ISA-Independent Workload Characterization and Implications for Specialized Architectures

Yakun Sophia Shao and David Brooks
Harvard University
{shao,dbrooks}@eecs.harvard.edu
Specialized architectures are decoupled from legacy ISAs.

Spectrum of Specialization:

- **General-Purpose CPU**
  - Low Efficiency
  - High Programmability
  - Tied to a Specific ISA

- **GPU**
  - Low Efficiency
  - High Programmability

- **Fixed-Function ASIC**
  - High Efficiency
  - Low Programmability
  - No ISA
Specialization requires workload intrinsic characteristics.

Specialized architecture is tailored to applications.
- e.g. special data path, memory access patterns.

I want to design specialized architectures for applications.

Where should I start first?

You need to first understand their characteristics.
Specialization requires workload intrinsic characteristics.

Yeah, good point! What should I do to understand those characteristics?

How about I run the program and collect performance-counter stats?

Hmmm... it's what you used to do for CPU designs.

but is what you get the true program characteristic?
Performance-Counter Based Workload Characterization

• **Metrics**
  – IPC
  – Cache miss rates
  – Branch mis-prediction rates
  – …

• **Microarchitecture-dependent**
  – What if there is a bigger cache/a better branch predictor?
  – Not program intrinsic characteristics
Specialization requires workload intrinsic characteristics.

Oh I also heard about microarchitecture-independent workload characterization.

We can perform the profiling analysis just using the instruction trace.

hmmm...that removes microarchitecture dependency. But it still ties to a specific ISA.
Specialization requires workload intrinsic characteristics.

“Ties to a specific ISA”? Will that be a problem?

Yes for specialized architectures!
ISA impacts program behaviors.

Stack Overhead
• Limited Registers
• Additional Load/Store

Complex Operations
• Memory Operands
• Vector Operations

Calling Conventions
Specialization requires workload intrinsic characteristics.

I see. So is there a way to get ISA-independent program characteristics?

That’s a good question. I found a paper in ISPASS this year which seems to answer this question. Let’s take a look!
Goal:
• An analysis tool to characterize workloads ISA-Independent characteristics for specialized architectures

Methods:
• Leverage compiler’s intermediate representation (IR)
• Categorize characteristics into compute, memory, and control

Takeaways:
• ISA-dependent characterization is misleading for specialization.
• ISA-independent characterization allows designers to quickly identify opportunities for specialization.
Tool Overview

Program

IR Trace

ISA-Independent

x86 Trace

ISA-Dependent

Characterization for Specialized Architecture

Design of Specialized Architecture

ISA-Dependent

Compute

Memory

Control
Program Representations

Program → ILD JIT → IR Trace → LLVM → x86 Trace
Program Representations

- SPEC CPU2000

Diagram:

Program
  ↓
 ILDJIT
  ↓
 IR Trace
  ↓
 LLVM
  ↓
 x86 Trace
Program Representations

ILDJIT
- A modular compilation framework
- Performs machine-independent classical optimizations at the IR level
- Uses LLVM’s back end to
  - Do machine-dependent optimizations
  - Generate machine code

Program Representations

ILDJIT IR

- High-level IR
- Machine-, ISA-, and system-library-independent
- Features:
  - 80 instructions
  - Unlimited registers
  - Only loads/stores access memory
  - No vector operations
  - Parameters are passed by variables
Program Representations

x86 Trace
- Used for ISA-dependent analysis
- Semantically equivalent to the IR code
- Collected with Pin instrumentation
Tool Overview

Program

IR Trace

x86 Trace

Design of Specialized Architecture

ISA-Independent

ISA-Dependent

Characterization for Specialized Architecture

Compute

Memory

Control
ISA-Independent Workload Characteristics

Compute
- Opcode Diversity
- **Static Instructions (I-MEM)**

Memory
- Memory Footprint (D-MEM)
- Global Address Entropy
- Local Address Entropy

Control
- Branch Instruction Counts
- Branch Entropy
Compute::Static Instructions

A bar chart showing the number of unique static instructions for various programs. The x-axis represents different benchmarks, including '179.art', '183.equake', '188.ammp', '164.gzip', '175.vpr', '175.vpr_2', '181.mcf', '186.crafty', '254.gap', '255.vortex', and '256.bzip2'. The y-axis represents the number of unique static instructions, ranging from 0 to 4500. The chart compares two categories: 'x86' (yellow bars) and 'IR' (blue bars). The '186.crafty' benchmark has the highest number of unique static instructions, significantly higher than the others.
Compute::Static Instructions

So if you use x86 trace instead of IR trace...

I will think those stack operations are part of the "hot code".
ISA-Independent Workload Characteristics

- **Compute**
  - Opcode Diversity
  - Static Instructions (I-MEM)

- **Memory**
  - Memory Footprint (D-MEM)
  - Global Address Entropy
  - Local Address Entropy

- **Control**
  - Branch Instruction Counts
  - Branch Entropy
Memory::Entropy

Entropy: a measure of the randomness

\[ Entropy = - \sum_{i=1}^{N} p(x_i) \log_2 p(x_i) \]

Case 1:
X is always a constant.

\[ p(X) = 1 \]
\[ \log_2 p(X) = 0 \]
\[ Entropy = 0 \]

Case 2:
N possible outcomes of X occur equally.

\[ p(X) = \frac{1}{N} \]
\[ \log_2 p(X) = \log_2 N^{-1} \]
\[ Entropy = -N \cdot \frac{1}{N} \cdot \log_2 N^{-1} \]
\[ Entropy = \log_2 N \]
Memory::Global Address Entropy

Temporal Locality

Address Stream A  
(less temporal locality)

\[
\begin{align*}
&0000 \\
&0001 \\
&0010 \\
&0011
\end{align*}
\]

Entropy = 2

Address Stream B  
(more temporal locality)

\[
\begin{align*}
&0011 \\
&0011 \\
&0011
\end{align*}
\]

Entropy = 0

Yen, Draper, and Hill. Notary: Hardware Techniques to Enhance Signatures. MICRO 08
Memory::Global Address Entropy

Temporal Locality

Address Stream A  (less temporal locality)  
0 0 0 0  
0 0 0 1  
0 0 1 0  
0 0 1 1  
Entropy = 2

Address Stream B  (more temporal locality)  
0 0 1 1  
0 0 1 1  
0 0 1 1  
0 0 1 1  
Entropy = 0

Yen, Draper, and Hill. Notary: Hardware Techniques to Enhance Signatures. MICRO 08
Memory::Global Address Entropy

Temporal Locality

So if you use x86 trace instead of IR trace...

I will have wrong locality estimate for workloads!
Memory::Local Address Entropy

Spatial Locality

Address Stream A
(less spatial locality)

0 0 0 0
0 1 0 0
1 0 0 0
1 1 0 0

Address Stream B
(more spatial locality)

0 0 0 0
0 0 0 1
0 0 1 0
0 0 1 1

# of Bits Skipped

Local Entropy

A
B

# of Bits Skipped

Memory Address Local Entropy

0 2 4 6 8 10
0 2 4 6 8 10
Memory::Local Address Entropy

Spatial Locality

So if you use x86 trace instead of IR trace...

I will think program has more spatial locality than it really has.
ISA-Independent Workload Characteristics

Compute
- Opcode Diversity
- Static Instructions (I-MEM)

Memory
- Memory Footprint (D-MEM)
- Global Address Entropy
- Local Address Entropy

Control
- Branch Instruction Counts
- **Branch Entropy**

_Yokota, et al., Introducing Entropies for Representing Program Behavior and Branch Predictor Performance, 07_
Control::Branch Entropy

![Bar chart showing branch entropy for various benchmarks. The chart compares two sets: x86 and IR. The benchmarks include: 179.art, 183.equake, 188.ammp, 164.gzip, 175.vpr, 175.vpr_2, 181.mcf, 186.crafty, 254.gap, 255.vortex, 256.bzip2.](chart.png)
Control::Branch Entropy

So if you use x86 trace instead of IR trace...

I won’t get much wrong for control.
ISA-Independent Workload Characteristics

Compute
- Opcode Diversity
- Static Instructions (I-MEM)

Memory
- Memory Footprint (D-MEM)
- Global Address Entropy
- Local Address Entropy

Control
- Branch Instruction Counts
- Branch Entropy

Is there a way to compare those across workloads?

Yes, Kiviat plot!
ISA-Independent Workload Characteristics

**Compute**
- Opcode Diversity
- Static Instructions (I-MEM)

**Memory**
- Memory Footprint (D-MEM)
- Global Address Entropy
- Local Address Entropy

**Control**
- Branch Instruction Counts
- Branch Entropy
Workload Characterization
Conclusions

• We demonstrate that ISA-dependent analysis can be misleading for specialized architectures.

• We present an analysis tool to characterize ISA-independent characteristics for specialization.

• We show that our tool provides opportunities for designers to compare workloads’ characteristics.