The Effective I/O Bandwidth Benchmark (b_eff_io)

Rolf Rabenseifner[1] and Alice E. Koniges[2]

[1]High-Performance Computing-Center (HLRS),
Rechenzentrum Universität Stuttgart,
Allmandring 30, D-70550 Stuttgart, Germany
rabenseifner@hlrs.de, www.hlrs.de/people/rabenseifner/

[2]Lawrence Livermore National Laboratory, Livermore, CA 94550
koniges@llnl.gov, www.ipp.mpg.de/~ack

Abstract— **The effective I/O bandwidth benchmark (b_eff_io) covers two goals: (1) to achieve a characteristic average number for the I/O bandwidth achievable with parallel MPI-I/O applications, and (2) to get detailed information about several access patterns and buffer lengths. The benchmark examines "first write", "rewrite" and "read" access, strided (individual and shared pointers) and segmented collective patterns on one file per application and non-collective access to one file per process. The number of parallel accessing processes is also varied and wellformed I/O is compared with non-wellformed. On systems, meeting the rule that the total memory can be written to disk in 10 minutes, the benchmark should not need more than 15 minutes for a first pass of all patterns. The benchmark is designed analogously to the effective bandwidth benchmark for message passing (b_eff) that characterizes the message passing capabilities of a system in a few minutes. First results of the b_eff_io benchmark are given for IBM SP and Cray T3E systems and compared with existing benchmarks based on parallel Posix-I/O.**

Keywords— **MPI, File-I/O, Disk-I/O, Benchmark, Bandwidth.**

I. Introduction

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.

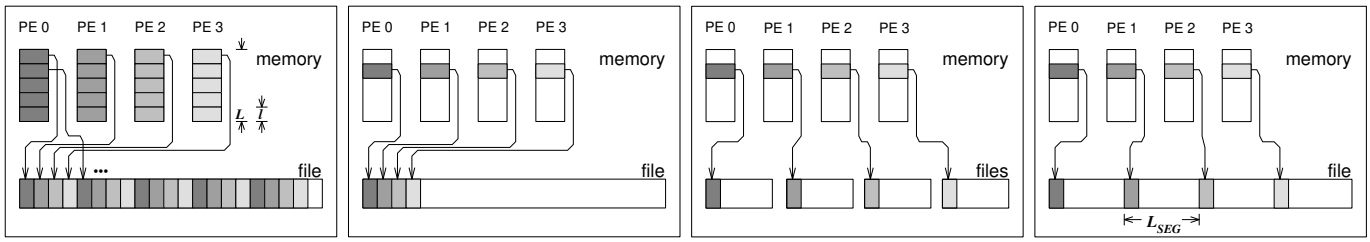
Our approach begins with consideration of the possible I/O requests of parallel applications. To formulate such I/O requests, the MPI Forum has standardized the MPI-I/O interface [10]. Major goals of this standardization are to express the user's needs and to allow optimal implementations of the MPI-I/O interface on all platforms [3], [11], [14], [15]. Based on this background, the effective I/O bandwidth benchmark (b_eff_io) should measure different access patterns, report these detailed results, and should calculate an average I/O bandwidth value that characterizes the whole system. This goal is analogue to the Linpack value reported in TOP500 [19] that characterizes the computational speed of a system, and also to the effective bandwidth benchmark (b_eff), that characterizes the communication network of a distributed system [12], [17], [18].

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 a I/O bandwidth which allows for writing or reading the total memory in approximately 10 **minutes**. For the communication bandwidth, the b_eff benchmark shows, 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. 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 such buffers and disk. Therefore a benchmark must ensure that a sufficient amount of data is transfered between disk and the application's memory. The communication benchmark b_eff can give detailed answers in about 2 minutes. Later we shall see that b_eff_io, our I/O counterpart, needs at least 15 minutes to get a first answer.

II. 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. 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 the aspects in b_eff_io is given. The detailed definition of b_eff_io is shown in section IV.

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.



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

Fig. 1. Data transfer of one (collective) call to MPI_Write_...

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 will be defined as a maximum over these aspects and one must report the usage parameters used to achieve the 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 vendors filesytem, 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., that the file will not be concurrently opened elsewhere, 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 the additional aspects, *unique* and *nonatomic* should be used.
5. Filesystem parameters are (a) how many nodes or processors are used as I/O servers, (b) how much memory is used as bufferspace on each application node, (c) the disk block size, (d) the striping unit size, and (e) the number of parallel striping devices that are used. These aspects are also outside the scope of b_eff_io. Any usage of non-default parameters must be reported.
6. Additional benchmarking aspects are (a) repetition factors, (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.

III. Criteria

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 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 2 minutes, but it must run on the whole system to compute the aggregate communication bandwidth.

Based on the rule mentioned in the introduction and expecting that MPI-I/O will offer at least 50 percent of the hardware I/O bandwidth, we can expect that a 10 minute b_eff_io run will transfer about half of the total memory of the benchmarked system. A first test on a T3E900-512 shows that based on the pattern-mix, only about the third of this theoretical value is transferred.

As third important criterion, we want to be able to compare different common access patterns.

IV. Definition of the Effective I/O Bandwidth

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

- a *set of partitions*,
- the access methods *initial write*, *rewrite*, and *read*,
- the *pattern types* (see Fig. 1)

Pattern Type	No.	l	L	U
0	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	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
1				*continued*
	15	1 kB +8B	$:=l$	1
	16	1 MB +8B	$:=l$	2
2	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	25f	same as patterns 17–24		
	33	fill up segments	$:=l$	0
4	34f	same as patterns 25–33		
			$\Sigma U = 64$	

TABLE I
THE PATTERN DETAILS USED IN B_EFF_IO

(0) strided collective access, scattering large chunks in memory to/from disk,
(1) strided collective access, but one read or write call per disk chunk,
(2) noncollective access to one file per MPI process,
(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 I. 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), scattering is done by repeating to write l bytes by each process to disk. 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., $T = 10$ 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).

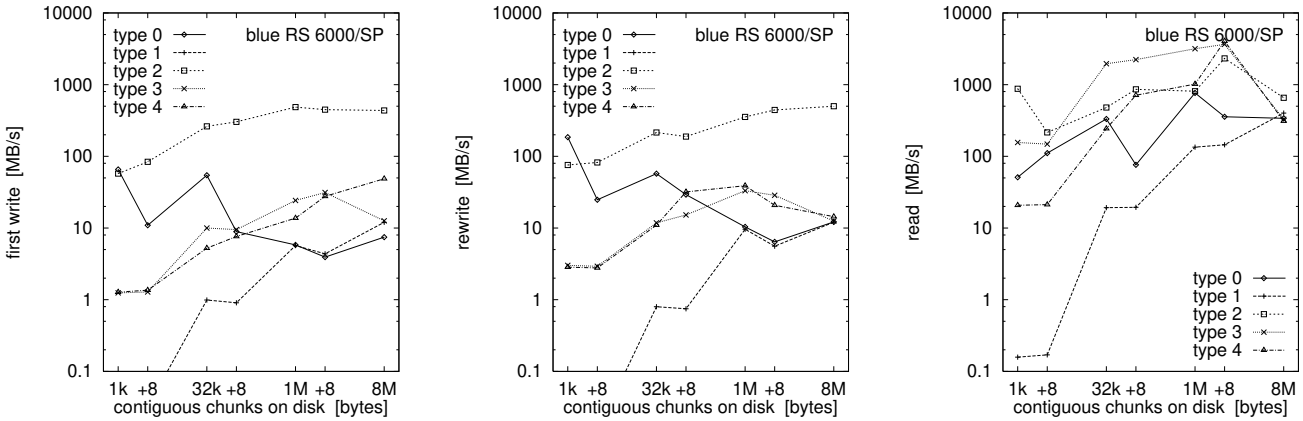
[1]The aligment is implicitly defined by the data written by all previous patterns in the same pattern type

The b_eff_io value **of one partition** is defined as sum of all transferred bytes divided by the total transfer time. If patterns do not need exactly the ideal allowed time, then the average is weighted by the unit U. At minimum, 10 minutes must be used for benchmarking one partition.

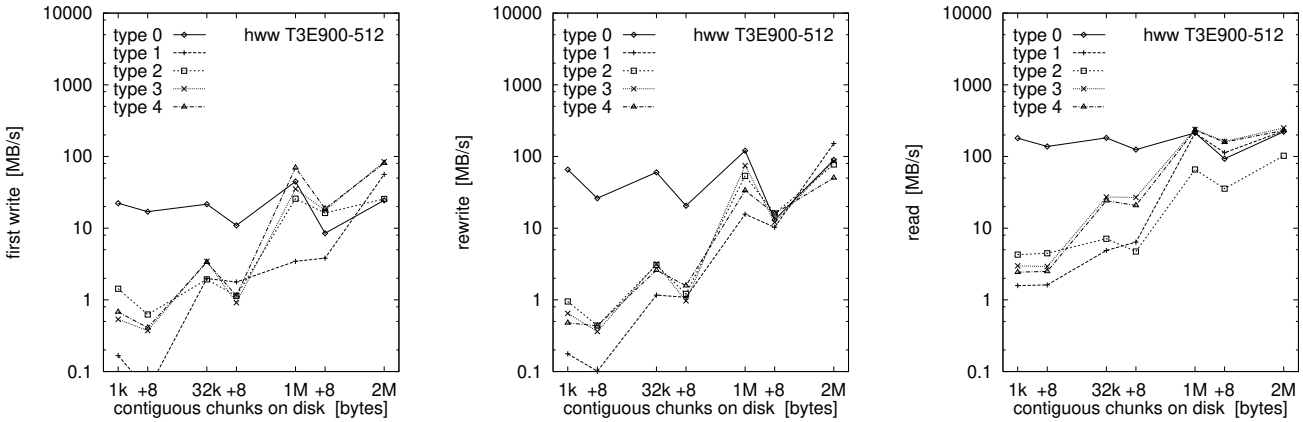
The b_eff_io of a system is defined as the maximum over any b_eff_io of a single partition of the system. This definition permits the user of the benchmark to freely choose the usage aspects and enlarge the total filesize as desired. The minimum filesize is given by the bandwidth for an initial write multiplied by 200 sec (= 10 minutes / 3 access methods).

V. Comparing Systems Using b_eff_io

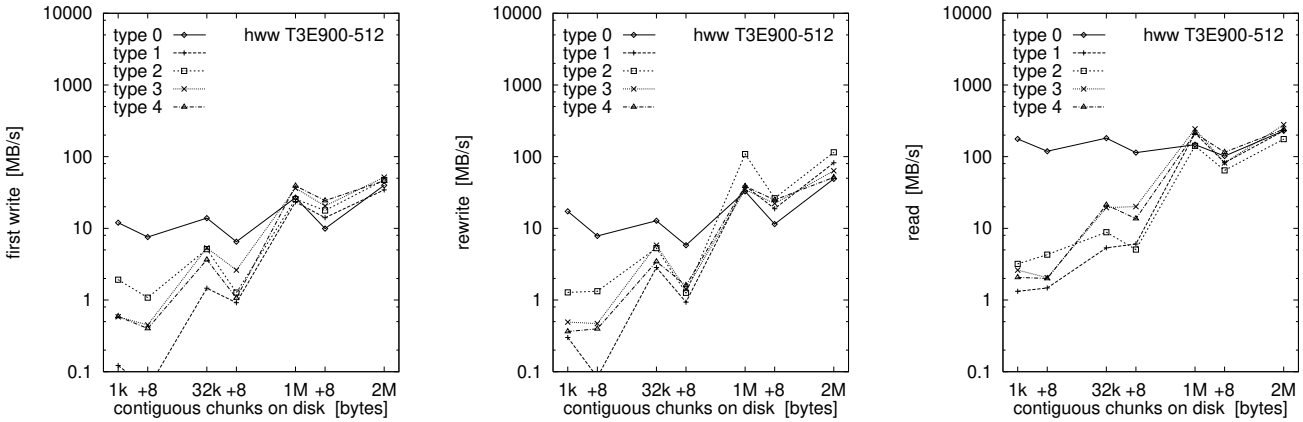
In this section, we present a detailed analysis of each run of b_eff_io on a partition. We test b_eff_io on two systems, the Cray T3E900-512 at HLRS/RUS in Stuttgart and an RS 6000/SP system at LLNL called "blue." On the T3E, we use the tmp-filesystem with 10 striped Raid-disks connected via a GigaRing 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



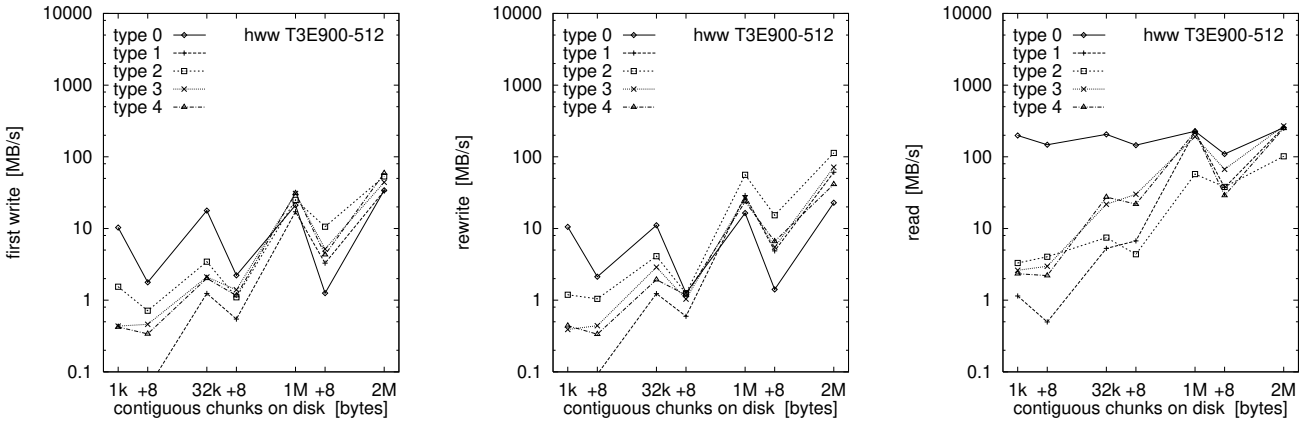
(a) 128 PEs on the "blue" RS 6000/SP at LLNL, $T = 10$ min, b_eff_io = 311 MB/s



(b) 32 PEs on the T3E900-512 at HLRS, $T = 10$ min, b_eff_io = 71 MB/s



(c) 32 PEs on the T3E900-512 at HLRS, $T = 30$ min, b_eff_io = 62 MB/s



(d) 64 PEs on the T3E900-512 at HLRS, $T = 30$ min, b_eff_io = 58 MB/s

Fig. 2. Comparison of the results for optimal numbers of processes on T3E and SP, and for $T = 10$ and 30 min.

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.

On both platforms, MPI-I/O is 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 [16]. On the RS 6000/SP blue machine, GPFS is used underneath the MPICH version of MPI with ROMIO.

Each row in fig. 2 shows the result of one benchmark on a system. Rows (a) and (b) compare the RS 6000/SP at LLNL with the T3E900-512 at HLRS, both benchmarked with an optimal number of processors and with $T = 10$ minutes. Rows (b) and (c) compare different values for the scheduled time T on the T3E, and rows (c) and (d) compare different numbers of processors used on the T3E. All benchmarks were captured while other applications were running on the other processors of the systems.

First, we look at **rows (a) and (b)**. They demonstrate the main differences between both MPI and filesystem implementations on SP at LLNL and T3E at HLRS. Based on the results in Fig. 3, which we discuss later on, we decided to run the benchmark on the T3E on 32 processors and on the RS 6000/SP on 128 processors. The three diagrams in each row of Fig. 2 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.

Type 0 is a strided access, but the buffer used in each I/O-call is at least 1 MB. In the case of a chunk length less than 1 MB, the buffer contents must be scattered to different places in the file. On the T3E, this pattern type is optimal, except for chunks larger than 1 MB, where the initial write of segmented files is faster. When non-wellformed chunk sizes are used, there is a substantial drop in performance. Additional measurements show that this problem increases with the total amount of data written to disk. On the RS 6000/SP, other pattern types show higher bandwidth.

Type 1 writes the same data to disk, i.e., each process has the same logical fileview, but MPI-IO is called for each chunk separately. In the current benchmark, this test is done with individual filepointers, because the MPI-I/O ROMIO implementation on both systems does not have shared filepointers. As default, b_eff_io measures this pattern type with shared pointers. On both platforms, this pattern type results in essentially the worst bandwidth for most access method and chunk size.

Type 2 is the writing winner on RS 6000/SP. Each process writes a separate file at the same time, i.e., parallel and independently. Type 3 writes the same, but the files of all processes are concatenated. To guarantee wellformed starting points for each process, the filesize of each process is rounded up to the next MByte. Type 4 writes the same as type 3, but the access is done collectively. On the T3E, we see that these three pattern types are consistently slow for small buffer sizes and consistently fast for large buffer sizes. In contrast on the RS 6000/SP, type 3 and 4 are about a factor[3] of **10–20** slower than type 2 for writing files. For reading files, the diagram cannot show the real speed for type 3 and 4 due to three effects: The repetition factor is only one for chunk sizes of 1 MB and more, the reading of the 8 MB chunk fills internal buffers, and currently, the b_eff_io does not perform a file sync operation before reading a pattern. Looking at the (non-weighted) average, one can see, that on the RS 6000/SP, reading the segmented files is a factor of **2.5** slower than reading individual files.

Finally, one can say that on both systems, the read access is clearly faster than the write access. On the T3E, the read access is 5 times faster than "first write" and 2.7 faster than "rewrite". On the RS 6000/SP blue machine, the read access is 10 times faster than both types of write access. The measurements were done with b_eff_io Release 0.5 [13]. By default, it measures 10 minutes the partition on which it was started, and 5 minutes the half partition.

Rows (b) and (c) in Fig. 2 compare T3E results for $T = 10$ and 30 minutes, respectively. With $T = 10$ min, 4.5 GB are written or read for each access method. With $T = 30$ min, the accumulated length of all files was 12.4 GB[4]. There are no significant differences between the two measurements, except for one curve: rewriting one file with pattern type 0 and $T = 30$ min shows a reduced bandwidth.

Comparing **rows (c) and (d)**, one can see that writing and rewriting non-wellformed chunk sizes result in a significantly worse bandwidth for 64 PEs on the T3E.

Figure 3 shows the b_eff_io values for different partition sizes and different values of T. All measurements were taken in a non-dedicated mode. For the T3E, one can see, that the maximum is reached at 32 application processes, but from 8 to 128 processors, there is only little variation. 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 nodes, especially during prime time usage. In this situation, I/O capabilities would not be requested by a sig-

[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.

[3] All factors in this section are computed, based on weighted averages using the time units U, if not stated else.

[4] The total amount of bytes written is less than b_eff_io$*T/3$ because the measurements for some patterns with smaller bandwidth have consumed more time than allowed according to the scheduled time $T/3 * U/\Sigma U$. This may be caused by additional execution time in MPI_File_sync, by the termination algorithm described in Chap. VI, and by the size driven approach for pattern types 3 and 4.

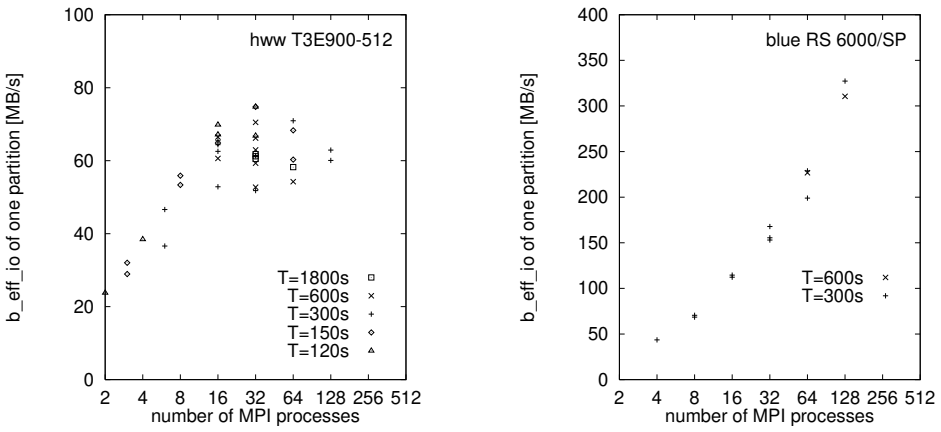


Fig. 3. Comparison of b_eff_io for different numbers of processes at HLRS and LLNL, measured partially without type 3. Here T is in seconds.

nificant 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 need the full I/O performance by all processors at the same time. The right diagram shows that the RS 6000/SP fits more to the 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.

Figure 3 also shows that on both systems, the results depend more on the I/O usage of the other concurrently running applications on the system than on the requested time T for each benchmark. Comparison of measurements with $T = 10$ and 30 minutes have shown that the analysis reported in Fig. 2 may vary in details. The differences between wellformed and non-wellformed I/O is more notable with $T = 30$ minuntes on the T3E.

Finally, we compare these results with other measurements. On the same RS 6000/SP, Posix read and write measurements ranging between 500 and 900 MB/s are measured [8].[5] The b_eff_io result is 311 MB/s in the presented measurement. This means that the MPI application programmer has a real chance to get a significant part of the I/O capabilities of that system. On the T3E studied, the peak I/O-performance is about 300 MB/s. Thus the b_eff_io value of 71 MB/s shows that on average, only a quarter of the peak can be attained with normal MPI programming. We also note that the ROMIO implementation on the RS 6000/SP has not been optimized for the GPFS filesystem. Vendor implementations and future versions of ROMIO should show performance closer to peak.

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] Again we note that upgrades to the AIX operating system and underlying GPFS software may have slightly altered these performance numbers between measurements.

VI. Details of b_eff_io

Following this presentation of the major results of this 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. This 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 filessytem 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 filessytem. 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 μs per call. In contrast, a barrier followed by a broadcast needs only about 60 μs 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/nGB or 16/nGB. 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.

If a system complies with our rule that the total memory can be written in 10 minutes for each access pattern, then one third of the total memory is written by this benchmark, and in each single pattern with U=1, one 1/192 of the total memory is written. If all processors are used for this benchmark, then the amount written by each node is not very much, but a call to MPI_File_sync in each pattern should guarantee that the data is really written to disk. If the benchmark is executed with n of N nodes and N is the total number of nodes of the system, then each of these n nodes has to write/read

$$M_{accessed\,per\,node} = M_{node} \frac{U}{3\Sigma U} \frac{b_{real}}{b_{rule}} \frac{N}{n} \frac{T}{10min}$$

with M_{node} = the memory size of one node, b_{real} = the real aggregated bandwidth for that pattern and $b_{rule} = N * M_{node}/10\,min$, the already mentioned I/O rule for HPC systems. The last two terms of the equation show that, as long as b_{real} is independent of n and T, the memory accessed in each process is linear in T/n, i.e., if one uses only half of the processors, then one can run the benchmark in half of the time, provided that one wants to write the same amount of data by each process. Based on these observations, a default run of b_eff_io first measures the I/O bandwidth with $T = 10$ minutes on the total partition of processors on which b_eff_io was started, and follows with $T = 5$ minutes on half of that partition.

Finally, with pattern types 3 and 4, it may be that all disk allocation is done with the first initial write pattern, which is not weighted in the average. This is shown in Table I by a zero U value for the patterns 0, 9, 17, 25 and 34.

VII. Outlook

It is planned, to use this benchmark to compare several systems. More investigation is necessary in the problems arising from 32 bit integer limits and handling read buffers in combination with file sync operations. 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.

Acknowledgments

The authors would like to acknowledge their colleagues and all the people that supported this project 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 thank Kim Yates and Dave Fox. 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 Internatinal 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] Message Passing Interface Forum. *MPI-2: Extensions to the Message-Passing Interface*, July 1997, `www.mpi-forum.org`.

[11] 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, Internal report.

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

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

[14] 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.

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

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

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

[18] 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`.

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

Alice E. Koniges is a Physicist and Member of the Parallel I/O research effort at the Lawrence Livermore National Laboratory (LLNL) in California. As leader of the Parallel Applications Technology Program at LLNL, she directed researchers in the largest set of agreements between industries and national laboratories ever funded by the U.S. Department of Energy. She has served as a consultant to the Max Planck Institutes of Germany on parallelization and high performance computing issues. She is editor of the book, "Industrial Strength Parallel Computing," recently published by Morgan Kaufmann Publishers of San Francisco. She has a Ph.D. in Applied and Numerical Mathematics from Princeton University, an MA and an MSME from Princeton, and a BA from the University of California, San Diego.

Rolf Rabenseifner studied mathematics and physics at the University of Stuttgart. Since 1984, he has worked at the High-Performance Computing-Center Stuttgart (HLRS). He led the projects DFN-RPC, a remote procedure call tool, and MPI-GLUE, the first metacomputing MPI combining different vendor's MPIs without loosing the full MPI interface. In his dissertation, he developed a controlled logical clock as global time for trace-based profiling of parallel and distributed applications. Since 1996, he has been a member of the MPI-2 Forum. From January to April 1999, he was an invited reseacher at the Center for High-Performance Computing at Dresden University of Technology. Currently, he is responsible for message passing programming models at the HLRS and he is involved in MPI profiling, benchmarking and teaching projects.