An Overview of the Blackcomb Simulator

Gabriel Marin
in collaboration with Collin McCurdy
Blackcomb Overview

- Identify opportunities for byte-addressable NVRAM in Exascale architectures
  - Memory size/core projected to drop by $x_{100}$
    - DRAM power usage constraints
    - Impact on performance and application design
  - Investigate use of NVRAM beyond disk replacement
    - Integrate NVRAM into the node design such that it is byte-accessible by applications
    - Characterize key DOE applications and investigate how they are impacted by these new technologies
Understanding Application Impact

• Understand impact on performance and power of different design choices

• Considered existing modeling and simulation techniques/tools
  – Empirical modeling
    • Limited to existing systems/designs
  – Cross-architecture analytical modeling
    • Difficult to capture overlap between miss events
  – Simulators
    • Published but unavailable, cumbersome to use, cycle accurate useful but expensive
Blackcomb Simulator

- Use an additive performance model

\[ T(x) = T_0(x) + \text{DynMissPenalty} \]

\[ T(x) = T_0(x) + ICacheP(x) + DCacheP(x) + BrMissP(x) \]

- \( T_0(x) \) – instruction schedule cost w/o dynamic misses
  - Use modeling and static analysis to reduce simulation overhead

- Use a functional directed simulator to understand overlapped miss events
  - first order model to estimate computation overlap
Blackcomb Simulator Diagram

Application Object Code

Dynamic Analysis:
- incremental CFG
- edge counts

Machine Independent Application Profile

Functional-directed Simulator:
- branch misspredictions
- cache misses
- cache miss bursts

Architecture Description

Static Analysis:
- loop nests
- executed paths & freq
- instruction dependencies
- instruction schedule cost
- mechanistic model

Machine Specific Miss Events

Performance Prediction:
- baseline performance
- dynamic miss penalties
Capturing Dynamic Miss Events

- Use COTSon infrastructure
  - HP’s open source research simulator
  - Built on top of SimNow
  - Supports multicore systems and multi-threaded applications
  - Extensible using plugins
Blackcomb COTSOn Module

• Functional-directed simulation, no timings
  – Cache simulator
  – Branch predictor
  – Prefetch predictor (planned)

• Output profile at instruction level
  – Only branch and memory instructions
  – Include distribution of overlapped misses
Dynamic Analysis

• Light weight tool on top of PIN
  – Discover CFGs incrementally at run-time
  – Selectively insert counters on edges
  – Save CFGs and select edge counts
Static Analysis: Baseline Performance

- **Input:**
  - CFGs with select edge counts

- **Methodology:**
  - Recover execution counts for all blocks and edges
  - Compute loop nesting structures
  - Infer executed paths and their execution frequencies
  - Compute instruction schedule for executed paths

Static Analysis:
- loop nests
- executed paths & freq
- instruction dependencies
- instruction schedule cost
Computing Instruction Schedule

- Build dependence graph for path
- Native instructions $\rightarrow$ generic instruction types
- Machine description language $\rightarrow$ model architecture
- Instantiate scheduler with architecture description

- **Modulo instruction scheduler**
  - Critical path based, bidirectional scheduler
    - Similar to [Huff93]
    - Aggressive scheduler to model effect of OoOE
  - Compute SESE regions in dependence graph
    - Cuts # distinct recurrences to be tracked
    - Compute modulo schedule even for large outer loops
Estimating Dynamic Miss Penalty


- Stalls on two CPU resources
  - Instruction window empty
  - Reorder buffer (ROB) full
Branch Missprediction Penalty

penalty = window drain + front-end refill

- Original model:
  - window drains at dispatch width rate
  - window full when branch enters window
  - penalty = \( W_{\text{size}} / D + c_{\text{fe}} \) (front-end pipeline length)

- Extensions
  - window drains at rate given by IPC computed for loop
  - num instructions = min(\( W_{\text{size}} \), instructions since previous miss event)
Isolated Data Cache Miss Penalty

penalty = miss latency – useful dispatch

- **Original model:**
  - \( \text{useful dispatch} = \frac{W}{D} - c_{lr} \) (load resolution time)
    - useful dispatch << miss latency \(\rightarrow\) useful dispatch = 0
  - \( \text{penalty} = \text{miss latency} \)

- **Extensions**
  - \( \text{useful dispatch} = \frac{\text{instructions since previous miss}}{D} \)
    - \([\text{Chen & Aamodt 2008}]\)
  - window drains at rate given by IPC computed for loop
Penalty of Parallel Data Cache Misses

- Misses must occur within an interval of $W$ instructions
  - Misses must be independent
- Penalties overlap completely
- Model generalizes to any number of independent misses within $W$ instructions
- $penalty = penalty$ of an independent miss
Blackcomb COTS-on Module

• Functional-directed simulation
  – Cache simulator
  – Branch predictor
  – Simple dependence tracking
    • Understand serialized vs. parallel miss events

• Branch instructions
  – Number of misspredictions
  – Number of instructions since previous miss

• Memory operations
  – Number of cache misses
  – Number of instructions since previous miss
  – Include distribution of overlapped misses
Machine Description Language

- List of execution units (EU)
- Restrictions between EUs
- Instruction templates
- Instruction replacement rules
- Memory hierarchy characteristics
- Front-end pipeline length
- Window/ROB size
Cycle Accounting From Scheduler

Execution time

- dynamic miss penalty
- schedule time

- penalty for each miss type
- application dependence time
  - bottleneck on resources time
  - scheduling extra time
  - bottleneck per each resource
  - extra time due to each resource
Machine Description Language

• List of execution units (EU)

\[
\text{CpuUnits} = U_{Alu} \times 6, U_{Int} \times 2, U_{IShift}, U_{Mem} \times 4, \\
U_{PAlu} \times 6, U_{PSMU} \times 2, U_{PMult}, U_{PopCnt}, \\
U_{FMAC} \times 2, U_{FMisc} \times 2, U_{Br} \times 3, \\
I_M \times 4, I_I \times 2, I_F \times 2, I_B \times 3;
\]

• Restrictions between EUs

Maximum 6 from I_M, I_I, I_F, I_B {"at most 6 instructions issued per cycle"};
Machine Description Language

• Instruction templates

Instruction LoadFp template = I_M + U_Mem, NOTHING*5;
Instruction StoreFp template = U_Mem[2:3](1)+I_M[2:3](1);
Instruction LoadGp template = U_Mem[0:1](1)+I_M[0:1](1);
Instruction StoreGp template = U_Mem[2:3](1)+I_M[2:3](1);

• Replacement rules

Replace FpMult $fX, $fY -> $fZ + FpAdd $fZ, $fT -> $fD with
  FpMultAdd $fX, $fY, $fT -> $fD {"MultiplyAdd rule"};

Replace StoreFp $fX -> [$rY] + LoadGp [$rY] -> $rZ with
  GetF $fX -> $rZ {"GetF rule"};

Replace IntMult32 $rX, $rY -> $rZ with
  SetF $rX -> $f1 +
  SetF $rY -> $f2 +
  FpMultAdd $f1, $f2 -> $f3 +
  GetF $f3 -> $rZ;
Machine Description Language

• List of memory hierarchy levels (MHL)

/* For each level, parameters are: [numBlocks, blockSize, assoc, bdwth from a lower level bytes/cyc), level to go for miss at this level, penalty in cycles for going to next level ] */

MemoryHierarchy = L1D [256, 64, 4, 32, L2D, 4],
L2D [2048, 128, 8, 32, L3D, 8],
L3D [12288, 128, 6, 6, DRAM, 110],
DRAM [*, 16384, *, 0.04, DISK, 10000],
TLB [128, 8, *, 8, L2D, 25];
Summary

• Work still in progress
  – Use static analysis to compute instruction schedule cost
  – Use functional simulation to capture dynamic miss statistics

• Future work
  – Prefetch predictor
  – Shared caches
  – Remote memory access
  – Stalls on store buffers