# Performance Prediction Models

**Shruti Padmanabha**, Andrew Lukefahr,
Reetuparna Das, Scott Mahlke

*{shrupad,lukefahr,reetudas,mahlke}@umich.edu*

**Gem5 workshop**

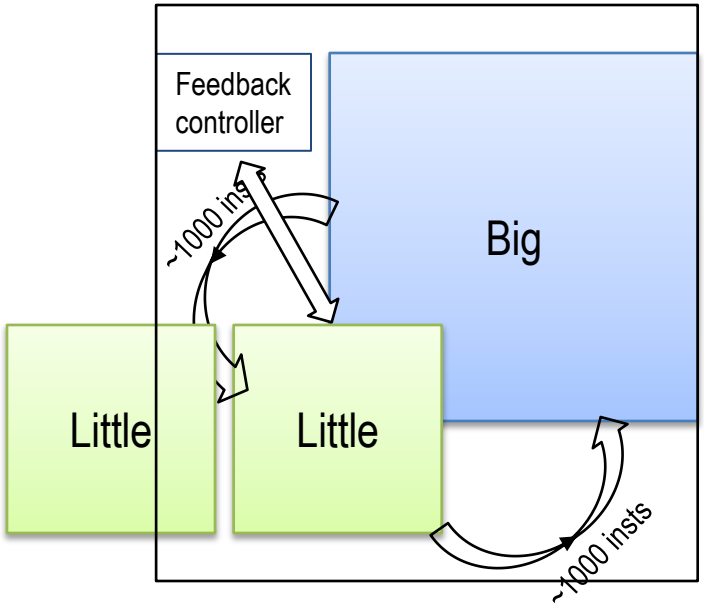
Micro 2012

December 2, 2012



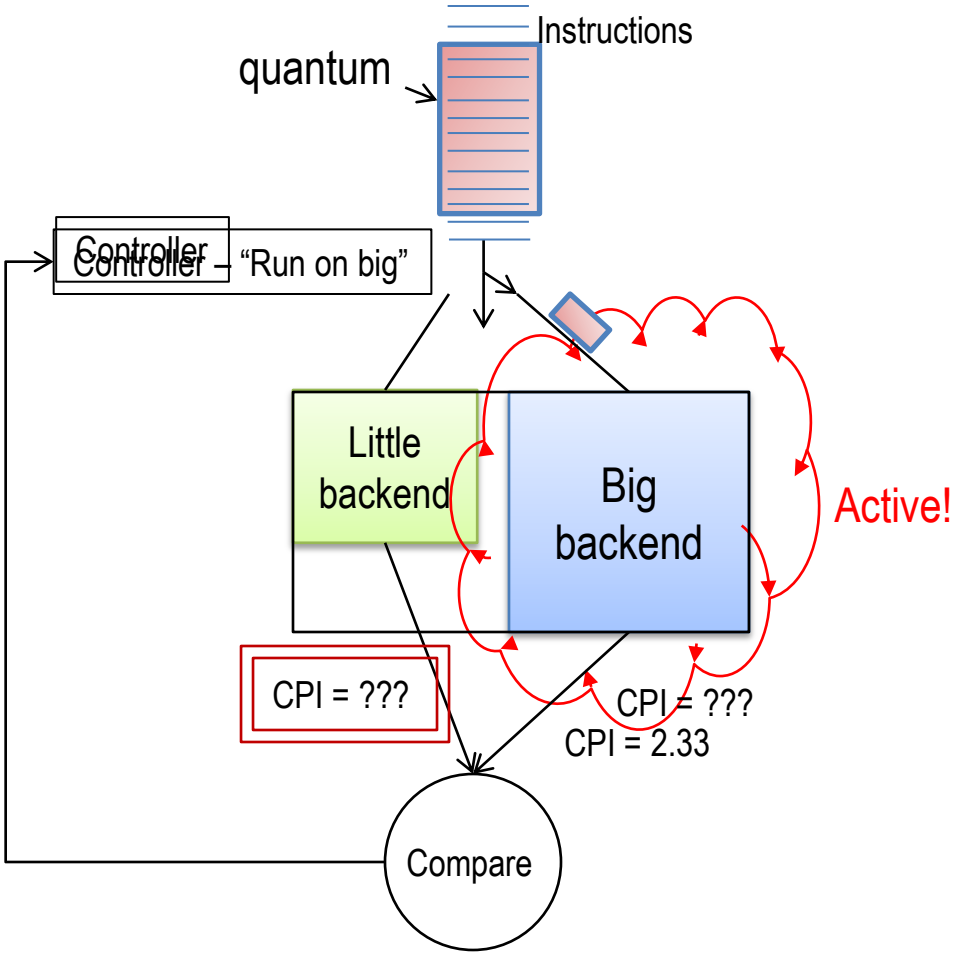
# Composite Cores

- Pushing heterogeneity into a core
- A tightly coupled o3 backend (big) and an inorder one (little)
  - Big – 3 wide OoO with large ROB, LSQ
  - Little – 2 wide inorder, modeled as a OoO core with simplified pipeline, small ROB, no LSQ
- Switch at fine granularity or **quantum** (controller)




For more details, please attend the paper presentation on Tuesday

# Operation of Composite Cores



# Objectives

- Run a quantum on one backend microarchitecture and project its performance on a different one dynamically
- Challenge: Only one is active at any given time
- Solution: Use a linear model to calculate the inactive core's performance using the slice's computational traits

$$y = a_0 + \sum a_i x_i$$

# Performance defining factors

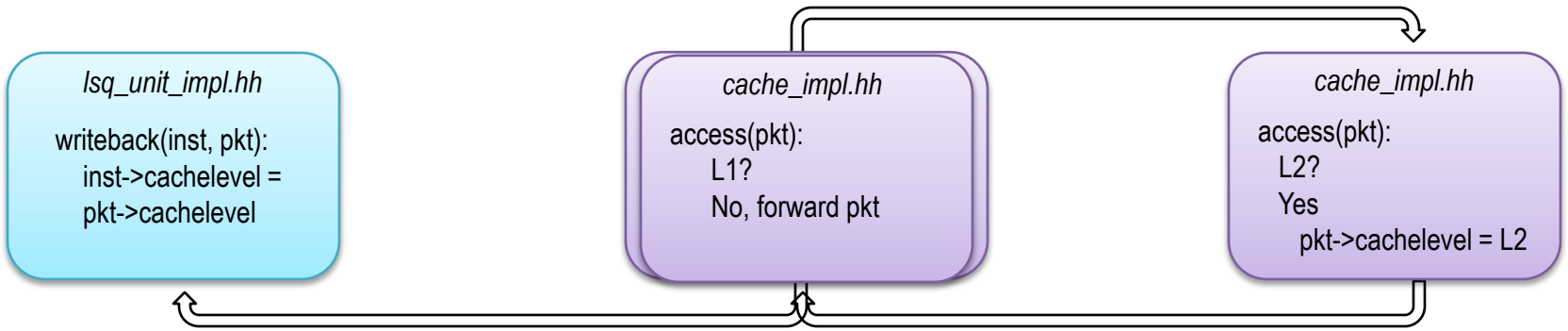
| Computational trait | Big | Little | Rel Performance Diff |
|---|---|---|---|
| Independent chain of instructions (high ILP) | Exploits larger superscalar width | Lower throughput | High |
| Dependent chain of instructions (low ILP) | Issues in order | Issues in order | Low |
| Branch mispredictions | Large drain time | Smaller drain time | High |
| Independent chain of L2 misses (high MLP) | Can have multiple outstanding loads | Stalls | High |
| Dependent chain on L2 misses (low MLP) | Stalls | Stalls | Low |
| Icache misses | Stalls | Stalls | Low |

# Performance defining Counters

- Per program slice, dynamically track
  - Active CPI
  - # of Branch misses
  - # of L1 misses
  - # of L2 misses
  - # of Icache misses
  - ILP
  - MLP
- Append dynamic instruction class with fields that identify above parameters

# Performance counters example

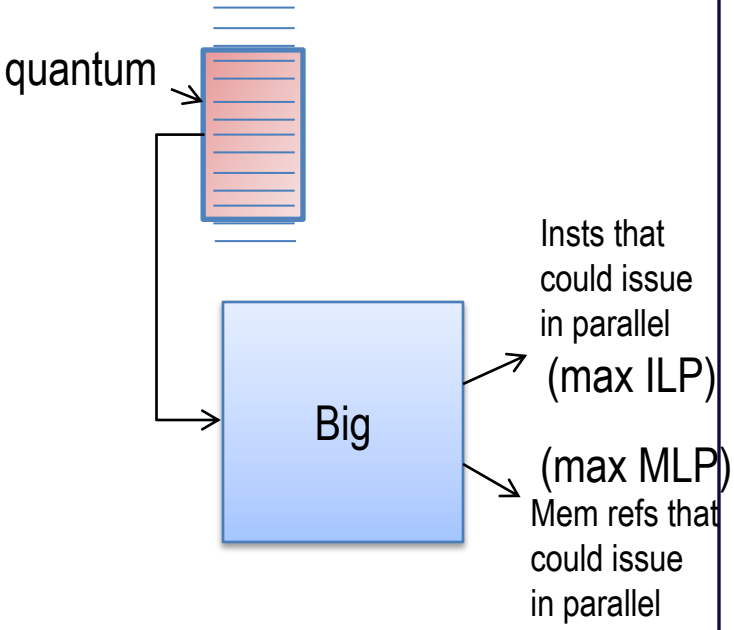
- Branch misses:
  - Set flag on branch mispredict discovery in iew_impl.hh
- Cache level:
  - Append the Packet class with a field to track the level of cache that satisfied request
  - Set field on packet return, in lsq_unit_impl.hh



# Measuring ILP & MLP – Big backend

While on big,

- Measure of ILP in a quantum:
  - Track # of instructions that are stalled due to dependencies in inst_queue_impl.hh



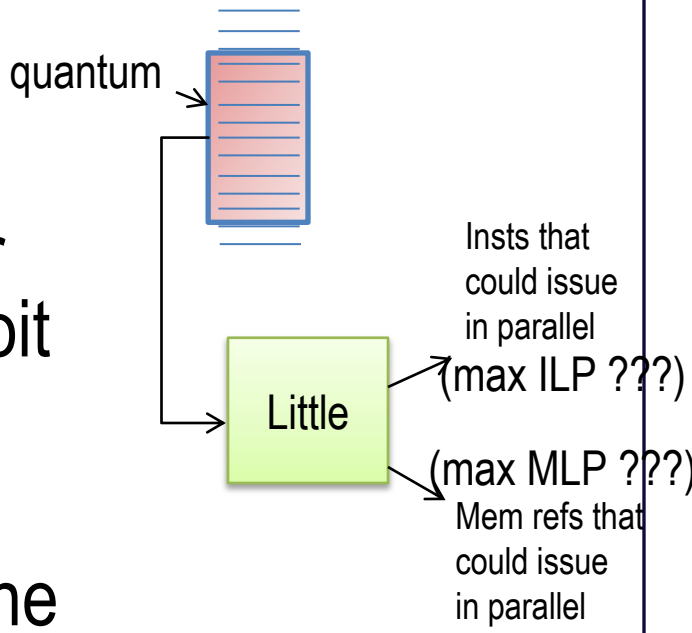

- Measure of MLP in a quantum:
  - Track # of MSHR entries in use at each L1 cache miss

# Measuring ILP & MLP – Little backend

While on little,

- More complicated, since the smaller core doesn't have the ability to exploit these characteristics

- Track data dependence chains for the window of instructions that big cares about (it's ROB length)

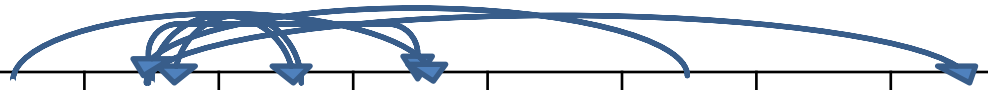


# Little's Dependence Tracker Table*

- Bit matrix – (#ROB entries in big) * (#registers)
- Implemented using multidimensional vectors in commit_impl.hh
- Passes information about the serialization inherent in code section to the instruction class

**L. Chen, S. Dropsho, and D. Albonesi, "Dynamic data dependence tracking and its application to branch prediction"*

# Dependence Tracker Table



| Inst # \ Reg # | r1 | r2 | r3 | r4 | r5 | r6 | r7 | r8 | Instruction |
|---|---|---|---|---|---|---|---|---|---|
| Inst 1 | | X | X | X | | | | X | r2 <- r3 + r5 |
| Inst 2 | | | | X | | | | | r4 <- r1 – r2 |
| Inst 3 | X Miss | | | | | | | | r1 <- Load(r7) |
| Inst 4 | | X Miss | X Miss | | | | | X Miss | r3 <- Load(r2) |
| Inst 5 | | X | | | | | | X | r2 <- r3 or r6 |
| Inst 6 | | | | | | | | X Miss | r8 <- Load(r2) |

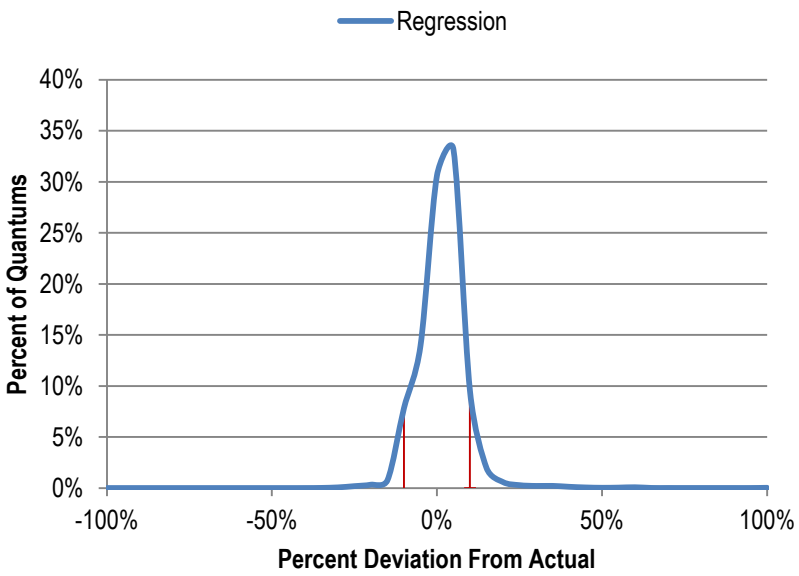
Big ROB Head
Big ROB Tail
Big ROB Tail
Big ROB Tail
Independent load!
Big ROB Tail
Big ROB Tail
Dependent load!

8 register system, with 7 big ROB entries

2 independent, overlappable and 1 dependent L2 miss seen in this window.
Max available MLP = 2

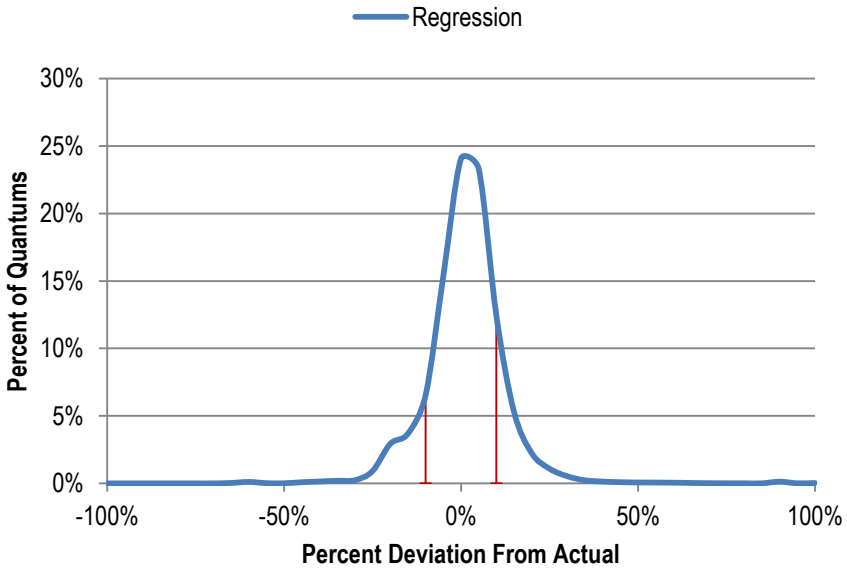
# Prediction Accuracy

big->little




95% of the quantums predicted with <10% error

little->big




82% of the quantums predicted with <10% error

# Conclusion

- Implemented a performance prediction model by measuring characteristics like MLP and ILP in a code section with prediction accuracy of 88.5% on average
- Gem5 modifications:
  - Created a new controller class to carry out prediction calculations dynamically. This was invoked on every instruction commit in commit_impl.hh
  - New fields added to the base_dyn_inst class
  - Tracker table implemented in commit_impl.hh in commitInsts()
  - Counters were also added in inst_queue_impl, iew_impl, cache_impl, lsq_unit_impl
- Total lines of code added: ~200

# Questions?