A Quantitative Framework for Pre-Execution Thread Selection

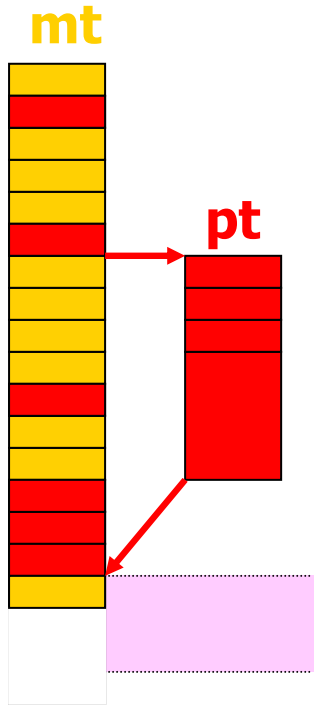
Amir Roth
University of Pennsylvania

Gurindar S. Sohi
University of Wisconsin-Madison

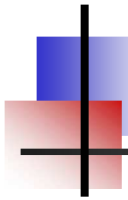
MICRO-35
Nov. 22, 2002

Pre-execution: proactive multithreading to tolerate latency

- Start with cache miss
- Extract computation (dynamic backward slice)
- Launch as redundant thread (p-thread)
  - **P-thread** = trigger (launch PC) + body (slice)



- Effect I: **compressed fetch**
  - P-thread fetches→initiates→completes miss earlier
- Effect II: **decoupling**
  - Miss in p-thread doesn't stall main thread
- Sum effect : **miss "moved" to p-thread**



A Quantitative Framework for Pre-Execution Thread Selection

- This paper is not about pre-execution

- These papers are
  - Assisted Execution [Song+Dubois, USC-TR98]
  - SSMT [Chappell+, ISCA99, ISCA02, MICRO02]
  - Virtual Function Pre-Computation [Roth+, ICS99]
  - DDMT [Roth+Sohi, HPCA01]
  - Speculative Pre-Computation [Collins+, ISCA01, MICRO01, PLDI02]
  - Speculative Slices [Zilles+Sohi, ISCA01]
  - Software-Controlled Pre-Execution [Luk, ISCA01]
  - Slice Processors [Moshovos+, ICS01]

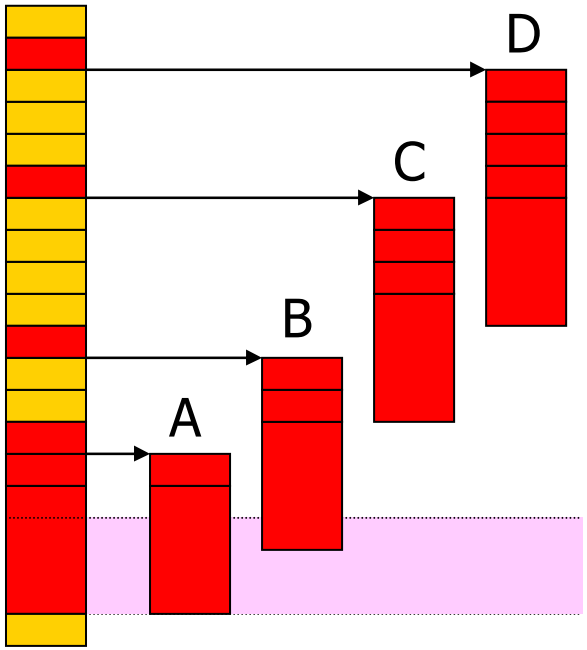
Thread Selection

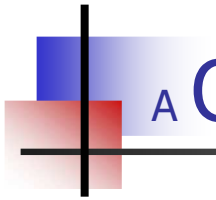
P-thread selection: pre-?-execution

- What p-threads? When to launch? (same question)

- **Static p-threads**
  - Target static problem loads
  - Choose <trigger:body> once
  - Launch many instances

- **Hard:** antagonistic criteria
  - Miss coverage
  - Latency tolerance per instance
  - Overhead per instance
  - Useless instances
  - Longer p-thread = better, worse





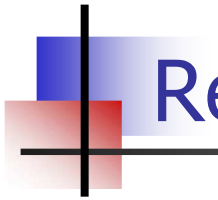
A Quantitative Framework for Pre-Execution Thread Selection

Quantitative p-thread selection framework

- Simultaneously optimizes all four criteria
- Accounts for p-thread overlap (later)
- Automatic p-thread optimization and merging (paper only)

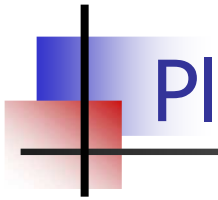
- Abstract pre-execution model (applies to SMT, CMP)
  - 4 processor parameters
- Structured as pre-execution limit study
  - May be used to study pre-execution potential

This paper: static p-threads for L2 misses



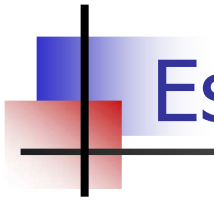
Rest of Talk

- Propaganda
- Framework proper
  - Master plan
  - Aggregate advantage
  - P-thread overlap
- Quick example
- Quicker performance evaluation



Plan of Attack and Key Simplifications

- Divide
  - P-threads for one static load at a time
- Enumerate all possible* static p-threads
  - Only p-threads sliced directly from program
  - A priori length restrictions
- **Assign benefit estimate to each static p-thread**
  - Number of cycles by which execution time will be reduced
- Iterative methods to find set with maximum advantage
- Conquer
  - Merge p-threads with redundant sub-computations



Estimating Static P-thread Goodness

Key contribution: simplifications for computational traction

1. One p-thread instance executes at a time (framework)
   ➔ P-thread interacts with main thread only
2. No natural miss parallelism (framework, not bad for L2 misses)
   ➔ P-thread interacts with one main thread miss only
3. Control-less p-threads (by construction)
   ➔ Dynamic instances are identical
4. No chaining (by construction)
   ➔ Fixed number of them

Strategy

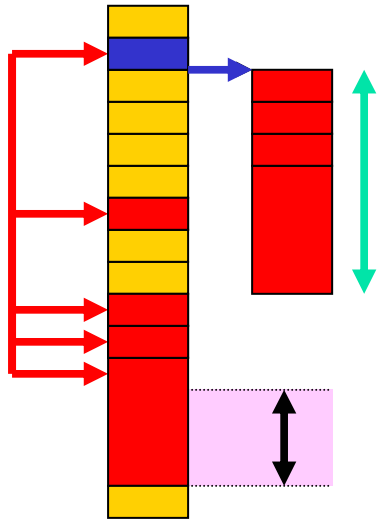
- Model interaction of one dynamic instance with main thread
- Multiply by (expected) number of dynamic instances

Aggregate Advantage (**ADV**agg)

- **ADV**agg: static p-thread goodness function
  - Cycles by which static p-thread will accelerate main thread
  - Combines four criteria into one number

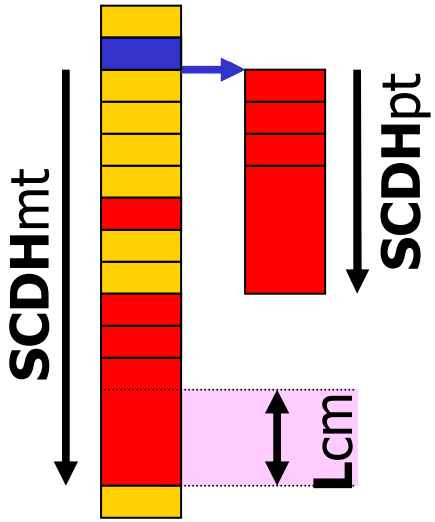
ADVagg = (**DC**pt-cm * **LT**) – (**DC**trig * **OH**)

- Collect raw materials from traces
  - **DC**trig = #instances launched
  - **OH** = overhead per (**SIZE** / **BWSEQ**proc)
  - **DC**pt-cm = #misses covered
  - **LT** = latency tolerance per (next slide)



Dynamic P-thread Latency Tolerance (**LT**)

- **LT**
  - Starting at trigger
  - Main thread miss execution time (**SCDH**mt)
  - Minus p-thread miss execution time (**SCDH**pt)
  - Bounded by miss latency (**L**cm)



LT = min(**SCDH**mt – **SCDH**pt, **L**cm)

- **SCDH:** sequencing constrained dataflow height
  - Estimate execution time of instruction sequence
  - For each insn: dataflow + **fetch constraint (insn# / BWSEQ)**
    - Computation: explicit
    - Other main thread work: sparse insn #'s

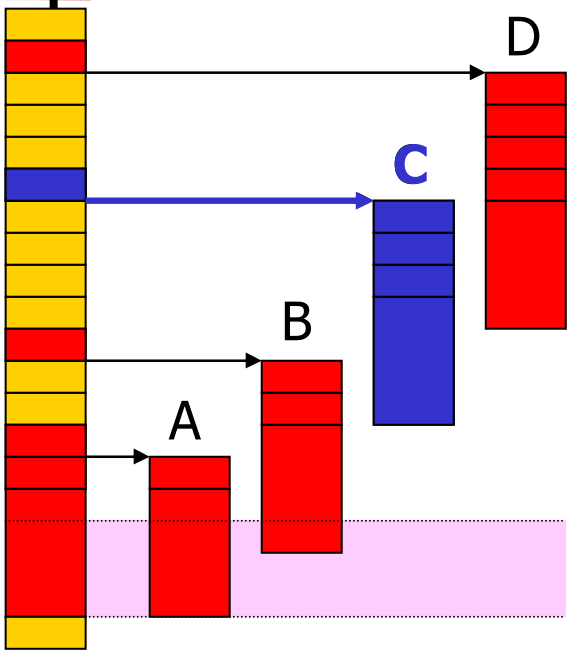
SCDH and LT Example

- **Dataflow constraints:**
  - Main-thread: miss latency = 8, other latency = 1, serial deps
  - P-thread: same
- **Sequencing constraints: insn# / fetch bandwidth**
  - Main thread: **sparse / 2**
  - P-thread: **dense / 1**

	SCDHmt	**SCDH**pt
addi R5, R5, #16	max(**0**/**2**, 0) + 1 = 1	max(**0**/**1**, 0) + 1 = 1
lw R6, 4(R5)	max(**7**/**2**, 1) + 1 = 5	max(**1**/**1**, 1) + 1 = 2
slli R7, R6, #2	max(**10**/**2**, 5) + 1 = 6	max(**2**/**1**, 2) + 1 = 3
addi R8, R7, #rx	max(**11**/**2**, 6) + 1 = 7	max(**3**/**1**, 3) + 1 = 4
lw R9, 0(R8)	max(**12**/**2**, 7) + 8 = **15**	max(**4**/**1**, 4) + 8 = **12**

LT = min(15 – 12, 8) = 3

ADVagg Example



- Params: 40 misses, 8 cycles each
  - Max **ADV**agg = [40*8] – [40*0] = 320
  - Impossible to achieve
- As p-thread length increases…
  - **LT** ↑, **DC**pt-cm ↓
  - **OH** ↑, **DC**trig ⇕

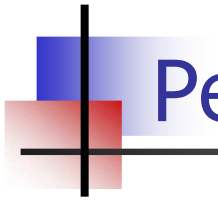
	DCpt-cm * **LT** = **LT**agg			–	**DC**trig * **OH** = **OH**agg			=	**ADV**agg
A	40	0	0		80	2/4	40		–40
B	40	5	200		80	3/4	60		140
C	**30**	**8**	**240**		**80**	**4/4**	**80**		**160**
D	30	8	240		100	5/4	125		115

P-thread Overlap

- F: B, C, D have positive **ADV**agg
- Q: Why not choose all three?
- A: They cover same misses (**LT**agg's overlap)
- **P-thread overlap:** framework…
  - Represents it: slice tree (paper)
  - Corrects for it: reduces **LT**agg by shared component

	OHagg	**LT**agg	**LT**agg-ovlp	**LT**agg-red	**ADV**agg-red
B	60	40*5 = 200	30*5 = 150	200–150 = 50	50–60 = –10
C	80	30*8 = 240			240–80 = 160

- Choose overlapping p-threads if **ADV**agg-red positive
  - Not in this example

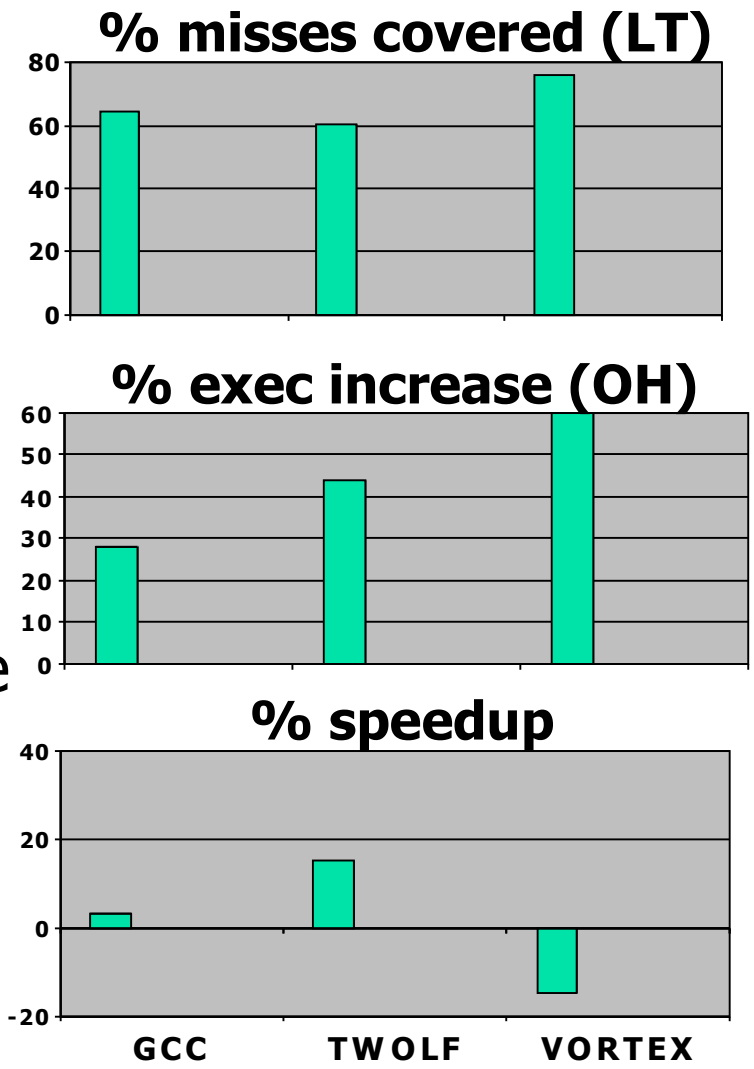


Performance Evaluation

- SPEC2000 benchmarks
  - Alpha EV6, –O2 –fast
  - Complete train input runs, 10% sampling
- Simplescalar-derived simulator
  - Aggressive 6-wide superscalar
  - 256KB 4-way L2, 100 cycle memory latency
  - SMT with 4 threads (p-threads and main thread contend)
- P-threads for L2 misses
  - Prefetch into L2 only

Contribution of Framework Features

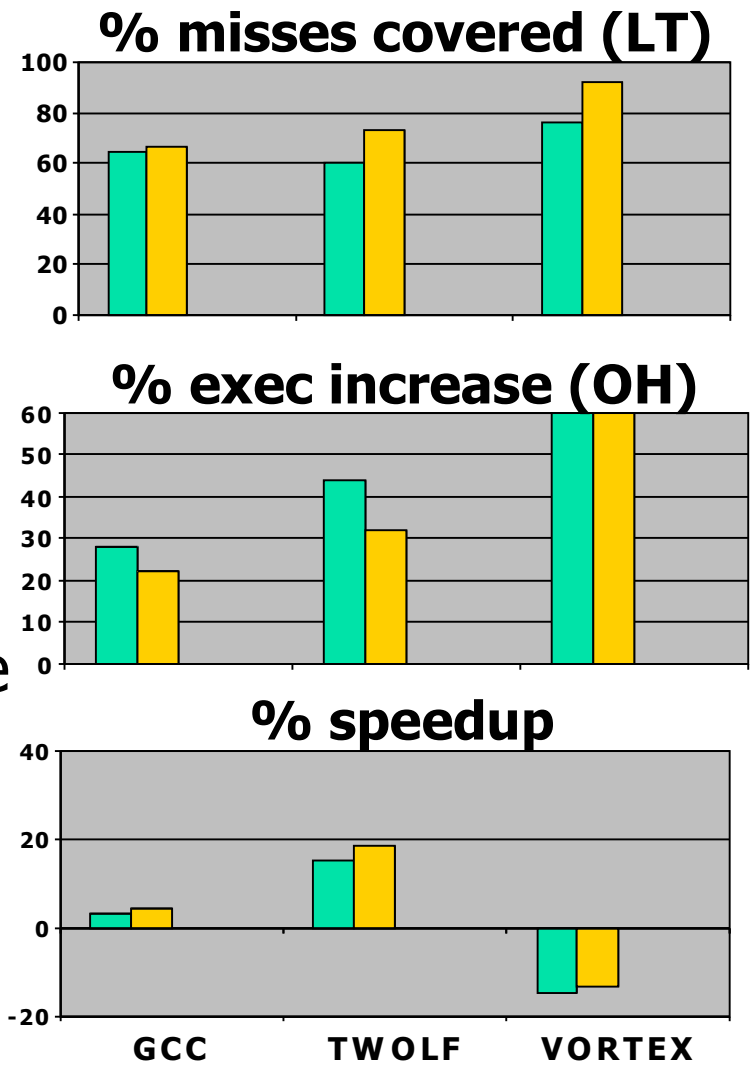
- Framework accounts for
  - Latency, overhead, overlap
  - Isolate considerations
- 4 experiments, 3 diagnostics
  - Measured via p-thread simulation
- **GREEDY**: as much LT as possible



Contribution of Framework Features

- Framework accounts for
  - Latency, overhead, overlap
  - Isolate considerations
- 4 experiments, 3 diagnostics
  - Measured via p-thread simulation

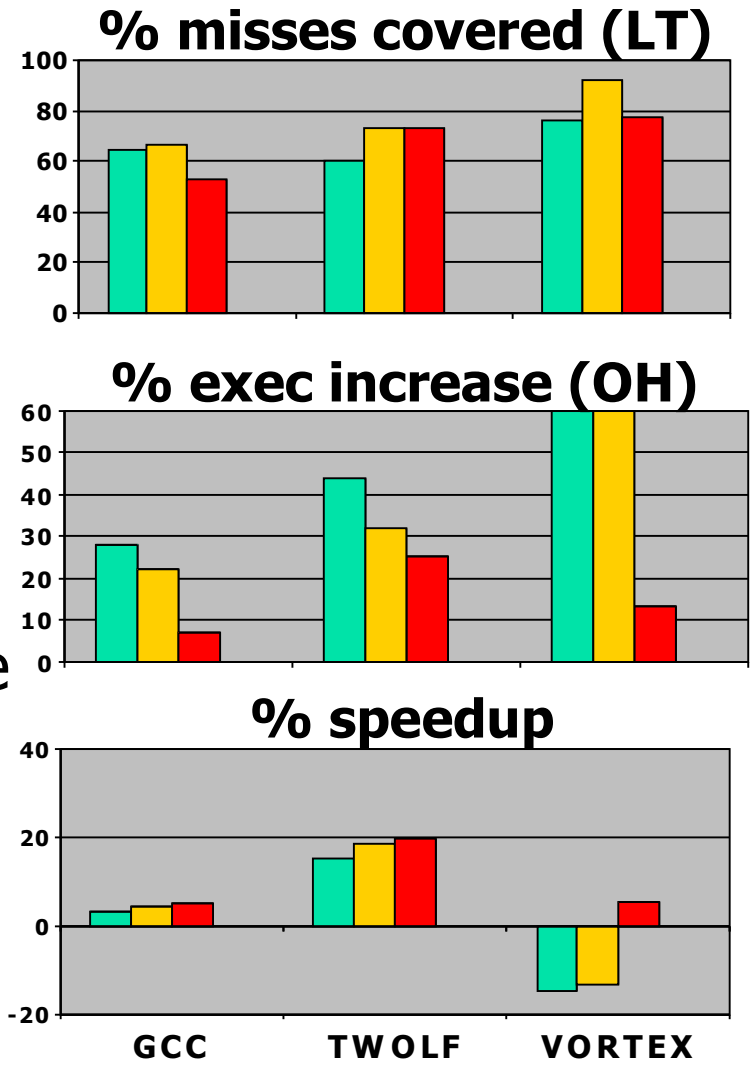
- **GREEDY**: as much LT as possible
- **+LT**: as much LT as needed



Contribution of Framework Features

- Framework accounts for
  - Latency, overhead, overlap
  - Isolate considerations
- 4 experiments, 3 diagnostics
  - Measured via p-thread simulation

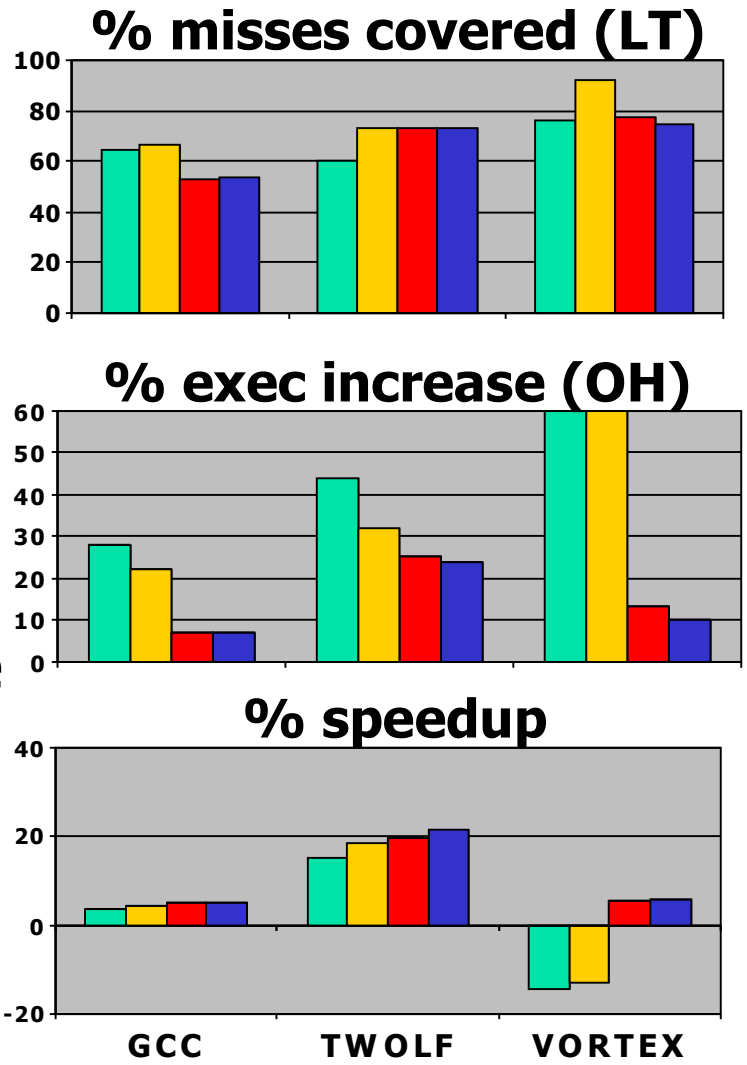
- **GREEDY**: as much LT as possible
- **+LT**: as much LT as needed
- **+OH**: account for overhead



Contribution of Framework Features

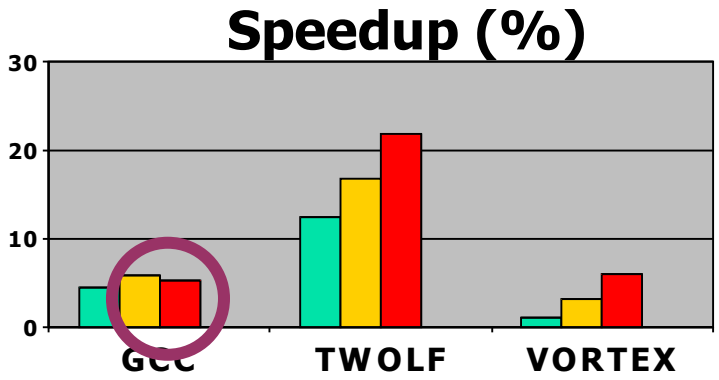
- Framework accounts for
  - Latency, overhead, overlap
  - Isolate considerations
- 4 experiments, 3 diagnostics
  - Measured via p-thread simulation

- **GREEDY**: as much LT as possible
- **+LT**: as much LT as needed
- **+OH**: account for overhead
- **+OVLP**: account for overlap

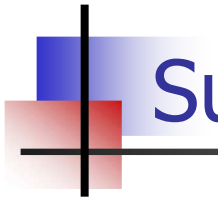


Pre-Execution Behavior Study

- Example: max p-thread length
  - Full framework
  - **8**, **16**, **32**
- Encouraging (intuitive) result
  - Flexibility increases performance
- Also in paper
  - Merging, optimization, input sets



- **ADV**agg just a model, not completely accurate
  - **ADV**agg **validation:** important part of paper
  - V1: **ADV**agg predictions should match simulated diagonostics
  - V2: lying to **ADV**agg should produce worse p-threads



Summary

- P-thread selection
  - Important and hard
- **Quantitative static p-thread selection framework**
  - Enumerate all possible static p-threads
  - Assign a benefit value (**ADV**agg)
  - Use standard techniques to find maximum benefit set
- Results
  - Accounting for overhead, overlap, optimization helps
  - Many more results in paper
- Future
  - **ADV**agg accurate? Simplifications valid?
  - Non-iterative approximations for real implementations