Automatic SMT Threading for OpenMP Applications on the Intel Xeon Phi Co-processor

Wim Heirman[1,2] Trevor E. Carlson[1] Kenzo Van Craeynest[1]
Ibrahim Hur[2] Aamer Jaleel[2] Lieven Eeckhout[1]

[1] Ghent University
[2] Intel Corporation

ROSS 2014, Munich

Intel Exascale Labs — Europe

Strong Commitment To Advance Computing Leading Edge:

Intel collaborating with HPC community & European researchers
4 labs in Europe - Exascale computing is the central topic

ExaScale Computing Research Lab, Paris

Performance and scalability of Exascale applications

Tools for performance characterization

ExaCluster Lab, Jülich

Exascale cluster scalability and reliability

ExaScience Lab, Leuven

Life Science applications

Architectural simulation

Scalable kernels and RT

Intel and BSC Exascale Lab, Barcelona

Scalable RTS and tools

New algorithms

EXASCIENCE LIFE LAB

High Performance Computing in Life Sciences

WU0
Management , Business Models & Infrastructure

janssen
intel
imec

WU1
High Performance Biostatistics Applications

janssen
universiteit hasselt
KU LEUVEN
Universiteit Antwerpen

WU2
Platforms and Flexible Performance for DNA/RNA Sequencing, Visualization and Analysis

imec
janssen
KU LEUVEN
Universiteit Antwerpen
UNIVERSITEIT GENT

WU3
Data Intensive Parallel Programming Models

imec
intel
Vrije Universiteit Brussel
universiteit hasselt

WU4
Architectural Simulation

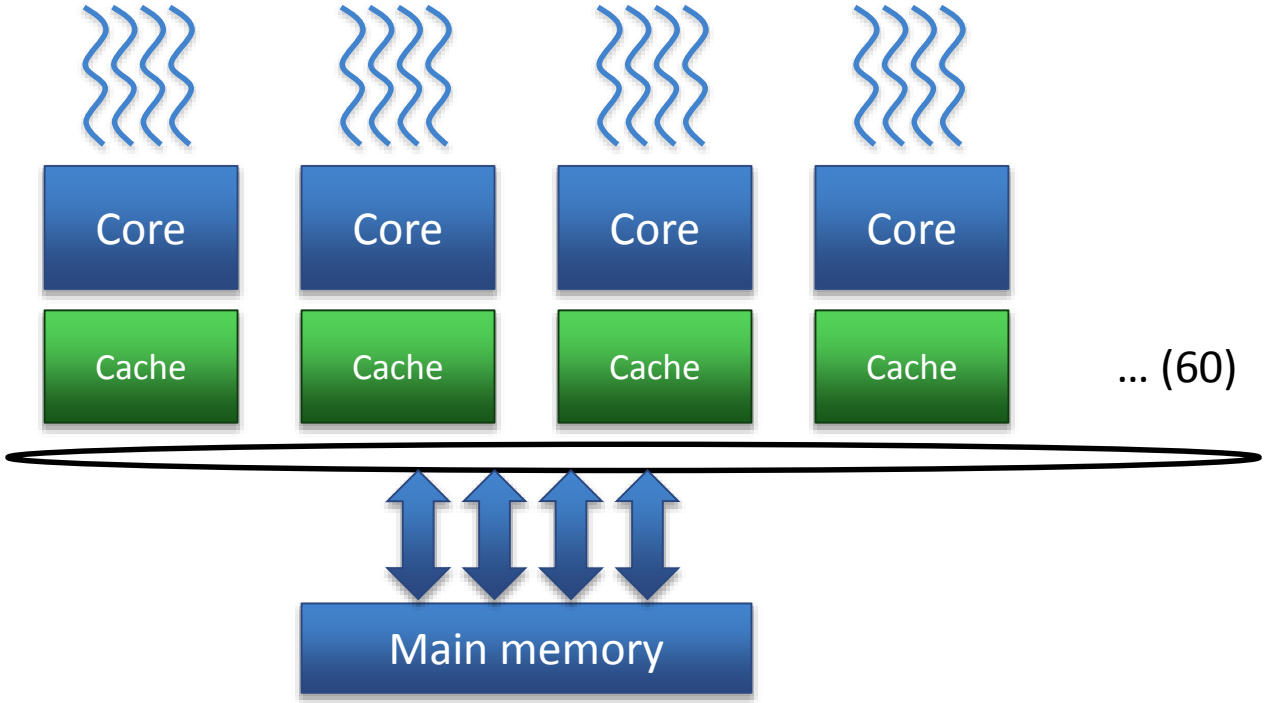
UNIVERSITEIT GENT
intel

Overview

- Introduction
  - Intel Xeon Phi architecture
- Motivation
  - Effects of per-core thread count on performance
- Dynamic threading
  - Algorithm and implementation of automatic thread count adaptation
- Results & conclusion

Xeon Phi Architecture

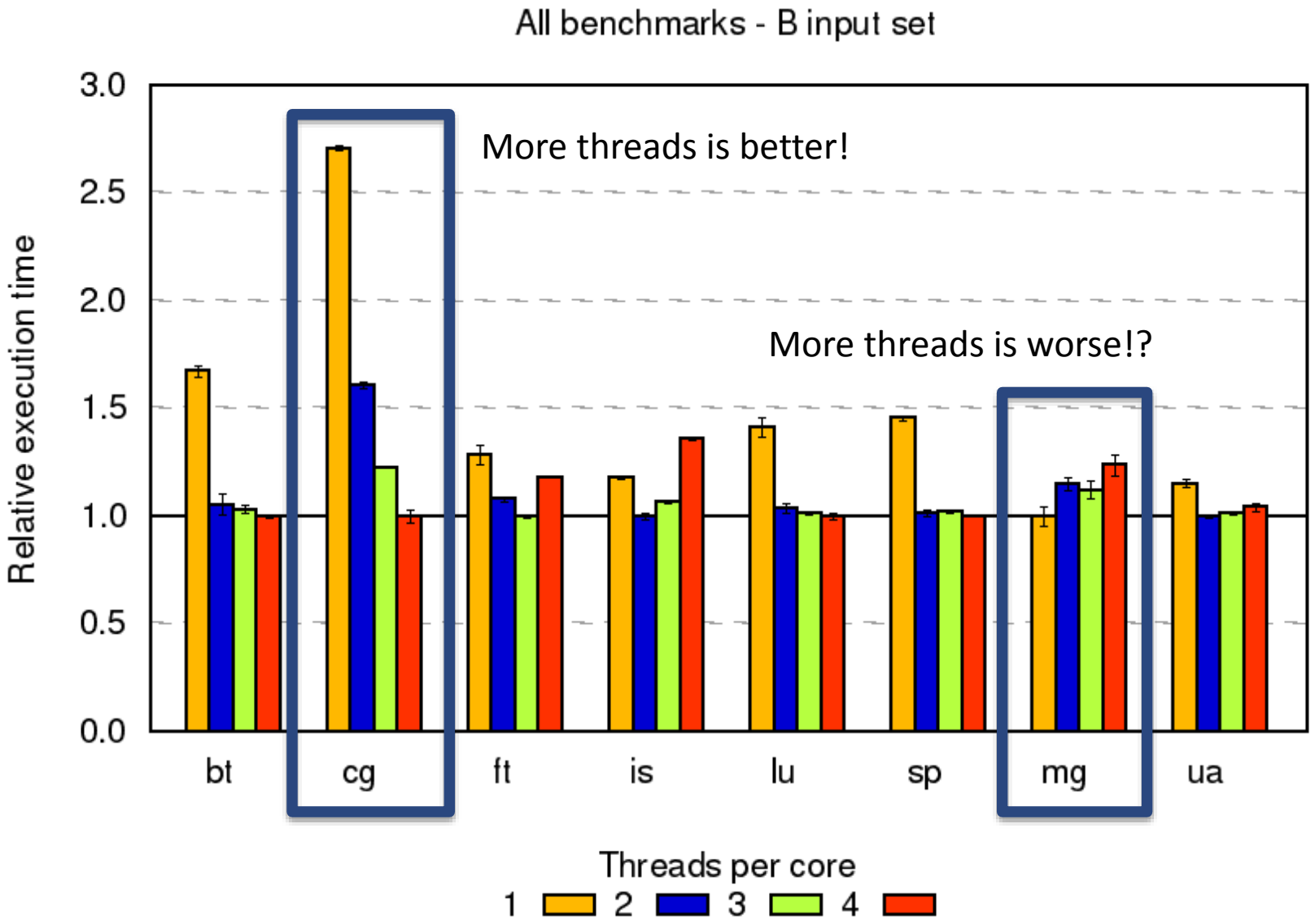
- Up to 4 SMT threads per core
- ~60 cores
- Per-core private L1+L2 caches, coherent
- Ring-based interconnect



How many SMT threads? Varies!

- Core:
  - at least two to keep pipeline fully occupied
  - more threads can overlap more latency
- Cache:
  - shared L1-I/D and L2, TLBs
  - threads can evict each other's data
- Memory:
  - more threads keeps more requests outstanding
  - no more gains once bandwidth is saturated

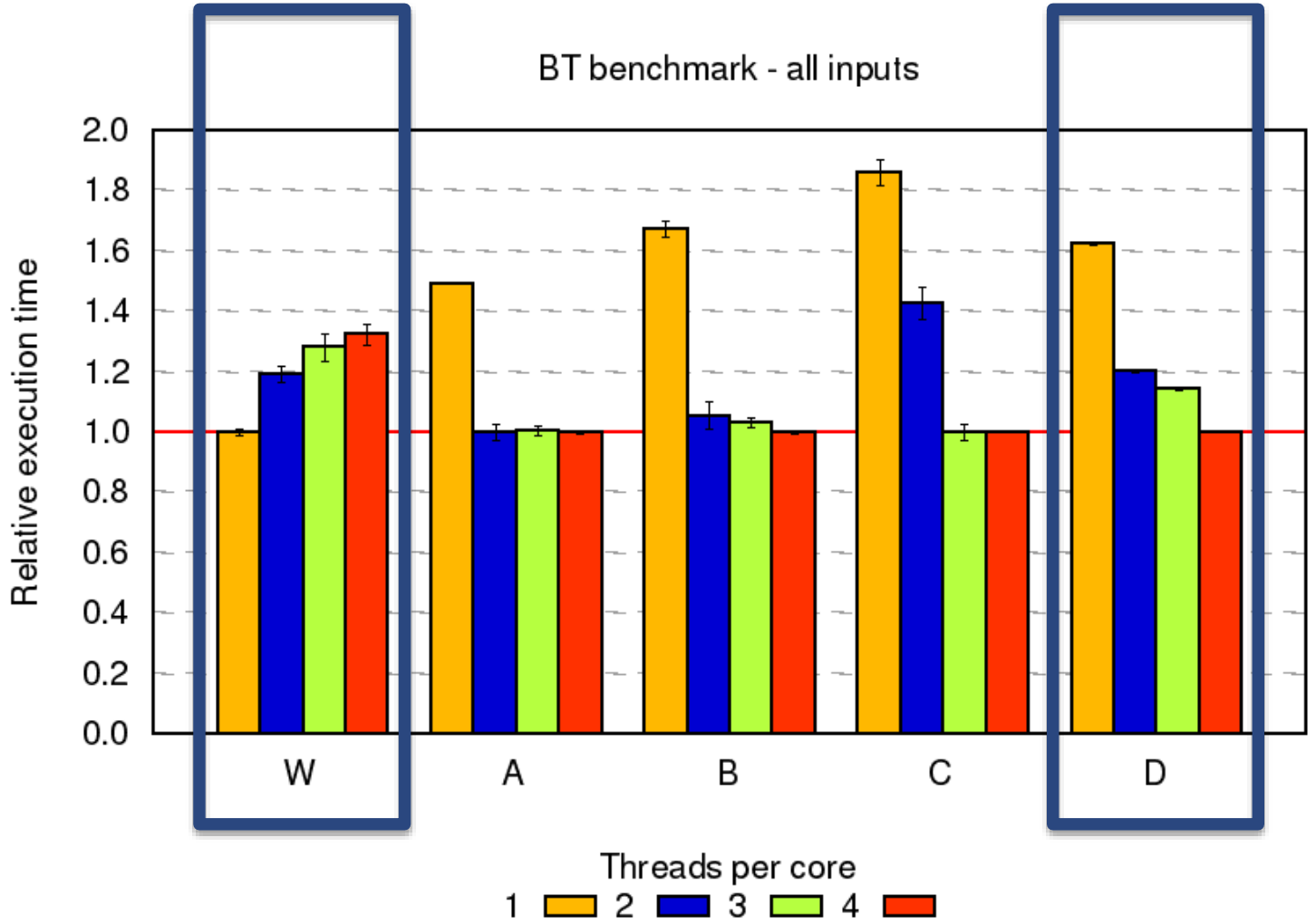
THREAD COUNT ACROSS APPLICATIONS



THREAD COUNT ACROSS INPUT SETS (BT)

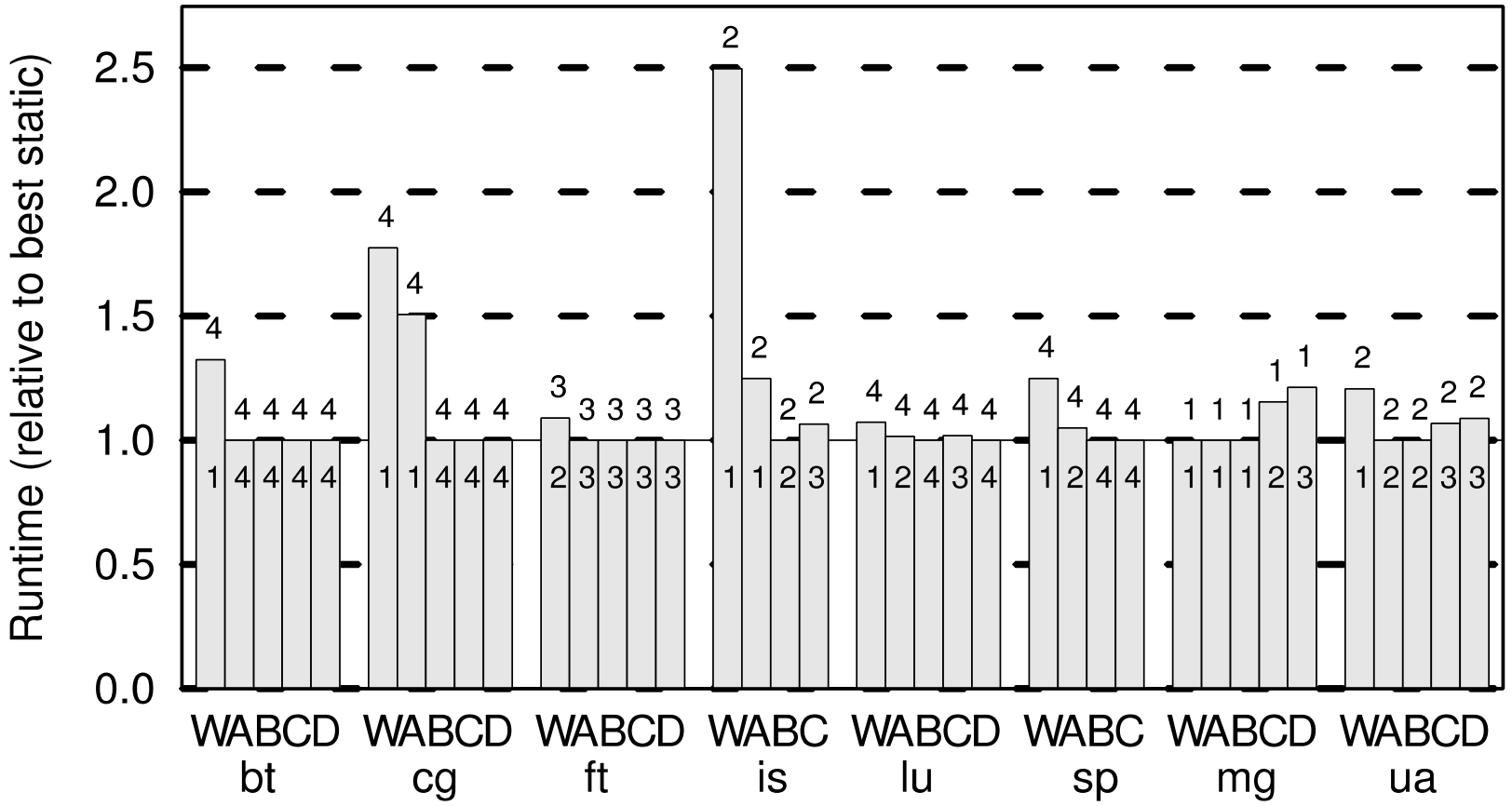
Small input set:
cache-fitting at low thread counts,
cache-thrashing at high thread counts

Large input set: many threads
to overlap memory latency



THREAD COUNT ACROSS INPUT SETS

- Best vs. per-application setting based on B



CLUSTER-AWARE UNDERSUBSCRIBED SCHEDULING OF THREADS (CRUST)

- Move decision from programmer to runtime
  - Integrated into the OpenMP runtime library
  - No application changes
- Dynamic undersubscription
  - Automatically find best setting per workload, input set, during the application
  - Exploit iterations / time steps in the workloads
- Adapt to each *#pragma omp parallel* individually
  - Recognizes workload phase behavior

PER-SECTION TUNING

- Calibration phase: for each section occurrence,
  - Measure hardware performance counters
    - Clock cycles, instruction count, L2 misses
    - More would have been nice but not supported by KNC
    - Use user-mode rdpmc for low overhead
  - Try all 4 thread counts in subsequent occurrences
    - Pick best cycle count
    - Instruction count not stable: spin loops
- Stable phase:
  - Keep watching L2 miss rate, recalibrate on change
  - Signifies data-dependent behavior

DYNAMIC AGGREGATION

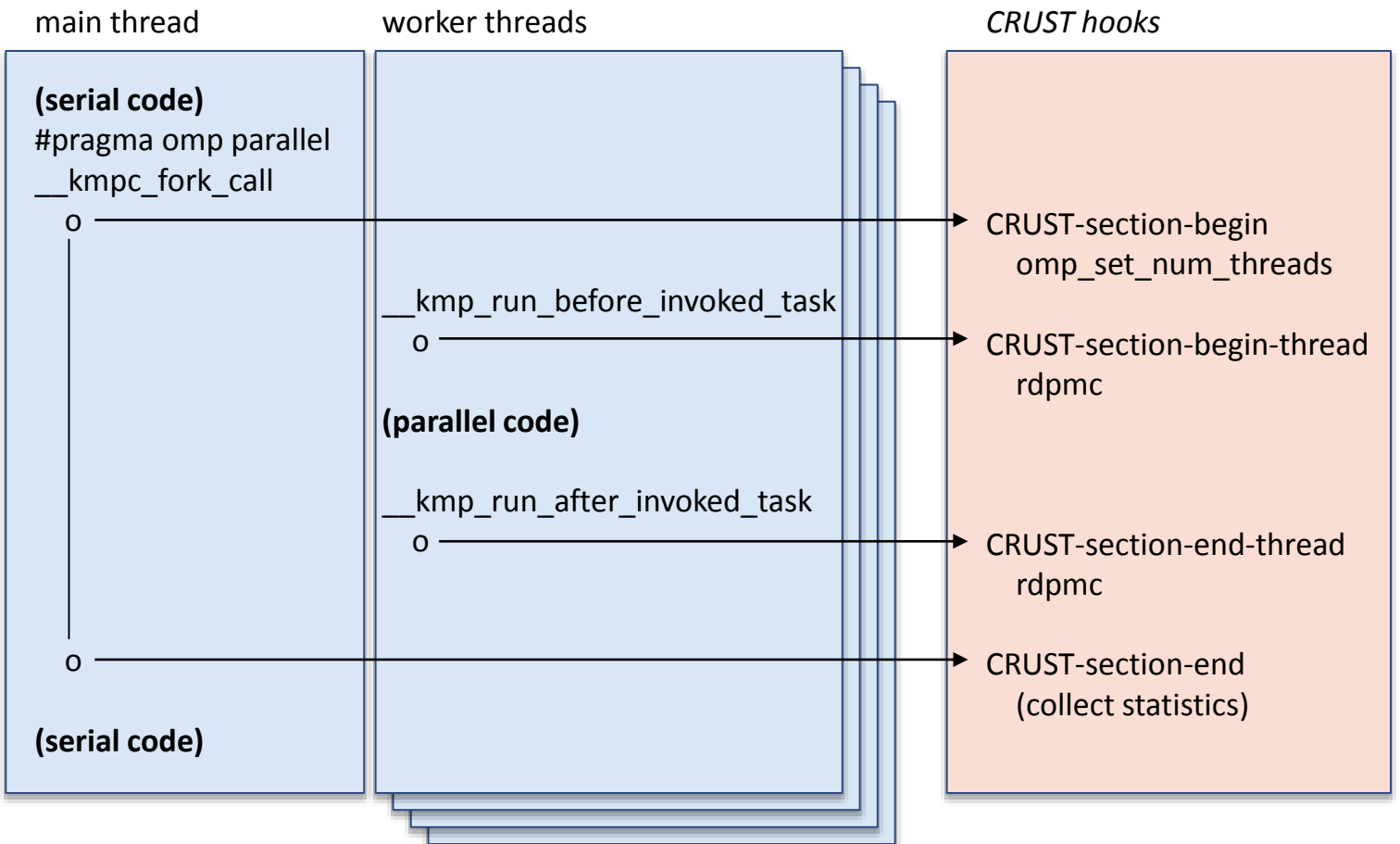
- Large sections: data reuse inside phase
- Small sections: data reuse *across* phases
  - No good to keep changing thread count / data partitioning
- Dynamically aggregate sections
  - Up to 50 million clock cycles (~50 ms)
  - “Aggregate” forms one new section id
  - Using IPC rather than runtime

EFFICIENT PMU COLLECTION

- Use Linux perf interface for counter setup
- Reading counters:
  - read system call
  - lots of inter-processor interrupts
  - many million clock cycles overhead
- Alternative:
  - user-mode rdpmc instruction, reads local counter
  - have each OpenMP worker thread read its own counters, communicate results through shared memory

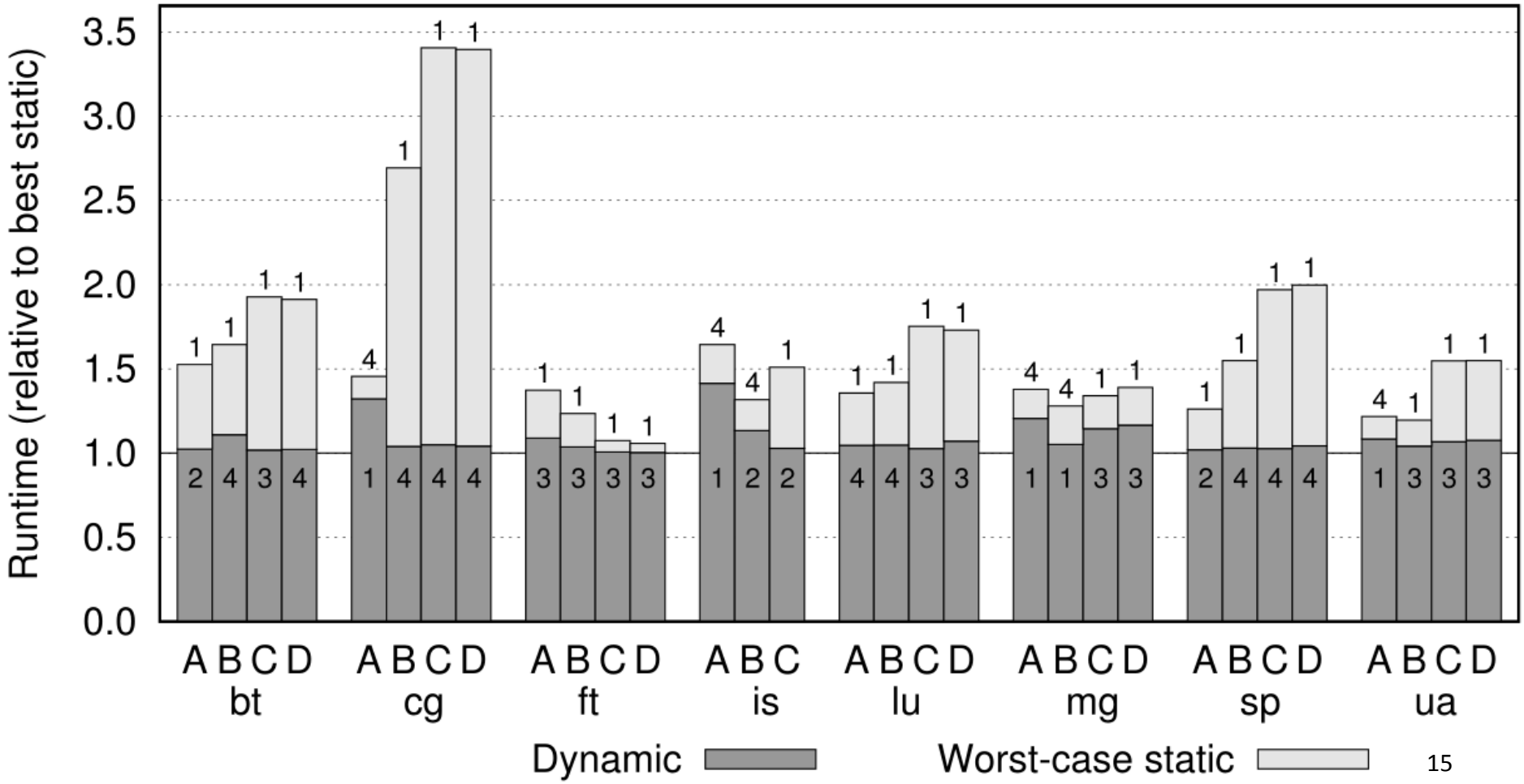
IMPLEMENTATION

- Open-source Intel OpenMP library at http://www.openmprtl.org
- Added 4 hooks, recompiled into new libiomp5.so (no application changes)
- Dynamic threading functionality in separate libcrust.so
- Easily portable to other parallel runtimes (TBB, Cilk, …)



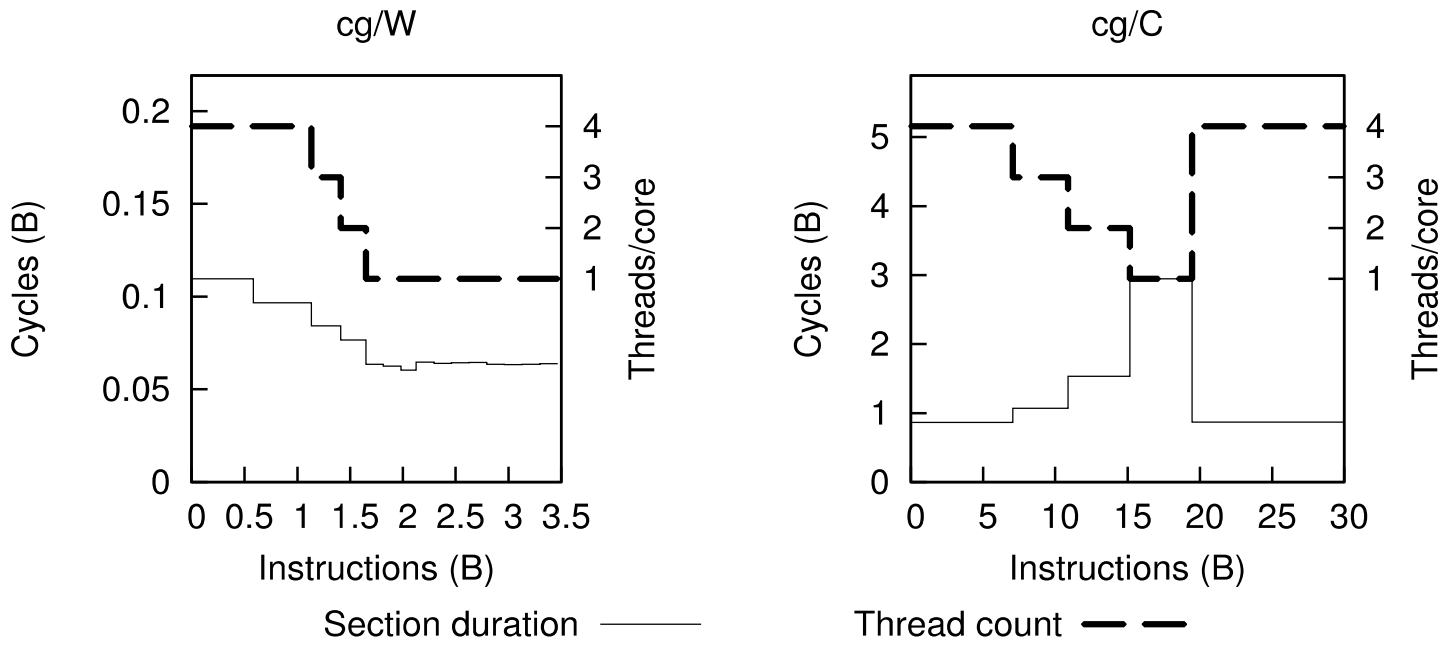
RESULTS

- Able to find best thread count for all applications
- Benchmarks too simple to benefit from per-region tuning



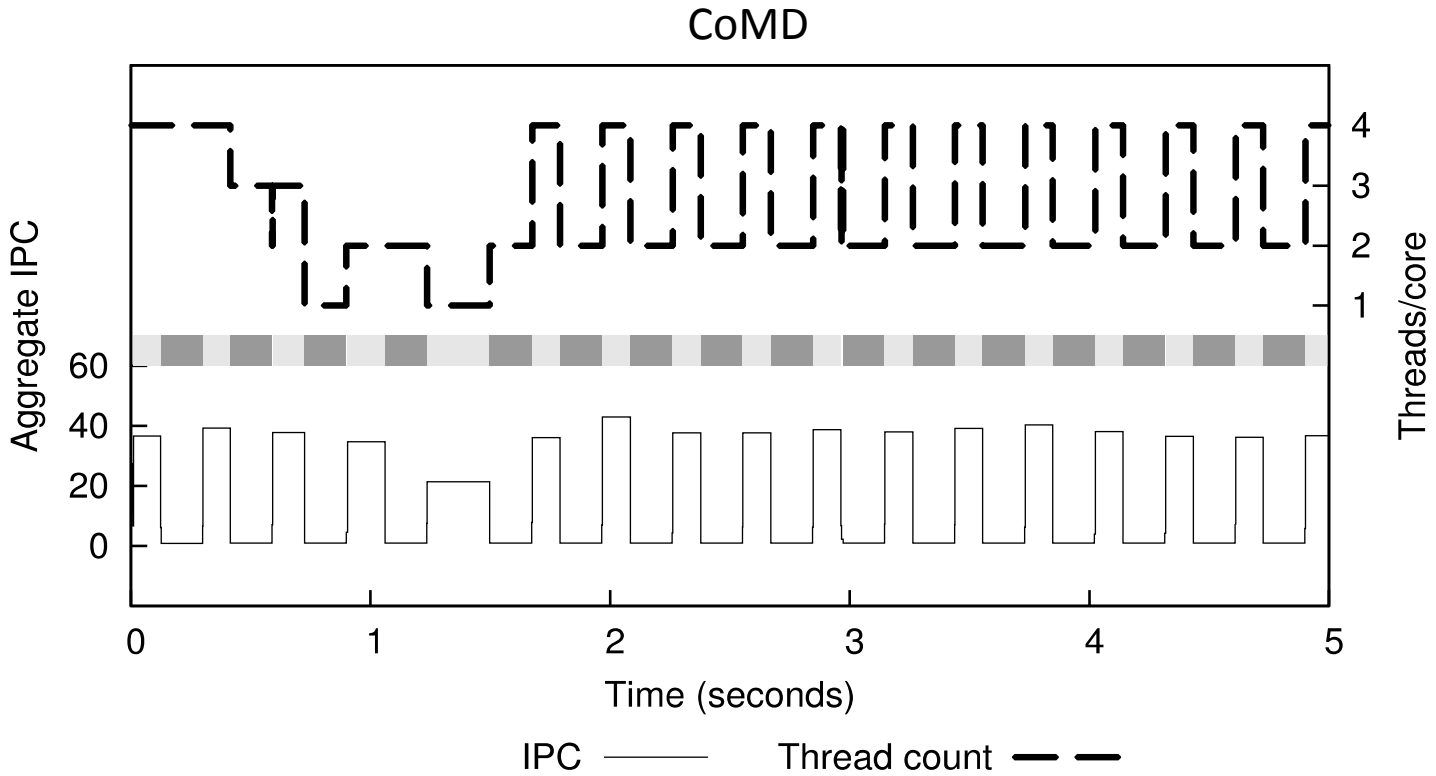
THROUGH-TIME BEHAVIOR

- Threads per core (top) and performance (cycles/section, bottom) through time
- Calibration phase (1st half), stable phase (2nd half)



COMPLEX APPLICATIONS

- Multiple distinct phases:
  - Light: compute bound, high thread count
  - Dark: memory bound, low thread count
- 2% speedup over best-static



Conclusions

- Per-core thread count on Xeon Phi affects performance in multiple ways
  - Core, cache, main memory effects
- Best thread count varies
  - Across applications, input sets, workload phases
- Dynamically determining best thread count
  - Per application phase (#pragma omp parallel)
  - Aggregate small phases
  - Use performance counters for more insight
- Prototyped in Intel OpenMP runtime library