Performance of an MPI-IO implementation using third-party transfer

Richard Hedges, Terry Jones, John May, Kim Yates

Parallel I/O Project

Lawrence Livermore National Laboratory

rkyates@llnl.gov

This work was performed under the auspices of the U.S. Department of Energy
by Lawrence Livermore National Laboratory under contract number W-7405-Eng-48.

Goals of this work

- Give users access to HPSS files via the MPI-IO interface.
  - Portability: common standard vs. HPSS-specific API
  - Ease of use: familiar MPI datatypes, no explicit threads
- Efficient implementation: low overhead.
- Improve on HPSS performance for some access patterns.
  - Can profit from MPI-IO's collective operations

Summary of HPSS

- Fast, large hierarchical archives (disks and tapes).
- Allows $m \times n$ parallelism with 3rd-party transfer
- For high-performance parallel I/O, uses explicit multithreading and nonstandard interface (`hpss_WriteList` and `hpss_ReadList`).

Summary of MPI-IO

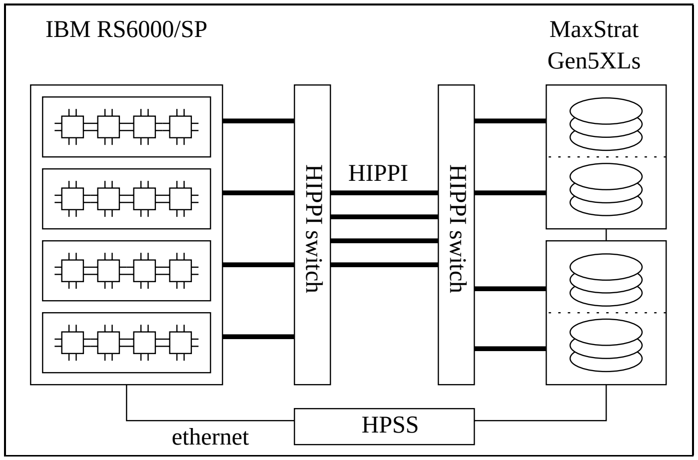
- Became official part of MPI-2 message-passing standard, 1997.
- Writes are like sends, reads are like receives.
- Designed to allow optimizaton of parallel I/O:
  - Collective read and write operations.
  - MPI "derived types" describe data layout.
  - Blocking and nonblocking transfers.
  - Performance "hints" from user, don't alter semantics.
  - Wide variety of features.
- Takes a major effort to implement fully.

Methodology

- Measure performance under various system and application parameters.
- Verify efficiency.
- True concurrent aggregate performance: earliest start to latest end time.
- Each data point is average of 5 runs.
- For writes, overwrite an existing file.
- In the tests reported here, files are small ($\leq$ 256 MB)
  - Only because test programs were inflexible.
  - But HPSS doesn't do any caching anyway.
- Configured system to perform well for large transfers

  (e.g., 8 MB stripe unit).

Testbed (old, small, gone, to be replaced soon)



- Four IBM RS/6000 SP 604 High nodes,

  each with 4 112-MHz PowerPC 604 processors.

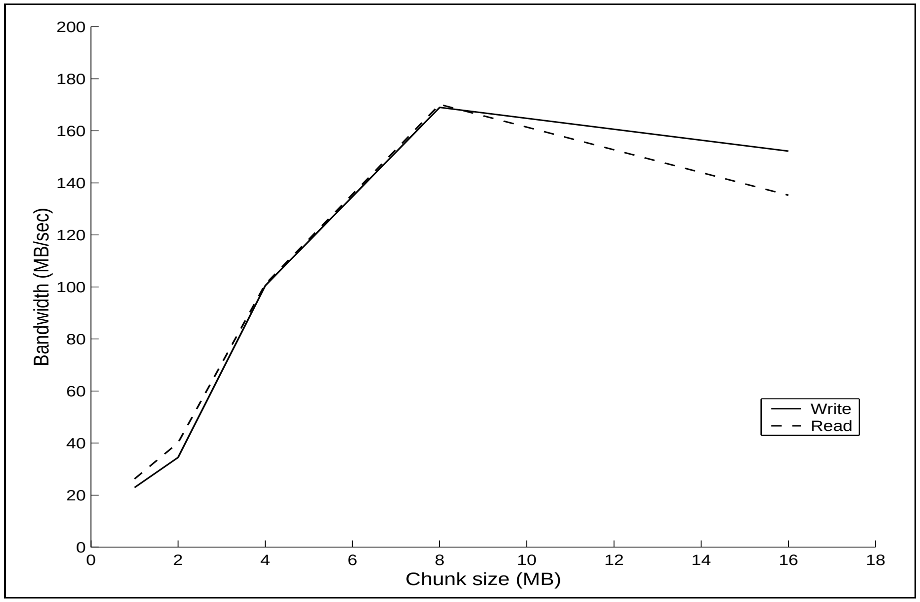
- Four HIPPI cards and crossbar switch.

- 2 MaxStrat Gen5XLs, configured as 4 RAIDs.

- Hardware throughput limit is 207 MB/sec

  (4 HIPPI adapters $\times$ 51.8 MB/s each).

Varying chunk size



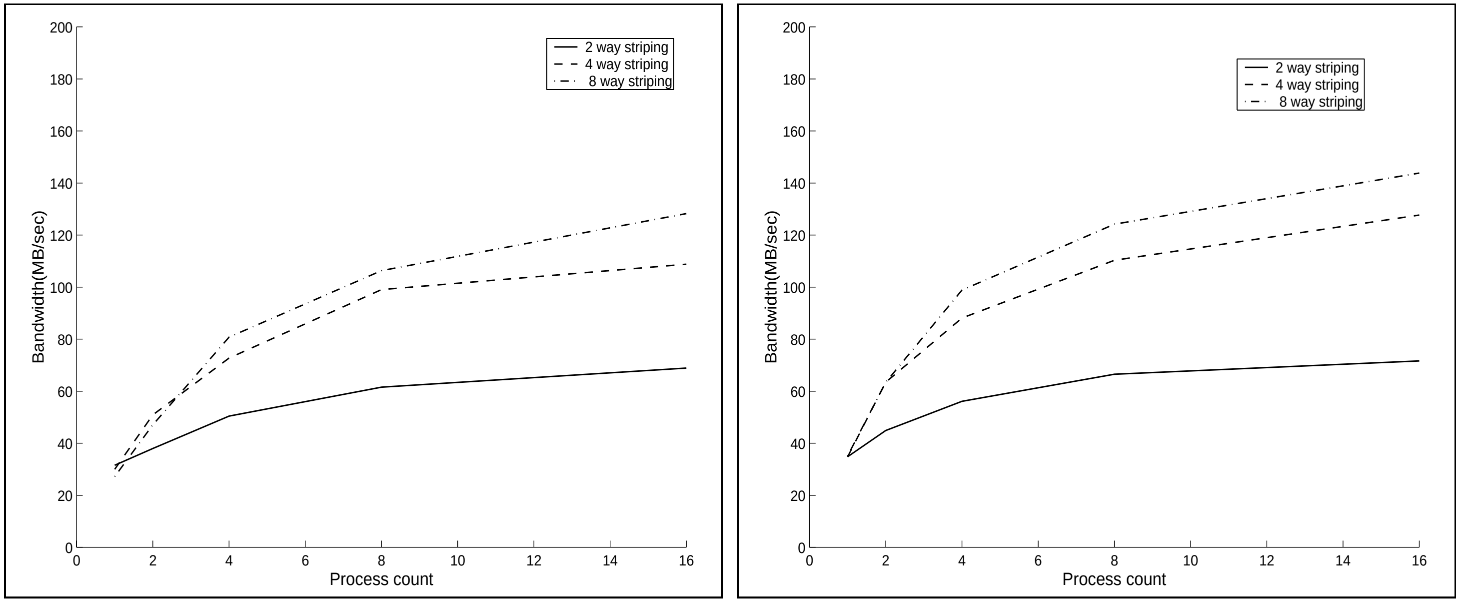
Collective read & write for varying chunk sizes.

(stripe factor = 8, number of processes = 16, file size = 256 MB)

Configured this system to have 8 MB stripe unit.

For chunks < 8MB, HPSS uses TCP/IP instead of IPI.

Varying number of clients and stripe factor



Collective read & write.

(stripe factor = 2/4/8, chunk size = 16 MB; file size = 16 MB per process).

Best performance: 197 MB/s read, 173 MB/s write (207 MB/s hard limit)

(32 procs, 8-way stripe, 8MB chunks)

Overlapping I/O and computation

$$\text{Tested with } t_{compute} = t_{i/o} = 2.5 \text{ sec}$$

$$\text{Total time using blocking i/o} = 5.0 \text{ sec}$$

$$\text{Total time using nonblocking i/o} = 3.1 \text{ sec} \quad (62\% \text{ of blocking i/o})$$

Using 4 client processes, 4-way striping, 16 MB chunks.

When there is more than 1 MPI process on a node,

thread contention reduced performance.

Future work

- Track future versions of HPSS.
- Further analyze and improve performance
  - Larger, faster testbed installed soon.
  - Performance in production use.

Conclusions

- Successful, complete implementation of MPI-IO API.
- Efficient use of HPSS parallel transfer capabilities.
- Working on enhancements to improve on HPSS performance.
- Will soon be exposed to rigors of production use.

Acknowledgements

Many thanks to Linda Stanberry (current principal designer/programmer),

Elsie Pierce and Jeanne Martin (erstwhile team members),

the entire HPSS team, and many sytem administrators

for their help over the years.