Experiments on accelerators

Martin Hilgeman

EMEA product technologist HPC

Agenda

- Intel Xeon Phi introduction
- Intel Xeon Phi technology
- Execution models
- Case studies
- TACC's Stampede
- Questions

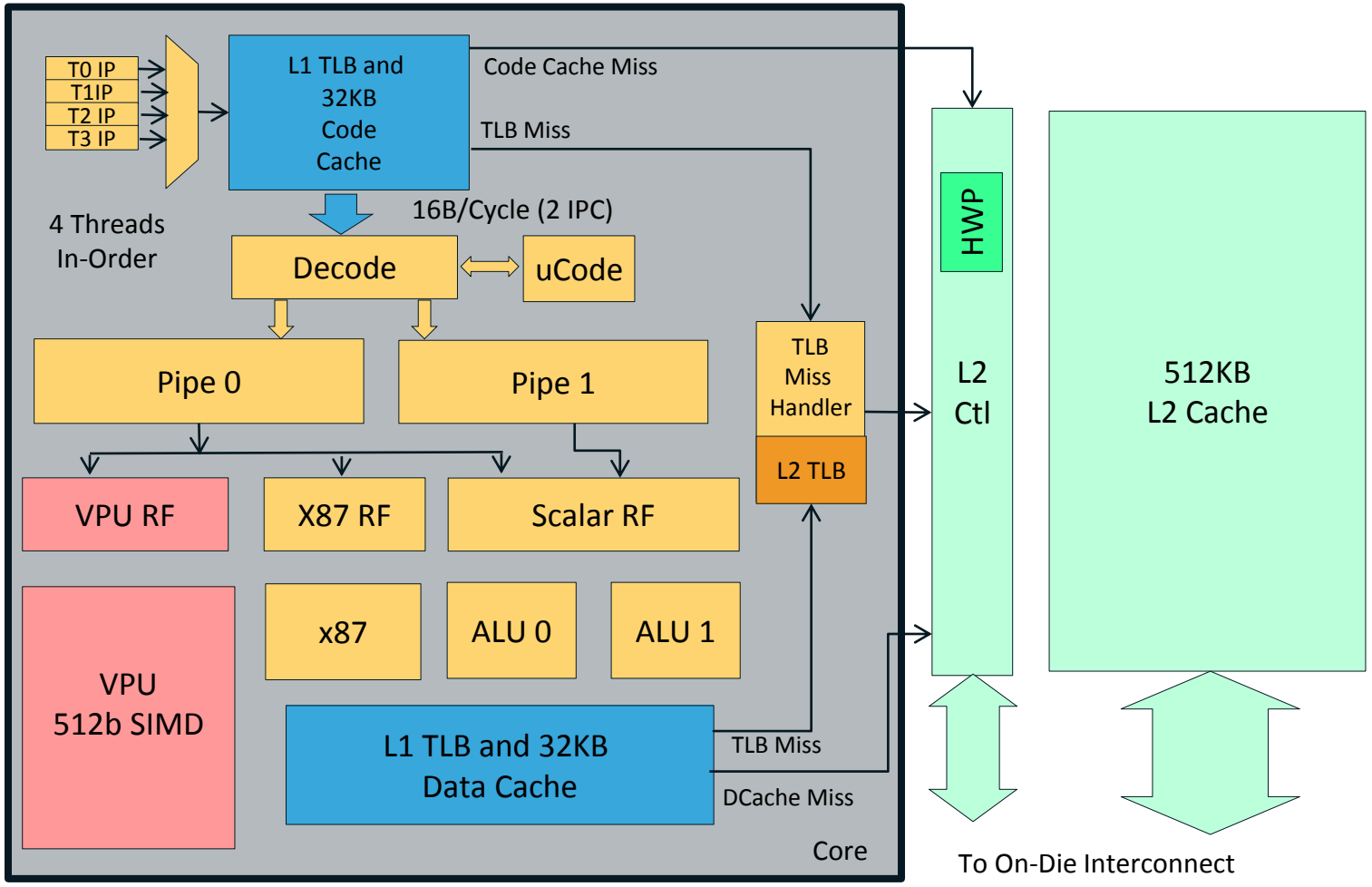
Intel Xeon Phi introduction

What is MIC?

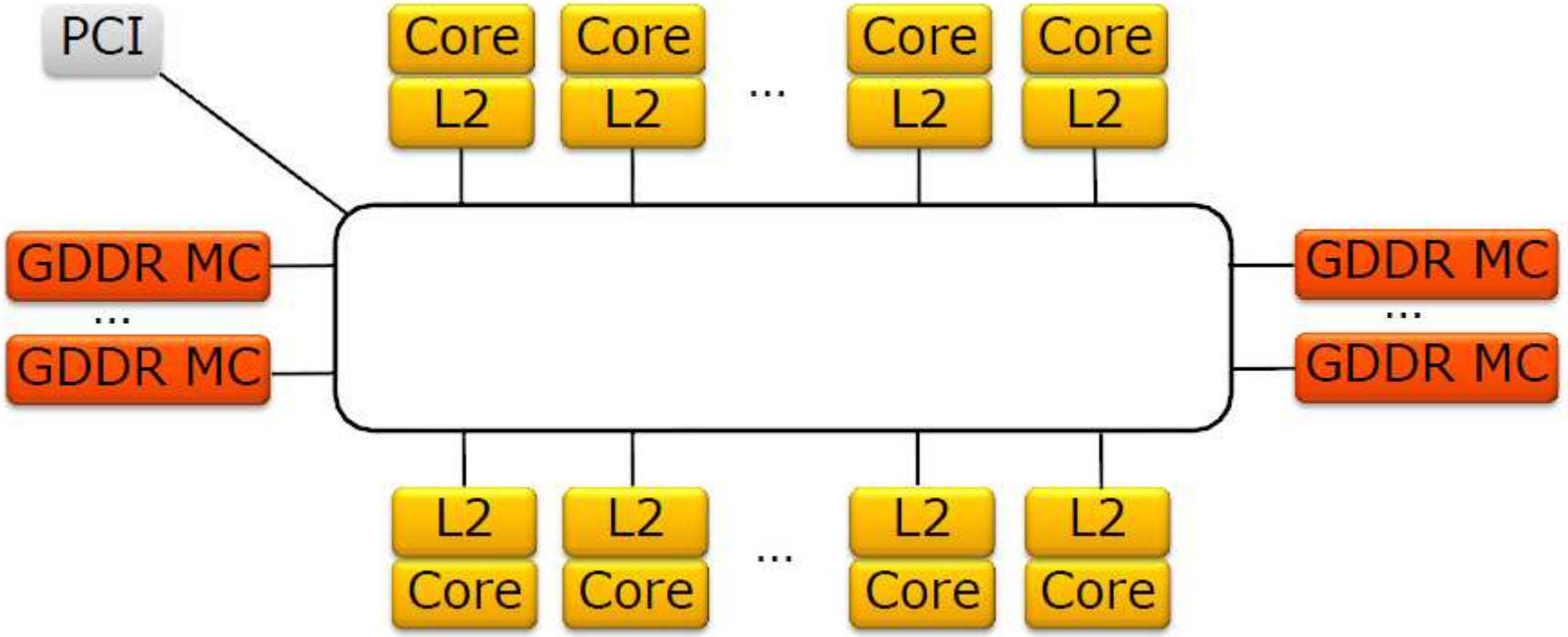
- Basic Design Ideas
  - Leverage x86 architecture (CPU with many cores)
    - x86 cores that are simpler, but allow for more compute throughput
  - Leverage (Intel's) existing x86 programming models
  - Dedicate much of the silicon to floating point ops., keep some cache(s)
  - Keep cache-coherency protocol
  - Increase floating-point throughput per core
  - Implement as a separate device
  - Strip expensive features (out-of-order execution, branch prediction, etc.)
  - Widen SIMD registers for more throughput
  - Fast (GDDR5) memory on card

Intel Xeon Phi technology

Knights Corner Silicon Core Architecture

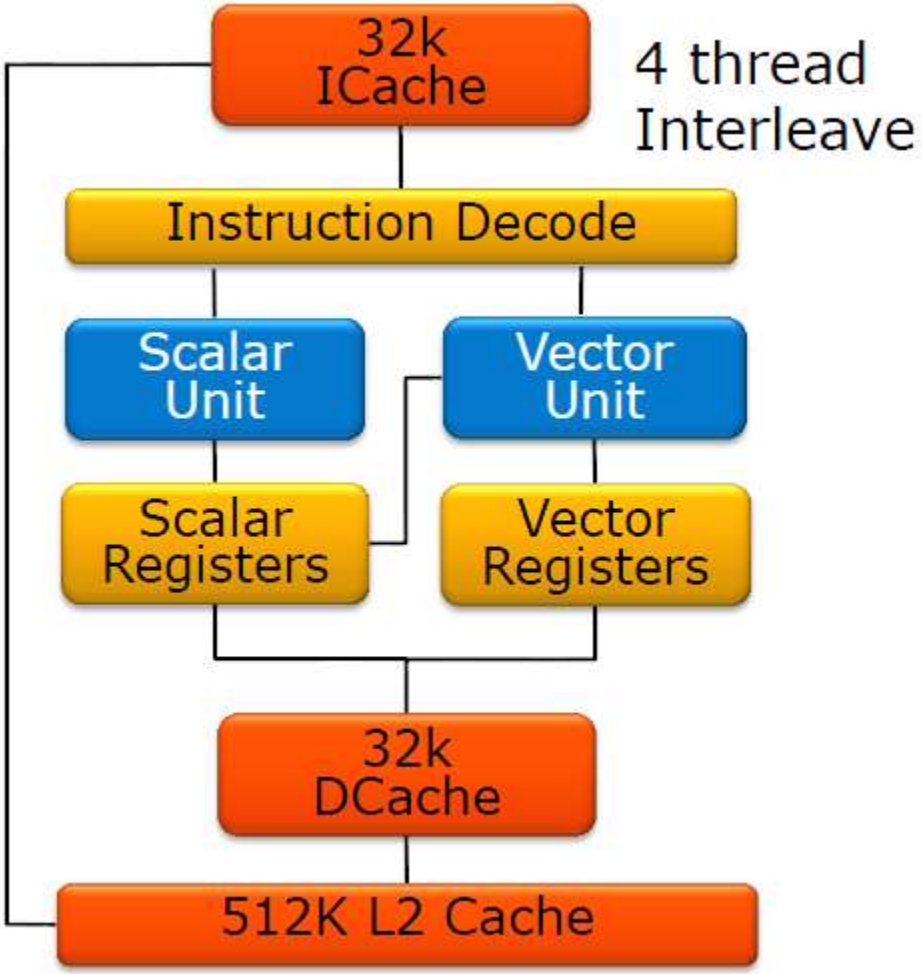


Knights Corner Architecture Overview



- 8GB GDDR5 Memory
- 8 memory controllers, 16 GDDR5 channels, up to 5.5GT/s
- 300 ns access
- aggregate 352GB/s peak memory bandwidth
- ECC

Knights Corner Architecture Overview Core



- Pentium scalar instruction set (x87!)
- Fully functional
- In order-operation
- Full 64bit addressing
- 4 HW threads/core
- Two pipelines:
  - Scalar
  - Vector/Scalar
- Intel Confidential – NDA

Porting checklist for a GPU

Investigation to utilize a GPU

GPU porting check list – will my application run on a GPU at decent speed?

1. High degree of instruction and loop level parallelism?
2. Are the parallel parts of the application self contained?
3. Data footprint on the GPU
   - If too large, can it be split into chunks?
   - If smallish, is there opportunity for data reuse?
4. Amount of host <-> device transfers need to be small, or otherwise overlapped with computation on the CPU

Case Studies

Case Study 1 – Computation of Stark Effects

Case study 1: computation of stark effects

Application characteristics:

- Written in Fortran 90
- Serial application, main matrix is 20000 x 20000 elements
- Iterative scheme

Optimization approach

- Use of compiler and libraries targeted for the system
  - For Intel: Intel Composer XE and Intel Math Kernel Library (MKL)
  - For AMD: Open64 compiler and AMD Core Math Library (ACML)
- Compile with optimization *and* debugging symbols
  - Debugging symbols to allow better code profiling
  - Start with standard, sane compiler flags

Test setup

UNIVERSITY OF CAMBRIDGE 800 YEARS 1209–2009

Jobs ran at the Dell/Cambridge HPC centre

PowerEdge C6220 with Intel Xeon E5-2670 2.6GHz

- 16 cores
- 64 GB memory
- RHEL 6.2

Building the application

Building the application

- Use the Intel C compiler 12.0.1.293
- Compile with medium optimization and debug info: '-O2 -g'
- Let the compiler generate vectorized code where possible: '-xAVX'
- Make small code modification to use BLAS3 C interface from Intel MKL

```
#include <mkl_cblas.h>
```

Run and gather profiling information

Find application hot spots using Perfsuite

- Home page at http://perfsuite.ncsa.illinois.edu/
- Allows profiling of unmodified applications
- Measures by default an instruction profile timer
- Uses PAPI to access CPU performance event registers
- Generates xml output
- Freely available

Start profiling run

```
% export PS_HWPC_CONFIG=itimer.xml

% psrun bin.x86_64/nomatdia.exe
```

Result

```
Profile Information
================================================================================
Class                          : itimer
Version                        : 1.0
Event                          : ITIMER_PROF (Process time in user and system mode)
Period                         : 10000
Samples                        : 6191
Domain                         : all
Run Time                       : 62.00 (seconds)
Min Self %                     : (all)

Module Summary
-------------------------------------------------------------------------------
Samples   Self %  Total %  Module

   6170   99.66%   99.66%  /home/dell-guest/martinh/app/nomatdia/work/02a_noopt_prof/nomatdia.exe
      9    0.15%   99.81%  /lib64/libpthread-2.5.so
      8    0.13%   99.94%  /usr/local/Cluster-Apps/intel/fce/11.1.073/lib/intel64/libifcore.so.5
      3    0.05%   99.98%  /usr/local/Cluster-Apps/intel/fce/11.1.073/lib/intel64/libintlc.so.5
      1    0.02%  100.00%  /lib64/libc-2.5.so

File Summary
-------------------------------------------------------------------------------
Samples   Self %  Total %  File

   5685   91.83%   91.83%  /home/dell-guest/martinh/app/nomatdia/work/02a_noopt/tred2.f
    472    7.62%   99.45%  /home/dell-guest/martinh/app/nomatdia/work/02a_noopt/tql2.f
     13    0.21%   99.66%  /home/dell-guest/martinh/app/nomatdia/work/02a_noopt/main.f
     13    0.21%   99.87%  ??
      8    0.13%  100.00%  interp.c
```

Detailed result

```
File Summary
------------------------------------------------------------------------------
Samples   Self %  Total %  File

   5685   91.83%   91.83%  /home/dell-guest/martinh/app/nomatdia/work/02a_noopt/tred2.f
    472    7.62%   99.45%  /home/dell-guest/martinh/app/nomatdia/work/02a_noopt/tql2.f
     13    0.21%   99.66%  /home/dell-guest/martinh/app/nomatdia/work/02a_noopt/main.f
     13    0.21%   99.87%  ??
      8    0.13%  100.00%  interp.c

Function:File:Line Summary
------------------------------------------------------------------------------
Samples   Self %  Total %  Function:File:Line

   1886   30.46%   30.46%  tred2:/home/dell-guest/martinh/app/nomatdia/work/02a_noopt/tred2.f:120
   1098   17.74%   48.20%  tred2:/home/dell-guest/martinh/app/nomatdia/work/02a_noopt/tred2.f:103
   1089   17.59%   65.79%  tred2:/home/dell-guest/martinh/app/nomatdia/work/02a_noopt/tred2.f:83
   1004   16.22%   82.01%  tred2:/home/dell-guest/martinh/app/nomatdia/work/02a_noopt/tred2.f:89
    393    6.35%   88.35%  tred2:/home/dell-guest/martinh/app/nomatdia/work/02a_noopt/tred2.f:82
    257    4.15%   92.51%  tql2:/home/dell-guest/martinh/app/nomatdia/work/02a_noopt/tql2.f:129
    138    2.23%   94.73%  tql2:/home/dell-guest/martinh/app/nomatdia/work/02a_noopt/tql2.f:128
    117    1.89%   96.62%  tred2:/home/dell-guest/martinh/app/nomatdia/work/02a_noopt/tred2.f:123
     48    0.78%   97.40%  tql2:/home/dell-guest/martinh/app/nomatdia/work/02a_noopt/tql2.f:127
     30    0.48%   97.88%  tred2:/home/dell-guest/martinh/app/nomatdia/work/02a_noopt/tred2.f:88
     26    0.42%   98.30%  tred2:/home/dell-guest/martinh/app/nomatdia/work/02a_noopt/tred2.f:102
     11    0.18%   98.48%  tred2:/home/dell-guest/martinh/app/nomatdia/work/02a_noopt/tred2.f:119
     10    0.16%   98.64%  tred2:/home/dell-guest/martinh/app/nomatdia/work/02a_noopt/tred2.f:47
      8    0.13%   98.77%  __write_nocancel:interp.c:0
```

What does tred2.f do?

```
 1        SUBROUTINE TRED2(NM,N,A,D,E,Z)
 2  C
 3        INTEGER I,J,K,L,N,II,NM,JP1
 4        REAL*8 A(NM,N),D(N),E(N),Z(NM,N)
 5        REAL*8 F,G,H,HH,SCALE
 6        REAL*8 DSQRT,DABS,DSIGN
 7  C
 8  C     THIS SUBROUTINE IS A TRANSLATION OF THE ALGOL PROCEDURE TRED2,
 9  C     NUM. MATH. 11, 181-195(1968) BY MARTIN, REINSCH, AND WILKINSON.
10  C     HANDBOOK FOR AUTO. COMP., VOL.II-LINEAR ALGEBRA, 212-226(1971).
11  C
12  C     THIS SUBROUTINE REDUCES A REAL SYMMETRIC MATRIX TO A
13  C     SYMMETRIC TRIDIAGONAL MATRIX USING AND ACCUMULATING
14  C     ORTHOGONAL SIMILARITY TRANSFORMATIONS.


78           DO 240 J  =  1, L
79              Z(J,I)  =  Z(I,J) / H
80              G  =  0.0D0
81  C     :::::::::: FORM ELEMENT OF A*U ::::::::::
82              DO 180 K  =  1, J
83    180       G  =  G + Z(J,K) * Z(I,K)
84  C
85              JP1  =  J + 1
86              IF (L .LT. JP1) GOTO 220
87  C
88              DO 200 K  =  JP1, L
89    200       G  =  G + Z(K,J) * Z(I,K)
90  C     :::::::::: FORM ELEMENT OF P ::::::::::
91    220       E(J)  =  G / H
92              F  =  F + E(J) * Z(I,J)
93    240    CONTINUE
```

Sounds like a linear algebra function

Bad memory access pattern

Who is calling the tred2 routine?

Profling with the ‘gprof’ command can generate a call stack

```
                0.00   62.20      11/11          MAIN__ [1]
[3]     99.8    0.00   62.20      11         rs_ [3]
               57.34    0.00      11/11          tred2_ [4]
                4.86    0.00      11/11          tql2_ [5]
-----------------------------------------------
               57.34    0.00      11/11          rs_ [3]
[4]     92.0   57.34    0.00      11         tred2_ [4]
-----------------------------------------------
                4.86    0.00      11/11          rs_ [3]
[5]      7.8    4.86    0.00      11         tql2_ [5]
```

RS is the only routine that calls tred2 for a total of 11 times

...and is also the sole caller of tql2...

What does the RS subroutine do?

RS

```
 SUBROUTINE RS(NM,N,A,W,MATZ,Z,FV1,FV2,IERR)
C
      INTEGER N,NM,IERR,MATZ
      REAL*8 A(NM,N),W(N),Z(NM,N)
      REAL (8)               :: fv1(n), fv2(n)
C
C     THIS SUBROUTINE CALLS THE RECOMMENDED SEQUENCE OF
C     SUBROUTINES FROM THE EIGENSYSTEM SUBROUTINE PACKAGE (EISPACK)
C     TO FIND THE EIGENVALUES AND EIGENVECTORS (IF DESIRED)
C     OF A REAL SYMMETRIC MATRIX.
C
C     ON INPUT:
C
C        NM  MUST BE SET TO THE ROW DIMENSION OF THE TWO-DIMENSIONAL
C        ARRAY PARAMETERS AS DECLARED IN THE CALLING PROGRAM
C        DIMENSION STATEMENT;
C
C        N  IS THE ORDER OF THE MATRIX  A;
C
C        A  CONTAINS THE REAL SYMMETRIC MATRIX;
C
C        MATZ  IS AN INTEGER VARIABLE SET EQUAL TO ZERO IF
C        ONLY EIGENVALUES ARE DESIRED;  OTHERWISE IT IS SET TO
C        ANY NON-ZERO INTEGER FOR BOTH EIGENVALUES AND EIGENVECTORS.

…

   20 CALL  TRED2(NM,N,A,W,FV1,Z)
      CALL  TQL2(NM,N,W,FV1,Z,IERR)
```

RS is an eigenvalue and/or eigenvector solver

And calls both TRED2 and TQL2

Inserting an optimized eigenvalue solver

- The LAPACK library contains various routines for solving linear algebra problems and matrix operations
  - From the http://www.netlib.org home page:

    LAPACK provides routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems. The associated matrix factorizations (LU, Cholesky, QR, SVD, Schur, generalized Schur) are also provided, as are related computations such as reordering of the Schur factorizations and estimating condition numbers. Dense and banded matrices are handled, but not general sparse matrices. In all areas, similar functionality is provided for real and complex matrices, in both single and double precision.

- There are highly optimized implementations available from various sources
  - Intel: Intel Math Kernel Library (MKL)
  - AMD: AMD Core Math Library (ACML)
  - Open source: Netlib, ATLAS, GOTOblas

Insert the LAPACK call in RS

```
 SUBROUTINE RS(NM,N,A,W,MATZ,Z,FV1,FV2,IERR)
C
      INTEGER N,NM,IERR,MATZ
      REAL*8 A(NM,N),W(N),Z(NM,N)
#if defined (DELL_OPT)
      REAL (8), ALLOCATABLE   :: work(:)
      INTEGER (4), ALLOCATABLE:: iwork(:)
      INTEGER (4)             :: lwork, liwork, info, i, j
#else
      REAL (8)                :: fv1(n), fv2(n)
#endif
…
#if defined (DELL_OPT)
C calculate the size of the work array
   20 lwork = 1 + 6*n + 2*n*n
      liwork = 3 + 5*n
      ALLOCATE (work(lwork), iwork(liwork))
C calculate eigenvalues and eigenvectors
      CALL  dsyevd ('v', 'u', n, a, nm, w, work, lwork, iwork, liwork,
     .              info)
      IF (info .NE. 0) THEN
         WRITE(*,*)'dsyevd failed:', info
         STOP
      END IF
      CALL fastbcopy8 (nm * n, a, z)
#else
   20 CALL  TRED2(NM,N,A,W,FV1,Z)
      CALL  TQL2(NM,N,W,FV1,Z,IERR)
#endif
```

LAPACK dsyevd call used

Results

	Wall Clock Time (hh:mm:ss)
Original version	4:59:31
Inserted LAPACK dsyev call	0:07:18
Additional source optimizations	0:06:58

Can this be improved further?

Apply the GPU check list to my application

1. High degree of instruction and loop level parallelism?

   No, but computational time is dominated by a LAPACK call

2. Are the parallel parts of the application self contained?

   Yes, the LAPACK call can probably be changed to a GPU aware call?

3. Data footprint on the GPU
   - If too large, can it be split into chunks?
   - If smallish, is there opportunity for data reuse?

   The matrix A is around 3 GB in size, so just fits on a GPU

4. Amount of host <-> device transfers need to be small, or otherwise overlapped with computation on the CPU

   Yes, the LAPACK function is only called once per iteration

Availability of numerical libraries for CUDA

1. CUBLAS
   - Developed by NVIDIA
   - Only contains BLAS functions
2. CULA
   - http://www.culatools.com/
   - Contains dense and sparse implementations of most LAPACK functions (factorization, eigenvalue problems, least squares, decompositions)
   - Costs $$$
3. MAGMA
   - http://icl.cs.utk.edu/magma/index.html
   - Dense LAPACK library targeted at hy\
   -]rid GPU – CPU implementations
   - Developed by Jack Dongarra's group
   - Freely available

Insert the MAGMA call

```
       USE magma
       USE cuda_alloc
       USE iso_c_binding
       INTEGER(4)           :: stat
       REAL(8), ALLOCATABLE :: wid(:), yint(:)
       REAL(8), DIMENSION(:,:), POINTER :: xx
       REAL(8), PARAMETER   :: real_size = DBLE(0.d0)
       type(C_PTR)          :: cptr_xx
…

C Initialization
       CALL cublas_init()
…
C Allocate pinned host memory
       stat = cudaMallocHost (cptr_xx, lsize * lsize *
     .                        sizeof(real_size))
       CALL c_f_pointer (cptr_xx, xx, [lsize, lsize])
       ALLOCATE (wid(lsize), yint(lsize))

C Issue the LAPACK call
       CALL  magmaf_dsyevd (‘v', 'u', n, a, nm, w, work, lwork, iwork,
     .                     liwork, info)


stat = cudaFreeHost (cptr_xx)

 CALL cublas_shutdown()
```

LAPACK dsyevd call used

GPU only version

```
      magma_devptr_t          :: devptrA
      INTEGER (4)             :: lda, ldda, stat, ldwa
      INTEGER (4), EXTERNAL   :: cublas_alloc
      REAL (8), ALLOCATABLE   :: wa(:,:)
      INTEGER (4), PARAMETER  :: size_of_double=8


C allocate GPU memory
      stat = cublas_alloc (ldda*n, size_of_double, devPtrA)
      IF (stat .NE. 0) THEN
         WRITE(*,*) "devPtrA allocation failed"
         STOP
      END IF


C copy A to device
      CALL cublas_set_matrix (nm, n, size_of_double, a, lda, devptrA,
     .                        ldda)

C issue the LAPACK call
      CALL magmaf_dsyevd_gpu ('v', 'u', n, devptrA, ldda, w, wa, ldwa,
     .                        work, lwork, iwork, liwork, info)

C copy A back to host
      CALL cublas_get_matrix (nm, n, size_of_double, devptrA, ldda, a,
     .                        lda)
```

LAPACK dsyevd call used

Results

	Wall Clock Time (hh:mm:ss)
Original version	4:59:31
Inserted LAPACK dsyev call	0:07:18
Additional source optimizations	0:06:58
MAGMA hybrid CPU/GPU	0:03:52
MAGMA GPU only	0:03:51
MAGMA hybrid pinned host memory	0:03:41
Same as above, with M2050 card	0:02:04

Case Study 2 – Geospatial imaging

Case study 2: Geospatial imaging

Application characteristics:

- Written in C90 and C++, ~ 150,000 lines of code
- Serial application, using Intel MKL DFTI functions
- Iterative scheme

Optimization approach

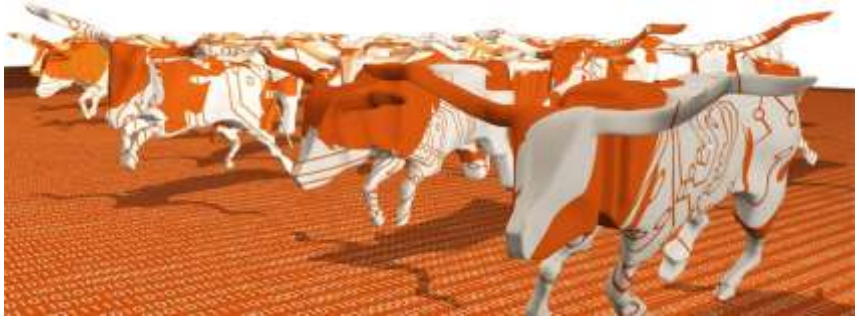
- Use of compiler and libraries targeted for the system
  - For Intel: Intel Composer XE and Intel Math Kernel Library (MKL)
- Compile with optimization *and* debugging symbols
  - Debugging symbols to allow better code profiling
  - Start with standard, sane compiler flags
- Profile the application to find hotspots and function usage

Test Setup

Jobs ran at the TACC Stampede system

PowerEdge C8220 with Intel Xeon E5-2680 2.7GHz

- 16 cores
- 32 GB memory
- RHEL 6.3
- Intel Xeon Phi 7120P



TACC

TEXAS ADVANCED COMPUTING CENTER

Code details

- MKL FFTs are already being used, little room for improvement
  - FFTs are too small to run in threaded mode
- Application only runs in serial mode, need to be parallelized in order to run on a Phi in reasonable speed
- Phase images are divided into patches, which is suitable for parallelization
  - Parallelize the main loop using OpenMP

Parallelization of big main loop

```
count = 1;
if (lastGCP != NULL) {
   app = lastGCP->next;
} else {
   app = bufferGCP;
}









while (app) {




   app->loc   [0] = (int) app->loc   [0];
   app->loc   [1] = (int) app->loc   [1];
   app->offset[0] = (int) app->offset[0];
   app->offset[1] = (int) app->offset[1];

   <lot of work>

   lastGCP = app;
   app = app->next;
   count++;
}
```

```
count = 1;
if (lastGCP != NULL) {
   app = lastGCP->next;
} else {
   app = bufferGCP;
}

int i = 0;
int nthreads, me;
#if defined _OPENMP
#pragma omp parallel \
   private(i,me,app2,err,suboffset,qc,m_block,s_block)
{
   nthreads = omp_get_num_threads();
   me = omp_get_thread_num();
#else
   nthreads = 1;
   me = 0;
#endif

#pragma omp for schedule(static)
   for (i = 0; i < buffer_ngcp; i++) {
#pragma omp critical
      {
         app2 = app;
         app = app->next;
      }
      app2->loc   [0] = (int) app2->loc   [0];
      app2->loc   [1] = (int) app2->loc   [1];
      app2->offset[0] = (int) app2->offset[0];
      app2->offset[1] = (int) app2->offset[1];

      <lot of work>

      lastGCP = app2;
      count++;
   }
#if defined _OPENMP
}
#endif
```

Results

	Wall Clock Time (ss:00)
Original version on E5440 2.83 GHz	102.00
Original version on E5-2680 2.70 GHz	27.43
Additional source optimizations	25.83
2 threads	17.37
4 threads	9.97
6 threads	7.17
8 threads	5.77
12 threads	5.57

A profile

```
Module Summary
-------------------------------------------------------------------------------
Samples   Self %  Total %  Module

   247   59.81%   59.81%  /lib64/libc-2.12.so
    79   19.13%   78.93%  /scratch/dell-guest/app
    38    9.20%   88.14%  /opt/intel/mkl/lib/intel64/libmkl_core.so
    34    8.23%   96.37%  /opt/intel/mkl/lib/intel64/libmkl_intel_thread.so
    10    2.42%   98.79%  /opt/intel/mkl/lib/intel64/libmkl_intel_lp64.so
     5    1.21%  100.00%  /lib64/ld-2.12.so

File Summary
-------------------------------------------------------------------------------
Samples   Self %  Total %  File

   332   80.39%   80.39%  ??
    64   15.50%   95.88%  malloc.c
    10    2.42%   98.31%  interp.c
     3    0.73%   99.03%  bsearch.c
     2    0.48%   99.52%  SRC/patch.c
     2    0.48%  100.00%  printf_fp.c

Function Summary
-------------------------------------------------------------------------------
Samples   Self %  Total %  Function

   126   30.51%   30.51%  brk
    69   16.71%   47.22%  ??
    60   14.53%   61.74%  _int_malloc
    43   10.41%   72.15%  pmatch
    34    8.23%   80.39%  memcpy
    13    3.15%   83.54%  get_correlation_real_mkl
    10    2.42%   85.96%  extract2double
     8    1.94%   87.89%  __intel_new_memcpy
```

How about Xeon Phi performance?

- Now runs multi-threaded using OpenMP directives
- Memory footprint is small, so should fit on the card
  - Running in native mode is possible
- Intel MKL has complete support for Xeon Phi

Case study: TACC's Stampede system

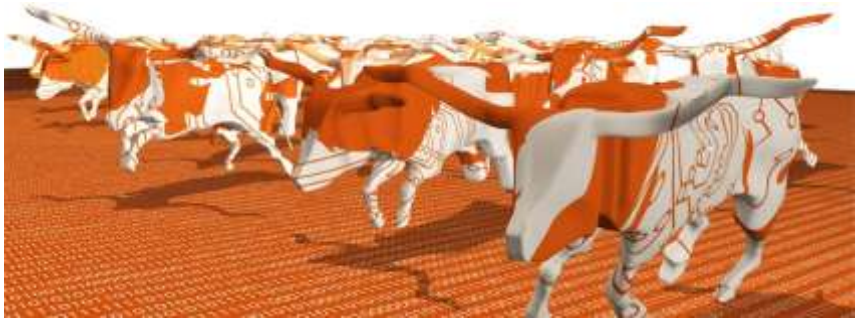
Most of the following slides are from Jay Boisseau, director of TACC

Stampede: Background

- TACC evaluation
  - Linux/x86 easiest path for current users
  - Needed add'l perf from innovative component
    - community really needs >> 2PF due to lack of recent large-scale NSF HPC systems
    - So, we considered 'acceleration' options for more PFs
  - Evaluated GPU and MIC—decided on MIC
    - Easier porting
    - More programming options (x86)—not just acceleration
    - Leveraging x86 optimization experience
    - More 'innovative?' (Didn't exist yet…)

Key Aspects of Acceleration

- We have lots of transistors… Moore's law is holding; this isn't necessarily the problem.
- We don't really need lower power per transistor, we need lower power per *operation*.
- How to do this?



TACC

TEXAS ADVANCED COMPUTING CENTER

Coprocessor vs. Accelerator

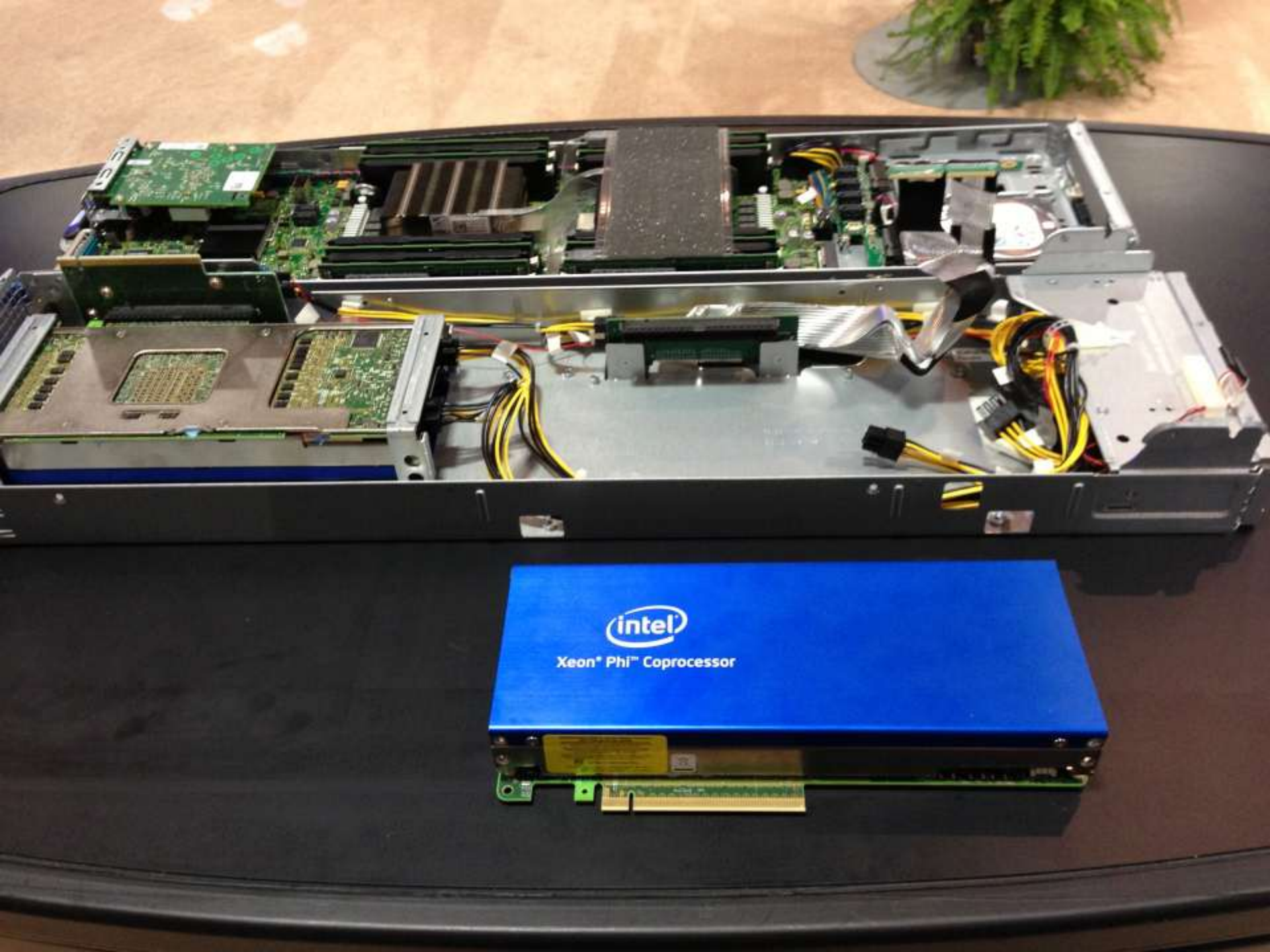
	Co-processor	Accelerator
Architecture	x86	Streaming processors
	Coherent caches	Shared memory and caches
Programming model	C/C++/Fortran + directives OpenCL support planned	CUDA, OpenACC, OpenCL
Threading model	OpenMP and multithreading MPI host/device/hybrid	Threads in hardware MPI on host only
Programming details	Offloaded regions Native mode	Kernels, OpenACC
Scripting	Yes	No
Run on device only	Yes	No

What We Like about MIC

- 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
  - **"Optimize once, run anywhere"**
  - Our early MIC porting efforts for codes "in the field" are frequently doubling performance on Sandy Bridge.

TACC's Stampede: The Big Picture

- Dell, Intel, and Mellanox are vendor partners
- Almost 10 PF peak performance in initial system (2013)
  - 2+ PF of Intel Xeon E5 (6400 dual-socket nodes)
  - 7+ PF of Intel Xeon Phi (MIC) coprocessors (several thousand)
    - Special edition/release for this project!
- 14+ PB disk, 150+ GB/s I/O bandwidth
- 250+ TB RAM
- 56 Gb/s Mellanox FDR InfiniBand interconnect
- 16 x 1TB large shared memory nodes
- 128 Nvidia Kepler K20 GPUs for remote viz
- 182 racks, 6 MW power!


intel
Xeon® Phi™ Coprocessor

Stampede: How Will Users Use It?

- 2+ PF Xeon-only system (MPI, OpenMP)
  - Many users will use it as an extremely powerful Sandy Bridge cluster—and that's OK!
    - They may also use the shared memory nodes, remote vis
- 7+ PF MIC-only system (MPI, OpenMP)
  - Homogeneous codes can be run entirely on the MICs!
- ~10PF heterogeneous system (MPI, OpenMP)
  - Run separate MPI tasks on Xeon vs. MIC; use OpenMP extensions for offload for hybrid programs

Will My Code Run on MIC?

- Yes

- That's the wrong question, it's:
  - Will your code run *best* on MIC?, or
  - Will you get great MIC performance without additional work?

Early MIC Programming Experiences at TACC

- Codes port easily
  - Minutes to days depending mostly on library dependencies
- Performance can require real work
  - While the sw environment continues to evolve
  - Getting codes to run *at all* is almost too easy; really need to put in the effort to get what you expect
- Scalability is pretty good
  - Multiple threads per core *really important*
  - Getting your code to vectorize *really important*

Key Questions

Why do we need to use a thread-based programming model?

MPI programming

Where to place the MPI tasks?

How to communicate between host and 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

➢ Threads (OpenMP, etc.) are a must on MIC

Which to Choose: Intel® Xeon® or Intel® MIC?

- For most applications, Intel® Xeon® processor will continue to be “best” choice
  - Multi-core performance
  - Great Vector and Integer performance
  - Best per thread performance
  - Large Memory Footprint
- For certain highly parallel applications, Intel® Many Integrated Core architecture will provide enhanced performance
  - Think HPC applications, but not limited to this segment
  - Highly Parallel
    - Computationally complex problems that can be broken down into smaller component problems run in parallel
    - Minimal Serial code (or serial code runs on host)
  - Vectorizable (able to use SIMD instructions)
  - Limited data dependencies
  - Limited data set size

Next Steps

- Intel® Xeon Phi™ coprocessor (Knights Corner) production in Q4 2012, General Availability Q1 '13
  - KNC pre-production is not for general availability
- Range of available platforms from Dell
  - Validating C8000, C410X, R720 and T620
- What to do now?
  - The first step in optimizing code for Intel® MIC is to optimize for Intel® Xeon® processors
    - Ensure you code scales well with cores and makes use of SIMD instructions
      - E.g Parallelize & Vectorize your code
  - If not already, use/become familiar with Intel SW tools
    - Because Intel tools will be the first to support Intel® MIC
  - Get help with parallel programming here:

Next Steps

- What to do now?
  - The first step in optimizing code for Intel® MIC is to optimize for Intel® Xeon® processors
    - Ensure you code scales well with cores and makes use of SIMD instructions
      - E.g Parallelize & Vectorize your code
  - If not already, use/become familiar with Intel SW tools
    - Because Intel tools will be the first to support Intel® MIC
  - Get help with parallel programming here:
    - intel.com/go/parallel
    - http://software.intel.com/en-us/articles/intel-guide-for-developing-multithreaded-applications/
- Intel / Dell team look at ways to support customer opportunities

The power to do more