published in the proceedings of the CUG SUMMIT 2001, Indian Wells, California, USA, May 21–25, 2001.

The parallel communication and I/O bandwidth benchmarks: b_eff and b_eff_io

Rolf Rabenseifner
RUS / HLRS
University of Stuttgart
Stuttgart, Germany
rabenseifner@hlrs.de

Alice E. Koniges
Lawrence Livermore
National Laboratory
Livermore, CA 94550, USA
koniges@llnl.gov

Abstract

We describe the design and MPI implementation of two benchmarks created to characterize the balanced system performance of high-performance clusters and supercomputers. We start with a communication-specific benchmark, called b_eff that characterizes the message passing performance of a system. Following the same line of development, we extend this work to the design and implementation of the effective I/O bandwidth benchmark (b_eff_io). Both of these benchmarks were developed on a Cray T3E-900 and have two goals: a) to get a detailed insight into the performance strengths and weaknesses of different parallel communication and I/O patterns, and b) to obtain a single bandwidth number that characterizes the average performance of the system namely processor communication for b_eff, and the I/O subsystem for b_eff_io. Both benchmarks use a time-driven approach and loop over a variety of communication and access patterns to characterize a system in a fairly automated fashion. Results of the two benchmarks are given for several systems including IBM SPs, Cray T3E, NEC SX-5, and Hitachi SR 8000.

1. Introduction

Characterization of a system's usable performance requires more than vendor-supplied tables of peak performance (Tflops), memory size (Tbytes), and other awe-inspiring statistics. On the other hand, a simple number characterization has much appeal in giving both the user of a system and those procuring a new system a basis for quick comparison. Indeed, the TOP500 [24] ordering (based on the Linpack performance) which characterizes a systems true or predicted application performance based on the computational speed of a system, is a number most often quoted in press releases detailing the world's fastest computers. Such application performance statistics are vital, yet do not tell the whole story. Usable high-performance systems require a balance between this application computational speed (as detailed by the TOP500 figures) and other aspects in particular communication scalability and I/O performance. We focus on these latter areas. We start with the design criteria of the effective bandwidth benchmark (b_eff), that characterizes the communication network of a distributed system [14, 21, 22], and extend the ideas to produce an additional benchmark which characterizes overall I/O performance. In this paper, we use the benchmarks to contrast several current high-performance computing systems in their production environment.

2. Design Criteria

There are several communication test suites that serve to characterize relative communication performance and I/O subsystem performance. Generally, the application of the benchmark suites results in a series of tabulated results for various types of tests (e.g., varying the communication pattern, the message size, etc.[10]). The key concept that differentiates the effective bandwidth benchmarks described here from these other test suites is the use of sampling techniques to automatically scan a subset of the parameter space and pick out key features, followed by averaging and use of maxima to combine the results into a single numerical value. Obviously, the difficultly lies in judicious choice of the benchmark parameters and averaging techniques. Additionally, both b_eff-type benchmarks are adjusted to give their results in an amount of time commensurate with the subsystem it is modeling. Specifically, the communication bandwidth achieves its result in 3-5 minutes, and the effective I/O bandwidth, adjusted appropriately for the slower I/O communication, reaches its conclusion in approximately 30 minutes. To get detailed insight, it is important to choose a set of patterns that reflects typical application kernels.

2.1. Effective Bandwidth Benchmark

The effective bandwidth benchmark (b_eff) measures the accumulated bandwidth of the communication network of parallel and/or distributed computing systems. Several message sizes, communication patterns and methods are used. A fundamental difference between the classical ping-pong benchmarks and this effective bandwidth benchmark is that **all** processes are sending messages to neighbors in parallel, i.e., at the same time. The algorithm uses an average to take into account that short and long messages are transferred with different bandwidth values in real application scenarios. The result of this benchmark is a single number, called the *effective bandwidth*. The averaging technique is described later in Sec. 4. Beside the patterns used in the averaging, additional patterns are measured to get more detailed information on the communication behavior of the system, e.g., two and three dimensional patterns are used to compare collective and non-collective MPI communication.

To measure the balance of a system, we introduce the *balance factor* defined as the ratio of interprocessor communication defined by b_eff and the floating point performance of an application. For the floating point performance, we use R_max as defined by the standard Linpack performance[24].

2.2. Effective I/O Bandwidth Benchmark

Most parallel I/O benchmarks and benchmarking studies characterize the hardware and file system performance limits [2, 5, 8, 9]. Often, they focus on determining under which conditions the maximal file system performance can be reached on a specific platform. Such results can guide the user in choosing an optimal access pattern for a given machine and file system, but do not generally consider the needs of the application over the needs of the file system.

To formulate b_eff_io, we first consider the likely I/O requests of parallel applications. We use the MPI-I/O interface [11] to implement the I/O requests. This interface serves both to express the user's needs in a concise fashion and to allow for optimized implementations based on the underlying file system characteristics [3, 13, 16, 17]. To be consistent with our benchmarking goals, we note that the effective I/O bandwidth benchmark (b_eff_io) should measure different access patterns, report the detailed results, and finally calculate an average I/O bandwidth value that characterizes the whole system.

A major difference between b_eff and b_eff_io is the magnitude of the bandwidth. On well-balanced systems in high performance computing we expect an **I/O** bandwidth which allows for writing or reading the total memory in approximately 10 **minutes**. (We call this the *coffee-cup rule,* since it is based on the application developers' requirement that a running application using most or all of the available memory for a given set of nodes should be able to perform its I/O needs by writing out approximately 1/2 of this memory during the 5 minutes it takes for the developer to go to another room and get a cup of coffee.) In contrast, early results from the **communication** bandwidth b_eff show that the total memory can be communicated in 3.2 **seconds** on a Cray T3E with 512 processors and in 13.6 seconds on a 24 processor Hitachi SR 8000, i.e., the I/O bandwidth is about two orders of magnitude slower than the communication bandwidth.

Another difference is that an I/O benchmark measures the bandwidth of data transfers between memory and disk. Such measurements are (1) highly influenced by buffering mechanisms of the underlying I/O middleware and filesystem details, and (2) high I/O bandwidth on disk requires, especially on striped filesystems, that a large amount of data must be transferred between these buffers and disk. Therefore an I/O benchmark must ensure that a sufficient amount of data is transferred between disk and the application's memory to minimize buffer effects. In practice, we find that our communication benchmark b_eff can give detailed answers in about 3-5 minutes, while the b_eff_io, our I/O counterpart, needs at least 30 minutes.

Although the design criteria for both benchmarks are similar, the benchmark software is very different. For b_eff, the time driven approach is realized by controlling the repeating factor for each loop. These are predefined at the beginning of each measurement loop and modified by the execution time of a previous loop. For b_eff_io, most loops are terminated when the measured time exceeds the scheduled time for a given access pattern.

3. Multidimensional Benchmarking Space

Often, benchmark calculations sample only a small subspace of a multidimensional parameter space. One extreme example is to measure only one point, e.g., a communication bandwidth between two processors using a ping-pong communication pattern with 8 Mbyte messages, repeated 100 times. Our goal here is to sample a reasonable amount of the relevant space.

3.1. Effective Bandwidth Benchmark

For communication benchmarks, the major parameters are message size, communication patterns, i.e., how many processes are communicating in parallel, how many messages are sent in parallel and which communication graph is used, and at least the communication method, e.g., whether MPI_Sendrecv, nonblocking or collective communication (MPI_Alltoallv) is used. For b_eff, 21 different message sizes are used, 13 fixed sizes (1 byte to 4 kb) and 8 variable sizes (from 4 kb to the 1/128 of the memory of each processor). The communication graphs are defined in two groups, (a) as rings of different sizes and (b) by a random polygon. Details are discussed later in the definition of the b_eff benchmark. A first approach by Karl Solchenbach, Hans-Joachim Plum and Gero Ritzenhoefer [22] was based on the bi-section bandwidth. This approach violates some of the benchmarking rules defined in [4, 6]. Therefore a redesign was necessary.

3.2. Effective I/O Bandwidth Benchmark

For I/O benchmarking, a huge number of parameters exist. We divide the parameters into 6 general categories. At the end of each category in the following list, a first hint about handling these aspects in b_eff_io is noted. The detailed definition of b_eff_io is given in Sec. 5.1.

1. Application parameters are (a) the size of contiguous chunks in the memory, (b) the size of contiguous chunks on disk, which may be different in the case of scatter/gather access patterns, (c) the number of such contiguous chunks that are accessed with each call to a read or write routine, (d) the file size, (e) the distribution scheme, e.g., segmented or long strides, short strides, random or regular, or separate files for each node, and (f) whether or not the chunk size and alignment are wellformed, e.g., a power of two or a multiple of the striping unit. For b_eff_io, 36 different patterns are used to cover most of these aspects.
2. Usage aspects are (a) how many processes are used and (b) how many parallel processors and threads are used for each process. To keep these aspects outside of the benchmark, b_eff_io is defined as a maximum over these aspects and one must report the usage parameters used to achieve this maximum.
3. The major programming interface parameter is specification of which I/O interface is used: Posix I/O buffered or raw, special filesystem I/O of the vendor's filesystem, or MPI-I/O. In this benchmark, we use only MPI-I/O, because it should be a portable interface of an optimal implementation on top of Posix I/O or the special filesystem I/O.
4. MPI-I/O defines the following orthogonal aspects: (a) access methods, i.e., first writing of a file, rewriting or reading, (b) positioning method, i.e., explicit offsets, individual or shared file pointers, (c) coordination, i.e., accessing the file collectively by (all) processes or noncollectively, (d) synchronism, i.e., blocking or nonblocking. Additional aspects are: (e) whether or not the files are open *unique*, i.e., the file will not be concurrently opened by a different open call, and (f) which consistency is chosen for conflicting accesses, i.e., whether or not atomic mode is set. For b_eff_io there is no overlap of I/O and computation, therefore only blocking calls are used. Because there should not be a significant difference between the efficiency of using explicit offsets or individual file pointers, only the individual and shared file pointers are benchmarked. With regard to (e) and (f), *unique* and *nonatomic* are used. All three access methods and five different pattern types implement a major subset of this parameter space. This is important to evaluate an MPI-I/O library on a given filesystem.
5. Filesystem parameters are (a) which filesystem is used, (b) how many nodes or processors are used as I/O servers, (c) how much memory is used as bufferspace on each application node, (d) the disk block size, (e) the striping unit size, and (f) the number of parallel striping devices that are used. These aspects are also outside the scope of b_eff_io. The chosen filesystem, its parameters and any usage of non-default parameters must be reported.
6. Additional benchmarking aspects are (a) repetition factors, and (b) how to calculate b_eff_io, based on a subspace of the parameter space defined above using maximum, average, weighted average or logarithmic averages.

To reduce benchmarking time to an acceptable amount, one can normally only measure I/O performance at a few grid points of a 1-5 dimensional subspace. To analyze more than 5 aspects, usually more than one subspace is examined. Often, the common area of these subspaces is chosen as the intersection of the area of best results of the other subspaces. For example in [8], the subspace varying the number of servers is obtained with segmented access patterns, and with well-chosen block sizes and client:server ratios. Defining such optimal subspaces can be highly system-dependent and may therefore not be as appropriate for a b_eff_io designed for a variety of systems. For the design of b_eff_io, it is important to choose the grid points based more on general application needs than on optimal system behavior. These needs were a major design goal in the standardization of MPI-2 [11]. Therefore the b_eff_io pattern types were chosen according to the key features of MPI-2. The exact definition of the pattern types are given in Sec. 5.1 and Fig. 1.

4. The Effective Bandwidth: Definition and Results

This section defines the patterns used to measure the communication bandwidth and the averaging rule for b_eff. Although this section mainly reports the averiging process, it is as same important, that all measured patterns are reported in the benchmark protocol and summarized in several categories (see Table 1) to allow a detailed analysis of a commu-

System	number of processors	b_eff MByte/s	b_eff per proc. MByte/s	L_{max}	ping-pong bandwidth MByte/s	b_eff at L_{max} MByte/s	b_eff per proc. at L_{max} MByte/s	b_eff per proc. at L_{max} ring pat.
Distributed memory systems								
Cray T3E/900-512	512	19919	39	1 MB	330	50018	98	193
	256	10056	39	1 MB	330	22738	89	190
	128	5620	44	1 MB	330	12664	99	195
	64	3159	49	1 MB	330	7044	110	192
	24	1522	63	1 MB	330	3407	142	205
	2	183	91	1 MB	330	421	210	210
Hitachi SR 8000 round-robin	128	3695	29	8 MB	776	11609	90	105
	24	915	38	8 MB	741	2764	115	110
Hitachi SR 8000 sequential	24	1806	75	8 MB	954	5415	226	400
Hitachi SR 2201	16	528	33	2 MB		1451	91	96
Shared memory systems								
NEC SX-5/8B	4	5439	1360	2 MB		35047	8762	8758
NEC SX-4/32	16	9670	604	2 MB		50250	3141	3242
	8	5766	641	2 MB		28439	3555	3552
	4	2622	656	2 MB		14254	3564	3552
HP-V 9000	7	435	62	8 MB		1135	162	162
SGI Cray SV1-B/16-8	15	1445	96	4 MB	994	5591	373	375

Table 1. Effective Benchmark Results

nication system. At the end of this section, some additional patterns are mentioned that are not part of the averaging process, but also important for this analysis.

The effective bandwidth is defined as (a) a logarithmic average over the ring patterns and the random patterns, (b) using the average over all message sizes, (c) and the maximum over all the three communication methods (d) of the bandwidth achieved for the given pattern, message size and communication method.

For the averaging over the message sizes can be expressed also by following formula: We take the number of MPI processes multiplied with the asymptotic bandwidth for the parallel communication patterns on each process, and reduce this result by a factor obtained by taking the area under the curve *bandwidth over message-sizes* divided by a rectangular area. The rectangular area is defined by the lengths of the abscissa and the asymptotic bandwidth.
As formula, the total definition can be expressed as:

$$
\begin{aligned}
\mathrm{b\_eff} = \; & \mathit{logavg} \\
& (\mathrm{logavg_{ringpatterns}}(\mathrm{sum_L}(\mathrm{max_{mthd}} \\
& \quad (\mathrm{max_{rep}}(\mathrm{b}(\mathrm{ringpat.}, \mathrm{L}, \mathrm{mthd}, \mathrm{rep}))))/21) \\
& , \mathrm{logavg_{randompatterns}}(\mathrm{sum_L}(\mathrm{max_{mthd}} \\
& \quad (\mathrm{max_{rep}}(\mathrm{b}(\mathrm{randompat.}, \mathrm{L}, \mathrm{mthd}, \mathrm{rep}))))/21) \\
&)
\end{aligned}
$$

with

- b(pat,L,mthd,rep) = L $*$ (total number of messages of a pattern "pat") $*$ looplength / (maximum time on each process for executing the communication pattern looplength times)
- Each measurement is repeated 3 times (rep=1..3). The maximum bandwidth of all repetitions is used (see $\mathrm{max_{mthd}}$ in the formula above).
- Each pattern is programmed with three methods. The maximum bandwidth of all methods is used ($\mathrm{max_{mthd}}$).
- The measurement is done for different sizes of a message. The message length L has the following 21 values: L = 1B, 2B, 4B, ... 2kB, 4kB, 4kB*(a**1), 4kB*(a**2), ... 4kB*(a**8) with and 4kB*(a**8) = L_{max} and L_{max} = (memory per processor) / 128 and looplength = 300 for the shortest message. The looplength is reduced dynamically to achieve a execution time for each loop between 2.5 and 5 msec. The minimum looplength is 1. The average of the bandwidth of all messages sizes is computed ($\mathrm{sum_L}$(...)/21).
- A set of ring patterns and random patterns is used (see details section below).
- The average for all ring patterns and the average of all random patterns is computed on the logarithmic scale ($\mathrm{logavg_{ringpatterns}}$ and $\mathrm{logavg_{randompatterns}}$)
- Finally the effective bandwidth is the logarithmic average of these two values: logavg($\mathrm{logavg_{ringpatterns}}$, $\mathrm{logavg_{randompatterns}}$)

Only for the detailed analysis of the communication behavior, the following additional patterns are measured:

- a worst case cycle,
- a best and a worst bi-section,
- the communication of a two dimensional Cartesian partitioning in the both directions separately and together,
- the same for a three dimensional Cartesian partitioning,
- a simple ping-pong between the first two MPI processes.

On communication methods: The communication is programmed with several methods. This allows the measurement of the effective bandwidth independent of which MPI methods are optimized on a given platform. The maximum bandwidth of the following methods is used:

- MPI_Sendrecv
- MPI_Alltoallv
- nonblocking with MPI_Irecv and MPI_Isend and MPI_Waitall.

On communication patterns: To produce a balanced measurement on any network topology, different communication patterns are used:

- Each node sends in each measurement a messages to its left neighbor in a ring and receives such a message from its right neighbor. Afterwards it sends a message back to its right neighbor and receives such a message from its left neighbor. Using the method MPI_Sendrecv, the two messages are sent one after the other in each node, if a ring has more than 2 processes. In all other cases, the two messages may be sent in parallel by the MPI implementation. Six ring patterns are used based on a one dimensional cyclic topology on MPI_COMM_WORLD:
  1. In the first ring pattern, all rings have the size 2, except the last ring which may have the size 2 or three. E.g. if MPI_COMM_WORLD has 7 processes, then the processes with the ranks 0 & 1 form the first ring, 2 & 3 form the second ring, and 4 & 5 & 6 form the third ring.
  2. In the second ring pattern, the ring size is 4, except the last rings, that may have the sizes 1*3, 1*5, or 2*5. If the number of processes is less or equal 7 then all processes form one ring.
  3. In the third ring pattern, the ring size is 8, except the last rings, that may have the sizes 3*7, ... 1*7, 1*9, ... 4*9. This rule cannot be used for less than 29 processes. In general the ring sizes are computed with ring_numbers.c [19]. For 2 to 28 processes the ring sizes of the 3rd pattern can be view in the list in [19] computed by this program.
  4. In the 4th ring pattern the standard ring size is min(max(16, size/4), size). The exact values can be viewed also in [19].
  5. In the 5th ring pattern the standard ring size is min(max(32, size/2), size).
  6. And in the last ring pattern, one ring includes all processes.

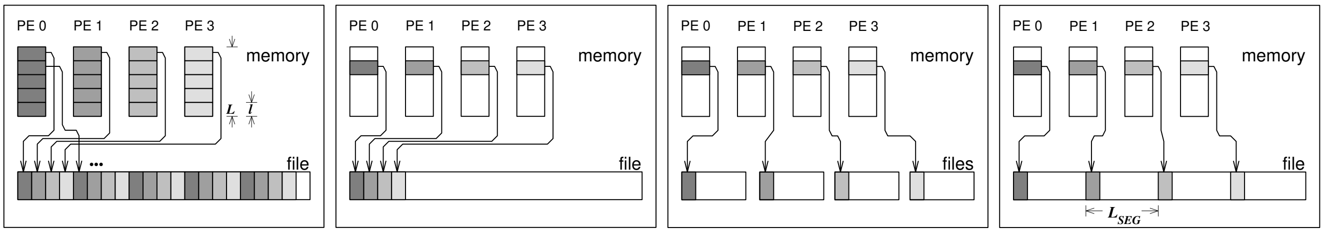
  In each ring, the processes are sorted by their ranks in the topology mentioned above.
- For the random patterns the same ring-communication as in the 1-dimensional topology is used, but the processes are sorted by random ranks.
- The average is computed in two steps to guarantee that the ring patterns and random patterns are weighted the same.

On maximum message size $\mathrm{L_{max}}$: On systems with sizeof(int)<64, $\mathrm{L_{max}}$ must be less or equal 128 MB, i.e., $\mathrm{L_{max}}$ = min(128 MB, (memory per processor)/128); on all other systems $\mathrm{L_{max}}$ is equal to the 128^{th} of the memory per processor.

The definition of b_eff can summarized as follows: The effective bandwidth is number of MPI processes multiplied with the asymptotic bandwidth multiplied with the ratio of the area under the curve *bandwidth over message-sizes* and the area under the horizontal constant asymptotic bandwidth line in the same diagram. To measure the bandwidth, several communication patterns are applied. The patterns are based on rings and on random distributions. The logarithmic average on all ring patterns and on all random patterns is computed and **b_eff** is the logarithmic average of these two values. The communication is implemented in three different ways with MPI and for each single measurement the maximum bandwidth of all three methods is used. For the ratio mentioned above the bandwidth is plotted over the message size and the used message sizes are plotted equidistant on the abscissa, i.e., along two logarithmic scales, one from 1 byte to 4 kbyte (12 intervals) and the next from 4 kbyte to $\mathrm{L_{max}}$ (8 intervals).

4.1. Effective Benchmark Results

Table 1 shows some results on distributed and shared memory platforms. On some platforms, either the total system was not available for the measurements or the system was not configured to be used by one dedicated MPI application. But the *b_eff per processor* column extrapolates to the network performance if all processors are communicating to a neighbor. On shared memory platforms, the results generally reflect half of the memory-to-memory copy bandwidth because most MPI implementations have to buffer the message in a shared memory section. To compare these results with the traditional asymptotic ping-pong bandwidth for large message sizes, one should remember that b_eff is defined as an average over several message sizes. In the last three columns, the result is based only on the maximum message size $\mathrm{L_{max}}$. In the last column, only the ring patterns are used. Comparing the last two columns, we see the negative effect of random neighbor locations. Comparing the last column with ping-pong results from the vendor we see the impact of communicating in parallel on each processor. For example, on a T3E the asymptotic ping-pong bandwidth is about 300 MByte/s, and for 2 processors. In contrast, b_eff per processor is 210 MByte/s. For ring patterns, there is virtually no degradation for larger number of processes. The measurement protocols can be found in [14]. The Hitachi results depend on the numbering of the



pattern type 0 | pattern type 1 | pattern type 2 | pattern type 3/4

Figure 1. Data transfer patterns used in b_eff_io.

Pattern Type	No.	l	L	U
0: scatter, collect.	0	1 MB	1 MB	0
	1	M_{PART}	:=l	4
	2	1 MB	2 MB	4
	3	1 MB	1 MB	4
	4	32 kB	1 MB	2
	5	1 kB	1 MB	2
	6	32 kB +8B	1 MB + 256B	2
	7	1 kB +8B	1 MB + 8kB	2
	8	1 MB +8B	1 MB + 8B	2
1: shared, collect.	9	1 MB	:=l	0
	10	M_{PART}	:=l	4
	11	1 MB	:=l	2
	12	32 kB	:=l	1
	13	1 kB	:=l	1
	14	32 kB +8B	:=l	1
	15	1 kB +8B	:=l	1
	16	1 MB +8B	:=l	2

Pattern Type	No.	l	L	U
2: separated files, non-coll.	17	1 MB	:=l	0
	18	M_{PART}	:=l	2
	19	1 MB	:=l	2
	20	32 kB	:=l	1
	21	1 kB	:=l	1
	22	32 kB +8B	:=l	1
	23	1 kB +8B	:=l	1
	24	1 MB +8B	:=l	2
3: segmented, non-coll.	25f	same as patterns 17–24		
	33	fill up segments	:=l	0
4: segmented, collective	34f	same as patterns 25–33		
				ΣU = 64

Table 2. The pattern details used in b_eff_io

MPI processes on the cluster of SMP nodes: *round-robin* means, that the numbering starts with the first processor on each SMP node, *sequential* means, that first all processors of the first SMP node are used, and so on. The numbering has a heavy impact on the communication bandwidth of the ring patterns and therefore of the b_eff result.

5. The I/O Benchmark: Definition and Results

The benchmark b_eff_io should characterize the I/O capabilities of the system. Should we use, therefore, only access patterns, that promise a maximum bandwidth? No, but there should be a good chance that an optimized implementation of MPI-I/O should be able to achieve a high bandwidth. This means that we should measure patterns that can be recommended to application developers.

An important criterion is that the b_eff_io benchmark should only need about 10 to 15 minutes for a given number of processes, i.e., that two or three measurements with different node numbers can be taken in 30 minutes. For first measurements, it need not run on an empty system as long as concurrently running other applications do not use a significant part of the I/O bandwidth of the system. Normally, the full I/O bandwidth can be reached by using less than the total number of available processors or SMP nodes. In contrast, the communication benchmark b_eff should not require more than 3-5 minutes, but it must run on the whole system to compute the aggregate communication bandwidth.

Based on the rule for well-balanced systems mentioned in the introduction and assuming that MPI-I/O will attain at least 50 percent of the hardware I/O bandwidth, we expect that a 10 minute b_eff_io run can write or read about 16 % of the total memory of the benchmarked system. For this estimate, we divide the total benchmark time into three intervals based on the following access methods: initial write, rewrite, and read. However, a first test on a T3E900-512 shows that based on the pattern-mix, only about the third of this theoretical value is transferred.

Finally, as a third important criterion, we want to be able to compare different common access patterns.

5.1. Definition of the Effective I/O Bandwidth

The effective I/O bandwidth benchmark measures the following aspects:

- a *set of partitions*: a partition is defined by the number of nodes used for the b_eff_io benchmark and – if a node is a multiprocessor node – by the number of MPI processes on each node,
- the access methods *initial write*, *rewrite*, and *read*,
- the *pattern types* (see Fig. 1)
  (0) strided collective access, scattering large chunks in memory with size L each with one MPI-I/O call to/from disk chunks with size l,
  (1) strided collective access, but one read or write call per disk chunk,
  (2) noncollective access to one file per MPI process, i.e., on separated files,
  (3) same as (2), but the individual files are assembled to one segmented file, and
  (4) same as (3), but the access to the segmented file is done with collective routines;
  for each pattern type, an individual file is used.
- the contiguous chunk size is chosen *wellformed*, i.e., as a power of 2, and *non-wellformed* by adding 8 bytes to the wellformed size,
- different chunk sizes, mainly 1 kB, 32 kB, 1 MB, and the maximum of 2 MB and $1/128$ of the memory size of a node executing one MPI process.

The total list of patterns is shown in Table 2. A pattern is a pattern type combined with a fixed chunk size and alignment of the first byte[1]. The column "l" defines the contiguous chunks that are written from memory to disk and vice versa. The value M_{PART} is defined as *max(2 MB, memory of one node / 128)*. The column "L" defines the contiguous chunk in the memory. In case of pattern type (0), non-contiguous fileviews are used. If l is less than L,, then in each MPI-I/O read/write call, the L bytes in memory are scattered/gathered to/from the portions of l bytes at the different locations on disk, see the left-most scenario in Fig. 1. In all other cases, the contiguous chunk handled by each call to MPI_Write or MPI_Read is equivalent in memory and on disk. This is denoted by ":=l" in the L column. U is a time unit.

Each pattern is benchmarked by repeating the pattern for a given amount of time. For write access, this loop is finished with a call to MPI_File_sync. This time is given by the allowed time for a whole partition (e.g., $T = 15$ minutes) multiplied by $U/\Sigma U/3$, as given in the table. This time-driven approach allows one to limit the total execution time. For the pattern types (3) and (4) a fixed segment size must be computed before starting the pattern of these types. Therefore, the time-driven approach is substituted by a size-driven approach, and the repeating factors are initialized based on the measurements for types (0) to (2).

The b_eff_io value **of one pattern type** is defined as the total number of transferred bytes divided by the total amount of time from opening till closing the file. The b_eff_io value **of one access method** is defined as the average of all pattern types with double weighting of the scattering type. The b_eff_io value **of one partition** is defined as the average of the access methods with the weights 25 % for *initial write*, 25 % for *rewrite*, and 50 % for *read*. **The b_eff_io of a system** is defined as the maximum over any b_eff_io of a single partition of the system, measured with a scheduled execution time T of at least 15 minutes. This definition permits the user of the benchmark to freely choose the usage aspects and enlarge the total filesize as desired. The minimum filesize is given by the bandwidth for an initial write multiplied by 300 sec (= 15 minutes / 3 access methods). For using this benchmark to compare systems as in the TOP 500 list, more restrictive rules are under development.

5.2. Comparing Systems Using b_eff_io

First, we test b_eff_io on two systems, the Cray T3E900-512 at HLRS/RUS in Stuttgart and an RS 6000/SP system at LLNL called "blue Pacific." Figure 2 shows the b_eff_io values for different partition sizes and different values of T.

On the T3E, we use the tmp-filesystem with 10 striped Raid-disks connected via a GigaRing for the benchmark. The peak-performance of the aggregated parallel bandwidth of this hardware configuration is about 300 MB/s. The LLNL results presented here are for an SP system with 336 SMP nodes each with four 332 MHz processors. Since the I/O performance on this system does not increase significantly with the number of processors on a given node performing I/O, all test results assume a single thread on a given node is doing the I/O. Thus, a 64 processor run means 64 nodes assigned to I/O, and no requested computation by the additional 64*3 processors. On the SP system, the data is written to the IBM General Parallel File System (GPFS) called blue.llnl.gov:/g/g1 which has 20 VSD I/O servers. Recent results for this system show a maximum read performance of approximately 950MB/sec for a 128 node job, and a maximum write performance of 690MB/sec for 64 nodes [8].[2] Note that these are the maximum values observed, and performance degrades when the access pattern and/or the node number is changed.

For this data on both platforms pre-releases (Rel. 0.x) of

[1]The alignment is implicitly defined by the data written by all previous patterns in the same pattern type

[2]Upgrades to the AIX operating system and underlying GPFS software may have altered these performance numbers slightly between measurements in [8] and in the current work. Additionally, continual upgrades to AIX and GPFS are bringing about improved performance overall.

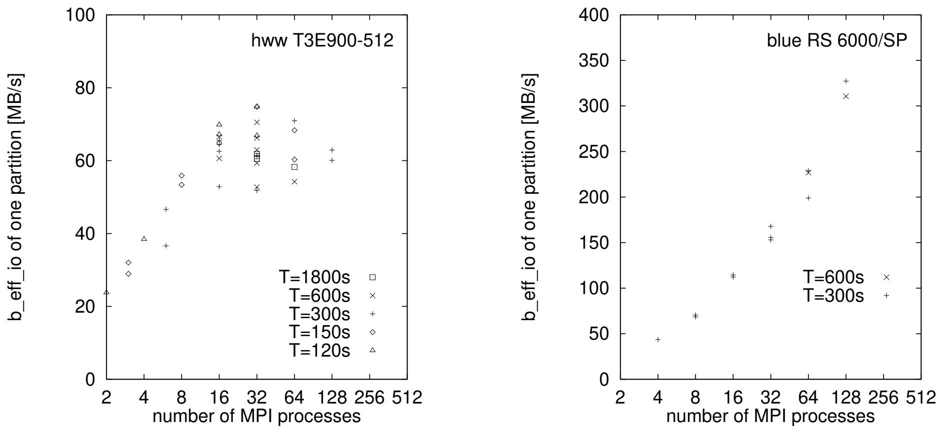


Figure 2. Comparison of b_eff_io for different numbers of processes at HLRS and LLNL, measured partially without pattern type 3. Here T is in seconds, b_eff_io releases 0.x.

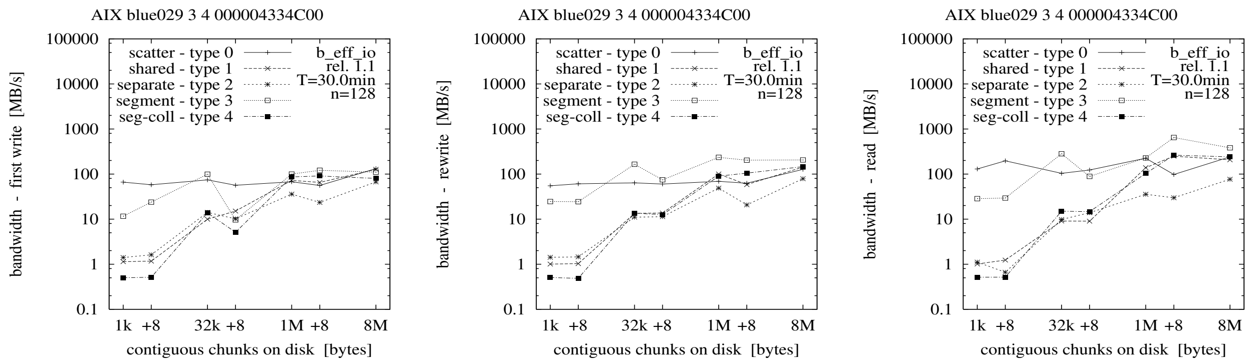
b_eff_io were used that had a different weighting of the patterns (type 0, type 1, etc). Therefore the values presented in this section cannot be directly compared with the results in the next section. MPI-I/O was implemented with ROMIO but with different device drivers. On the T3E, we have modified the MPI Release mpt.1.3.0.2, by substituting the ROMIO/ADIO Unix filesystem driver routines for opening, writing and reading files. The Posix routines were substituted by the asynchronous counter part, directly followed by the the wait routine. This trick enables parallel disk access [18]. On the RS 6000/SP blue machine, GPFS is used underneath the MPICH version of MPI with ROMIO. Figure 2 shows the b_eff_io values for different partition sizes and different values of T, the time scheduled for benchmarking one partition. All measurements were taken in a non-dedicated mode.

Besides the different absolute values that correlate to the amount of memory in each system, one can see very different behavior. For the T3E, the maximum is reached at 32 application processes, with little variation from 8 to 128 processors, i.e., the I/O bandwidth is a global resource. In contrast, on the IBM SP the I/O bandwidth tracks the number of compute nodes until it saturates. In general, an application only makes I/O requests for a small fraction of the compute time. On large systems, such as those at the High-Performance Computing Center at Stuttgart and the Computing Center at Lawrence Livermore National Laboratory, several applications are sharing the I/O nodes, especially during prime time usage. In this situation, I/O capabilities would not be requested by a significant proportion of the CPU's at the same time. "Hero" runs, where one application ties up the entire machine for a single calculation are rarer and generally run during non-prime time. Such hero runs can require the full I/O performance by all processors at the same time. The right-most diagram shows that the RS 6000/SP fits more to this latter usage model. Note that GPFS on the SP's is configurable, i.e., number of I/O servers and other tunables, and the performance on any given SP/GPFS system depends on the configuration of that system.

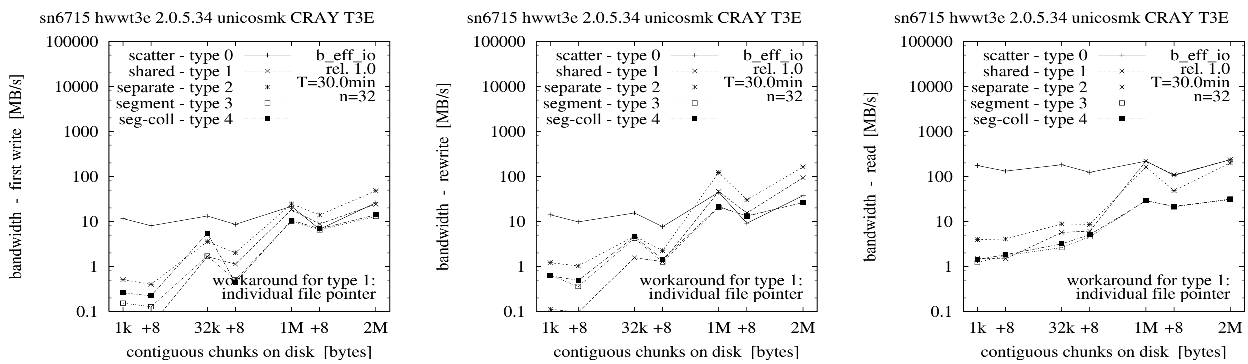
5.3. Detailed Insight

In this section, we present a detailed analysis of each run of b_eff_io on a partition. For each run of b_eff_io, the I/O bandwidth for each chunk size and pattern is reported in a table that can be plotted as in the pictures shown in each row in Fig. 3. The three diagrams in each row show the bandwidth achieved for the three different access methods: writing the file the first time, rewriting the same file, and reading it. On each diagram, the bandwidth is plotted on a logarithmic scale, separately for each pattern type and as a function of the chunk size. The chunk size on disk is shown on a pseudo-logarithmic scale. The points labeled "+8" are the non-wellformed counterparts of the power of two values. The maximum chunk size is different on both systems because the maximum chunk size was chosen proportional to the usable memory size per node to reflect the scaling up of applications on larger systems. On the SX-5, a reduced maximum chunk size was used. Except on NEC SX-5, we have used b_eff_io releases 1.x. On the IBM SP, a new MPI-I/O prototype was used. This prototype is used for the development and improvement of MPI-I/O. Performance of the actual product may vary.

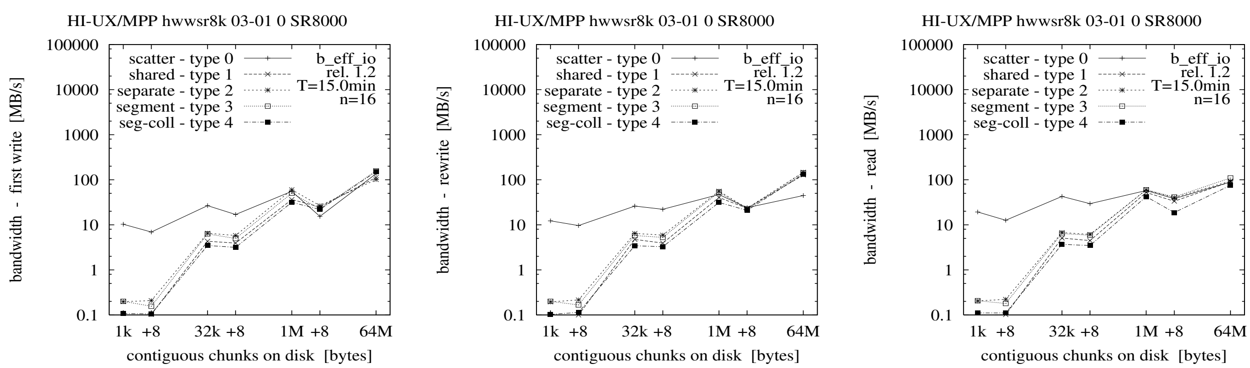
The four rows compare the I/O bandwidth on four different systems from IBM, Cray, Hitachi and NEC. The IBM SP and the Cray T3E were described in the last section. The NEC SX-5 system has four striped RAID-3 arrays DS 1200,



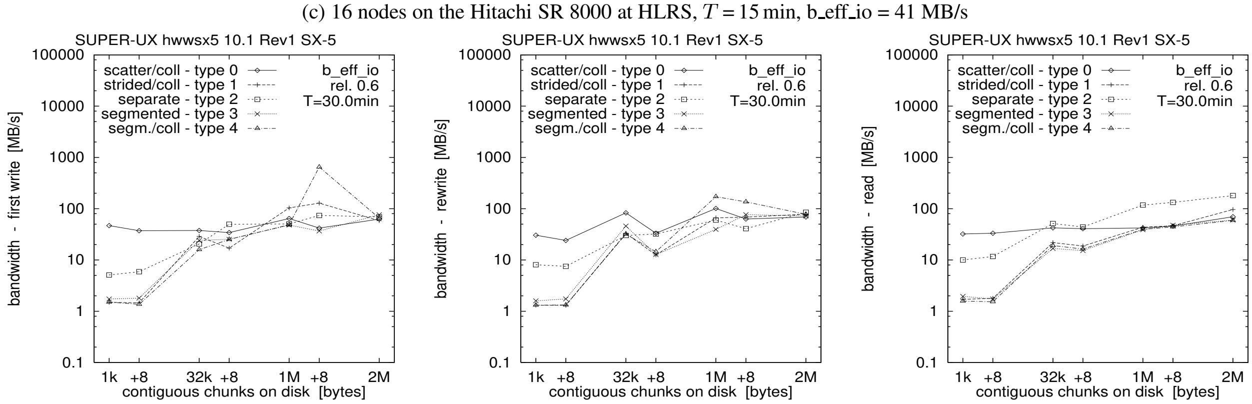
(a) 128 nodes on the "blue Pacific" RS 6000/SP at LLNL, $T = 30$ min, b_eff_io = 63 MB/s



(b) 32 PEs on the T3E900-512 at HLRS, $T = 30$ min, b_eff_io = 57 MB/s



(c) 16 nodes on the Hitachi SR 8000 at HLRS, $T = 15$ min, b_eff_io = 41 MB/s



(d) 4 processors on the NEC SX-5Be/32M2 at HLRS, $T = 30$ min, b_eff_io = 60 MB/s

Figure 3. Comparison of the results for optimal numbers of processes on T3E and SP, and for $T = 10$ and 30 min.

connected by fibre channel. The SFS filesystem parameters are: 4 MB cluster size (=block size), and if the size of an I/O request is less than 1 MB then a 2 GB filesystem-cache is used. On the SX-5, we use MPI/SX 10.1.

First notice that the scattering pattern type 0 is the best on all platforms for small chunk sizes on disk. Thus all I/O implementations can effectively handle the 1 MB memory chunks that are given in each MPI-I/O call to be scattered to disk or gathered from disk. In all other pattern types, the memory chunk sizes per call are identical to the disk chunk sizes, i.e., in the case of 1 kB or 32 kB, only a small or medium amount of data is accessed per call on disk.

Note that due to the logarithmic scale, a vertical difference of about 7 mm reflects an order of magnitude change! Comparing the wellformed and non-wellformed measurements, especially on the T3E, there are huge differences. Also on the T3E, we see a large gap between write and read performance in the scattering pattern type.

On the IBM SP MPI-I/O prototype, one can see that segmented non-collective pattern type 3 is also optimized. On the other hand, the collective counterpart is more than a factor of 10 worse. Such benchmarking can help to uncover advantages and weakness of an I/O implementation and can therefore help in the optimization process. In the case of the prototype, hints are allowed based on specific I/O patterns and these can drastically increase the performance. (See e.g., Prost, et al. [13]). These hints are necessarily pattern specific, since if they worked for all pattern types, they would naturally be a part of the standard MPI I/O implementation. In the future release of b_eff_io, we plan to allow such special hints for each pattern type or pattern as introduced by the MPI-2 standard with the *info* argument.

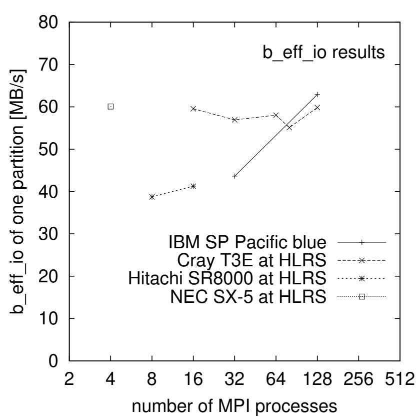


Figure 4. Comparison of b_eff_io for different numbers of processes at HLRS and LLNL, measured partially without pattern type 3. B_eff_io releases 1.x, except for NEC.

Fig. 4 compares the final b_eff_io results on these four platforms. Values for other partition sizes are added.

In general, our results show that the b_eff_io benchmark is a very fast method to analyze the parallel I/O capabilities available for applications using the standardized MPI-I/O programming interface. The resulting b_eff_io value summarizes I/O capabilities of a system in one significant I/O bandwidth value.

5.4. Discussion of b_eff_io

In this section, given the primary results of the benchmark, we reflect on some details of its definition. The design of the b_eff_io tries to follow the rules about MPI benchmarking defined by Bill Gropp, Ewing Lusk [4] and Rolf Hempel [6], but there are a few problematic topics.

Normally, the same experiment should be **repeated** a few times to compute a **maximal bandwidth**. To achieve a very fast I/O benchmark suite, this methodology is substituted by **weighted averaging** over a medium number of experiments i.e., the patterns. (We note that in the case of the IBM SP data, repeated calculation of b_eff_io on different days produced nearly identical answers). The weighted averaging is done for each experiment after calculating the average bandwidth over all repetitions of the same pattern. Any maximum is calculated only after repeating the total b_eff_io benchmark itself. For this maximum, one may vary the number of client processes, the schedule time T, and file system parameters.

The major problem with this definition is that one may use any schedule time T with $T > 10$ minutes. First experiments on the T3E have shown that the b_eff_io value may have its maximum for $T = 10$ minutes. This is likely since for any larger time interval, the caching of the filesytem in the memory is reduced.

Indeed, **caching issues** may be problematic for I/O benchmarks in general. For example, Rolf Hempel [7] has reported that on SX-5 systems other benchmark programs have reported a bandwidth significantly higher than the hardware peak performance of the disks. This is caused by a huge 4 GB memory cache used by the filesytem. In other words, the measurement is not able to guarantee that the data was actually written to disk. To help assure that data is written, we can add MPI_File_sync. The problem is, however, that MPI_File_sync influences only the consistency semantics. Calling MPI_File_sync after writing on a file, guarantees that any other process can read this newly written data, but it does **not** guarantee that the data is stored on a permanent storage medium, i.e., that the data is written to disk. There is only one way to guarantee, that the MPI-I/O routines have stored 95 % of the written data to disk: One must write a dataset 20 times larger than the memory cache length of the filesystem. This can be controlled by verifying that the datasize accessed by each b_eff_io access method is larger than 20 times of the filesystems' cache length.

The next problem arises from the **time driven approach** of this benchmark: A pattern is repeating for a given time interval, which is $T_{pattern} = T/3 * U/\Sigma U$ for each pattern. The termination condition must be computed after each call to a write or read routine. In all patterns defining a collective fileview or using collective write or read routines, the termination condition must be computed globally to guarantee that all processes are stopped after the same iteration. In the current version this is done by computing the criterion only at a root process. The local clock is read after a barrier synchronization. Then, the decision is broadcasted to all other nodes. This termination algorithm is based on the assumption that a barrier followed by a broadcast is at least 10 times faster than a single read or write access. For example, the fastest access on the T3E for $L = 1$ kB chunks is about 4 MB/s, i.e., $250\,\mu$s per call. In contrast, a barrier followed by a broadcast needs only about $60\,\mu$s on 32 PEs, which is not 10 times faster than a single I/O call. Therefore, this termination algorithm should be modified in future versions of this benchmark. Instead of computing the termination criterion in each iteration, a geometric series of increasing repeating factors should be used.

Pattern types 3 and 4 require a predefined **segment size** L_{SEG}, see Fig. 1. In the current version, for each chunk size "l", a repeating factor is calculated from the measured repeating factors of the pattern types 0–2. The segment size is calculated as the sum of the chunk sizes multiplied by these repeating factors. The sum is rounded up to the next multiple of 1 MB. This algorithm has two drawbacks:

1. The alignment of the segments are multiples of 1 MB. If the striping unit is more than 1 MB, then the alignment of the segments is not wellformed.

2. On systems with **32 bit integer/int** datatype, the segment size multiplied by the number of processes (n) may be more than 2 GB, which may cause internal errors inside of the MPI library. Without such internal restrictions, the maximum segment size would be 16/n GB, based on a 8 byte element type. If the segment size must be reduced due to these restrictions, then the total amount of data written by each processes does no longer fit into one segment.

On large MPP systems, it may be also necessary to reduce the **maximal chunk size** (M_{PART}) to 2/n GB or 16/n GB. This restriction is necessary for the pattern types 0, 1, 3 and 4.

Another aspect is the mode used to open the benchmark files. Although we want to benchmark **unique** mode, i.e., ensure that a file is not accessed by other applications while it is open by the benchmark program, we must **not** use MPI_MODE_UNIQUE_OPEN because it would allow an MPI-I/O implementation to delay all MPI_File_sync operations until the closing of the file.

6. The Time-Driven Approach

Figure 2 shows interesting results. There is a difference between the maximum I/O bandwidth and the sampled bandwidth for several partition sizes. In the redesign from release 0.x to 1.x we have incorporated that the averaging for each pattern type can not be done by using the average of the bandwidth values for all chunk sizes. The bandwidth of one pattern must be computed as the total amount of transfered data divided by the total amount of time used for all chunk sizes. With this approach, it is possible to reduce caching effects and to allow a total scheduled time of 30 minutes for measuring all five patterns with the three access directions (write, rewrite, read) for **one** compute partition size.

Both benchmarks are proposed for the *Top Clusters* list [23]. For this, the I/O benchmark can be done automatically in 30 minutes for **three** compute partition sizes. This is implemented by reorganizing the sequence of the experiments: First, all files are written with the three different compute partition sizes, followed by rewriting, and then by all reading. Additionally, the rewriting experiments only use pattern type 0. Of course, if one wants to achieve very specific results, one can run this b_eff_io release 2.0 benchmark for the longer time period and with all rewriting patterns included.

7. Current and Future Work

We plan to use this benchmark to compare several additional systems. Although [1] stated, that "the majority of the request patterns are sequential", we should examine whether random access patterns can be included into the b_eff_io benchmark. It is planned to use both benchmarks in the *Top Clusters* list [23]. For this, it is necessary that the I/O benchmark can be done automatically in 30 minutes. Both benchmarks will also be enhanced to write an additional output that can be used in the SKaMPI *comparison page* [20].

8. Summary

In this paper we have described in detail two benchmarks, the effective bandwidth and its I/O counterpart. We use these two benchmarks to characterize the performance of common computing platforms. We have shown how these benchmarks can provide both detailed insight into the performance of high-performance platforms and how they can reduce these data to a single number averaging important information about that system's performance. We give suggestions for interpreting and improving the benchmarks, and for testing the benchmarks on one's own system.

9. Acknowledgments

The authors would like to acknowledge their colleagues and all the people that supported these projects with suggestions and helpful discussions. At HLRS, they would especially like to thank Karl Solchenbach and Rolf Hempel for productive discussions for the redesign of b_eff. At LLNL, they especially thank Kim Yates. We also gratefully acknowledge discussions with Jean-Pierre Prost and Richard Treumann of IBM. Work at LLNL was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

References

[1] P. Crandall, R. Aydt, A. Chien, D. Reed, *Input-Output Characteristics of Scalable Parallel Applications*, In Proceedings of Supercomputing '95, ACM Press, Dec. 1995, `www.supercomp.org/sc95/proceedings/`.

[2] Ulrich Detert, *High-Performance I/O on Cray T3E*, 40th Cray User Group Conference, June 1998.

[3] Philip M. Dickens, *A Performance Study of Two-Phase I/O*, in D. Pritchard, J. Reeve (eds.), Proceedings of the 4th International Euro-Par Conference, Euro-Par'98, Parallel Processing, LNCS–1470, pages 959–965, Southampton, UK, 1998.

[4] William Gropp and Ewing Lusk, *Reproducible Measurement of MPI Performance Characteristics*, in J. Dongarra et al. (eds.), Recent Advances in Parallel Virtual Machine and Message Passing Interface, proceedings of the 6th European PVM/MPI Users' Group Meeting, EuroPVM/MPI'99, Barcelona, Spain, Sept. 26-29, 1999, LNCS 1697, pp 11–18. (Summary on the web: `www.mcs.anl.gov/mpi/mpptest/hownot.html`).

[5] Peter W. Haas, *Scalability and Performance of Distributed I/O on Massively Parallel Processors*, 40th Cray User Group Conference, June 1998.

[6] Rolf Hempel, *Basic Message Passing Benchmarks, Methodology and Pitfalls*, SPEC Workshop on Benchmarking Parallel and High-Performance Computing Systems, Wuppertal, Germany, Sept. 13, 1999, `www.hlrs.de/mpi/b_eff/hempel_wuppertal.ppt`.

[7] Rolf Hempel, Huber Ritzdorf, *MPI/SX for Multi-Node SX-5*, SX-5 Programming Workshop, High-Performance Computing-Center, University of Stuttgart, Germany, Feb. 14–17, 2000, `www.hlrs.de/news/events/2000/sx5.html`.

[8] Terry Jones, Alice Koniges, R. Kim Yates, *Performance of the IBM General Parallel File System*, to be published in Proceedings of the International Parallel and Distributed Processing Symposium, May 2000. Also available as UCRL JC135828.

[9] Kent Koeninger, *Performance Tips for GigaRing Disk I/O*, 40th Cray User Group Conference, June 1998.

[10] Glenn R. Luecke and James J. Coyle, *Comparing the Performance of MPI on the Cray T3E-900, the Cray Origin 2000, and the IBM P2SC*, `hpc-journals.ecs.soton.ac.uk/PEMCS/Papers`.

[11] Message Passing Interface Forum. *MPI-2: Extensions to the Message-Passing Interface*, July 1997, `www.mpi-forum.org`.

[12] *Frangipani: A Scalable Distributed File System*, `http://www.research.digital.com/SRC/personal/thekkath/frangipani/home.html`

[13] J.P. Prost, R. Treumann, R. Blackmore, C. Harman, R. Hedges, B. Jia, A. Koniges, A. White, *Towards a High-Performance and Robust Implementation of MPI-IO on top of GPFS*, EuroPar2000, Munich, August 2000,in A. Bode et al. (Eds.): Euro-Par 2000, LNCS 1900, pp. 1253–1262, 2000. (Springer-Verlag: Berlin).

[14] Rolf Rabenseifner, *Effective Bandwidth (b_eff) Benchmark*, `www.hlrs.de/mpi/b_eff/`.

[15] Rolf Rabenseifner, *Effective I/O Bandwidth (b_eff_io) Benchmark*, `www.hlrs.de/mpi/b_eff_io/`.

[16] Rajeev Thakur, William Gropp, and Ewing Lusk, *On Implementing MPI-IO Portably and with High Performance*, in Proc. of the Sixth Workshop on I/O in Parallel and Distributed Systems, pages 23–32, May 1999.

[17] Rajeev Thakur, Rusty Lusk, Bill Gropp, *ROMIO: A High-Performance, Portable MPI-IO Implementation*, `www.mcs.anl.gov/romio/`.

[18] Rolf Rabenseifner, *Striped MPI-I/O with mpt.1.3.0.1*, `www.hlrs.de/mpi/mpi_t3e.html#StripedIO`.

[19] Rolf Rabenseifner, *Ring Pattern List*, Nov. 1999. `www.hlrs.de/mpi/ring_pattern_list` and `www.hlrs.de/mpi/ring_numbers.c`

[20] Ralf Reussner, Peter Sanders, Lutz Prechelt and Matthias Müller, *SKaMPI: A detailed, accurate MPI benchmark*, in Recent Advances in Parallel Virtual Machine and Message Passing Interface, 5th European PVM/MPI Users' Group Meeting, LNCS 1497, pages 52-59, 1998. `wwwipd.ira.uka.de/~skampi/`

[21] Karl Solchenbach, *Benchmarking the Balance of Parallel Computers*, SPEC Workshop on Benchmarking Parallel and High-Performance Computing Systems, Wuppertal, Germany, Sept. 13, 1999.

[22] Karl Solchenbach, Hans-Joachim Plum and Gero Ritzenhoefer, *Pallas Effective Bandwidth Benchmark – source code and sample results*, `ftp://ftp.pallas.de/pub/PALLAS/PMB/EFF_BW.tar.gz`.

[23] TFCC – IEEE Task Force on Cluster Computing, `www.ieeetfcc.org`, and Top Clusters `www.TopClusters.org`.

[24] Universities of Mannheim and Tennessee, *TOP500 Supercomputer Sites*, `www.top500.org`.