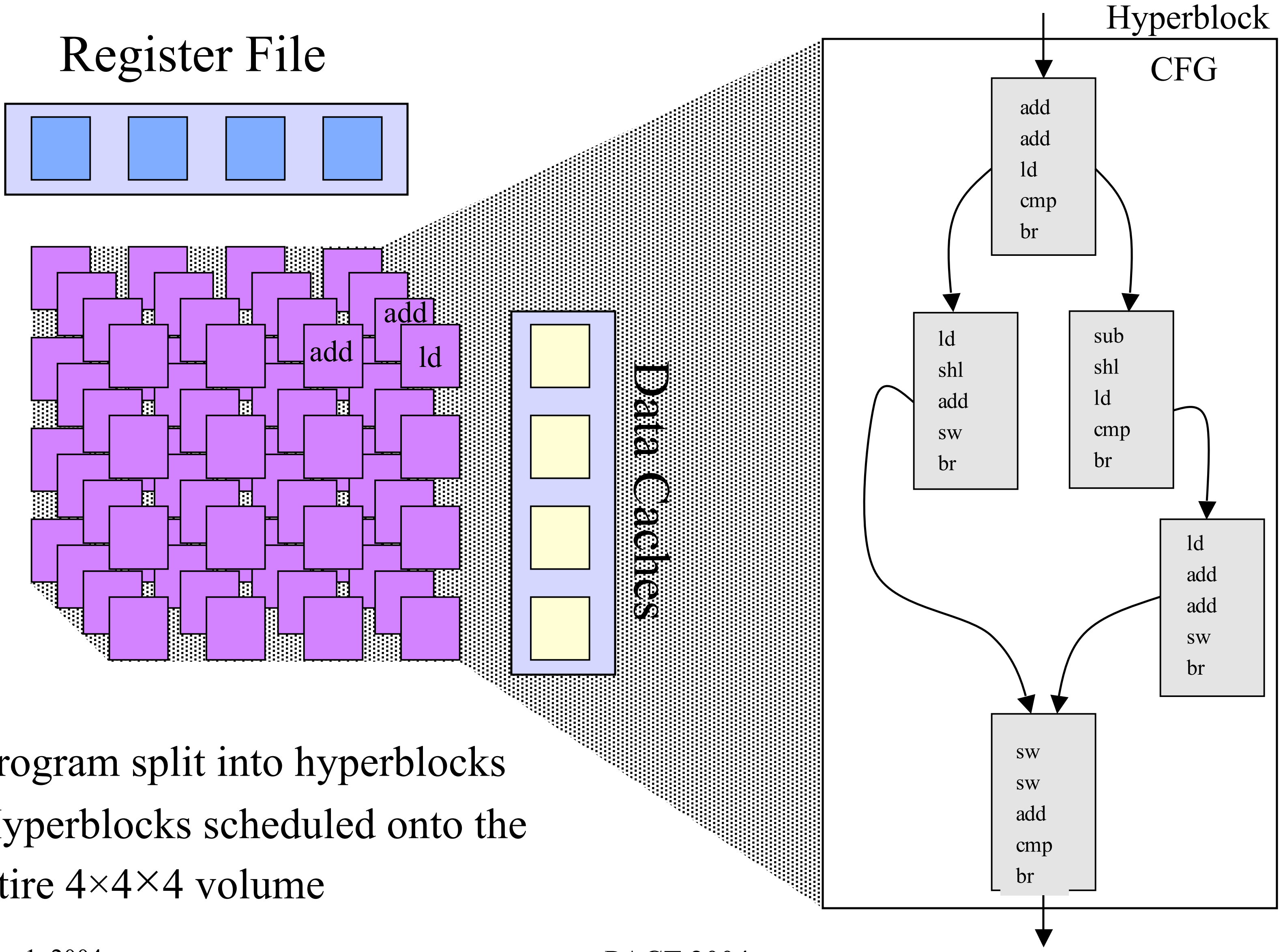
Execution Node

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

3D scheduling problem

- Instruction buffers add depth to the execution array
  - 2D array of ALUs; 3D volume of instructions

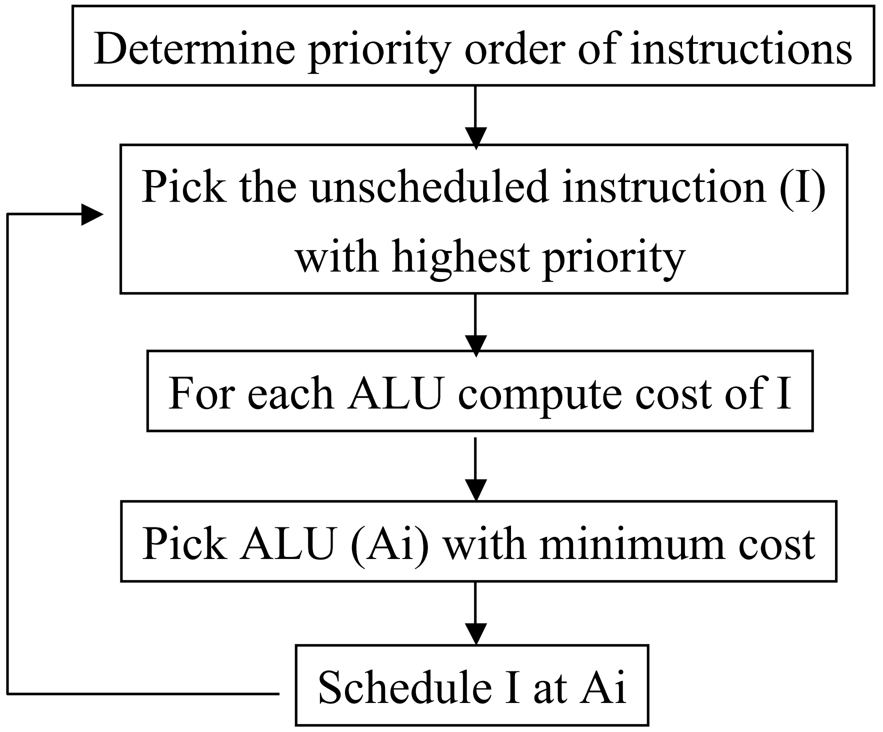
Static Scheduling Problem

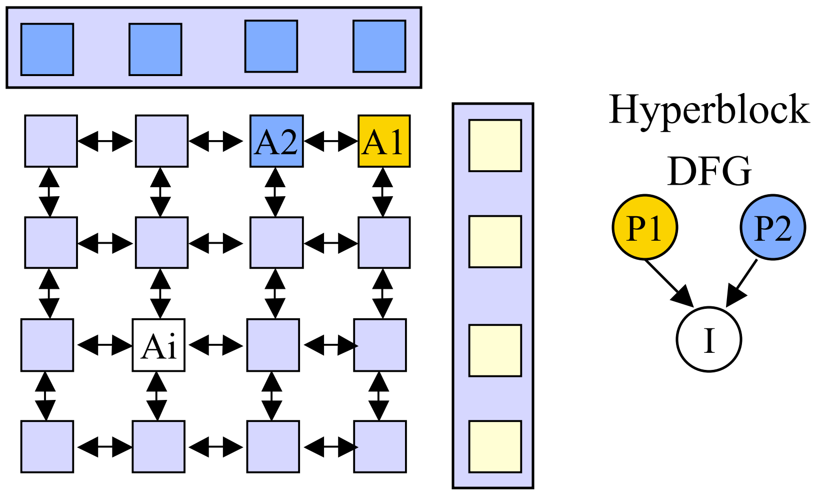


•Program split into hyperblocks

•Hyperblocks scheduled onto the entire 4×4×4 volume

List Scheduling Algorithm


Determine priority order of instructions
Pick the unscheduled instruction (I) with highest priority
For each ALU compute cost of I
Pick ALU (Ai) with minimum cost
Schedule I at Ai

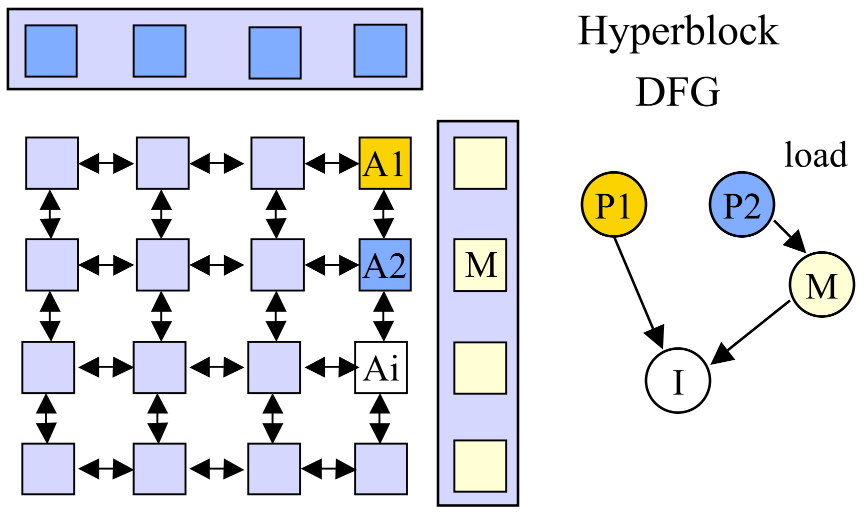


Cost[I] = max (Cost[P1]+Distance[A1,Ai],
Cost[P2]+Distance[A2,Ai])
+
Latency(I)

- Local algorithm – one hyperblock at a time
- No backtracking or re-placement of instructions

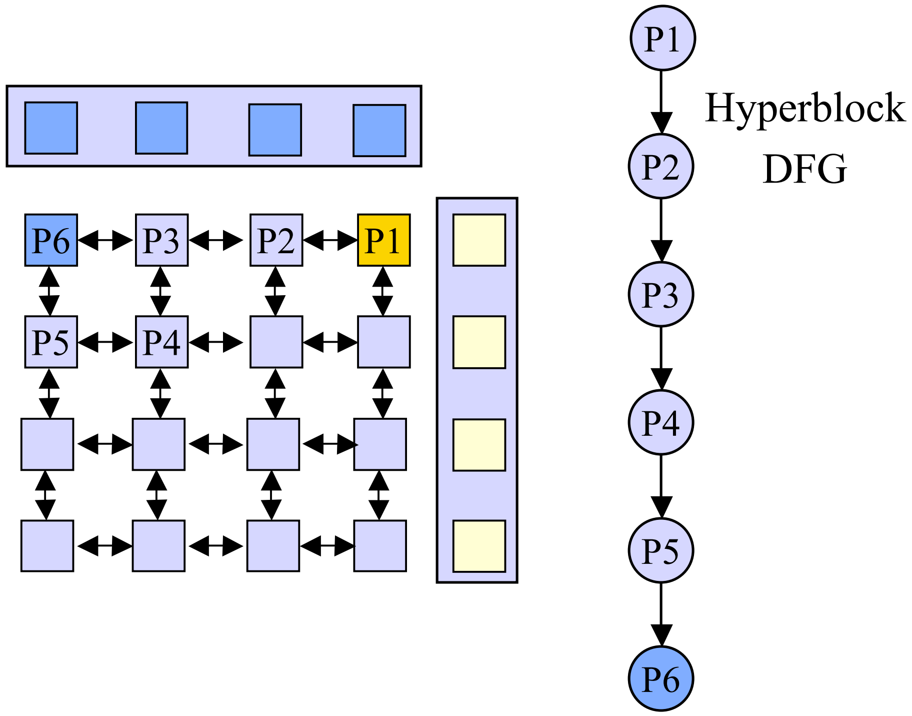
Scheduler Optimizations: 1 of 2

- Balance load among ALUs
  - Estimate ALU contention
- Locality optimization
  - Place loads and their consumers close to caches
  - Place register reads close to registers



Cost[I] = max (Cost[P1]+Distance[A1,Ai]
Cost[P2]+Distance[A2,Ai])
+
Contention (Ai)
+
Latency(I)

Scheduler Optimizations: 2 of 2



Cost[I] = max (Cost[P1]+Distance[A1,Ai]
Cost[P2]+Distance[A2,Ai])
+
Contention (Ai)
+
Lookahead (I)
+
Latency(I)

- Lookahead optimization
  - Estimate future use for register outputs or loads

- Critical path re-computation

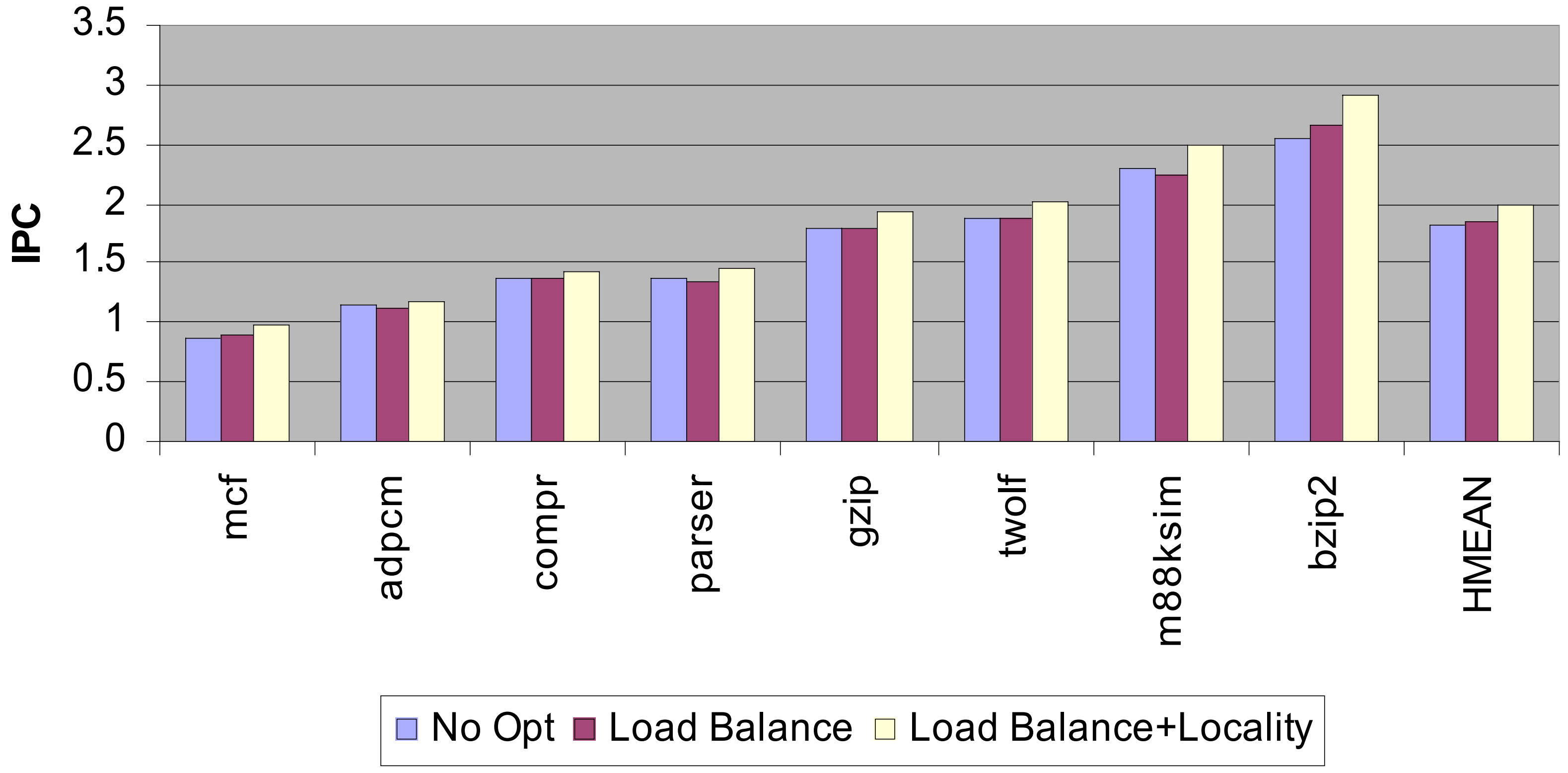
Prototype Evaluation

- Experimental Methodology
  - Use Trimaran infrastructure to produce hyperblocks
  - Schedule instructions using a custom greedy scheduler
  - Evaluate performance using a detailed microarchitecture simulator

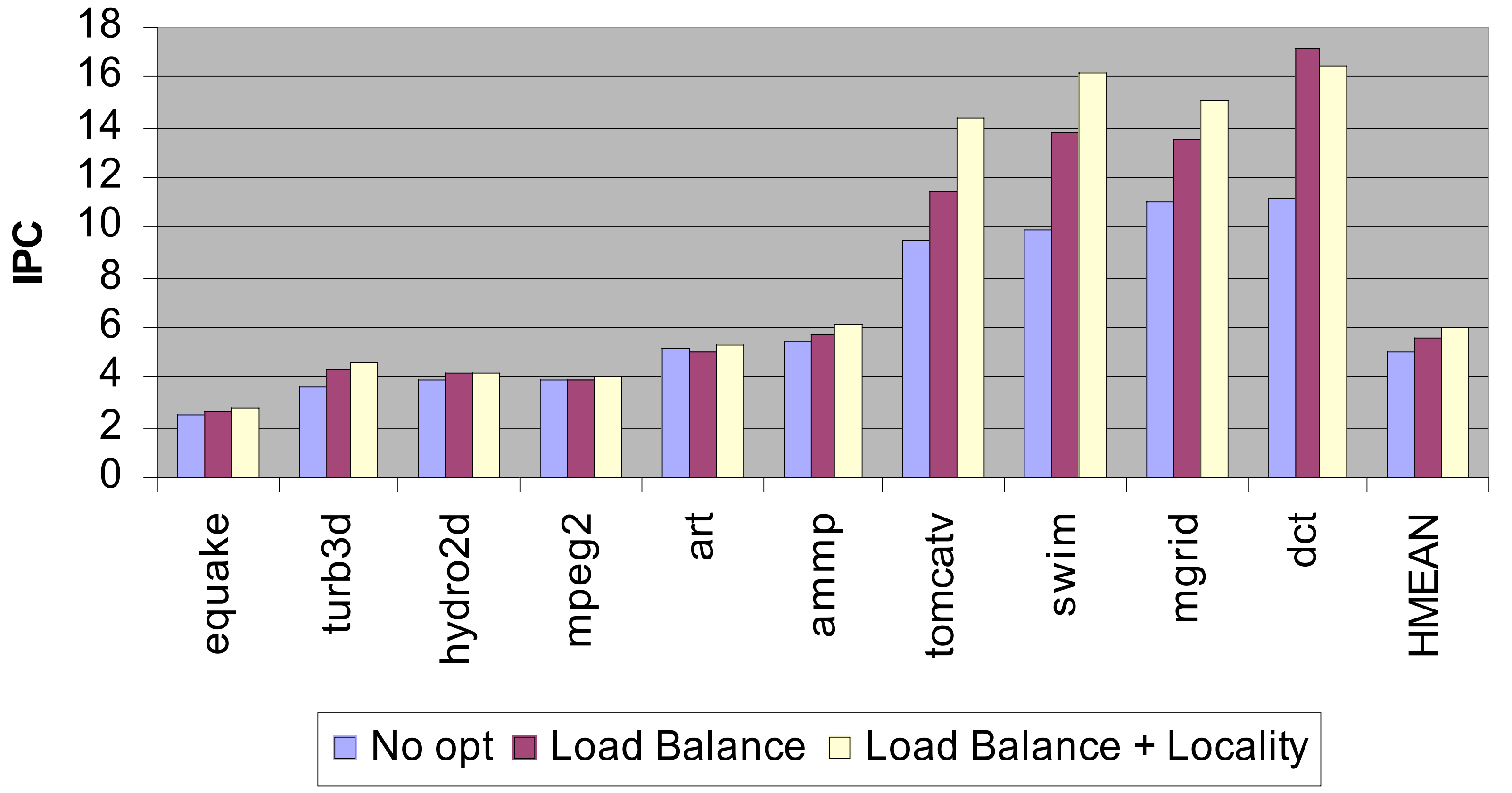
- Simulated Machine Parameters
  - 8×8 array of ALUs, 128 instruction slots
  - 0.5 cycle hop-hop latency
  - 64KB, 2-way L1 Instruction and L1 Data caches
  - 32Kbits two-level local/global tournament-style branch predictor
  - Optimistic assumptions: Oracular memory disambiguation, no TLBs, centralized data cache

- Benchmarks
  - 8 SpecInt, 8 SpecFP, 3 MediaBench

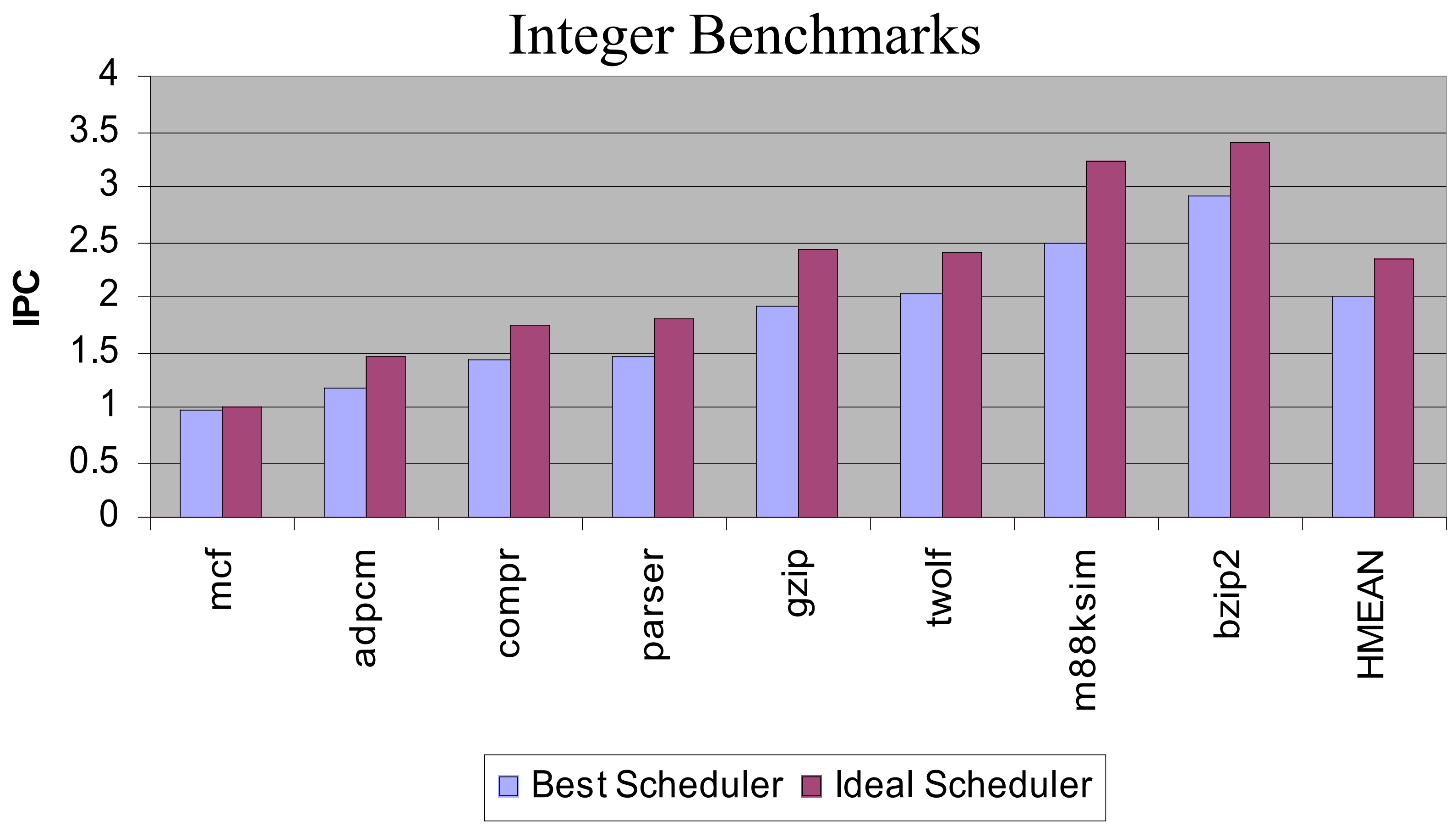
Scheduler Results – Integer Benchmarks



Scheduler Results – Floating Point

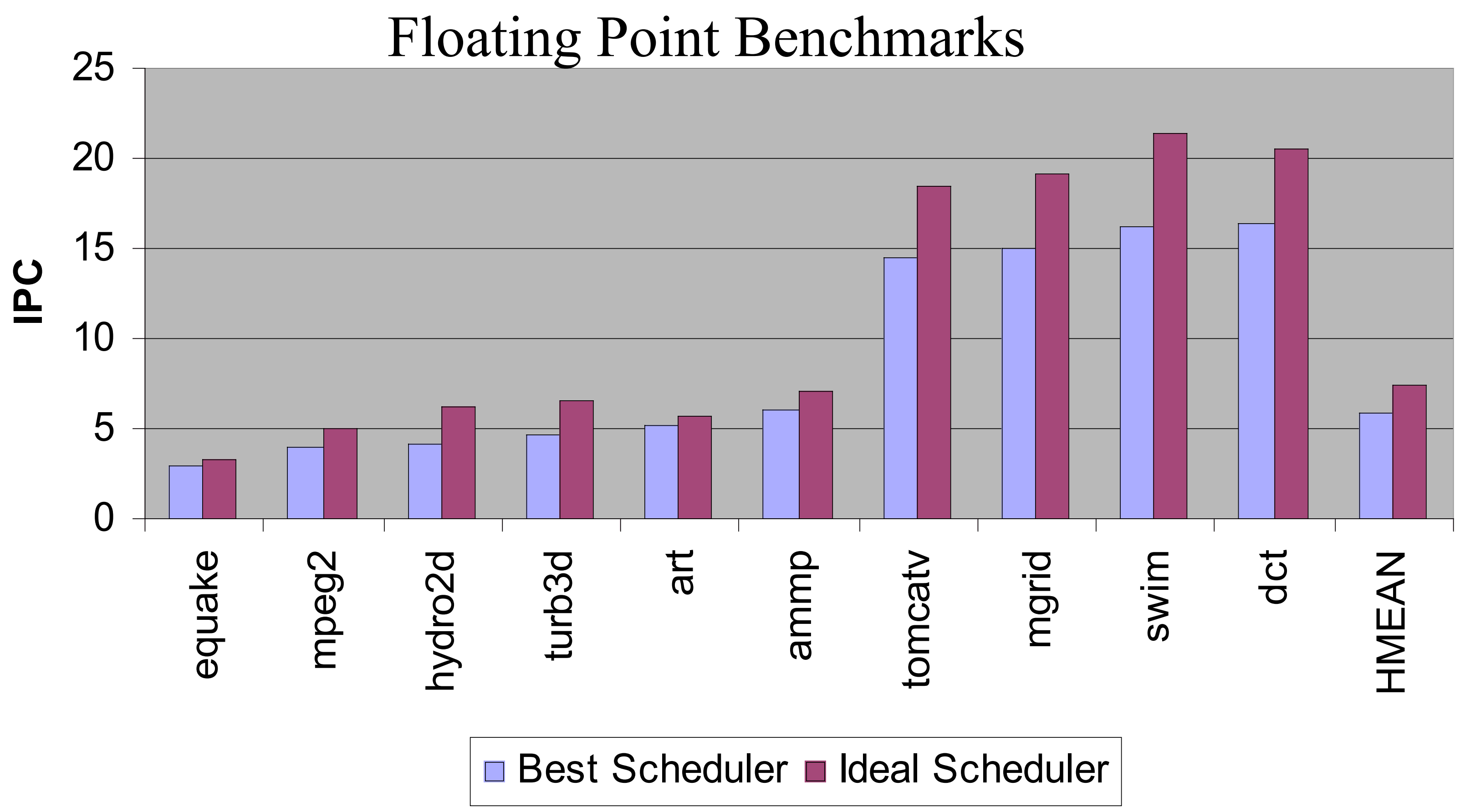


Comparison with Ideal Scheduler: 1 of 2



Ideal schedules do not have communication latencies on the critical path

Comparison with Ideal Scheduler: 2 of 2



Ideal schedules do not have communication latencies on the critical path

On-Going Work

- Improving the scheduler:
  - Profile guided scheduler optimizations
  - Code-specific heuristics
    - Select heuristics based on properties of the hyperblock
  - Minimize network contention
    - Analysis shows avoidable performance loss due to network contention
- Improving our evaluation with TRIPS-specific compiler:
  - Build larger hyperblocks
  - Aware of TRIPS-specific scheduling constraints

Conclusions

- Scheduling has two components that can be separated
  - Placement and issue

- EDGE architectures enable a new scheduling model
  - Static Placement, Dynamic Issue
  - Hardware dynamically tolerates unknown latencies
  - Compiler gives the hardware the ILP
  - Simpler static instruction scheduler

- Scheduler summary
  - Simple algorithm with well-chosen heuristics suffices
  - Load balancing heuristics are important
  - Register and cache locality heuristics are important
  - Performance within 20% of an optimistic upper bound