Optimizing Applications for Summit

Peter Ruprecht
peter.ruprecht@colorado.edu

www.rc.colorado.edu

Outline

- Overview of Summit's hardware components
- Why is optimization important?
- Brief intro to parallel programming
- More info on vectorization
- I/O optimization
- Running over OPA
- Best practices for Phi
- High-throughput computing

Summit: RMACC's Next-Generation Supercomputer

- Funded via an NSF MRI grant awarded jointly to CU-Boulder and CSU
- Installed and running by fall semester 2016
- *About 10% of core-hours available to RMACC*
- Theoretical peak performance over 400 TFLOPS* (compared with about 170 for Janus)
- Mix of established and cutting-edge technology
- Expected 5-year useful lifetime
- Principal vendor and integrator is Dell

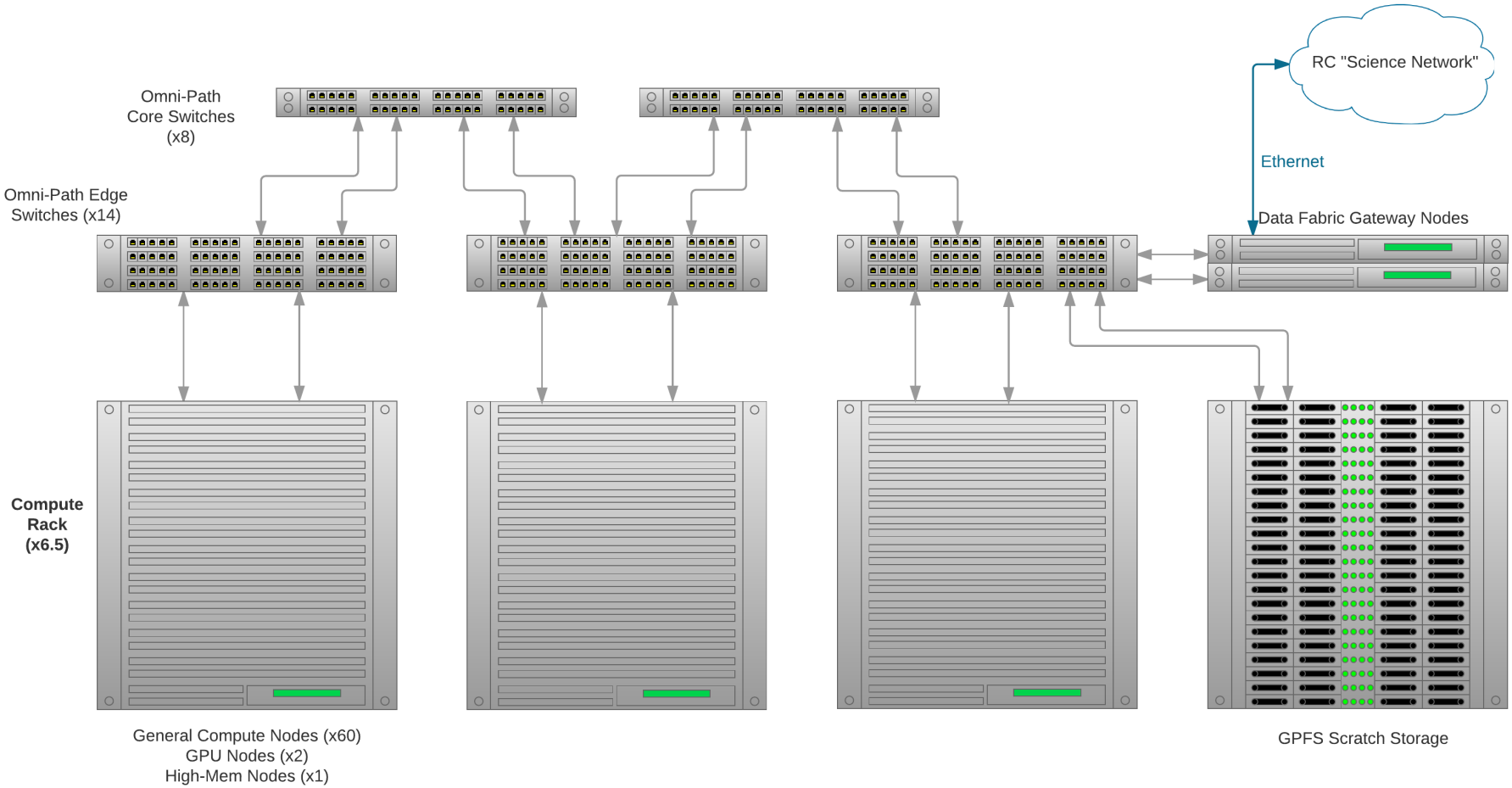
*trillion floating point operations per second

Schedule

- Late April – mini-Summit test cluster available for software builds and small-scale testing
- Mid July – Summit delivered and installed in HPCF
- Early Aug – initial acceptance testing
  - Dell and DDN are performing the hardware tests
- Late Aug – application testing and early user access
  - After handoff of the cluster to CU-RC
- Mid Oct – general availability
- Dec-Jan – Knights Landing Phi nodes installed; possible expansion of general compute

SUMMIT SCHEMATIC

Peter Ruprecht | July 15, 2016



Summit: Node Types

- 380 general compute nodes
  - 24 real cores and 128 GB RAM
  - local SSD
- 10 GPGPU/visualization nodes
  - 2x NVIDIA K80 GPUs
- 5 High-memory nodes
  - 2 TB RAM each
- 20 Xeon Phi (“Knights Landing”) nodes
  - Phi installed directly in CPU socket; not a separate card
  - 72 cores / 288 threads “many core”
  - installed in second phase, late 2016

Omni-Path (OPA) Interconnect

- Cutting-edge network product from Intel
- Same role as the InfiniBand fabric on Janus or Yellowstone
  - Also carries NFS traffic from /home, /projects, /work ...
- 100 Gb/s bandwidth
- Extremely low latency for MPI performance
- Fat-Tree topology with 8 core switches and 32 nodes per edge switch (easily expandable)
- "Islands" of 32 nodes fully non-blocking
- 2:1 blocking factor between islands

Omni-Path (OPA) Interconnect

- Remote Direct Memory Access – allows processes on one node to access memory on other nodes
- Verbs – OpenFabrics API allowing user applications to use RDMA
- PSM – Performance Scaled Messaging; supposed to provide higher message rates than conventional IB verbs
- MPI uses these

GPFS Scratch Storage

- 1 PB of high-performance scratch storage
- GPFS for parallel access and improved small-file performance
- Directly-attached to OPA fabric (in a later update)
- Expandable to about 2 PB; performance also scales up as more drives are added
- DataDirect Networks SFA14K "GridScaler" appliance with embedded and external file servers
- Expect >20 GB/s parallel throughput and >10K file creations per second

Summit vs Janus

Feature	Janus	Summit
CPU cores/node Clock frequency	12 2.8 - 3.2 GHz	24 2.5 - 3.3 GHz
RAM/core Memory bandwidth	2 GB 32 GB/s	5 GB 68 GB/s
Interconnect type Bandwidth Latency	QDR InfiniBand 40 Gb/s 1.2 us	Omni-Path 100 Gb/s 0.4 us
Filesystem	Lustre – optimized for large parallel transfers	GPFS – good at parallel transfers; also good at small file operations
Vectorization	SSE4 (4 operations/cycle)	AVX2 (8 operations/cycle) AVX-512 (16 ops/cycle)

Why Optimize for Summit?

- Summit has fewer nodes than Janus, thus we can allocate fewer core-hours (SU)
- Summit is shared between CU, CSU, and RMACC, thus we can only allocate a fraction of core-hours to each

- If you take advantage of Summit's performance features, your allocation will go much farther, so
  - You will get more results
  - You will publish more papers
  - You will take fewer years to graduate
  - You will get { postdoc | awesome faculty position | tenure | Nobel prize }

Getting the most out of Summit

- Any application that you have built for Janus, the CSU Cray, Yellowstone, etc will need to be recompiled
- Applications ideally should take advantage of multi-core / many-core architectures (ie, parallelize)
- Performance improvements in latest processors are mainly through more cores and SIMD* rather than faster clock speed
- Applications should be parallelized and “vectorized” … otherwise Summit may *seem* slower than older systems

*single instruction on multiple data

Parallel Programming in 45s

- SIMD/Vectorization : parallelization within a CPU core
- OpenMP/threading : parallelization within a node
- MPI : parallelization across multiple nodes
- Hybrid parallelization : using MPI and threading together

Vectorization overview

- Vectorizing a computation lets it take advantage of SIMD instruction sets
- That is, during one CPU cycle a single CPU core can do the same operation on several data objects (8 or 16 at once in the case of Summit)
- The compiler can auto-vectorize some loops; the programmer needs to write code that the compiler can easily vectorize
- Some math libraries, such as Intel MKL, are fully optimized for vectorization
- Many applications have vectorization built in
  - Matlab
  - Python provided by RC

More on Vectorization

Data register:

data 1	data 2	data 3	data 4

Consider this loop:

```
for (i=1 ; i<=N ; i++)
    b[i] = 3 + a[i];
```

(inspired by Intel Autovectorization Guide)

Without vectorization:

3 + a[1]	not used	not used	not used

With vectorization:

3 + a[1]	3 + a[2]	3 + a[3]	3 + a[4]

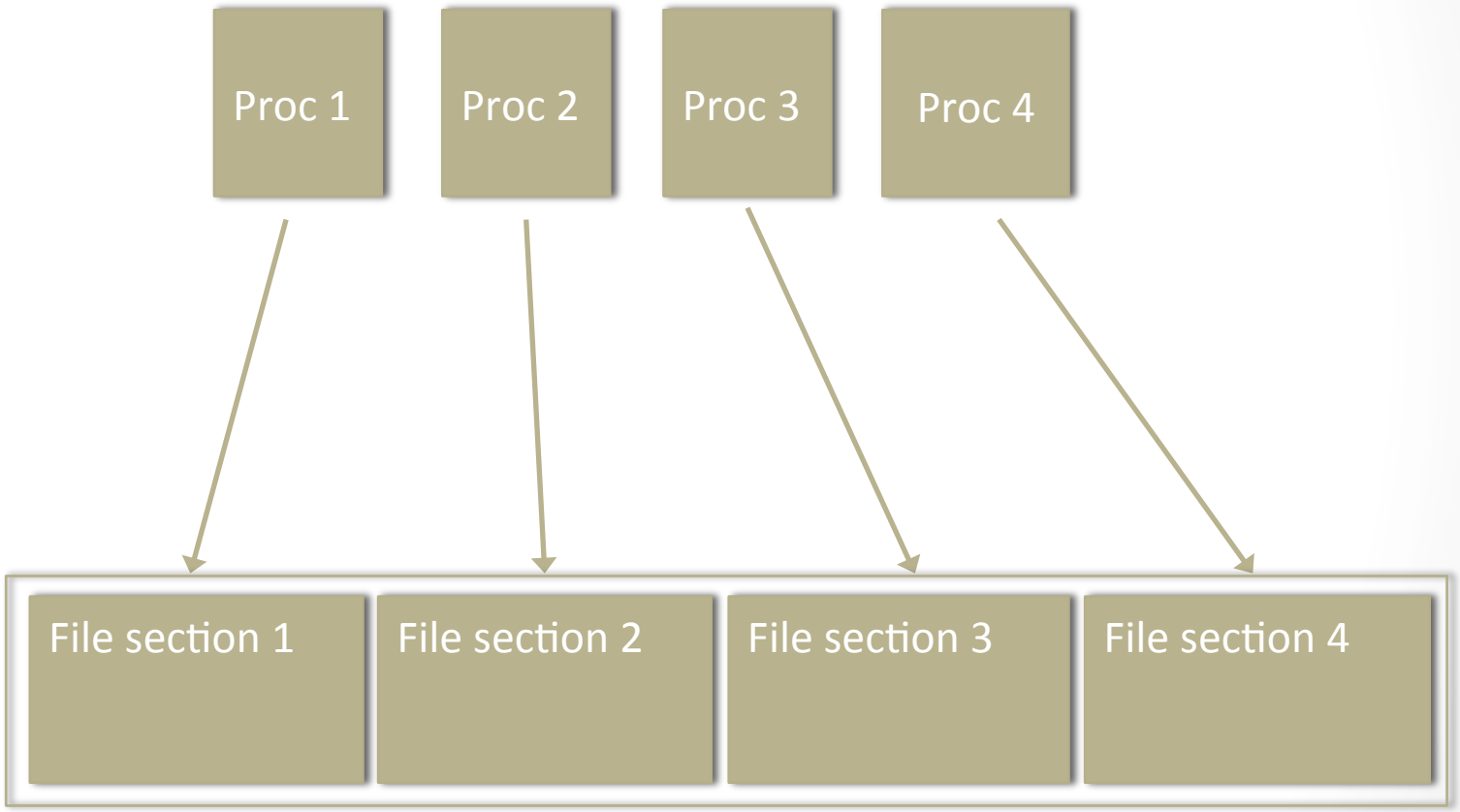
Enabling Vectorization

- Set appropriate compiler flag
  - `-march=core-avx2`
  - `-vec-report` or `-qopt-report`
- Loop requirements:
  - Exit from loop must not be data-dependent, i.e. loop trip count must be known at runtime
  - No branching within the loop (`if`-statements allowed in some cases)
  - No function calls within the loop (except intrinsic math functions that are vectorized)
- Align data on register boundaries
- Minimize use of Fortran global variables
- Be careful with C pointers (try `-restrict` flag)
- Give the compiler hints whenever possible

I/O Optimization

- Total Job Time = Computational Time
  + Communication Time + I/O time
- Thus, optimizing and parallelizing I/O can be important
- Parallel access requires special programming constructs or libraries, such as MPI-IO
  - MPI-IO consists of MPI functions for data reading and writing
  - Each MPI process can read or write a portion of a shared file
- Structured data formats such as HDF5 can help a lot

MPI I/O


Proc 1
Proc 2
Proc 3
Proc 4
File section 1
File section 2
File section 3
File section 4

More on I/O Optimization

- Put no more than 10K files in a single directory
  - Use a structure of subdirectories for holding >10K files
- Avoid metadata intensive operations like `ls -l`
- When programming:
  - If a file only needs to be read, open it read-only
  - Do not open and close files too frequently
  - Assign a subset of cores for handing I/O
  - Give MPI-IO “hints” about how your I/O operations should be tuned

Optimizing for OPA

- Need to define TMI (Tag Matching Interface) drivers and libraries

- Then inform MPI what TMI API it should use, eg:

```
mpirun -genv I_MPI_FABRICS=shm:tmi \
 -genv I_MPI_TMI_PROVIDER=psm2
```

- If your job is highly communicative and fits on 32 nodes or less, tell Slurm to place it on a non-blocking island

Optimizing for Phi

- 72 real cores, 4 threads per core
- 16 GB memory on-die

- Set up your computation with
  - one MPI process per real core;
  - four threads per MPI process

- Some trial and error may be necessary depending on your data size, memory access

High-Throughput Computing

- Many independent serial jobs/tasks that all run at once

- This can still be done efficiently:
  - If jobs are fairly long, submit each onto one core of a shared node
  - Run 24 processes at once on a single node
  - Use GNU parallel to farm out >24 tasks to the cores on a single node
  - Use LoadBalancer to run >24 tasks across multiple nodes using MPI (without having to know anything about MPI!)

High-Throughput Computing

For many more details, see

https://github.com/ResearchComputing/Final_Tutorials/blob/master/EfficientSerialSubmission/EfficientSerial.pdf

Thank you! Questions?

CU-Boulder Research Computing

www.rc.colorado.edu rc-help@colorado.edu