

# Challenges in Capturing Real World Workloads into Benchmarks

---

David J. Lilja

Electrical and Computer Engineering

University of Minnesota

# Why is this Hard?

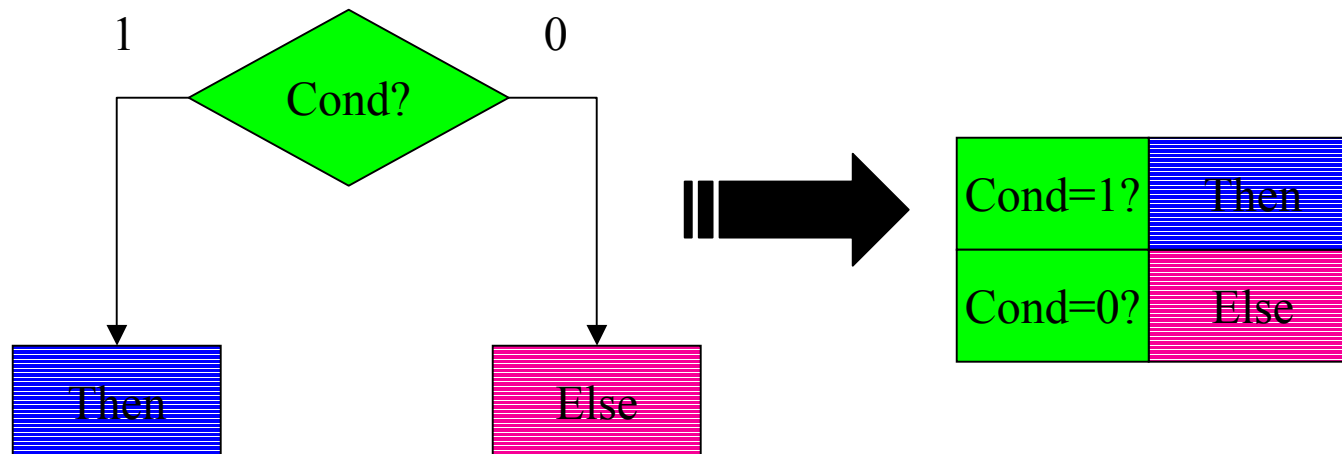
---

- **Benchmarks are not predictive**
- Who's **world** defines the “**real-world**” anyway?
  - Desktop applications?
  - Transaction processing?
  - Multimedia?
  - Portable applications?
  - **What will be important tomorrow?**



# Example: Predicated Execution

- Microarchitecture technique to reduce impact of branch execution
- Speculatively execute both branch paths
- Throw out results from wrong path



# Example: Predicated Execution

	<b>Research</b>	
<b>Speedup</b>	+30-63%	
<b># mispredictions</b>	-56%	
<b># branches</b>	-27%	
<b>Source</b>	Mahlke, '94, '95; August, '98	

# Example: Predicated Execution

	<b>Research</b>	<b>Intel Itanium</b>
<b>Speedup</b>	+30-63%	
<b># mispredictions</b>	-56%	
<b># branches</b>	-27%	
<b>Source</b>	Mahlke, '94, '95; August, '98	

# Example: Predicated Execution

	<b>Research</b>	<b>Intel Itanium</b>
<b>Speedup</b>	+30-63%	+2%
<b># mispredictions</b>	-56%	-24%
<b># branches</b>	-27%	-7%
<b>Source</b>	Mahlke, '94, '95; August, '98	Choi, MICRO, 2001

# Crash Reconstruction

---

- What went wrong here?
- The design is excellent
  - Does precisely what predication is supposed to do
- So what happened? ...



# A Benchmark Failure?

	<b>Research</b>	
<b>Benchmark</b>	SPEC92, 95	
<b>Key characteristic</b>	<b>Branch behavior</b>	
<b>HW assumptions</b>	8-issue to symmetric ALUs	
<b>Input set</b>	Reduced	
<b>OS effects?</b>	No	

# A Benchmark Failure?

	<b>Research</b>	<b>Intel Itanium</b>
<b>Benchmark</b>	SPEC92, 95	SPEC2000
<b>Key characteristic</b>	Branch behavior	Memory delays
<b>HW assumptions</b>	8-issue to symmetric ALUs	6-issue to 4 ALU, 2 ld/st; 3 branch
<b>Input set</b>	Reduced	Full
<b>OS effects?</b>	No	Yes

# Lesson 1

---

- Shouldn't use **yesterday's benchmarks** to design for **tomorrow's applications**.
  - Doesn't accurately **predict** what **will be** needed
  - Emphasizes wrong program characteristics



# Lesson 2

---

□ Can't ignore **operating system effects**.

- The OS is a big application!

- A lot of time is spent executing the OS

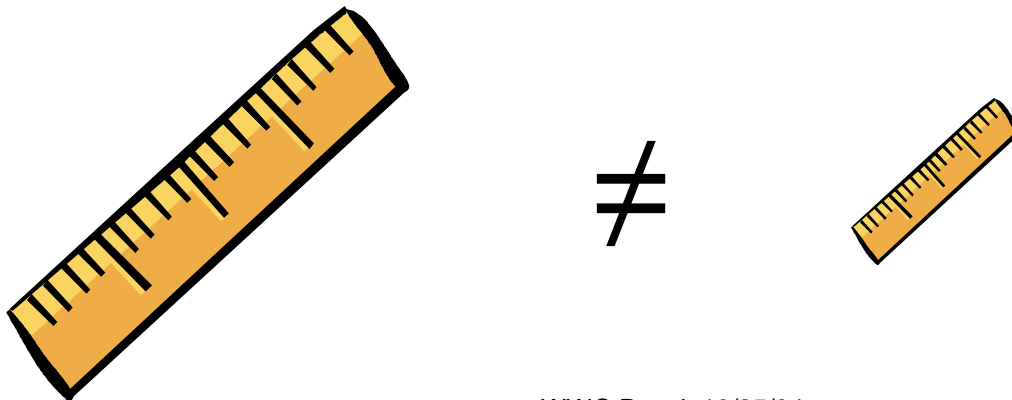
- The (unwritten) **Law of Operating Systems**

*“As system performance improves, the OS will grow to use all of the available processing power and memory space, plus 10%.”*

# Lesson 3

---

- Arbitrarily **scaled input sets** can be dangerous
  - Distorts memory behavior
  - Focuses attention on limited aspects of program behavior
  - Leads to *designing to the (wrong?) benchmark*





# Lesson 4

---

- Can the **software** keep up?
  - Will the compiler really exploit those nice new features?
  - Will the OS understand the new features?
  - What about binary compatibility?
    - Recompiling the world is not an option



# Lesson Summary

---

1. Shouldn't use **yesterday's benchmarks** to design for **tomorrow's applications**
2. Can't ignore the **OS**
3. Must be very careful about **scaling** input sets
4. The **software may not exist** to exploit the new features.



# Conclusion

---

- Real-world application benchmarks
  - Good for comparing **today's** machines
- Micro-benchmarks
  - Good for **component stress testing**
- Research question
  - *How to make good **predictive** benchmarks?*

# A Parting Thought

---

“Experimental **confirmation of a prediction** is merely a measurement. An experiment **disproving a prediction is a discovery.**”

*Enrico Fermi*

- In computer design, we want
  - **More confirmations**
  - **Fewer discoveries!**
- No performance surprises

