Baring it All to Software: Raw Machines

Agarwal Et. Al

Presented By: Daniel Gaultney
Raw Architecture Overview

- Small, replicated tiles
  - Simple RISC-like pipeline
  - Each has its own cache

- High-bandwidth Point-to-point network

- No register-renaming logic or dynamic issue logic

- “Configurable Logic”

- Relies heavily on static-routing provided by the compiler
Programmable Interconnect

- Each switch has its own cache (SMEM) which is the same size as the I-cache
- Single-cycle communication between tiles
- Suggested design devotes 30% of chip area to interconnects
- 2 networks (Static & Dynamic)
Diagram of the Raw Architecture

Figure 1. A Raw processor is constructed of multiple identical tiles. Each tile contains instruction memory (IMEM), data memories (DMEM), an arithmetic logic unit (ALU), registers, configurable logic (CL), and a programmable switch with its associated instruction memory (SMEM).
Configurable Logic

- FPGA-like logic that can be used to create customized functional units
- Allow for Multigranular operations
- Theoretically synthesized in the compiler
Compilation

- Far more control over program execution than a typical processor.
- Attempts to statically route as much as possible
- One-to-one mapping from threads to tiles
- Parallelizes code across multiple tiles
- Micro-architecturally dependent!
Dynamic Scheduling

- Dynamic Scheduling -> Static Scheduling + Slack
- “Plan to implement a system similar to Multiscalar”
- Possible All-to-all personal communication schedule in some approaches
- In general, to be avoided at all costs on Raw Machines
Simulations

- Implemented on an array of FPGAs
- Speedup obtained from:
  - Configurable Logic
  - Multigranularity
  - Parallelism
- Slowdown from Communication overhead
- Toy benchmarks highly optimized for this system: Conway’s Game of Life had over a 1,700X speedup!
Questions

- Is it even feasible to effectively compile code for this system that is more substantial than a toy benchmark?

- In light of the x86 dynasty, is it reasonable to expect us to compile code for each new micro architecture? Could there be a clever way around this?

- How could this system be modified to better handle dynamically scheduled code sections?

- How might the ratios of feature sizes be affected by the ~13 years of progress on CMOS technology that have been made since this paper was written? How would this change the behavior of the system?