PIO: The Parallel I/O Library

Raymond Loy

Leadership Computing Facility /
Mathematics and Computer Science Division
Argonne National Laboratory

With

John Dennis, National Center for Atmospheric Research
Jim Edwards, National Center for Atmospheric Research
Robert Jacob, Argonne National Laboratory

The 13th Annual CCSM Workshop, June 19, 2008

Trends in Climate Model Resolution

- High resolution configuration: 1/10th degree ocean/ice with 0.5 degree atmosphere.
  - Ocean: 3600 x 2400 x 42
  - Sea ice: 3600 x 2400 x 20
  - Atmosphere: 576 x 384 x 26
  - Land: 576 x 384 x 17

- Compared to CCSM3:
  - Ocean: 73x larger
  - Atmosphere: 7x larger

Trends in Climate Model Resolution

- History output sizes for high-resolution configuration *for one write of a single monthly average*
  - Atmosphere: 0.8 GB
  - Ocean: 24 GB (reduced; 100GB if full)
  - Sea Ice: 4 GB
  - Land: 0.3 GB

- Restart output:
  - Atmosphere: 0.9 GB
  - Ocean: 29 GB (96 GB with extra tracers)
  - Sea Ice: 5 GB
  - Land: 0.2 GB
  - Coupler: 6.5 GB

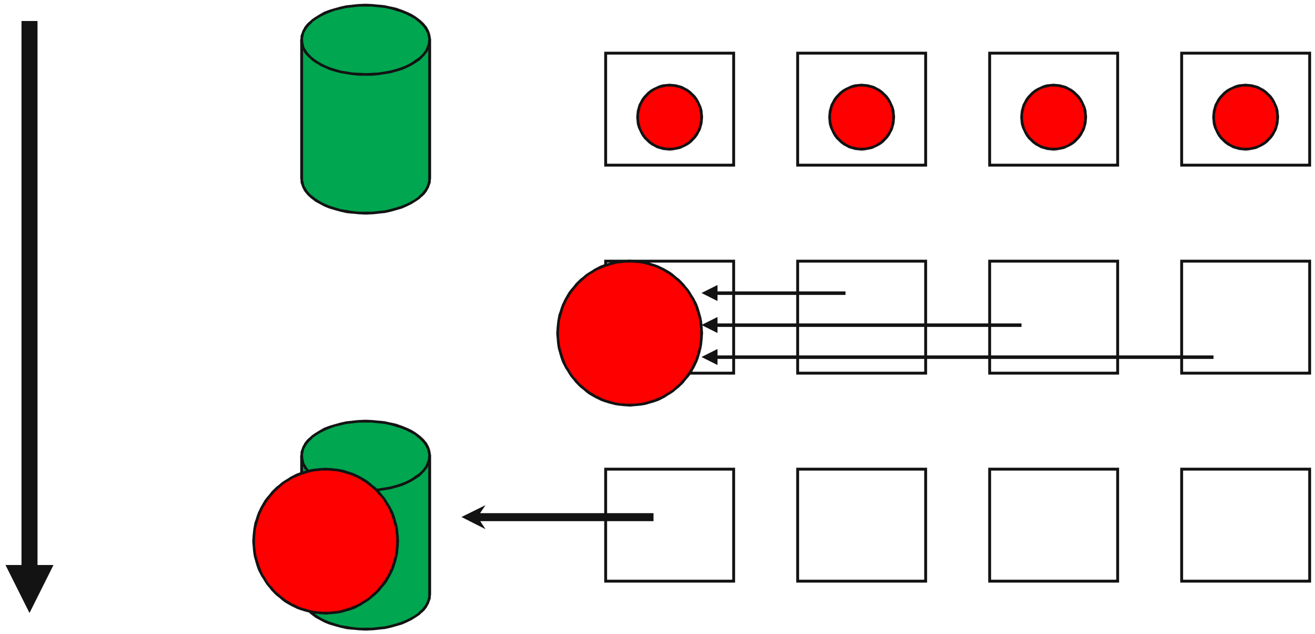
Trends in High Performance Computing systems

- Moore's Law is still increasing transistor count but now they are grouped in to **multiple cores**.

- Memory/core is nearly constant.

- Power/cooling constraints promote design for maximum flops/watt
  - BlueGene: low power nodes; **low memory**
    - *BG/L node: 2 440 PowerPC, 0.7GHz; 512MB* ***(256MB/core)***
    - *BG/P node: 4 450 PowerPC, 0.85 GHz; 2GB* ***(512MB/core)***
  - SciCortex node: 6 MIPS64 cores, 0.5 GHz; (600 mW each!)

Ye Olde Gather/Scatter with Serial Read/Write



Example: gather and write

- As old as the first parallel program
- Still state-of-the-art

Solution: Parallel I/O!

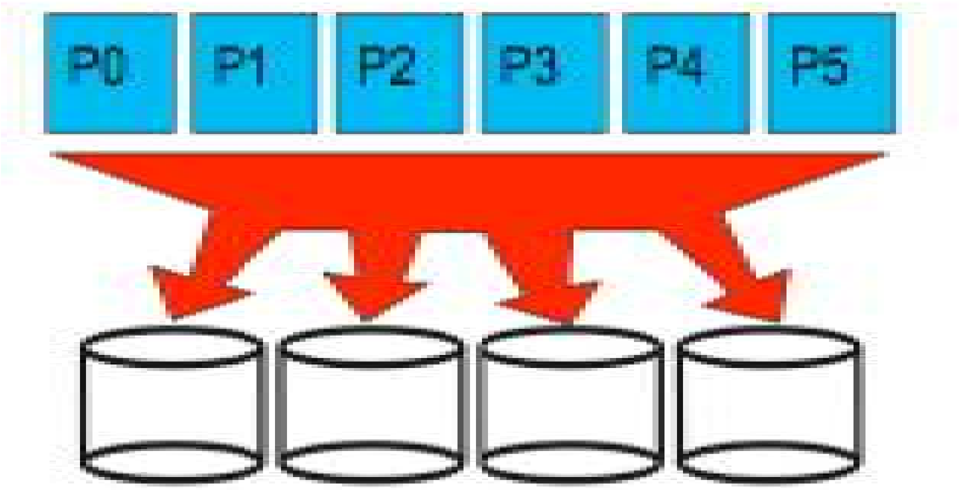


Figure and following general parallel I/O overview provided by **Rob Latham** (Argonne)

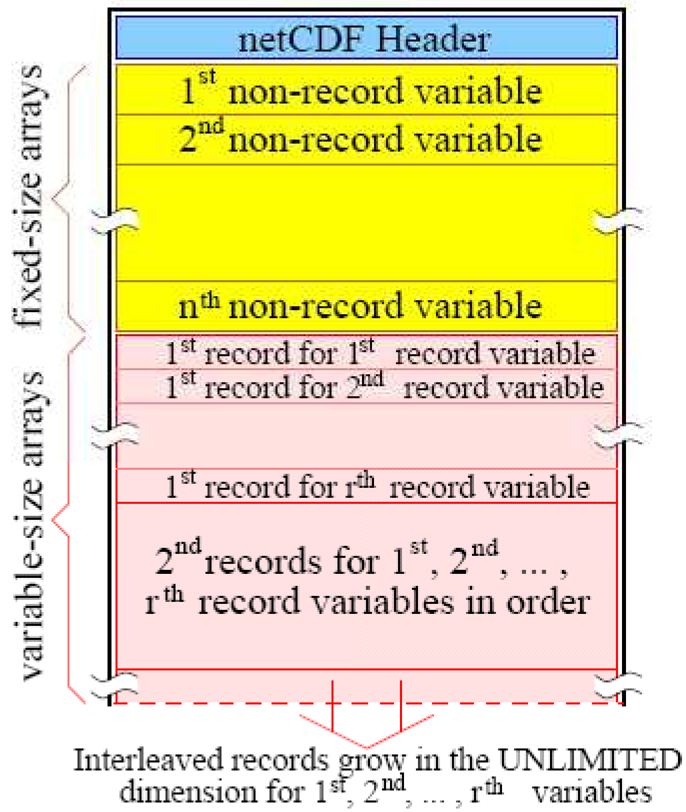
- Parallel I/O beings with hardware and **low-level** software forming a parallel file system
  - Many disks look like one big disk.
  - Related: old parallel I/O method of each processor writing its own file to local disk. Postprocessing needed to complete output.
  - Examples: PVFS, Lustre, GPFS.

MPI-IO

- The Message Passing Interface (MPI) is an interface standard for writing message passing programs
  - Most popular programming model on HPC systems
- MPI-IO is an I/O interface specification for use in MPI apps
- Data model is same as POSIX
  - Stream of bytes in a file
- Features:
  - Collective I/O
  - Noncontiguous I/O with MPI datatypes and file views
  - Nonblocking I/O
  - Fortran bindings (and additional languages)
- Implementations available on most platforms

NetCDF: Standard file format used in climate modeling

- Data Model:
  - Collection of variables in single file
  - Typed, multidimensional array variables
  - Attributes on file and variables
- Features:
  - C and Fortran interfaces
  - Portable data format
- Data is always written in a big-endian format
- NetCDF files consist of three regions
  - Header
  - Non-record variables (all dimensions specified)
  - Record variables (ones with an unlimited dimension)



Parallel NetCDF: NetCDF output with MPI-IO

- Based on NetCDF
  - Derived from their source code
  - API slightly modified
  - Final output is indistinguishable from serial NetCDF file
- Additional Features
  - Noncontiguous I/O in memory using MPI datatypes
  - Noncontiguous I/O in file using sub-arrays
  - Collective I/O
- Unrelated to netCDF-4 work

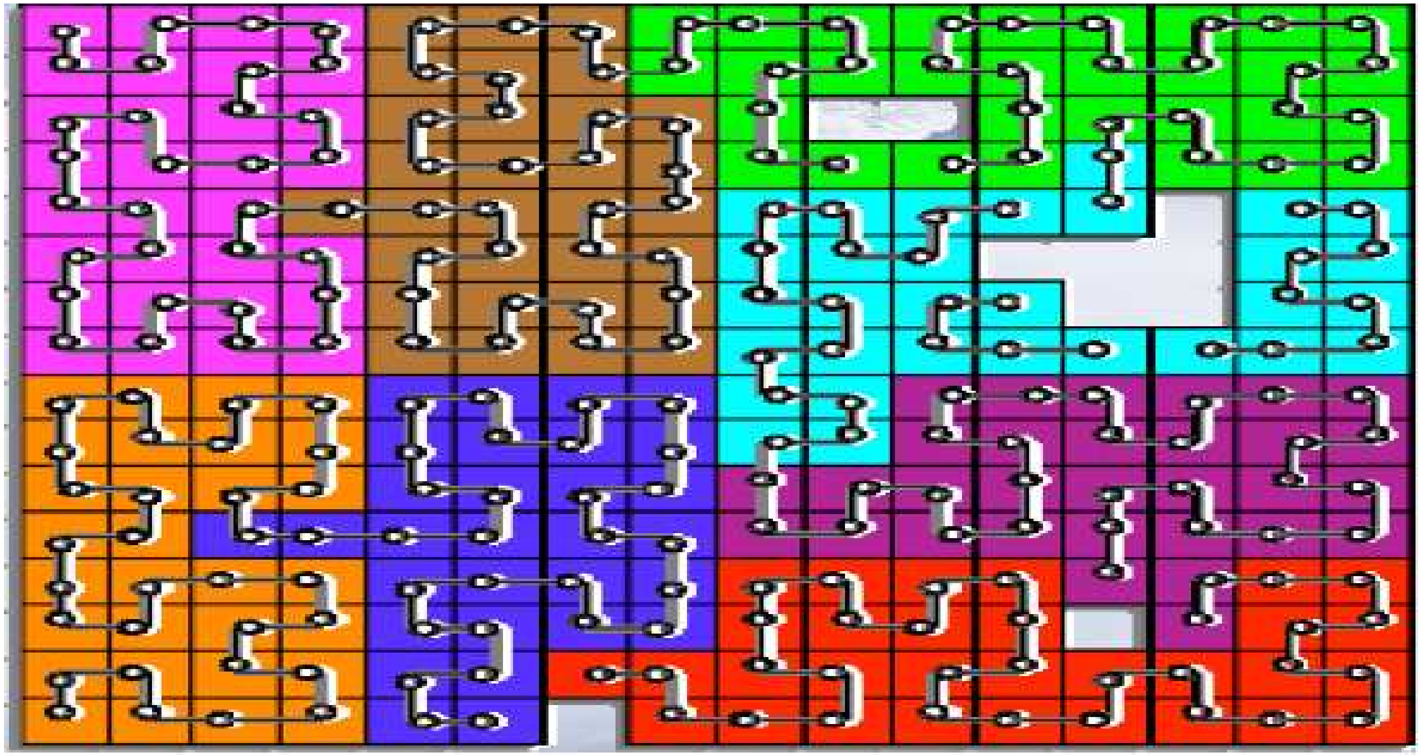
Goals for Parallel I/O in CCSM

- Provide parallel I/O for all component models
- Encapsulate complexity into library
- Simple interface for component developers to implement
- Extensible for future I/O technology

Goals for Parallel I/O in CCSM

- Backward compatible (node=0)
- Support for multiple formats
  - {sequential,direct} binary
  - netcdf
- Preserve format of input/output files
- Supports 1D, 2D and 3D arrays

Climate model decompositions can be complex



Ocean decomposition with space-filling curve

PIO Terms and Concepts:

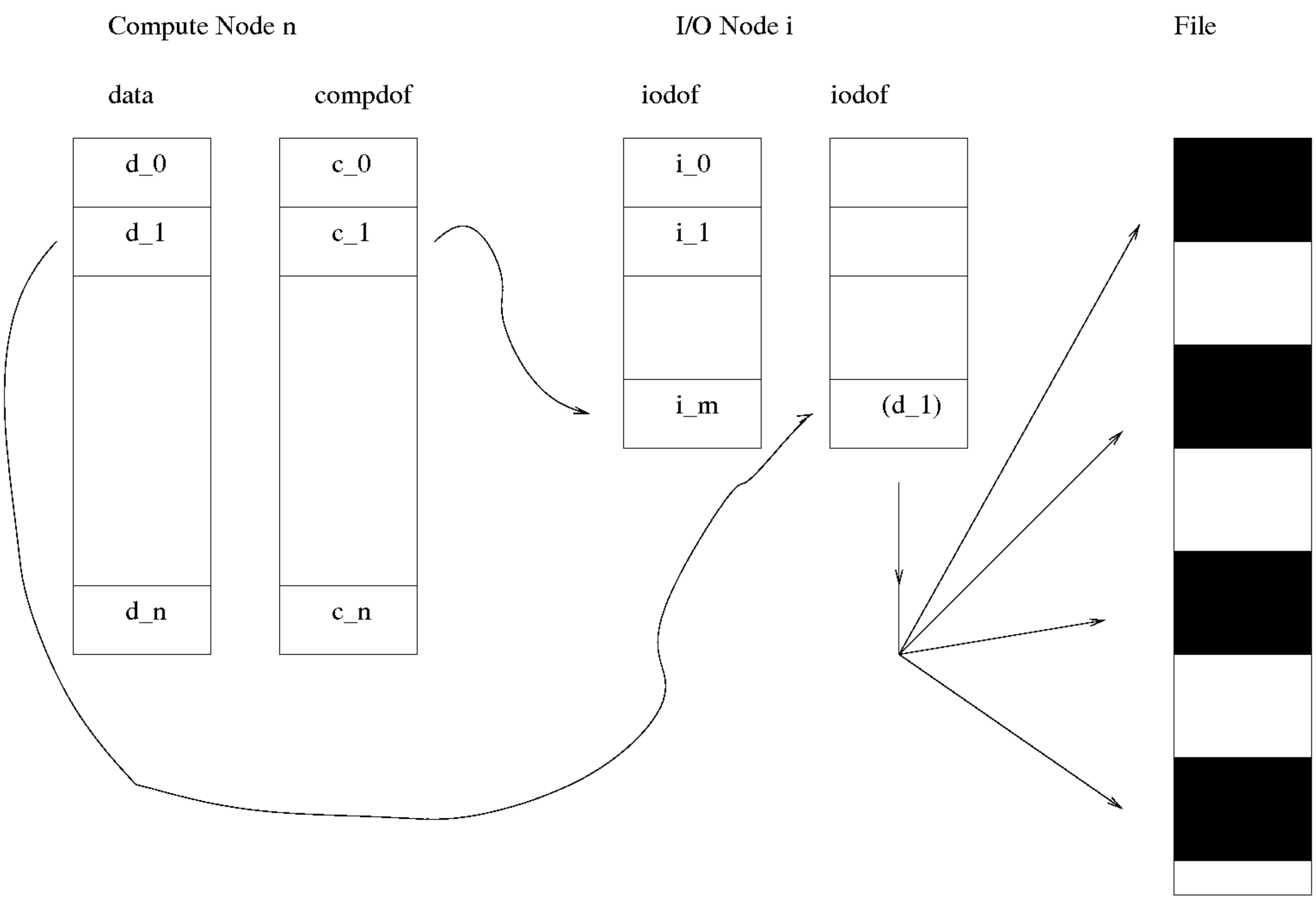
- I/O decomp vs. physical model decomp
  - I/O decomp == model decomp
    - *MPI-IO+ message aggregation*
  - I/O decomp != model decomp
    - *Need Rearranger: MCT, custom*

- No component-specific info in library
  - Pair with existing communication tech
  - 1-D arrays input to library; component must flatten 2-D and 3-D arrays

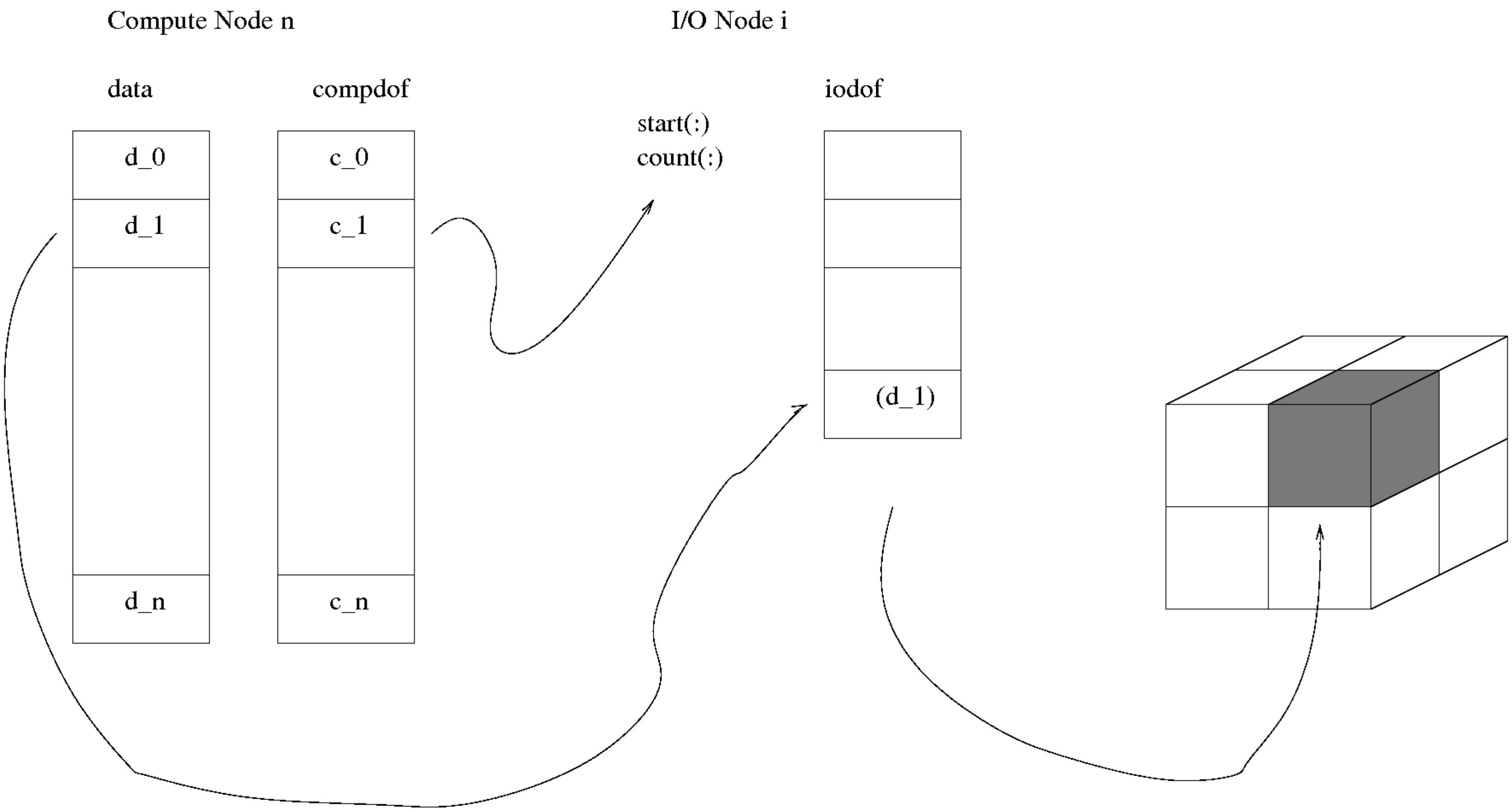
PIO Data Rearrangement

- Goal: redistribute data from computational layout of the model ("*compdof*") to a subset of processors designated for I/O ("*iodof*").
  - Provides direct control of number of procs reading/writing to maximize performance on a platform
  - This level of control not possible with pnetcdf API, also more portable than MPI-IO hints
  - I/O decomposition matched to actual read/write
- Initial method: MCT
  - Pro: MCT Rearranger is general, allows arbitrary pattern
  - Con: Setup is expensive (all-to-all matching); description of the decompositions can be large due to poor compression of small runs of indices
- Improved method: Box Rearranger
  - Netcdf/Pnetcdf reads/writes naturally operate on rectangular "box" subsets of output array variables

Data rearrangement



Box Rearranger



PIO Box Rearranger

- Mapping defined by extents of box for each I/O node
  – Extremely compact representation easily distributed
  – Reverse mapping computed at runtime
- Supports features needed for e.g. ocean vs. land
  – “holes” in computational decomposition
  – fill values for I/O dofs not covered
- Design evolved driven by performance of CAM integration
  – Initial design conserved space by creating send/receive types on-the-fly. *MPI too slow*.
  – Important to performance to cache MPI types and compute reverse mapping up-front during Rearranger creation

PIO API

```
subroutine PIO_init(comp_rank, comp_comm, num_iotasks, num_aggregator,
   stride, Rearranger, IOsystem, base)
      integer(i4), intent(in) :: comp_rank          ! (MPI rank)
      integer(i4), intent(in) :: comp_comm          ! (MPI communicator)
      integer(i4), intent(in) :: num_iotasks
      integer(i4), intent(in) :: num_aggregator
      integer(i4), intent(in) :: stride
      integer, intent(in) :: Rearranger  !defined in pio_types
    + PIO_rearr_none ! pio does no data rearrangment, data is assumed to be in it's final form when passed to pio
    + PIO_rearr_mct ! pio uses mct to rearrange the data from the computational layout to the io layout.
    + PIO_rearr_box ! pio uses an internal rearranger to rearrange the data from the computational layout to the io layout.
      type (IOsystem_desc_t), intent(out) :: IOsystem   ! Output
```

IOsystem stores the context

PIO API

subroutine **PIO_initDecomp**(Iosystem,baseTYPE,dims,compDOF,IOdesc)

```
type(IOSystem_desc_t), intent(in) :: IOsystem
integer(i4), intent(in) :: baseTYPE      ! type of array {int,real4,real8}
integer(i4), intent(in) :: dims(:)       ! global dimensions of array
integer (i4), intent(in) :: compDOF(:)   ! Global degrees of freedom for comp decomposition
type (IO_desc_t), pointer, intent(out) :: IOdesc
```

Automatically computes start(:) and cnt(:) to define the I/O mapping

PIO API

subroutine **PIO_initDecomp**(Iosystem,baseTYPE,dims,lenBLOCKS,compDOF, ioDOFR,ioDOFW,start,cnt,IOdesc)

```
type(IOSystem_desc_t), intent(in) :: IOsystem
integer(i4), intent(in) :: baseTYPE      ! type of array {int,real4,real8}
integer(i4), intent(in) :: dims(:)            ! global dimensions of array
integer (i4), intent(in) :: lenBLOCKS
integer (i4), intent(in) :: compDOF(:)   ! Global degrees of freedom for comp decomposition
integer (i4), intent(in) :: ioDofR(:)       ! Global degrees of freedom for I/O decomp (Read op)
integer (i4), intent(in) :: ioDofW(:)      ! Global degrees of freedom for IO decomp (Write op)
integer (PIO_OFFSET), intent(in) :: start(:), cnt(:) ! pNetCDF domain decomosition information
type (IO_desc_t), pointer, intent(out) :: IOdesc
```

start(:) and cnt(:) define the I/O mapping

PIO API

subroutine **PIO_write_darray**(data_file, varDesc, IOdesc, array, iostat, fillval)

```
type (File_desc_t), intent(inout) :: data_file          ! file information (netcdf or binary)
type (IOsystem_desc_t), intent(inout) :: iosystem ! io subsystem information
type (var_desc_t), intent(inout) :: varDesc           ! variable descriptor
type (io_desc_t), intent(inout) :: iodesc               ! io descriptor defined in initdecomp
intent(in) :: array ! array to be written (currently integer, real*4 and real8 types are supported, 1
                                                          dimension
integer, intent(out) :: iostat ! error return code
intent(in), optional :: fillvalue ! same type as array, a fillvalue for pio to use in the case of missing
                                                                      data
```

Cached I/O mapping and structures reusable for multiple writes/reads (via IOdesc)

PIO API

subroutine **PIO_read_darray**(data_file, varDesc, iodesc, array, iostat)

type (File_desc_t), intent(inout) :: data_file ! info about data file

type (var_desc_t), intent(inout) :: varDesc ! variable descriptor

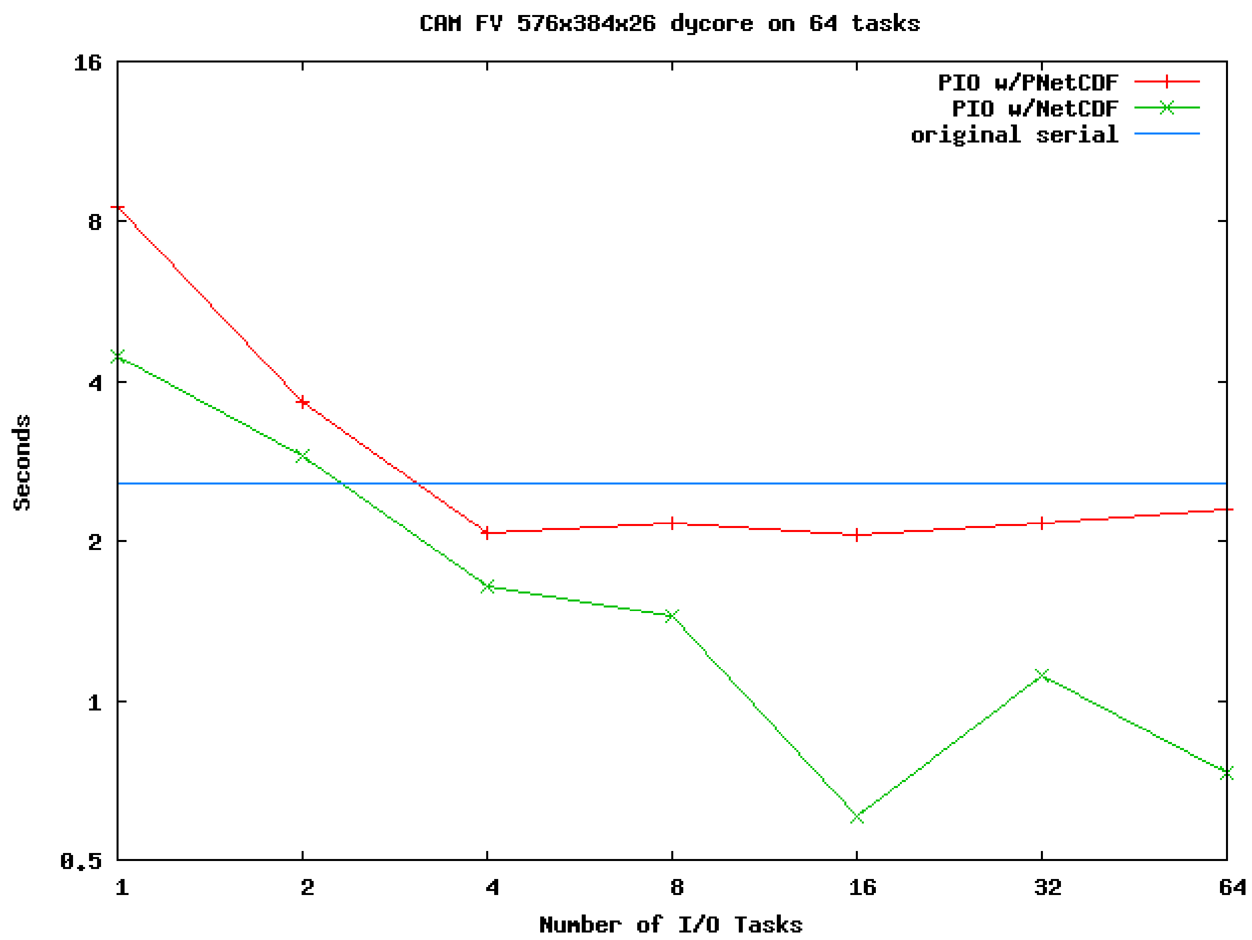
type (io_desc_t), intent(inout) :: iosystem

intent(in) :: array ! array to be read currently integer, real*4 and real8 types are supported, 1 dimension)

integer, intent(out) :: iostat ! error return code

No fillval needed in this direction (holes not modified)

PIO in CAM



CAM FV 576x384x26 dycore on 64 tasks
PIO w/PNetCDF
PIO w/NetCDF
original serial
Seconds
16
8
4
2
1
0.5
1
2
4
8
16
32
64
Number of I/O Tasks

PIO Success Stories

- PIO implementation in CCSM
  - Atmosphere: read and write history and restart; all dycores
  - Ocean: read and write history and restart
  - Land: write history
  - Sea Ice and Coupler: in progress

- PIO being used in high-resolution coupled model.

- Backwards-compatible NetCDF mode has value-added
  - Rearrangement to IO proc subset followed by gather/write one piece at a time.
  - Avoids overflowing memory of root processor

PIO success stories

- High resolution atmosphere model test cases with the HOMME dynamical core.

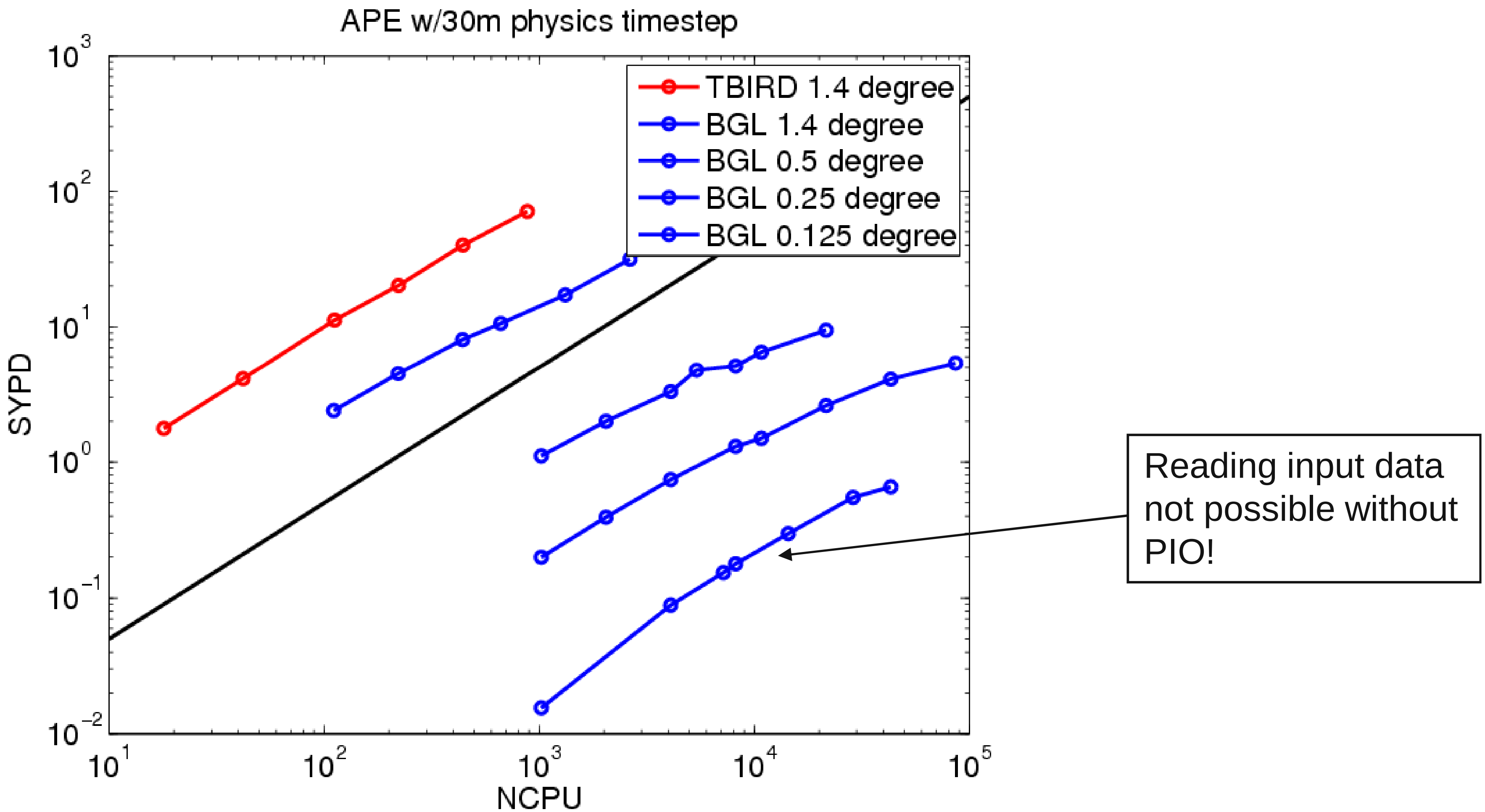


Figure provided by Mark Taylor, Sandia National Lab

CAM-HOMME on BG/P

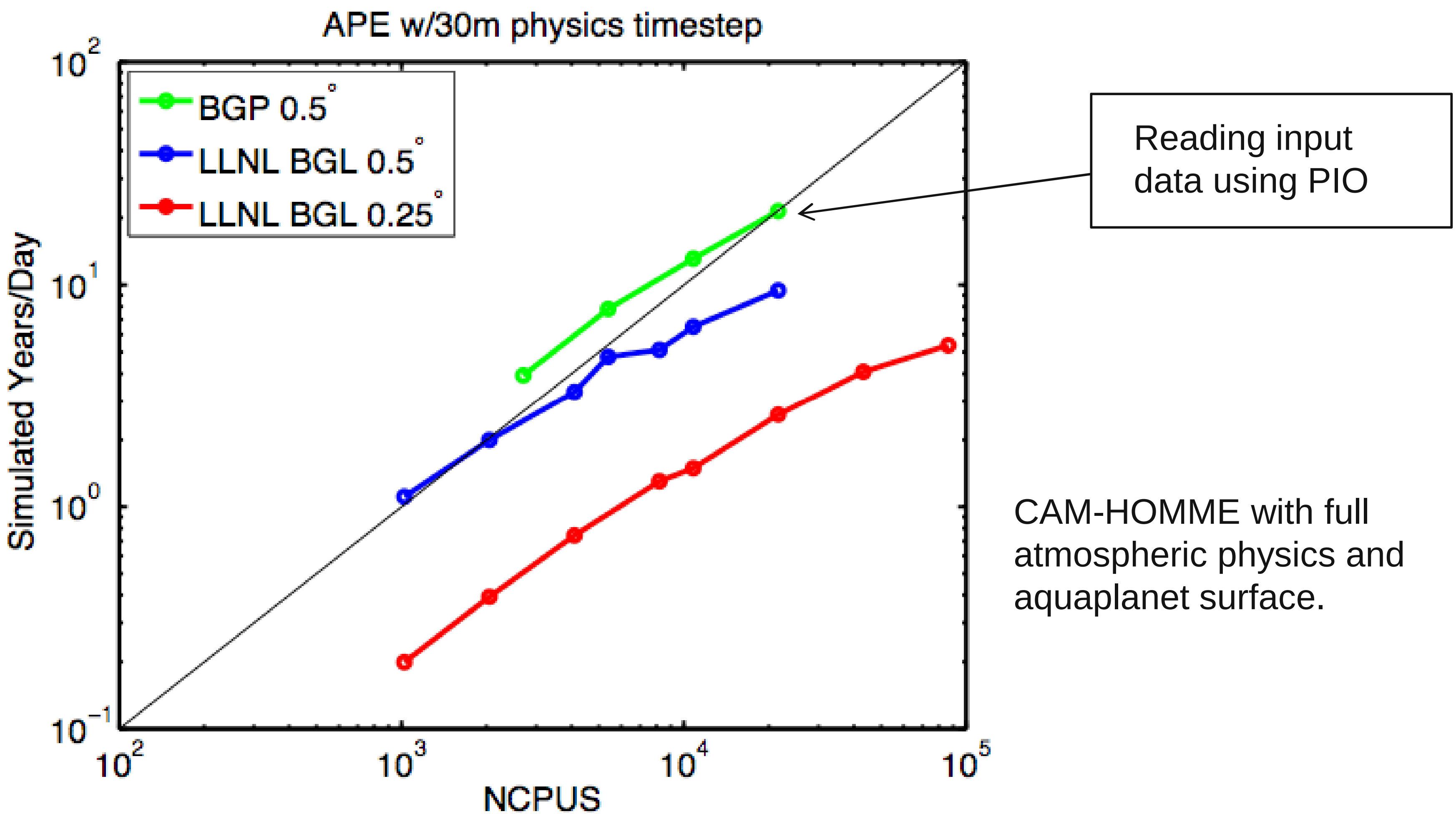


Figure provided by Mark Taylor, Sandia National Lab

CAM-HOMME with full atmospheric physics and aquaplanet surface.

PIO Deployment

- Developed configure system for portability across all CCSM platforms and sites.
  - Supports a large set of options (Enable/disable MCT,Parallel NetCDF,NetCDF,MPI-IO,serial compatibility,MPI-2,diagnostic modes,...)
- In current use on
  - Argonne BG/L, Intrepid (BG/P), Jazz (Intel,Linux)
  - Blueice (Power5+,AIX), Bangkok (Intel,Linux)
  - Jaguar (Opteron,XT4)
  - Sandia cluser (Intel+Infiniband)
- PIO currently developed within CCSM repository
  - Transitioning development to Google Code

Future work

- Clean up documentation
- More unit tests/ system tests
- Understanding performance across zoo of parallel I/O hardware/software
- Add to rest of CCSM
- You will soon be able to download, use and help develop PIO!
  - http://code.google.com/p/parallelio