Published in Recent Advances in Parallel Virtual Machine and Message Passing Interface, Jack Dongarra and Yiannis Cotronis (Eds.), Proceedings of the 8th European PVM/MPI Users' Group Meeting, Santorini, Greece, Sep. 23–26, LNCS, Springer-Verlag, 2001.


# Effective Communication and File-I/O Bandwidth Benchmarks

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

[1] High-Performance Computing-Center (HLRS), University of 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.rzg.mpg.de/~ack

**Abstract.** We describe the design and MPI implementation of two benchmarks created to characterize the balanced system performance of high-performance clusters and supercomputers: b_eff, the communication-specific benchmark examines the parallel message passing performance of a system, and b_eff_io, which characterizes the effective I/O bandwidth. Both benchmarks have two goals: a) to get a detailed insight into the performance strengths and weaknesses of different parallel communication and I/O patterns, and based on this, b) to obtain a *single* bandwidth number that characterizes the *average* performance of the system namely communication and I/O bandwidth. Both benchmarks use a time-driven approach and loop over a variety of communication and access patterns to characterize a system in an automated fashion. Results of the two benchmarks are given for several systems including IBM SPs, Cray T3E, NEC SX-5, and Hitachi SR 8000. After a redesign of b_eff_io, I/O bandwidth results for several compute partition sizes are achieved in an appropriate time for rapid benchmarking.

## 1 Introduction and Design Criteria

Characterization of a system's usable performance requires more than vendor-supplied tables such as peak performance or memory size. On the other hand, a simple number characterizing the computational speed (as detailed by the TOP500 figures [8]) has much appeal in giving both the user of a system and those procuring a new system a basis for quick comparison. Such application performance statistics are vital and most often quoted in press releases, yet do not tell the whole story. Usable high-performance systems require a balance between this computational speed and other aspects in particular communication scalability and I/O performance. We focus on these latter areas.

There are several communication test suites that serve to characterize relative communication performance and I/O performance. 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. But this single value is only half of our goal. The **detailed insight** given by the numerous results for each measured pattern is the second salient feature. Additionally, both benchmarks are optimized in their execution time; b_eff needs about 3-5 minutes to examine its communication patterns, and b_eff_io, adjusted appropriately for the slower I/O communication, needs about 30 minutes. To get detailed insight, it is important to choose a set of patterns that reflects typical application kernels.

**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*.

**Effective I/O Bandwidth Benchmark:** Most parallel I/O benchmarks and benchmarking studies characterize the hardware and file system performance limits [1, 5]. Often, they focus on determining conditions that maximize file system performance. To formulate b_eff_io, we first consider the likely I/O requests of parallel applications using the MPI-I/O interface [7]. 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 [2, 9, 11]. Based on our benchmarking goals, note that the effective I/O bandwidth benchmark (b_eff_io) should measure different access patterns, report the detailed results, and calculate an average I/O bandwidth value that characterizes the whole system. Notably, 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. On well-balanced systems an **I/O** bandwidth should be sufficient to write or read the total memory in approximately 10 **minutes**. Based on this rule, an I/O benchmark should be able to examine several patterns in 30 minutes accounting for buffer effects.

## 2 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. Our goal here is to sample a reasonable amount of the relevant space.

**Effective Bandwidth Benchmark:** For communication benchmarks, the major parameters are message size, communication patterns, (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 (MPI_Sendrecv, nonblocking or collective communication, e.g., MPI_Alltoallv). 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 [16, 17] was based on the bi-section bandwidth, but it has violated some of the benchmarking rules defined in [3, 4]. Therefore a redesign was necessary.

**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. 4.

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 parameters are (a) how many processes are used and (b) how many parallel processors and threads are used for each process. To keep these parameters outside of the benchmark, b_eff_io is defined as a maximum over these parameters and one must report the usage parameters used to achieve this maximum. Filesystem parameters are also outside the scope of b_eff_io.
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, which is a standard designed for high performance parallel I/O [12] and therefore used in b_eff_io.
4. MPI-I/O defines the following orthogonal parameters: (a) access methods, i.e., first writing of a file, rewriting or reading, (b) positioning method, (c) collective or noncollective coordination, (d) synchronism, i.e., blocking or not. For b_eff_io there is no overlap of I/O and computation, therefore only blocking calls are used. Because explicit offsets are semantically identical to individual file pointers, only the individual and shared file pointers are benchmarked. All three access methods and five different pattern types implement a major subset of this parameter space.

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 [7]. 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. 4 and Fig. 1.

## 3 The Effective Bandwidth: Definition and Results

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. As formula, the total definition can be expressed as:

$$\begin{aligned} \text{b\_eff} = \mathit{logavg} \\ & (\ \text{logavg}_{\text{ringpat.s}}\ (\text{avg}_{\text{L}}\ (\text{max}_{\text{mthd}}\ (\text{max}_{\text{rep}}(\text{b}_{\text{ringpat.,L,mthd,rep}}\ )))\ ) \\ & ,\text{logavg}_{\text{randompat.s}}\ (\text{avg}_{\text{L}}\ (\text{max}_{\text{mthd}}\ (\text{max}_{\text{rep}}(\text{b}_{\text{randompat.,L,mthd,rep}}\ )))\ )\ ) \end{aligned}$$

with $\text{b}_{\text{pat,L,mthd,rep}} = \text{L} *$ (total number of messages of a pattern "pat") $*$ looplength / (maximum time on each process for executing the communication pattern looplength times)

Additional rules are: Each measurement is repeated 3 times (rep=1..3). The maximum bandwidth of all repetitions is used (see $\text{max}_{\text{mthd}}$ in the formula above). Each pattern is programmed with three methods. The maximum bandwidth of all methods is used ($\text{max}_{\text{mthd}}$). 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) = $\text{L}_{\text{max}}$ and $\text{L}_{\text{max}}$ = (memory per processor) / 128 and looplength = 300 for the shortest message. The looplength is dynamically reduced to achieve an 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 ($\text{sum}_{\text{L}}$(...)/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 (geometric average): $\text{logavg}_{\text{ringpatterns}}$ and $\text{logavg}_{\text{randompatterns}}$. Finally the effective bandwidth is the logarithmic average of these two values: logavg($\text{logavg}_{\text{ringpatterns}}$, $\text{logavg}_{\text{randompatterns}}$).

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: (a) MPI_Sendrecv, (b) MPI_Alltoallv, and (c) 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: In the first ring pattern, all rings have the size 2 (except the last ring which may have the size 2 or three). In the 2nd and 3rd ring pattern, the size of each ring is 4 and 8 (except last

| System | number of processors | b_eff MByte/s | b_eff per proc. MByte/s | $L_{max}$ | ping-pong bandw. MB/s | b_eff at $L_{max}$ MB/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

rings, see [14]). In the 4th and 5th ring pattern the standard ring size is max(16, size/4) and max(32, size/2). And in the 6th ring pattern, one ring includes all processes. For the random patterns, one ring with all processes 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 $L_{max}$:** On systems with sizeof(int)<64, $L_{max}$ must be less or equal 128 MB, i.e., $L_{max} = \min(128\text{ MB}, \text{(memory per processor)}/128)$; on all other systems $L_{max}$ is equal to the $128^{th}$ of the memory per processor.

**Averaging method:** The effective bandwidth value should represent the accumulated communication capability of the total system usable for large-scale applications. The geometric mean, i.e., the average on the logarithmic values, is chosen because it takes all network components into account. Thus the fastest connection and the slowest one are considered. The arithmetic mean is not used because the average would never be less than 50 %[1] of the fastest bandwidth value and thus the lower bound of the average is independent of the magnitude of the lowest measured value, i.e., the speed of the weakest network component.

**Latency:** For small message sizes, the b_eff value is dominated by latency effects, starting with a message size of one byte. The value *b_eff per process* is determined mainly by the asymptotic bandwidth ($b_\infty$) and by the message size that is necessary to achieve a significant part of $b_\infty$. If the transfer time $t$ is modeled by $latency + messagesize/b_\infty$, then $b_\infty/2$ is achieved if the message size is larger than $latency * b_\infty$. This product can be viewed as a characteristic value for the balance of latency and bandwidth. The first 13 message sizes are fixed, i.e., independent of the memory size of the system, to reduce the influence

[1] 50 % in the case of averaging two values. (One Hundred%/$n$ in the case of $n$ values.)

of the memory size on calculating *b_eff per process* as function of *latency* and $b_\infty$. The upper 9 message sizes depend on the memory size and reflect that the size of a large application's message typically scales with the application's data size, which in turn scales with the size of the available memory.

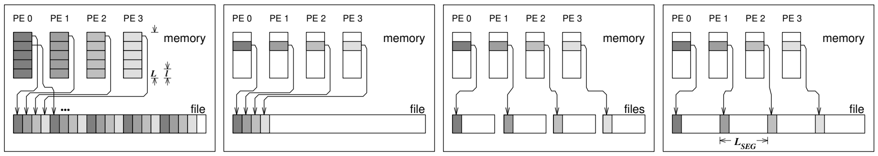
### 3.1 Effective Benchmark Results

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 $L_{max}$. 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 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 [10]. The Hitachi results depend on the numbering of the 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.

## 4 The I/O Benchmark: Definition and Results

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) is the same as (2), but the individual files are assembled to one segmented file; (4) is the 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,




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

**Fig. 1.** Data transfer patterns used in b_eff_io. Each diagram shows the data transferred by **one** MPI-I/O write call.

| 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 | | |
| | | | | |
| | | | | |
| | | | | $\Sigma U = 64$ |

**Table 2.** The pattern details used in b_eff_io

- 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[2]. 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

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

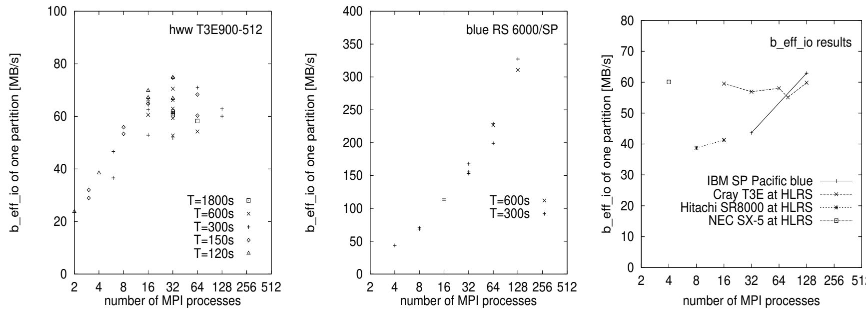



**Fig. 2.** Comparison of b_eff_io for different numbers of processes at HLRS and LLNL, measured partially without pattern type 3. Here $T$ is in seconds, b_eff_io releases 0.x (left pictures and NEC on right picture) and release 1.x (right picture).

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.

### 4.1 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 $T$, 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 Com-

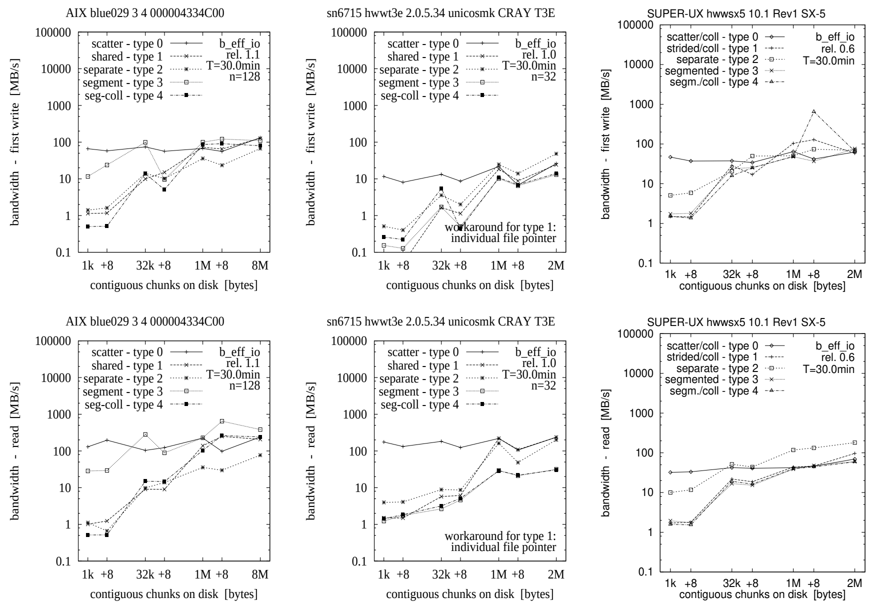



**Fig. 3.** Comparison of the results for optimal numbers of processes on
– IBM RS 6000/SP *blue Pacific* at LLNL, 128 nodes used, b_eff_io = 63 MB/s,
– Cray T3E-900/512 at HLRS, 32 PEs used, b_eff_io = 57 MB/s [13],
– NEC SX5-5Be/32M2 at HLRS, 4 CPUs used, b_eff_io = 60 MB/s.

puting 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 middle diagram shows that the RS 6000/SP fits more to this latter usage model.

The b_eff_io benchmark gives also a detailed insight into the I/O bandwidth for several chunk sizes and patterns. The bandwidth is reported in a table that can be plotted as in the pictures shown in each column in Fig. 3. The two diagrams in each column show the bandwidth achieved for *writing* and *reading* with different patterns and chunk sizes. The *rewriting*-diagrams are omited because they show similar values as the *writing*-diagrams on these platforms. 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 the 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. Further topics on b_eff_io results are discussed in [6].

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 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 [18]. 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.

## 6 Summary and Future Work

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.

We plan to use this benchmark to compare several additional systems. Both benchmarks will also be enhanced to write an additional output that can be used in the SKaMPI *comparison page* [15].

## Acknowledgments

The authors would like to acknowledge their colleagues and all the people that supported these projects with suggestions and helpful discussions. They would especially like to thank Karl Solchenbach and Rolf Hempel for productive discussions for the redesign of b_eff. 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. Ulrich Detert, *High-Performance I/O on Cray T3E*; Peter W. Haas, *Scalability and Performance of Distributed I/O on Massively Parallel Processors*; Kent Koeninger, *Performance Tips for GigaRing Disk I/O*; 40th Cray User Group Conf., June 1998.
2. 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.
3. William Gropp and Ewing Lusk, *Reproducible Measurement of MPI Performance Characteristics*, in J. Dongarra et al. (eds.), 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`).
4. 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`.
5. 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.
6. Alice E. Koniges, Rolf Rabenseifner, Karl Solchenbach, *Benchmark Design for Characterization of Balanced High-Performance Architectures*, in proceedings, 15th International Parallel and Distributed Processing Symposium (IPDPS'01), Workshop on Massively Parallel Processing, April 23-27, 2001, San Francisco, USA.
7. Message Passing Interface Forum. *MPI-2: Extensions to the Message-Passing Interface*, July 1997, `www.mpi-forum.org`.
8. Hans Meuer, Erich Strohmaier, Jack Dongarra, Horst D. Simon, *TOP500 Supercomputer Sites*, `www.top500.org`.
9. 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).
10. Rolf Rabenseifner, *Effective Bandwidth (b_eff) and I/O Bandwidth (b_eff_io) Benchmark*, `www.hlrs.de/mpi/b_eff/` and `www.hlrs.de/mpi/b_eff_io/`.
11. 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, pp 23–32, May 1999. `www.mcs.anl.gov/romio/`.
12. Rajeev Thakur, William Gropp, and Ewing Lusk, *I/O in parallel applications: The weakest link*, in The International Journal of High Performance Computing Applications, Vol. 12, No. 4, Winter 1998, pp. 389–395.
13. Rolf Rabenseifner, *Striped MPI-I/O*, `www.hlrs.de/mpi/mpi_t3e.html#StripedIO`.
14. Rolf Rabenseifner, *Ring Pattern List*, Nov. 1999. `www.hlrs.de/mpi/b_eff/ring_pattern_list` & `www.hlrs.de/mpi/b_eff/ring_numbers.c`
15. Ralf Reussner, Peter Sanders, Lutz Prechelt and Matthias Müller, *SKaMPI: A detailed, accurate MPI benchmark*, in proceedings, 5th European PVM/MPI Users' Group Meeting, LNCS 1497, pages 52-59, 1998. `wwwipd.ira.uka.de/~skampi/`
16. Karl Solchenbach, *Benchmarking the Balance of Parallel Computers*, SPEC Workshop on Benchmarking Parallel and High-Performance Computing Systems, Wuppertal, Germany, Sept. 13, 1999.
17. 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`.
18. TFCC – IEEE `www.ieeetfcc.org`, and Top Clusters `www.TopClusters.org`.