Monte Carlo Processor Modeling of Contemporary Computer Architectures

Jeanine Cook

Students: Waleed Alkohlani, Ram Srinivasan

New Mexico State University

Problem

- Need tools to do performance analysis of contemporary architectures (design, prediction, procurement)
- Cycle-accurate simulation
  - Great for accuracy, hard on time!
  - Lack of freely available simulators that simulate contemporary architectures
- Analytic models
  - Hard to use
  - Not very accurate or robust

Solution

- Statistical model
  - Based on processor and application characteristics
  - Generates fast, accurate predictions
  - Can do more than just predict execution time
  - Robust

Monte Carlo Processor Modeling

- Processor pipeline abstracted into statistical model using
  - dynamic application profiles
  - processor microarchitecture characteristics

- Based on $CPI = CPI_I + CPI_S$
  - CPI_I ==> Intrinsic CPI based on issue width
  - CPI_S ==> CPI due to stalls

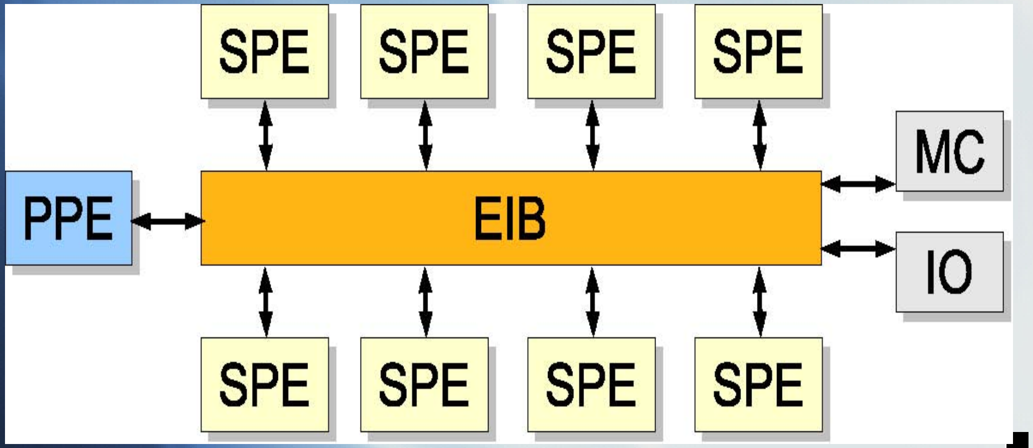
Current Capabilities

- Single and multi-core
- In-order instruction execution
- Flexible cache model
- Captures instruction sequence relationships
- Niagara 1 and 2, Cell, Itanium

Future Capabilities

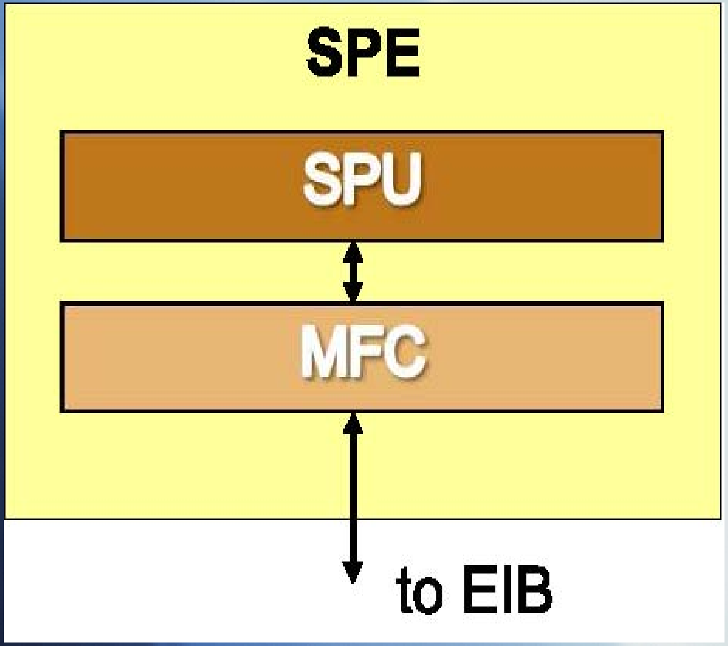
- Improved flexible cache model
- Implement out-of-order model methodology
- Develop method for modeling multi-threaded processors
- Implement power models for consumption prediction
- Integrate into communication model ==> MP model
- Modeling framework

Cell Model



- PPE (PowerPC) - 2-issue, in-order, 2-way SMT
- SPEs - 2-issue, in-order, SIMD
- EIB - 96 bytes/cycle

Synergistic Processing Elements (SPEs)



- SPU - statically scheduled, 128x128-bit regs, 256KB local store (LS)
- MFC - handles communication; DMA requests (from PPE and SPUs), mailboxes, signals

SPU EUs

Partition	**EU (and latency in cycles)**
Even	FP6(6), FP7(7), FPD(13), FX2(2), FX3(4), FXB(4), NOP(0)
Odd	LSU(6), BR(4), SHUF(4), SPR(4), LNOP(0)

- FPD not fully pipelined; when insn issued, stalls global insn issue for 6 cycles

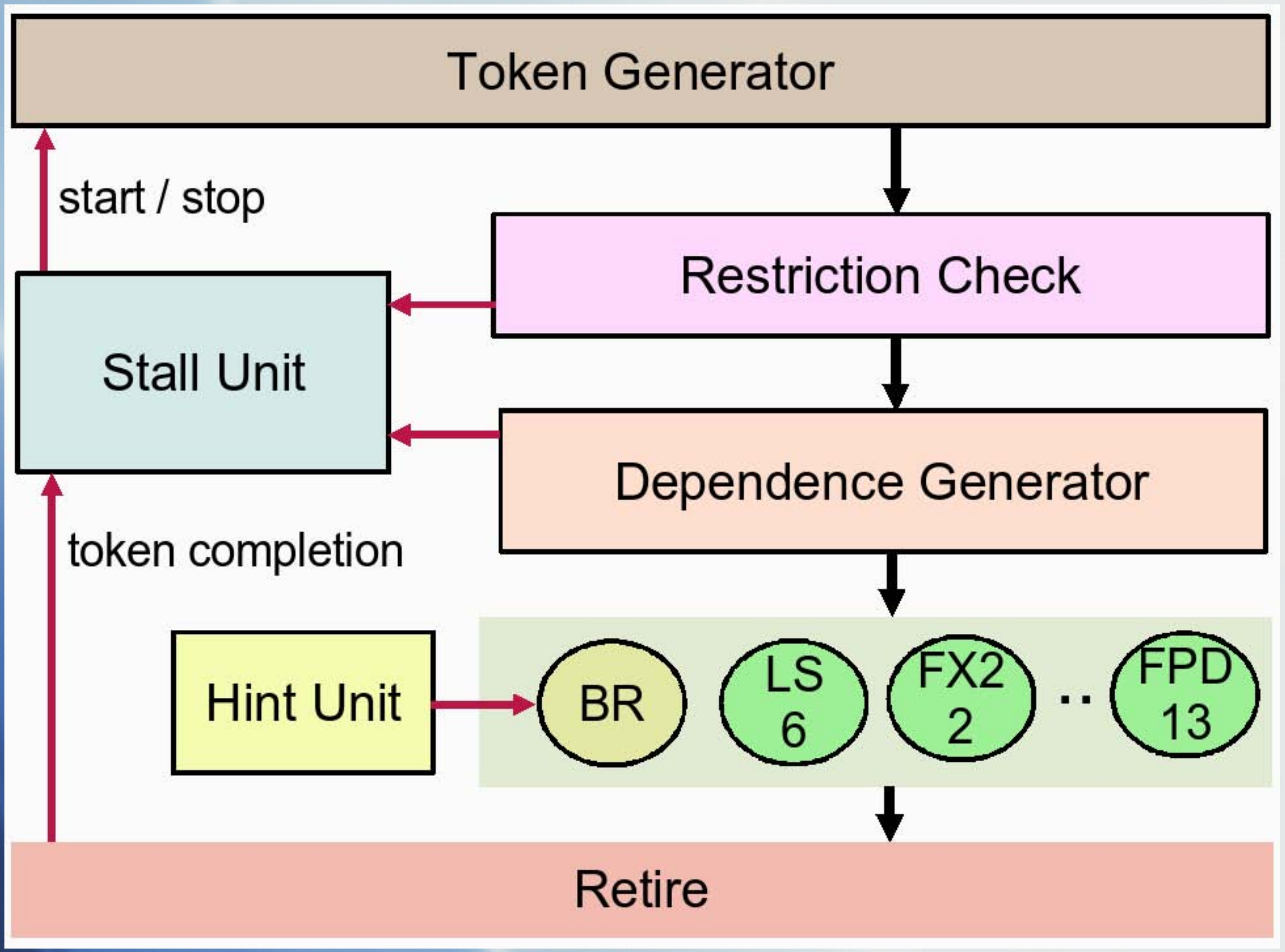
12 Steps (1)

1. **Issue mechanism**: SPU stall-at-use
2. **CPIi**: should be 1/2, but due to even/odd restrictions, we measured from dynamic insn stream
3. **Stall reasons**: unresolved dependences, mis-speculated branches
4. **EU characteristics**: on prior slide; not shared
5. **Cache characteristics**: no cache hierarchy
6. **Memory characteristics**: only modeled SPUs; no access directly to memory; access latency LS 6 cycles
7. **Branch predictor characteristics**: 18 cycle fixed penalty branch mis-predict

12 Steps (2)

8. **Variable latency**: branch unit depends on quality of branch hints
9. **Application characteristics**: CPI_i, dynamic insn mix (generate transition probs), dependence distance histograms, hint-to-branch histogram, prob of taken and hinted branches
10. **Collect application profile**: designed instrumentation tool for Cell
11. **Model**
12. **Validation**

Cell SPU Model

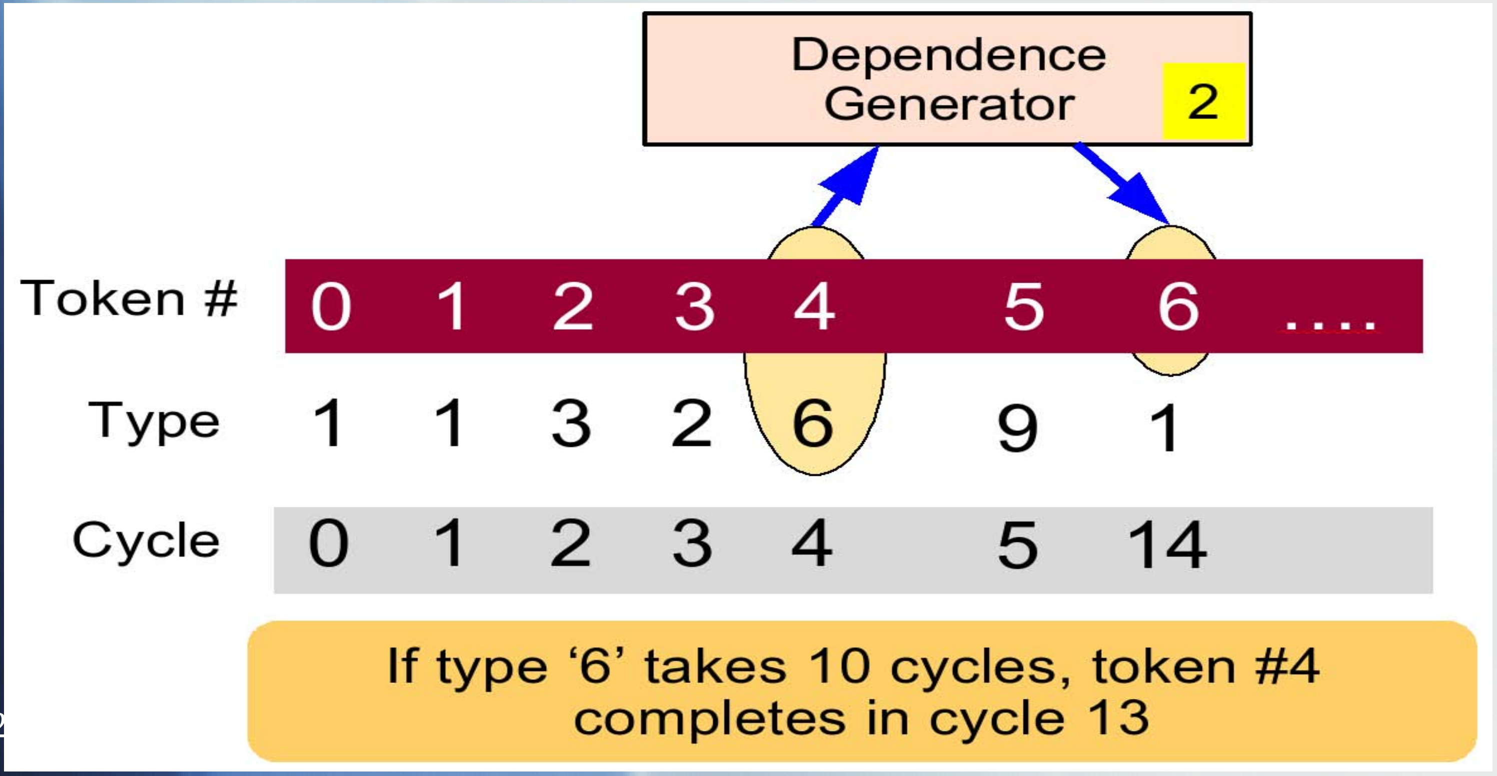


Token Generation

- Instruction mix translated to probability
- Tokens encoded as integers for each insn class
- Markov token generator

Dependence Generator and Stall Unit

- Based on application dependence histograms (e.g., FP-use, LD-use)

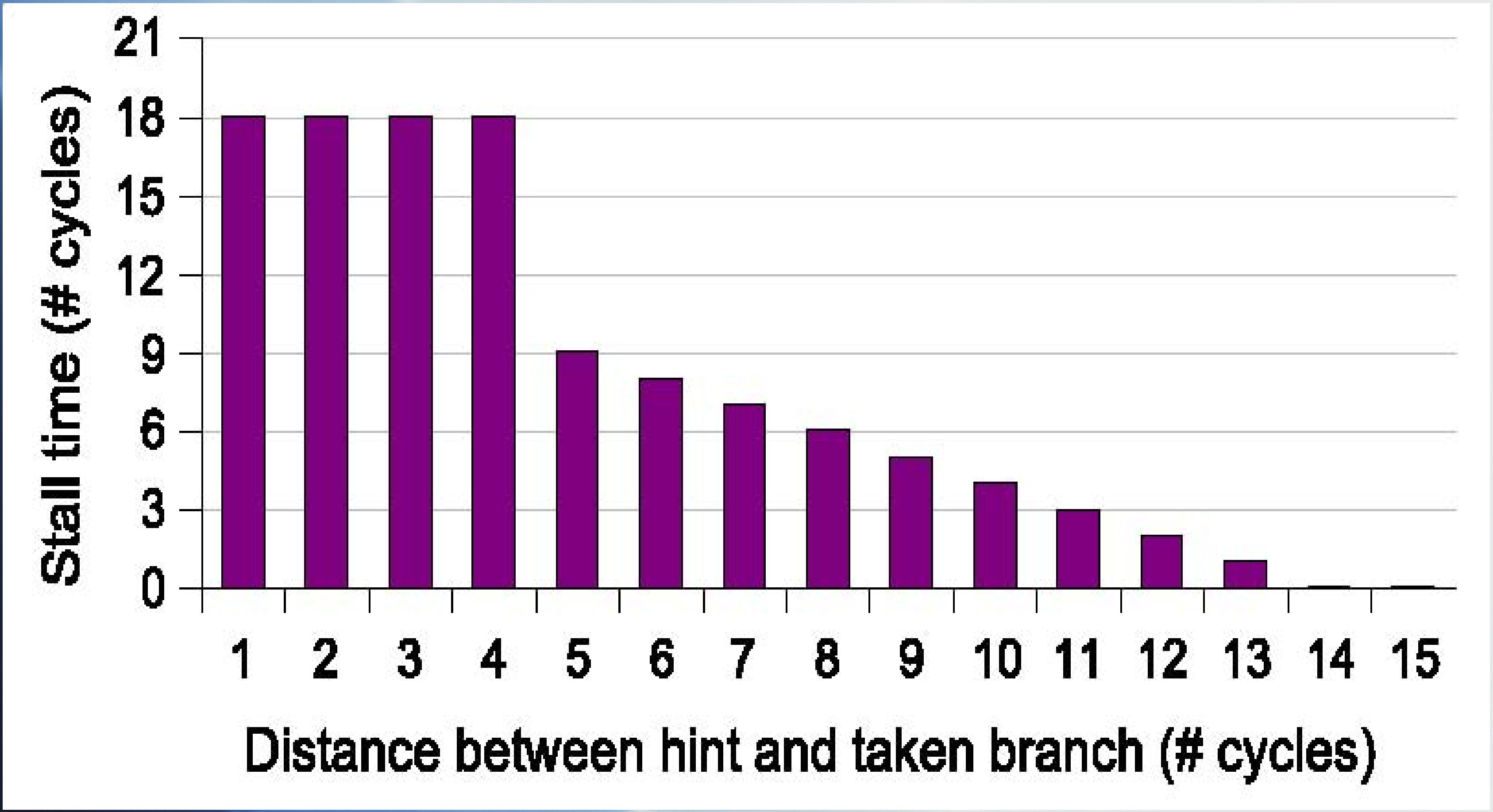


Branch Hints

- SPUs statically predict branches not taken (0 cycle penalty)
- 18 cycle penalty for taken (mis-predicted) branches
- Hinting mechanism to reduce penalty
  - Hints take effect after 4th pipe stage
  - Take 9 more cycles to fetch target
  - If branch appears within 4 cycles of hint, hint does nothing

NM STATE

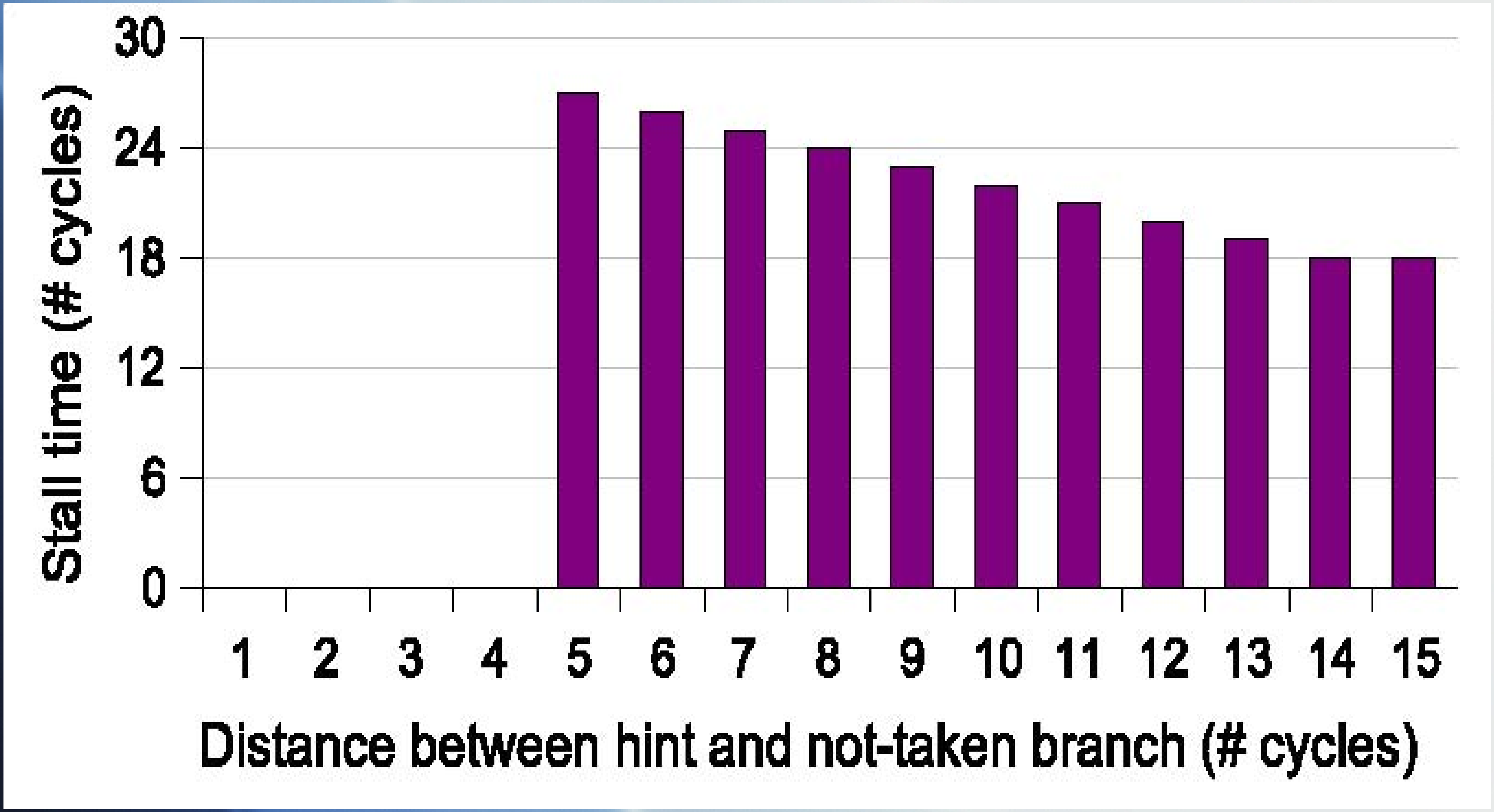
Hint-to-Branch: Taken



Problem with Branch Hints

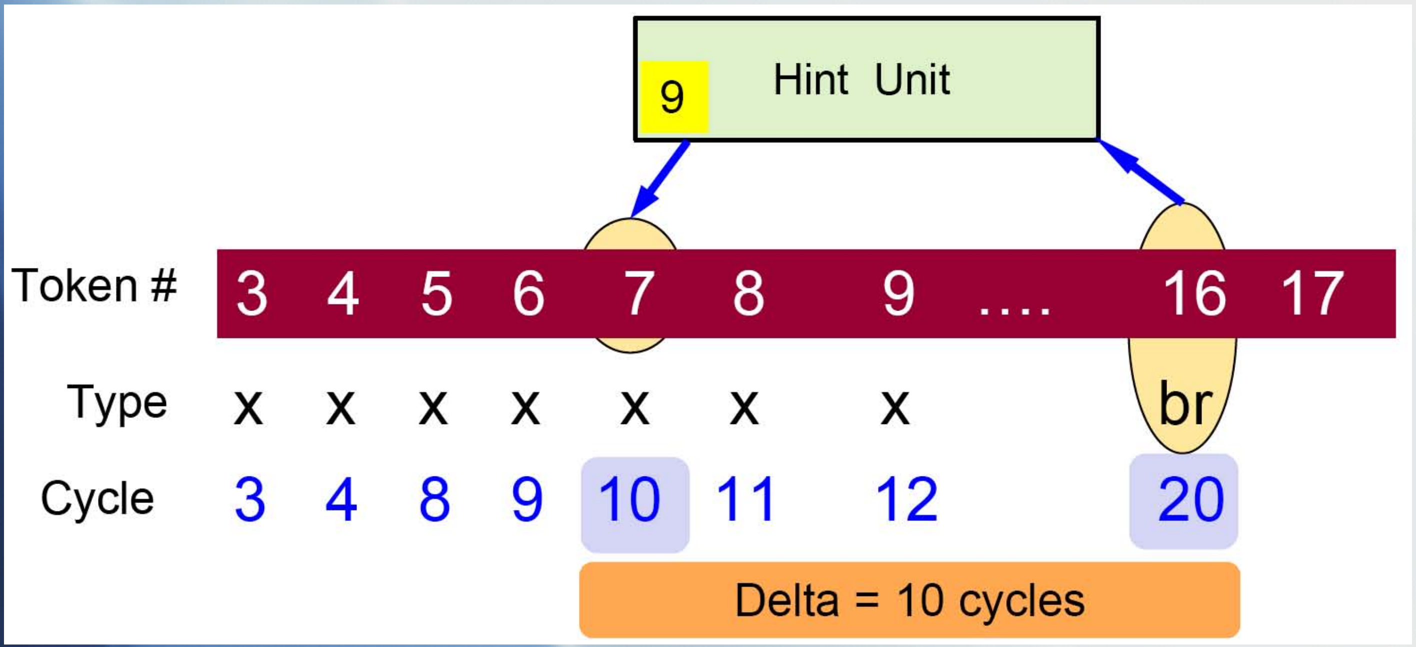
- Hinted, not-taken branches can stall up to 27 cycles!
  - Hinting not-taken branch - hint is probably wrong

Hint-to-Branch: Not Taken



Hint Unit

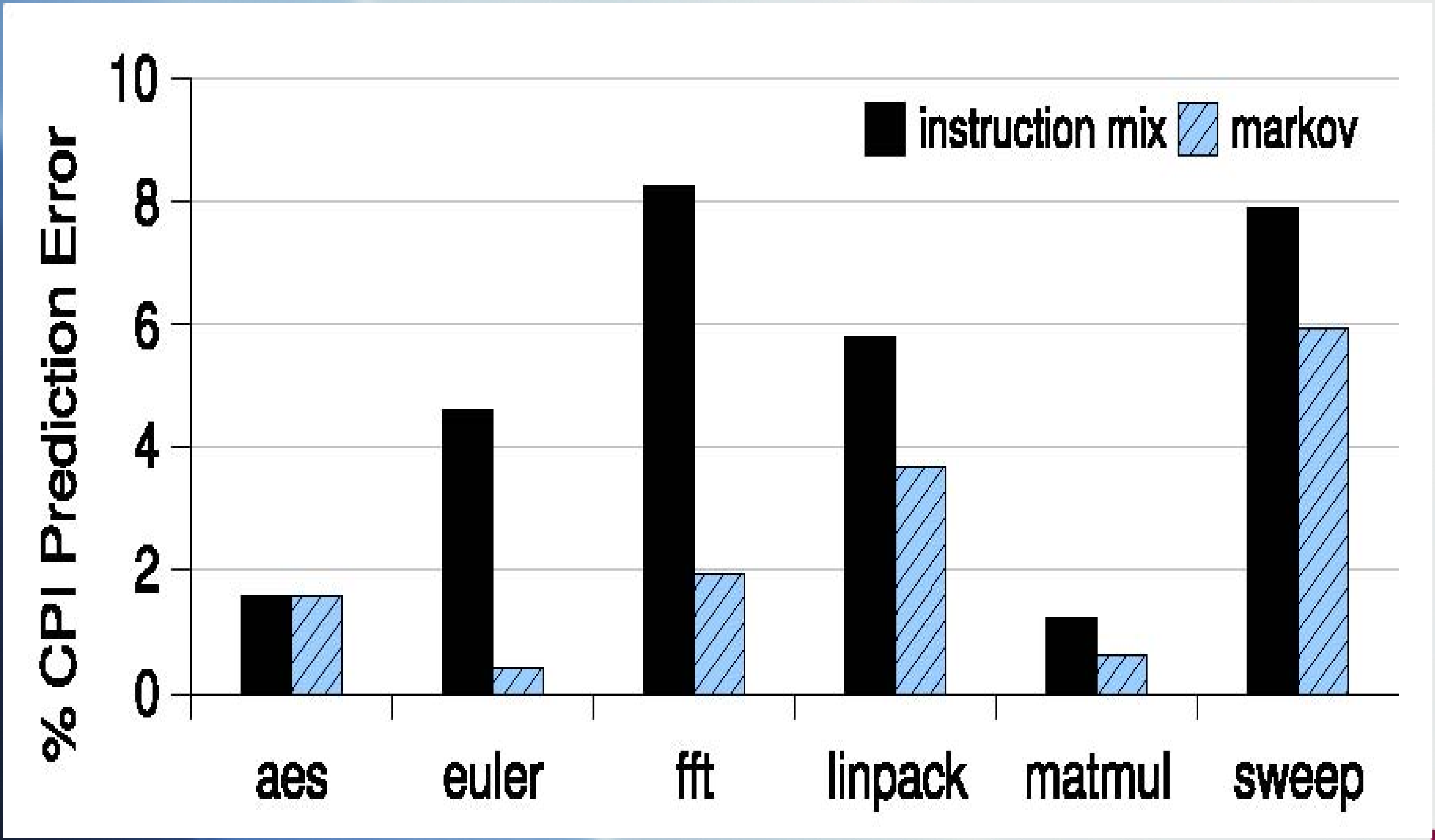
- Service time of BR unit based on hint-to-branch distance histogram
- Statistically determine from application
  - probability that branch taken/not taken; branch is hinted



Model Parameters

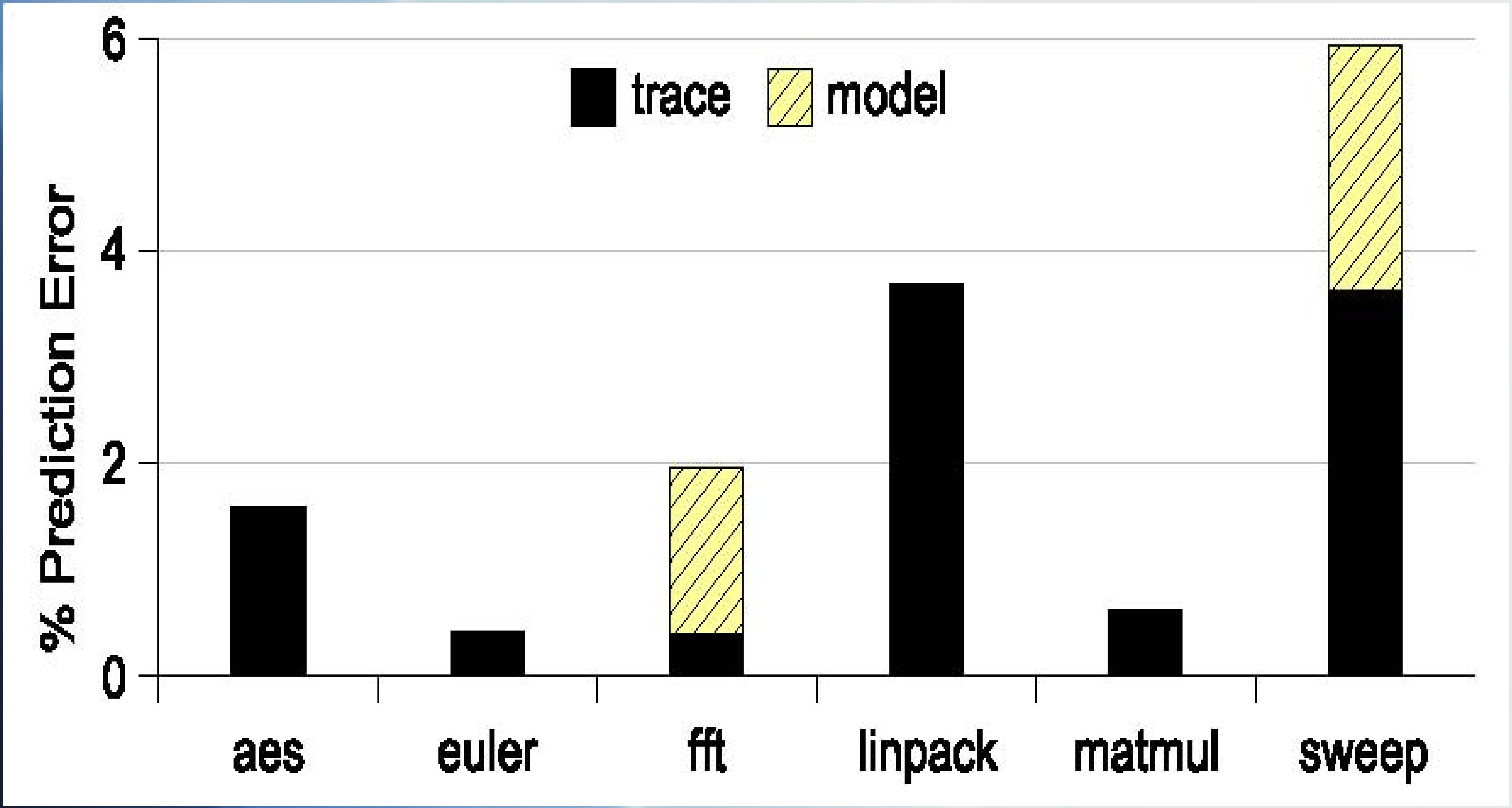
- CPI_i
- Instruction transition probabilities
- Dependence distance histograms
- Hint-to-branch distance histograms
- Probability of taken branch
- Probability of hinted branch

Results

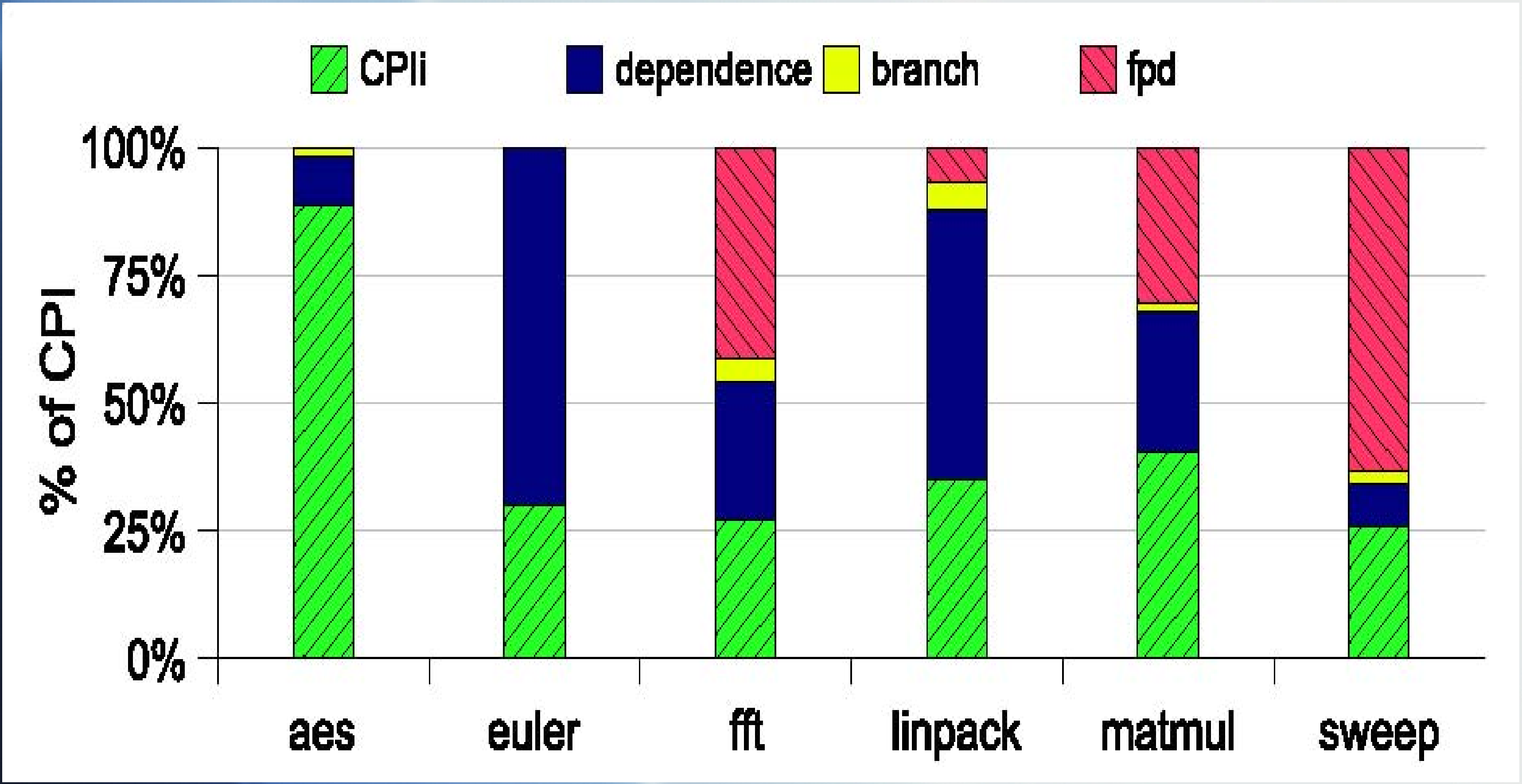


% CPI Prediction Error
10
8
6
4
2
0
instruction mix
markov
aes
euler
fft
linpack
matmul
sweep

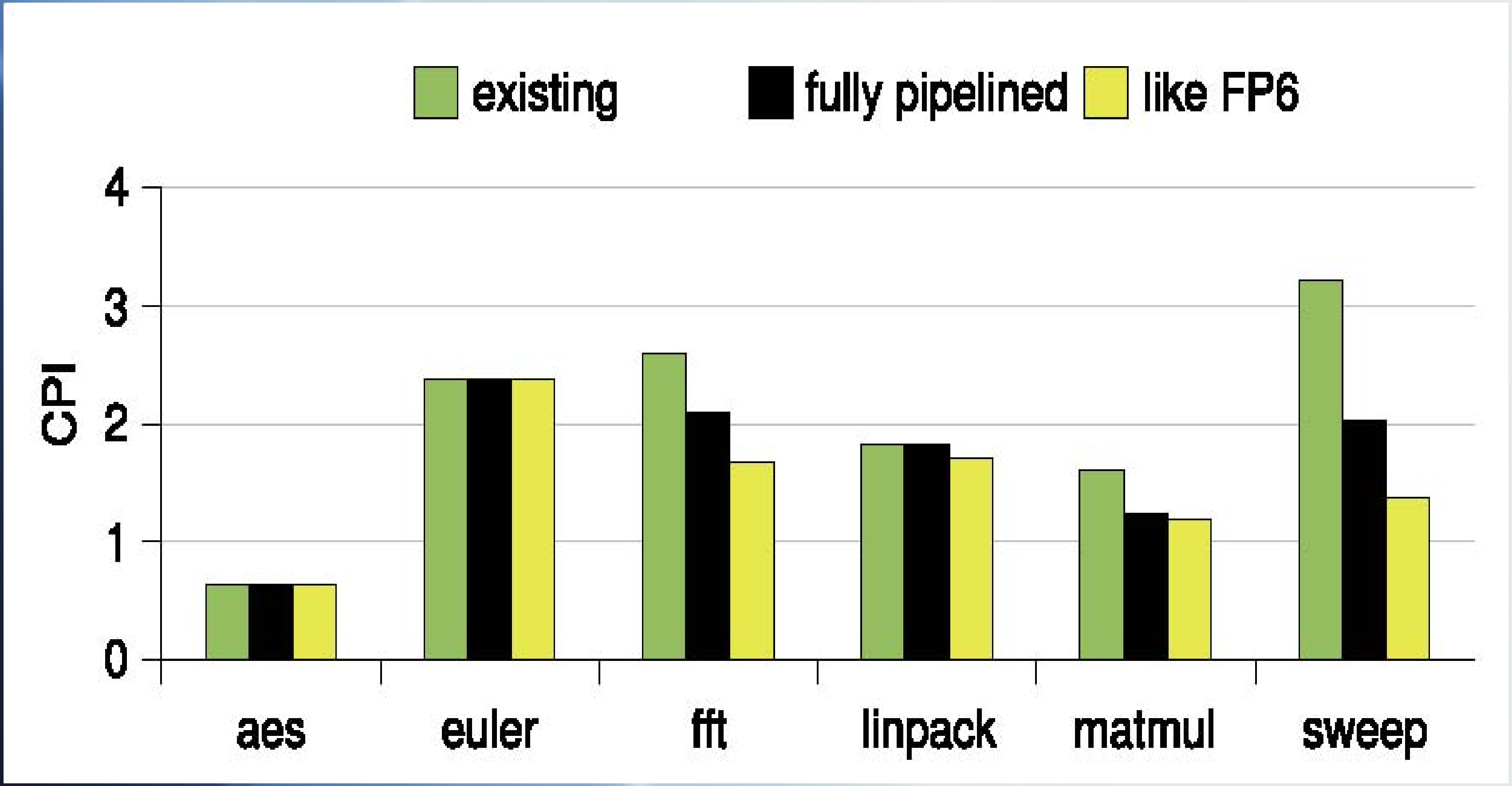
Error Attribution



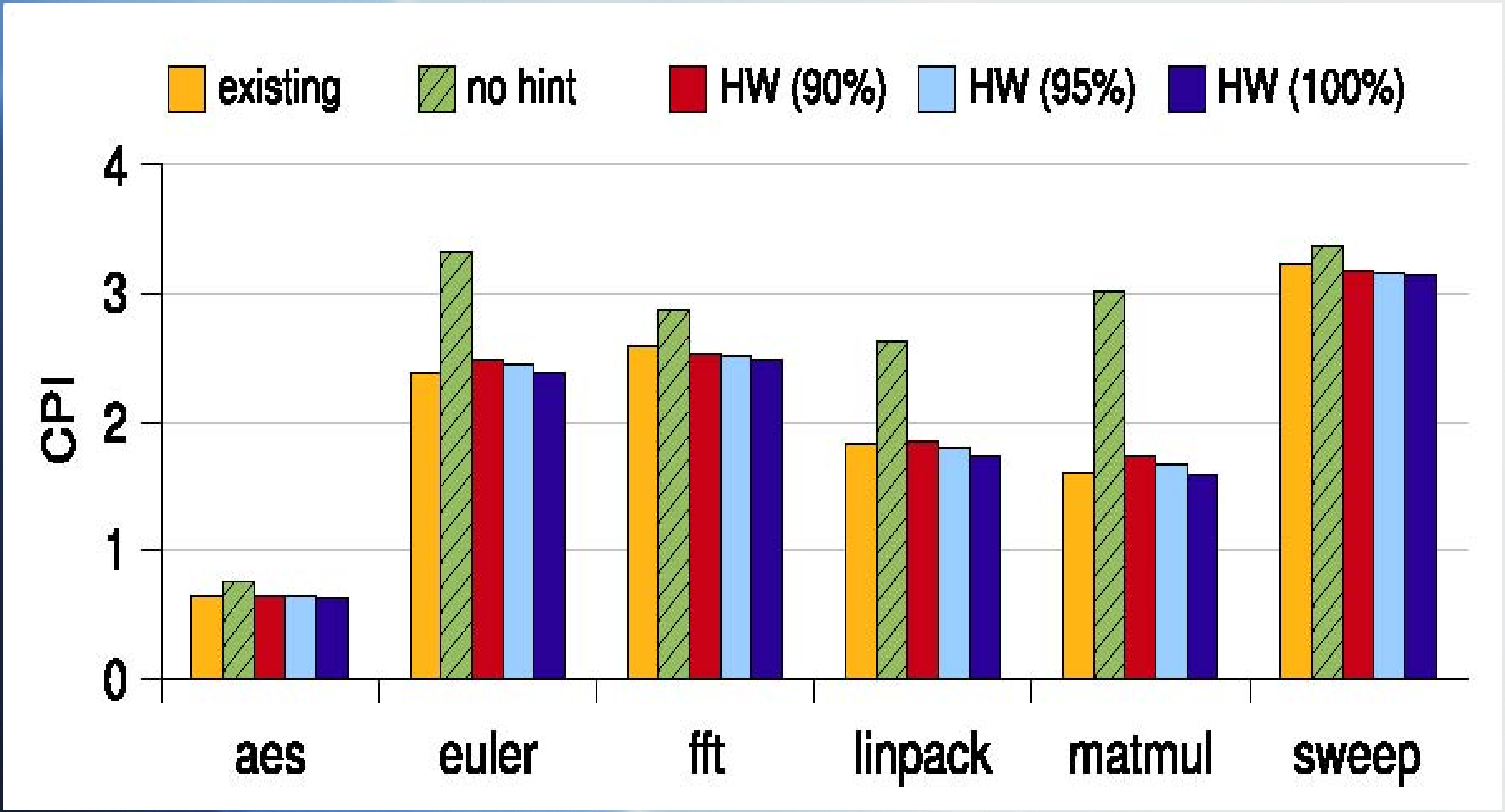
Decomposing CPI



Fully Pipelined, Faster FPD



Hardware Branch Predictor



Conclusions and Future Work

- Method produces accurate, fast predictive models
  - Cell, Niagara 1 and 2, Itanium, Opteron
- Complete Niagara 2, Opteron
- Cell improvements
  - Model communication (DMA)
  - Model the PPE
- Extend methodology for multithreading, power models, better flexible cache model
- Integrate into communication model for MP system model

Thanks

Any Questions???

QuickTime™ and a
decompressor
are needed to see this picture.

Extra Slides

12-Step Method (1)

1. Determine if stall-at-issue or stall-at-use
2. Determine CPI_I
3. Determine factors that influence CPI_S (e.g., data dependences, branch mis-predicts, partially pipelined EUs
4. Identify type, number, latency, behavior under contention of EUs
5. Determine cache access times
6. Determine main memory access latency
7. Determine branch mis-prediction penalty

12-Step Method (2)

8. Determine microarchitectural structures with variable service time (e.g., EUs, memory, branch predictors)
9. Collect application profile
   - CPI_I
   - Dynamic instruction mix
   - Dependence distance histograms
   - Cache hit/miss statistics
   - Branch predictor accuracy statistics
   - Special histograms such as prefetch-load distance (Itanium), hint-branch distance (Cell) relevant to Step 8

12-Step Method (3)

10. Identify tools to collect application profile (typically performance counters and binary instrumenters)
11. Build model
12. Validate model