EDGE, TRIPS, and CLP
Bending architecture to fit workload

Zachary Weinberg

22 Jan 2009
The superscalar problem

- To have many instructions in flight at once, need huge on-chip control structures
  - issue queue, reorder buffer, rename registers, hazard detection, bypass network, result bus, ... 
- Wire delays limit these to 100 instructions or so
  - Alpha 21264: 80
  - Intel Core 2: 96
  - Intel Nehalem: 128 (64 with two threads)
- Heavy load on branch prediction
  - Have only a few gate delays to make a prediction
  - Need to issue a speculative branch nearly every cycle
  - Need near-perfect accuracy to avoid frequent pipeline flushes
TRIPS prototype block diagram.

TRIPS micronetworks.

- Limit communication between tiles
- Limit size of global control logic (G-tile only)
- Tremendous execution bandwidth (64 insns per E-tile)
- This is what we’d like to build, but how?
Data flow architectures avoid the superscalar problem

- Instructions activate when they have all their inputs
- There is no “program order” to maintain
- Each instruction says where its output goes
- Much control flow is replaced with predicated execution
  which means...

- Issue queue is simpler and can be distributed
- No reorder buffer necessary
- No need for rename regs or shared result bus
- Reduced load on branch prediction
But they have their own problems

- Loops must be “throttled” to avoid swamping the system with tokens
- Some implementations require exotic memory hardware (e.g. I-structures)
- Arguably superior but *unfamiliar* concurrency model (also true for exceptions, virtual memory, multitasking)
- Difficult or impossible to support code written in a conventional language
EDGE: a middle ground

- ISA designed for grid processors
- Lay out code in *hyperblocks*
  - one entry, many exits
  - instructions within form a data flow graph
  - static assignment to execution tiles
  - commit all instructions at once
- Conventional control flow between blocks
- Exceptions delayed to block boundary
- Can speculatively execute ahead of current block
Benefits

- Within a hyperblock, get benefits of data flow architecture
- Global structures only need to track inter-hyperblock state
  - can be done with smaller structures
  - gives more time to make decisions
  - allows far more instructions in flight overall
- Avoids problems of pure data flow architecture
  - Can execute conventional language code
  - More familiar concurrency and exception model
  - No looping within hyperblock, so no loop throttling needed
Problems

- Only one branch target per hyperblock
  - Not uncommon to need one per 5–10 instructions
  - Compiler must aggressively if-convert and unroll loops
  - Code size may increase significantly
- Intra-hyperblock scheduling is fragile
  - Goal is to put dependent instructions near each other
  - Optimal schedule depends on processor details
  - Like VLIW, may need recompilation for good performance
- Exception model is awkward for virtual memory
  - must repeat entire blocks for each page fault
  - worst case $O(n^2)$ penalty
First iteration: TRIPS

- Concrete design of a grid processor
- Simulated with simplifications (e.g. no page faults)
- 128-instruction hyperblocks
- Very simple execution tiles
- Three operational modes:
  - **D-morph**: From one thread, take one definite and many speculative blocks
  - **T-morph**: From several threads, take one definite and a few speculative blocks each
  - **S-morph**: Unroll a computational kernel across many blocks and run them all at once
- Values forwarded from producer to consumer blocks as available, through register file
Most like a regular single-thread OOO
Tested on a subset of SPEC (what compiler could handle)
Skip initialization, count only “useful” insns
Competitive with Alpha even with no speculation
Leaves Alpha in the dust with deeper speculation, esp. for floating point
Mispredictions hurt, especially integer code
Like any SMT, sacrifices per-thread resources for concurrency.

Biggest hit is from lower speculative depth, higher network contention.

Selected eight SPEC benchmarks and ran them in pairs, fours, or all at once.
- Two threads: 87% single thread throughput, 1.7x speedup
- Four threads: 61% single thread throughput, 2.4x speedup
- Eight threads: 39% single thread throughput, 2.9x speedup

No comparison to multitasking on D-morph.
S-morph

- Unroll a loop into many concurrent hyperblocks
- Can repeat without refetching
- Software control of L2 cache
- Benchmarked on 7 streaming kernels; hand-coded, machine-scheduled assembly
- Graph shows several different design points
- 2–4x D-morph performance
- Requires extra control logic
Second iteration: CLP/TFlex

- Dynamically aggregates cores as workload demands
- Distributes all control logic (abolish the G-tile)
- Cores are now dual-issue for integer ops
- Each core has its own L1 caches
- More operand network bandwidth
Dynamic aggregation of cores

- Threads can run on any number of cores
- Authors anticipate operating system (or even hardware!) will assign each task an appropriate number of cores
- Benchmarking done with core counts chosen by hand
- Will execution with varying core count mess up scheduling?
Distributing all the control logic

Each EDGE block has an owner core
- Chosen by hash of block address
- Directs the other cores through fetch, execution, commit

Instructions spread across cores as in TRIPS
- Fetch done by all cores in parallel
- Block commit by four-way handshake
- Branch history either kept with owner core (local) or transmitted along with branch predictions (global)
Fetch has significant fixed costs

As cores increase, both fetch and commit can use more parallelism accessing architectural state

But pay for it in more inter-core communication delay

4–8 cores seems to be a sweet spot

These are overhead per block

32 cores suffer less than 2% overhead per unit time
Yeah, they crammed too much stuff into this graph.

Take-homes seem to be:

- 19% faster than TRIPS with matched configuration
- 42% faster than TRIPS with configuration tuned for each application
- Makes more of a difference for benchmarks with more ILP
- Hand-optimized code much better than compiled code

This time around they have a cycle-accurate simulator
Unaddressed issues

- Code size
- Bandwidth to main memory
- Compiler quality
- Robustness of schedule
- Virtual memory
- Context switching
- Debugging