Argonne NATIONAL LABORATORY

… for a brighter future

U.S. Department of Energy

UChicago ▶ Argonne LLC

Office of Science U.S. DEPARTMENT OF ENERGY

A U.S. Department of Energy laboratory managed by UChicago Argonne, LLC

Update on PIO: The Parallel I/O Library

Robert Jacob, Argonne National Laboratory

(telling you about work done by….)

Tony Craig, National Center for Atmospheric Research

Ray Loy, Argonne National Laboratory

Jim Edwards, National Center for Atmospheric Research

Art Mirin, Lawrence Livermore National Laboratory

John Dennis, National Center for Atmospheric Research

Jay Larson, Argonne National Laboratory

14th Annual CCSM Workshop, June 17, 2009

Motiviation: 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 Output

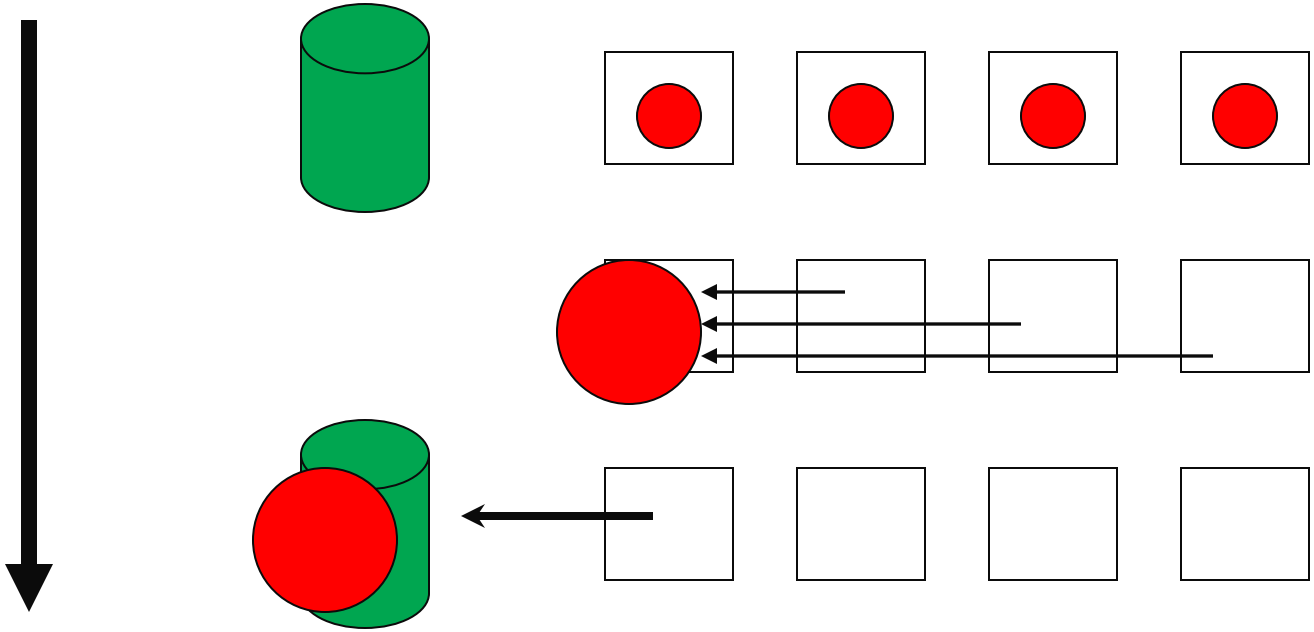
- History output sizes for high-resolution configuration *for one write of a single monthly average*
  - Atmosphere: 0.8 GB
  - Ocean: 24 GB (reduced; 100GB if full)
  - Sea Ice: 4 GB
  - Land: 0.3 GB

- 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

Versus: 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 Power PC, 0.7GHz;* ***512MB/node***
    - *BG/P node: 4 450 PowerPC, 0.85 GHz;* ***2GB/node***

Ye Olde Gather/Scatter with Serial Read/Write



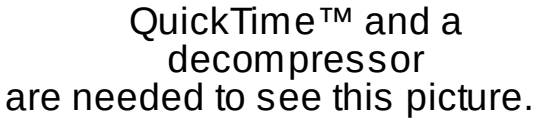
Example: gather and write

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

So..How does one gather 30GB of data on to a 2GB node for writing?

- Serialization: send one piece at a time and write it out. Slows down with more nodes or more data/node.
- Each node writes its own file: Then you have new post-processing headache of gathering up all the individual files into one file.
- Use nodes with more memory! Even large-memory nodes will be overrun by output from future models.

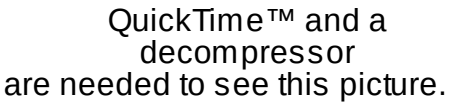
Solution: Parallel I/O!



(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.
  - Examples: PVFS, Lustre, GPFS.

Software for Parallel I/O



- Applications require more software than just a parallel file system
- Support provided via multiple layers with distinct roles:
  - Parallel file system maintains logical space, provides efficient access to data (e.g. PVFS, GPFS, Lustre)
  - I/O Forwarding found on largest systems to assist with I/O scalability
  - Middleware layer deals with organizing access by many processes(e.g. MPI-IO, UPC-IO)
  - High level I/O library maps app. abstractions to a structured,portable file format (e.g. HDF5, Parallel netCDF)
- Goals: scalability, parallelism (high bandwidth), and usability

I/O presentation from Rob Latham (Argonne National Lab)

MPI-IO

- 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

PNetCDF: 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 a single interface to multiple I/O options: pnetcdf, binary with MPI-IO, “plain” netcdf.
- General enough to be used as I/O interface in all CCSM components.
- Simple interface for component developers to implement.
- Extensible for future I/O technology
- Preserve format of input/output files
- Supports 1D, 2D and 3D arrays

PIO Terms and Concepts:

- IO decomposition vs. physical model decomp
  - IO decomp == model decomp
    - *MPI-IO+ message aggregation*
  - IO decomp != model decomp
    - *Need to rearrange data from one decomposition to another.*

- No component specific info in library
  - Pair with existing communication technology
  - 1D arrays in library; component must flatten2D and 3D arrays

PIO Terms and Concepts:

- 3 modes of writing within PIO
  - PIO + MPI-IO: ("binary mode"). Call MPI-IO directly to write binary files
  - PIO + NetCDF ("netcdf mode" or "serial mode"). Gather pieces of data to node 0 (or other designated node) one piece at a time for writing netcdf files.
  - PIO + PNetCDF ("pnetcdf mode"). PIO calls PNetCDF directly to do parallel write of netcdf file.
- "degrees of freedom" or DOF = decomposition

PIO status at last workshop:

- PIO API defined
- API implemented in CAM (all dycores with input and output) on branch
- New serial netcdf mode avoids memory bottleneck through serializing the data gather.
- Early performance successes: I/O was faster as I/O processors were added
- Early feasibility success: Some high-res HOMME runs not possible without using PIO for reading input.
- Set up PIO Google group with publicly readable email list. Also set up repo at GoogleCode

Progress since last workshop:

- Pushing PNetCDF uncovered some memory leaks. Fixes released in PNetCDF 1.0.3 (December, 2008)

- Reorderd PIO file and directory structure so it can more easily build as both a component of CCSM or CAM and as a standalone library.

- PIO has made more progress in component models:
  - CAM: now a runtime option for history and restart on main versions of CAM (no longer just on a branch)
  - CPL7: PIO is default method for I/O.
  - CICE: implementation is nearly complete
  - POP and CLM: implementations using older versions of PIO need to be revisited or replaced.

Progress since last workshop:

- Flow control and throttling added to box-rearranger communication to improve performance and work around limitations in some MPI implementations.

- New I/O decomposition algorithm provides more optimal decomposition.

Major new feature: Comprehensive, extendable test suite

- Two methods for setting up a grid on which to test I/O:
  - Very flexible methods for setting a 3D rectangular grid of arbitrary size and regular decompositions of that grid in 1, 2 or 3 dimensions OR
  - Write out decomposition from real code (CAM, POP) and read it back in for testing (pio_writedof, pio_readdof)!

- Grid, decomposition and I/O method to test all specified in a namelist.

- Test program testpio.F90:
  - Reads namelist
  - Generates test data (integers, 4-byte reals and 8-byte reals)
  - Writes data with specified method (binary, netcdf, pnetcdf)
  - Reads data back and checks for correctness
  - Times reads and writes.

Major new feature: Comprehensive, extendable test suite

- Testpio directory already provides 47 namelists for testing various configurations.

- Namelist naming convention: testpio_in.nnXX
  - nn = “pn” = “pnetcdf with no rearrangement”
  - nn = “sb” = “serial netcdf with box rearrangement”
  - XX = 02 = “simple 2d xy decomp across all pes with all pes active in I/O”
  - XX = 04 = “simple 2d xy decomp with yxz ordering and stride=4 pes active in I/O

- Run scripts available for all CCSM target platforms (bluefire, jaguar, intrepid, franklin, kraken)

- Testpio used mostly for testing before checkins. Timings being done with CCSM and components.

PIO Recent Results:

- PIO is a success as an *enabling technology*: Allows simulations never before possible.
  - FV 0.25 degree with trop_mam3 chemistry on jaguar
  - FV 0.25 degree with trop_mozart chemistry and a total of 399 tracers on jaguar.
- PIO timings: continue to be mixed
  - PIO with MPI-IO is always fast. 1-2GB/s read/write obtained.
  - PIO with serial netcdf surprisingly good for some cases. Around Nov, 08, was always faster then pnetcdf.
  - PIO with pnetcdf has shown improvement in past month:
    - *FV 2-degree: pnetcdf now twice as fast as netcdf*
    - *HOMME on 512 BG/P nodes: pnetcdf now 100x faster than netcdf*
    - *FV 0.5 degree with trop-mozart: pio 2x faster than native.*

PIO Recent Results: Dealing with the zoo

- Example of odd timings:
  - 0.25 HOMME on Blackrose (intel linux cluster with inifinband and Lustre) using PIO+pnetcdf. Run on 128 cpus with different strides for I/O processors
    - *32: 21 MB/s*
    - *64: 102 MB/s*
    - *128: 32 MB/s*
- Strange animals and deep magic:
  - Lustre: PIO performance was improved when Lustre upgraded from 1.4 to 1.6 but reproducible performance still elusive.
  - BG: Need to add “bglockless:” to filename to avoid NFS-type file locking.
  - Bluefire: task binding degrades PIO performance. Use MP_TASK_AFFINITY=CORE instead.
  - MPI-IO hints: different hints needed for different hardware.

PIO and the broader community

- PIO currently developed within CCSM repository.

- Latest version of code **now available** at Google Code: parallelio.googlecode.com
  - Tags in CCSM repo are exported by PIO developers and then imported to Google Code.
  - Anonymous checkout allowed by anyone.
  - Checkin limited to “project members”. Membership can be extended to external users who would like to contribute.
  - Merging any contributed code back to CCSM repo up to CCSM PIO developers.

PIO future work

- Add more unit tests based on real decompositions
- Understanding performance across zoo of parallel I/O hardware/software. (John Dennis has two summer students who will be working on this.)
- Add to rest of CCSM
- Ask other application areas to try PIO and provide feedback.