Parallel I/O Benchmarking Consortium

May 21, 2002

1 Statement of Purpose

The purpose of the Parallel I/O Benchmarking Consortium is to provide a set of benchmarks for use in characterizing the performance of a parallel I/O system. A concise set will be used for obtaining an "at a glance" view of I/O system performance, and an extended set will be provided for discovering regions of poor performance within a large space of access characteristics. These tests will follow a consistent testing methodology, defined by the consortium, to help with comparing results across tests.

The test will not attempt to measure correctness of the underlying I/O system implementation, nor will it attempt to "score" the I/O system to aid in generation of a "Top 500 I/O systems" or the like.

2 Overview of Test Suite

Tests will consist of parallel applications utilizing MPI for communication and startup and MPI-IO or POSIX interfaces for I/O access.

These two interfaces are in some sense the lowest common denominator and can be expected to exist on most parallel I/O systems. We will avoid the use of alternative native interfaces, such as the PVFS libpvfs library, in order to maintain the portability of the benchmark suite. [This can be revisited as time goes on; perhaps there is a mechanism for hiding these particular interfaces under some generic interface in order to keep the details out of the individual tests?]

The suite will consist of many small test applications, each designed to focus on a particular mode of access or I/O system characteristic, rather than a small set of monolithic test applications. This should ease modification of particular tests when groups find that they want to explore performance of a certain access mode more thoroughly.

It is likely that these tests will share a great deal of functionality (passing parameters out to all clients, calculating final performance values, etc.). For this reason we may find it advantageous to provide a skeleton code to serve as a starting point.

2.1 I/O Access Space

The methods by which applications access I/O systems vary widely between applications, application domains, and users. It is unrealistic for us to attempt to cover this entire space; rather we intend to focus on what we believe to be some of the most important regions in this I/O access space.

One such region is interactive use. Tests examining this portion of the access space would examine the latency of common user operations such as copying files, listing directories, and working with archives. Many such tests might simply be written as scripts wrapping around common UNIX tools.

Another region of interest is that of large, contiguous I/O to large files. Many applications have been written to use large, contiguous operations in order to perform more optimally on parallel I/O systems. Testing of this region would include both POSIX I/O and MPI-IO with simple datatypes.

Many applications that treat the parallel I/O system in a more database-oriented manner utilize small I/O accesses and many files. A separate set of test would examine performance in this space. Of equal importance here is the cost of creating and destroying these many, small files.

The use of more complicated MPI derived datatypes in I/O operations has become more popular as MPI-IO implementations have matured. These types of accesses lie in a separate space from the two above, as often these are large but noncontiguous accesses.

Across all of these tests, the degree to which performance scales with increasing numbers of clients is extremely important. All tests will need to perform a sweep across numbers of clients to see if there are bottlenecks in this dimension.

Finally, the use of MPI-IO collective operations allows for important optimizations. Testing of large contiguous, small contigouous, and large noncontiguous accesses in a collective I/O mode is important for understanding the impact of collective calls on I/O performance.

Many of these patterns could be summarized in a table...

3 Existing Projects

There are a number of projects from which we could pull tests and/or perhaps find collaborators. The most obvious of these are:

- be_eff_io
- ior
- mib
- mpi-tile-io

[Some text on these would be great.]

3.1 mpi-tile-io

mpi-tile-io is a test application that implements tile access to a two dimensional dense dataset. This type of workload is seen in tiled displays (for small numbers of tiles) and in some numerical applications.

From the README (until we know what we want in here):

```
The parameters controlling the interpretation of the file are as
follows:
nr_tiles_x - number of tiles in the X dimension (rows)
nr_tiles_y - number of tiles in the Y dimension (columns)
sz_tile_x  - number of elements in the X dimension of a tile
sz_tile_y  - number of elements in the Y dimension of a tile
sz_element - size of an element in bytes
overlap_x  - number of elements shared between adjacent tiles in the X
             dimension
overlap_y  - number of elements shared between adjacent tiles in the Y
             dimension
header_bytes - number of bytes at beginning of file to consider as
               header data (skipped over)
filename - name of file to operate on

Additionally the following parameters control the behavior of the
application:
collective - perform I/O collectively
cb_config_list - control aggregation
```

4 Standardization

The most important goal of this project is the standardization of terminology, testing, and reporting for tests in the benchmark. This section details standards that we have (by some mechanism that has yet to be defined) agreed upon.

4.1 Suggested Terminology

We need to define some terms to use in testing. This includes things such as what we mean by aggregate bandwidth, open time, whatever.

4.2 Standardized Testing

We need to define HOW we are going to test. This includes: - what operations are included in timing (e.g. do we include open time in I/O operations? close time? sync time?) - how times are reported, best of N tests, worst of N tests, average. what about variation?

4.3 Standardized Reporting

We need to choose how we want to report values. Specifically we want a standard "language" for reporting so that the results of multiple test runs can be integrated easily.

Standardized terminology ensures that we are all talking about the same things. Standardized testing ensures that we are testing the same things in the same way. Standardized reporting ensures that the results of multiple test can be put together into a meaningful picture with little additional work.

5 Gotchas

Hints on avoiding caching, etc.

6 Proposals

This section lists proposed standards (of one type or another) for discussion by the consortium. It might also list proposals for the mechanism for adopting a standard, etc.

Eventually we'll probably be passing around proposals outside this document, but while we're still getting started, this should do.

Comments on things in here are mandatory :). No really, we need comments in order to make progress... Comments can be on the concept, on wording, to clarify, to disagree, whatever.

All proposals should have a name and a primary author. A submission date will be added when the proposal is first received.

6.1 Access Patterns – Bill Loewe, 04/23/2002

SEGMENTED PATTERN - Using n clients, create, write, read, and close a single common file. Each client accesses x bytes, which are contiguous in the file. For each client c in 0.n-1, c's first byte is located at offset c*x. Thus, for a file of the form:

A B C

aaaa bbbb cccc

each client (A, B, or C) accesses the contiguous block A, B, or C in transfer of a, b, or c. The contiguous region ABC is defined as a segment.

STRIDED PATTERN - Using n clients, create, write, read, and close a single common file. The defining characteristic of the strided pattern is that it is a segmented pattern repeated at an offset of a stride. Thus, for a file of the form:

```
A    B    C    -   A'   B'   C'   -   A"   B"   C"   -
```

```
aaaa bbbb  cccc -   aaaa bbbb cccc -   aaaa bbbb cccc -
```

the client A accesses blocks A, A', and A" in transfers of a. In addition to the segment ABC, there may be padding '-' to fill out an entire stride of ABC-.

RANDOM PATTERN - Either a segmented or strided pattern that allows for random variation of the size of any transfer (a, a', a", a'", e.g.) for any client (A, B, or C) in any stride. Accordingly, this allows for random block sizes.

```
A            B            C            A            B            C

a a' a" a'"  b b' b" b'"  c c' c"c'"   a a' a" a'"   b b' b" b'"  c c' c"c'"
```

GENERAL ACCESS PATTERN - Using n clients, create, write, read, and close a single common file. This file is accessed with a random, strided pattern with the addition of an initial offset (in the form of padding) to offer the option of making the block alignment askew.

```
-   A   B   C   -   A'  B'  C'  -   A"  B"  C"  -
```

6.2 Additions to Suggested Terminology – Bill Loewe, 04/23/2002

6.2.1 General Terminology

TRANSFER RATE - aka (AGGREGATE) BANDWIDTH - (in MB/sec, where MB = 2^{20} bytes). Transfer rate includes the time to open, write/sync (or read), and close.

TRANSFER - a single data buffer transferred (writing or reading) in a single I/O call.

TRANSFER SIZE - the size (in bytes) of a transfer.

BLOCK - a contiguous chunk of data written or read by a single client. It is comprised of one or more transfers.

BLOCK SIZE - the size (in bytes) of a block.

SEGMENT - a contiguous chunk of data comprised of blocks written or read by multiple clients.

PADDING - the skipped offset devoid of data in a file; a gap in a sparse file.

STRIDE - a segment (as defined above) of data together with a padding offset; it is the distance covered (in bytes) between the start of successive segments.

6.2.2 Access Terminology

SEGMENTED PATTERN - the entirety of each client's chunk of data is written contiguously, with each chunk of data from each client also written contiguously [see section 2.1 for detail]

STRIDED PATTERN - the repetition of a segmented pattern with the addition of [see section 2.1 for detail]

RANDOM PATTERN - a segmented or strided pattern in which the size of the individual transfer is randomly varied. [see section 2.1 for detail]

6.2.3 Timing Terminology

TIMINGS OPEN, WRITE, READ, CLOSE - counted in seconds using double- precision floating-point. All timings performed with a single barrier for write or read. For example:

```
-  barrier -          -  barrier -

startTime             startTime
open                  open
stopTime              stopTime


startTime             startTime
write                 read
(sync)
stopTime              stopTime


startTime             startTime
close                 close
stopTime              stopTime
```

WRITE/READ TIME - timing of entirety of write (or read) calls in the data transfer, not an individual write() or MPI_File_write(), e.g.

OPEN/CLOSE TIME - time elapsed between the start time of the collective open/close and its stop time.

TRANSFER RATE - amount of aggregate data in MB transferred to disk per second. The timing is begin with the start time before the collective open and end with the stop time after the collective close.

For timings, it is assumed that using MPI_Wtime() will return a value synchronized between all clients in the communicator. As such, the overall timing interval will be determined as the earliest start time and the latest stop time.

MAXIMUM - the highest transfer rate of an iteration of a test. This assumes that the maximum performance timing shows what the system is capable under optimal conditions.

DEVIATION - square root of the variance, in which the variance is the average squared deviation of each value from its mean.

6.3 Parameter Input – Rob Ross, 03/15/2002

I really like "–" format long name parameters on the command line, but I've noticed that `getopt_long()` isn't available on all the platforms (e.g. Frost) that we would like to run on. I propose that all tests implement both an environment variable and command line argument systems, and that command line arguments take precedence over environment variables.

All environment variable parameters should be upper case with underscores as separators. All command line parameters should be lower case with underscores as separators.

mpi-tile-io implements such a system.

6.4 Standard Parameters – Rob Ross, 03/15/2002

Based on mpi-tile-io, here are some proposed standard parameter names:

- `filename` – the name of a single file on which the test operates
- `collective` – for tests that can optionally operate with collective calls (rather then independent ones), setting this to a non-zero value will enable the use of MPI collective I/O calls
- `header_bytes` – for tests that can append a variably sized "header" to perturb block alignment, this parameter sets the size of this offset in bytes.

6.5 MPI-IO Hints – Rob Ross, 03/15/2002

To facilitate passing MPI-IO hints into tests using MPI-IO, we should ensure that parameters are available to pass in appropriate values. These parameter names should match the corresponding MPI-IO hint key (except for case as discussed above). Examples: `cb_nodes`, `romio_cb_read`, `romio_cb_write`, `romio_no_indep_rw`, `cb_config_list`, `ind_rd_buffer_size`, `ind_write_buffer_size`, `romio_ds_read`, `romio_ds_write`, `striping_factor`, `striping_unit`, `start_iodevice`, `pfs_svr_buf`, `direct_read`, `direct_write`.

Because these are MPI hints, they can be safely set and passed in for all cases; they will simply be ignored by implementations that don't understand them.

This is another place where some common skeleton code would be particularly useful.

[Don't forget the newer IBM ones for data shipping control.]

6.6 Timing Results – Rob Ross, 03/15/2002

Concurrent access tests should return min/mean/max/variance for the times *between* processes.

Timing values should be obtained using `MPI_Wtime()`.

6.7 Aggregate Throughput – Rob Ross, 03/15/2002

A test measuring "aggregate throughput" of any kind must:

- synchronize start of I/O operations with an MPI barrier or similar means
- measure the time to complete on each process independently
- use the maximum of these times as the "aggregate" time

6.8 Sustained Read Performance – Rob Ross, 03/15/2002

A test that measures "sustained read performance" must:

- read a "large" amount (how much?) of data from storage into a user buffer
- include open and close costs (if any) in the measurement
- start in a state with no cached data on clients or servers
- avoid re-reading the same stored data during a single test
- avoid re-using the same bytes in the buffer during a single test

The goal of such a test is to measure the real storage system to user-buffer performance for a given workload.

[What about small accesses? Should they be included here in some way?]

6.9 Sustained Write Performance – Rob Ross, 03/15/2002

A test that measures "sustained write performance" must:

- write a "large" amount of data from a user buffer to storage
- make every effort to ensure the data reached the storage device(s)
- include open and close costs (if any) in the measurement
- start in a state with no cached data on clients or servers
- avoid re-writing to the same storage region during a single test
- avoid re-using the same bytes in the buffer during a single test
- provide a parameter for specifying new file creation or use of preallocated space

The goal of such a test is to measure the real user-buffer to storage system performance for a given workload.

[What about small accesses? Should they be included here in some way?]

6.10 Cached Write Performance – Rob Ross, 03/15/2002

A test that measures "cached write performance" must:

- write a "large" amount of data from a user buffer to storage
- include open and close costs (if any) in the measurement
- start in a state with no cached data on clients or servers
- avoid re-writing to the same storage region during a single test
- avoid re-using the same bytes in the buffer during a single test

The difference between this and "sustained write performance" tests is the relaxing of the requirement that data hit the disk. This allows one to measure bursty performance peaks. This type of performance can be important to applications that rely on writeback to overlap computation and I/O.

[What about small accesses? Should they be included here in some way?]

7 Consortium Members

This is a list of the "members" of the consortium. At this time this basically means that someone (maybe not the person) said that the individual should be listed here. These are in no particular order.

```
Mark Miller
Robb Matzke
Matt O'Brien
Tyce McLarty
Bill Loewe
Richard Hedges
Linnea Cook
Larry Schoof
Peter Espen
Jake Jones
Judy E. Sturtevant
Mike Folk
Albert Cheng
Elena Pourmal
Rob Ross
Rajeev Thakur
Rob Latham
Bill Anderson
Walt Ligon
Troy Baer
Rob Latham
Dave Glover
```

We don't want to leave anyone out that is interested! If someone else should be on the list, or if you want to be removed, please email Rob Ross <rross@mcs.anl.gov>.