BarrierPoint: Sampled Simulation of Multi-Threaded Applications

Trevor E. Carlson, Wim Heirman

Kenzo Van Craeynest and Lieven Eeckhout

Monday, March 24th, 2014

ISPASS 2014 – Monterey, CA, USA

Universiteit Gent

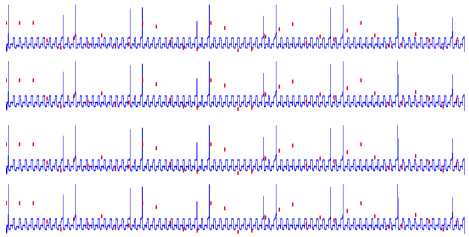
intel

ExaScience Lab
Intel Labs Europe
EXASCALE COMPUTING

DEMANDS ON SIMULATION ARE INCREASING

- Simulation targets are evolving
  - Increasing core counts per processor
  - More complex memory hierarchies
- Traditional cycle-level simulation is single-threaded
  - Single-threaded performance is not improving significantly
- Results in a large simulation gap
- New solutions are needed

Xeon Phi, Source: Intel



Simulation Workload Reduction is Key

- Many reduction techniques exist today
  - Application reduction
    - Smaller input sizes
    - Reduced numbers of iterations
  - Sampling: same workload, but
    - Only part of the workload is simulated in detail
    - Whole-program performance is extrapolated
    - Examples:
      - SimPoint
      - SMARTS/Flex Points
      - Time-based MT-Sampling

MT-Sampling Wish List

- Multi-Threaded SimPoints-like solution
  - Simulation Time = O(# SimPoints) instead of O(# insns)
  - Easy to use, fast to run (in parallel)

- Multi-threaded SimPoints is not a valid solution
  - Operates on average CPI, not application runtime
    - Does not allow for runtime (non-idle + idle) reconstruction
  - What is the starting point of a SimPoint region?
    - Must constitute a valid thread ordering for all architectures

Current Sampling Solution Space

Native Application Simulation 1000 s

Time-based Multi-Threaded Sampling Sim O(# insns) limits speedup 100 s

Workload-specific methodologies < 1 s

SMARTS/Flex Points
Averages User-IPC

BarrierPoint
Slowest thread matters

Throughput (server) workloads: client/server, etc.

Unstructured synchronization mechanisms: Work-stealing, mutexes

Barrier-synchronizing workloads: OpenMP, auto-parallelized

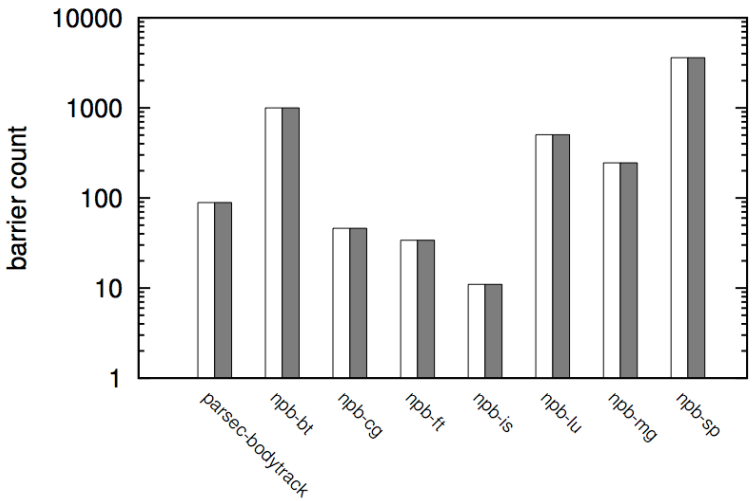
Thread Synchronization Amount

BarrierPoint

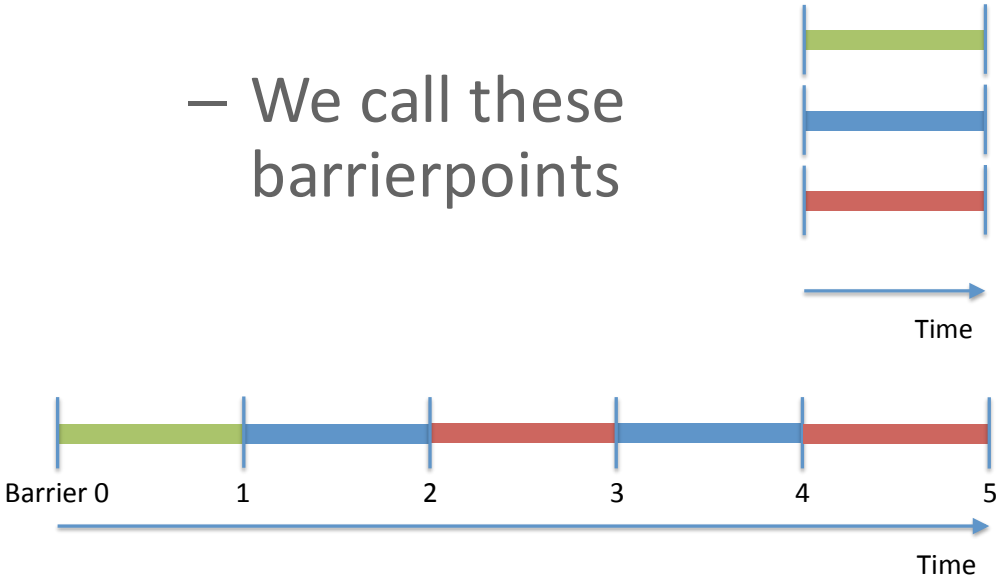
- Key Contributions
  - Micro-architecture independent selection of representative multi-threaded regions
  - Extrapolate and estimate total application runtime
  - Evaluation with realized speedups and errors
  - Propose a straight-forward multi-threaded warmup technique

BARRIERPOINT

- Application Trends
  - Scientific applications use barriers (OpenMP)
  - Auto-parallelization of applications uses fork-join parallelism

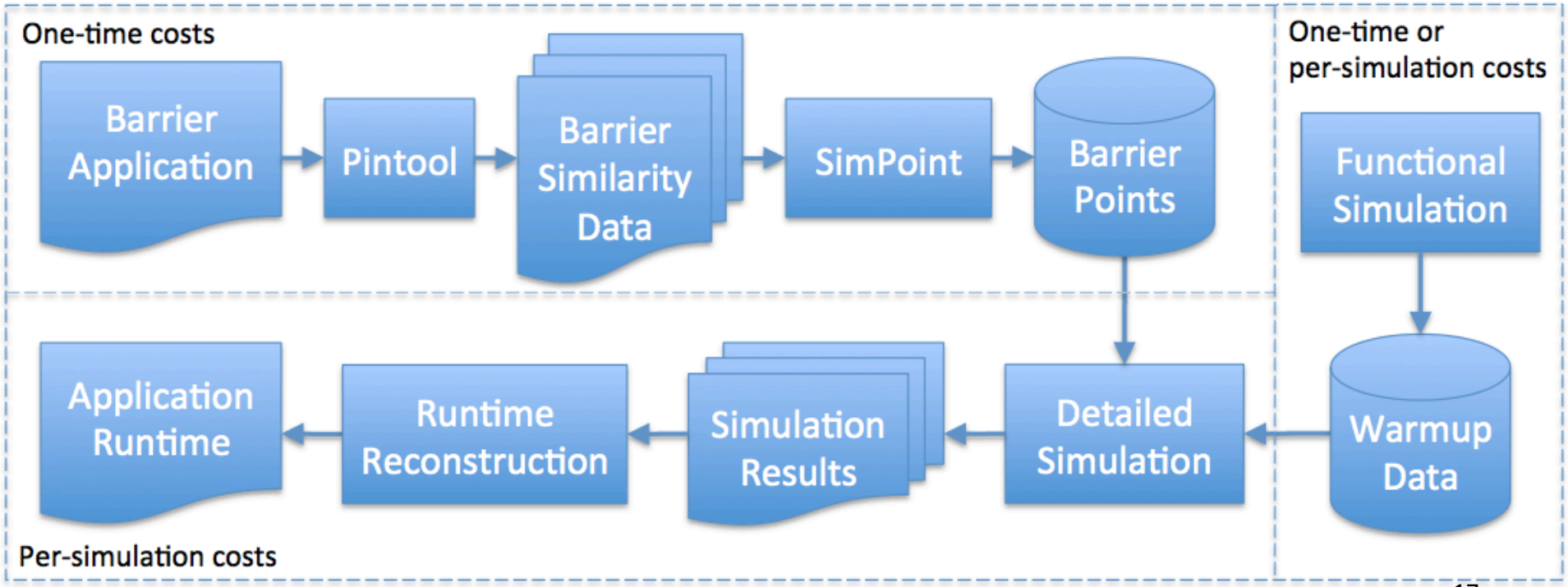


- Main Idea
  - Simulate just the representative regions between barriers (potentially in parallel)
  - We call these barrierpoints



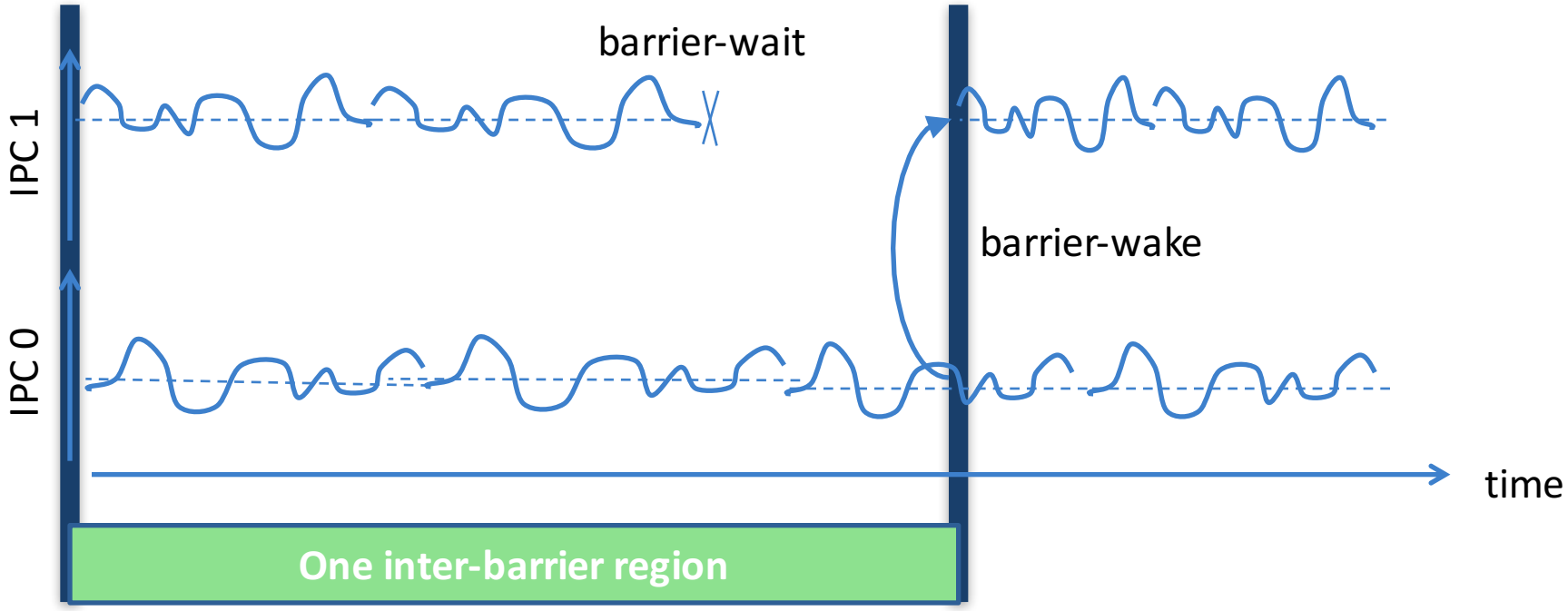
BarrierPoint Methodology

- Compare workloads between barriers for similarity
- Select and simulate the representative barrierpoints
- Reconstruct the runtime from the barrierpoints' results



Multi-threaded Regions

- What is an inter-barrier region?
  - The execution of all threads after a barrier, up to and including the completion of the following barrier



mArch-Independent Region Selection

- Basic-block vectors (BBVs)
  - Application execution fingerprint
  - Captures basic-block execution
- LRU-stack distance vectors (LDVs)
  - Application data access fingerprint
  - Counts the number of *unique* address accesses that occur between two accesses to the same address (at cache line granularity)
- BBVs + LDVs
  - Combine instruction and data fingerprint into a single inter-barrier signature

Unique Address Warmup

- Multi-threaded warmup technique for Barrierpoint
  - Avoid long execution-driven simulation before ROI
  - Warmup data part of checkpoint, relatively µarchitecture independent
  - Ensure cache coherency
- Unique Address Warmup
  - Similar to MTR[1], but avoids cache-specific reconstruction
  - Collect, from program start up to barrierpoint start
    - Each core records the most recent read, write and instruction cache accesses (by cache line)
    - We collect (M * (last N cache lines)), where N is the number required to fill up the entire cache hierarchy, and M is the number of threads
  - Replay:
    - Issue per-core list of unique addresses in parallel
    - Feed into real cache models, which remain coherent during warmup

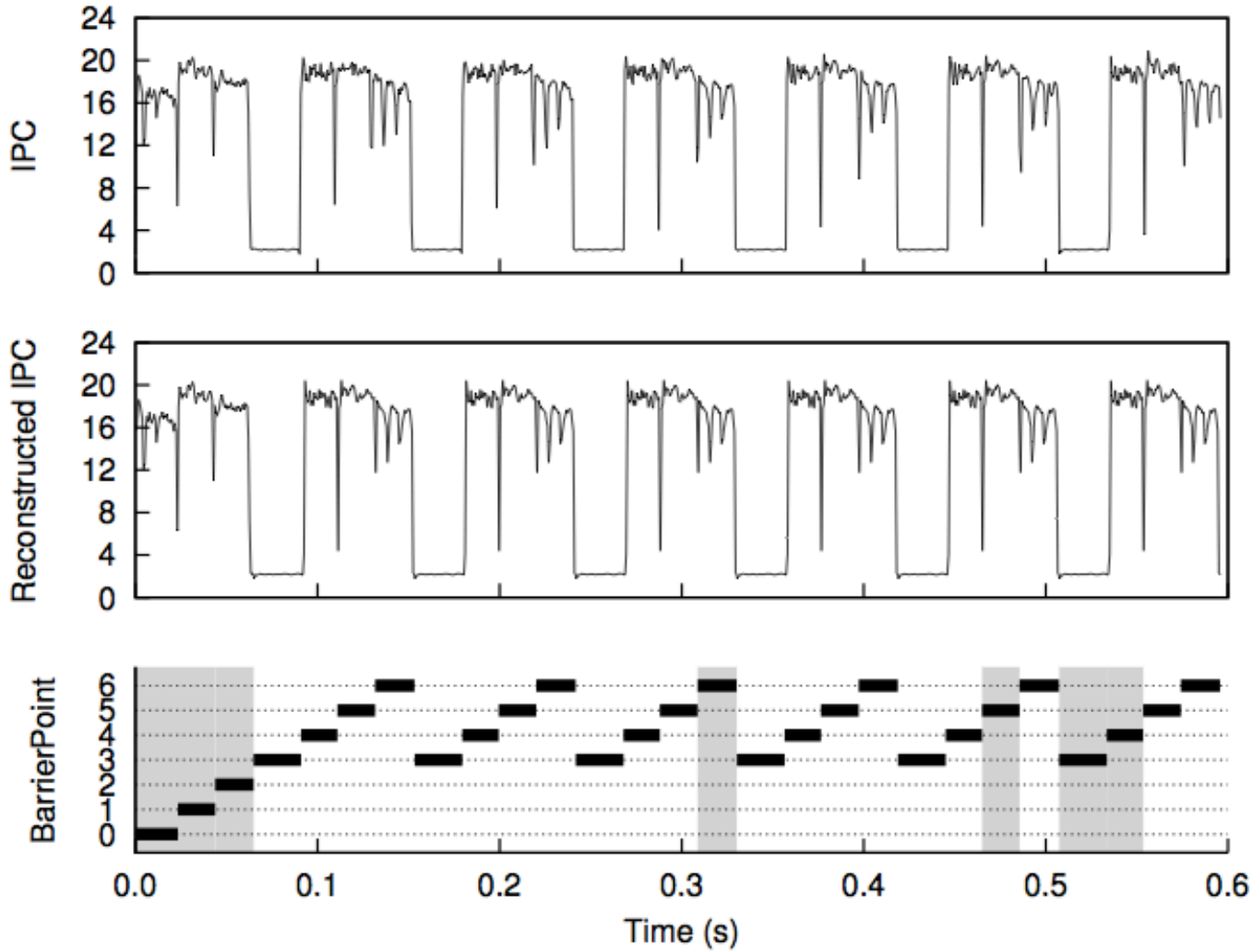
[1] K. C. Barr, et al., "Accelerating Multiprocessor Simulation with a Memory Timestamp Record," in ISPASS 2005

Reconstructing Program Metrics

- Each barrierpoint is given a weight
  - The number of times that it occurs in the run
- With the list and weights, we can reconstruct the runtime
  - $\text{Runtime}_{cg/A/8} = \text{Runtime}_{cg/A/8}(bp_0) * 1.0$
    $+ \text{Runtime}_{cg/A/8}(bp_{15}) * 12.0$
    $+ \text{Runtime}_{cg/A/8}(bp_{21}) * 2.0 + \ldots$
- Similar to SimPoint reconstruction, but now with *time* (including idle/sync.) rather than CPI
- Also works for other application metrics: MPKI, etc.

BM/input	cores	barriers	barrierpoints	barrierpoint # and multiplier
npb-cg/A	8	46	5	0 (1.0), 15 (12.0), 21 (2.0), …
npb-mg/A	8	245	8	2 (2.0), 52 (4.6), 57 (9.0), …

Arch-Independent Region Selection



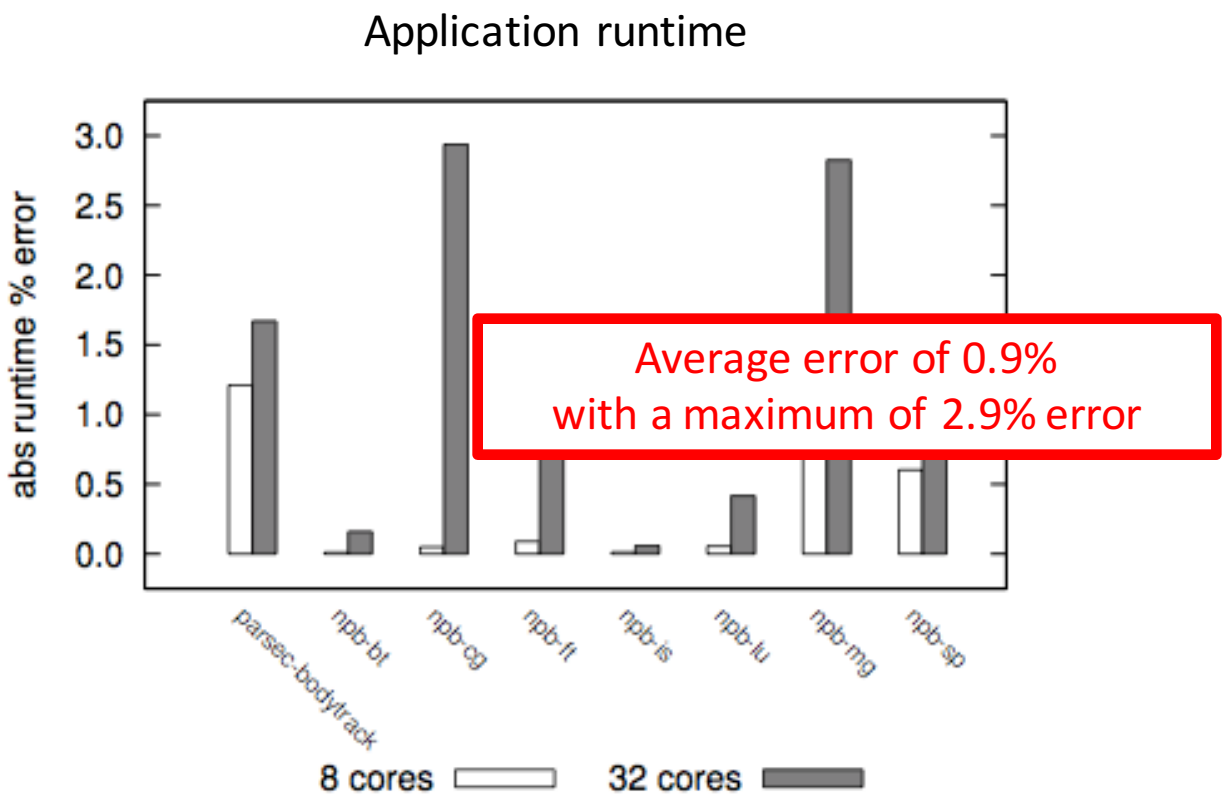
NPB, A input, 32-cores; Aggregate IPCs shown

Experimental Setup

- We model a Xeon/Nehalem-like machine
  - 8-core and 32-core architecture
  - 8-cores share an LLC
- Sniper Multi-Core Simulator
- Benchmarks
  - Most NAS Parallel Benchmarks (NPB)
    - A inputs
  - Parsec
    - Bodytrack Large
- Implemented for OpenMP applications
  - Fork/join parallelism, one barrier per #omp parallel for
  - Can be extended to other types of global synchronization, e.g.
    - pthread_barrier()
    - MPI_(All)Reduce(MPI_COMM_WORLD)

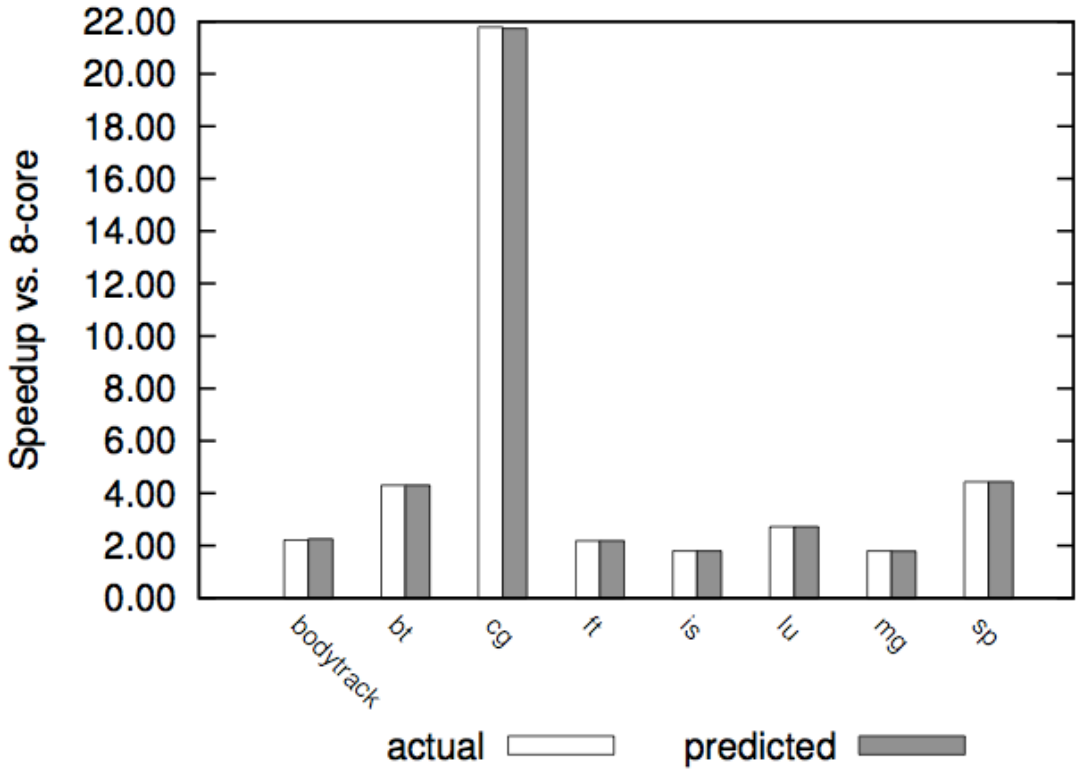
RESULTS

- BarrierPoint shows accurate absolute results



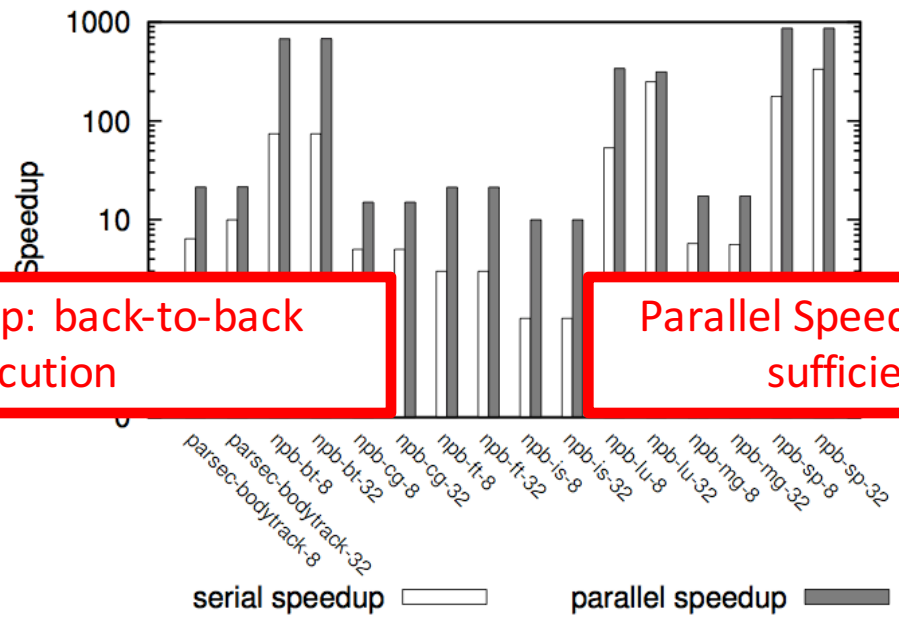
RESULTS

- BarrierPoint shows accurate absolute results and relative scaling results



RESULTS

- Realized simulation speedups are good
  - Resource utilization reduction (improved throughput) by 78x
  - Speedup of 25x on average, 867x maximum



Serial Speedup: back-to-back execution

Parallel Speedup: execution with sufficient resources

Additional Results

- Barrierpoints are a common unit of work across architecture configurations
  - 8-core vs. 32-core
  - Allows for a single characterization run

- Fingerprinting across both instruction and data profiles provide the best results
  - Equal combination of BBVs and LDVs

BARRIERPOINT

- Key Contributions
  - Micro-architecture independent selection of representative multi-threaded regions
    - Explore alternatives to BBVs, such as LRU-stack distances
    - Extrapolate and estimate total application runtime
  - Evaluation
    - Average reduction of machine resources of 78x
    - Realized an average speedup of 25x and maximum of 867x
    - Average error of 0.9%, maximum of 2.9%
  - Propose a straight-forward multi-threaded warmup technique
  - Technology Preview to be released soon
    - http://snipersim.org

BarrierPoint: Sampled Simulation of Multi-Threaded Applications

Trevor E. Carlson, Wim Heirman

Kenzo Van Craeynest and Lieven Eeckhout

Monday, March 24th, 2014

ISPASS 2014 – Monterey, CA, USA

Universiteit Gent

intel

ExaScience Lab
Intel Labs Europe
EXASCALE COMPUTING