Beyond Embarrassingly Parallel Big Data

William Gropp
www.cs.illinois.edu/~wgropp

Messages

- Big is ***big***
  - Data driven is an important area, but not all data driven problems are big data (despite current hype). The distinction is important
  - There are different measures of big, but a TB of data that can be processed by a linear algorithm is not big
- Key feature of an extreme computing system is a fast interconnect
  - Low latency, high link bandwidth, high bisection bandwidth
  - Provides fast access to data everywhere in system, particularly with one-sided access models
    - Think map(r1,r2, …) – function that requires more than one record, where the specific input records are unpredictable (e.g., data dependent on previous result)

Messages (2)

- I/O operations must reflect data objects, access patterns, latency tolerance, consistency
  - ♦ Uncoordinated I/O is easy to program but costly in performance and correctness
  - ♦ "Bulk Synchronous" style easy to program but costly in performance
- This is a talk about highly scalable parallel I/O and how extreme system capabilities may differ from other systems
  - ♦ See other talks for great things to accomplish with big data and extreme computing

My Research Areas

- Scalable algorithms
  - Communication optimizations
  - Latency tolerance
  - Performance analysis and modeling
- Programming models and systems for parallel computing at scale
  - MPI standard design (e.g., MPI-3 RMA)
  - MPICH: Algorithms and system design for implementation
  - Hybrid programming, esp. coordination of resources
  - Decoupled execution models and programming systems
- Exploit hierarchical, collective, and dynamic features
  - PETSc: Domain decomposition in scalable numerical algorithms
  - pnetCDF: Collective I/O in interoperable data models
  - MPI Slack: light-weight, locality-sensitive, communication-informed load balancing

Blue Waters Computing System

10/40/100 Gb Ethernet Switch

IB Switch

>1 TB/sec

120+ Gb/sec

100 GB/sec

WAN

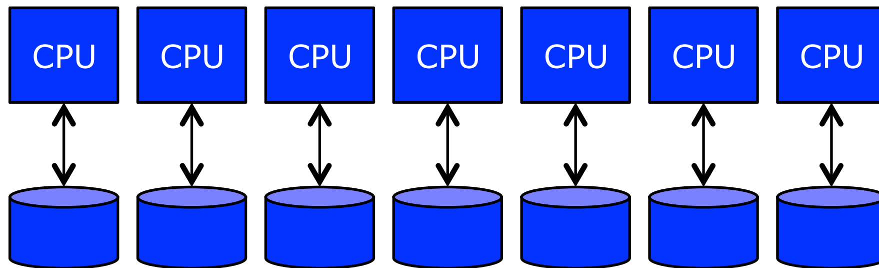
Spectra Logic: >300 PBs

Sonexion: 26 PBs

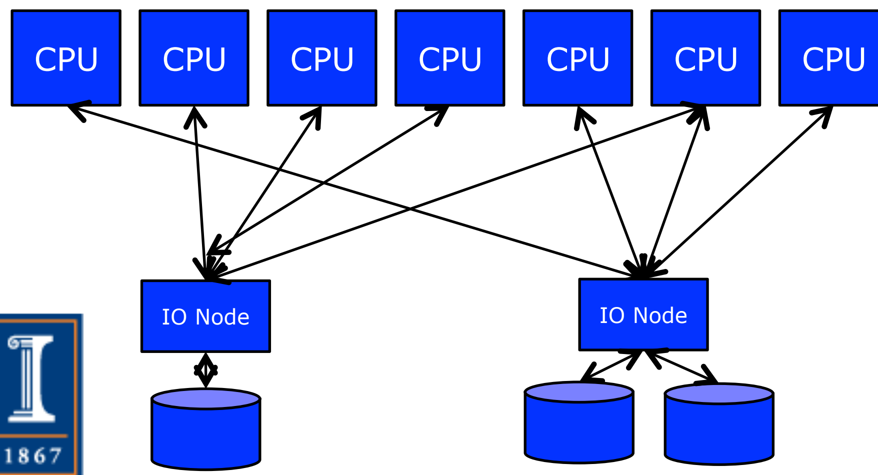
PARALLEL@ILLINOIS

Independent and Collective I/O

Independent I/O Abstraction



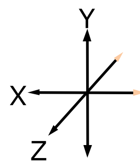
Independent I/O Reality



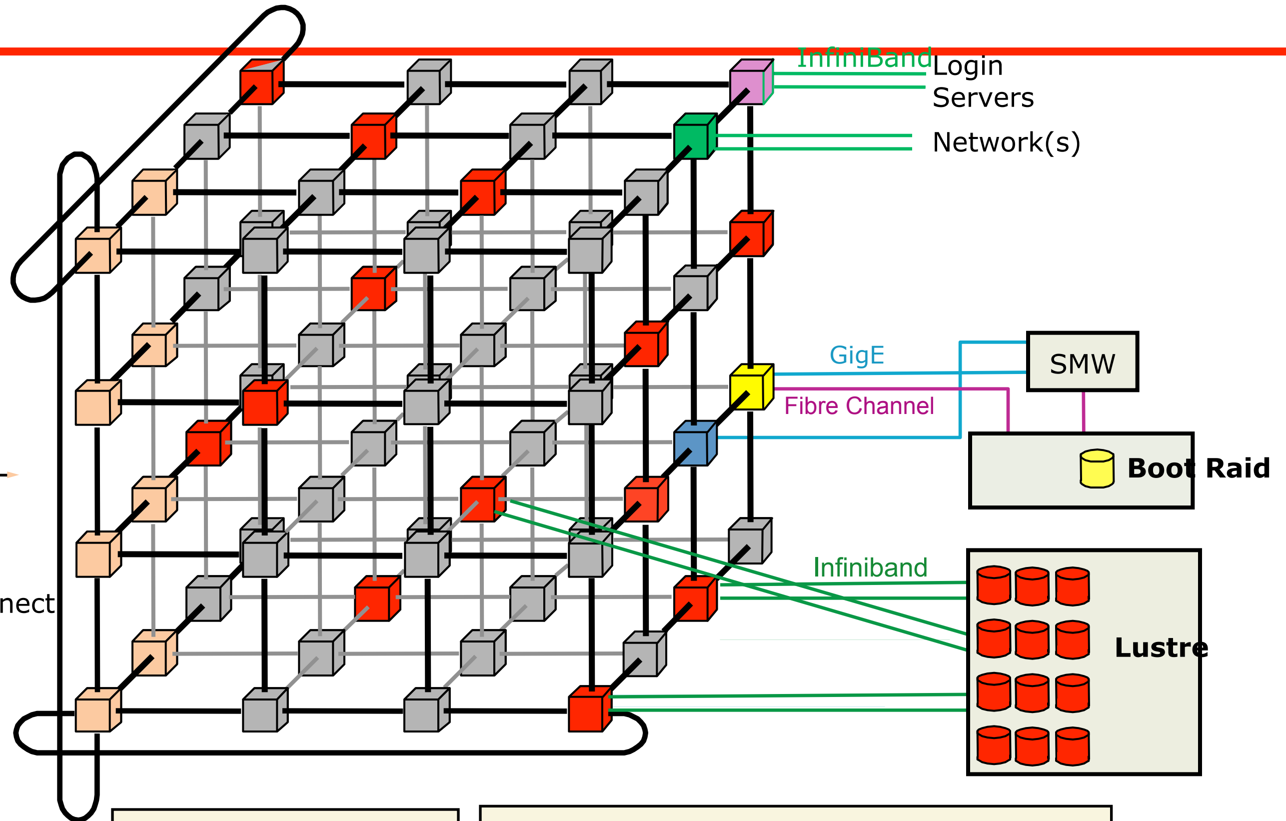
- Independent I/O
  - Processes/threads/tasks write to I/O system without coordinating with others in same parallel job
- Collective I/O
  - Processes/etc coordinate to make access efficient
- Sophisticated caching and forwarding strategies can improve performance, but adds complexity, cost, energy

Gemini Interconnect Network

Blue Waters
3D Torus
Size
23 x 24 x 24



Interconnect
Network



Compute Nodes
- Cray XE6 Compute
- Cray XK7 Accelerator

Service Nodes

Operating System
- Boot
- System Database

Login/Network
- Login Gateways
- Network

Lustre File System
- LNET Routers

Service Nodes spread throughout the torus

Cross-Cutting Themes

- Latency
  - ♦ All levels of memory hierarchy
  - ♦ Strategies (Algorithms and Implementations)
    - Active (Prefetch)
    - Passive (Latency tolerant/overlap)
- Representation of data
  - ♦ Fields (data is discrete approximation to continuous field)
  - ♦ Graphs and other discrete data
  - ♦ Choice has a strong impact on performance **and** productivity
- Access to data and consistency
  - ♦ Independent access is convenient but with penalties in performance and correctness
- Performance modeling and performance/flexibility tradeoffs
  - ♦ E.g., collective I/O
  - ♦ Quantify design and evaluation

PARALLEL@ILLINOIS

Taxonomy: How to define “Big”

- All of memory or more (size)
- As fast as or faster than I/O Bandwidth (velocity)
- Too complex to process
  - Computation of data is not linear in data size
  - Variety of data formats, representation, and models requires experts to grok
- Leaves out: embarrassingly parallel data (nearly independent records)
  - Many “mid size” data
  - Many “MapReduce” applications (important, but others leading here)
- Need some clear examples of the different kinds of workflows (the NAS PB of Big Data) that illustrate different needs
  - High velocity – real time filter/compression; lossy. Large scale instruments (SKA, LSST); ubiquitous low-quality sensors
  - Large numbers of nearly independent records – web, financial transactions, twitter feeds. MapReduce and slightly better; cloud platforms, Databases; large scale instrument data (images)
  - Large single records; highly and unpredictably correlated data. Simulation results, large-scale graphs

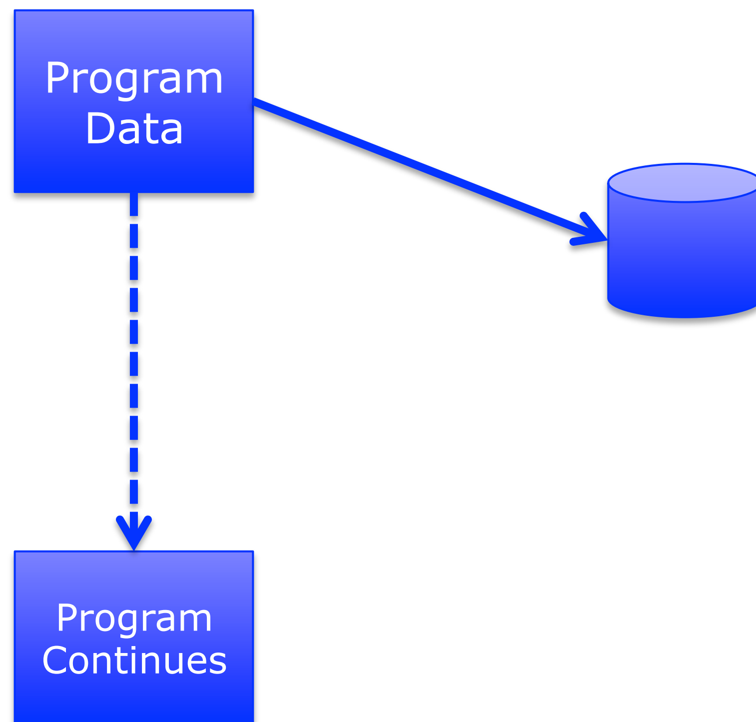
Some Architecture Issues for Big Data

- Parallelism in I/O
  - Systems optimized for zillion independent files or records can use cloud resources
  - Deeper hierarchy in I/O system
    - BW example: 26 PB disk, 380 PB tape with 1.2 PB cache for the 26 PB cache; use of RAIT to improve performance, reliability
    - Important distinction for extreme scale systems: All data accessible at nearly same performance from ***all*** nodes
  - Metadata design has a major impact on performance, reliability
- Other architectural features important
  - One-sided access with remote operations
    - At least multi-element compare-and-swap
    - Even better, compute to data (active messages, parcels, …)
  - And others (better stream processing, custom control logic…)

Workflows

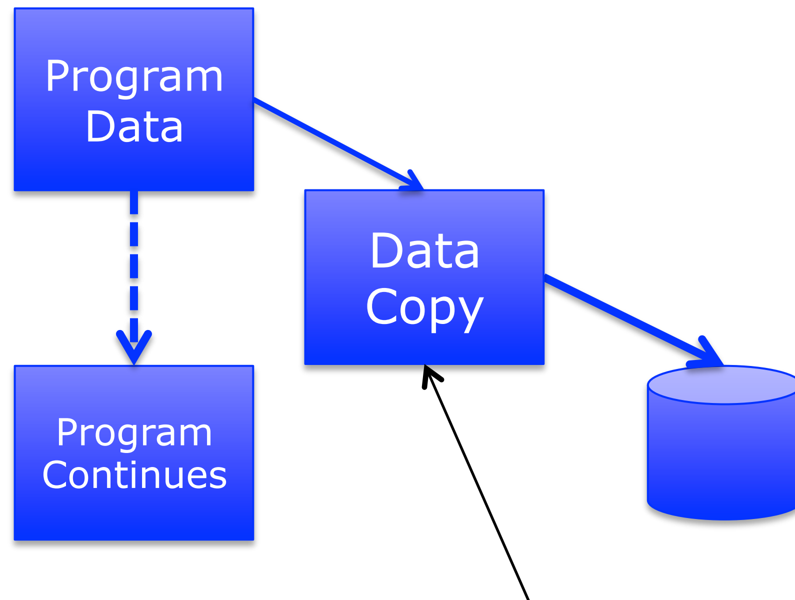
- Simulation reads input data, performs simulations (perhaps ensembles, which may be computed cooperatively), writes results
  - ♦ Bulk synchronous vs. data flow
- Challenge: exploring data. E.g., many data sets now map to multiple value per pixel, even for 2-d slice of 3-d data
  - ♦ Many data sets represent unstructured data (e.g., unstructured mesh); access not easy to precompute, data dependent
    - Read data about mesh (transfer data trom file to processors)
    - Compute data region to access, issue I/O requests
    - Read data with values
    - Operation is doubly bad: requires two separate I/O operations and has strong data dependency
  - ♦ Data representation can make a huge difference in performance

Common Simulation I/O Pattern



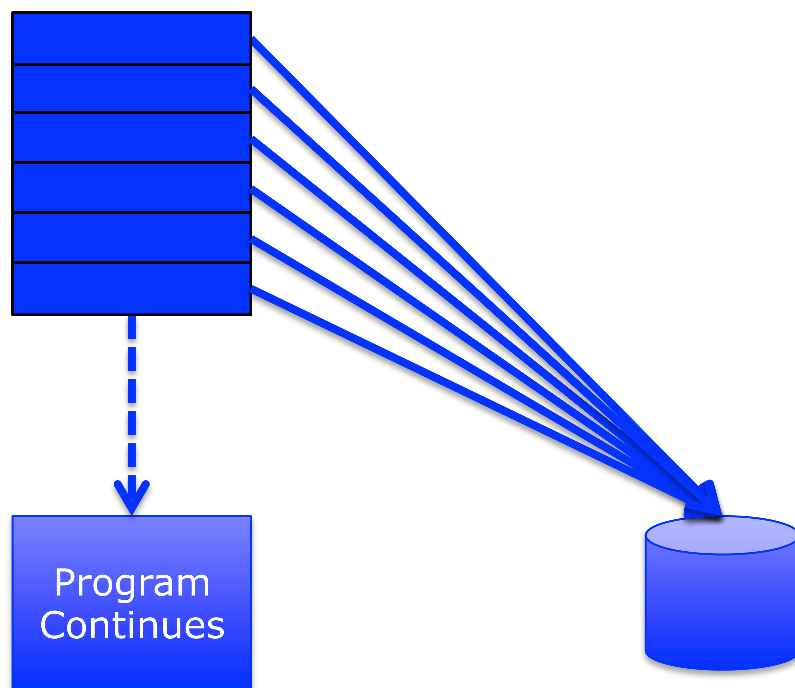
- Program writes data, waits for data to be "written"
  - Data may be in I/O buffers
- Minimizes extra memory needed by application
  - Relevant for memory constrained applications and extreme scale systems
- Variations include parallel collective I/O
  - MPI_File_write_all

Double Buffer



- Memory copy to permit application to continue
  - Memory may be same node (memory-to-memory copy)
  - Memory may be another node (send/put over fast interconnect)
- Significantly greater memory required
- Variations include parallel independent and collective I/O
- Still constrains progress – data write must be complete before next I/O step
  - Of course, can fix in short term with even more memory
  - Long term; sustained I/O bandwidth to file system must be at least rate at which data is generated

DataFlow



- Break the "BSP" style of compute/IO phases
    - Deliver data to I/O system when ready, in sensible-sized block
    - Can avoid copy if data access well-marked (don't overwrite until I/O completes or data copied)
    - Few (no?) good programming models or systems for this

All Programs Are Parallel

- But (natural) data representation is not parallel
  - Single file/database/object/timestamp/checkpoint is the natural unit
  - At extreme scale, the *number* of parallel processing elements (nodes/cores/etc.) likely to vary from run to run
    - Reliability, resource availability, cost
- In preceding, assumption is that "program" was a parallel program, writing data to a file/store that hides/ignores the fact that n processes/threads/teams wrote the file

HPC Software A Good Base

- MPI-IO, HDF5, pnetCDF, HPSS, other ad hoc solutions provide good building blocks
- Needed: Better abstract models, for both high and low level abstractions
  - "DSL" for data manipulation at scale
  - Such systems are data structure + methods (operators)
- Implementations that fully exploit good and clean semantics of access

Avoid Bad Science

- What is wrong with this statement:
  - "Our results show that XX is faster than MPI-IO"
- Testing the performance of an implementation on a platform provides little data about a language or specification
  - Confusing a test of an implementation with fundamental properties of a specification is ***bad science***.
- There are many ghastly mismatches between what an MPI IO implementation should accomplish and what it does in current implementations
  - Leads to the development of ad hoc solutions that work around limitations of the implementations, not the definitions.
  - We can repeat this error if we aren't very careful

Define Consistency Models for Access and Update

- Need consistency models that match use in applications
  - Or trade accuracy for speed
  - Already happened in search, e-commerce, even when solution is to trade accuracy for speed
    - Witness Amazon's pseudo cart implementation – items aren't really under your control ("in your cart") until you complete the purchase. But greatly simplifies data model.
    - Even though it angers customers on popular deals
- POSIX consistency model is stronger than sequential consistency and almost never what applications require
  - Even when strong consistency is needed, it is almost always on the granularity of a data object, not bytes in a file
  - Long history of file systems falsely claiming to be POSIX
- A bad alternative is the "do what is fast" consistency model – usually but not always works
  - Some systems have taken this route – both I/O and RDMA

Interoperability

- HDF5 provides strong support for many aspects of data provenance. Mechanisms exist in pnetCDF.
  - Should a base set be "automatic", much as file creation/ modify time is today?
  - Can we evolve to better interoperability, or are radically new models needed?
- Mathematical representation for continuous data
  - How should the information about the mapping of discrete ➔ continuous be stored *in the file?*
  - How should this be generalized to other representations?
- Accuracy of data values
  - How should accuracy be *efficiently* stored with file?
- Data formats impact performance and scalability
  - Optimizing for interoperability or performance *alone* may impede application
  - You **cannot** pick the format and then (successfully) say "make it fast"

Conclusions

- Extreme scale systems offer opportunities for unmatched data-centric computing
  - ♦ Memory as large as many databases
  - ♦ Order 10μsec access to *all* data
  - ♦ I/O system optimized for large, complex objects
- HPC software recognizes essential role of locality, latency, consistency
  - ♦ But inadequate implementations have diverted attention from core issues – lets not make that mistake again
  - ♦ Data structures + algorithms = problems is true here
    - choice of data representation has strong effect on performance
- Big Data and Extreme-Scale Systems should focus on problems that can't be done on lesser systems
  - ♦ Focus on
    - Data dependent, fine-grain compute
    - Truly large *single* problems