Performance Evaluation of NAS Parallel Benchmarks on Intel Xeon Phi

ICPP
6th International Workshop on Parallel Programming Models and Systems Software for High-End Computing

October 1, 2013
Lyon, France

Vienne, Ramachandran, Wijngaart, Koesterke, Shaparov

THE UNIVERSITY OF TEXAS AT AUSTIN
TEXAS ADVANCED COMPUTING CENTER

Increasingly Complex HPC World

- Mainframes
  - HPC language (Fortran, then also C) + vectorization (no caches!)
- Clusters: MPI
- SMP machines (shared memory): OpenMP + threads
- Clusters with multi-core nodes: MPI + OpenMP
  - Vectorization + NUMA
- Clusters with accelerators
  - Fast compute, high memory BW, limited memory
  - Data transfer through PCIe bus → Double buffering
  - GPUs: CUDA/OpenCL (partially mastered)
  - MICs: Standard languages + OpenMP
- Evaluation of Phis through 'standard' benchmarks

TACC

How do we evaluate new technologies?

Programming and optimizing for MIC?

How hard can it be?
What performance can I expect?

The NAS Parallel Benchmark (NPB) gives very valuable insight

TACC

Programming and Optimizing for the MIC Architecture

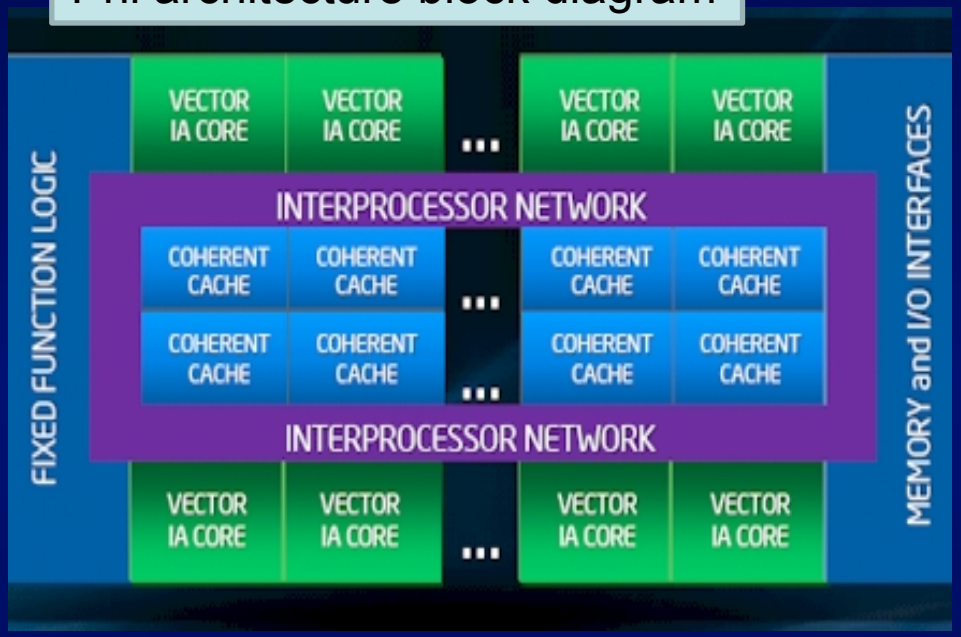
- Intel's® MIC is based on x86 technology
  - x86 cores w/ caches and cache coherency
  - SIMD instruction set
- Programming for MIC is similar to programming for CPUs
  - Familiar languages: C/C++ and Fortran
  - Familiar parallel programming models: OpenMP & MPI
  - MPI on host and on the coprocessor
  - Any code can run on MIC, not just kernels
- Optimizing for MIC is similar to optimizing for CPUs
  - Make use of existing knowledge!

TACC

MIC Architecture

- Many cores on the die
- L1 and L2 cache
- Bidirectional ring network
- Memory and PCIe connection

Phi architecture block diagram



Xeon Phi

- ~60 cores
- Few GB of GDDR5 RAM
- 512-bit wide SIMD registers
- L1/L2 caches
- Multiple threads (up to 4) per core

Xeon Phi in Stampede

- SE10P
- 61 cores
- 8 GB of GDDR5 memory

MIC is similar to Xeon

- x86 cores with caches
- Cache coherency protocol
- Supports ‘traditional’ threads

TACC

MIC vs. GPU

- Differences
  - Architecture: x86 vs. streaming processors

    coherent caches vs. shared memory and caches
  - HPC Programming model:

    extension to C++/C/Fortran vs. CUDA/OpenCL

    OpenCL support

  Threading/MPI:

  OpenMP and Multithreading vs. threads in hardware

  MPI on host <u>and/or</u> MIC vs. MPI on host only
  - Programming details

    offloaded regions vs. kernels
  - Support for any code: serial, scripting, etc.

    Yes No
- Native mode: Any code may be "offloaded" as a whole to the coprocessor

TACC

Adapting Scientific Code to MIC

- Today: Most scientific code for clusters
  - Languages: C/C++ and/or Fortran,
  - Communication: MPI
  - may be thread-based (Hybrid code: MPI & OpenMP),
  - may use external libraries (MKL, FFTW, etc.).
- With MIC on Stampede:
  - Languages: C/C++ and/or Fortran,
  - Communication: MPI
  - may run an MPI task on the MIC
    or may offload sections of the code to the MIC,
  - will be thread-based (Hybrid code: MPI & OpenMP),
  - may use external libraries (MKL),
    that automatically use MIC

TACC

Stampede Cluster at the

Texas Advanced Computing Center

What do we do with 6800 MIC cards?

How to program and optimize for MIC?

TACC

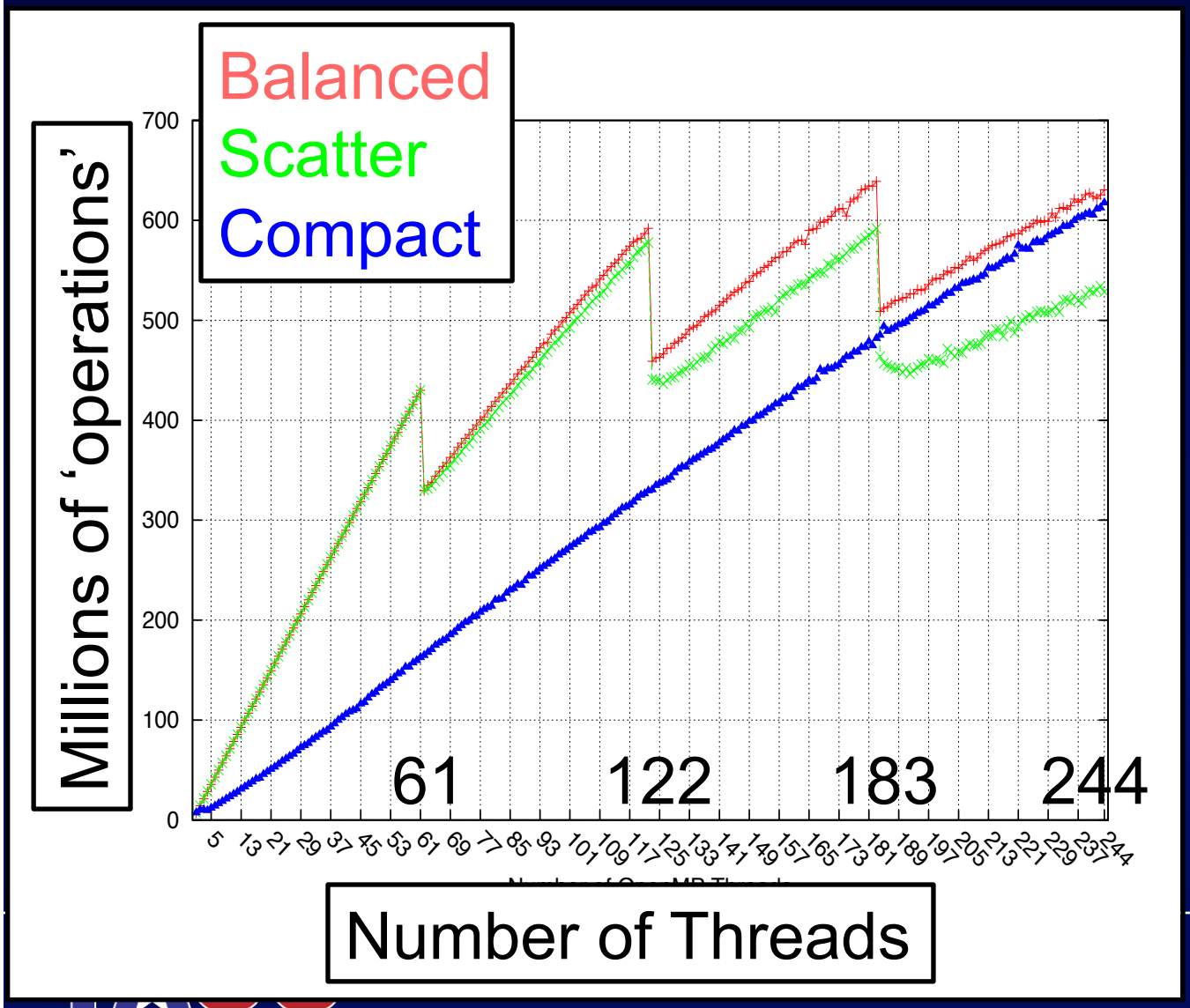
NAS Parallel Benchmark

- Suite of parallel workloads
- Testing performance of a variety of components
- Computational kernels
  - IS: Integer sorting
  - FT: Fourier transform
  - CG: Conjugate gradient
  - MG: Multi-grid
- Mini applications
  - BT & SP: Factorization techniques
  - LU: LU decomposition
- Variety of sizes: class A, B, and C used
- Parallelized with OpenMP & MPI

Evaluation

- Timings → Flops
- Ratio of active vector lanes to number of vector instructions
  - VPU_ELEMENTS_ACTIVE
  - VPU_INSTRUCTIONS_EXECUTED
  - Vector width: 8/16 (DP/SP)
- Reading from caches
  - DATA_READ_OR_WRITE
- Missing caches
  - DATA_READ_MISS_OR_WRITE
- TLB misses
  - DATA_PAGE_WALK, LONG_DATA_PAGE_WALK

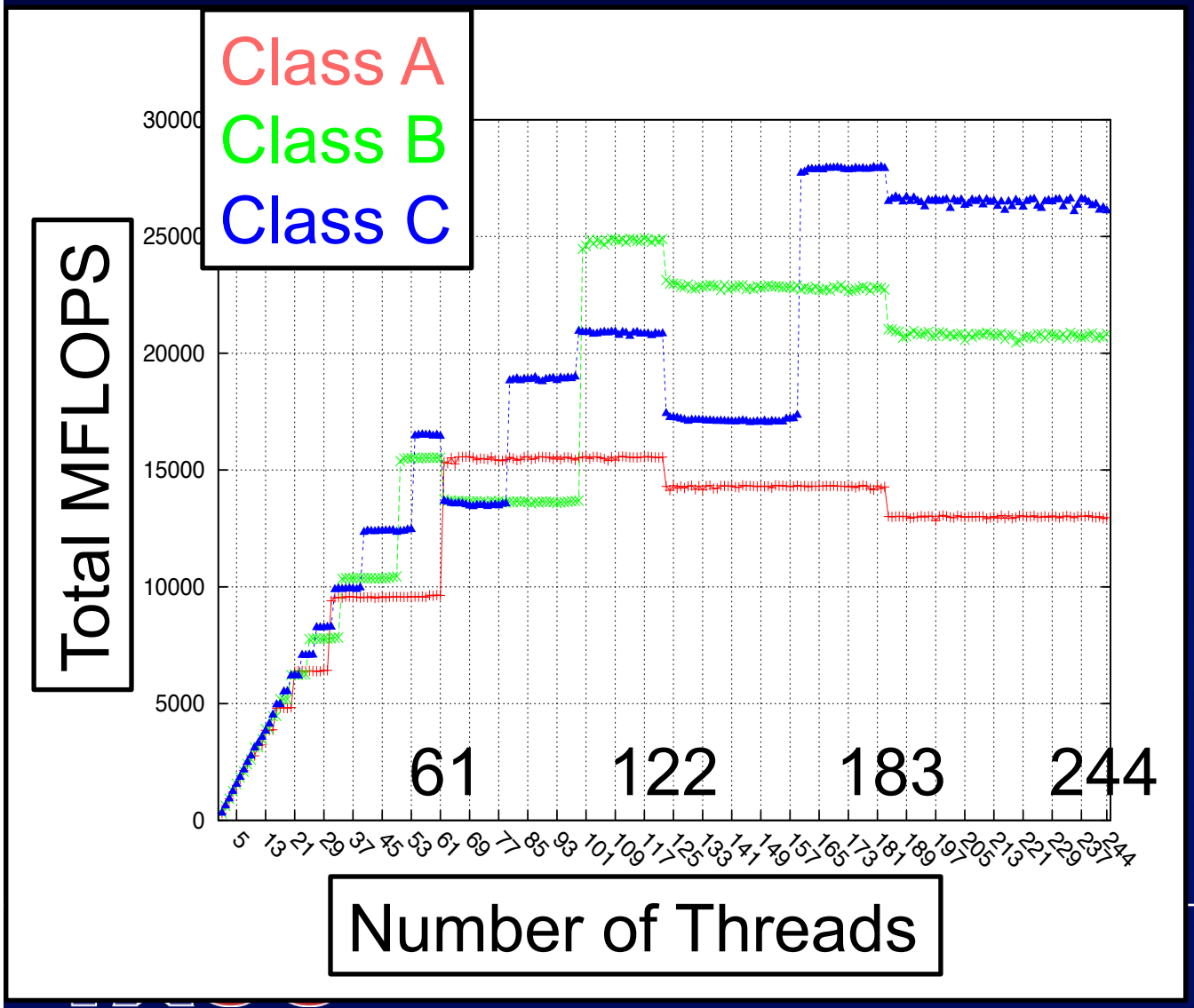
Affinity and Scaling: IS



IS, class C

Balanced provides best performance

Performance: BT

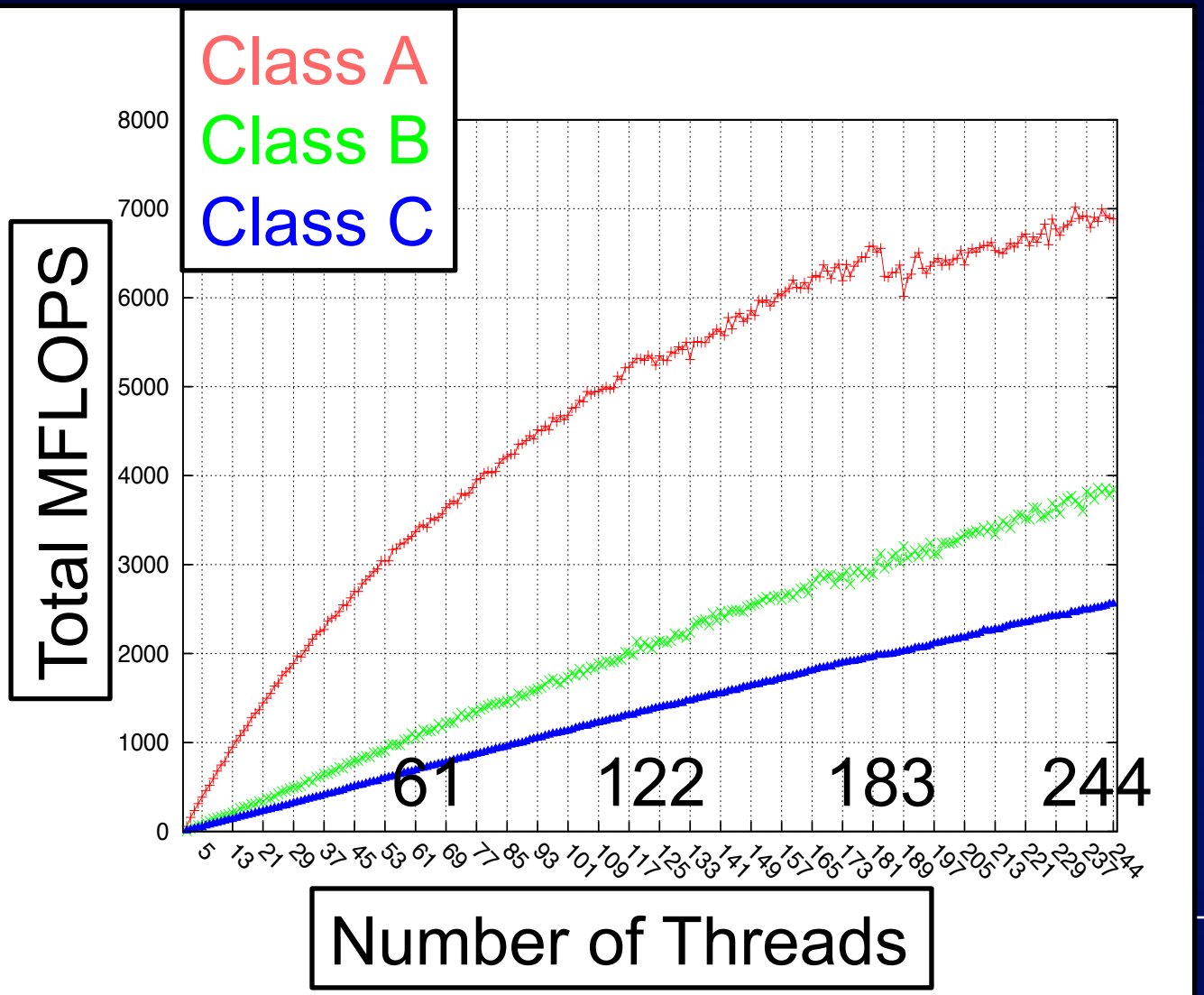


BT, three sizes

Best number of threads per core derived by trial

Jumps at 'natural' boundaries and in between boundaries

Performance: CG



CG, three sizes

Best performance at maximum number of threads

Xeon vs. Phi: 16 vs 61 cores

- 16 Sandy-Bridge cores are up to 2.5x faster
- Best Phi performance to break even

Class C	SB: time [s]	SB: threads	Phi: time [s]	Phi: threads
BT	69	16	100	162
CG	22	32	57	241
FT	16	32	42	105
IS	1	32	2.4	177
LU	49	32	110	162
MG	7	16	6	172
SP	115	16	135	162

TACC

Analysis: Loop Iterations

- Phi needs more loop iterations (concurrency, ‘n’) for good performance
  - Either ‘n’ is a multiple of 61 or 122
  - Or ‘n’ is much larger than 122
- Technique: loop collapse
  - BT, class A: 2x performance increase
  - BT, other classes: no gain; Larger sizes provide already enough concurrency

Analysis: Division and Square Root

- Divisions and square roots are ‘extra slow’ on Phi
- Division consumes 25% of execution time in BT
- Similar for square root in SP

Analysis: Strided Access

- Strided access in BT (stride = 5)
- Rearrangement of loop iterations did not increase speed, because other loops changed from stride-1 to stride-5

Analysis: Strided Access and Vectorization

- Non-unit stride access in LU
- Loop vectorized by autovectorizer
- Non-vectorized loop (-no-vec) faster
- Gather/scatter load and stores
- Reduced hardware prefetching

Analysis: Gather/Scatter vs. Masked Load/Store

- Access for low strides (stride-2) through 2 masked load/stores, instead of a gather/scatter
- Test code revealed a 1.8x speed-up for vectorized code
- Hand-coded intrinsics

Analysis: Vectorization (general)

- Some benchmarks (particularly MG) showed good performance gain from vectorizations
  - MG faster on Phi than on Xeon
- Most benchmarks performed better without vectorization
  - Phi slower than Xeon

Analysis: Indirect Memory Access

- Memory latency (main GDDR5) higher on Phi than on Xeon (main DDR3)
- Decreased performance for CG and IS

Summary

- NBP gives very valuable insight
- For code to perform well on Phi:
  - More concurrency required (more loop iterations)
  - Vectorization absolutely crucial
  - But not all vectorized code performs well
  - Stride-1 data access
  - No indirect data access
  - Manual hardware prefetching may be required
  - No ‘slow’ operations (div and sqrt)
- Only certain codes will benefit from this generation of Xeon Phi

Thank You!

Questions?

TACC