Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 18 (2013) 2233 – 2240

Procedia Computer Science

International Conference on Computational Science, ICCS 2013

Evaluation of x32-ABI in the Context of LHC Applications

Nathalie Rauschmayr[a], Achim Streit[b]

[a]CERN - European Organization for Nuclear Research, Switzerland
[b]Steinbuch Centre for Computing, Karlsruhe Institute of Technology, Germany

Abstract

The application binary interface x32 (x32-ABI) was introduced in Linux kernel 3.4 and is based on the x64 instruction set. Instead of 64-bit it uses 32-bit as size for pointers and C-data type long, reducing consequently memory overhead. As several LHC applications, especially reconstruction and analysis software, suffer memory problems, since the change from 32- to 64-bit, this binary interface has been evaluated. This paper shows that in most of the applications, used at CERN, the memory overhead can be reduced between 3% and 35% and the corresponding difference in CPU-time can be improved up to 30%.

Keywords:
Application Binary Interface x32, HEPSPEC-benchmarks, ROOT data analysis framework, LHCb

1. Introduction

Many applications used in the LHC experiments suffer from memory problems due to an increased memory usage. The upgrade from 32-bit to 64-bit marks an inflection point for LHC applications, leading to an even more noticeable memory overhead, especially on reconstruction and analysis software [1]. This is caused by the fact that LHC software uses millions of pointers [1]. A reconstruction job in the LHCb experiment [2] processes 50000 events on average, whereat each event is stored several times as a pointer inside the application. Furthermore, the framework used in LHCb [1] consists of many virtual functions, leading to virtual tables with many hidden and function pointers. A reconstruction job produces several histograms, generated via the framework ROOT [3], which internally stores them as a tree of pointers. In total this increases the memory footprint of the reconstruction software by a factor of 1.6 during the application main loop. As a result, the footprint of reconstruction jobs is larger than 1 GB and 2 GB for analysis jobs. As the LHC Computing Grid [4] provides on average about 2 GB per core, many jobs run out of memory. The application binary interface x32 gives the chance to combine the advantages of 32-bit and 64-bit applications. This translates into lower memory footprint and also performance improvement due to faster system calls. This paper evaluates the impact of x32-ABI in the context of LHC applications: the reconstruction and analysis software of the LHCb experiment, several ROOT benchmarks and HEPSPEC-benchmarks. It compares the results of x32-ABI with 32-bit and 64-bit binaries in terms of memory consumption and CPU-time with explanations for performance differences. An introduction of x32-ABI is given in section 2, results regarding memory footprint and CPU-time are presented in section 3, an analysis of results is presented in section 4, related work is shown in section 5 and a conclusion is given in section 6.

*Corresponding author. Tel.: +41227675769 .
E-mail address: nathalie.rauschmayr@cern.ch.

doi:10.1016/j.procs.2013.05.394

2. The Application Binary Interface x32

2.1. Technical information

x86 is an architecture which uses a word size of 32-bit and consequently limits the addressable memory to 4 GB. Due to the increased capacity of memory in the last decades, it has been necessary to enlarge the word size in order to overcome this limit. x86 has been extended to x86-64, which supports both 32-bit and 64-bit applications. In addition, the new instruction set facilitates the usage of new hardware features, so that 64-bit applications can profit from an increased performance. As described in detail in [5], the main advantages are: faster system calls, better floating point performance and larger CPU-registers. Function calls are passed directly via registers and not through the stack memory, eliminating consequently move instructions to the stack. Computations based on 64-bit integers can also be performed faster due to 64-bit registers. Returning floating point values via SSE registers contributes as well to better performance, since those values do not have to be loaded into the x87 register, as required in 32-bit architecture. Consequently, assembler code generated for x86-64 is often much smaller. Nevertheless, there are also drawbacks, mainly related to the memory overhead caused by the extended size of data types. As a result, performance can be slowed down due to an increased number of cache misses, page faults, higher memory contention or paging.

The application binary interface x32 is a new interface, which has been developed in order to combine advantages of both architectures. It reduces the size of pointers and C-data type long to 32-bit, but it is based on x64 instruction set. Therefore, x32 applications can profit from new hardware features in the same way. Nevertheless, modifications of 28 out of 300 system calls have been necessary according to [5], as they use pointers in memory.

2.2. Preconditions

Linux kernel 3.4 introduced x32-ABI [6]. However, the interface is switched off by default. Enabling the parameter `CONFIG_X86_X32` and recompiling the kernel allows it to distinguish system calls made by 32-bit, 64-bit and x32 applications. All three types can coexist on the same system. Support for x32 was introduced in glibc 2.16, gcc 4.7 and binutils 2.22 [7][8][9]. The gcc-flag `mx32` allows the creation of x32 applications, which presents a new executable and linkable format (ELF) with the following ELF-header:

```
ELF 32-bit LSB shared object, x86-64, version 1 (SYSV)
```

It indicates that the binary is based on the x86-64 instruction set but uses 32-bit pointers. Since this represents a new ELF-format, x32 applications cannot be linked neither against 32-bit nor 64-bit libraries.

3. Evaluation of LHC-applications

For the evaluation, the HEPSPEC-suite [10] is chosen in first place, as it represents a good mix of typical High Energy Physics (HEP) workloads. Apart from this, applications used by the LHCb experiment are evaluated. More specifically reconstruction and analysis software, since these often suffer memory problems. In a last step, the data analysis framework ROOT is tested. It is the core data analysis package, used by all LHC-experiments for histogramming, IO, mathematical functions and networking.

3.1. Test environment

An Intel Core i7 with 3.4 GHz providing 4 physical and 8 logical cores is used. The test environment consists of a main memory with 8 GB, a L3 cache with 8 MB, a L2 cache with 256 KB and two L1-Caches with 32 KB each. A gentoo distribution is used as operating system supplying a x32-release as described in [11]. It provides all necessary system libraries in x32-ELF-format and enables this interface by default.

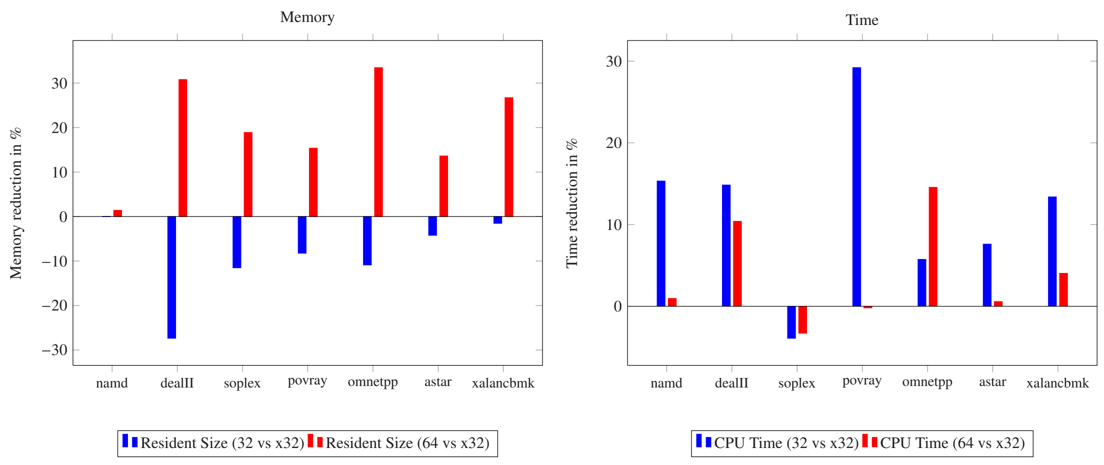

Fig. 1. Comparison of memory consumption and CPU-time within the HEPSPEC-benchmarks

3.2. Caveats

The system memory allocator can behave differently on a 64-bit platform than on a system, which provides only 4 GB address space. In the first case, it assumes infinite memory space, as the memory limit is 2^{64} Byte (16384 Petabyte). Under certain circumstances, memory is not given back to the system for performance reasons. This is a consequence of how the function *malloc* of glibc is implemented. It allocates memory either via the functions *mmap* or *sbrk* depending on a certain threshold. According to [12], the default threshold is $4 \cdot 1024 \cdot 1024 \cdot sizeof(long)$ on 64-bit and $512 \cdot 1024$ on 32-bit systems. The *mmap*-calls have the advantage that memory can be independently given back to the system. If a process releases a *sbrk*-allocated memory block, it is not returned to the operating system. This might lead to differences in memory profiles seen by the system and process, as is shown in section 3.5. As a consequence, additional tests are necessary in order to verify that reduction in memory are caused by x32-ABI and not due to different thresholds. For the evaluation, memory profiles are generated by counting memory maps of processes and heap profiling. The first one shows the size of the heap, stack and loaded libraries and it indicates thus the total consumption of a process, seen by the system. Heap profiling shows memory consumption seen by the process, and it counts objects, which are allocated via malloc, realloc, new and released via free, delete. This is done with a preloaded library, which intercepts those calls and records the size of allocated and released objects. The results are compared to the size of the heap measured with memory maps. Small differences can occur, because *mmap*-calls, directly used from inside an application, do not allow an interception. Those calls are neglected in the heap profiling. Large discrepancies between both results indicate that the memory allocator has worked differently.

3.3. HEPSPEC-Benchmarks

HEPSPEC is a de facto benchmark suite in High Energy Physics for measuring CPU performance. It is based on the SPEC-benchmark test suite [13]. It scales with the performance of HEP-software [10] and thus is representative. Consequently, performance gains, seen at the level of HEPSPEC-benchmarks, will certainly apply at the level of LHC-software. The test suite consists of test cases for XML-processing (*xalancbmk*), path finding algorithms (*astar*), discrete event simulation (*omnetpp*), numerical solver (*soplex*, *namd*, *dealII*) and visualization algorithms (*povray*). As those benchmarks support also a 32-bit recompilation, x32 is compared to the 64-bit as well as the 32-bit version.

Left diagram of figure 1 shows the reduction in memory, in which reduction means $\frac{memory_{64/32} - memory_{x32}}{memory_{64/32}}$. As indicated by the blue columns x32 applications do not consume less memory than the equivalent 32-bit tests. Compared to the 64-bit version the physical memory consumption (resident size) is reduced by about 15% in *astar*, *soplex* and *povray* and up to 30% in *xalancbmk*, *dealII* and *omnetpp*, as indicated by the red columns.

Right diagram of figure 1 shows the reduction in CPU-time, in which reduction means $\frac{time_{64/32} - time_{x32}}{time_{64/32}}$. Most tests show a performance gain, except for the case *soplex* which looses 4%. Compared to the equivalent 64-bit

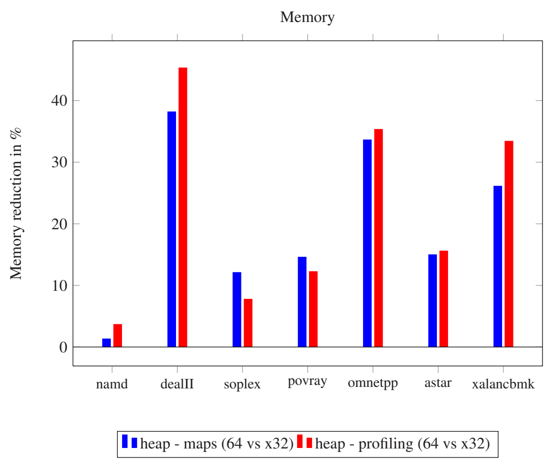

Fig. 2. Comparison of heap-profiles within the HEPSPEC-benchmarks

applications (red columns), significant differences can be observed in *omnetpp* and *dealII*. Between 10% and 15% of CPU-time can be gained. In all other cases, the performance gain is between 0% and 5%. Regarding the 32-bit tests x32 binaries consume by far less CPU-time as indicated by the blue columns in the right diagram of figure 1. The reduction is about 30% for *povray*, 14% for *namd*, *dealII*, *xalancbmk* and between 5% and 7% for *omnetpp* and *astar*.

As mentioned in section 3.2 memory seen by the system and by the process can differ due to diverse thresholds in the function *malloc*. For validation purposes it is consequently necessary to count memory allocation calls inside the application. This is representative for the size of the heap, which is then compared to the size of the heap reported by the operating system. Figure 2 shows the reduction in percentage, which have been measured as the heap size, seen by the operating system (blue columns) and by the process (red columns). As explained in section 3.2 small differences can occur, because not all calls for memory allocation, like for example *mmap*, can be intercepted. However, no significant difference occurs and it allows the assumption that memory seen by the process and system is the same and reduction in memory corresponds consequently to x32-ABI.

3.4. *LHCb-Applications*

Since the HEPSPEC-benchmarks perform better as x32 binaries, apart from the *soplex* case, some commonly used applications of the LHCb experiment are evaluated. The reconstruction and analysis applications of the LHCb software framework [1] are chosen. This requires the recompilation of the complete stack, which consists of 16 external packages excluding additional sub packages, 5 CERN-specific applications including ROOT and all LHCb specific packages, where each one consists of several sub packages.

3.4.1. *Reconstruction*

During the reconstruction, particle tracks are calculated. Reconstruction represents a combinatorial problem whose complexity depends on the number of particles and on the number of detector layers which a particle crossed. The higher the complexity the larger the memory footprint is. As already explained in the introduction, the memory footprint increases by a factor of 1.6 when going from 32- to 64-bit, which means that the reconstruction uses at least 1.1 GB during the main loop instead of 700 MB. After recompiling the application as x32-binary a reduction of about 230 MB for the real physical memory consumption (resident size) can be reached (Fig. 3). This corresponds to a reduction of about 20% compared to the 64-bit version. Reduction of virtual memory is important, since it is limited to 4 GB in the Computing Grid [15]. Left diagram of figure 3 shows that on average a memory saving of about 28% can be reached in the virtual address space (virtual size). The memory profile shown in figure 3 is generated via the memory maps of a process, which presents memory consumption seen by the system. In order to exclude side effects due to the memory allocator, the memory values, recorded inside the application, are also compared. The 64-bit test case consumes 1.9 GB during the main loop while the x32 version

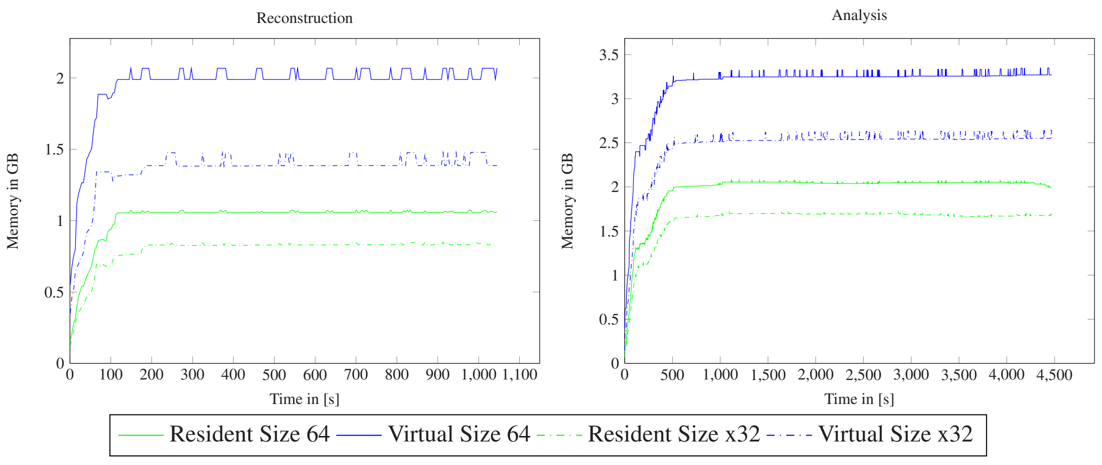

Fig. 3. Comparison of memory consumption within the LHCb-applications

shows only 1.4 GB. This reduction corresponds to the value, reported by the memory maps as shown in figure 3 and concludes, that the memory allocator does not work differently.

The reconstruction test case performs slightly better as x32-binary. The difference is about 2.7% faster in the total time as shown in table 1.

3.4.2. *Analysis*

The analysis software calculates descriptive statistics on the reconstructed datasets and deals with several input and output buffers. As those ones are kept in memory during the main loop, this application is highly memory bounded. The standard stripping case 2012 starts with 2.0 GB of physical memory consumption (resident size) and increases during the main loop. Running this job as x32 binary consumes about 17% less physical and about 21% less virtual memory (Fig. 3). The application addresses only 2.5 GB of virtual memory instead of 3.2 GB and requires 1.65 GB instead of 2.0 GB of resident memory during the main loop. The total CPU time increased slightly by 2.11% processing 1000 events, and 1.59% for processing 10000 events. It concludes that the impact on the CPU-time decreases with the number of events as shown in tabular 1.

Table 1. Results for time measurements

	Reconstruction	Analysis	
Number of Events	1000	1000	10000
Average Total Time (64) in [s]	580.91	554.85	3701.09
Average Total Time (x32) in [s]	565.05	566.59	3760.27
Difference in %	2.73	-2.11	-1.59

3.5. *ROOT-Benchmarks*

ROOT is a data analysis framework, which provides functionalities for plotting functions, writing and compressing files, creating histograms and statistic tools. ROOT provides many different benchmarks for histograms (*stressHistogram*), vector and sparse matrices (*stressLinear*), finding curves, which fit a set of given points (*stress-Fit*), peak search (*stressSpectrum*), testing IO and CPU (*stressEvents*).

In the first three tests an improvement of total CPU time between 2% and 4% is observed (Fig. 4). A slight decline of 1.5% occurs in the case *stressSpectrum*. Wall clock time indicates the time which the CPU spent in kernel and user mode, including the sum of all waiting times and time slices used by other processes. As long as a machine is idle, this time can be used for evaluating the gain in performance in respect to the total time. In most cases, apart from the *stressFit* test, the improvement in wall clock time is less than in CPU time. Large differences indicate that a process spends much time in waiting for resources, like it is the case for the *stressEvents* test.

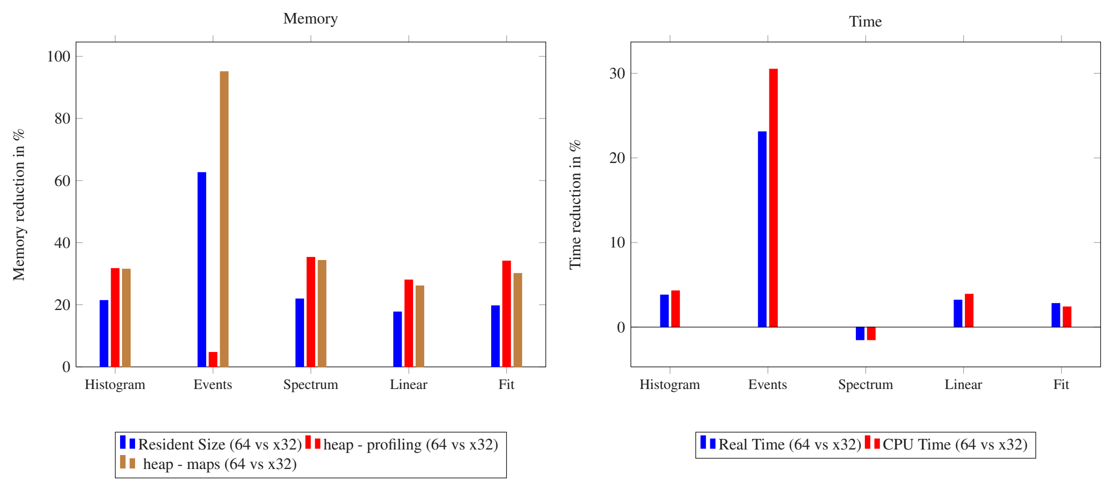

Fig. 4. Comparison of memory consumption and CPU-time within the ROOT-benchmarks

A significant performance difference can be observed in the *stressEvents* case, where an improvement up to 30% is reached. The memory profiles (Fig. 4) show, that the size of the footprints, seen from the system's point of view (blue and brown columns), differ by more than 50%. However, reducing pointers and C-data type long to half of the size leads to an expected maximum reduction of 50%. This conducts to the result that the system memory allocator works differently in this test case. Comparing the values for the heap, seen by the process and the system, validate this assumption. It appears that memory is not given back to the system due to performance reasons, as described in section 3.2. Investigating the memory maps of the x32 test case shows indeed that several memory blocks have been allocated via *mmap*, which has not been the case for the 64-bit version. According to the results of the heap profiling, the actual reduction in memory due to x32-ABI is about 5% for this test case. In all other ROOT test cases the reduction of the physical memory consumption (resident size) is about 20% and memory seen by the process and the system is the same.

4. Reasons for performance differences

Performance differences between 32-bit, 64-bit and x32 applications are caused by several factors. According to [5], x32-ABI currently faces problems with loop unrolling, as it always fully unrolls loops. Loop unrolling is done by the compiler in order to reduce the amount of instructions, which are necessary to control a loop. That increases the size of the binary, but it reduces the number of false branch predictions. These occur, if a branch is predicted, which is then not taken due to a false condition. This leads to a larger number of stall cycles, as the CPU has to remove the false predicted instructions from the instruction pipeline.

Another problem occurs according to [16] in the context of instructions, which deals with pointers and thus requires a zero-extension to 64-bit. According to [16] this is done in the Intel compiler (*icc*) via an address prefix, but the GNU C compiler (*gcc*) does it via an additional system call. The last approach is less efficient and thus influences the overall CPU-time. The amount of instructions can be counted via the performance monitoring unit of a CPU and regarding the test cases it appears that the number of executed instructions does not differ significantly. Another factor, according to [5], is the data alignment, which is influenced by the shorter size of pointers and C-data type long. If a data structure is not aligned, padding bytes are inserted. As an example:

```
struct Example{
   int* pointer;
   bool example;
}
```

As a 64-bit application the data segment will require 8 bytes for the pointer and 1 byte for the bool, which means that it will be extended by 7 padding bytes. As x32 binary the size of the pointer is decreased to 4 bytes, which lead to 3 padding bytes. This can influence 64-bit as well as x32 applications in a negative way. Imagine the above example with 7 additional bool values, would mean that it would be perfectly aligned in 64-bit test case but padding bits are necessary for the case of x32. Consequently, applications which have been optimized for good alignment in 64-bit, will probably lose performance as x32 binary. Furthermore, a CPU can handle short pointers faster, which allows x32 applications to reduce CPU-time compared to 64-bit.

Differences can be evaluated at the level of assembly code and at the level of hardware counter events. Regarding assembly code, each tested application generates thousands of lines of machine code. Hardware counters show how many instructions a CPU has executed and how many stalled cycles, cache misses, page faults etc. have occurred. Those events fluctuate in every program execution. Consequently, it is not reasonable evaluating such events for tests, which show just a small performance difference. However, it is the case in most LHC applications. In the context of the HEPSPEC-benchmarks *omnetpp*, *dealII* and the ROOT test *stressEvents* a performance difference is observed, which is large enough in order to evaluate differences in hardware counter events. In the *omnettpp* case it appears that the number of page faults, is reduced by 38% in the x32 version. A page fault occurs when a process requests a memory page which is mapped into virtual memory but not available in physical memory. An interrupt occurs, which leads to stalled CPU-cycles. The equivalent 64-bit version shows more stalled cycles and an higher total CPU-time. The *dealII* test also shows a significant difference in the number of page faults, which are reduced by about 10%. In the ROOT test (*stressEvents*) the performance difference is certainly caused by the fact that memory is not released.

5. Related Work

As explained in section 2.1, an application can better profit from new hardware features as 64-bit binary. Nevertheless, it does not guarantee that runtime will decrease. Due to the larger size of pointers and instructions, memory footprint increases slowing down the overall performance. This is investigated in [17], where differences between 32-bit and 64-bit SPEC-benchmark applications are analysed. The idea of combining features of 32-bit and 64-bit binaries came up very early and is firstly investigated in [18]. In that paper the same approach was used, as it is currently implemented in the x32-ABI. The size of pointers and of the C-data type long was reduced to 32-bit. Detailed performance measurements of the x32-ABI within the SPEC-benchmark test suite are investigated and analysed in [16].

6. Conclusion

The application binary interface x32 allows a program to reduce memory and CPU-time at the same time. It was originally developed for applications running on embedded systems providing only small memory capacity or which, do not require more than 4 GB. Regarding the LHC Computing Grid virtual memory is in any case limited to 4 GB per process. In one hand, as shown throughout this paper, many HEPSPEC-benchmarks can have a performance improvement up to 30%. On the other hand, most applications, like the reconstruction and analysis software of the LHCb experiment, neither reduce nor increase much the run time. Considering, that the x32-ABI is still under development, further improvements will follow. These will have an impact on the performance of x32 binaries. In the context of LHC applications, memory presents the main limitation for most of the jobs, executed in the LHC Computing Grid. Within many CERN experiments, a lot of effort is done in order to reduce the overall memory footprint of their applications via parallelization [19]. The application binary interface x32 might be a good extension in this context in order to reach some further improvements in memory consumption.

References

[1] G. Corti, M. Cattaneo, P. Charpentier, M. Frank, P. Koppenburg, P. Mato, F. Ranjard, S. Roiser, I. Belyaev, G. Barrand, Software for the lhcb experiment, Nuclear Science, IEEE Transactions on 53 (3) (2006) 1323 – 1328.

[2] O. Schneider, Overview of the lhcb experiment, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 446 (12) (2000) 213 – 221. doi:10.1016/S0168-9002(00)00014-0. URL http://www.sciencedirect.com/science/article/pii/S0168900200000140

[3] R. Brun, F. Rademakers, Root an object oriented data analysis framework, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 389 (12) (1997) 81 – 86, new Computing Techniques in Physics Research V. doi:10.1016/S0168-9002(97)00048-X.
URL `http://www.sciencedirect.com/science/article/pii/S016890029700048X`

[4] Wlcg worldwide lhc computing grid, http://wlcg.web.cern.ch/ (2012).

[5] M. Girkar, H. P. Anvin, H. Lu, D. V. Shkurko, V. Zakharin, Intel Technology Journal Volume 16, the x32 ABI: A New Software Convention for Performance on Intel 64 Processors.

[6] Linux kernel 3.4, http://kernelnewbies.org/Linux_3.4 (2012).

[7] Gcc, the gnu compiler collection, http://gcc.gnu.org/ (2012).

[8] The gnu c library (glibc), http://www.gnu.org/software/libc/ (2012).

[9] Gnu binutils, http://www.gnu.org/software/binutils/ (2012).

[10] Hep-spec06 benchmark, http://w3.hepix.org/benchmarks/doku.php/ (2011).

[11] x32 abi release candidates, http://www.gentoo.org/news/20120608-x32_abi.xml (2012).

[12] Linux programmer's manual, http://man7.org/linux/man-pages/man3/mallopt.3.html (2012).

[13] J. L. Henning, Spec cpu2000: Measuring cpu performance in the new millennium, Computer 33 (7) (2000) 28–35. doi:10.1109/2.869367.
URL `http://dx.doi.org/10.1109/2.869367`

[14] R. Mankel, Pattern recognition and event reconstruction in particle physics experiments, Rept.Prog.Phys. 67 (2004) 553. arXiv:physics/0402039, doi:10.1088/0034-4885/67/4/R03.

[15] Central operations portal, http://operations-portal.egi.eu/vo (2012).

[16] H. P. Anvin, X32 a new 32bit abi for intel architecture, 2011, linux Plumbers Conference.
URL `http://www.linuxplumbersconf.org/2011/ocw/sessions/531`

[17] D. Ye, J. Ray, C. Harle, D. Kaeli, Performance characterization of spec cpu2006 integer benchmarks on x86-64 architecture, in: Workload Characterization, 2006 IEEE International Symposium on, 2006, pp. 120 –127. doi:10.1109/IISWC.2006.302736.

[18] J. Liu, Y. Wu, Performance characterization of the 64-bit x86 architecture from compiler optimizations perspective, in: Proceedings of the 15th international conference on Compiler Construction, CC'06, Springer-Verlag, Berlin, Heidelberg, 2006, pp. 155–169.
URL `http://dx.doi.org/10.1007/11688839_14`

[19] S. Binet, P. Calafiura, S. Snyder, W. Wiedenmann, F. Winklmeier, Harnessing multicores: Strategies and implementations in atlas, Journal of Physics: Conference Series 219 (4) (2010) 042002.
URL `http://stacks.iop.org/1742-6596/219/i=4/a=042002`