# 64-bit Software Performance Misconceptions

Scott Trent *

There are many excellent reasons to implement 64-bit programs on IBM pSeries platforms, but some commonly accepted performance beliefs are actually misconceptions. This paper describes some of these reasonable sounding beliefs, then proceeds to demonstrate the reality and offers both standard and creative alternatives to achieve the original goal. This paper offers concrete suggestions to assist information system design/development and programming projects (including an unusual technique to allow 32-bit programs to directly utilize significantly more than 4GB of real memory). Familiarity with this material will assist professionals who are involved with implementing projects based on 64-bit pSeries technology.

Key Words & Phrases: 64-bit, performance, scalability, programming, system implementation

## 1. Introduction

There is on-going Customer demand to improve the performance of new and existing information systems while also meeting increasing scalability requirements. [1] The 64-bit application environment provided by the pSeries product line is designed to improve scalability, functionality, and performance while maintaining maximal compatibility with existing applications. [2, 3, 4, 5]

Although there is a strong industry-wide interest in providing and using 64-bit capable products, there are also some misconceptions that professionals involved in implementing or designing projects based on 64-bit pSeries machines should be aware of. An understanding of the reality behind some of these misconceptions will help IT professionals maintain realistic expectations while making correct design decisions.

Along with common sense suggestions, this paper proposes a technique which can be used to allow custom developed 32-bit programs to easily and directly utilize more than the commonly accepted maximum of 4GB of real memory.

## 2. Myth: 64-bit programs are twice as fast as 32-bit programs

There is a common perception, perhaps not unrelated to extensive marketing, that 64-bit environments necessarily perform significantly faster than 32-bit environments. There is some truth to this, since a hardware architecture that transfers data in 64-bit chunks in a fixed period of time has a clear performance advantage over an architecture that can only transfer 32-bit chunks in the same period of time. Likewise theoretical performance advantages can be realized in certain applications which benefit from being able to perform a 64-bit arithmetic operation rather than the equivalent 32-bit operation. Also, a 64-bit program which is able to address massive amounts of in-memory data will perform better than a program which is forced to wait on disk I/O.

### 2.1 64-bit Performance: The Reality

Clearly a 64-bit program running on a recent 64-bit SMP with fast CPU's and a large memory, would have a performance advantage over a similar program compiled in 32-bit mode, running on an obsolete 32-bit system. The primary performance advantages that a 64-bit program has over a 32-bit program are (1) the ability to simultaneously address over 4GB of memory, and (2) the ability to use instructions for calculation and data manipulation of 64-bit quantities. [1, 6, 7]

The ability to simultaneously address over 4GB of memory in a 64-bit process is largely a programmer convenience, since there are techniques (described later in this paper) which 32-bit processes can use to take advantage of large memory systems. However, "programmer convenience" can not be discounted, since program simplification constructs result in easier to understand, and maintain programs, which contain fewer errors.

Depending on which operations and function calls are used, one can observe varying positive and negative performance differences between the same code running in 32-bit and 64-bit mode. Performance testing on multiple pSeries machines demonstrates that, using a 32-bit AIX 5.1 kernel, the performance change between many operations running on 32-bit mode compared with 64-bit mode is less than around 10%. (See Figure 1.) Notable

Submission Date: August 29, 2003

* trent@jp.ibm.com, Server Systems Department

exceptions include the following:

- 8-byte multiplication saw a 50% reduction in time when run in 64-bit mode.
- strcpy() had a 30% performance hit in 64-bit mode.
- Simple system calls such as getpid() had a 40% performance hit in 64-bit mode.
- Function call overhead saw a 30% performance hit in 64-bit mode.

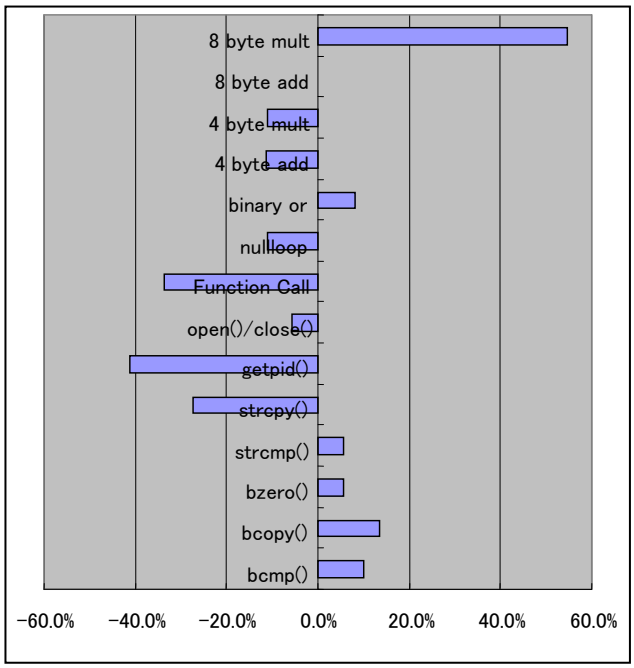


Figure 1. Percentage performance difference measured between 32-bit and 64-bit mode. (AIX 5.1, 32-bit Kernel, multiple pSeries systems)

In addition to whether the program runs in 32-bit mode or 64-bit mode, the type of kernel used will also affect performance. The largest impact observed when changing between a 32-bit and 64-bit AIX 5L kernel was on simple system calls such as getpid(). Although a 32-bit program will have the same performance with respect to getpid() on either a 32-bit or a 64-bit kernel, a 64-bit program running on a 64-bit kernel will run getpid() over 50% faster than a 32-bit program on the same kernel. (See Figure 3.)

The program used to measure the time required to execute the getpid() system call is shown below as "Program 1". This program was compiled once as a 32-bit executable, and once as a 64-bit executable, and then ran under 32-bit and 64-bit kernels on various 64-bit capable pSeries machines, keeping other variables constant. Other operations and API's were measured in a similar fashion by placing the code to be measured between read_real_time() function calls. (Sections of this code are documented in the *AIX Base Operating System and Extensions Technical Reference* for the time_base_to_time() function.)

```
main()
{    timebasestruct_t start, finish;
     int i, seconds, nanoseconds;

     read_real_time(&start, TIMEBASE_SZ);

/* START CODE MEASUREMENT */
     for (i=0;i<500000;i++)
          getpid();
/* FINISH CODE MEASUREMENT */

     read_real_time(&finish, TIMEBASE_SZ);

     time_base_to_time(&start, TIMEBASE_SZ);
     time_base_to_time(&finish, TIMEBASE_SZ);

     seconds = finish.tb_high - start.tb_high;
     nanoseconds = finish.tb_low - start.tb_low;

     if (nanoseconds < 0)    {
          seconds--;
          nanoseconds += 1000000000;
          }

     printf("%d.%d seconds¥n", seconds, nanoseconds);
}
```

Program 1: Sample program used to measure time to execute getpid() system call.

### 2.1.1 64-bit Performance: Real Life Analysis

It could be argued that programs which call getpid() or other functions in a tight loop are not representative of real life programming. To see the affect that compiler mode and selected kernel has on a real life program, I recompiled the AIX 5.1 source code for the grep command as both a 32-bit executable and a 64-bit executable. I then tested each of these programs on a large file using the AIX 5.1 32-bit kernel, and retested it using the same machine and file with the AIX 5.1 64-bit kernel. Although the 64-bit grep running on the 32-bit kernel ran about 1% slower than the 32-bit grep on the 32-bit kernel, both grep programs ran several percent faster on the 64-bit kernel, with the best performer being the 64-bit grep running on a 64-bit kernel which ran 3.4 percent faster than the 32-bit grep running on the 32-bit kernel. The performance improvements observed in both the 32-bit and 64-bit executables when running on a 64-bit kernel lends credence to claims that a 64-bit kernel can efficiently transport data internally as 64-bit chunks

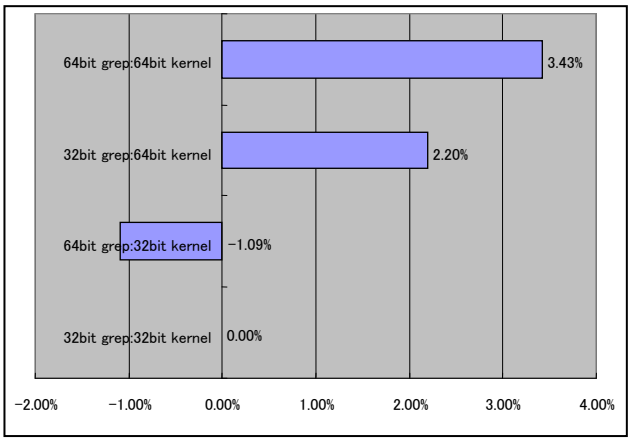


Figure 2: Performance difference as a percentage between 32-bit grep and 64-bit grep running on AIX 5.1 32-bit and 64-bit kernels.

## 2.2 Alternative Performance Improvement

Performance improvement is a highly involved topic with many potential solutions depending on the exact workload and application. Hardware oriented contributors to improved performance include faster clock speed, larger L1/L2 caches, more physical memory, more CPU's, more and faster disk drives, more and faster adapters, etc. When developing applications, the utilization of correct algorithms and efficient programming practices is fundamental to realizing good performance. Finally, an improperly tuned system may realize performance improvements through system parameter modification. [1, 8]

# 3. Myth: A 64-bit kernel is required to run 64-bit programs

Since it is true that 64-bit capable hardware is required to run 64-bit programs, it seems logical to assume that since AIX 5L offers a 64-bit kernel, that it too would be required to run 64-bit programs. Additionally, an impression that 32 and 64-bit architectures are significantly different, may lead to the assumption that all components (hardware, kernel, applications, etc.) must use the same mode whether that be 32-bit or 64-bit.

## 3.1 32-bit Compatibility: The Reality

An understanding of the history of AIX 64-bit development, and current 64-bit environment will easily resolve this misconception. When IBM-Austin provided initial support for 64-bit programs in AIX 4.3, the only kernel provided ran in 32-bit mode. This kernel provides simultaneous support for both 32-bit and 64-bit programs. [6] When a 64-bit program passes an 8-byte pointer to the kernel in a system call, the pointer is mapped into an equivalent datatype by libc.a which the 64-bit kernel can then use to access the referenced memory. [7] (This process is also referred to as "shaping".)

Starting in AIX 5L, a system administrator can choose to use either a 32-bit or a 64-bit kernel on 64-bit hardware. Binary compatibility is a key design point in AIX, so both AIX 5L kernels were designed to be able to support both 32-bit and 64-bit programs, much in the same way that the 32-bit 4.3.3 kernel could run both 32 and 64-bit programs. [9] Not only does this enable the continued use of existing Customer applications with either kernel, but this simplifies AIX itself since the majority of AIX commands themselves are compiled as 32-bit programs, and will remain such for the foreseeable future.

It is an interesting but logical twist that 32-bit programs have their pointers mapped (shaped) when using system calls on a 64-bit kernel. All other things the same, one can expect a slight performance advantage to running 32-bit programs on 32-bit kernels, and 64-bit programs on 64-bit kernels. In fact this was observed in the results of Program 1 which issues the system call getpid() in a tight loop. The key results of this test, as shown in Figure 3, show that there is a performance penalty for 64-bit programs using system calls on a 32-bit kernel, and that 64-bit programs running on a 64-bit kernel have a performance advantage over all other combinations. (These results also follow the same pattern as observed for the grep program in Figure 2.)

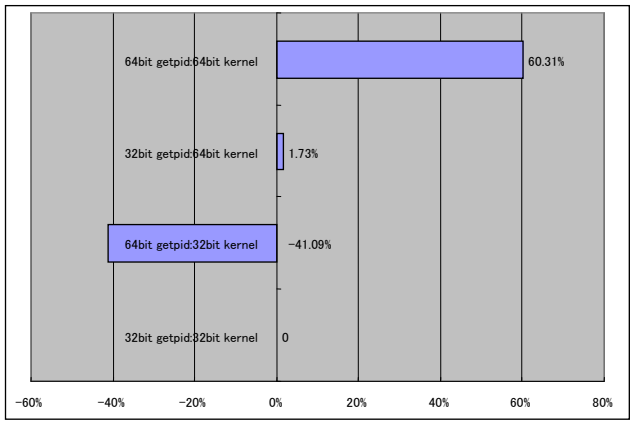


Figure 3: Performance difference as a percentage between 32-bit getpid() and 64-bit getpid() running on AIX 5.1 32-bit and 64-bit kernels.

4. Myth: Only 64-bit programs benefit from 64-bit hardware.

If one assumes that the only benefit of 64-bit capable hardware is the ability to run 64-bit applications, then it would seem logical to conclude that 32-bit applications have nothing to gain from running on 64-bit hardware.

4.1 32-bit Programs on 64-bit Hardware: The Reality

In addition to adding 64-bit capability, IBM has provided enhancements such as increased clock frequency, Silicon on Insulator, copper technology, increased on-chip cache and memory, increasingly parallel instruction decoding, improved speculative execution, etc., to the 64-bit capable CPU's used in the pSeries product line. [2, 3, 4, 5] These offer performance improvements to applications running in both 32-bit and 64-bit mode. Besides these enhancements, newer systems also support larger amounts of physical memory, more scalable SMP's, and tend to have high performance adapters, peripherals, etc. which can also benefit 32-bit applications.

5. Myth: Only 64-bit programs can benefit from more than several gigabytes of physical memory

Since a 32-bit program can only address $2^{32}$ or 4GB of memory it seems reasonable to assume that only 64-bit programs can directly use over 4GB of memory.

5.1 Benefits of Large Memory: The Reality

Regardless of how much memory a single 32-bit process can address, it is obvious that all processes on a machine stand to benefit from reduced memory contention and increased file cache utilization which result from increased physical memory.

Additionally, although a 32-bit process can not simultaneously address more than 4GB of memory, it is possible for a 32-bit process to use shared memory mechanisms in order to serially access all available virtual memory on a system. Conceptually, a 32-bit kernel uses a similar technique when it modifies segment registers in order to access the full range of physical memory on an AIX system, which can exceed 4GB even on a 32-bit machine.

5.2 Large Scale Memory Access from 32-bit Programs: The Details

A 32-bit program can use shmget() to request multiple shared memory segments. The total size of all requested segments for a single 32-bit process CAN exceed 4GB, however, the total number of shared memory segments on a single AIX 4.3.3 system can not exceed 131072. Additionally, there are various limitations on how many of these can be used simultaneously by the same process. Clearly, a 32-bit process can not exceed the 4GB addressability limitation, and there is a limited amount of memory in the 32-bit process space which is available for shared memory usage. For example, an AIX 4.3.3 32-bit process has a maximum of 2.75 GB of space available for use as shared memory. (For detailed information please refer to the AIX documentation for the shmget() function.)

Once a program has created multiple shared memory segments, it can use shmat() and shmdt() to attach to and detach from these segments. (A shared memory segment must be "attached" before it can be used by a process. Segments can be freely attached and detached without significant performance impacts.) The primary restriction is that more than 2.75 GB of shared memory can not be simultaneously addressed from the same process. There are also limitations on the number of shared memory segments that can be simultaneously accessed (i.e., attached) from a 32-bit process. (Once again, the shmget() documentation describes these in detail.)

Within these limitations it is possible to write a 32-bit program which uses a massive amount of memory in a serial fashion. The sample code in Program 1 creates thirty-two 1 GB sized shared memory segments, then precedes to write to and read from each segment in serial. This 32-bit program can perform random-access attaches and detaches to each segment, and thus has the ability to effectively use 32 GB of memory! With virtual memory, a full 32 GB of physical memory is not required. In fact, due to the delayed memory acquisition mechanism, this program can also run on a machine with less than 32GB of paging space. Of course, physical memory or paging space will be required for each page of memory which is actually touched. Another advantage of using shared memory is that these segments are persistent and will not be reclaimed by the system unless the system is rebooted, or these segments are explicitly released. (Note: In the interest of simplicity, this

program as shown below does not perform fundamental error checking which would normally be considered mandatory.)

```
#include <sys/shm.h>

main() {
   int shmid[32];
   int i;
   char *p;

   /* create 32 1GB shared memory segments */
   for (i=0;i<32;i++)
      shmid[i]=shmget(IPC_PRIVATE,
      1024*1024*1024,S_IRUSR|S_IWUSR);

   /* Attach, use, detach memory */
   for (i=0;i<32;i++) {              // segments can be
      p=(char *)shmat(shmid[i],0,0);// attached and
      /* use memory */               // detached at will
      shmdt(p);                     // allowing for full
      }                             // random access!
   /* repeat attach, use, detach loop */
   for (i=0;i<32;i++) {
      p=shmat(shmid[i],0,0);
      /* use memory again: it is persistent until released */
      shmdt(p);
      }

   /* release memory */
   for (i=0;i<32;i++)
      shmctl(shmid[i],IPC_RMID,0);
}
```

Program 2: Using 32GB of memory in a 32-bit process

Similar results can be accomplished with the use of mmap() and memory mapped files.

## 6. Myth: There is no benefit to running 64-bit programs on small memory systems

Since a primary advantage of using 64-bit programs is the ability to address more than 4GB of memory simultaneously, there may seem to be no advantage to running 64-bit programs on a system with less than 4GB of physical memory.

### 6.1 Small Memory 64-bit Systems: The Reality

#### 6.1.1 Scalability

In general, the scalability offered by larger kernel tables in the 64-bit kernel can benefit all sized systems. [9] Also, the increased number of segments available to a 64-bit program enable the use of more than 11 shared memory segments or mmap()ed memory areas without resorting to the sometimes problemantic EXTSHM functionality.

#### 6.1.2 Large Virtual Address Space

Although a 64-bit program can theoretically address $2^{64}$ bytes of memory, it is sometimes easy to forget that the memory that a 64-bit process can use is not limited by physical memory, but rather by virtual memory. Thus, a system with a relatively small physical memory and a significantly larger paging space can support 64-bit programs which can use all of the paging space as virtual memory. Under common programming situations, this could lead to memory over-commitment which would cause severe performance problems, but there are classes of programming problems which could benefit from a large sparsely used address space. For example, a program could allocate a 10 GB table, and then use a simple, easy to maintain indexing scheme which only uses a small fraction of the table. The entire table would be addressable, but only that which is currently used would consume system resources.

#### 6.1.3 Persistent Memory

The ability of a 64-bit program to use a large address space makes it an ideal tool to implement or use large scale persistent memory systems.

#### 6.1.4 64-bit Data Manipulation and Calculation

As demonstrated by the data in Section 2 of this paper, although doubled performance is not observed for most 64-bit operations, performance improvements can be realized by using the 64-bit mode.

## 7. In Conclusion

Although the pSeries product line provides an excellent 64-bit environment, it is important to have realistic performance expectations regarding 64-bit technology. 64-bit technology is one of the tools available to IT professionals to maximize performance along with scalable hardware/Operating System, algorithms, shared memory, and tuning. As a key scalability enabler, 64-bit technology provides both direct and indirect benefits to applications, which an IT professional must be aware of when participating in information system projects.

## Bibliography

[1] Hoetzel, Fernandez, Koh, Marshall, Martin, *AIX*

*64-bit Performance in Focus*, IBM, SG24-5103-00, 1998.
[2] Papermaster, Dinkjian, Mayfield, Lenk, Ciarfella, O'Connell, DuPont, "POWER3: Next Generation 64-bit PowerPC Processor Design", www.ibm.com/servers/eserver/pseries/library/wp_systems.html, 1998.
[3] Borkenhagen, Storino, "4th Generation 64-bit PowerPC-Compatible Commercial Processor Design", www.ibm.com/servers/eserver/pseries/library/wp_systems.html, 1999.
[4] Borkenhagen, Storino, "5th Generation 64-bit PowerPC-Compatible Commercial Processor Design", www.ibm.com/servers/eserver/pseries/library/wp_systems.html, 1999.
[5] Tendler, Dodson, Fields, Le, Sinharoy, "IBM eServer POWER4 System Microarchitecture", www.ibm.com/servers/eserver/pseries/library/wp_systems.html, 2001.
[6] Cannon, Jones, Trent, *Simply AIX 4.3, Edition 2*, Prentice Hall, ISBN 0130213446, 1999. (pp 215-217)
[7] Vetter, Baba, Iacopetta, Vagnini, *AIX Version 4.3 Differences Guide*, IBM, SG24-2014-02, 1999.
[8] Trent, Monterey/64: Technical Introduction and Software Porting presentation materials, ISE Technical Conference, 2000.
[9] Akeret, Hollanders, Lane, Peterson, *AIX 5L Differences Guide Version 5.1 Edition*, IBM, SG24-5765-01, 2001.