A Framework for Modeling and Optimization of Prescient Instruction Prefetch

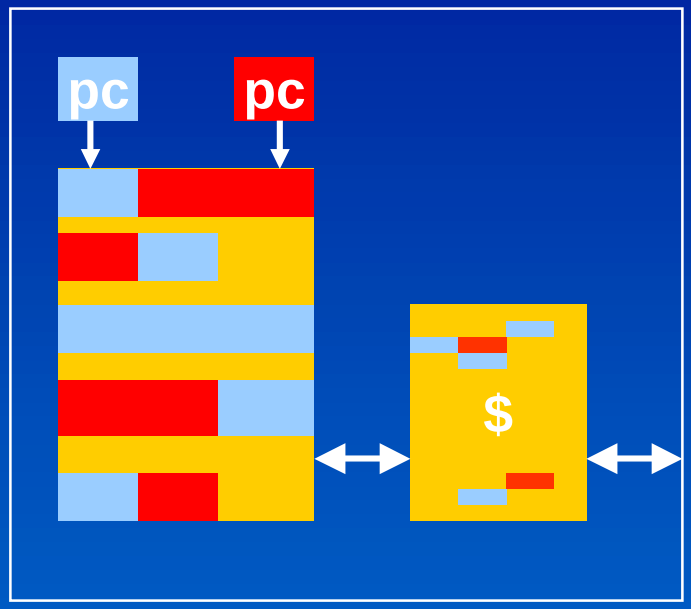
ACM Sigmetrics 2003 – June 12, 2003

Tor M. Aamodt[†‡], Pedro Marcuello[§], Paul Chow[‡], Antonio Gonzalez[§]

Per Hammarlund[¶], Hong Wang[†], John P. Shen[†]

[†]Microprocessor Research, Intel Labs
[‡]Dept. of Electrical and Computer Engineering, University of Toronto
[§]Intel Barcelona Research Center
[¶]Desktop Products Group, Intel Corp

Multithreading



Single chip, multiple flows of control

Question: How might a single-threaded application exploit this hardware capability?

Helper Threads

Use spare thread context(s) to reduce μArch bottlenecks. Typically do not need to satisfy all correctness constraints.

Related work on helper threads

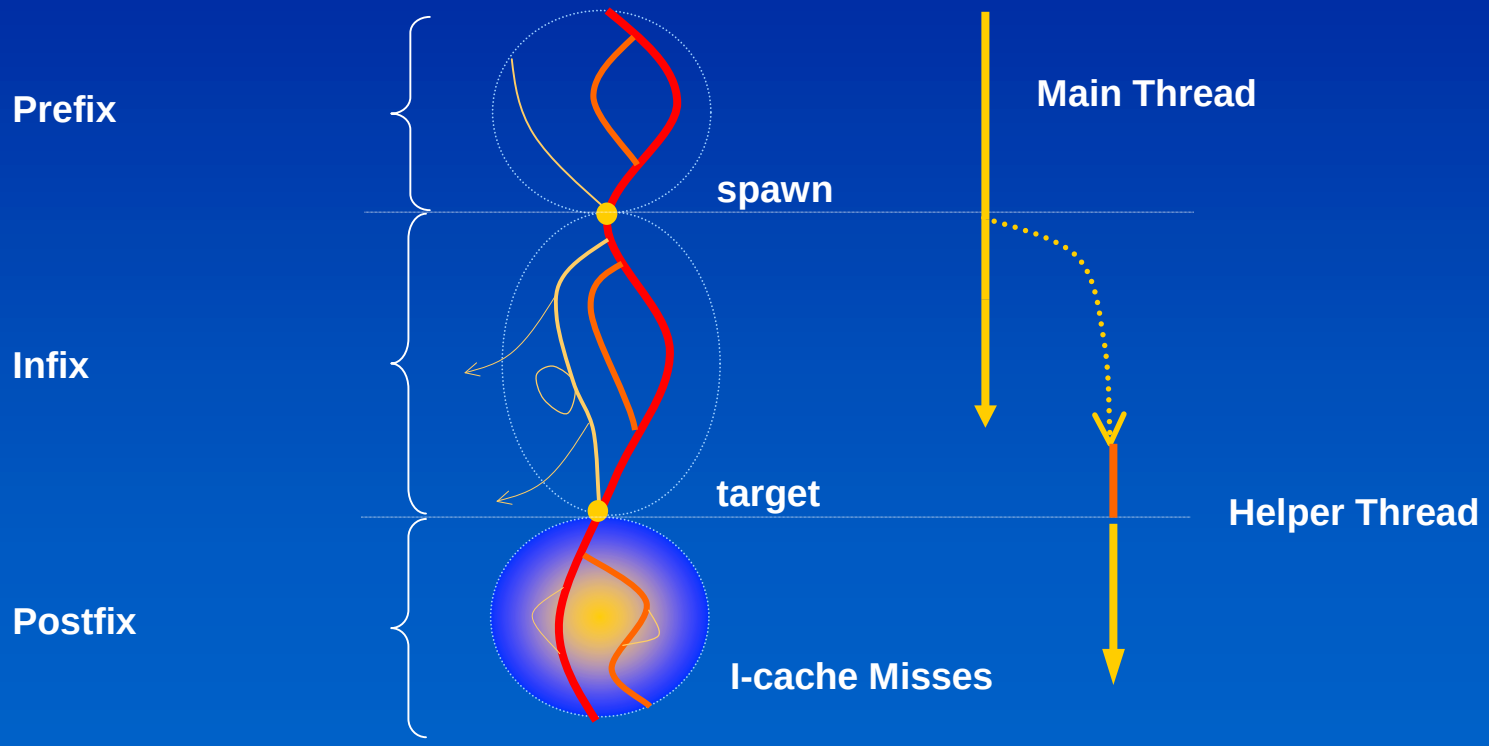
- Helper threads : Chappell & Patt, Dubois & Song
- Slices: Zilles & Sohi, Roth & Sohi
- Data prefetch: Zilles & Sohi, Collins et al., Annavaram & Davidson, Luk, Moshovos et al., Liao et al.
- Branch prediction: Chappell & Patt

This work: first work to study using helper threads for instruction prefetch (may also help TC pre-building)

Existing/Proposed Techniques

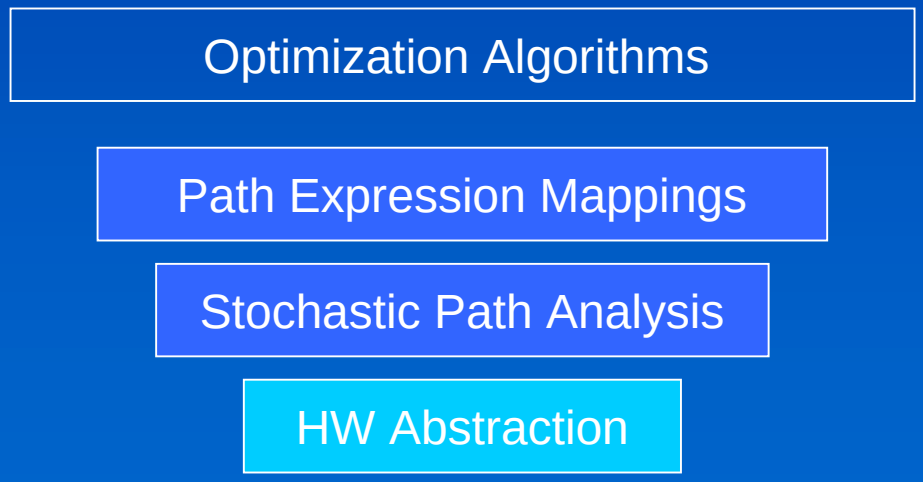
- Traditional hardware - scalability
- Helper thread – a few “delinquent” instruction
- Runahead – need simultaneous I & D miss

Prescient Instruction Prefetch

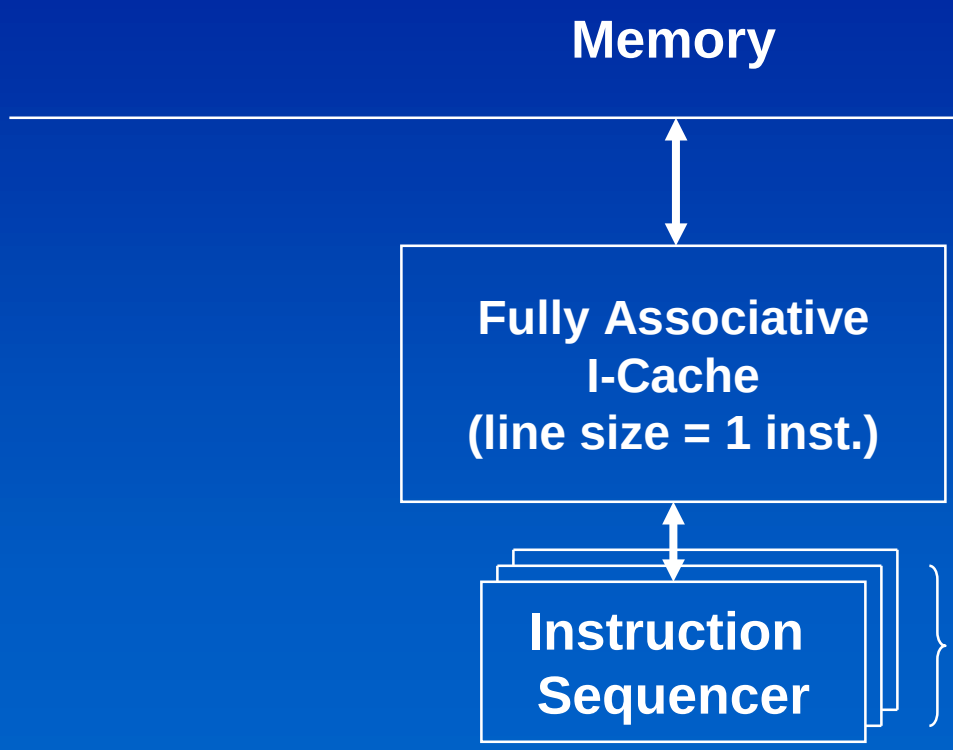


Optimization of Prescient Instruction Prefetch

- Optimization problem can be divided into **two** parts
  1. **Selection of SPAWN-TARGET pairs**
  2. Optimization of resulting thread code, and hardware used to run it
- This paper focuses on the first issue only



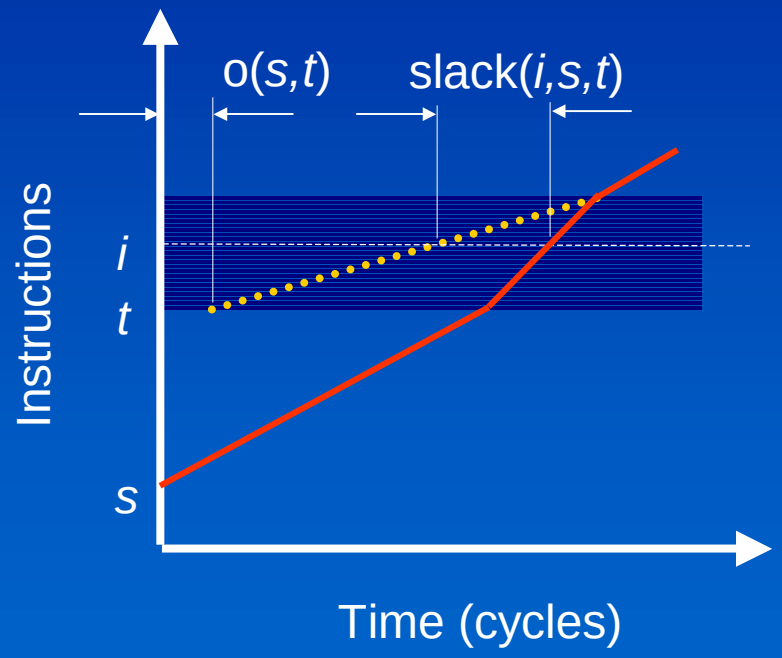
HW Abstraction



Intra-procedural control flow = Markov Chain

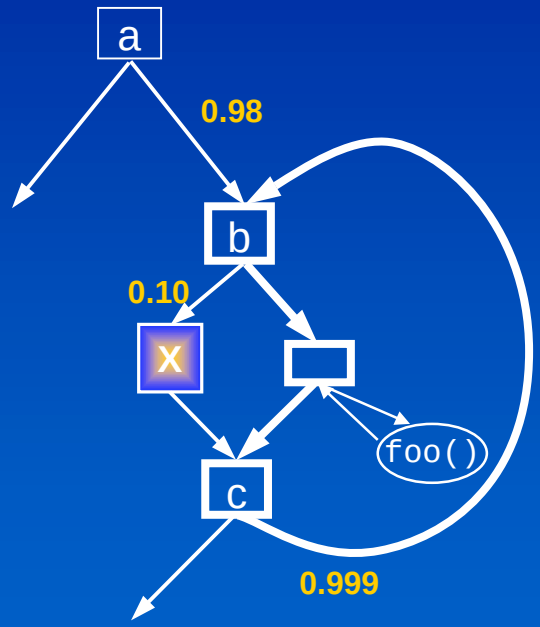
Call/returns paired

HW Abstraction
Prescient Instruction Prefetch



Spawn-Target Selection Tradeoffs

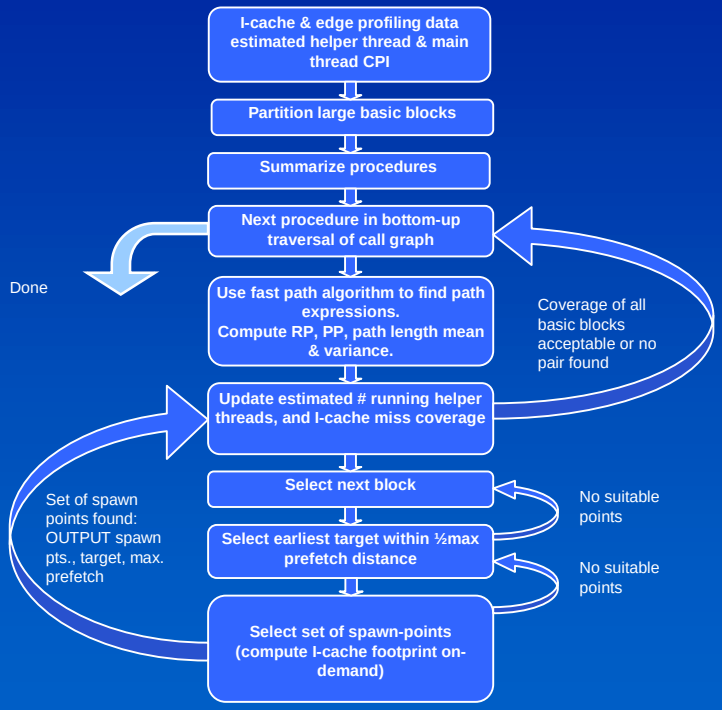
- S&T highly correlated
- S and T should be far apart so slack is larger than memory latency.
- S->T instruction footprint should fit in I-cache; T->S should not.



Quantifying Tradeoffs

METRIC	Aspect Quantified
Reaching Probability	accuracy
Posteriori Probability	coverage
Expected Path Length	timeliness
Path Length Variation	timeliness
Path Footprint	timeliness, necessity

Spawn-Target Selection Algorithm

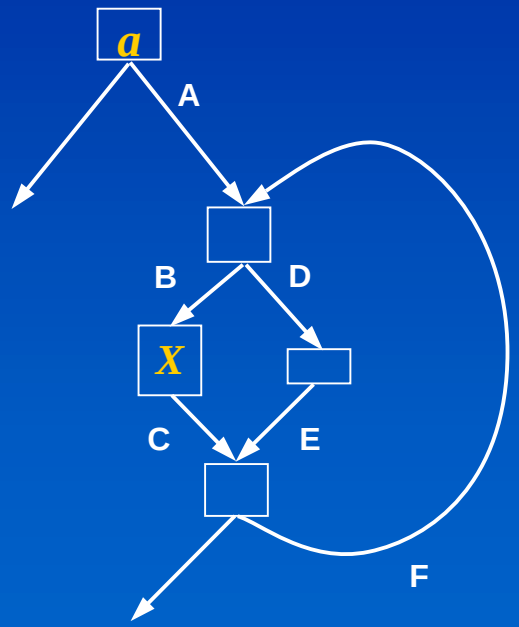


- Inputs: Profile data, estimated CPI
- Compute metrics / spawn-target value function
- Select using greedy heuristic

Path Expressions

- Regular expression describing all paths between two points.

$$P(a, X) = A \cdot \left(\left(\left(B \cdot C\right) \cup \left(D \cdot E\right)\right) \cdot F\right)^* \cdot B$$



Fast Path Expression Algorithm

- [Tarjan 1981] : general approach to solving path problems efficiently.
- Examples: solving Ax=b, shortest paths, data flow analysis.

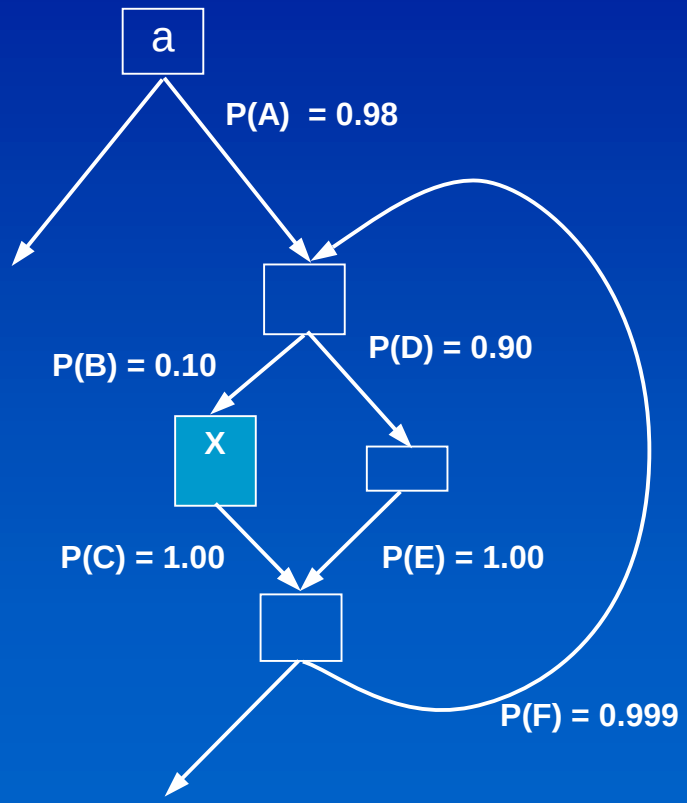
Example:
Reaching Probability

concatenation	$[R_1 \cdot R_2]$	$=$	pq
union	$[R_1 \cup R_2]$	$=$	$p + q$
closure	$[R_1^*]$	$=$	$\frac{1}{1-p}$

$$P(a, X) = A \cdot \left(\left(\left(B \cdot C\right) \cup \left(D \cdot E\right)\right) \cdot F\right)^* \cdot B$$

$$[P(a, X)] = 0.98 \cdot \left(\frac{1}{1.0 - (0.1(\underline{0.0}) + 0.90(1.0)) \cdot (0.999)}\right) \cdot 0.10$$

$$\cong 0.97$$



Mappings

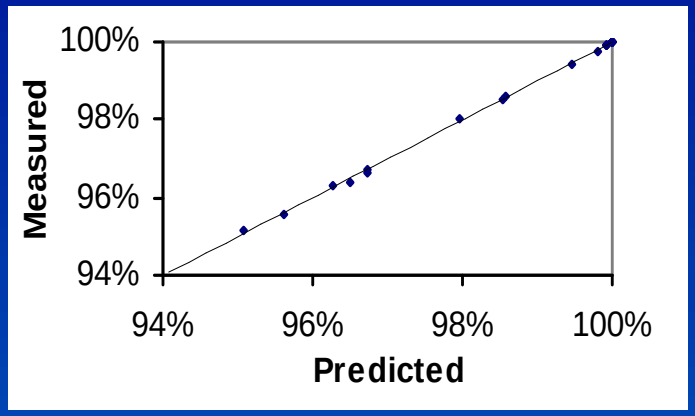
	Reaching Probability	Expected Path Length	Path Length Variance
concatenation $[R_1 \bullet R_2]$	pq	$X + Y$	$v + w$
union $[R_1 \cup R_2]$	$p + q$	$\frac{pX}{1-p}$	$\frac{p(v+X^2)+q(w+Y^2)}{p+q} - \left(\frac{pX+qY}{p+q}\right)^2$
closure $[R_1^*]$	$\frac{1}{1-p}$	$\frac{pX}{1-p}$	$\frac{p(v+X^2)}{1-p} + \left(\frac{pX}{1-p}\right)^2$

Decompose problem: $\sum$ **E[X| follow p $\in$ R]•P[follow p $\in$ R]**

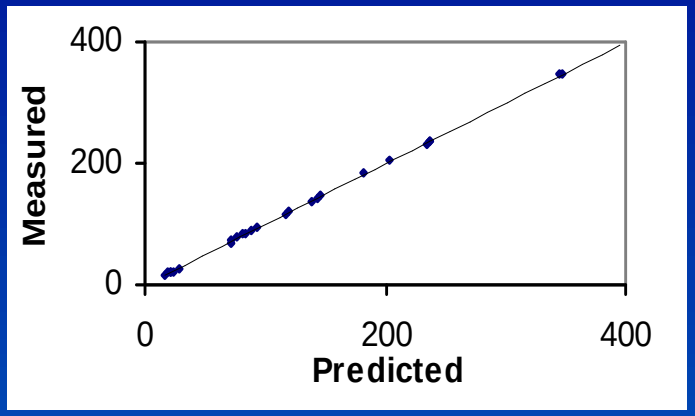
Path Footprint

$$F(x,y) = \frac{1}{RP(x,y)} \sum_{v} size(v) \cdot RP_{\alpha}(x, v \mid \neg y) \cdot RP_{\beta}(v, y)$$

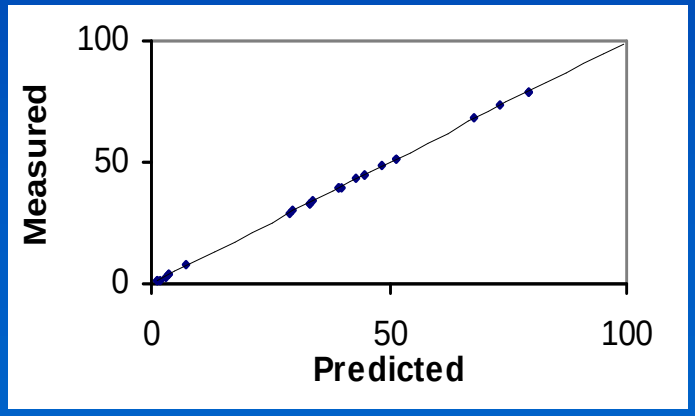
Accuracy: vs. Monte Carlo



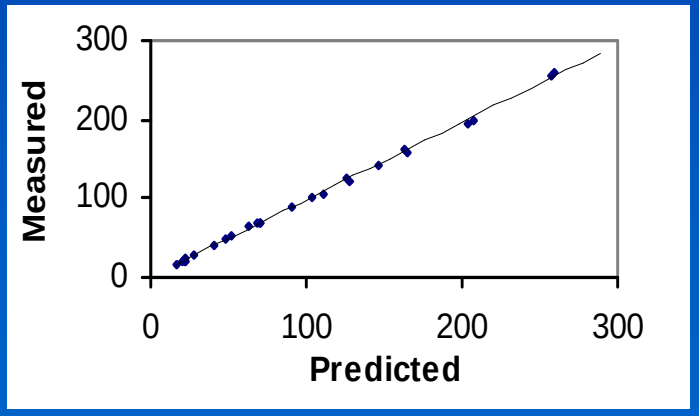
Reaching Probability



Expected Path Length

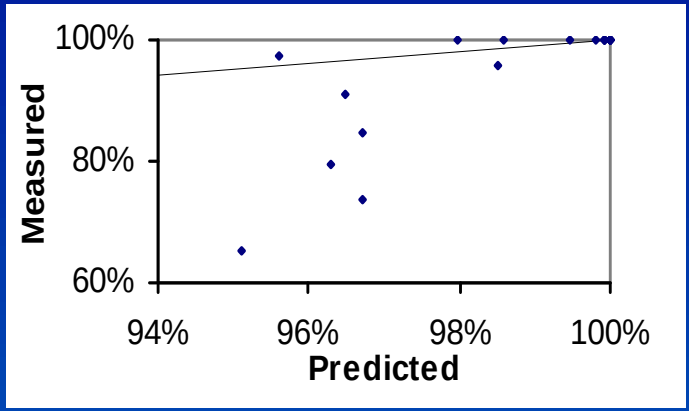


Path Length Variation

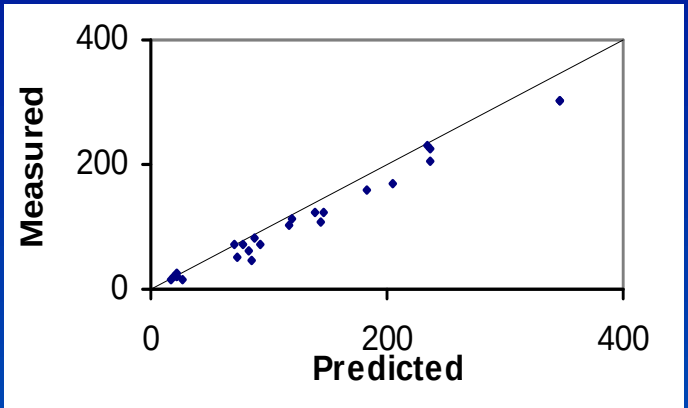


Path Footprint

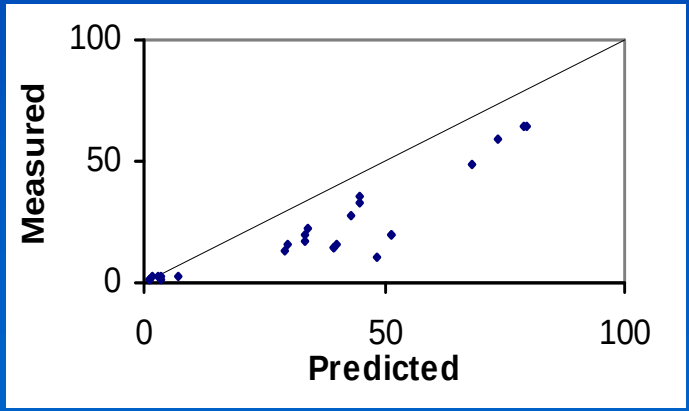
Accuracy: vs. Execution



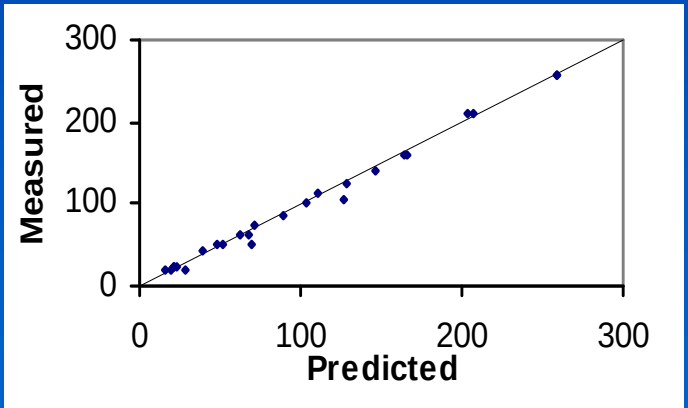
Reaching Probability



Expected Path Length

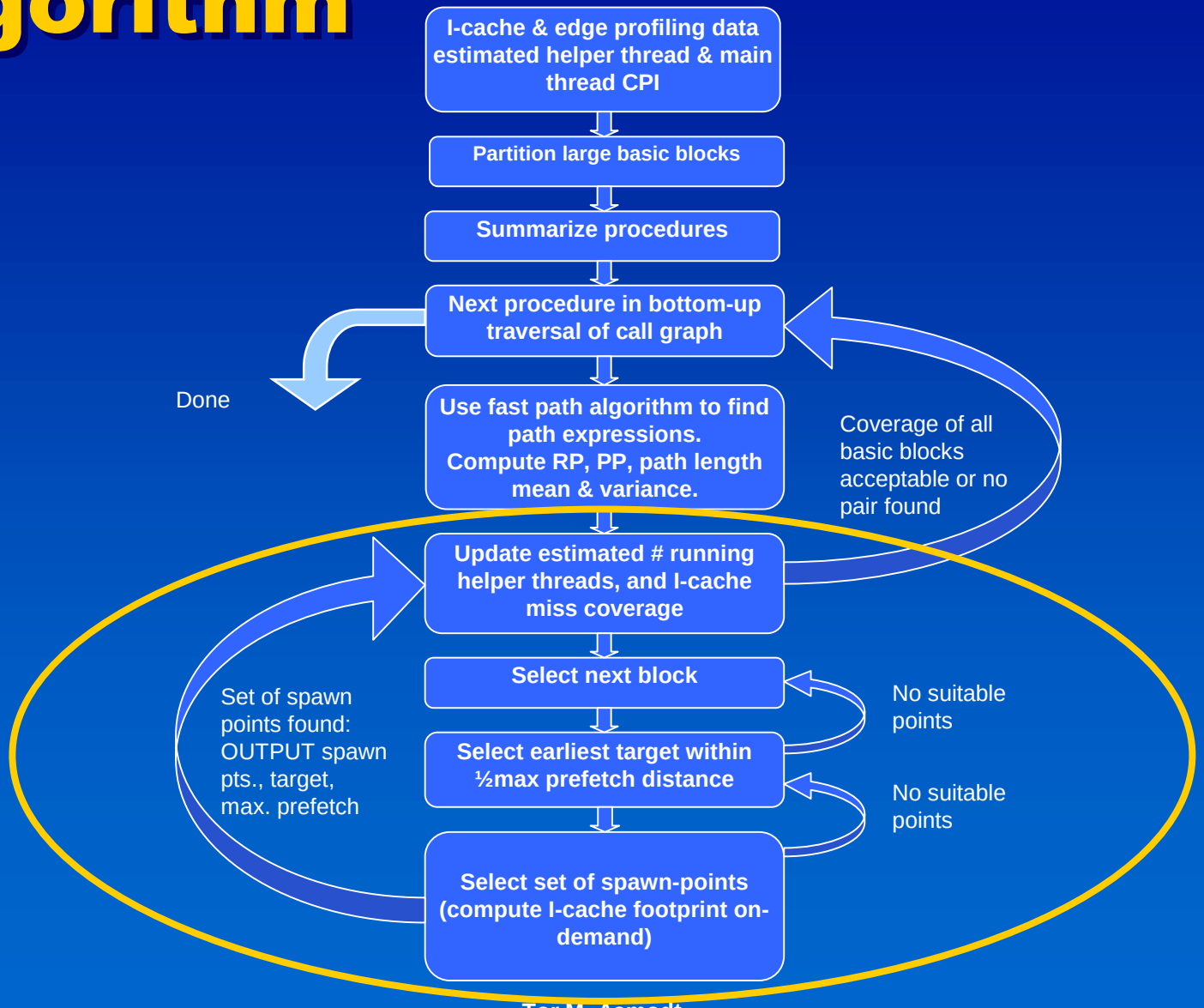


Path Length Variation



Path Footprint

Spawn-Target Selection Algorithm



Selection Algorithm Details

Loop over basic blocks (ranked by E[#i-misses])

1. Select target, then select spawn

   value(spawn) $\propto$

   PP•RP•E[postfix size]•P[miss]•P[!evicted]•P[# ht < # ctx]

2. Update coverage metrics

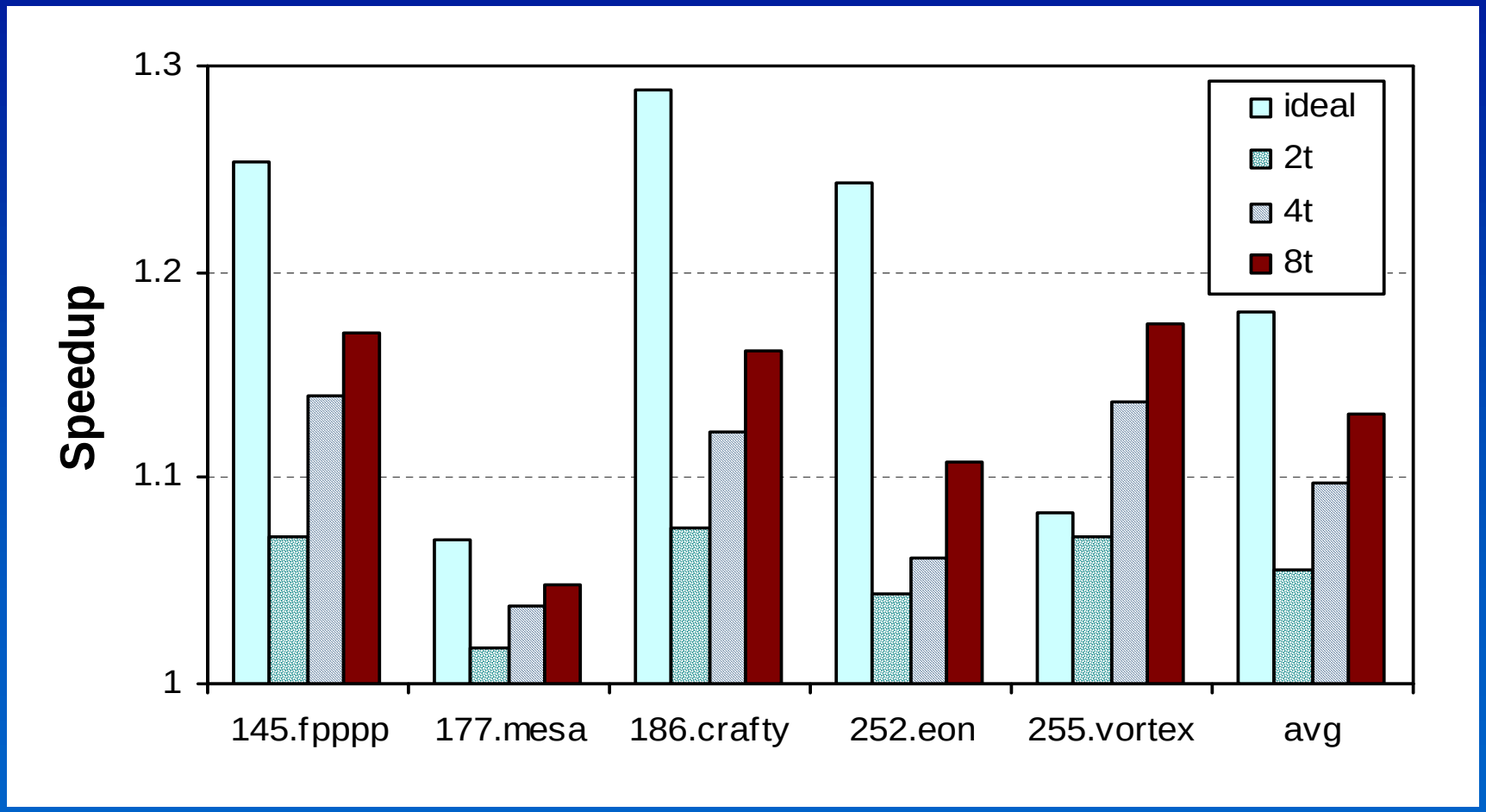
   - # helper threads = $\sum$ PP(s,i) • P[still running]
   - # i-cache misses -=

     PP(t,s) •PP(t,i) • P[still running] • (#i-cache misses)

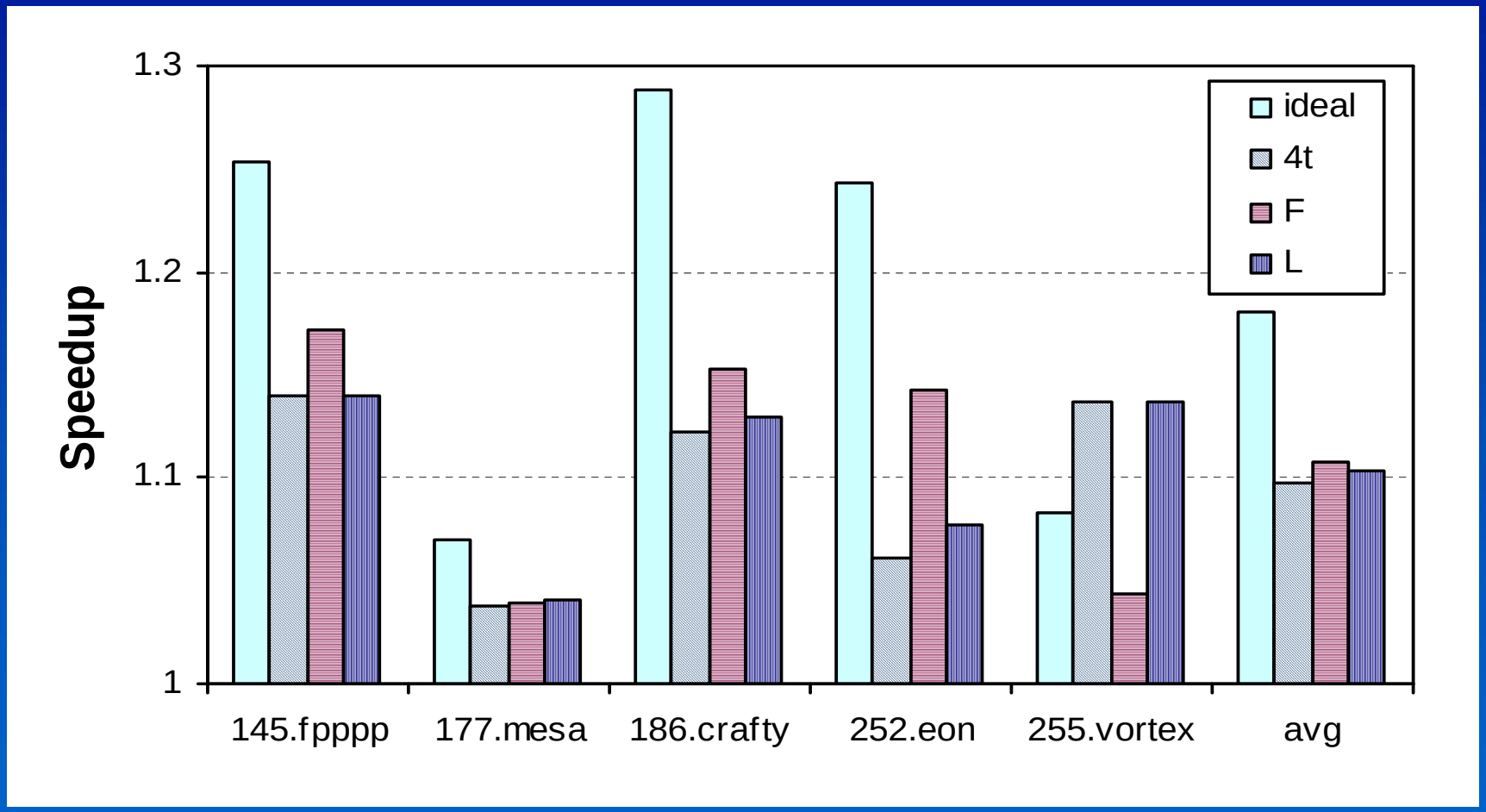
Hardware Details

Threading	SMT processor with 2, 4, or 8 hardware contexts
Pipelining	In-order 12-stage pipeline
Fetch	2 bundles from 1, or 1 bundle from 2 threads, prioritizing main thread, helpers ICOUNT
I-prefetch	next line prefetch (triggered on miss) Stream prefetch triggered by compiler hints (max. 4 outstanding prefetches per context)
Branch Pred.	2k-entry gshare. 256-entry 4-way assoc. BTB. Helper threads oracle branch prediction (always follow correct path)
Issue	2 bundles from 1, or 1 bundle from 2 threads Prioritize main thread, helpers: round-robin
Caches	L1 (separate I&D): 16KB 4-way 1-cycle L2 (shared) : 256 KB 4-way 14-cycles L3 (shared) : 3072 KB 12-way 30-cycles
Memory	230-cycles, TLB miss = 30 cycles

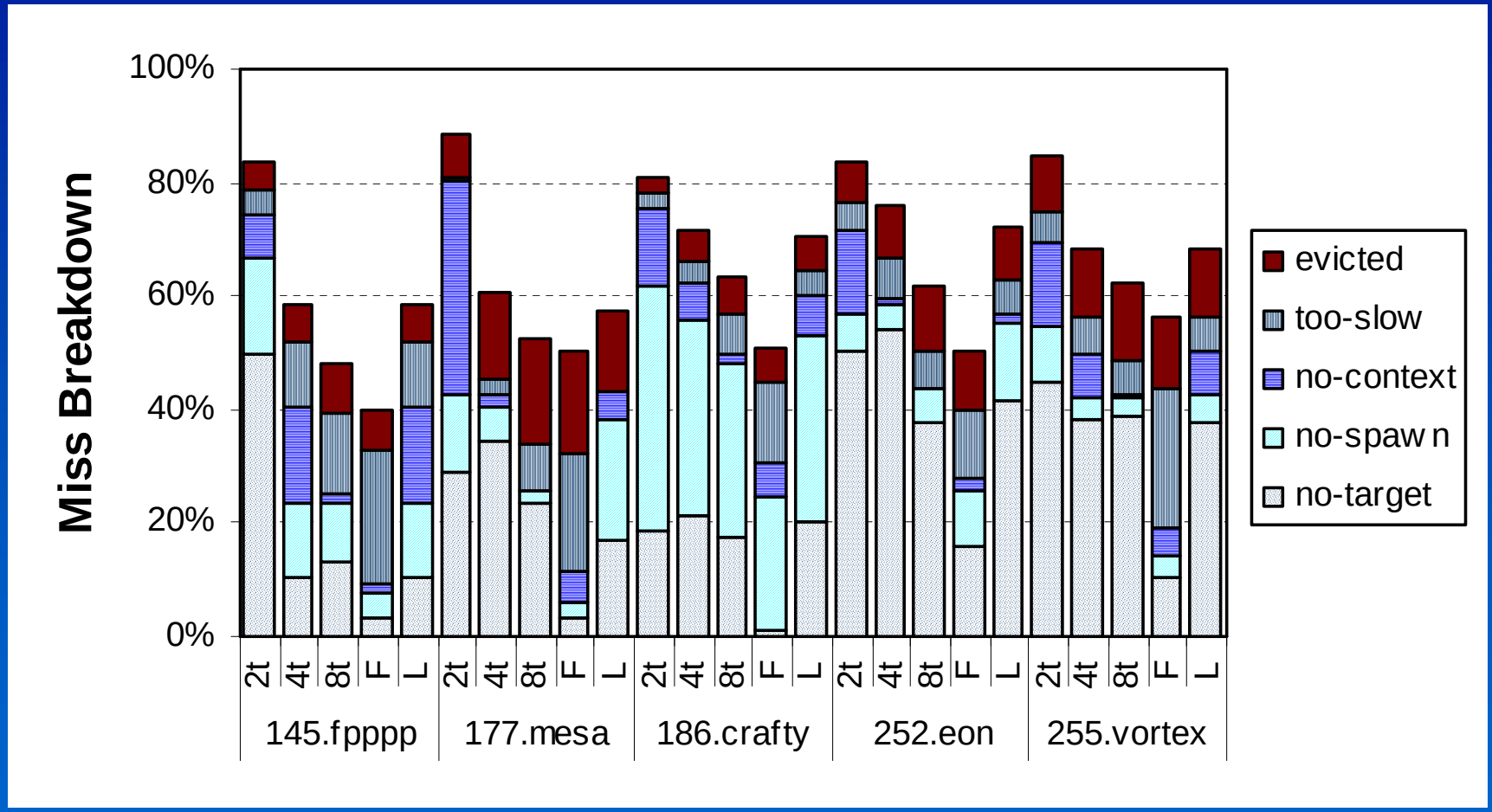
Performance Impact



Performance Impact



Source of remaining I-cache misses



Helper Thread Characteristics

benchmark	static # pairs	dyn. #pairs	infix region size	postfix region size
145.fpppp	62	378528	622	162
177.mesa	34	210519	1186	255
186.crafty	166	560200	573	129
252.eon	152	407516	691	120
255.vortex	1348	438722	1032	142

Summary

- Limit study indicates Prescient Instruction Prefetch may yield speedups from 4.8% to 17%
- Introduce a framework for selecting spawn-target pairs based upon statistical analysis using path expressions.
- Future work
  - hardware & slice optimization (under review)
  - more sophisticated modeling (branch corr., phases)
  - optimization algorithms