Advances in HPC for Oil & Gas: The Intel® MIC (Many Integrated Core) Architecture

Oil & Gas High Performance Workshop

March1, 2012
Rice University
Houston, TX

Lars Koesterke
Yaakoub El Khamra

Texas Advanced Computing Center

TACC

THE UNIVERSITY OF TEXAS AT AUSTIN
TEXAS ADVANCED COMPUTING CENTER

What is MIC?

Why is the MIC technology so exciting, and how do we know?

How to program and optimize for MIC?

MIC Overview

- 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
  - Any code can run on MIC, not just kernels
  - MPI on host and on the coprocessor
  - Use Intel's MKL library on MIC

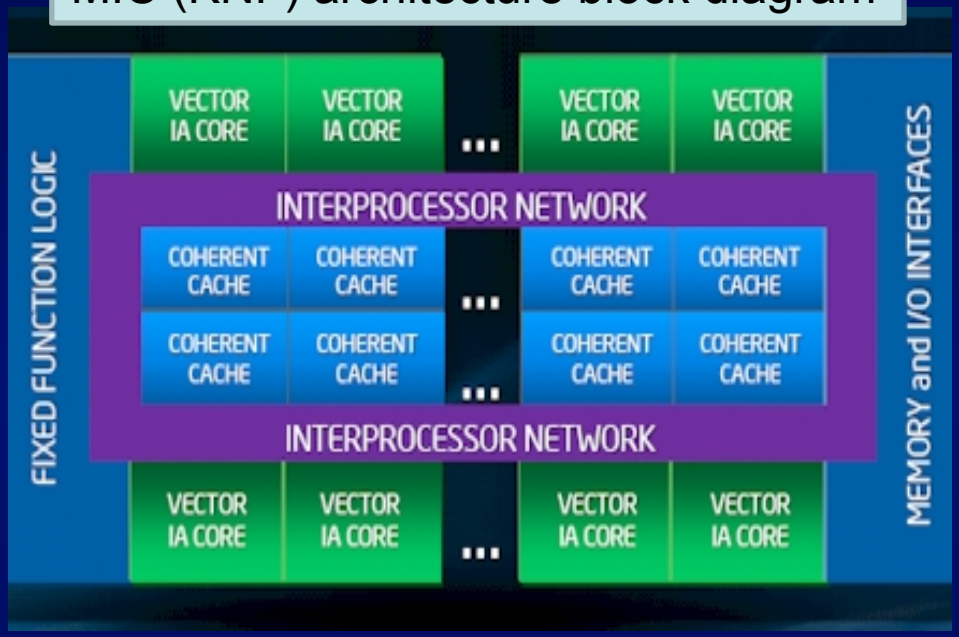
Second Part of the talk

- Optimizing for MIC is similar to optimizing for CPUs
  - Make use of existing knowledge!

MIC Architecture

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

MIC (KNF) architecture block diagram



Knights Ferry SDP

- Up to 32 cores
- 1-2 GB of GDDR5 RAM
- 512-bit wide SIMD registers
- L1/L2 caches
- Multiple threads (up to 4) per core
- Slow operation in double precision

Knights Corner (first product)

- 50+ cores
- Increased amount of RAM
- Details are under NDA
- 22 nm technology

TACC

Coprocessor vs. Accelerator

- GPUs are often called “Accelerators”
- Intel calls MIC a “Coprocessor”

- Similarities
  - Large die device, all silicon dedicated to compute
  - Fast GDDR5 memory
  - Connected through PCIe bus, two physical address spaces
  - Number of MIC cores similar to number of GPU Streaming Multiprocessors
  - Expect Knights Corner’s peak performance to be competitive
    - Bandwidth: host to device
    - Bandwidth: on card
    - Floating point operations (double precision)

Coprocessor vs. Accelerator

- 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 and/or MIC vs. MPI on host only
  - Programming details
    offloaded regions vs. kernels
  - Support for any code: serial, scripting, etc.
    Yes No

Any code may execute on MIC

TACC

The MIC Promise

- Competitive Performance: Peak and Sustained
- Familiar Programming Model
  - HPC: C/C++, Fortran, and Intel's TBB
  - Parallel Programming: OpenMP, MPI, Pthreads, Cilk Plus, OpenCL
  - Intel's MKL library (later: third party libraries)
  - Serial and Scripting, etc. (anything a CPU core can do)
- "Easy" transition for OpenMP code
  - Pragmas/directives added to "offload" OMP parallel regions onto MIC
  - See examples later
- Support for MPI
  1. MPI task on host, communication through offloading
  2. MPI on MIC (`a.out` started on Host and on MIC)
     - All communication through MPI

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 starting 2013:
  - 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

Programming MIC with Threads

Key aspects of the MIC coprocessor

- A lot of local memory, but even more cores
- 100+ threads or tasks on a MIC

One MPI task per core? — Probably not

- Severe limitation of the available memory per task
- Some GB of Memory & 100 tasks
  → some ten's of MB per MPI task

➢ **Threaded Execution (OpenMP, etc.) is essential on MIC**

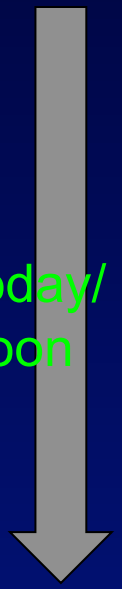
Adapting and Optimizing Code for MIC

- Hybrid Programming
  - MPI + Threads (OpenMP, etc.)

- Optimize for higher thread performance
  - Minimize serial sections and synchronization
- Test different execution models
  - Symmetric vs. Offloaded
- Optimize for L1/L2 caches

- Test performance on MIC
- Optimize for specific architecture
- Start production

Today "Any" resource



Today/ Soon Knights Ferry

Stampede 2013+ Knights Corner

TACC

Stampede is almost here!

TACC — Texas Advanced Computing Center

- World-wide reputation for computational excellence
- Large clusters for compute and visualization
  - Ranger w/ 579 TFlops — Lonestar w/ 302 Tflops
  - Longhorn: 512 GPUs
- Large research projects

Intel® MIC Architecture

- Fascinating technology — **Inviting** programming models
- Tremendous potential for Scientific Computing
- Opens a road to Exascale computing with Intel® Xeon® processor

TACC + Intel + Dell + Academic Partners
Stampede Cluster in Q1 2013

- **10 PFlops, 80% from MIC**

TACC

HPC Applications for Oil & Gas

Early Experiences with the MIC Architecture

In A Nutshell

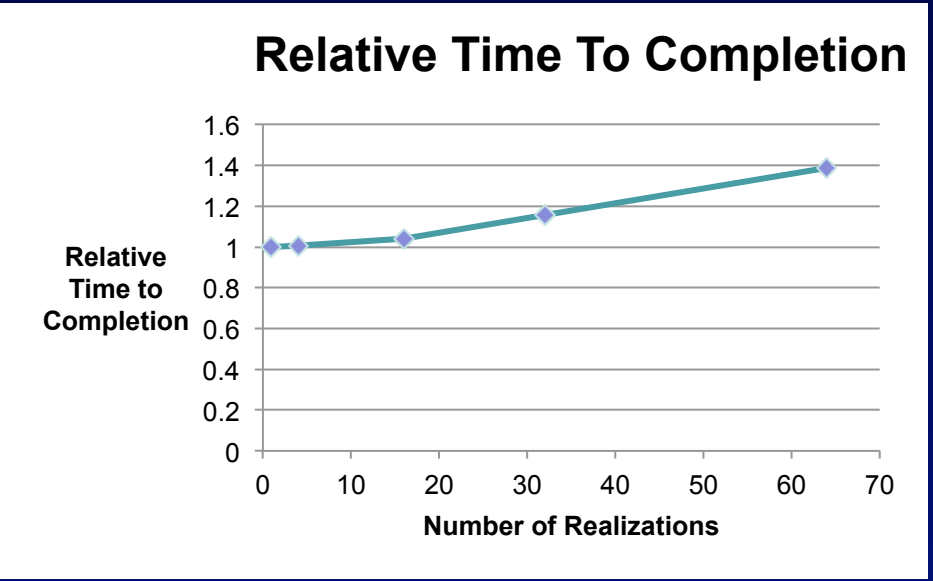
- MIC co-processors are designed to process highly parallel workloads
- Interfaces in C/C++ and Fortran
- Parallel programming paradigms follow traditional established models: OpenMP/MPI
- Automatic off-loading capability for BLAS/ LAPACK routines
- How can we put them to use in HPC for Oil & Gas?

Typical Applications on MIC

- Lowest hanging fruit, little to no code modification:
  - Multiple realizations of a reservoir simulation
  - OpenMP/multi-threaded Streamline simulations
  - Automatic offloading of BLAS/LAPACK through MKL
    - Reservoir simulation
    - Ensemble Kalman filter calculations (dense matrix linear algebra)
- Practical implications:
  - Parallel reservoir characterization/history matching
  - Quicker turn-around for reservoir simulations (Streamline and grid-based)
  - Quicker turn-around for pipe-line/wellbore flow simulations

Multiple Realizations

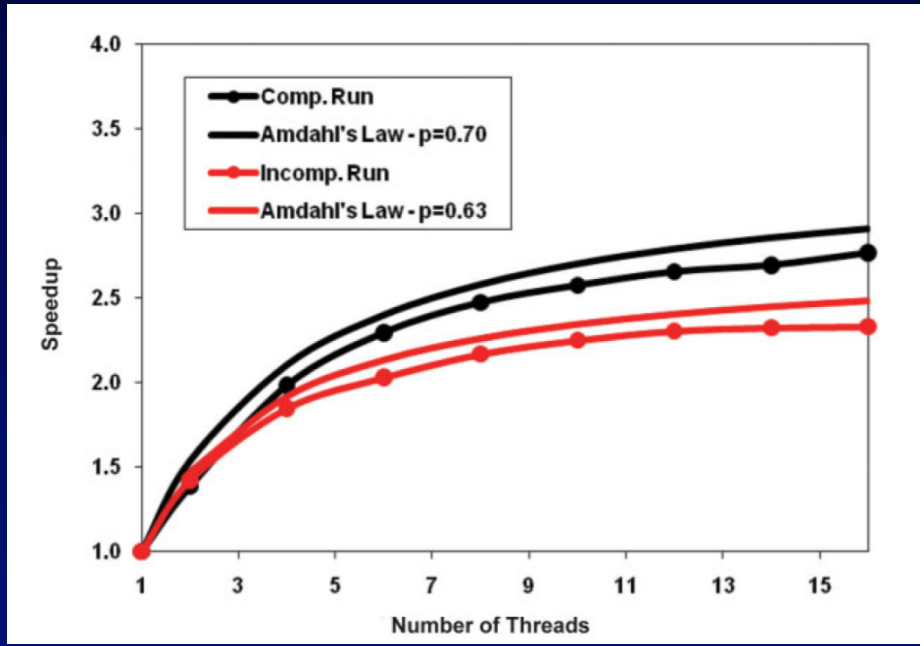
- Compile with "-mmic" (cross compiling)
- No special handling for libraries
- Launch in parallel on the MIC
- Be careful not to run out of memory
- We ran 1, 4, 16, 32 and 64 realizations concurrently



TACC

Streamline Simulations

- Threaded streamline reservoir simulators scale well
  - SPE118684 (R.P. Batyky et al)
  - SPE 113543 (H. Loef et al)
- Individual streamlines can be offloaded to the MIC to run as threads
- Pressure solve can make use of automatic offloading



Compressible and Incompressible streamline simulations (reproduced from SPE 118684)

TACC

Automatic Offloading BLAS/LAPACK

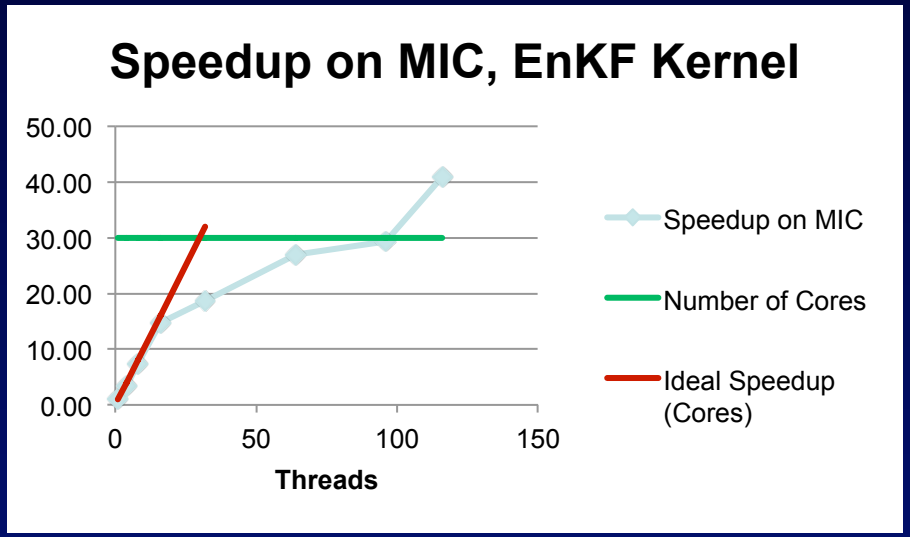
- Some common libraries, such as the Intel® Math Kernel Library (Intel® MKL) will be available in CPU versions as well as target versions
- A large number of applications use BLAS/ LAPACK routines that are available in MKL
- A lot of BLAS/LAPACK routines used in:
  - Reservoir Simulation (mesh and streamline based)
  - EnKF reservoir characterization
  - Flow through wellbores/pipelines
  - CAE for mechanical/civil/chemical applications

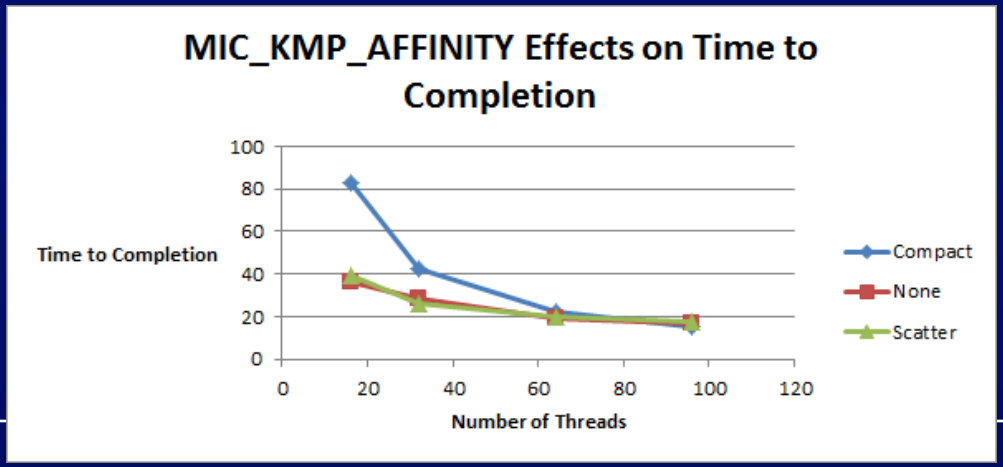
Automatic Offloading: Reservoir Simulator

- IMPES reservoir simulator implemented using:
  - PETSc: Portable Extensible Toolkit for Scientific Computing
  - MUMPS: Multifrontal Massively Parallel Sparse Direct Solver
- Only the BLAS/LAPACK calls in the direct solver are offloaded
- Tangible speedup depended on:
  - Problem size: it has to be big enough to offset offload costs
  - Number of threads
- Next step: MPI reservoir simulator runs with automatic offloading

Automatic Offloading: EnKF

- Ensemble Kalman filter analysis step is essentially dense matrix linear algebra: a few calls to BLAS/LAPACK
- EnKF Implemented in PETSc (Mat type set to Dense), no modification to code
- Ran a dense matrix linear algebra kernel
- Thread affinity is important at low thread count





TACC

The Future

- Native parallel code on the MIC:
  - PETSc supports PTHREADS, currently benchmarking
- Compiling against native MPI implementation on the MIC
  - Launching on the MIC as part of the “host list”
  - Launching exclusively (natively) on the MIC
- Writing offload code
  - MIC code is C/C++/F90, no need to write in any special language

Conclusions

- The low hanging fruit is easy to get at
- Not a single line of code was written
- No code modification
- A lot of codes will see speedup without any effort (automatic offloading)
- Actual time to get MKL automatic offloading operational was 2 days
- MIC Code is C/C++/F90 code, with familiar OpenMP directives