Dataflow-Analysis-Based Dynamic Parallel Monitoring

Michelle L. Goodstein[*], Shimin Chen[†], Phillip B. Gibbons[‡], Michael A. Kozuch[‡] and Todd C. Mowry[*]

[*]CMU [†]ICT CAS [‡]Intel Labs

INTRODUCTION

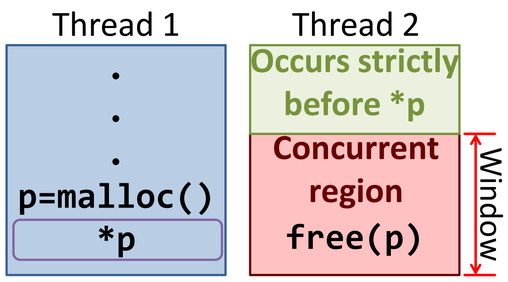
- Software bugs are common, even in sequential code
- Parallel s/w increasingly important → cause new "species" of bugs

Goal: Dynamic parallel monitoring without total order of app events
- Avoid inter-thread dependence tracking overheads
- No assumption of sequential consistency

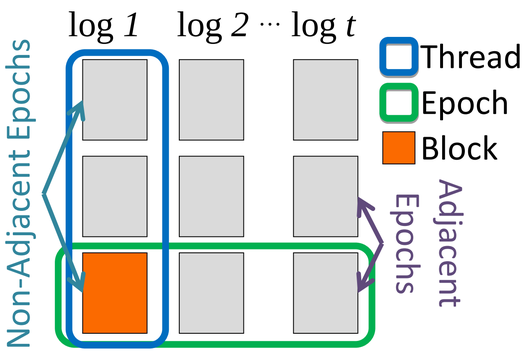
Contribution: *Dataflow-Analysis-Based Dynamic Parallel Monitoring*
- Software f/w to detect bugs in parallel programs at runtime

Challenges: Partial orderings, imprecision

KEY INSIGHT: WINDOWS OF UNCERTAINTY

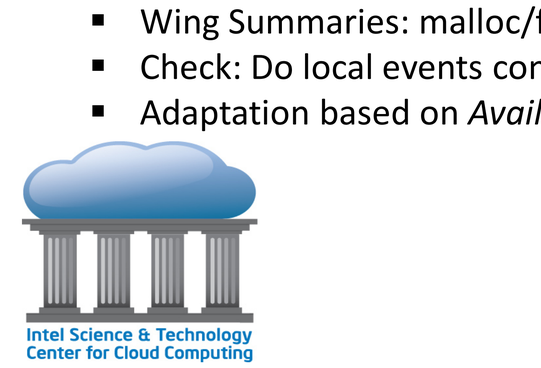


Proceed *without* capturing inter-thread data dependences
- Intuition: insts issued "far enough" apart → ordered within system
- Far enough: 1000s-10000s insts/thread



Divide thread execution into *epochs*
- Epoch size accounts for buffering in pipeline, memory system
- *Adjacent*: arbitrary inst interleavings
- *Non-adjacent*: known block ordering

BUTTERFLY ANALYSIS [ASPLOS '10]

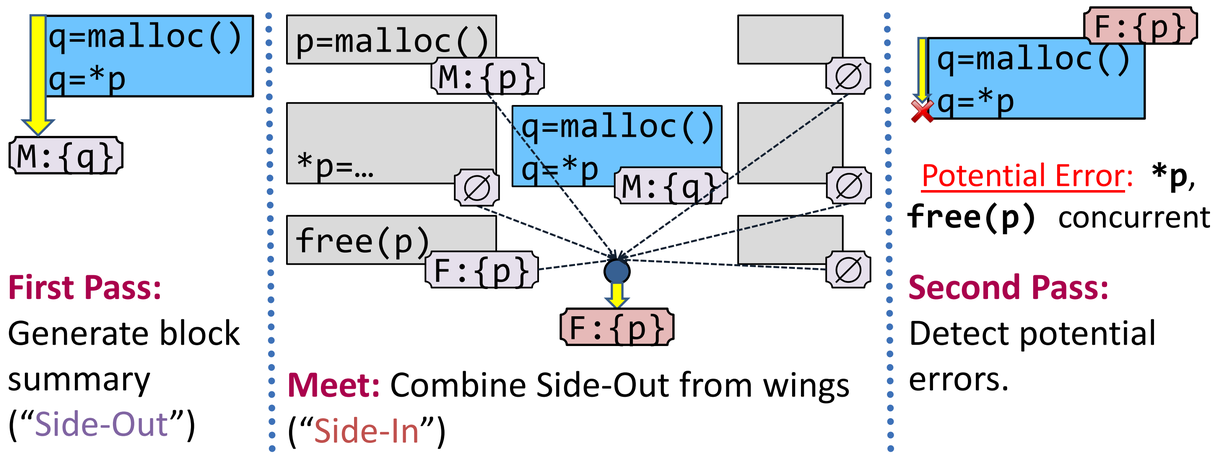


Analyze sliding window of only 3 epochs

Respect intra-thread data dependences:
- Head executes before Body
- Body executes before Tail

Instructions in wings may interleave with instructions in butterfly's body!

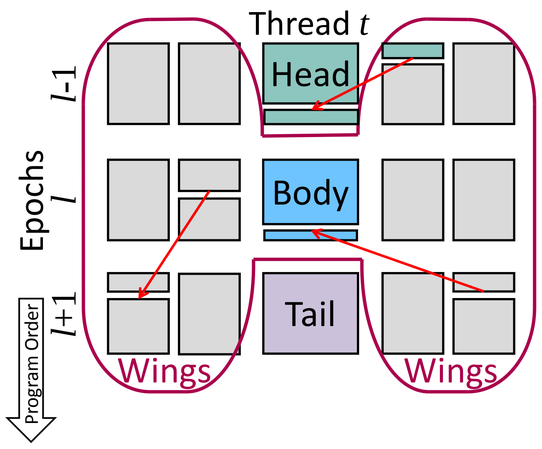
ADDRCHECK CASE STUDY



ADDRCHECK: Verifies accesses are to allocated regions
- Wing Summaries: malloc/free events from wings
- Check: Do local events conflict with wing summaries?
- Adaptation based on *Available Expressions* (dataflow analysis)

ADDRCHECK: [Nethercote, PhD Thesis '04]

CHRYSALIS ANALYSIS [PACT '12]



Incorporate high-level synch arcs
- Synch further subdivides execution; use *vector clocks*

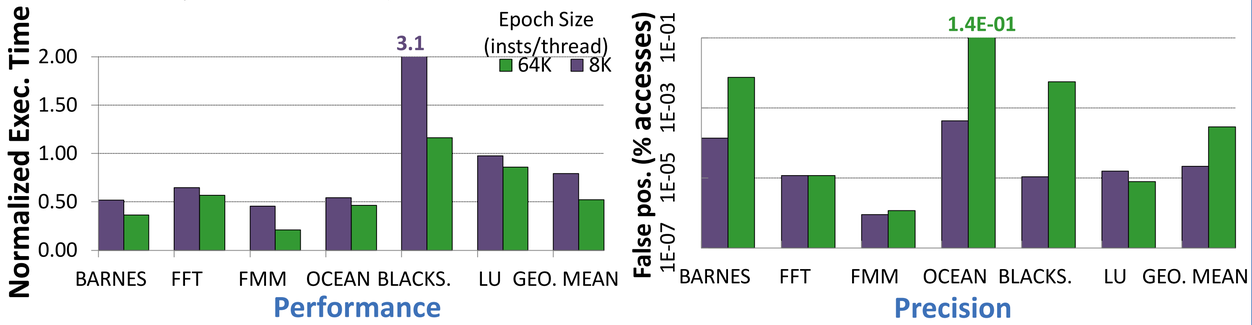
Increases **complexity** of:
- Thread execution model
- Local/global state computations
- Dataflow primitive computations

EXPERIMENTAL RESULTS

Prototypes built on *Log-Based Architecture* (LBA) f/w [Chen, ISCA '08]
- Full Butterfly/Chrysalis stack implemented in software
- Simulated hardware on shared-memory CMP using Simics
- LBA: execution traces, epoch boundary insertion

Butterfly Analysis ADDRCHECK Results:

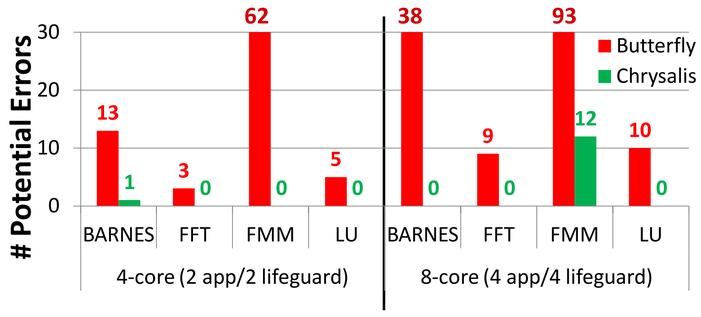
16-core CMP, mix of SPLASH/Parsec benchmarks



Larger epochs → better performance
- Can amortize cost of 2nd pass analysis over more instructions

Smaller epochs→ improved precision
- More ordering information, less need for conservative analysis

Chrysalis Analysis TAINTCHECK Results:

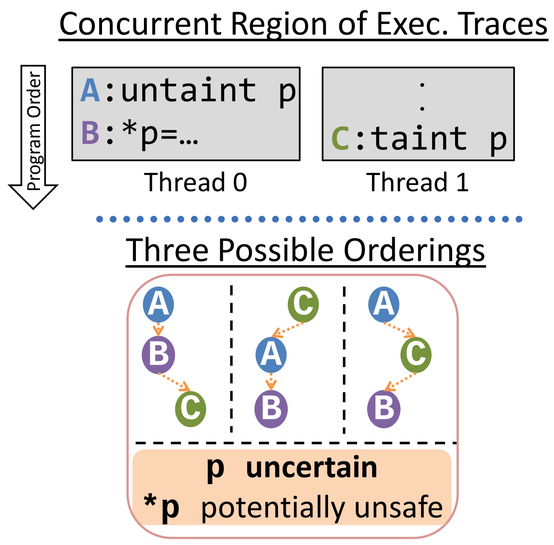


Large precision improvement
- **97%** avg reduction in potential errors (relative to Butterfly)
- **1.9x** slower than Butterfly implementation (on average)

TAINTCHECK: Check if ***p** is trusted (**untaint**) or untrusted (**taint**)
- Safety of ***p** depends on order of ***p** and **[un]taint(p)** operations
- Prevent/detect malicious-input based stack smashing attacks
- Adaptation based on *Reaching Definitions* (dataflow analysis)

TAINTCHECK: [Newsome & Song, NDSS '05]

RECENT WORK: DETECTING NEAR MISSES



Modeling Uncertainty to Improve Precision
- Partition real errors from *potential* errors
- False pos. from failed checks of **uncertain**
- Analysis adapts in response to uncertainty

Not all **uncertain** classifications are genuine "false positives"
- Some are spurious, due to conservative analysis/heuristics
- Others may be **near misses:** no synch prevents future bad ordering

Intel Science & Technology Center for Cloud Computing

Carnegie Mellon University · Georgia Tech · intel · Princeton University · UC Berkeley · University of Washington