Runtime Data Management for Data-Intensive Scientific Applications

Xiaosong Ma

NC State University

Joint Faculty: Oak Ridge National Lab

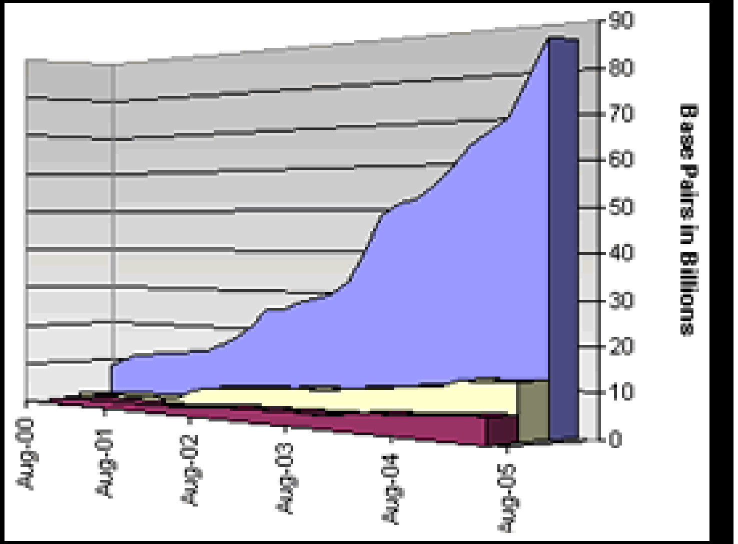
ECPI: 2005 – 2008

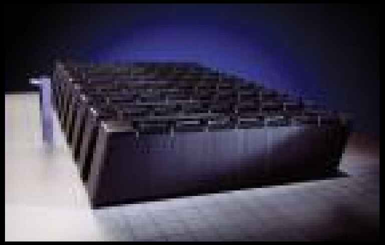
Presentation Roadmap

- Problem definition
- Proposed approaches
- Preliminary results
- On-going research

Data-Intensive Applications on NLCF

- Data-processing applications
  - Bio sequence DB queries, simulation data analysis, visualization
- Challenges
  - Rapid data growth (data avalanche)
    - Computation requirement
    - I/O requirement
    - needs ultra-scale machines
  - Less studied than numerical simulations
    - Scalability on large machines
  - Complexity and heterogeneity
    - Case-by-case static optimization costly





Run-Time Data Management

- Parallel execution plan optimization
  - Example: genome vs. database sequence comparison on 1000s of processors
  - Data placement crucial for performance/scalability
  - Issues
    - Data partitioning/replication
    - Load balancing
- Efficient parallel I/O w. scientific data formats
  - I/O subsystem performance lagging behind
  - Scientific data formats widely used (HDF, netCDF)
    - Further limits applications' I/O performance
  - Issues
    - Library overhead
    - Metadata management and accesses

Proposed Approach

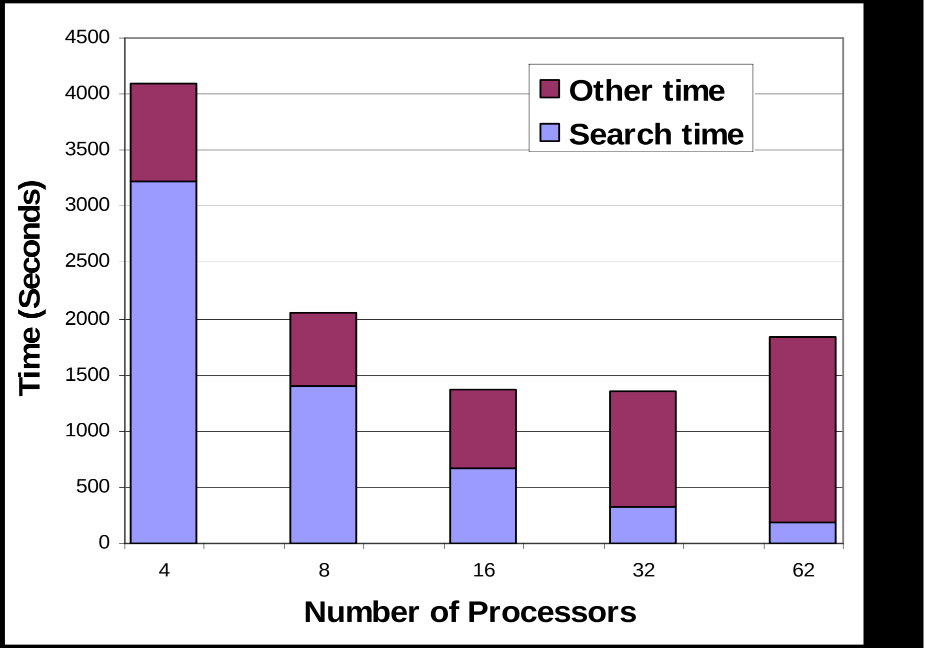
- Adaptive run-time optimization
  - For parallel execution plan optimization
    - Connect scientific data processing to relational databases
    - Runtime cost modeling and evaluation
  - For parallel I/O w. scientific data formats
    - Library-level memory management
    - Hiding I/O costs
      - Caching, prefetching, buffering

Prelim Result 1: Efficient Data Accesses for Parallel Sequence Searches

- BLAST
  - Widely used bio sequence search tool
  - NCBI BLAST Toolkit
- mpiBLAST
  - Developed at LANL
  - Open source parallelization of BLAST using database partitioning
  - Increasingly popular: more than 10,000 downloads since early 2003
  - Directly utilizing NCBI BLAST
  - Super linear speedup with small number of processors

Data Handling in mpiBLAST Not Efficient

- Databases partitioned statically before search
  - **Inflexible**: re-partitioning required to use *different* No. of procs
  - **Management overhead**: generating large number of small files, hard to *manage*, *migrate* and *share*
- Results processing and output serialized by the master node
- Result: **rapidly growing non-search overhead** as
  - No. of procs grows
  - Output data size grows



- Search 150k queries against NR database

- Non searching time increases sharply as number of processors grows

pioBLAST

- Efficient, highly scalable parallel BLAST implementation [IPDPS ’05]
  - Improves mpiBLAST
  - Focus on data handling
  - Up to order of magnitude improvement on overall performance
  - Currently being merged with mpiBLAST

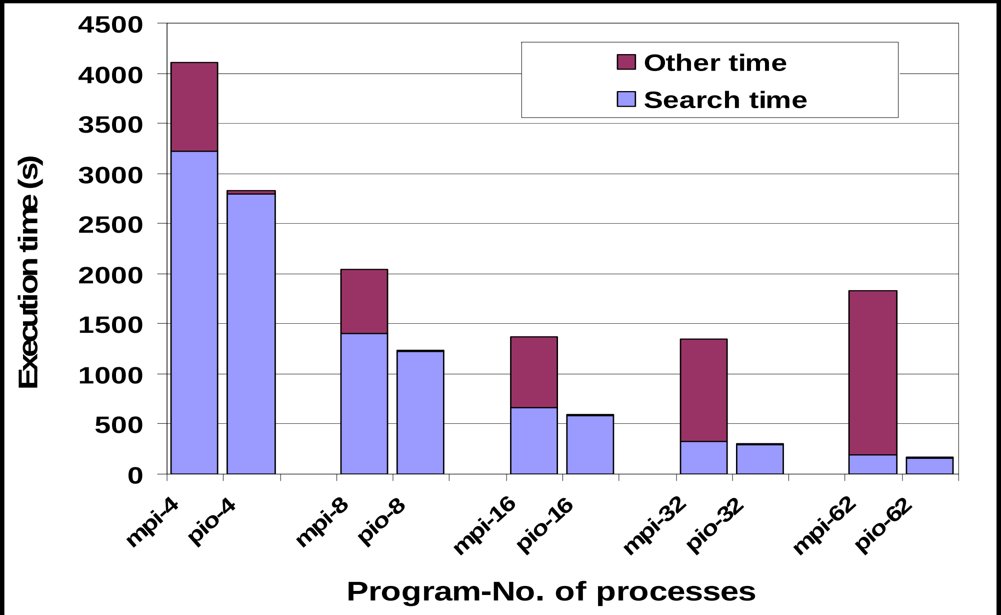
- Major contributions
  - Applying **collective I/O** techniques to bioinformatics, enabling
    - Dynamic database partitioning
    - Parallel database input and result output
  - Efficient **result data processing**
    - Improved result caching for reducing I/O
    - Enhanced worker-master communication for reducing data transfer volume and removing master bottleneck

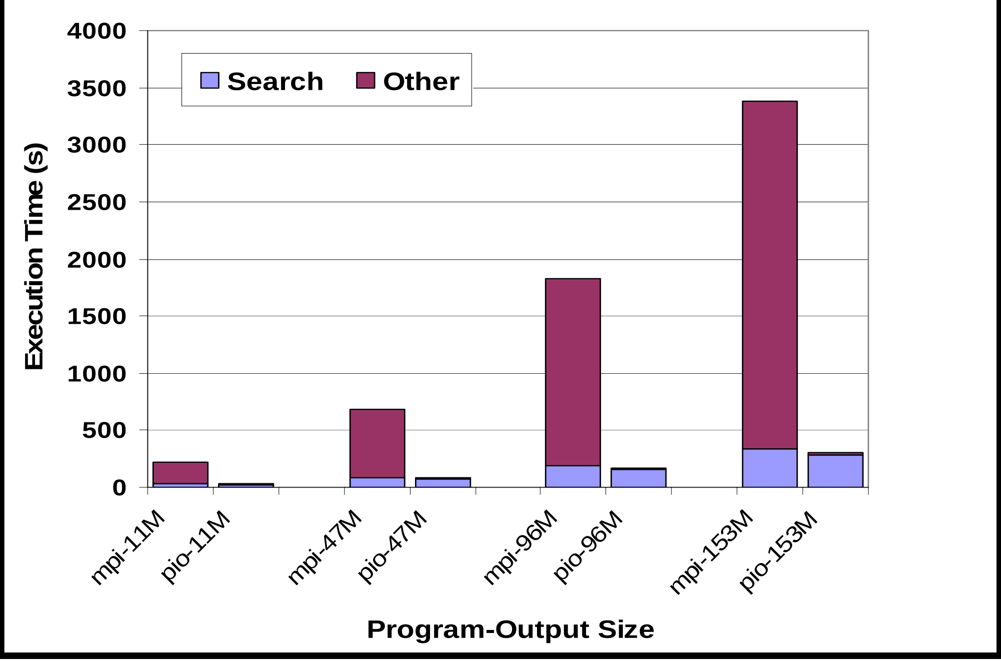
pioBLAST Sample Performance Results

- Platform: SGI Altix at ORNL
  - 256 1.5GHz Itanium2 processors
  - 8GB memory per processor
- Database: NCBI nr (1GB)

- Node scalability tests (top figure)
  - Queries – 150k queries randomly sampled from nr
  - Varied no. of processors

- Output size scalability tests (bottom figure)
  - 62 processors
  - Varied input query sets to generate different output data sizes



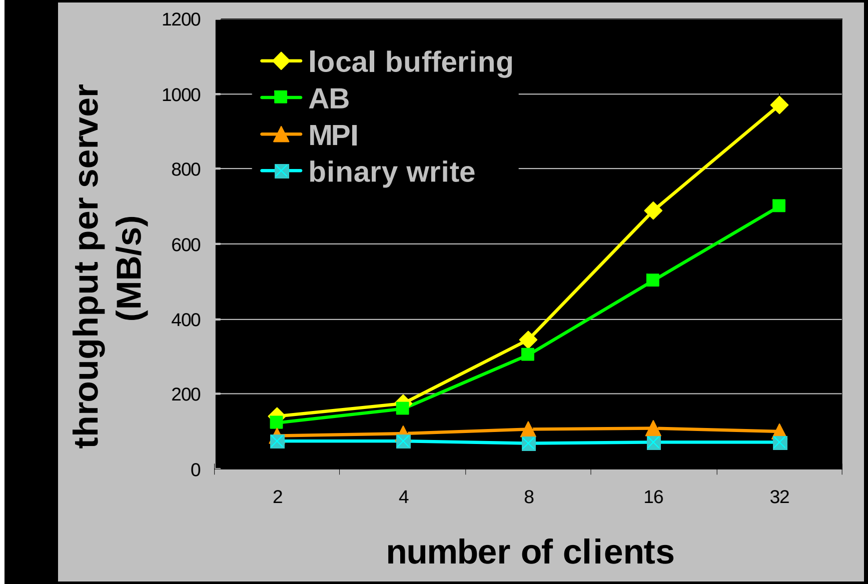


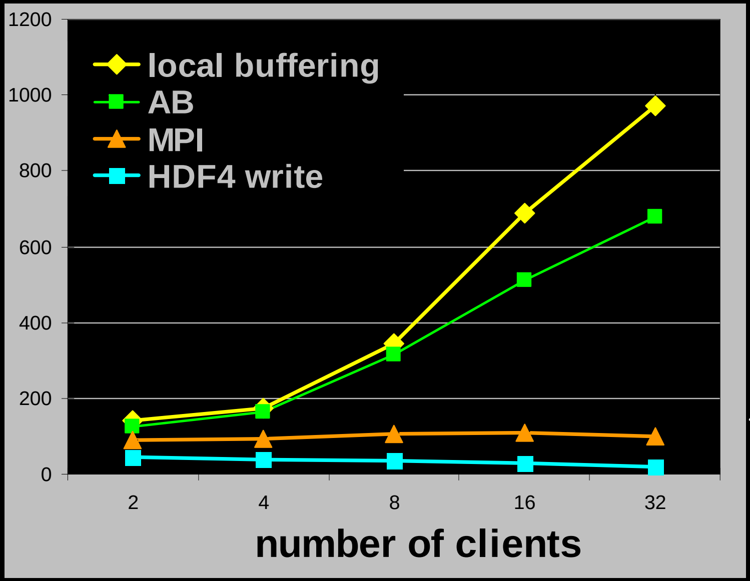
Prelim Result 2: Active Buffering

- **Hides periodic I/O costs behind computation phases [IPDPS ’02, ICS ’02, IPDPS ’03, IEEE TPDS (to appear)]**
- **Organizes idle memory resources into buffer hierarchy**
- **Masks costs of scientific data formats**

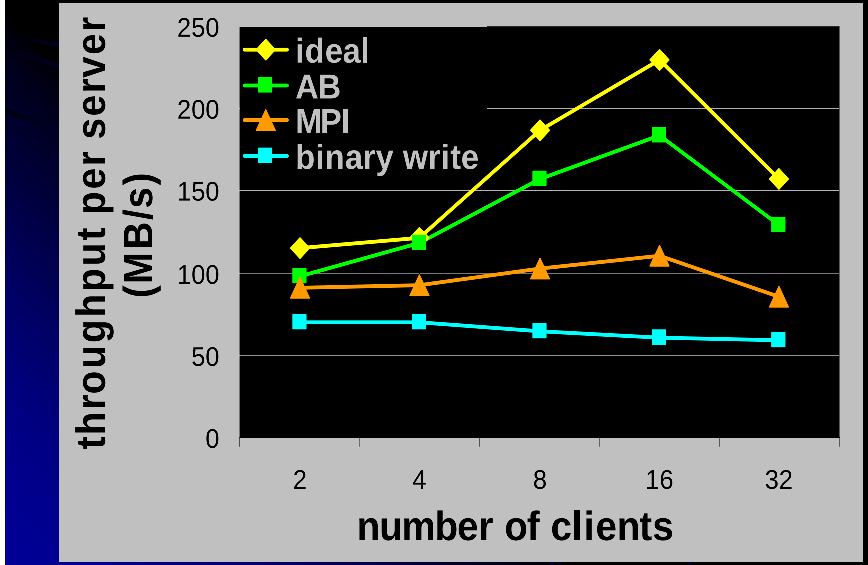
- **Panda Parallel I/O Library**
  - University of Illinois
  - Client-server architecture
- **ROMIO Parallel I/O Library**
  - Argonne National Lab
  - Popular MPI-IO implementation, included in MPICH
  - Server-less architecture
  - ABT (Active Buffering with Threads)

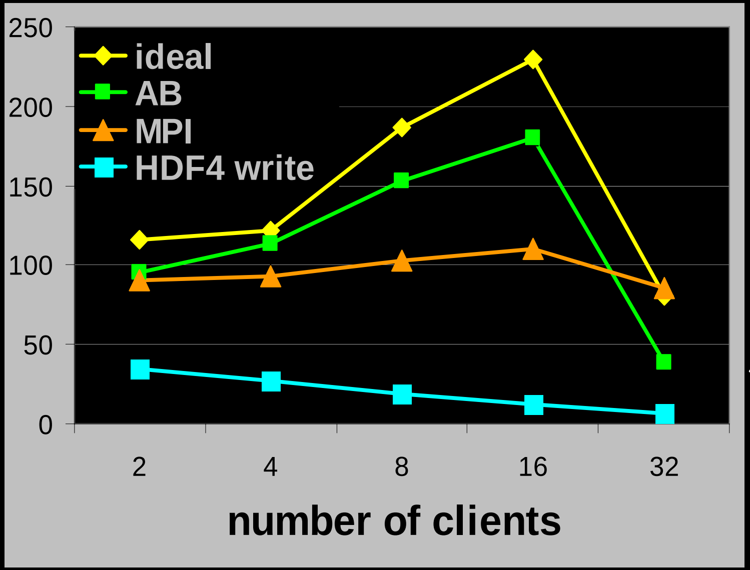
Write Throughput w. Active Buffering





w/o buffer overflow

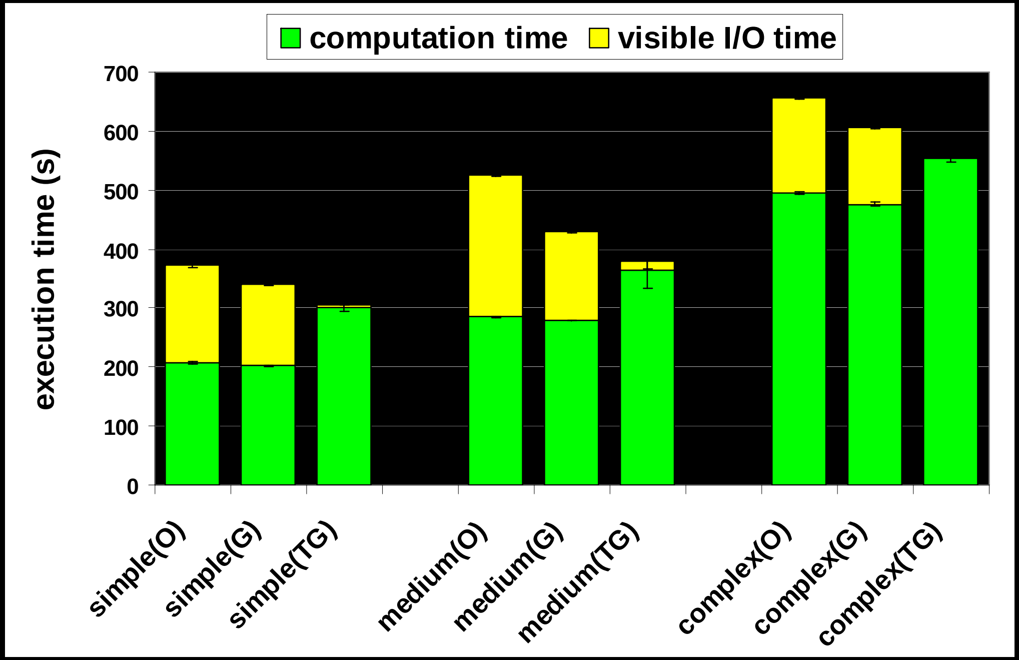




w. buffer overflow

Prelim Result 3: Application-level Prefetching

- **GODIVA Framework: hides periodic input costs behind computation phases** [ICDE '04]
  - **G**eneral **O**bject **D**ata **I**nterface for **V**isualization **A**pplications
  - In-memory database managing data buffer locations
  - Relational database-like interfaces
  - Developer controllable prefetching and caching
  - Developer-supplied read functions



On-going Research

- Parallel execution plan optimization
  - Explore optimization space of bio sequence processing tools on large-scale machines
  - Develop algorithm-independent cost models
- Efficient parallel I/O w. scientific data formats
  - Investigate unified caching, prefetching and buffering