Scalable Approaches for Supporting MPI-IO Atomicity

Peter Aarestad, Avery Ching, Rajeev Thakur, and Alok Choudhary
Northwestern University
Evanston, IL, USA

NORTHWESTERN
UNIVERSITY

George K. Thiruvathukal
Loyola University Chicago
Chicago, IL, USA

LOYOLA
UNIVERSITY CHICAGO

Data Integrity in Large-Scale Applications

- Programmatic methods:
  - MPI_Barrier()
  - MPI_File_set_atomicity()
- Guarantees atomicity, but tends to be inefficient and more effort-intensive

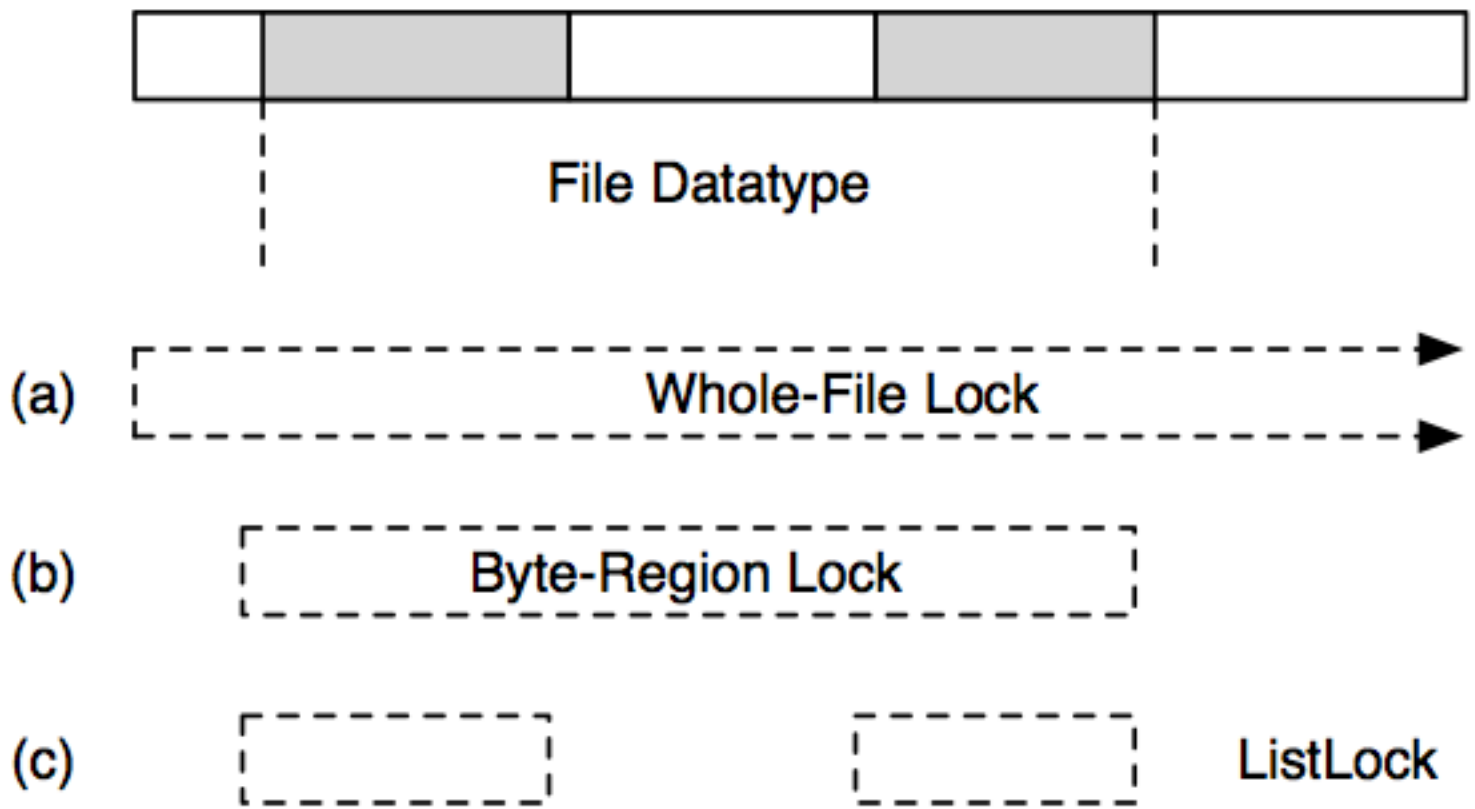
File Locking

- Used by many popular distributed/parallel file systems:
  - GPFS (Schmuck et al, 2002)
  - Petal (Lee et al, 1996)
  - NFS versions 3 (RFC 1813) and 4 (RFC 3530)
- Main strategies: Whole-File Locking and Byte-Range Locking

A Better Way to Lock: List Locks

- Other methods: Easy to implement, but cause a great deal of false sharing
- Our approach: Lock only what you need in the file
- Bottom line: Outperforms simplistic (whole-file and byte-range) locking by a factor of up to 8, and generally performs almost as well as non-atomic I/O access

Summary of Locking Approaches



Whole-file Locking

- One process requests a lock on a file; if available, it gains exclusive access to the entire file
- Another process that needs access to the file will block until the file is unlocked
- Concurrent access to the file is prohibited, but this prevents different processes from accessing different portions of the file

Byte-Range Locking

- Locks granted on a file are now a single byte range - the extent of the datatype being used by the process
- More efficient than whole-file locking; multiple processes can now access the file if their datatypes do not overlap
- Still not ideal: if there are “holes” in the datatype, the lock covers portions of the file not being accessed by the process

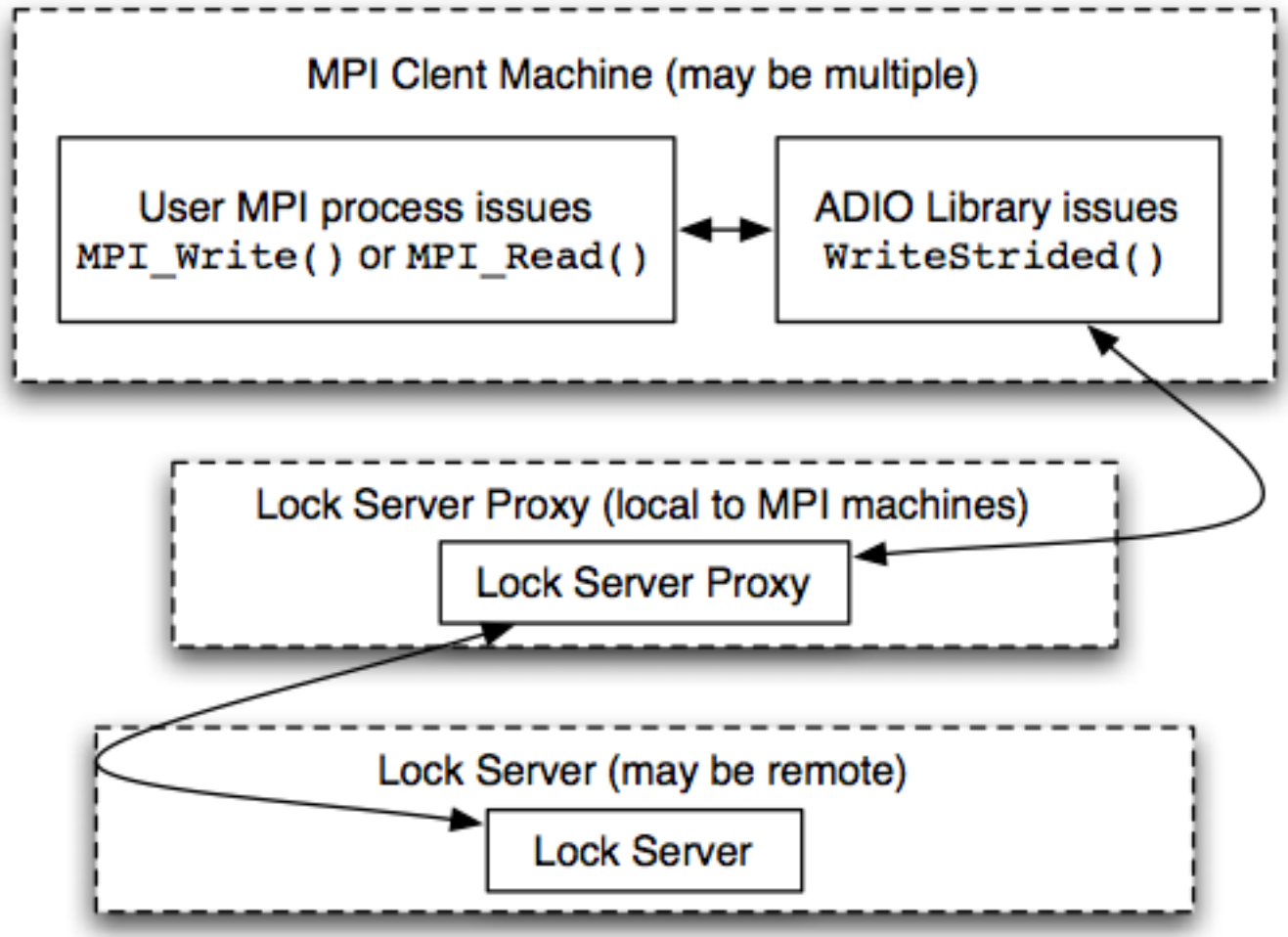
List Locking

- Lock only those parts you need
- How?
  - Examine the data types' offsets and lengths
  - Compute the exact byte ranges you need, consolidating as necessary
  - Send the results to the lock server to be processed in ascending order (to avoid deadlock)

List Locking

- Complications:
  - Computation of the locks - nontrivial, though not too expensive
  - Transmission of the offset-length pairs to the server - potentially thousands of these for a complex data type, so transmission must be efficient

The List Lock Server



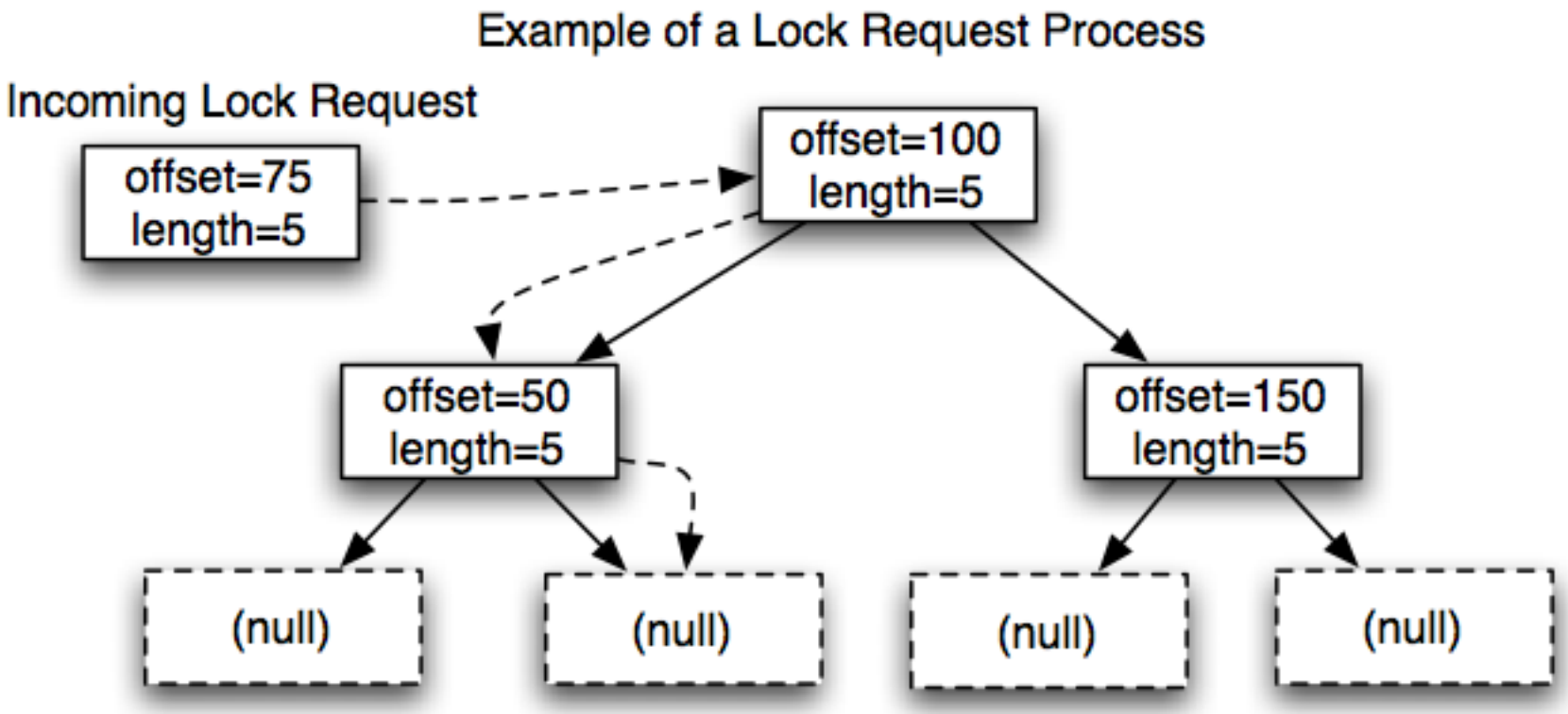
Software Architecture

- Java 1.4.2 (Sun Microsystems, http://java.sun.com)
  - Chosen for implementation speed and simplicity of code; JIT compiler makes simple computations very close in speed to compiled machine-language code
  - Efficient network implementation; important since most of the "think time" is spent transmitting the lock information

Software Architecture

- Locks stored in red-black tree (Guibas et al 1978)
- Time complexity of operations (m = number of lock requests for a file; n = number of existing locks for that file)
  - Insertion: O(m log n)
  - Deletion: O(m log n) worst-case

Software Architecture



Client-Server Communication

- JHPC class library (Thiruvathukal and Christopher, 2000; http://www.jhpc.info)
- Object-oriented transmission protocol
- Modified to send arrays of integers efficiently
- Interface code written to allow communication between clients written in C and servers written in Java

Lock Server API

- MPI-based API
- Client calls lock_datatype() on a datatype instance, which:
  - computes offset-length pairs
  - communicates them to the server
  - waits for affirmative response that locks were acquired

Testbed

- Jazz cluster at Argonne National Laboratory (http://www.lcrc.anl.gov/jazz):
  - 350 nodes, each with single 2.4 GHz Pentium Xeon processor
  - 175 have 2 GB RAM, 175 have 1 GB RAM
  - 80 GB scratch disk, Myrinet 2000 connection, Fast Ethernet connection
  - 10 TB global disk with NFS and PVFS
  - Linux 2.4.29-rc2

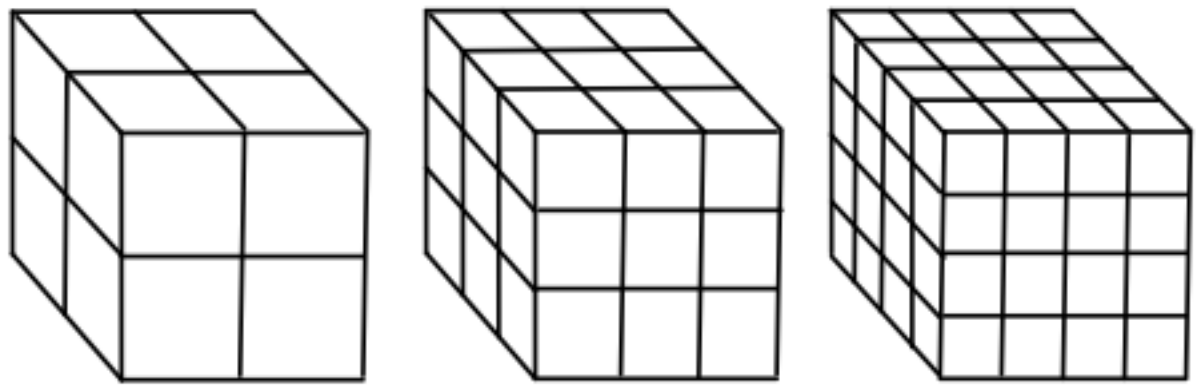
Testbed

- For our tests:
  - MPICH2 version 1.0.2p1
  - PVFS shared disk space
  - Fast Ethernet connections used (MPICH did not recognize the Myrinet host names)

ROMIO 3-D Block Test

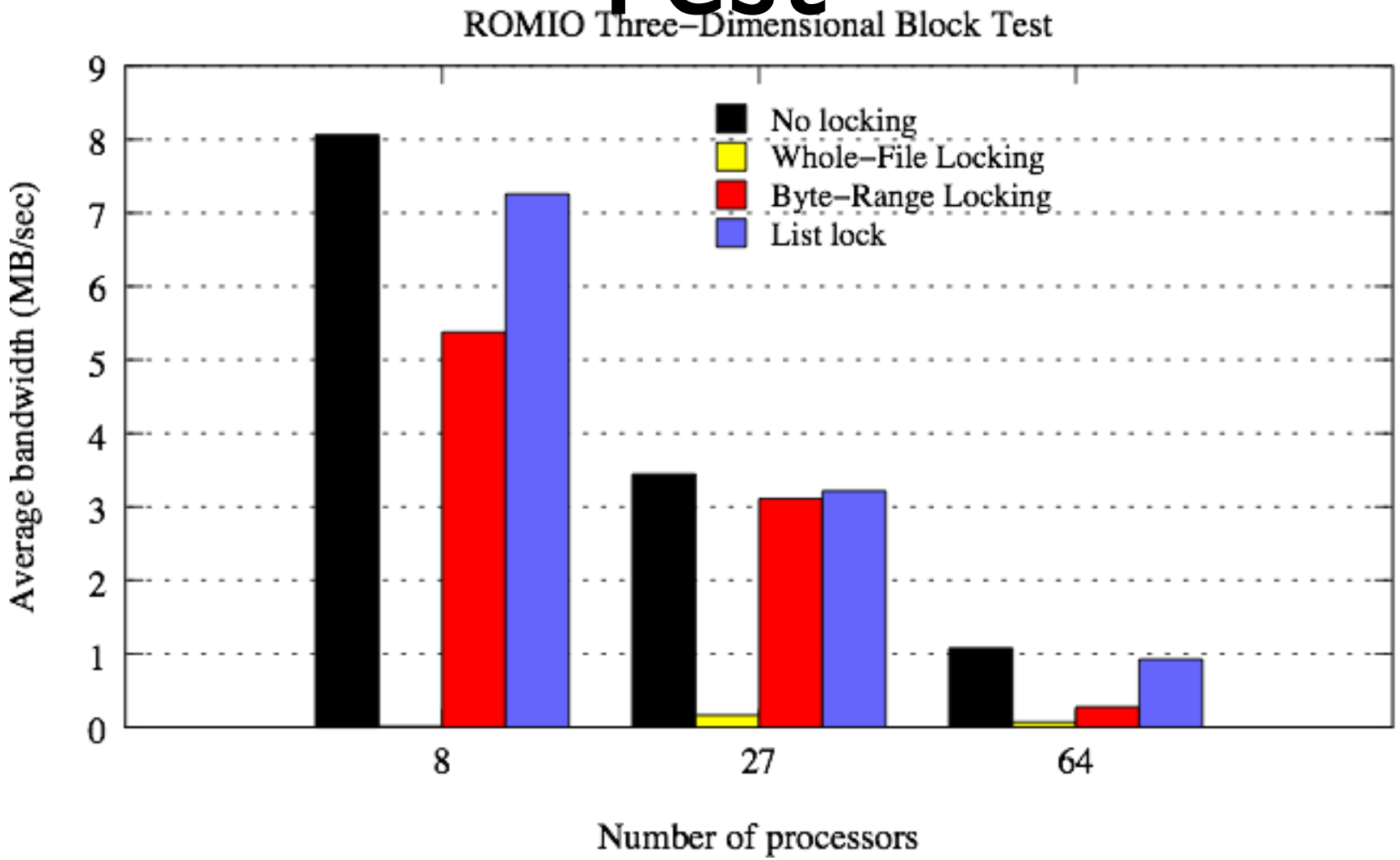
- coll_perf.c test from the ROMIO test suite
- Measures bandwidth writing to a cube of data of size 100 integers
- 8, 27, and 64 processors used

ROMIO 3-D Block Test



	Number of Processors	Number of Locks per Client	Maximum Concurrent Processes
Whole-File Locking	8	1	1
	27	1	1
	64	1	1
Byte-Range Locking	8	1	4
	27	1	9
	64	1	16
List Lock	8	25	8
	27	12	27
	64	64	64

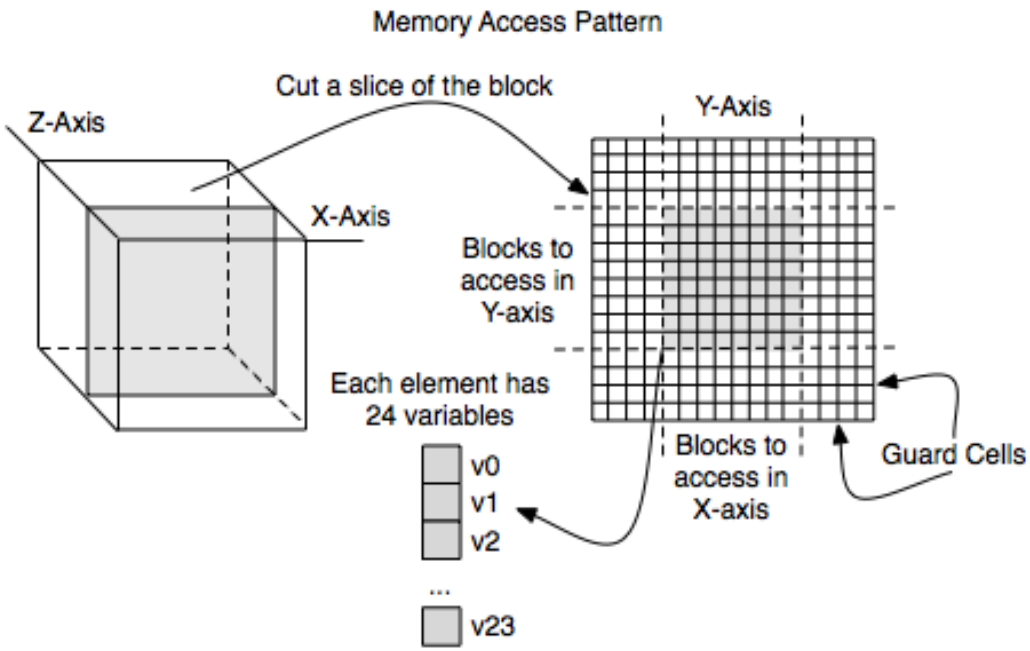
ROMIO 3-D Block Test

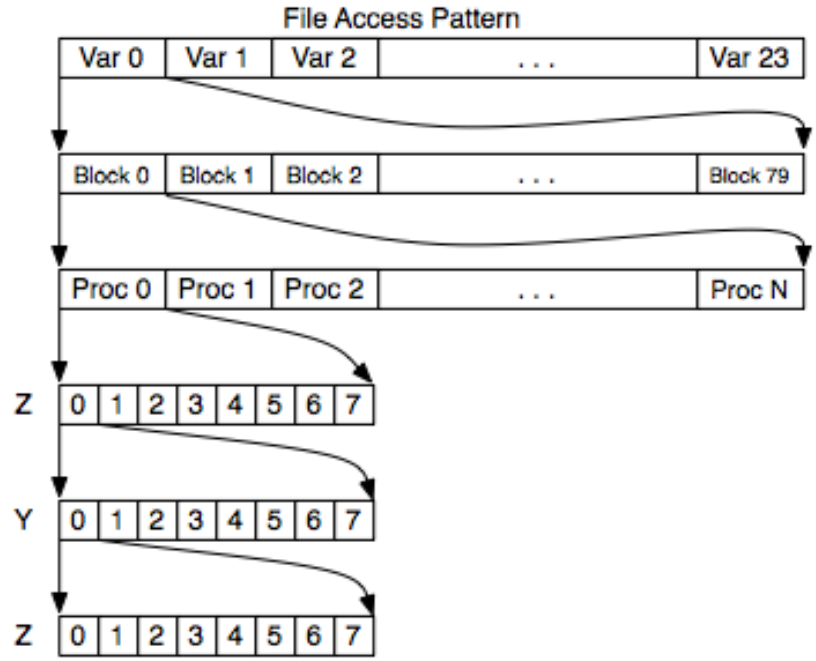


FLASH I/O Simulation

- Mesh refinement application used to study astrophysics
- HDF5 used for writing checkpoints
- Non-contiguous datatypes used, which is useful for exercising novel non-contig I/O systems

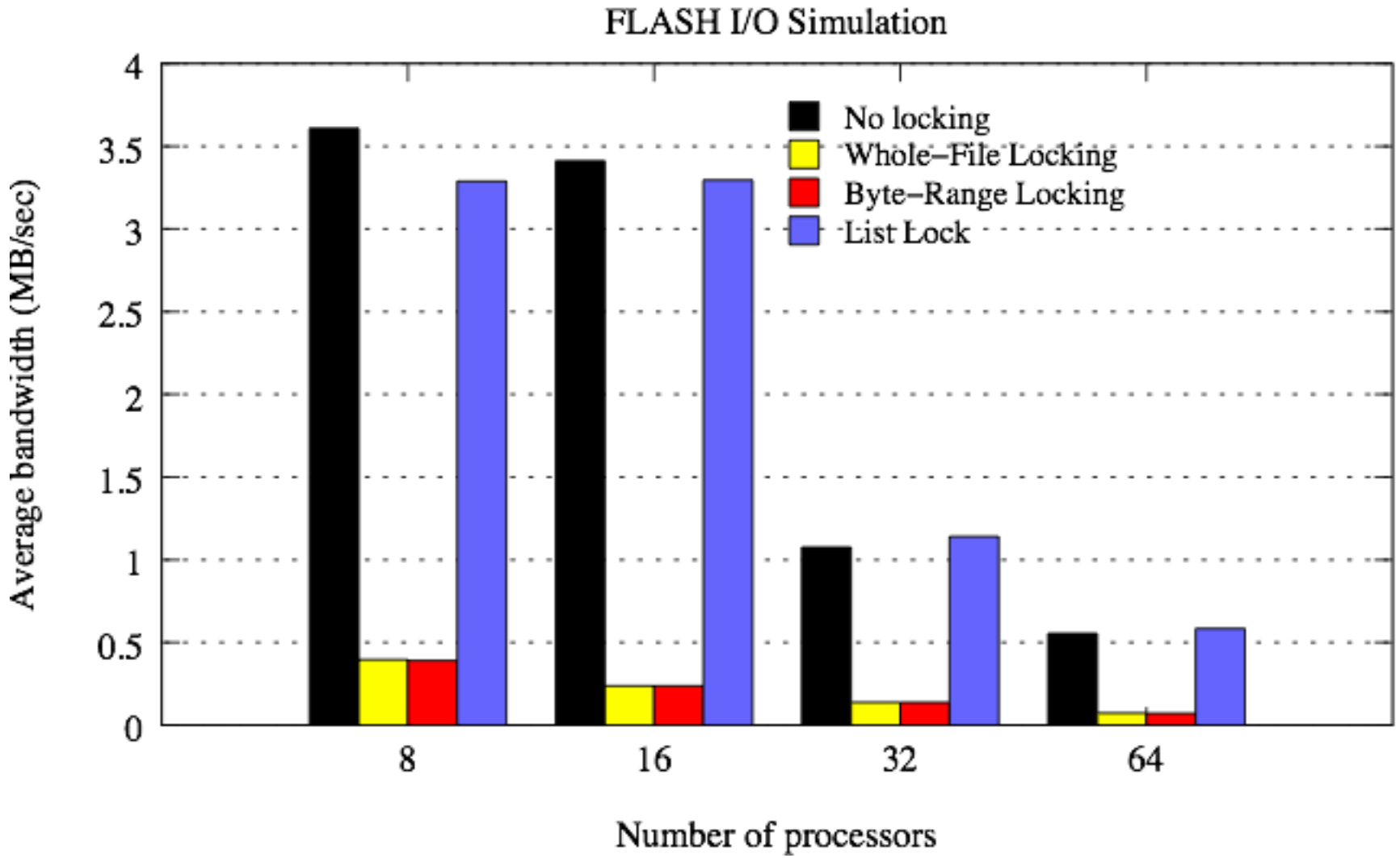
FLASH I/O Simulation





	Number of Processors	Number of Locks per Client	Maximum Concurrent Processes
Whole-File Locking	8	1	1
	16	1	1
	32	1	1
	64	1	1
Byte-Range Locking	8	1	1
	16	1	1
	32	1	1
	64	1	1
List Lock	8	64	8
	16	64	16
	32	64	64
	64	64	64

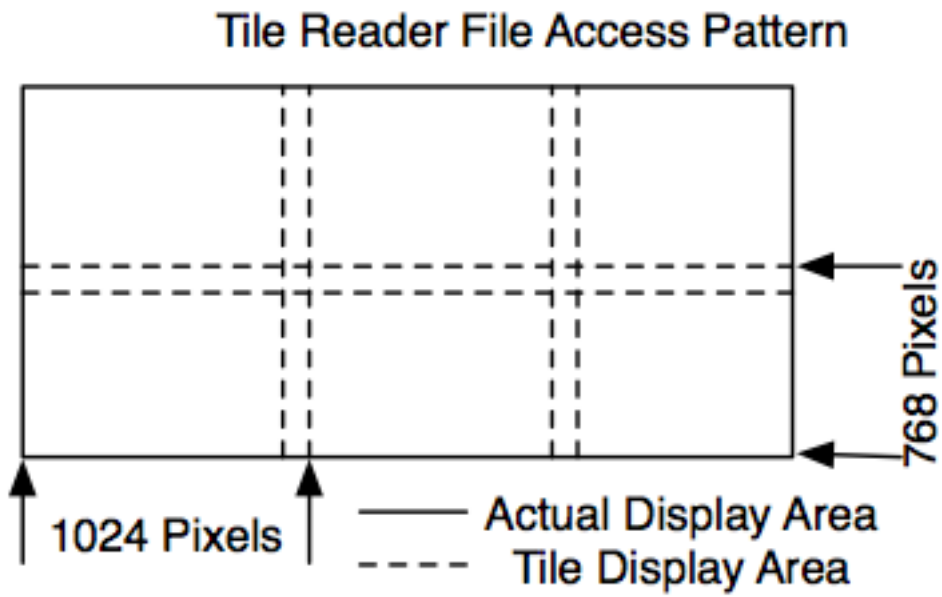
FLASH I/O Simulation



Tile Reader Benchmark

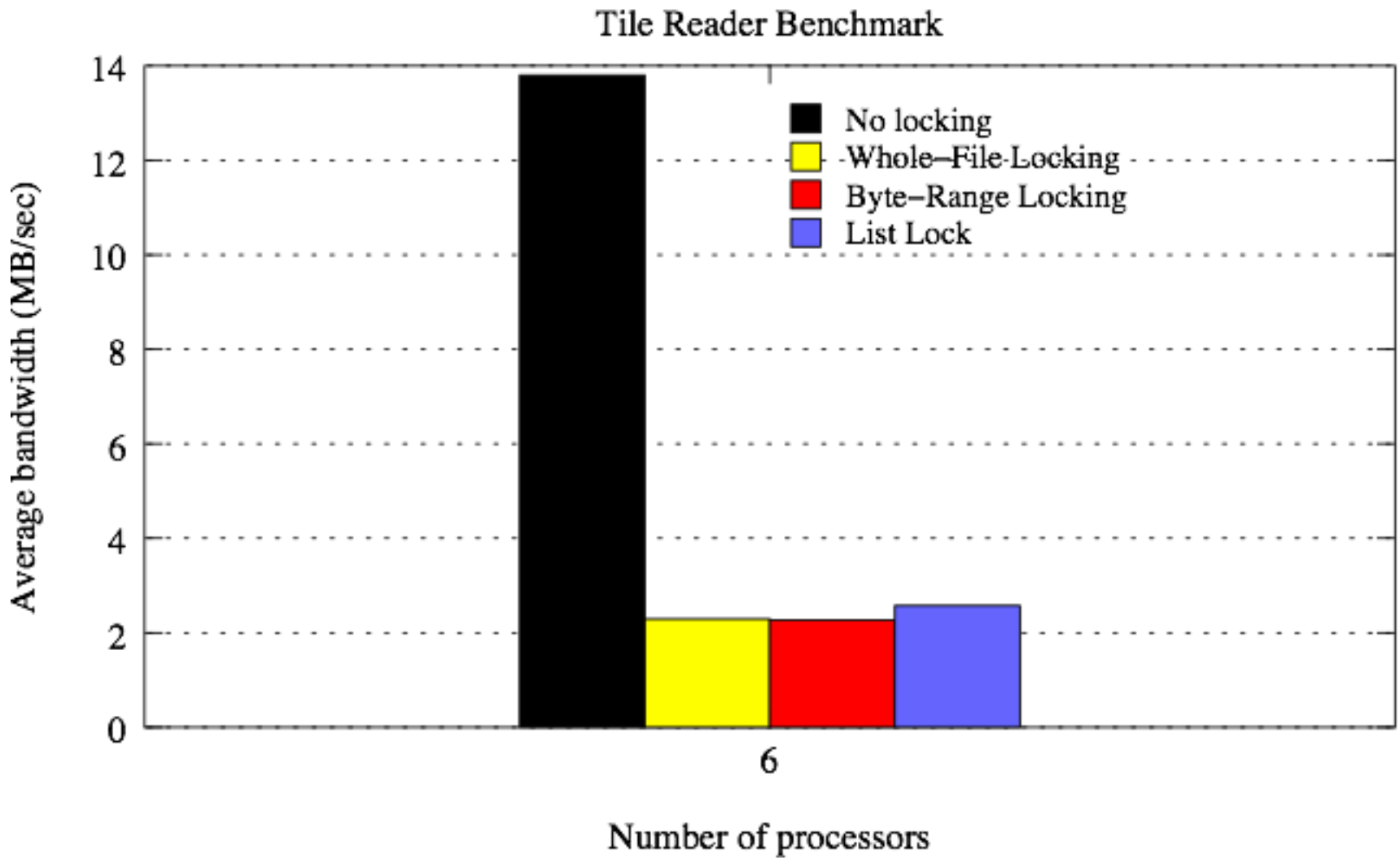
- Used to test performance of commodity-based graphics systems
- Each node simulates reading 1024x768 blocks for displaying to its own display
- 270-pixel horizontal overlap and 128-pixel vertical overlap to hide merging of edges
- 6 processes running simultaneously

Tile Reader Benchmark



	Number of Locks per Client	Maximum Concurrent Processes
Whole-File Locking	1 per file	1
Byte-Range Locking	1 per file	2
List Lock	64 per file	2

Tile Reader Benchmark



Conclusions

- Tests show a fair comparison of locking strategies; list locking is far superior to naïve locking methods in realistic tests
- Demonstration of feasibility of using Java in a high-performance computing application

Future Work

- Handling stale locks - important for a robust production environment
- Compression of lock requests to further optimize lock request transmission time
- Block-oriented lock approach using bit vectors on the server side

Acknowledgments

- National Science Foundation grant to Loyola University Chicago (CCF-0444197)
- McCormick School of Engineering at Northwestern Univ. - Cabell Fellowship
- Kenin Coloma, Wei-keng Liao, Gokhan Memik from Northwestern Univ.
- Narayan Desai, Rick Bradshaw, Rob Ross from Argonne National Laboratory
- Benjamín Gonzáles from Loyola Univ. Chicago