Opportunities for WRF Model Acceleration

John Michalakes
Computational Sciences Center
NREL

Andrew Porter
Computational Science and Engineering Dept.
STFC Daresbury Laboratory
Daresbury, U.K.

WRF Users Workshop

June 26, 2012

Outline

- Improving node speed with accelerators (Intel MIC)
- Improving output speed with parallel I/O and quilting

Acknowledgements

- Intel Corp.
  - Michael Greenfield
  - Antonio Valles
  - Lawrence Meadows
  - Indraneil Gokhale
  - Mark West
  - Brian Rea

Improving Node Speed

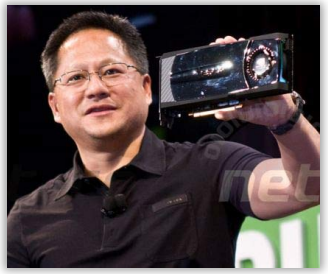
- Increasing performance depends on parallelism
  - Coarse-grained (MPI or “DM”) parallelism between nodes
  - Medium-grained parallelism (OpenMP or “SM”) parallelism between cores on the nodes
  - Fine-grained parallelism
    - Computation on successive elements of arrays on the WRF grid
    - Vector supercomputers lived here
    - Increasing numbers (billions) of transistors on a chip
    - Thousands of floating point units for peak 1 TF/s on a single chip

Cray C90 (1990)
2 GF/s peak/node

INTEL MIC

Intel Many Integrated Core Architecture (2011)
~1000 GF/s peak/device

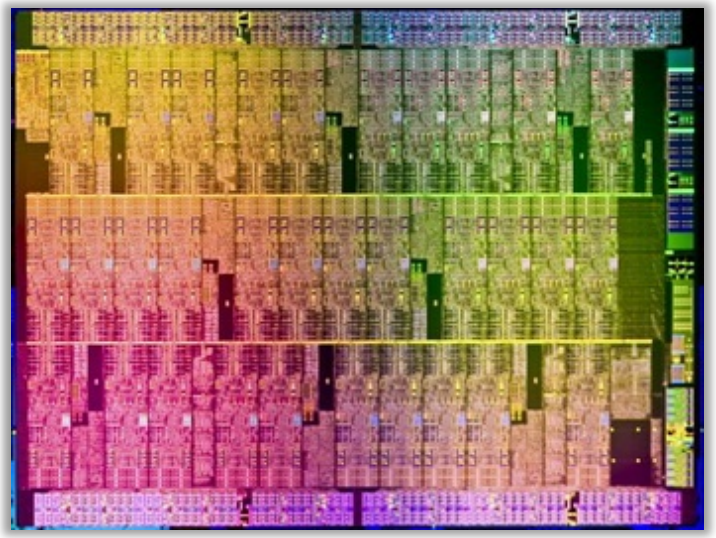
GPU



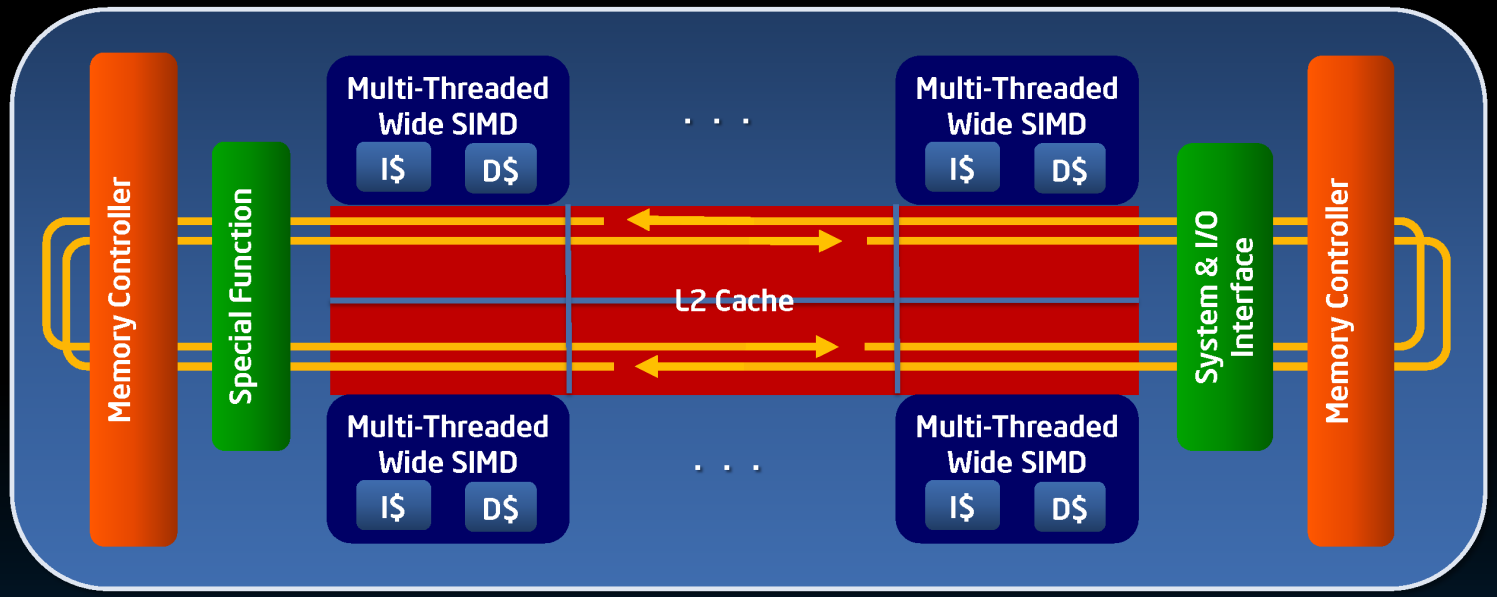
NVIDIA Fermi (2010)
~1000 GF/s peak/device

Many Integrated Core (MIC)

- Teraflop "Cluster on a Chip"
  - 50+ Intel x86 CPUs (cores)
  - Same form factor as GPU
  - Separate co-processor
- Status
  - "Knight's Ferry" pre-production release to NREL and other early users
  - Production version, "Knight's Corner" at NREL and being tested



Intel® MIC Architecture – Knights Family



Multiple IA cores
- **In-order, short pipeline**
- **Multi-thread support**

16-wide vector units (512b)
- **Extended instruction set**

Fully coherent caches

1024-bit ring bus

GDDR5 memory
- **Supports virtual memory**

Standard IA Shared Memory Programming

For illustration only.
Future options subject to change without notice.

http://www.many-core.group.cam.ac.uk/ukgpucc2/talks/Elgar.pdf

Accelerators: GPU and Intel MIC

- In common:
  - Same ~number of transistors, FPUs, peak FP, and on-card memory bandwidth
  - Exist as co-processor with own memory and separate from host by PCI bus
  - Performance issues:
    - Require a lot of application parallelism (WRF has)
    - Require high arithmetic intensity to overcome memory bandwidth and latency bottlenecks (WRF doesn't have)
    - Minimize host-device transfers over PCI bus
  - Programming issues:
    - Offloading work from host to co-processor
    - Code that runs on co-processor

Accelerators: GPU and Intel MIC

- GPU
  - Tflop/s provided by hundreds of simple cores working in lockstep
  - Reliant on host CPU for flow of program control
  - Many threads in flight to hide memory latency
  - About 1MB local store/cache
  - Custom thread programming language (CUDA, OpenCL)

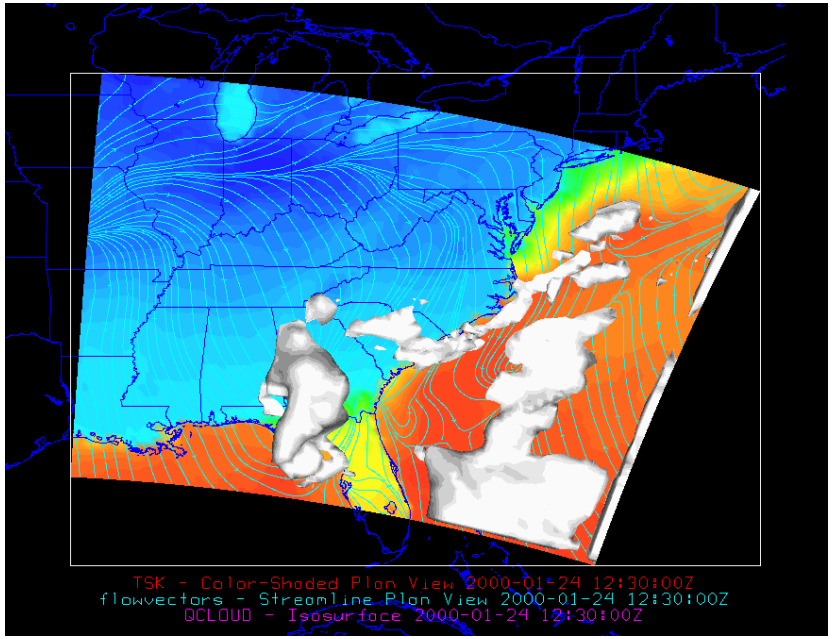
- Intel MIC
  - Tflop/s provided by 50+ traditional CPU cores, each with 16-way vector units (8-way if dbl. prec.)
  - Up to four threads per core to hide memory latency
  - Order 10x more local store/cache than GPU
  - Conventional programming model: program like “cluster on a chip”
    - C, C++, Fortran, etc.
    - Compiler generated vectorization for utilizing SIMD
    - OpenMP and MPI
  - Tuning effort improves host performance

Intel MIC Coprocessor Programming

- WRF ported to Intel MIC Architecture
  - Used standard Intel compilers, MPI, OpenMP, & tools
  - 500K line WRF model ported to Knights Ferry and demonstrated at SC11. (Being shown at ISC '12 this month too).

SC11 Intel demo

- WRF code ported to MIC in 4 days
- Running 1/24/2000 East Coast Winter Storm on MIC
- Using OpenMP threads (already in WRF)
- MPI for multi-MIC runs coming

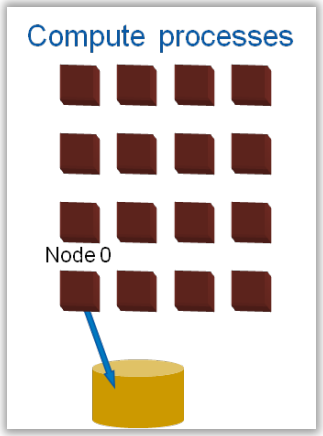


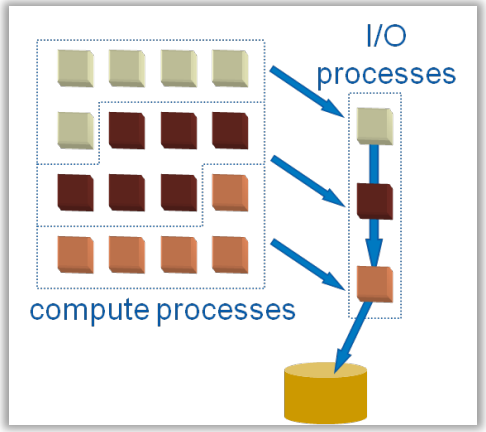
Ongoing/Future work with WRF on Intel MIC

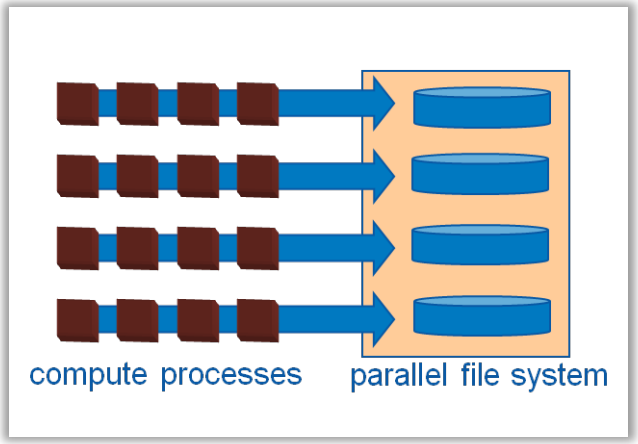
- Adapt and tune atmospheric model kernels to Intel MIC
  - WSM5 microphysics
  - Tracer advection
  - Reactive chemistry
- Whole code performance analysis and optimization
- Apply to other NREL mission-critical codes (OpenFOAM)

Improving WRF I/O

- WRF, like most atmospheric models, is output-bound
  - CONUS 2.5 benchmark generates 4GB per history write
  - Several I/O modes. Examples:
    - Serial NetCDF collected and written from MPI task 0
    - Serial NetCDF written to separate files by each MPI task
    - Serial NetCDF collected and written by gangs of MPI tasks (quilting)
    - Parallel NetCDF written to single files by all MPI tasks



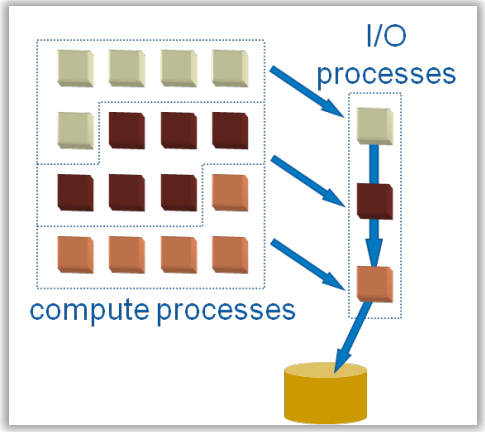


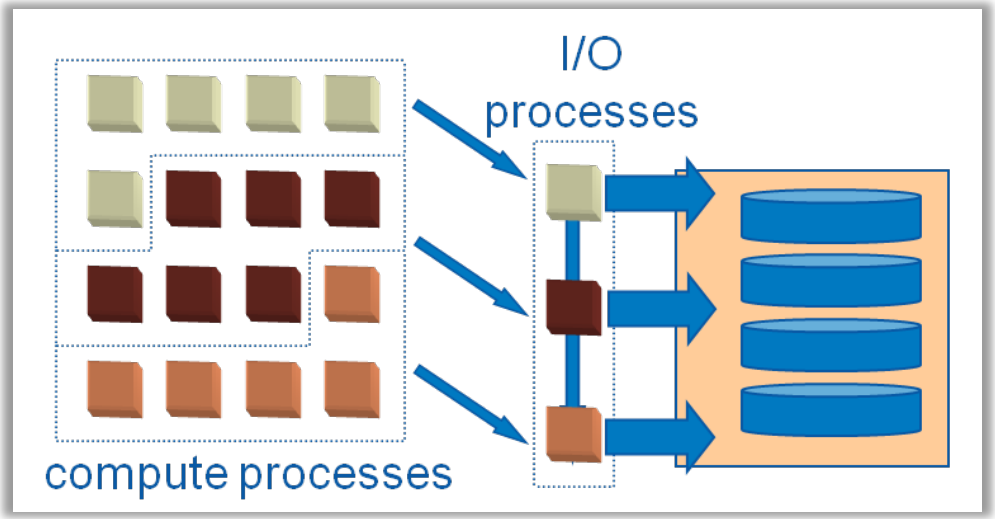


Improving WRF I/O

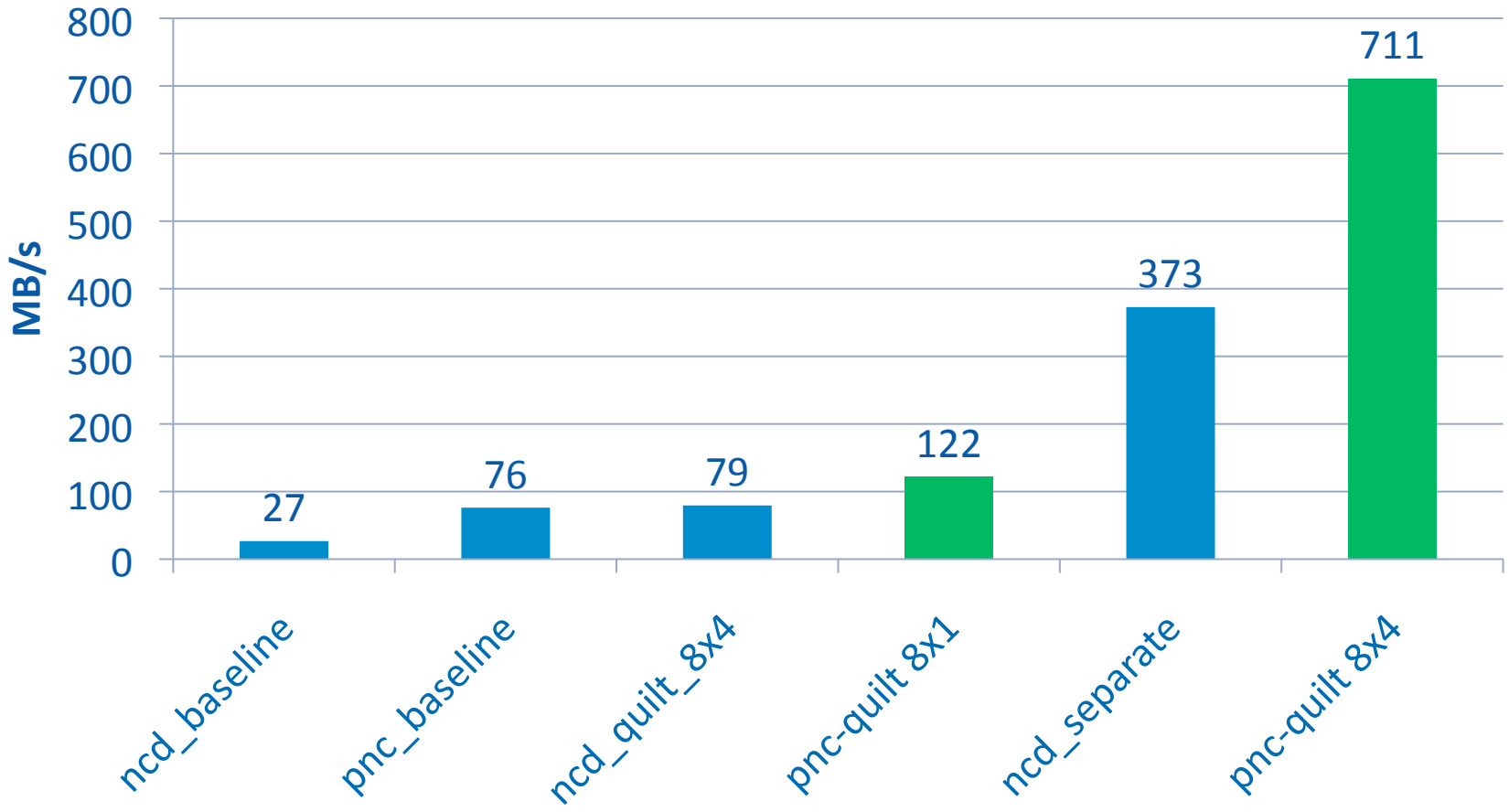
- WRF, like most atmospheric models, is output-bound
  - CONUS 2.5 benchmark generates 4GB per history write
  - Several I/O modes. Examples:
    - Serial NetCDF collected and written from MPI task 0
    - Serial NetCDF written to separate files by each MPI task
    - Serial NetCDF collected and written by gangs of MPI tasks (quilting)
    - Parallel NetCDF written to single files by all MPI tasks in a gang

Andrew Porter

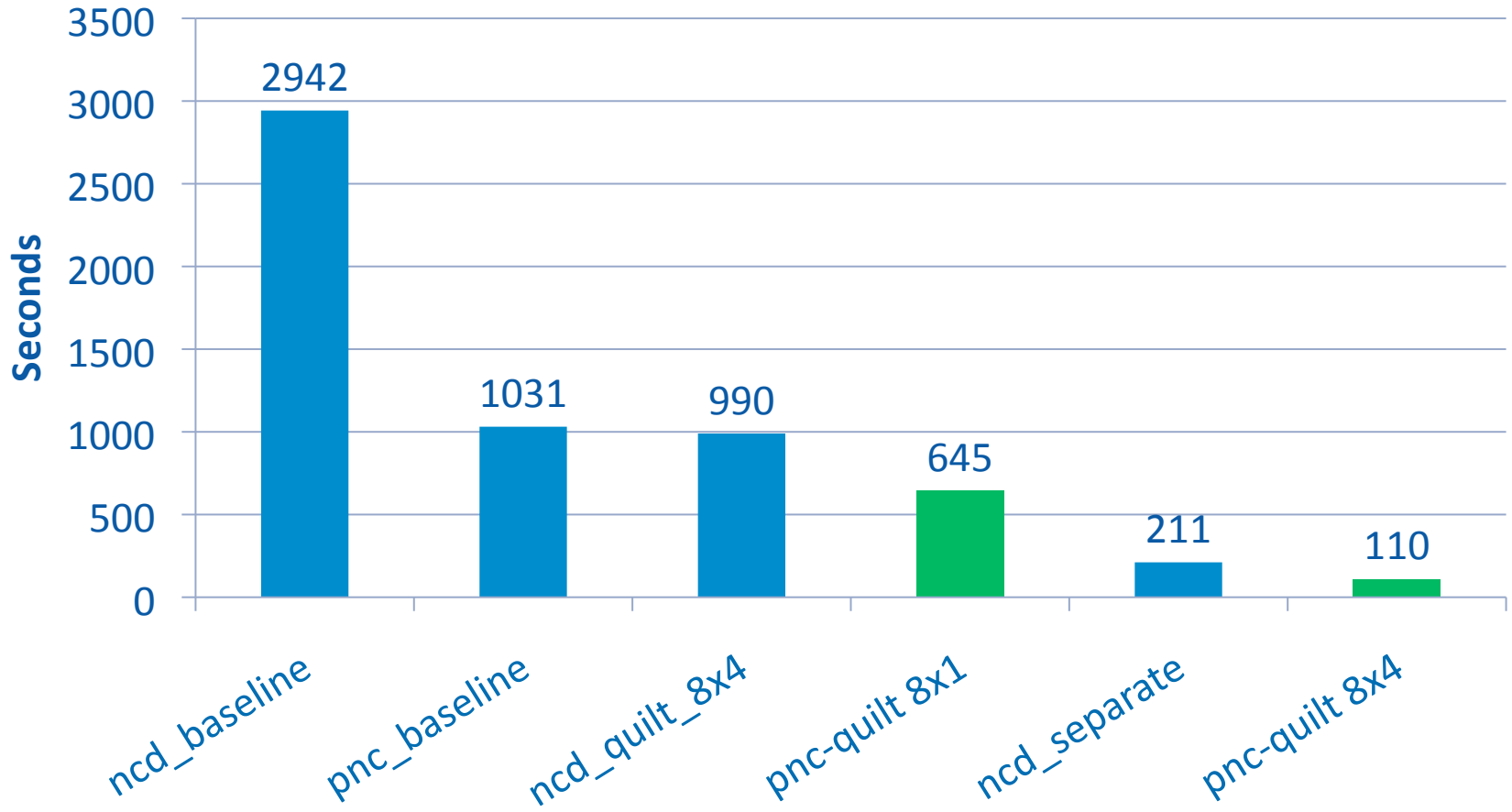




Output performance (MB/s)



Output performance (Time for 17 writes)



Using Parallel NetCDF with Output Quilting

- Prerequisites
  - Parallel file system (Lustre, GPFS, …)
  - Set striping if needed (and possible), e.g.
    - Lustre: lfs setstripe –c 4 *directory*
    - GPFS does not have user level controls
  - Parallel NetCDF (http://trac.mcs.anl.gov/projects/parallel-netcdf/)
- Specifying at compile and run times
  - ./configure with PNETCDF set in shell env. to installation dir. for pnetcdf
  - Edit configure.wrf and
    - modify ARCHFLAGS to add/remove -DPNETCDF_QUILT
  - Touch frame/module_io_quilt.F and frame/module_quilt_outbuf_ops.F
  - Recompile (no need to recompile the whole code)
  - Current limitations.
    - If compiled for pnc and quilt, then these must be used together for output
    - Error in rsl.error file for one of the quilt processes. Possible 32-bit overflow on output server. Try larger nio_tasks_per_group in namelist.

Summary

- Future increases in performance require fully exploiting parallelism, esp. fine grain for increasing node speed
- WRF and atmospheric models:
  - \+ Lots of parallelism on many levels, including vector/SIMD
  - – Low arithmetic intensity and large temporal working sets
  - ? Programmability will be one key to exploiting many- and multi-core devices
- Output performance benefits from combined approach that exploits MPI parallelism, both between compute and I/O tasks, and fully utilizes parallel I/O