SPEC CPUint2006 characterization

Technical Report TR-HPC-01-2009

Rafael Rico, Virginia Escuder

Department of Computer Engineering, Universidad de Alcalá, Spain

April 2009

Index

Abstract

SPEC CPU benchmark suite has become the most frequently used suite for computer architecture research. The workload is designed to stress the hardware of the machines for the next following years. Consequently, both the executed instructions count and the memory map size have been considerably increased in the currently in use release, the SPEC CPU2006, compared to the previous one. But this fact results in prohibitive experimentation time or resources requirements for research using simulation techniques or embedded developing tools.

There is considerable material in the relevant literature discussing about the correct use of SPEC CPU benchmarks and alternatives to avoid problems in the experimentation area. There are some studies undertaking characterizations of the SPEC CPU benchmarks from different points of view, but these are always focused in the reference workload set.

In this technical report, the three sets of workloads supported by SPEC for the CPUint2006 suite, test, train and reference, are analysed in attention to three microarchitecture-independent characteristics: dynamic instruction count, memory usage, and subroutine call distribution. The contents include explanations of why they are selected, how they are used to characterize each benchmark program as well as how much microarchitecture-independent they really are. Results are presented for each individual invocation of the programs, including some remarks and conclusions.

The results from this work will help researchers to find a representative set of workloads for the SPEC CPU int2006 program binaries to use in their experiments whenever they have to discard using the reference workload due to time or resource constraints.

1. SPEC benchmarks

SPEC is the acronym for Standard Performance Evaluation Corporation, a non-profit organization whose purpose is to define and maintain a set of standard benchmarks for computer systems and make them available to the users of such systems as a common reference point in the evaluation of computer performance. It is participated by computer manufacturers, system integrators, consultants, publishers, universities and research organizations [22].

Since its foundation in 1988, the SPEC consortium has developed and distributed technically reliable benchmarks based on real applications. The selection of inputs and workload is performed by the consensus amongst consortium members willing for a transparent, comparable, reproducible and non-proprietary solution, as the organization aim is "*an ounce of honest data is worth a pound of marketing hype*".

SPEC currently offers benchmark suites for the evaluation of different aspects of computation such as performance of CPU, graphics, distributed Java computing, web-servers, and network file systems.

1.1. The SPEC CPU suite

The benchmark suite from the SPEC organization in the field of performance quantification for intensive-computing is the SPEC CPU suite. The SPEC CPU suite aim is to be representative of programming style and application fields of real worldwide computer-intensive workload.

The first delivered set in 1989 had 10 programs and it was known as SPECmark. The most recent generation of the set was announced on August 24, 2006 (SPEC CPU2006) and it is made of 29 programs classified into two groups: one for integer computation (SPEC CPUint2006) and the other for floating point computation (SPEC CPUfp2006). A more complete historical perspective of computer-intensive tests can be found, for instance, in Henning's work [10].

SPEC CPU programs are well known real world applications written in high level, portable, language (C, C++ or FORTRAN) with slight code modifications in order to minimize input/output and thus let the processor, memory and compiler be the factors under evaluation. In fact, it is a requirement that input/output workload is less than 5%. Another requirement is that memory consumption and execution time should be significant for each generation of computers with growing power and capacity. The organization has strict restrictions of evaluation rules affecting code, compilation flags and other aspects of execution environment of the test programs.

The workload sets the amount of processing performed by each benchmark run. The tools distributed by SPEC allow the specification of three different sizes of input data producing different workloads: test, train and reference ("ref" for short). The reference size stands for the reference workload, that is, the input data and command-line options when applicable, used for actual measurements. The test input sets are only used to check that programs compile and execute correctly before launching a real run or to tune up optimization options. Similarly, train inputs are used for profile-based compiler optimizations, so the reference set is the only reportable set.

The SPEC CPU suite has a widespread usage by computer vendors, it is widely accepted by consumers and it is very commonly found too in the academic and research worlds although there is an outstanding debate about how to use it, its convenience and drawbacks, and whether is it or not necessary to design an alternative for research usage, etc. [1, 2].

1.2. SPEC CPU2006 description

The SPEC CPU2006 benchmark suite consists on a set of 12 programs for integers (SPEC CPUint2006) written in C and C++ and 17 programs for floating point (SPEC CPUfp2006) written in C, C++ and FORTRAN. The objective of these computer-intensive programs is to provide portable, credible and real-world application-based benchmarks for quantifying the performance of the set processor, memory and compiler.

In the relevant literature we can find several papers undertaking the description of the suite from different perspectives. A description for all of the programs of the suite SPEC CPU2006 can be found in one of the Henning works [9] and a detailed explanation about the C++ suite programs, in an article written by Wong [26]. Design requirements of the suite SPEC CPU2006 have established the memory consumption top in about 900MB, allowing the suite to run on machines with 1GB of memory. The memory utilization of the programs running under the reference workload is analyzed in Henning [11] and Gove [4]. The paper by Korn and Chang studies how different page sizes affect the performance of the benchmarks [16]. From the I/O perspective, an article from Ye, Ray and Kaeli [29] show that the I/O activity of the SPEC CPU2006 is far below the limit of 5% imposed by the organization. The subroutine call profile of the suite SPEC CPU2006 for the reference workload is presented in Weicker and Henning [25]. The article by Gove and Spracklen provides an interesting evaluation of correspondence between train and reference workloads in SPEC CPU2006 [5]. Other works discuss how to interpret performance counters in the context of SPEC CPU2006 [12], or how the benchmark tools work [23], or which are the main performance differences between x86-32 and x86-64 binaries.

1.3. Concerns about SPEC CPU and alternatives

As computer systems get faster and have more memory because it gets cheaper, the benchmark run times and memory consumption have also increased across generations [10] to ensure that the benchmarks can stress the target systems enough to make meaningful measurements. The SPEC CPU benchmark suite is used by manufacturers to report performance of their systems, by customers to make purchasing decisions, and by designers and researchers to evaluate novel ideas. However, although it is a pretty good tool for the computer market, it significantly increases the cost of performance evaluation as explained by KleinOsowski and Lilja in [15] and Haskins *et al.* in [8] among other authors. So, although this set has become the most frequently used suite for computer architecture research, it can also be detected that experimentation is actually carried using the SPEC CPU benchmarks only partially, and this sub-utilization is not always well justified [1, 2].

The strategy followed by the research community to cope with this situation follows two principal directions. One is to develop techniques to reduce experimentation time, and the other is finding a smaller representative subset of the benchmark programs.

1.3.1. Simulation time reduction

The reduction of input data sets and sampling methods belong to this first approach. They can also be combined: sampling is often used together with reduced input data sets.

The idea of reducing input data sets is to use the same programs from the original suite and decrease the workload submitted by acting upon the arguments the binaries are invoked with. In addition to evaluation time reduction, this approach may also allow to decrease the memory map size. One of the pros of the method is that programs execute completely including its initialization phase, computing phase and cleaning phase [8]. Among the cons is that different input sets could cause the program to exercise different paths [5, 20, 25]. One of the most extended reduced input data sets was the *MinneSPEC* [15]. This reduced workload was developed for the SPEC CPU2000 suite seeking equivalence of results compared to the reference workload.

Sampling reduces workload by using segments of execution only. The selection of segments can be done in a blindly manner by random or uniform collection or in a smart manner by representative analysis based on statistical methods [27, 30]. A big problem to deal with is that machine state at the simulation time starting point is not the same as it is upon real execution time [8].

1.3.2. Subsetting

The second approach to decrease the resources used by the benchmarks for research consists on finding a smaller but representative subset of programs. The method lies in the hypothesis that the suite is redundant. Redundancy in SPEC CPU benchmarks has been predicated since its first release. In the case of SPEC89 suite it was outlined by Saavedra and Smith [21], and by Giladi and Ahituv [3]. For the SPEC CPU95 suite it was reported in several works from Gustafson [7, 6]. Then, Vandierendonck and De Bosschere and Luo *et al.* conclude that the SPEC CPU2000 suite is redundant [17, 24]. Finally, about the SPEC CPU2006, McGhan ensures that the suite application programs are redundant [18] and so do other later publications which apply statistical methods in their analysis reaching the same conclusion [14, 20].

Subsetting is a common technique that has been applied quite often. A high percentage of research works make a partial use of SPEC CPU suite using not very convincing or not well-founded arguments sometimes [1, 2]. The most frequently used technique to find representative suite subsets is the quantification of similarity based on statistical analysis [13, 30]. The problem is that results depend strongly on both the selected characteristics for describing similarity and the statistical way to measure similarity (namely “distance”) [17].

As far as we know, for the SPEC CPU2006 only Phansalkar *et al.* have proposed a representative subset whose justification lies in statistical analysis [19].

2. SPEC CPUint2006 characterization

The matter of this technical report is confined to the integer benchmark suite (SPEC CPUint2006) because this set is far more used for research purposes than the floating point set of the SPEC CPU suite. Justifications of this fact are diverse: floating point code is highly branch predictable and is easier to parallelize, integer programs are more suitable for general propose exploration and so on. Citron in [1] makes a deeper analysis about the convenience of using the integer set.

Programs can be characterized using microarchitecture-dependent characteristics or microarchitecture-independent characteristics. Cycles per instruction (CPI), cache miss-rate, branch prediction accuracy or execution time belong to the first group whereas memory consumption, subroutine call distribution, instruction mix, instruction level parallelism (ILP) or dynamic instruction count usually are classified in the second group. But to be precise, we must make a few points arguing upon the so claimed microarchitecture-independence of several figures from the second group.

- Some of these group characteristics highly rely upon the instruction set architecture (ISA) used to compile the program and obtain the binary code and so, they cannot be considered just a pure quality of the HLL (High-Level Language) programs in the suite. This is the case of instruction mix and dynamic instruction count. Thus, for instance, both of these figures depend on whether the instruction set is RISC or CISC.
- Some others characteristics, such as subroutine call distribution, are influenced by compiler optimizations, operating system, and/or underlying hardware. Thus, the same program flow can derive in different subroutine profiles caused by variations of individual instruction execution time or latency. Also library functions, compiler optimizations, pointers size or the method used to passing arguments, can change the results.
- The memory usage profile can be influenced by many factors affecting memory consumption: compiler optimizations, size of pointers, memory pages allocation policy, page size and so on.
- In the case of ILP, the method used to quantify instruction level parallelism may jeopardize results introducing microarchitecture-dependent information. So for instance, if the method relies upon indirect measures like a ratio between executed instructions vs. cycles consumed, then things like cache misses or number of functional units present in the hardware or simulator configuration may contaminate the figures obtained for the ratio.

Despite the above stated conditions, we have selected dynamic instruction count, memory map size and subroutine call distribution for our work. The reasons are explained next.

Dynamic instruction count, as stated above, depends on ISA. However, if the information we are looking for is the relative computational load among benchmarks, this figure can be considered microarchitecture-independent if obtained using the same hardware. In these conditions, dynamic instruction count reveals the relative length of each benchmarks compared with the rest for a given architecture or, in other words, which are the most CPU time consuming benchmarks. With very few exceptions, the results obtained also stand for other instruction set architectures, as it is a relative figure. One of these exceptions is produced when no 64-bit arithmetic operations are supported directly in hardware and thus a single 64-bit arithmetic operation has to be implemented with multiple 32-bit arithmetic operations. That is the case of *462.libquantum* benchmark as has been reported in [28] which causes a change in the relative proportion kept with the rest of benchmarks for this program.

We have also chosen memory usage profile because it provides a special and particular footprint of each benchmark and workload. Additionally, memory consumption is a specific concern in some areas like embedded system research where low level memory utilization profiles are a must and consequently, it is useful to know how sensitive is the amount of memory consumed by each benchmark to its input data sets.

Finally, subroutine call distribution is an efficient method to observe differences in the behavior of a given program using different input data. This provides for characterization of a given benchmark against the three SPEC workloads and beyond: a profile for each invocation with different inputs and arguments. Again, gathering relative figures is appropriate to extrapolate among architectures where the same differences, generally, still apply.

In the forthcoming sections we review the three selected characteristic for the integer programs of the suite. The dynamic instruction count depends on the target instruction set architecture for which the source code is compiled, as stated above. We have selected the x86-32 instruction set architecture for two reasons: first because it is very extended and second because there is another work that reports dynamic instruction counts for the reference workload using this ISA that we can use for contrasting results [20].

2.1. Executed instruction count in the SPEC CPUint2006

As stated in a previous section the size of the suite has grown along its different editions (SPEC89, SPEC92, SPEC CPU95, SPEC CPU2000 and SPEC CPU2006) both in lines of code and in number of modules.

Table 1 shows x86-32 executed instruction counts (dynamic instruction count) for the three workloads (test, train and reference) of the dynamically linked SPEC CPUint2006 programs. Values presented are from a *Pentium M* (1.5GHz, 1MB L2, 1GB) processor, running the *Linux* (*Fedora Core 6*, kernel 2.6.x) operating system and using the *gcc* compiler (version 3.4.2). They were obtained using the `ptrace` system call. The values for the reference workload are quite similar to the ones reported by Phansalkar *et al*. [20] although they were taken in a *Pentium D* processor running *Linux Suse*.

In general, statically linked executables exhibit a lower number of instruction counts, about less than 1%.

Table 1. Dynamic instruction count for each program and workload in the integer suite SPEC CPUint2006.

	test - billions	train - billions	ref - billions
400.perlbench	0.6	125	2,018
401.bzip2	33.6	186	2,665
403.gcc	5.2	4	1,428
429.mcf	4.8	24	357
445.gobmk	61.4	290	1,814
456.hmmer	18.0	322	3,377
458.sjeng	16.4	521	2,507
462.libquantum	0.5	21	4,033
464.h264ref	99.4	567	4,384
471.omnetpp	2.3	610	745
473.astar	26.7	353	1,446
483.xalancbmk	0.4	337	1,410

For the cases when a test program is executed several times with different data input sets, the instruction count shown is an accumulation for all the runs. However we must keep in mind that every input set may be exercising a different part of the hardware.

The SPEC CPUint2006 executed instruction count for the reference workload are about a few trillions ($\propto 10^{12}$) whereas they were only about a few hundred billions ($\propto 10^{11}$) in the SPEC CPUint2000.

As table 1 shows, there is no uniform increasing order of the number of instructions executed for the test – train – reference sets for every benchmark program. Although in many cases we do observe an increase of about one order of magnitude for each set like for instance in *401.bzip2*, *456.hmmer* or *464.h264ref*, this is not always the case. Frequently the gap between test and train is more than two orders of magnitude like in *400.perlbench*, *483.xakancbmk* or *471.omnetpp* but sometimes there is not much difference or even, like for *403.gcc* it is a bit lower for the train workload. The train – reference relation has fewer exceptions to the one-order-of-magnitude gap *vs.* reference (*403.gcc*, *462.libquantum*, *471.omnetpp*) but still they stand.

The lower figure in the test workload is in the order of billions (10^9) of instructions. Half of the programs perform around this figure and the other half are around one order of magnitude more (10^{10}). The programs under the train workload execute around 10^{11} instructions with few exceptions and under the reference workload totals are beyond 10^{12} executed instructions.

Annex I presents dynamic instruction counts for each program invocation and each workload. The results allow a comparative insight among binaries and its different invocations (see "*Annex I: dynamic instruction count in the SPEC CPUint2006*"; there we show figures instead of values since the comparison of computational load is more important that the actual values). We comment on this information in the final conclusions section.

Obviously, the actual instruction counts may be different for other instruction sets. Ye *et al*. in [28] make a performance comparison between x86-32 and x86-64 instruction sets for the SPEC CPUint2006 suite. For other instruction sets there are no studies published in the relevant literature as far as we know.

2.2. SPEC CPUint2006 memory usage

The committee selecting the programs of the suite established the limit of memory usage having in mind 1GM main memory machines. For the SPEC CPU2000 the limit was 256MB. If we account for some room for the operating system, the maximum memory size (*memory footprint*) used for the suite SPEC CPU2006 is, approximately 900 MB.

For of RSS (*Resident Set Size*) or VSZ (*Virtual Size*), memory utilization profile is divided into two large groups: in some cases it grows rapidly and then remains constant while in others it varies along time. Henning presents this graphically in "*SPEC CPU2006 Memory Footprint*" [11] which, essentially agrees with the graphics ones we obtained (see "*Annex II: memory usage profiles in the SPEC CPUint2006*"). In addition to Henning's work, we offer results of test and train workloads that can slightly change some assumptions reported there (as we comment on the final conclusions section) since we are more interested in the behavior across different workloads than in the evolution of an individual invocation of a binary.

It is also worth mentioning that both analysis, Henning's and ours, were made with dynamically linked executables and it is possible that some smaller memory usage could be registered if statically linked programs were used instead, especially for cases where included library sizes are large enough to impact the amount of text memory used by the benchmark.

We obtained memory usage figures out from the information in the `/proc` pseudodirectory in a machine running *Linux* (*Fedora Core 6*, kernel 2.6.x) whereas in Henning's work they used the `ps` command in a machine running Solaris under a SPARC processor. Annex II shows memory usage profiles graphically for each benchmark invocation within each workload set. The X axis is not time based; instead it presents the complete execution lifetime of the program with invoked with a given input data set.

Although the SPEC organization made the new suite memory size usage to be larger than the previous edition (and that seems to be the case in terms of RSS or VSZ), Gove uses an alternative metric known as WSS (*Working Set Size*) that measures the memory size of the data area really used, thus showing that the actual memory size is almost the same as the SPEC CPU2000 [4]. Only a small percentage of time (less than 5%) the programs use over 256MB. An explanation for this may be that applications from the suite make large memory reservations but then they use only reduced data area as explained in the referred article.

2.3. SPEC CPUint2006 subroutine call distribution

Subroutine profiling is both a helpful performance tool and a precise characterization method. Subroutine profiling is especially useful to find out if different invocations of the same program using different inputs are similar or not. In the relevant literature we find the work of Weicker and Henning reporting the subroutine profile for the reference workload [25]. In this technical report we present the profiling results for each individual invocation of the benchmark program for each of the three workloads, test, train, and, of course, reference.

It is understood that profiling results can vary from an experimental environment to another as they are not completely independent from the hardware and the software on which the program is running as we mentioned above. Even for the same instruction set and the same compiler, execution times for individual instructions may differ among implementations of processors of the same family.

In fact, the experimental conditions used here are different from the work previously mentioned: here the target instruction set is the x86, all binaries are 32 bits and the optimization level is –O2 whereas the previous cited work reports profiling results using the SPARC instruction set and optimization level –O. In any case, we think that these differences can contribute to a much deeper knowledge of binaries and workloads.

To obtain the subroutine call profile in *Linux* we compiled the programs using the `-pg` option of the *gcc* compiler, which causes the insertion of code for collecting information, and then we used the `gprof` command to display the collected data of each benchmark. Annex III contains the subroutine call distribution we obtained (see "*Annex III: subroutine profiles in the SPEC CPUint2006*"). In order to facilitate a possible comparative analysis by the reader with the work of Weicker and Henning, we are presenting information in the same fashion and using the same convention for long names that they used. That is, each table list consists of the 20 highest-scoring subroutines only; routines called less than 1% of the time are not reported and long names from C++ routines have been truncated too.

3. Conclusions

Several conclusions can be derived concerning dynamic instruction counts. The first observation is that some programs from the SPEC CPUint2006 suite can produce extremely different dynamic counts depending on the input data sets whereas others are more regular in this aspect. Thus we could say that some of them are more *elastic* that others in computational terms. The fact is that it takes several invocations of some programs to reach a similar computational work that others get with just a single invocation. Among the elastic binaries, ranging up to four orders of magnitude in the number of executed instructions with a single invocation are *458.sjeng*, *462.libquantum*, *471.omnetpp*, and *483.xalancbmk*. On the other extreme for non-elastic binaries we have *400.perlbench*, *403.gcc*, *429.mcf*, and *445.gobmk*.

A final statement on instruction dynamic counts is that, comparing the figures obtained for the three sets test, train, and reference workloads, we observe that the values obtained for each single invocation of a program are more regular in the reference workload set than they are within the test and train workloads.

We may also state some conclusions concerning memory usage. For the reference workload, the results we present are virtually identical to the ones reported in Henning's article [11] although we go into more detail providing a graphic per-invocation of each benchmark while the named work consolidates all the invocations in a single graph. The only exception is program *456.hmmer* for which we measured considerably lower numbers of memory usage for both invocations, although it is not excessively surprising because it is well known that there are many factors affecting memory consumption (operating system, compiler, etc.).

In the graphic samples for the test and train workloads we gather more information of the benchmarks for different input data sets than in [11]. We can see, for example that *429.mcf*, considered stable together with *458.sjeng* in [11] for the reference workload, actually has a variation over time in its memory requirements, as it becomes apparent for the test and train workloads; also, it reaches different top values which lead us to consider it as a non-stable benchmark (in memory usage), sensible to input data set changes.

On the other hand, if we look at all the different graphs for *445.gobmk* we can see how it uses practically the same amount of memory for every invocation in every workload, reached short after it begins executing. So, if there is room for some flexibility, we may consider that *445.gobmk* is also stable in addition to *458.sjeng* across different workloads.

As for comparative maximums of memory consumed among workloads test, train and reference, if we exclude *458.sjeng* and *445.gobmk* which remain constant independently of the input data set, as we explained above, every benchmark program follows the tendency of increasing memory consumption, that is: every program uses more memory in reference than it does in train; and more in train than it does in test.

About lowest tops, all benchmark but *429.mcf* and *448.sjeng* have several invocations using less than 30MB in the test workload. The same stands for the train workload with additional exceptions of *471.omnetpp* reaching close to the 45 MB top and *483.xalancbmk* that goes beyond 120MB, which is unusual because only in the reference workload set benchmarks often use more than 100MB.

If we look at the shapes of the memory profile graphs, we find that there is a large variety. But what we are most interested is on learning whether or not this shape changes for different invocations of the same program. In this sense, some maintain the memory profile shape for all invocations: *401.bzip2*, *445.gobmk*, *456.hmmer*, *458.sjeng*, *462.libquatum*, *464.h264ref*, *473.astar*, *483.xalancbmk*. Programs *429.mcf* and *471.omnetpp* maintain the memory profile shape in all invocations of test and train workloads only. In contrast, programs *400.perlbench* and *403.gcc* show different shapes per invocation, indicating possible different program flows.

Finally, analyzing subroutine call distributions leads us to the following conclusions: Programs *400.perlbench* and *403.gcc* exhibit quite different subroutine call distributions for each invocation, which means that the program follows different execution flow paths depending on the inputs and explains the observations made above about memory usage.

Program *456.hmmer* has a peaky profile as it uses just one subroutine around 95% of time except in the test workload where it uses it close to 70% of time. Program *429.mcf* shows the same profile for test and train workloads whereas it exercises other routines for the case of the reference workload, which provides an explanation for the memory behavior of this program. Programs *462.libquantum*, *464.h264ref*, and *473.astar* exhibit the same subroutine call profile for every invocation and workload always consuming 80%, 55%, and around 70% of total execution time respectively. The rest seven programs do show more differences in the profile, calling approximately the same set of subroutines but having different usage time depending on the actual invocation.

Consequently, we can say that the control flow of benchmark programs is typically rather independent of workload, although we can find some exceptions where different input data sets cause the program to exercise different control paths, and that the subroutine distribution may vary considerably between invocations. For this reason we should be quite cautious when looking for workloads representatives of the reference workload as Gove and Spracklen recommend [5].

4. Some recommendations for selecting a representative workload

In this section we make a few considerations to those looking for alternatives for applying the reference workload of the SPEC CPUint2006 suite in their experimentation fields because of its excessive weight, in terms of dynamic instruction count, memory size or both. Despite using strategies as those named in sect. 1.3 to find representatives, we then propose other alternatives for two types of scenario: execution time constrained and memory size constrained environments.

4.1. Execution time constrained scenario

Fig. 11 in Annex I shows that even for the lightest workload (the test workload), the amount of executed instructions is far too high for experimentation. It would be most convenient to get it down by around two orders of magnitude, that is, reaching dynamic counts of approximately 100 million (10^8) executed instructions.

For the so called elastic binaries, the suggestion is to look for a reduced input data set that achieves the required reduced dynamic count. That requires to study the behavior of the execution time as a function of input arguments together with the subroutine call distribution in order to determine a correct representativeness.

In the case of non-elastic binaries, things depend on the typical computational load per invocation. For instance, program *400.perlbench* under the test workload is called several times with different inputs and each invocation produces a very low dynamic count each. Something similar happens with *445.gobmk* although it shows higher computational load (about one order of magnitude higher than *400.perlbench)*. In both cases, any one input data set from the test workload can be chosen to decrease the executed instruction count, although using different argumentation to judge its representativeness. For *400.perlbench*, the subroutine call distribution suggests that every invocation exercises different execution flow paths and then it is reasonable to say that selecting one invocation or another would be as much adequate as incomplete at the same time. In contrast, for each invocation of *445.gobmk* the program exercises similar execution flow paths, thus it can be considered a more stable choice.

The program for which we found most difficult to reduce execution time is *401.bzip2* because it has a programmed fixed minimum size of 1 Mbyte buffer of data to compress and always executes three cycles of compressing-uncompressing. This is performed before the actual invocation to the compressing/decompressing logic, precisely to increase computation load. In this case, to reach a dynamic count of around 10^8 executed instructions it is necessary to modify this pre-algorithmic source code.

4.2. Memory size constrained scenario

As it has been stated before, all benchmarks but *429.mcf* and *448.sjeng* have several invocations among test and train workloads that use less than 30MB. Some of them could be used for experimentation when memory size constrains have to be observed. The representativeness of those invocations based on its subroutine call distribution again depends on each binary.

In the case of *400.perlbench* and *403.gcc* they exhibit quite different subroutine call profiles for each invocation, thus concluding that any one invocation can be considered adequate and incomplete at the same time.

For *445.gobmk*, from a memory usage profile point of view, every invocation is similar since it consumes the same amount of memory. The same could be said for *448.sjeng* although it has a prohibitive amount of reserved memory that can not be decreased through input data set adaptation.

In general, a reduced input data set generating lower dynamic instruction counts may also produce a lower memory consumption profile in the case of many benchmark programs, it is a statement worth to study more carefully: a combined effect is not discardable.

Another recommendation is to use benchmark programs statically linked (instead of dynamically linked) because they generally produce a lower memory usage profile, as well as a few less executed instructions.

References

[1] D. Citron. MisSPECulation: partial and misleading use of spec CPU2000 in computer architecture conferences, in *Proceedings of the 30th Annual International Symposium on Computer Architecture*, pp. 52-59, June 9-11, 2003.

[2] D. Citron, J. Hennessy, D. Patterson, and G. Sohi. The use and abuse of SPEC: An ISCA panel, *IEEE Micro*, Vol. 23, No. 4, pp. 73-77, July/August 2003.

[3] R. Giladi and N. Ahituv. SPEC as a Performance Evaluation Measure, *IEEE Computer*, Vol. 28, No. 8, pp. 33-42, August 1995.

[4] D. Gove. CPU2006 Working Set Size. *Computer Architecture News*, Vol. 35, No. 1, pp. 90-96, March 2007.

[5] D. Gove and L. Spracklen. Evaluating the correspondence between training and reference workloads in SPEC CPU2006, *Computer Architecture News*, Vol. 35, No. 1, pp. 122-129, March 2007.

[6] J. L. Gustafson and Q. O. Snell. HINT: A new way to measure computer performance, in *Proceedings of the 28th Annual Hawaii International Conference on System Sciences*, Vol. 2: Software Technology, IEEE Computer Society Press, Editors: H. El-Rewini and B. D. Shriver, pp. 392-401, January 1995.

[7] J. L. Gustafson and R. Todi. Conventional Benchmarks as a Sample of the Performance Spectrum, in *Performance Evaluation and Benchmarking with Realistic Applications*, MIT Press, Editor: R. Eigenmann, p. 304. January 2001.

[8] J. Haskins, K. Skadron, A. KleinOsowski, and D. J. Lilja. Techniques for Accurate, Accelerated Processor Simulation: Analysis of Reduced Inputs and Sampling, *Technical Report CS-2002-01*, University of Virginia*,* 2002.

[9] J. L. Henning. SPEC CPU2006 benchmark descriptions, *Computer Architecture News*, Vol. 34, No. 4, pp. 1-17, September 2006.

[10] J. L. Henning. SPEC CPU Suite Growth: An Historical Perspective, *Computer Architecture News*, Vol. 35, No. 1, pp. 65-68, March 2007.

[11] J. L. Henning. SPEC CPU2006 Memory Footprint, *Computer Architecture News*, Vol. 35, No. 1, pp. 84-89, March 2007.

[12] J. L. Henning. Performance counters and development of SPEC CPU2006, *Computer Architecture News*, Vol. 35, No. 1, pp. 118-121, March 2007.

[13] R. Jain. The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling, Wiley-Interscience, 1991.

[14] A. Joshi, A. Phansalkar, L. Eeckhout, and L. K. John. Measuring Benchmark Similarity Using Inherent Program Characteristic". *IEEE Transactions on Computers*. Vol. 55, No. 6, June 2006.

[15] A. J. KleinOsowski and D. J. Lilja. MinneSPEC: A New SPEC Benchmark Workload for Simulation-Based Computer Architecture Research, *Computer Architecture Letters*, Vol. 1, No. 1, pp. 7-10, January 2002.

[16] W. Korn and M. S. Chang. SPEC CPU2006 sensitivity to memory page sizes, *Computer Architecture News*, Vol. 35, No. 1, pp. 97-101, March 2007.

[17] Y. Luo, A. Joshi, A. Phansalkar, L. John, and J. Ghosh. Analyzing and improving clustering based sampling for microprocessor simulation, in *Proceedings of the 17th International Symposium on Computer Architecture and High Performance Computing*, pp. 193-200, October 2005.

[18] H. McGhan. SPEC CPU2006 Benchmark Suite, *Microprocessor Report*, October 2006.

[19] A. Phansalkar, A. Joshi and L. K. John. Subsetting the SPEC CPU2006 benchmark suite, Computer Architecture News, Vol. 35, No. 1, pp. 69-76, March 2007.

[20] A. Phansalkar, A. Joshi, and L. K. John. Analysis of Redundancy and Application Balance in the SPEC CPU2006 Benchmark Suite. In *Proceedings of the International Symposium on Computer Architecture (ISCA'07)*, 2007.

[21] R. H. Saavedra and A. J. Smith. Analysis of Benchmark Characteristics and Benchmark Performance Prediction, *Technical Report USC-CS-92-524*, Computer Science Division, University of California, Berkeley, September 1992.

[22] SPEC. http://www.spec.org/

[23] C. D. Spradling. SPEC CPU2006 benchmark tools, *Computer Architecture News*, Vol. 35, No. 1, pp. 130-134, March 2007.

[24] H. Vandierendonck and K. De Bosschere. Many Benchmarks Stress the Same Bottlenecks, in *Proceedings of the Workshop on Computer Architecture Evaluation using Commercial Workloads (CAECW-7)*, pp. 75-71, 2004.

[25] R. P. Weicker and J. L. Henning. Subroutine profiling results for the CPU2006 benchmarks,

Computer Architecture News, Vol. 35, No. 1, pp. 102-111, March 2007.

[26] M. Wong. C++ Benchmarks in SPEC CPU2006, *Computer Architecture News*, Vol. 35, No. 1, pp. 77-83, March 2007.

[27] R. E. Wunderlich, T. F. Wenisch, B. Falsafi, and J. C. Hoe. SMARTS: Accelerating microarchitecture simulation via rigorous statistical sampling, in Proceedings of the International Symposium on Computer Architecture, pp. 84-95, June 2003.

[28] D. Ye, J. Ray, C. Harle, and D. Kaeli. Performance Characterization of SPEC CPU2006 Integer Benchmarks on x86-64 Architecture, in *Proceeding of the IEEE International Symposium on Workload Characterization*, pp. 120-127, October 2006.

[29] D. Ye, J. Ray and D. Kaeli. Characterization of file I/O activity for SPEC CPU2006, *Computer Architecture News*, Vol. 35, No. 1, pp. 112-117, March 2007.

[30] J.Yi, D. Lilja., Simulation of Computer Architectures: Simulators, Benchmarks, Methodologies, and Recommendations, In *IEEE Transactions on Computers*, Vol. 55, No. 3, pp. 268-280, March 2006.

Annex I: dynamic instruction count in the SPEC CPUint2006

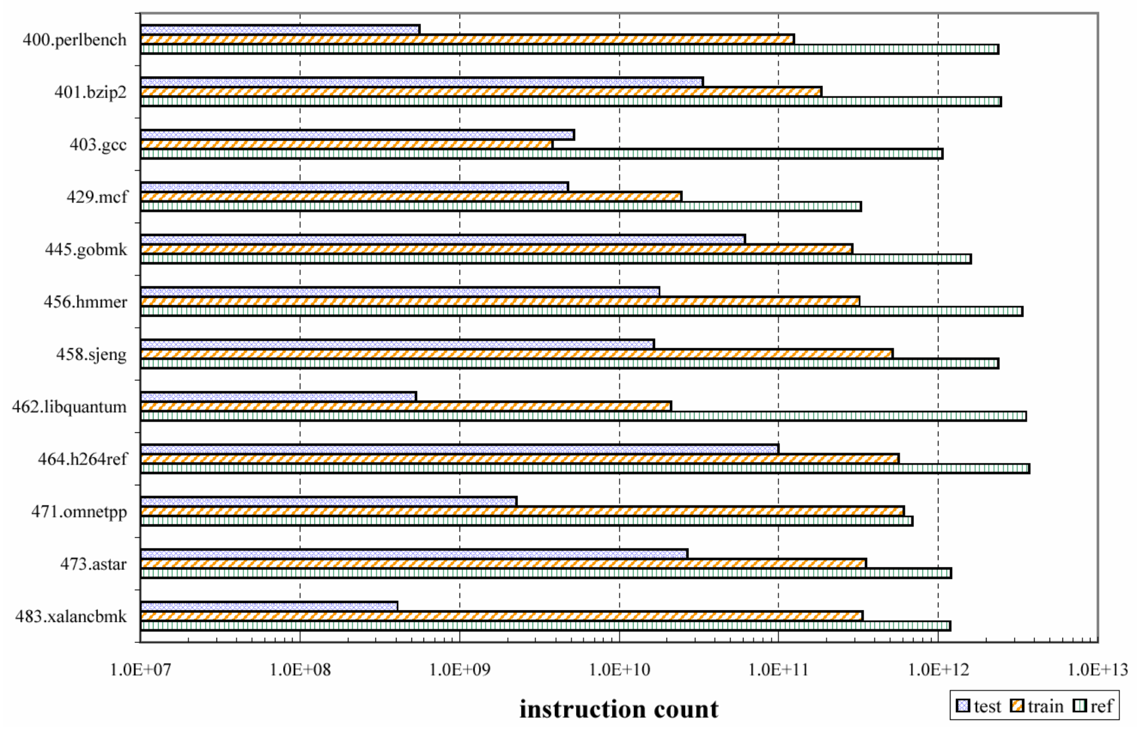


Fig. 1. Dynamic instruction count for each program in the integer suite SPEC CPUint2006 for each workload (test, train and reference). The instruction count axis is presented in logarithmic scale.

TEST workload

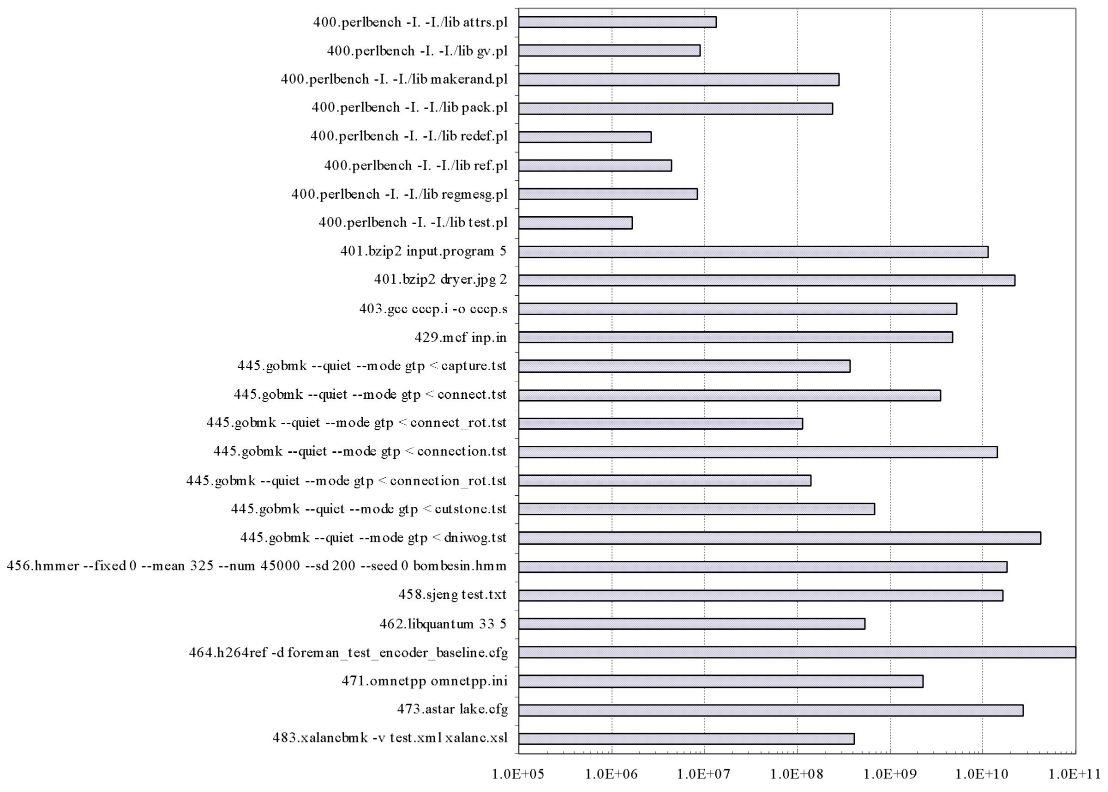


Fig. 2. Dynamic instruction count for each program invocation in the integer suite SPEC CPUint2006 for test workload. The instruction count axis is presented in logarithmic scale.

TRAIN workload

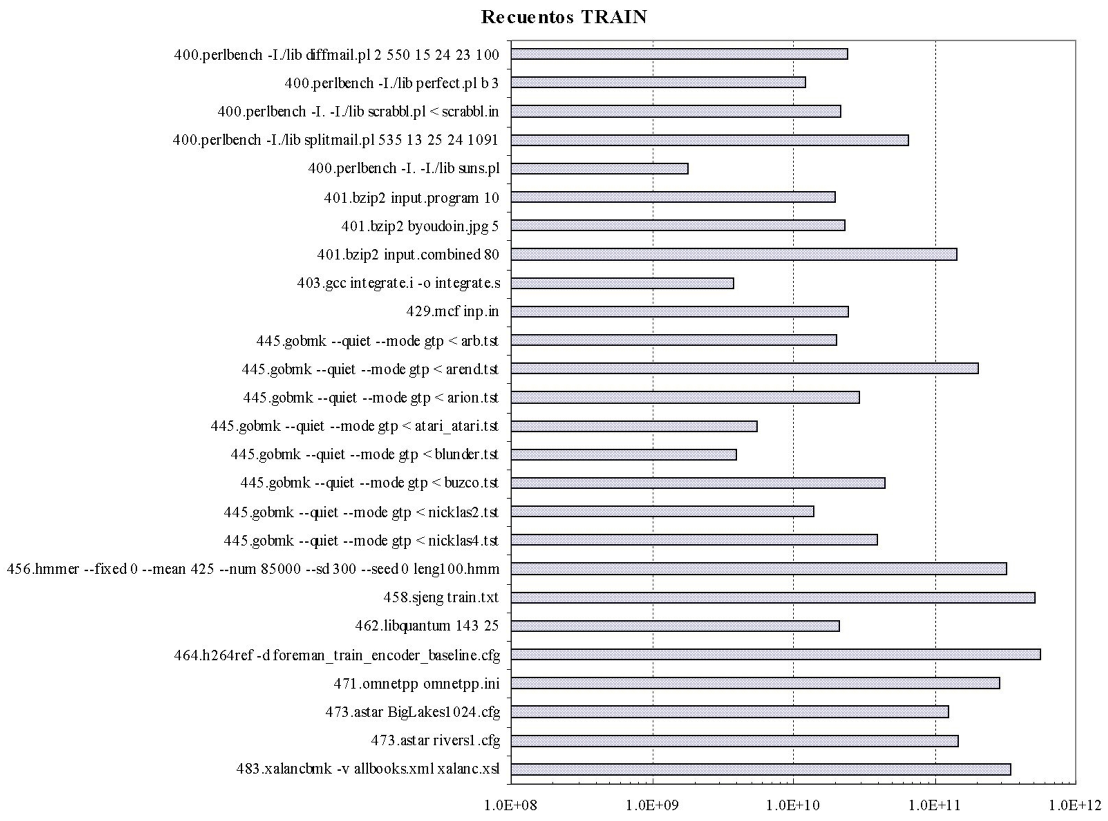


Fig. 3. Dynamic instruction count for each program invocation in the integer suite SPEC CPUint2006 for train workload. The instruction count axis is presented in logarithmic scale.

REF workload

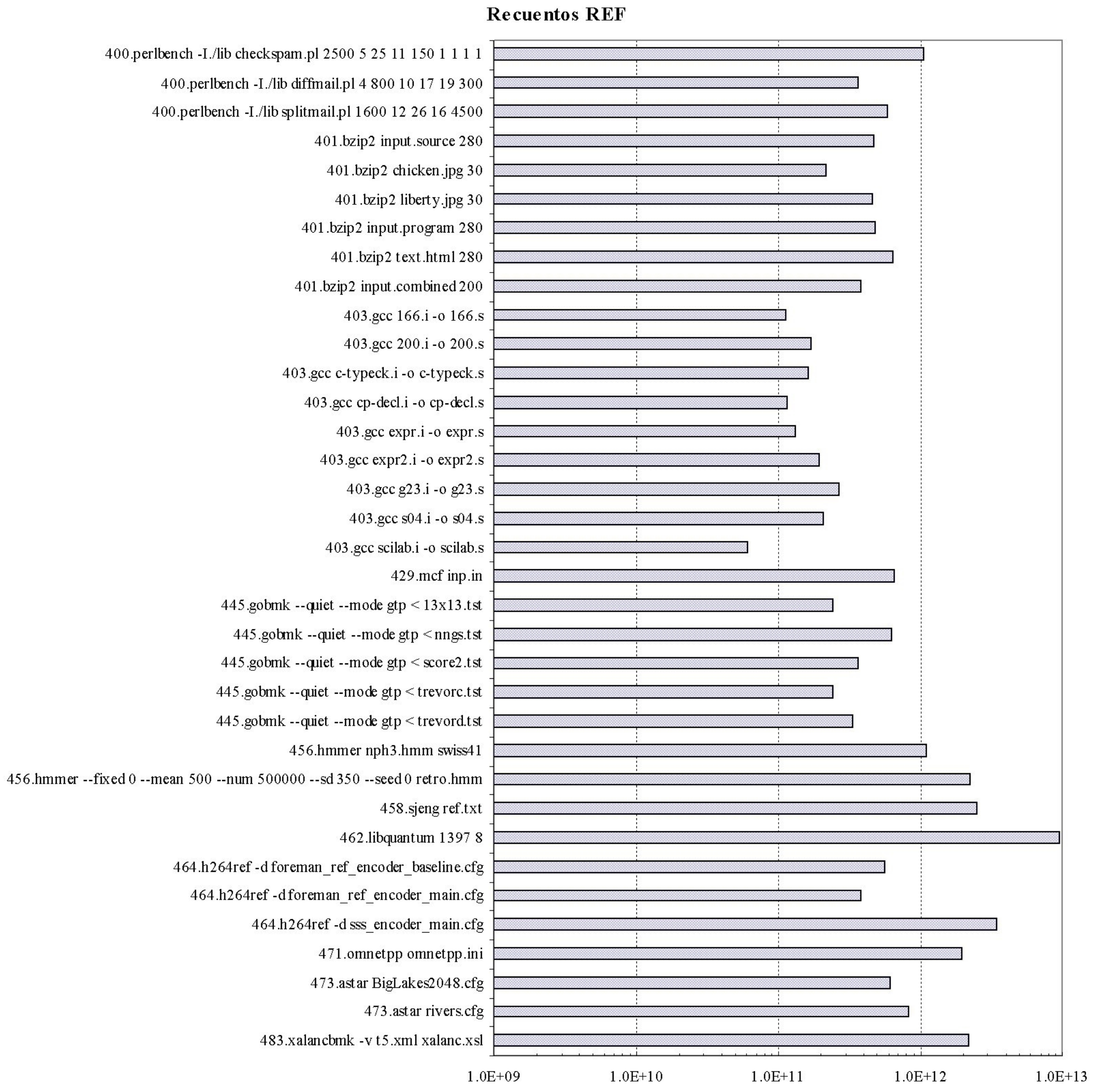
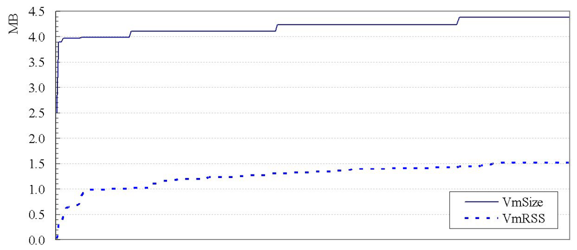


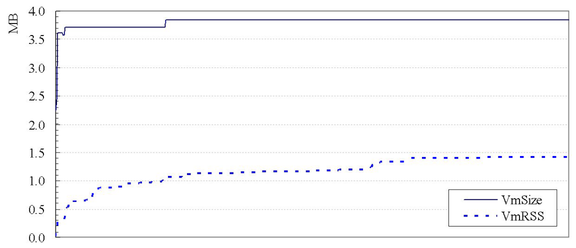
Fig. 4. Dynamic instruction count for each program invocation in the integer suite SPEC CPUint2006 for reference workload. The instruction count axis is presented in logarithmic scale.

Annex II: memory usage profiles in the SPEC CPUint2006

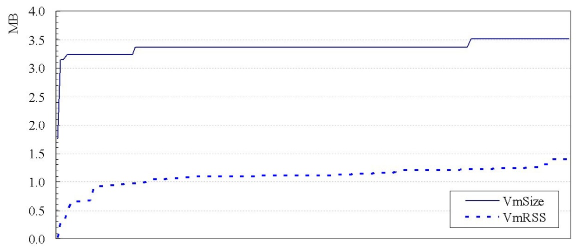
TEST workload



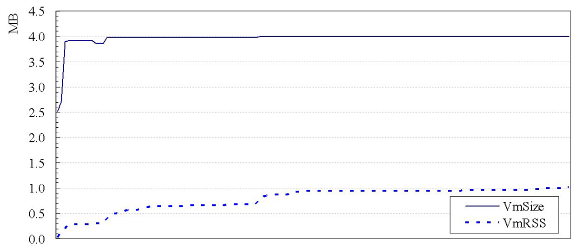
400.perlbench -I. -I./lib attrs.pl



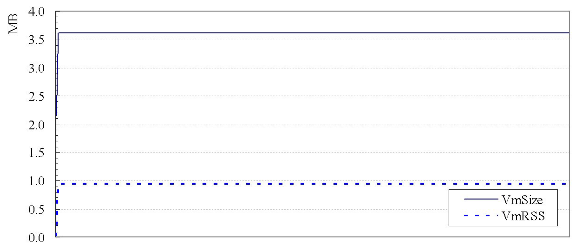
400.perlbench -I. -I./lib regmesg.pl



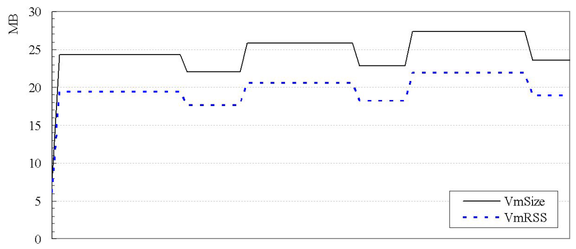
400.perlbench -I. -I./lib gv.pl



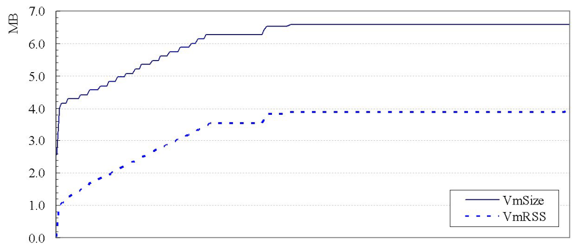
400.perlbench -I. -I./lib test.pl



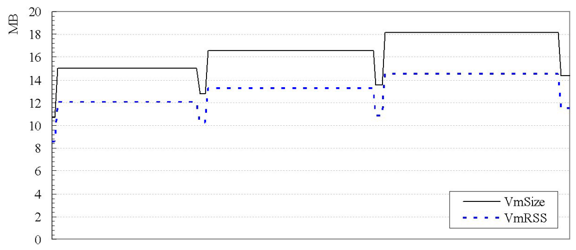
400.perlbench -I. -I./lib makerand.pl



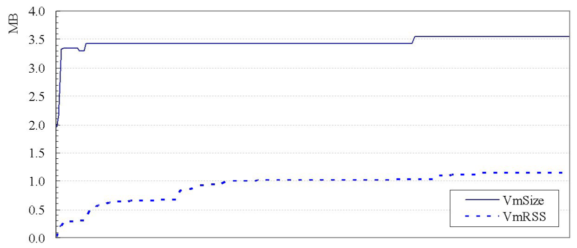
401.bzip2 input.program 5



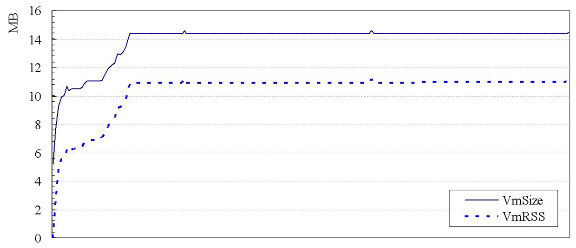
400.perlbench -I. -I./lib pack.pl



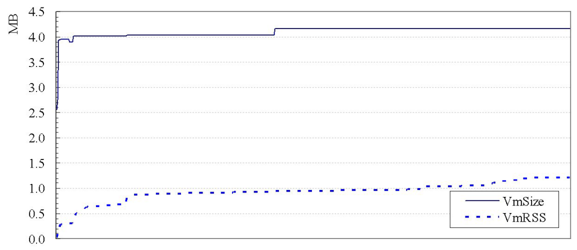
401.bzip2 dryer.jpg 2



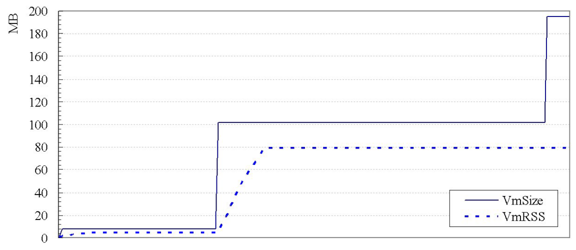
400.perlbench -I. -I./lib redef.pl



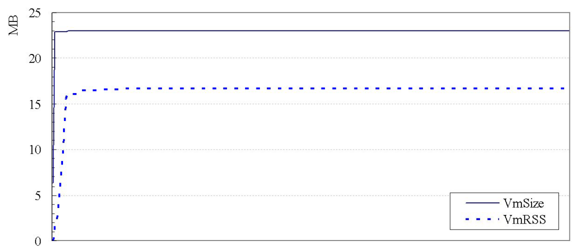
403.gcc cccp.i -o cccp.s



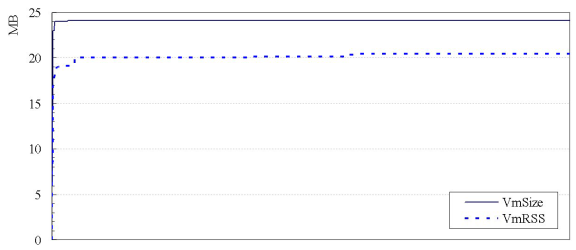
400.perlbench -I. -I./lib ref.pl



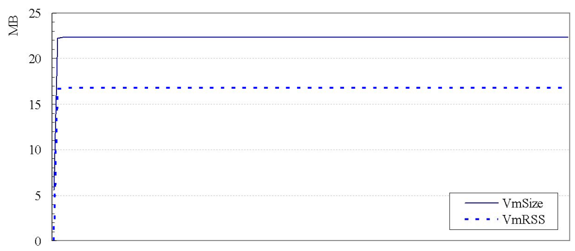
429.mcf inp.in



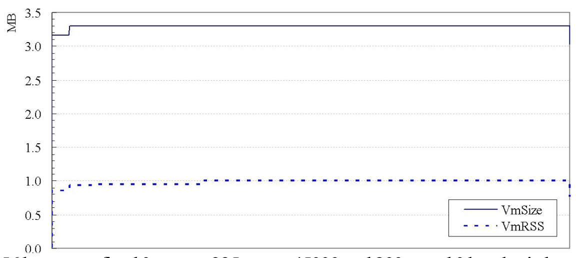
445.gobmk --quiet --mode gtp < capture.tst



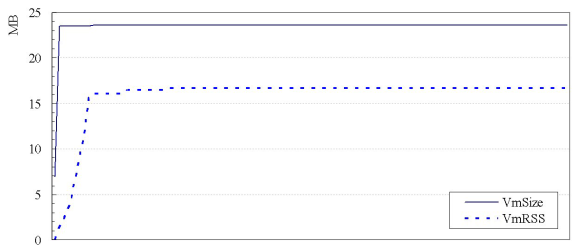
445.gobmk --quiet --mode gtp < dniwog.tst



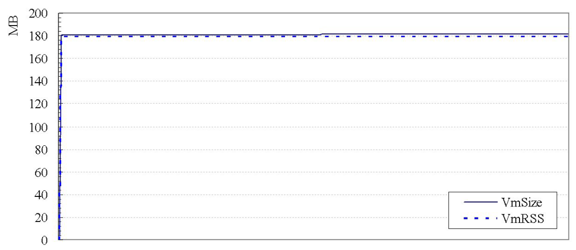
445.gobmk --quiet --mode gtp < connect.tst



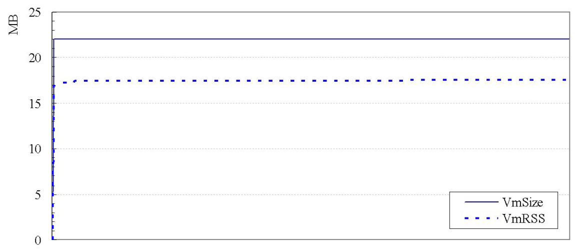
456.hmmer --fixed 0 --mean 325 --num 45000 --sd 200 --seed 0 bombesin.hmm



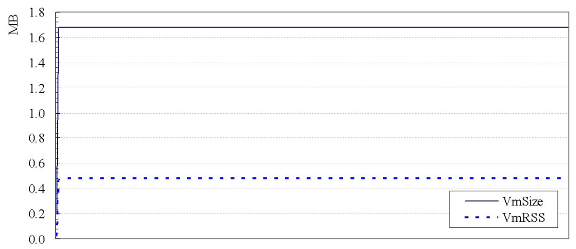
445.gobmk --quiet --mode gtp < connect_rot.tst



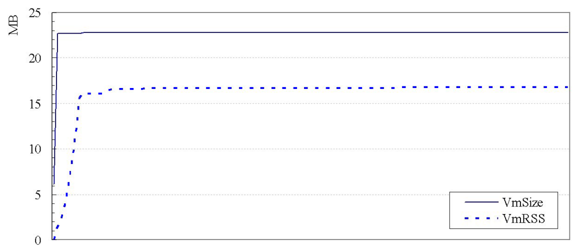
458.sjeng test.txt



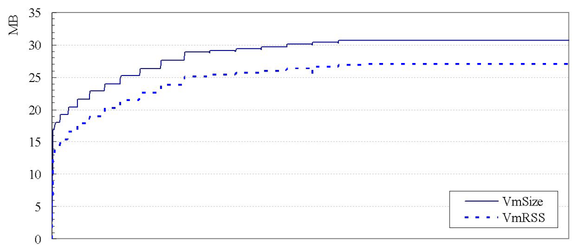
445.gobmk --quiet --mode gtp < connection.tst



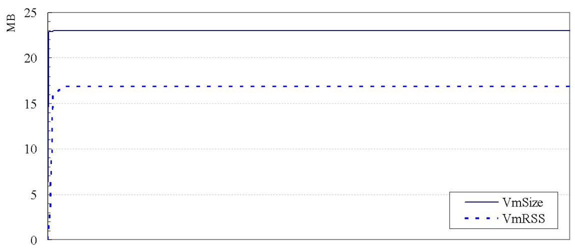
462.libquantum 33 5



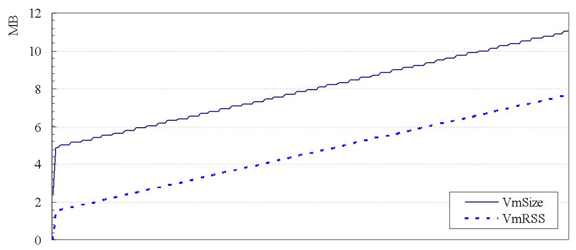
445.gobmk --quiet --mode gtp < connection_rot.tst



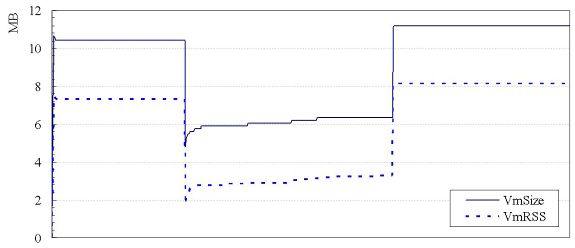
464.h264ref -d foreman_test_encoder_baseline.cfg



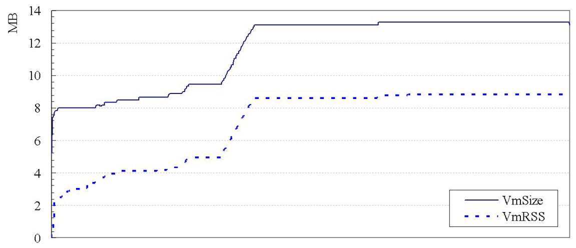
445.gobmk --quiet --mode gtp < cutstone.tst



471.omnetpp omnetpp.ini

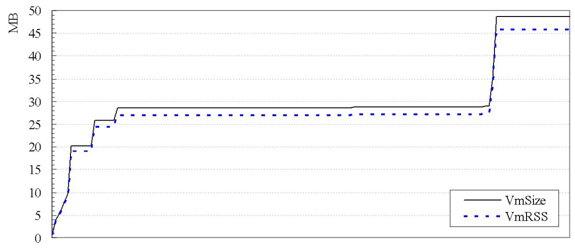


473.astar lake.cfg

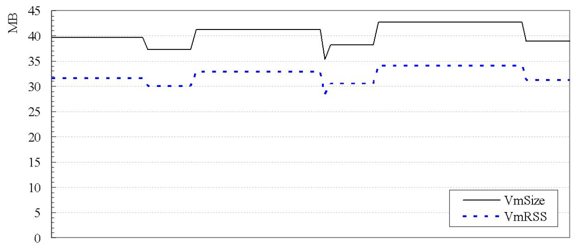


483.xalancbmk -v test.xml xalanc.xsl

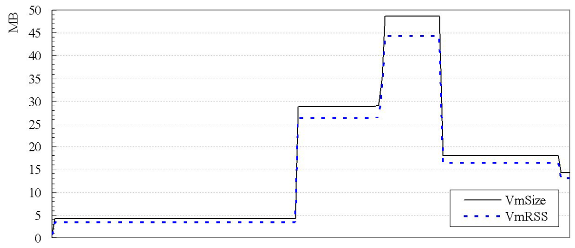
TRAIN workload



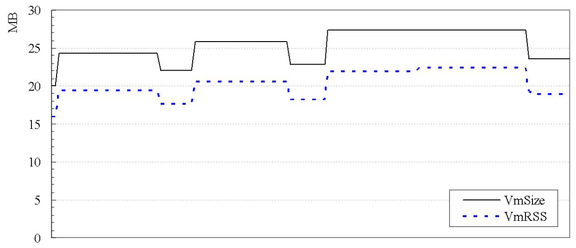
400.perlbench -I./lib diffmail.pl 2 550 15 24 23 100



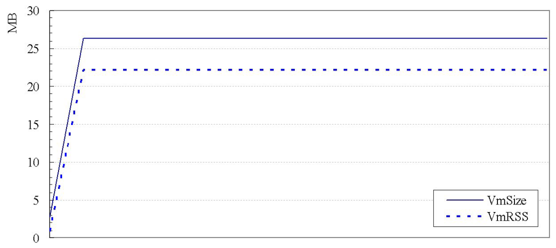
401.bzip2 input.program 10



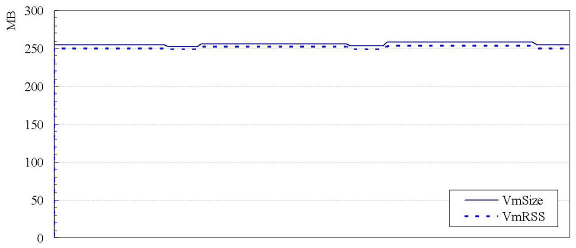
400.perlbench -I./lib perfect.pl b 3



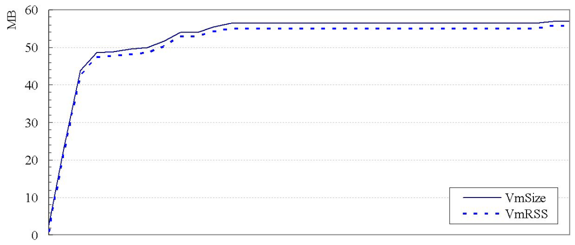
401.bzip2 byoudoin.jpg 5



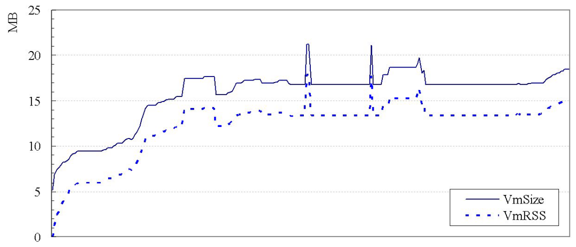
400.perlbench -I. -I./lib scrabbl.pl < scrabbl.in



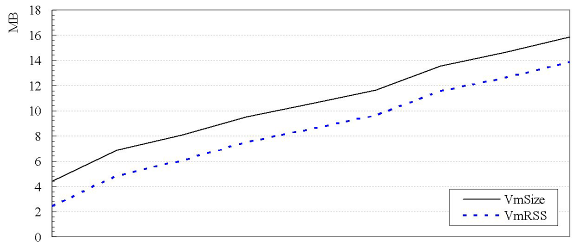
401.bzip2 input.combined 80



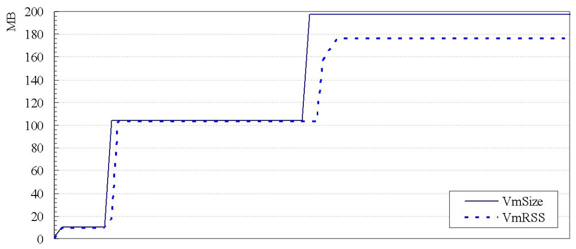
400.perlbench -I./lib splitmail.pl 535 13 25 24 1091



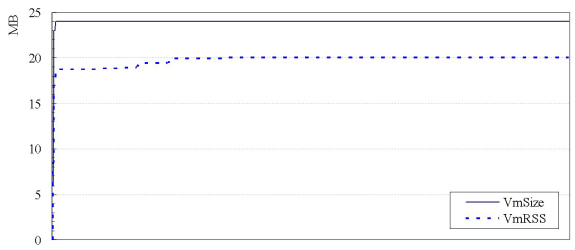
403.gcc integrate.i -o integrate.s



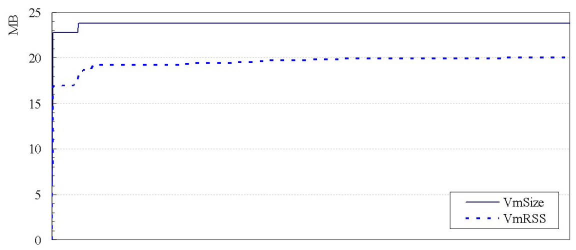
400.perlbench -I. -I./lib suns.pl



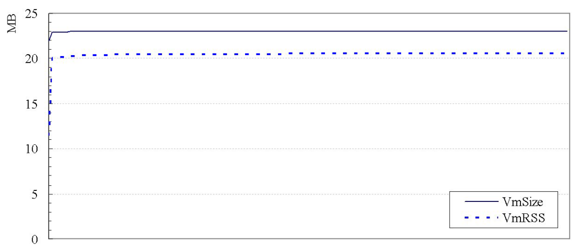
429.mcf inp.in



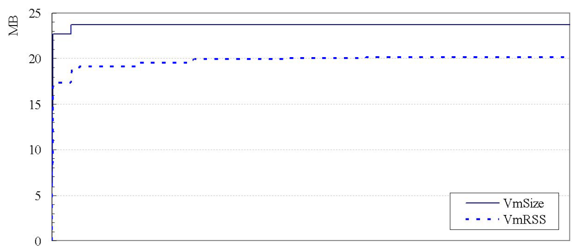
445.gobmk --quiet --mode gtp < arb.tst



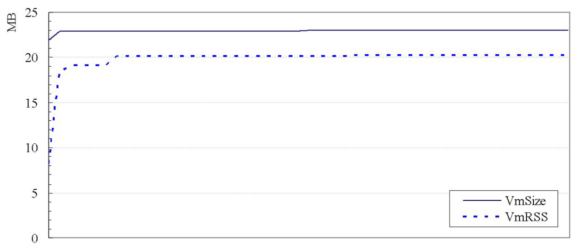
445.gobmk --quiet --mode gtp < nicklas2.tst



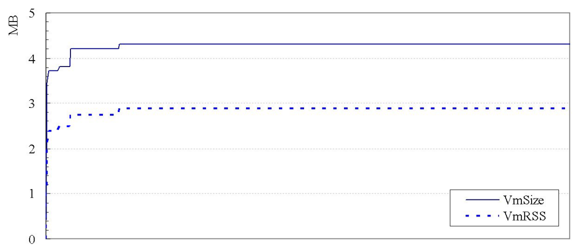
445.gobmk --quiet --mode gtp < arend.tst



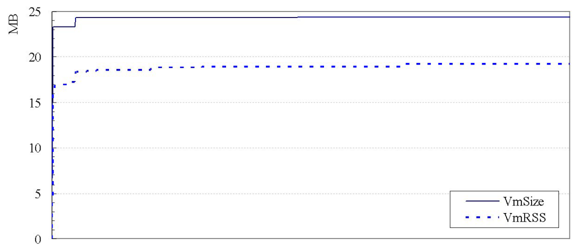
445.gobmk --quiet --mode gtp < nicklas4.tst



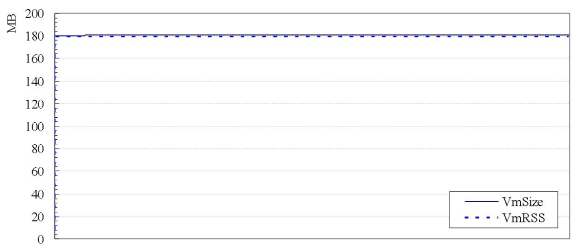
445.gobmk --quiet --mode gtp < arion.tst



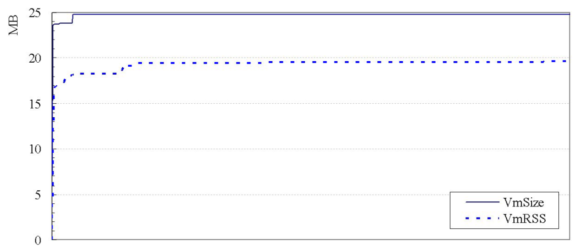
456.hmmer --fixed 0 --mean 425 --num 85000 --sd 300 --seed 0 leng100.hmm



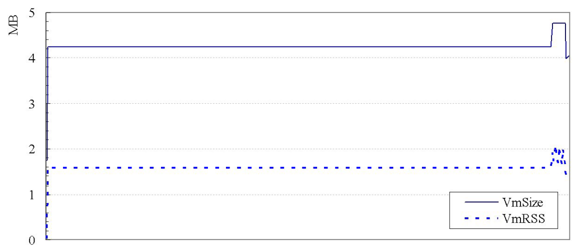
445.gobmk --quiet --mode gtp < atari_atari.tst



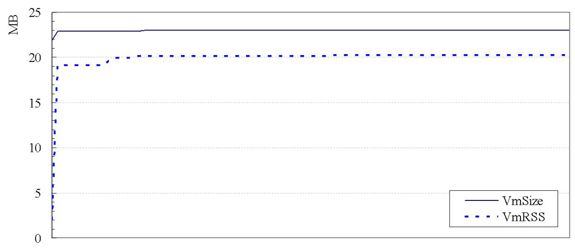
458.sjeng train.txt



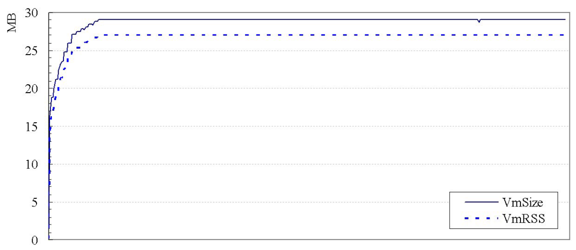
445.gobmk --quiet --mode gtp < blunder.tst



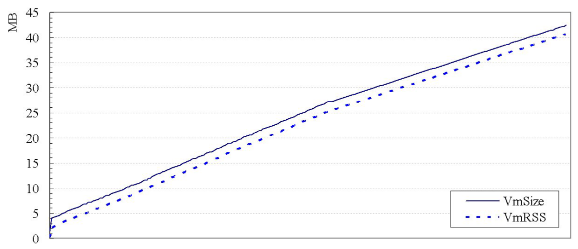
462.libquantum 143 25



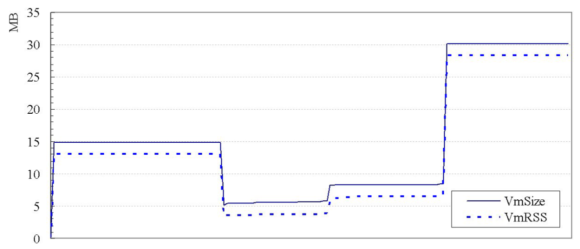
445.gobmk --quiet --mode gtp < buzco.tst



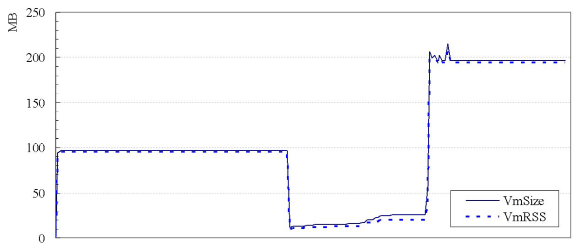
464.h264ref -d foreman_train_encoder_baseline.cfg



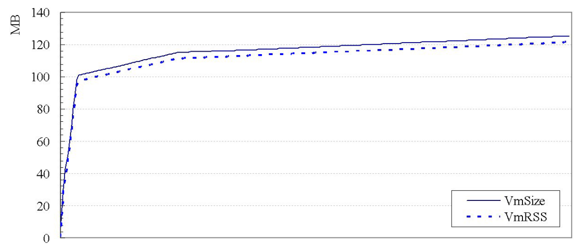
471.omnetpp   omnetpp.ini



473.astar   rivers1.cfg

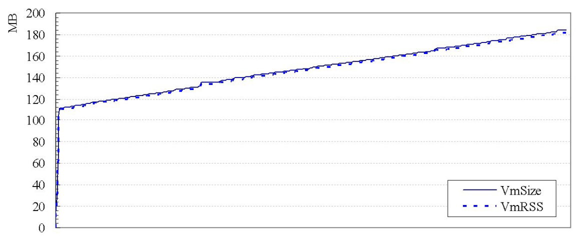


473.astar   BigLakes1024.cfg

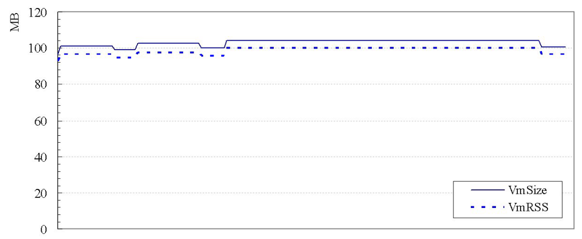


483.xalancbmk   -v allbooks.xml xalanc.xsl

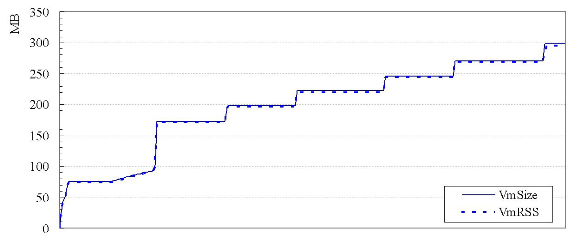
REF workload



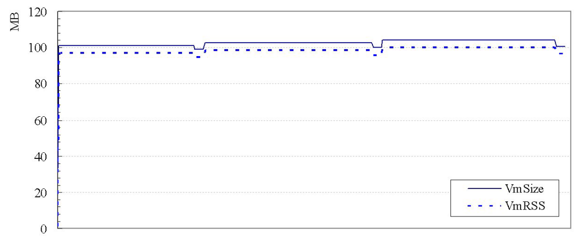
400.perlbench   -I./lib checkspam.pl 2500 5 25 11 150 1 1 1 1



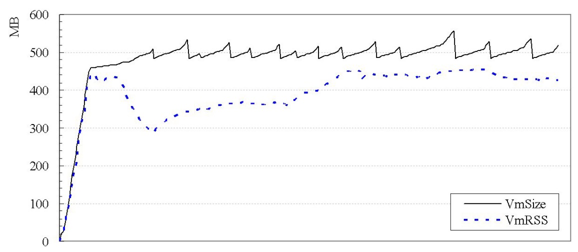
401.bzip2   chicken.jpg 30



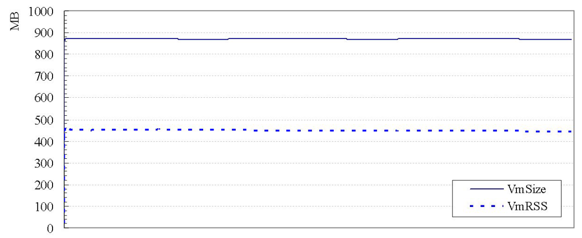
400.perlbench   -I./lib diffmail.pl 4 800 10 17 19 300



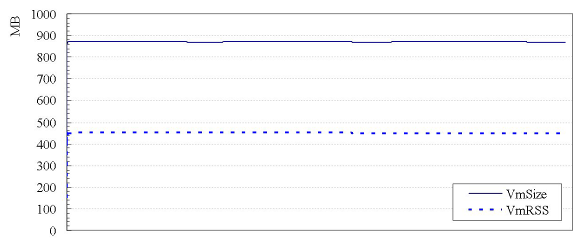
401.bzip2   liberty.jpg 30



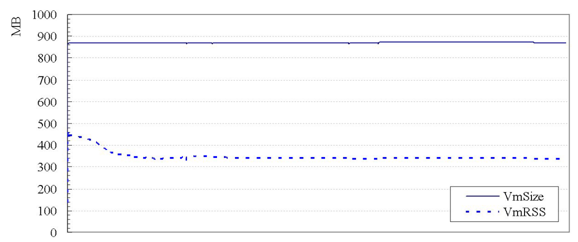
400.perlbench   -I./lib splitmail.pl 1600 12 26 16 4500



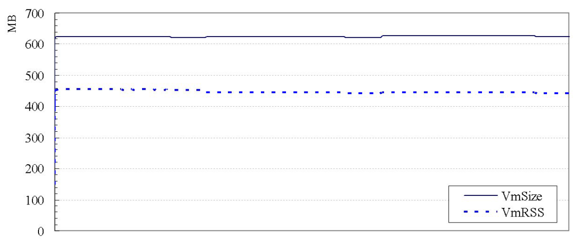
401.bzip2   input.program 280



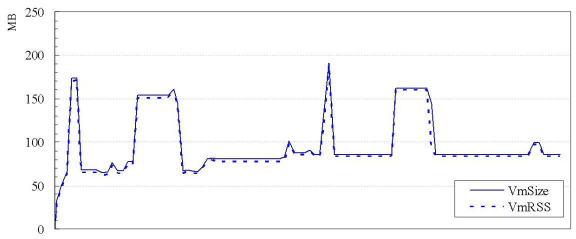
401.bzip2   input.source 280



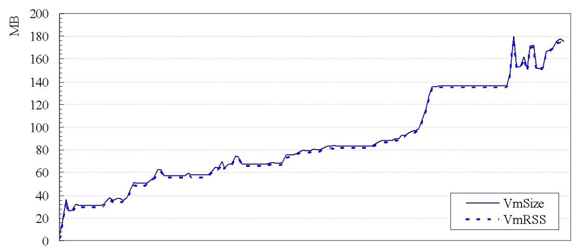
401.bzip2   text.html 280



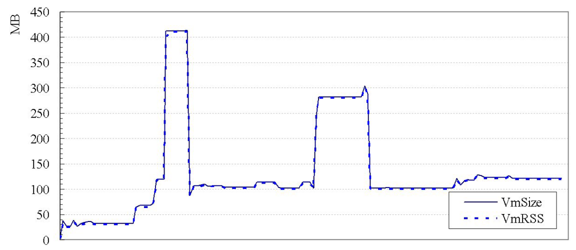
401.bzip2 _input.combined 200



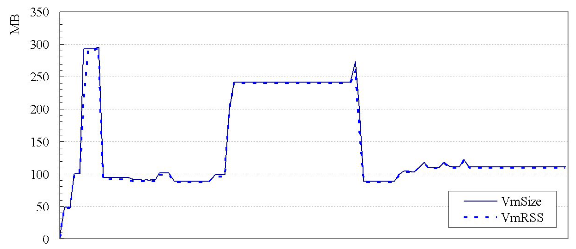
403.gcc 166.i -o 166.s



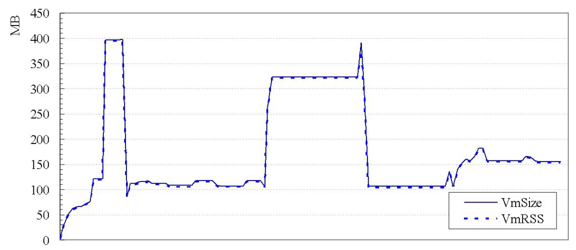
403.gcc 200.i -o 200.s



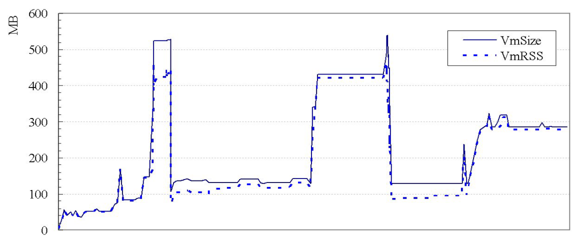
403.gcc c-typeck.i -o c-typeck.s



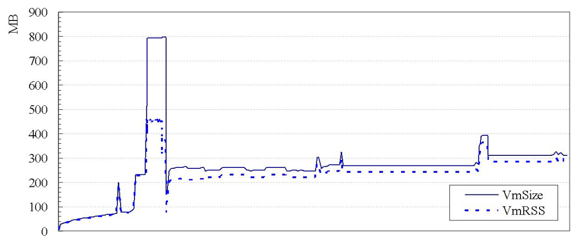
403.gcc cp-decl.i -o cp-decl.s



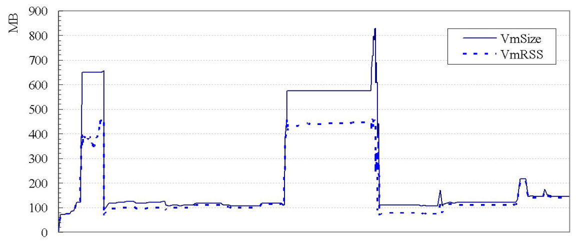
403.gcc expr.i -o expr.s



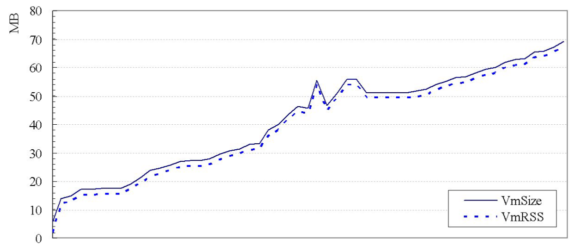
403.gcc expr2.i -o expr2.s



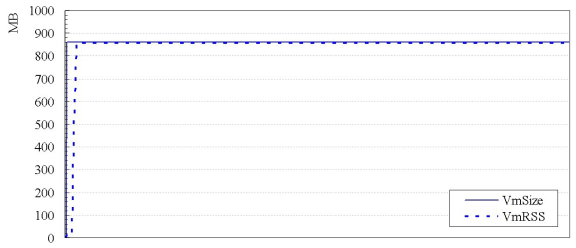
403.gcc g23.i -o g23.s



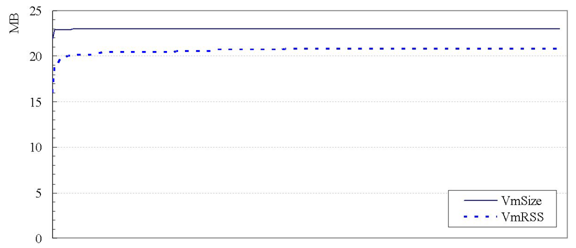
403.gcc s04.i -o s04.s



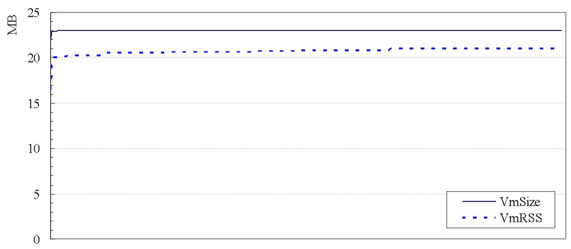
403.gcc scilab.i -o scilab.s



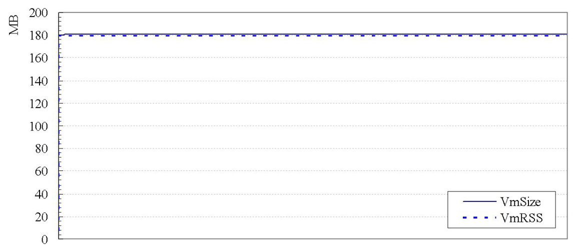
429.mcf inp.in



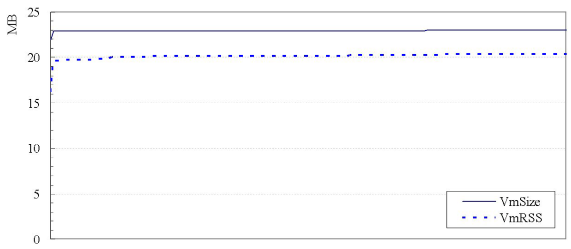
445.gobmk --quiet --mode gtp < 13x13.tst



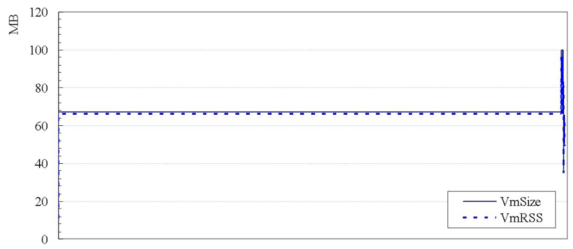
445.gobmk --quiet --mode gtp < nngs.tst



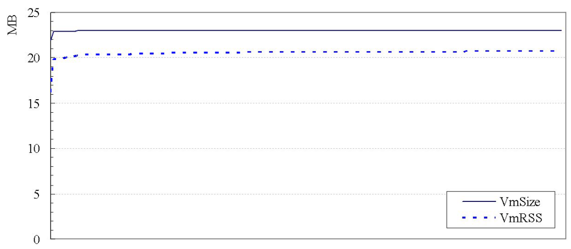
458.sjeng ref.txt



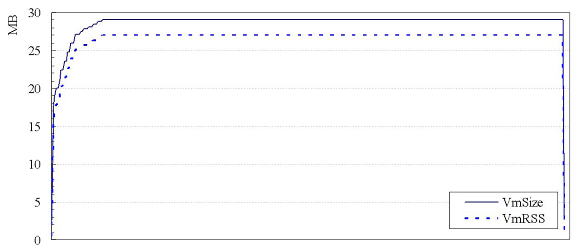
445.gobmk --quiet --mode gtp < score2.tst



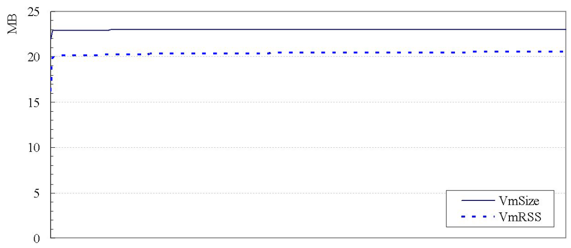
462.libquantum 1397 8



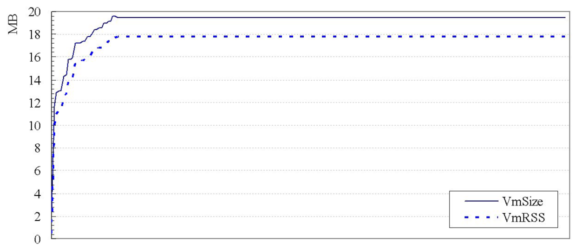
445.gobmk --quiet --mode gtp < trevorc.tst



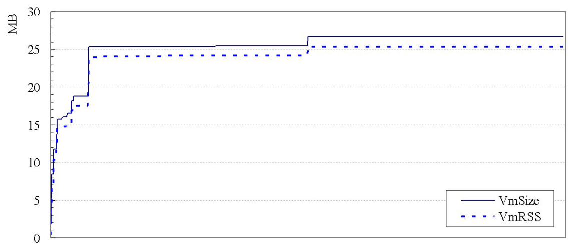
464.h264ref -d foreman_ref_encoder_baseline.cfg



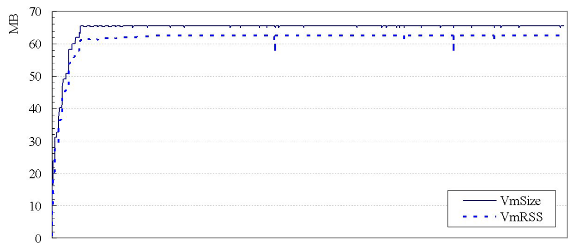
445.gobmk --quiet --mode gtp < trevord.tst



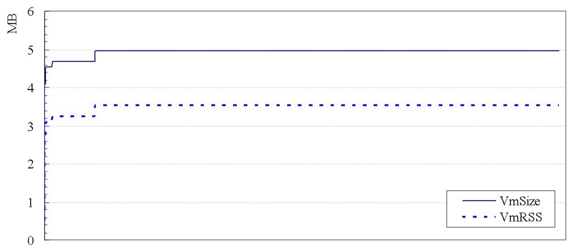
464.h264ref -d foreman_ref_encoder_main.cfg



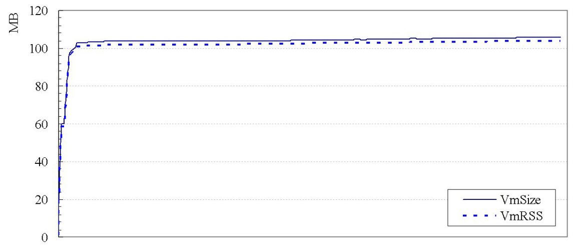
456.hmmer nph3.hmm swiss41



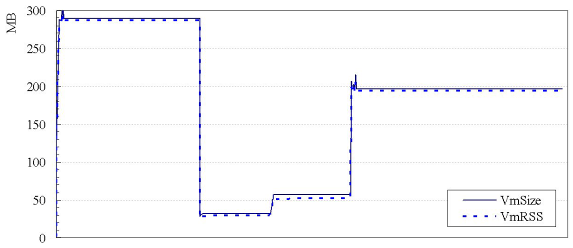
464.h264ref -d sss_encoder_main.cfg



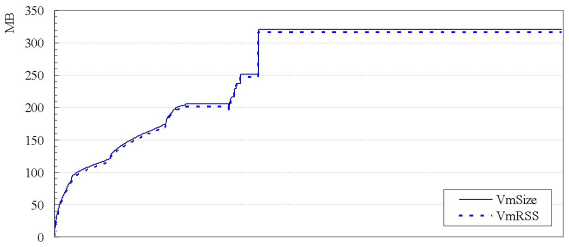
456.hmmer --fixed 0 --mean 500 --num 500000 --sd 350 --seed 0 retro.hmm



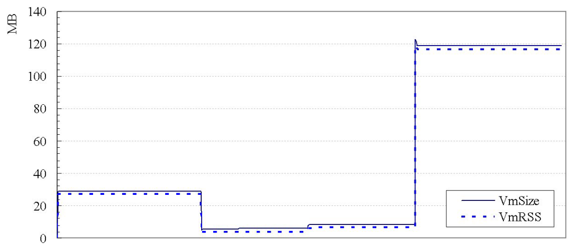
471.omnetpp omnetpp.ini



473.astar   BigLakes2048.cfg



483.xalancbmk   -v t5.xml xalanc.xsl



473.astar   rivers.cfg

Annex III: subroutine profiles in the SPEC CPUint2006

TEST workload

400.perlbench

Invocation: 400-perlbench-attrs.pl
```
  % time  name
   22.22  Perl_yylex
   11.11  Perl_sv_upgrade
   11.11  Perl_sv_grow
   11.11  S_hv_fetch_common
   11.11  Perl_av_fetch
   11.11  Perl_pad_findmy
   11.11  Perl_sv_setpv
   11.11  Perl_yyparse
```

Invocation: 400-perlbench-gv.pl
```
  % time  name
   14.29  Perl_grok_hex
   14.29  Perl_newSVOP
   14.29  Perl_sv_gets
   14.29  S_scan_word
   14.29  Perl_peep
   14.29  Perl_yyparse
   14.29  perl_parse
```

Invocation: 400-perlbench-makerand.pl
```
  % time  name
   10.87  Perl_pp_modulo
    8.15  Perl_pp_padsv
    7.61  Perl_pp_predec
    7.07  Perl_pp_rand
    6.52  Perl_pp_gt
    5.98  Perl_pp_const
    5.43  Perl_runops_standard
    4.89  Perl_pp_int
    4.35  Perl_sv_setuv
    4.35  Perl_sv_setsv_flags
    4.35  spec_rand
    3.80  Perl_pp_and
    3.80  Perl_pp_sassign
    3.26  Perl_pp_nextstate
    3.26  Perl_cast_uv
    3.26  Perl_pp_gvsv
    2.17  Perl_sv_2iv
    2.17  S_dopoptoloop
    2.17  Perl_cast_iv
    1.63  Perl_sv_setiv
```

Invocation: 400-perlbench-pack.pl
```
  % time  name
    5.56  Perl_yyparse
    4.32  Perl_pp_padsv
    3.70  Perl_sv_setsv_flags
    3.70  S_hv_fetch_common
    3.70  Perl_pp_entersub
    3.70  Perl_gv_fetchpv
    3.70  S_unpack_rec
    3.09  Perl_sv_clear
    3.09  Perl_yylex
    3.09  uiv_2buf
    2.47  Perl_sv_upgrade
    2.47  Perl_pp_return
    2.47  Perl_sv_gets
    1.85  Perl_sv_catpvn_flags
    1.85  Perl_pp_nextstate
    1.85  Perl_pp_pushmark
    1.85  Perl_leave_scope
    1.85  Perl_pp_concat
    1.85  S_skipspace
    1.85  Perl_runops_standard
```

Invocation: 400-perlbench-redef.pl
```
  % time  name
   50.00  Perl_yyparse
   25.00  Perl_yylex
   25.00  Perl_sv_gets
```

Invocation: 400-perlbench-ref.pl
```
  % time  name
   16.67  Perl_yylex
   16.67  Perl_newSVpv
   16.67  Perl_keyword
   16.67  Perl_pp_pushmark
   16.67  Perl_yyparse
   16.67  perl_destruct
```

Invocation: 400-perlbench-regmesg.pl
```
  % time  name
   25.00  PerlIOBuf_get_cnt
   25.00  Perl_leave_scope
   25.00  Perl_pp_regcreset
   25.00  Perl_pp_cond_expr
```

Invocation: 400-perlbench-test.pl
```
  % time  name
   11.11  Perl_pp_nextstate
   11.11  Perl_yylex
   11.11  Perl_sv_setpvn
   11.11  Perl_leave_scope
   11.11  Perl_sv_setiv
   11.11  PerlIOBuf_write
   11.11  uiv_2buf
   11.11  Perl_pp_waitpid
   11.11  Perl_sv_inc
```

401.bzip2

Invocation: 401-bzip2-dryer-2
```
  % time  name
   67.37  fallbackSort
   18.36  mainGtU
    4.15  BZ2_compressBlock
    3.49  BZ2_bzDecompress
    3.14  BZ2_decompress
    2.57  BZ2_blockSort
```

Invocation: 401-bzip2-input.program-5
```
  % time  name
   35.50  BZ2_blockSort
   19.17  BZ2_compressBlock
   18.74  BZ2_decompress
   11.21  BZ2_bzDecompress
    8.96  mainGtU
    3.61  handle_compress
    1.20  bsW
```

403.gcc

Invocation: 403-gcc-test
```
  % time  name
    3.22  propagate_one_insn
    2.65  approx_reg_cost
    2.53  for_each_rtx
    2.38  cse_insn
    2.14  ggc_mark_rtx_children_1
    1.99  init_alias_analysis
    1.90  mark_set_1
    1.87  ggc_set_mark
    1.75  constrain_operands
    1.60  ggc_alloc
    1.51  bitmap_operation
    1.42  reg_scan_mark_refs
    1.33  bitmap_set_bit
    1.18  canon_reg
    1.11  fold_rtx
    1.11  record_reg_classes
    1.08  ggc_mark_rtx_children
    1.05  bitmap_bit_p
    1.05  ggc_mark_trees
```

429.mcf

Invocation: 429-mcf-inp.in
```
  % time  name
   67.47  primal_bea_mpp
   11.39  price_out_impl
    6.96  refresh_potential
    3.22  bea_is_dual_infeasible
    2.62  sort_basket
    2.24  insert_new_arc
    2.05  update_tree
    1.09  primal_iminus
```

445.gobmk

Invocation: 445-gobmk-capture.tst
```
  % time  name
   73.04  hashtable_clear
    4.41  new_position
    1.96  fastlib
    1.96  propagate_string
    1.96  do_play_move
    1.72  is_self_atari
    1.47  do_get_read_result
    1.47  propose_edge_moves
    1.23  order_moves
    1.23  update_liberties
```

Invocation: 445-gobmk-connect.tst
```
  % time  name
   36.58  hashtable_clear
    9.28  compute_connection_distances
    8.78  do_play_move
    3.96  fastlib
    3.41  incremental_order_moves
    3.07  assimilate_string
    3.01  order_moves
    2.61  popgo
    2.40  new_position
    1.72  approxlib
    1.63  is_suicide
    1.29  do_trymove
    1.26  remove_liberty
    1.20  simple_ladder_attack
    1.11  is_self_atari
    1.07  do_attack
    1.01  update_liberties
    1.01  do_find_defense
```

Invocation: 445-gobmk-connect_rot.tst
```
  % time  name
   57.14  hashtable_clear
    7.14  do_play_move
    4.29  compute_connection_distances
    2.86  order_moves
    2.14  count_common_libs
    2.14  do_get_read_result
    2.14  propagate_string
    2.14  do_find_defense
    1.43  remove_liberty
    1.43  is_suicide
    1.43  incremental_order_moves
    1.43  assimilate_string
    1.43  edge_block_moves
    1.43  store_persistent_reading_cache
```

Invocation: 445-gobmk-connection.tst
```
  % time  name
   15.15  do_play_move
   11.44  compute_connection_distances
    6.81  fastlib
    5.29  incremental_order_moves
    4.99  hashtable_clear
    4.82  order_moves
    4.04  popgo
    3.85  assimilate_string
    2.52  approxlib
    2.26  hashtable_search
    2.10  is_self_atari
    2.05  do_find_defense
    1.96  is_suicide
    1.95  update_liberties
    1.89  remove_liberty
    1.80  do_attack
    1.68  do_trymove
    1.63  chainlinks2
    1.54  do_get_read_result
    1.46  count_common_libs
    1.05  remove_neighbor
    1.01  komaster_trymove
```

Invocation: 445-gobmk-connection_rot.tst
```
  % time  name
   42.86  hashtable_clear
    8.44  do_play_move
    7.79  compute_connection_distances
    3.90  remove_liberty
    3.90  fastlib
    3.90  incremental_order_moves
    3.25  store_persistent_reading_cache
    2.60  order_moves
    1.95  popgo
    1.95  count_common_libs
    1.30  is_suicide
    1.30  do_trymove
    1.30  chainlinks2
    1.30  assimilate_string
    1.30  simple_ladder_defend
    1.30  update_liberties
    1.30  do_find_defense
    1.30  find_connection_moves
```

Invocation: 445-gobmk-cutstone.tst
```
  % time  name
   23.93  hashtable_clear
    8.02  do_play_move
    5.43  fastlib
    5.30  order_moves
    4.66  incremental_order_moves
    4.53  hashtable_search
    4.14  popgo
    4.14  do_find_defense
    3.62  do_attack
    3.49  approxlib
    3.49  is_self_atari
    2.72  propose_edge_moves
    2.20  do_get_read_result
    2.20  assimilate_string
    1.94  update_liberties
    1.55  chainlinks2
    1.42  count_common_libs
    1.16  do_find_superstring
    1.03  komaster_trymove
    1.03  edge_clamp_moves
    1.03  attack3
```

Invocation: 445-gobmk-dniwog.tst
```
  % time  name
   10.75  dfa_matchpat_loop
    9.01  do_play_move
    5.67  fastlib
    4.72  order_moves
    4.42  incremental_order_moves
    3.42  hashtable_search
    3.26  assimilate_string
    3.12  accumulate_influence
    2.81  popgo
    2.70  do_find_defense
    2.69  compute_primary_domains
    2.55  get_next_move_from_list
    2.41  approxlib
    2.40  compute_connection_distances
    2.32  do_attack
    1.93  is_self_atari
    1.79  chainlinks2
    1.74  do_get_read_result
    1.48  count_common_libs
    1.37  matchpat_loop
```

456.hmmer

Invocation: 456-hmmer-bombesin
```
  % time  name
   69.20  P7Viterbi
   14.50  sre_random
   13.03  FChoose
    1.89  SymbolIndex
```

458.sjeng

Invocation: 458-sjeng-test
```
  % time  name
   15.63  std_eval
    9.02  clear_tt
    8.24  setup_attackers
    7.61  gen
    5.31  remove_one
    4.77  order_moves
    4.41  search
    3.99  QProbeTT
    3.63  push_slidE
    3.51  rook_mobility
    3.39  is_attacked
    3.16  Pawn
    2.97  ProbeTT
    2.95  make
    2.65  checkECache
    2.11  unmake
    2.08  check_legal
    1.73  bishop_mobility
    1.65  see
    1.57  Rook
```

462.libquantum

Invocation: 462-libquantum-33-5
```
    % time  name
     55.14  quantum_toffoli
     15.14  quantum_cnot
     11.35  quantum_sigma_x
      5.95  quantum_state_collapse
      5.41  quantum_swaptheleads
      4.32  __umoddi3
```

464.h264ref

Invocation: 464-h264ref-foreman-test
```
    % time  name
     25.03  SetupFastFullPelSearch
     20.36  FastFullPelBlockMotionSearch
     11.83  SATD
      6.74  SubPelBlockMotionSearch
      6.60  FastPelY_14
      4.13  dct_luma
      3.58  SetupLargerBlocks
      3.23  UMVLine16Y_11
      2.47  FastLine16Y_11
      1.13  BlockMotionSearch
      1.07  getNonAffNeighbour
      1.07  UMVPelY_14
      1.07  writeCoeff4x4_CAVLC
```

471.omnetpp

Invocation: 471-omnetpp-test
```
    % time  Name
      6.59  cObject::setOwner
      5.74  cMessageHeap::insert
      5.42  cModule::findGate
      4.89  TCmdenvApp::simulate
      3.24  EtherMAC::printState
      3.08  cMessageHeap::shiftup
      2.87  EtherMAC::processReceivedDataFrame
      2.34  cSimpleModule::sendDelayed
      2.23  EtherMAC::handleMessage
      2.23  EtherMAC::handleEndTxPeriod
      2.02  cSimulation::doOneEvent
      2.02  cGate::deliver
      1.70  cMessageHeap::getFirst
      1.54  MACAddress::equals
      1.49  cSimpleModule::scheduleAt
      1.38  EtherLLC::processPacketFromHigherLayer
      1.28  cSimulation::selectNextModule
      1.28  cMessage::operator=
      1.17  cObject::~cObject
      1.17  cMessage::cMessage
```

473.astar

Invocation: 473-astar-lake
```
    % time  name
     45.95  wayobj::makebound2
     18.30  way2obj::releasepoint
     15.44  regwayobj::makebound2
      4.84  regmngobj::getregfillnum
      4.22  regwayobj::isaddtobound
      3.00  way2obj::addtobound
      2.11  way2obj::releasebound
      1.70  regwayobj::addtobound
      1.40  way2obj::isaddtobound
```

483.xalan

Invocation: 483-xalan-test
```
    % time  name
     13.41  __gnu_cxx::__normal_iterator
      5.36  xalanc_1_8::ReusableArenaBlock::ownsObject
      3.07  xalanc_1_8::FunctionSubstring::execute
      2.68  xalanc_1_8::DoubleSupport::round
      2.30  xalanc_1_8::XalanReferenceCountedObject::addRefer
            ence
      2.30  xalanc_1_8::XalanReferenceCountedObject::removeRe
            ference
      2.30  xalanc_1_8::ReusableArenaBlock::blockAvailable
      2.30  xalanc_1_8::VariablesStack::findEntry
      2.30  xalanc_1_8::XStringCachedAllocator::destroy
      1.92  xalanc_1_8::XalanDOMString::equals
      1.92  xalanc_1_8::XPath::executeMore
      1.92  xalanc_1_8::ElemTemplateElement::executeChildren
      1.92  xalanc_1_8::StylesheetExecutionContextDefault::ge
            tParams
      1.53  xalanc_1_8::XalanBitmap::isSet
      1.53  xalanc_1_8::XalanDOMString::equals
      1.53  xalanc_1_8::VariablesStack::push
      1.53  xalanc_1_8::XPath::runFunction
      1.53  xalanc_1_8::ElemChoose::execute
      1.15  xalanc_1_8::VariablesStack::StackEntry
      1.15  xalanc_1_8::XPath::variable
```

TRAIN workload

400.perlbench

Invocation: 400-perlbench-diffmail.pl
```
    % time  name
     18.83  S_regmatch
      5.60  Perl_sv_setsv_flags
      4.67  Perl_regexec_flags
      4.06  Perl_pp_padsv
      3.26  S_hv_fetch_common
      3.00  Perl_leave_scope
      2.80  Perl_pp_nextstate
      2.71  Perl_sv_clear
      2.44  Perl_sv_upgrade
      2.28  S_regrepeat
      1.98  Perl_pp_entersub
      1.92  Perl_sv_setpvn
      1.91  Perl_pp_helem
      1.70  S_regtry
      1.57  Perl_sv_eq
      1.49  Perl_pp_and
      1.46  Perl_runops_standard
      1.30  Perl_sv_free
      1.18  S_find_byclass
      1.17  Perl_save_alloc
```

Invocation: 400-perlbench-perfect.pl
```
    % time  name
      9.89  S_hv_fetch_common
      6.19  Perl_pp_padsv
      5.03  Perl_sv_setsv_flags
      3.64  Perl_pp_entersub
      3.39  Perl_pp_nextstate
      3.14  Perl_leave_scope
      2.82  Perl_gv_fetchpv
      2.65  Perl_pp_and
      2.64  Perl_pp_rv2av
      2.55  Perl_sv_clear
      2.35  Perl_pp_helem
      2.31  Perl_runops_standard
      1.96  Perl_sv_upgrade
      1.93  S_regmatch
      1.70  Perl_pp_aassign
      1.64  Perl_pp_pushmark
      1.63  Perl_amagic_call
      1.54  Perl_pp_ref
      1.42  S_method_common
      1.37  Perl_pp_aelemfast
```

Invocation: 400-perlbench-scrabbl.pl
```
    % time  name
      9.05  Perl_pp_gvsv
      7.82  Perl_sv_setsv_flags
      4.82  Perl_sv_eq
      4.59  Perl_pp_rv2av
      4.32  Perl_runops_standard
      3.66  Perl_sv_clear
      3.62  Perl_pp_gv
      3.33  Perl_pp_nextstate
      3.33  Perl_pp_aelem
      3.28  Perl_pp_and
      3.14  S_hv_fetch_common
      2.99  Perl_leave_scope
      2.25  Perl_sv_upgrade
      2.12  Perl_pp_entersub
      2.02  Perl_pp_enter
      2.00  Perl_pp_aassign
      1.92  Perl_pp_preinc
      1.84  Perl_sv_free
      1.77  Perl_pp_next
      1.76  Perl_pp_const
```

Invocation: 400-perlbench-splitmail.pl
```
 % time  name
  46.05  S_regmatch
   3.51  S_reginclass
   2.43  S_find_byclass
   2.42  Perl_leave_scope
   2.39  S_regtry
   2.36  Perl_pp_padsv
   2.14  Perl_sv_setsv_flags
   2.00  Perl_pp_match
   1.93  Perl_pp_gvsv
   1.62  Perl_pp_nextstate
   1.51  S_hv_fetch_common
   1.45  Perl_pp_and
   1.42  Perl_save_alloc
   1.40  Perl_pp_aelem
   1.29  Perl_regexec_flags
   1.17  Perl_pp_helem
   1.15  Perl_runops_standard
```

Invocation: 400-perlbench-suns.pl
```
 % time  name
  14.08  Perl_sv_cmp
   8.00  S_mergesortsv
   7.29  S_hv_fetch_common
   4.76  Perl_regexec_flags
   4.05  S_regmatch
   3.34  Perl_sv_setsv_flags
   2.74  Perl_pp_split
   2.63  Perl_sv_setpvn
   2.53  Perl_pp_gvsv
   2.33  S_regtry
   2.33  S_hsplit
   2.23  Perl_runops_standard
   2.03  Perl_pp_rv2av
   1.82  Perl_sv_clear
   1.72  Perl_sv_upgrade
   1.42  Perl_sv_catpvn_flags
   1.42  Perl_free_tmps
   1.32  Perl_pp_helem
   1.22  Perl_sv_catsv_flags
   1.22  Perl_pp_pushmark
```

401.bzip2

Invocation: 401-bzip2-byoudoin-5
```
 % time  name
  22.49  BZ2_compressBlock
  20.19  BZ2_blockSort
  15.87  mainGtU
  14.23  BZ2_decompress
  12.48  BZ2_bzDecompress
  11.70  fallbackSort
   1.92  handle_compress
```

Invocation: 401-bzip2-input.combined-80
```
 % time  name
  37.28  BZ2_blockSort
  14.12  mainGtU
  12.76  BZ2_bzDecompress
  10.99  BZ2_compressBlock
  10.94  BZ2_decompress
   9.50  fallbackSort
   2.81  handle_compress
```

Invocation: 401-bzip2-input.program-10
```
 % time  name
  35.70  BZ2_blockSort
  19.32  BZ2_compressBlock
  18.67  BZ2_decompress
  11.51  BZ2_bzDecompress
   8.47  mainGtU
   3.77  handle_compress
   1.14  bsW
```

403.gcc

Invocation: 403-gcc-integrate
```
 % time  name
   5.46  init_alias_analysis
   3.01  bitmap_operation
   2.77  propagate_one_insn
   2.58  reg_is_remote_constant_p
   2.20  ggc_mark_rtx_children_1
   2.03  approx_reg_cost
   1.91  note_stores
   1.70  for_each_rtx
   1.66  cse_insn
   1.41  ggc_set_mark
   1.38  sbitmap_vector_alloc
   1.31  mark_set_1
   1.29  reg_scan
   1.26  find_basic_blocks
   1.20  reg_scan_mark_refs
   1.16  bitmap_set_bit
   1.12  propagate_block
   1.12  mark_used_regs
   1.09  find_reg_note
   1.03  convert_to_ssa
```

429.mcf

Invocation: 429-mcf-inp.in
```
 % time  name
  56.91  price_out_impl
  27.45  primal_bea_mpp
   6.12  refresh_potential
   4.38  replace_weaker_arc
   1.32  update_tree
```

445.gobmk

Invocation: 445-gobmk-arb.tst
```
 % time  name
   8.68  do_play_move
   6.27  fastlib
   5.52  order_moves
   5.12  incremental_order_moves
   4.97  dfa_matchpat_loop
   3.71  assimilate_string
   3.48  approxlib
   3.44  do_find_defense
   3.24  hashtable_search
   3.09  popgo
   2.96  is_self_atari
   2.88  matchpat_loop
   2.77  hashtable_clear
   2.60  compute_connection_distances
   2.59  do_attack
   2.41  update_liberties
   2.38  chainlinks2
   2.24  do_get_read_result
   1.96  do_find_superstring
   1.80  count_common_libs
```

Invocation: 445-gobmk-arend.tst
```
 % time  name
  12.76  do_play_move
   6.66  fastlib
   6.62  compute_connection_distances
   5.07  incremental_order_moves
   5.01  order_moves
   4.10  dfa_matchpat_loop
   3.73  popgo
   3.65  assimilate_string
   2.93  hashtable_search
   2.81  approxlib
   2.33  get_next_move_from_list
   2.24  do_find_defense
   2.10  is_self_atari
   2.06  accumulate_influence
   1.91  do_attack
   1.86  chainlinks2
   1.70  do_get_read_result
   1.63  remove_liberty
   1.53  is_suicide
   1.51  count_common_libs
```

Invocation: 445-gobmk-arion.tst
```
 % time  name
   9.87  do_play_move
   6.43  fastlib
   5.28  order_moves
   4.99  dfa_matchpat_loop
   4.86  compute_connection_distances
   4.77  incremental_order_moves
   3.89  hashtable_search
   3.32  assimilate_string
   3.11  approxlib
   3.04  popgo
   2.98  do_find_defense
   2.94  accumulate_influence
   2.40  hashtable_partially_clear
   2.38  do_attack
   2.24  is_self_atari
   2.22  chainlinks2
```

```
    2.00  do_get_read_result
    1.86  update_liberties
    1.82  matchpat_loop
    1.71  count_common_libs
```

Invocation: 445-gobmk-atari.tst

```
  % time  name
    8.36  do_play_move
    6.16  fastlib
    5.67  hashtable_clear
    5.07  dfa_matchpat_loop
    4.60  order_moves
    4.58  incremental_order_moves
    3.80  matchpat_loop
    3.71  assimilate_string
    3.51  hashtable_search
    3.02  do_attack
    3.00  do_find_defense
    2.91  approxlib
    2.91  is_self_atari
    2.91  popgo
    2.49  do_get_read_result
    2.22  update_liberties
    2.13  chainlinks2
    1.73  propose_edge_moves
    1.62  accumulate_influence
    1.58  count_common_libs
```

Invocation: 445-gobmk-blunder.tst

```
  % time  name
   16.44  hashtable_clear
    9.48  matchpat_loop
    6.38  do_play_move
    4.68  fastlib
    3.66  order_moves
    3.51  incremental_order_moves
    3.48  dfa_matchpat_loop
    2.78  assimilate_string
    2.72  do_find_defense
    2.66  popgo
    2.40  do_attack
    2.31  accumulate_influence
    2.17  do_get_read_result
    2.14  approxlib
    2.14  compute_connection_distances
    2.11  hashtable_search
    1.99  is_self_atari
    1.70  update_liberties
    1.58  store_persistent_reading_cache
    1.35  count_common_libs
```

Invocation: 445-gobmk-buzco.tst

```
  % time  name
    9.86  do_play_move
    6.80  fastlib
    5.49  order_moves
    5.30  incremental_order_moves
    4.55  dfa_matchpat_loop
    3.97  hashtable_search
    3.43  assimilate_string
    3.17  popgo
    3.16  approxlib
    3.08  accumulate_influence
    2.75  do_find_defense
    2.64  compute_connection_distances
    2.62  do_attack
    2.33  chainlinks2
    2.22  is_self_atari
    2.21  hashtable_partially_clear
    2.17  do_get_read_result
    1.79  update_liberties
    1.67  count_common_libs
    1.45  matchpat_loop
```

Invocation: 445-gobmk-nicklas2.tst

```
  % time  name
    7.90  do_play_move
    6.35  fastlib
    5.53  order_moves
    5.47  incremental_order_moves
    4.29  hashtable_clear
    4.22  hashtable_search
    3.87  approxlib
    3.61  assimilate_string
    3.55  do_find_defense
    3.30  do_attack
    3.19  dfa_matchpat_loop
    2.97  popgo
    2.84  do_get_read_result
    2.76  is_self_atari
    2.35  chainlinks2
    2.33  update_liberties
    2.27  propose_edge_moves
    1.96  count_common_libs
    1.53  do_find_superstring
    1.49  get_next_move_from_list
```

Invocation: 445-gobmk-nicklas4.tst

```
  % time  name
    9.84  do_play_move
    9.29  dfa_matchpat_loop
    6.23  fastlib
    4.79  order_moves
    4.61  incremental_order_moves
    4.05  compute_connection_distances
    3.55  hashtable_search
    3.42  assimilate_string
    3.07  popgo
    2.63  approxlib
    2.57  do_find_defense
    2.21  do_attack
    2.02  is_self_atari
    1.94  do_get_read_result
    1.87  chainlinks2
    1.80  matchpat_loop
    1.79  accumulate_influence
    1.76  get_next_move_from_list
    1.68  compute_primary_domains
    1.54  update_liberties
```

456.hmmer

Invocation: 456-hmmer-leng100

```
  % time  name
   95.75  P7Viterbi
    1.94  sre_random
    1.74  FChoose
```

458.sjeng

Invocation: 458-sjeng-train

```
  % time  name
    20.1  std_eval
    8.04  gen
    6.52  setup_attackers
    5.15  QProbeTT
    5.01  remove_one
    4.32  search
    4.23  ProbeTT
    4.12  Pawn
    4.08  order_moves
    3.60  checkECache
    3.59  push_slidE
    3.58  rook_mobility
    3.20  bishop_mobility
    3.04  is_attacked
    2.64  make
    1.98  Rook
    1.96  unmake
    1.67  check_legal
    1.42  see
    1.39  Bishop
```

462.libquantum

Invocation: 462-libquantum-143-25

```
  % time  name
   59.15  quantum_toffoli
   15.33  quantum_sigma_x
   13.84  quantum_cnot
    3.95  quantum_swaptheleads
    3.79  __umoddi3
    2.54  quantum_state_collapse
```

464.h264ref

Invocation: 464-h264ref-foreman-train

```
  % time  name
   25.96  SetupFastFullPelSearch
   21.54  FastFullPelBlockMotionSearch
   11.90  SATD
    6.84  FastPelY_14
    6.67  SubPelBlockMotionSearch
    3.68  SetupLargerBlocks
    3.54  dct_luma
    3.23  UMVLine16Y_11
    2.58  FastLine16Y_11
    1.25  BlockMotionSearch
    1.20  UMVPelY_14
    1.00  getNonAffNeighbour
```

471.omnetpp

Invocation: 471-omnetpp-train
```
 % time  name
  21.92  cMessageHeap::shiftup
   8.87  cMessageHeap::insert
   4.76  cObject::setOwner
   4.30  cModule::findGate
   3.52  cGate::deliver
   3.20  TCmdenvApp::simulate
   2.82  EtherMAC::printState
   1.94  opp_strdup
   1.86  EtherMAC::processMsgFromNetwork
   1.70  EtherMAC::handleMessage
   1.52  EtherBus::handleMessage
   1.50  cSimpleModule::scheduleAt
   1.46  cMessage::operator=
   1.42  EtherMAC::processReceivedDataFrame
   1.39  cSimulation::doOneEvent
   1.36  cSimulation::selectNextModule
   1.29  cObject::~cObject
   1.23  cArray::get
   1.21  cMessageHeap::getFirst
   1.17  TOmnetApp::checkTimeLimits
```

473.astar

Invocation: 473-astar-Biglakes
```
 % time  name
  35.27  wayobj::makebound2
  15.23  way2obj::releasepoint
  12.17  regwayobj::isaddtobound
  11.86  regwayobj::makebound2
  10.85  regmngobj::getregfillnum
   2.88  way2obj::releasebound
   2.42  way2obj::isaddtobound
   2.19  way2obj::addtobound
   1.69  regboundobj::makebound2
   1.18  regmngobj::defineneighborhood1
```

Invocation: 473-astar-rivers1
```
 % time  name
  39.97  wayobj::makebound2
  23.57  way2obj::releasepoint
  11.11  regwayobj::makebound2
   6.30  regwayobj::isaddtobound
   5.73  regmngobj::getregfillnum
   4.66  way2obj::releasebound
   3.60  way2obj::addtobound
   2.82  way2obj::isaddtobound
```

483.xalan

Invocation: 483-xalan-train
```
 % time  name
   4.41  xalanc_1_8::VariablesStack::findEntry
   3.52  xalanc_1_8::FunctionSubstring::execute
   3.37  xalanc_1_8::XPath::executeMore
   3.23  xalanc_1_8::XalanDOMString::equals
   3.04  xalanc_1_8::XalanDOMString::equals
   2.90  xalanc_1_8::XalanReferenceCountedObject::addRefer
         ence
   2.53  xalanc_1_8::XalanReferenceCountedObject::removeRe
         ference
   2.47  xalanc_1_8::XalanBitmap::isSet
   2.41  xalanc_1_8::DoubleSupport::round
   2.30  xalanc_1_8::XPath::runFunction
   2.13  xalanc_1_8::XObjectFactoryDefault::doReturnObject
   2.02  __gnu_cxx::__normal_iterator
   1.66  xalanc_1_8::StylesheetExecutionContextDefault::ge
         tParams
   1.43  __gnu_cxx::__normal_iterator
   1.43  xalanc_1_8::ElemTemplateElement::executeChildren
   1.25  xalanc_1_8::VariablesStack::findXObject
   1.24  xalanc_1_8::XStringCachedAllocator::createString
   1.21  xalanc_1_8::XPath::variable
   1.18  xalanc_1_8::ReusableArenaBlock::allocateBlock()
   1.16  xalanc_1_8::XalanDOMString::erase
```

REF workload

400.perlbench

Invocation: 400-perlbench-checkspam.pl
```
 % time  name
  30.23  S_regmatch
  12.33  S_find_byclass
   4.99  S_regtry
   3.78  S_hv_fetch_common
   2.81  Perl_pp_entersub
   2.70  Perl_leave_scope
   2.62  Perl_pp_padsv
   2.45  Perl_pp_helem
   2.22  Perl_pp_nextstate
   2.18  Perl_sv_setsv_flags
   1.77  Perl_pp_rv2hv
   1.52  Perl_pp_match
   1.45  Perl_save_alloc
   1.22  Perl_pp_and
   1.13  Perl_regexec_flags
   1.01  Perl_fbm_instr
   1.01  S_share_hek_flags
```

Invocation: 400-perlbench-diffmail.pl
```
 % time  name
  17.56  S_regmatch
   6.02  Perl_sv_setsv_flags
   5.75  Perl_regexec_flags
   4.22  Perl_sv_clear
   4.06  Perl_sv_upgrade
   2.91  Perl_pp_padsv
   2.67  Perl_leave_scope
   2.54  Perl_sv_free
   2.51  S_regtry
   2.45  S_hv_fetch_common
   2.32  Perl_sv_grow
   2.01  S_regrepeat
   2.00  Perl_pp_nextstate
   1.97  Perl_sv_setpvn
   1.86  Perl_pp_split
   1.66  Perl_pp_entersub
   1.62  Perl_free_tmps
   1.48  Perl_pp_helem
   1.40  S_find_byclass
   1.36  Perl_save_alloc
```

Invocation: 400-perlbench-splitmail.pl
```
 % time  name
  53.98  S_regmatch
   4.19  S_reginclass
   2.95  S_regtry
   2.87  S_find_byclass
   2.47  Perl_leave_scope
   1.74  Perl_save_alloc
   1.65  Perl_pp_gvsv
   1.58  Perl_pp_match
   1.48  Perl_sv_setsv_flags
   1.48  Perl_pp_padsv
   1.40  Perl_pp_aelem
   1.16  Perl_regexec_flags
   1.15  S_hv_fetch_common
   1.14  Perl_pp_and
   1.11  Perl_pp_nextstate
   1.07  MD5Transform
```

401.bzip2

Invocation: 401-bzip2-chicken-30
```
 % time  name
  43.84  fallbackSort
  14.26  BZ2_compressBlock
  12.75  mainGtU
  10.46  BZ2_blockSort
   8.93  BZ2_decompress
   7.82  BZ2_bzDecompress
   1.21  handle_compress
```

Invocation: 401-bzip2-input.combined-200
```
 % time  name
  37.90  BZ2_blockSort
  13.82  mainGtU
  13.12  BZ2_bzDecompress
  11.26  BZ2_decompress
  11.14  BZ2_compressBlock
   8.57  fallbackSort
   2.51  handle_compress
```

Invocation: 401-bzip2-input.program-280
```
  % time  name
   35.91  BZ2_blockSort
   19.69  BZ2_compressBlock
   18.98  BZ2_decompress
    10.9  BZ2_bzDecompress
    8.17  mainGtU
    3.60  handle_compress
    1.21  bsW
```
Invocation: 401-bzip2-input.source-280
```
  % time  name
   43.32  BZ2_blockSort
   14.26  BZ2_bzDecompress
   14.17  mainGtU
   11.81  BZ2_decompress
   10.91  BZ2_compressBlock
    2.77  handle_compress
```
Invocation: 401-bzip2-liberty-30
```
  % time  name
   63.36  fallbackSort
   20.50  mainGtU
    4.69  BZ2_compressBlock
    4.16  BZ2_bzDecompress
    3.77  BZ2_decompress
    2.59  BZ2_blockSort
```
Invocation: 401-bzip2-text-280
```
  % time  name
   39.13  BZ2_blockSort
   38.02  mainGtU
    7.30  BZ2_bzDecompress
    6.58  BZ2_decompress
    5.60  BZ2_compressBlock
    2.17  handle_compress
```

403.gcc

Invocation: 403-gcc-166
```
  % time  name
   27.08  reg_is_remote_constant_p
    7.14  compute_transp
    5.85  bitmap_operation
    5.81  clear_table
    2.96  single_set_2
    2.87  sbitmap_union_of_diff
    1.88  delete_null_pointer_checks
    1.65  init_alias_analysis
    1.50  loop_regs_scan
    1.36  sbitmap_vector_alloc
    1.25  splay_tree_splay_helper
    1.08  sbitmap_intersection_of_preds
    1.01  canon_rtx
```
Invocation: 403-gcc-200
```
  % time  name
    5.64  bitmap_operation
    5.46  ggc_mark_rtx_children_1
    4.10  ggc_set_mark
    4.03  reg_is_remote_constant_p
    3.64  init_alias_analysis
    2.29  loop_regs_scan
    2.20  ggc_mark_rtx_children
    1.87  note_stores
    1.62  try_combine
    1.46  propagate_one_insn
    1.37  cse_insn
    1.37  compute_transp
    1.30  ggc_pop_context
    1.26  reg_scan_mark_refs
    1.20  for_each_rtx
    1.19  htab_traverse
    1.10  approx_reg_cost
    1.09  clear_table
    1.09  ggc_alloc
    1.08  convert_to_ssa
```
Invocation: 403-gcc-cp-decl
```
  % time  name
   12.56  compute_transp
    6.50  bitmap_operation
    5.84  clear_table
    3.75  canon_rtx
    3.31  delete_null_pointer_checks
    2.89  find_base_term
    2.65  sbitmap_union_of_diff
    2.48  mems_in_disjoint_alias_sets_p
    2.16  ix86_find_base_term
    1.94  expunge_block
    1.72  init_alias_analysis
    1.71  nonoverlapping_memrefs_p
    1.62  ggc_mark_rtx_children_1
    1.61  sbitmap_vector_alloc
    1.50  compute_dominance_frontiers_1
    1.39  htab_traverse
    1.36  sbitmap_zero
    1.35  try_combine
    1.23  propagate_one_insn
    1.16  ggc_set_mark
```
Invocation: 403-gcc-expr2
```
  % time name
   18.09 clear_table
    9.29 compute_transp
    6.18 bitmap_operation
    5.88 loop_regs_scan
    4.33 delete_null_pointer_checks
    3.08 expunge_block
    2.46 sbitmap_union_of_diff
    2.10 htab_traverse
    1.72 sbitmap_vector_alloc
    1.51 init_alias_analysis
    1.38 sbitmap_zero
    1.29 compute_dominance_frontiers_1
    1.22 reg_is_remote_constant_p
    1.22 in_expr_list_p
    1.01 mems_in_disjoint_alias_sets_p
```
Invocation: 403-gcc-expr
```
  % time name
   19.03 clear_table
   10.77 compute_transp
    6.12 bitmap_operation
    4.34 loop_regs_scan
    4.13 delete_null_pointer_checks
    3.15 htab_traverse
    3.09 expunge_block
    2.59 sbitmap_union_of_diff
    1.59 sbitmap_vector_alloc
    1.49 compute_dominance_frontiers_1
    1.43 mems_in_disjoint_alias_sets_p
    1.42 init_alias_analysis
    1.34 sbitmap_zero
    1.25 scan_loop
    1.20 canon_rtx
```
Invocation: 403-gcc-g23
```
  % time name
   24.92 reg_is_remote_constant_p
   11.68 htab_traverse
   10.35 clear_table
    4.90 bitmap_operation
    4.43 delete_null_pointer_checks
    4.26 fixup_var_refs_insns
    3.58 fixup_var_refs_1
    3.13 single_set_2
    2.57 expunge_block
    2.13 fixup_var_refs_insn
    1.89 htab_empty
    1.79 loop_regs_scan
    1.77 sbitmap_union_of_diff
    1.74 try_combine
    1.08 compute_dominance_frontiers_1
```
Invocation: 403-gcc-s04
```
  % time name
   22.59 clear_table
   12.17 loop_regs_scan
   10.04 reg_is_remote_constant_p
    7.09 bitmap_operation
    6.51 compute_transp
    4.42 delete_null_pointer_checks
    2.45 sbitmap_union_of_diff
    1.43 sbitmap_vector_alloc
    1.36 find_base_term
    1.33 expunge_block
    1.31 single_set_2
    1.21 mems_in_disjoint_alias_sets_p
    1.20 sbitmap_zero
    1.18 canon_rtx
```
Invocation: 403-gcc-scilab
```
  % time name
    5.49 ggc_mark_rtx_children_1
    4.46 ggc_set_mark
    4.40 init_alias_analysis
    2.53 ggc_mark_rtx_children
    2.34 bitmap_operation
    2.30 ggc_pop_context
    2.21 propagate_one_insn
    1.91 cse_insn
    1.90 note_stores
    1.81 approx_reg_cost
    1.68 for_each_rtx
    1.62 constrain_operands
    1.36 ggc_mark_trees
    1.22 reg_scan_mark_refs
    1.20 ggc_alloc
    1.13 find_reloads
    1.04 reg_is_remote_constant_p
    1.00 bitmap_set_bit
```

Invocation: 403-gcc-typeck

```
   % time name
   17.94 clear_table
   15.10 loop_regs_scan
    5.69 compute_transp
    5.56 bitmap_operation
    2.80 sbitmap_union_of_diff
    2.75 expunge_block
    2.69 delete_null_pointer_checks
    1.50 init_alias_analysis
    1.46 reg_is_remote_constant_p
    1.34 sbitmap_vector_alloc
    1.21 scan_loop
    1.18 compute_dominance_frontiers_1
    1.11 sbitmap_zero
```

429.mcf

Invocation: 429-mcf-inp.in

```
   % time name
   20.53 refresh_potential
   19.94 price_out_impl
   18.68 primal_bea_mpp
   10.64 update_tree
    9.82 replace_weaker_arc
    5.59 bea_is_dual_infeasible
    4.45 primal_iminus
    2.76 sort_basket
    2.15 insert_new_arc
    1.29 write_circulations
    1.14 dual_feasible
    1.01 suspend_impl
```

445.gobmk

Invocation: 445-gobmk-13x13.tst

```
   % time name
   10.14 do_play_move
    6.87 dfa_matchpat_loop
    6.25 fastlib
    5.06 incremental_order_moves
    5.06 order_moves
    3.55 compute_connection_distances
    3.34 hashtable_search
    3.31 assimilate_string
    3.16 popgo
    3.14 get_next_move_from_list
    2.80 approxlib
    2.61 do_find_defense
    2.36 do_attack
    2.32 is_self_atari
    1.98 do_get_read_result
    1.90 chainlinks2
    1.77 compute_primary_domains
    1.64 update_liberties
    1.52 count_common_libs
    1.48 find_persistent_reading_cache_entry
```

Invocation: 445-gobmk-nngs.tst

```
   % time name
     10.8 do_play_move
     6.68 fastlib
     5.24 dfa_matchpat_loop
     5.17 order_moves
     5.15 incremental_order_moves
     4.06 compute_connection_distances
     3.46 hashtable_search
     3.41 assimilate_string
     3.32 popgo
     2.96 approxlib
     2.74 accumulate_influence
     2.72 do_find_defense
     2.43 do_attack
     2.21 is_self_atari
     2.15 chainlinks2
     2.01 do_get_read_result
     1.64 update_liberties
     1.59 count_common_libs
     1.55 matchpat_loop
     1.40 hashtable_partially_clear
```

Invocation: 445-gobmk-score2.tst

```
   % time name
   13.59 matchpat_loop
    8.77 hashtable_clear
    7.30 do_play_move
    5.06 dfa_matchpat_loop
    5.05 accumulate_influence
    4.99 compute_connection_distances
    4.10 fastlib
    3.14 incremental_order_moves
    3.10 order_moves
    2.41 assimilate_string
    2.28 popgo
    2.14 update_liberties
    1.90 approxlib
    1.79 hashtable_search
    1.60 do_find_defense
    1.49 is_self_atari
    1.47 do_attack
    1.27 do_get_read_result
    1.12 chainlinks2
    1.07 count_common_libs
```

Invocation: 445-gobmk-trevorc.tst

```
   % time name
   10.98 do_play_move
    6.54 fastlib
    5.37 incremental_order_moves
    5.15 order_moves
    4.96 compute_connection_distances
    4.20 dfa_matchpat_loop
    3.65 assimilate_string
    3.44 popgo
    3.32 hashtable_search
    3.01 approxlib
    2.68 do_find_defense
    2.45 do_attack
    2.37 is_self_atari
    2.10 do_get_read_result
    2.07 hashtable_clear
    1.90 chainlinks2
    1.85 get_next_move_from_list
    1.76 update_liberties
    1.64 count_common_libs
    1.50 is_suicide
```

Invocation: 445-gobmk-trevord.tst

```
   % time name
   12.75 do_play_move
    7.85 compute_connection_distances
    6.63 fastlib
    4.97 incremental_order_moves
    4.93 order_moves
    3.83 assimilate_string
    3.75 popgo
    3.52 dfa_matchpat_loop
    3.04 accumulate_influence
    2.82 approxlib
    2.60 hashtable_search
    2.27 matchpat_loop
    2.10 do_find_defense
    1.98 is_self_atari
    1.76 chainlinks2
    1.75 do_attack
    1.70 remove_liberty
    1.60 is_suicide
    1.55 do_get_read_result
    1.55 count_common_libs
```

456.hmmer

Invocation: 456-hmmer-retro

```
   % time name
   95.19 P7Viterbi
    2.24 sre_random
    1.96 FChoose
```

Invocation: 456-hmmer-swiss41

```
   % time name
   98.97 P7Viterbi
```

458.sjeng

Invocation: 458-sjeng-ref
```
    % time name
    19.57 std_eval
     8.10 setup_attackers
     8.01 gen
     5.56 remove_one
     4.94 order_moves
     4.74 search
     4.17 is_attacked
     4.10 push_slidE
     4.00 rook_mobility
     3.67 Pawn
     3.41 checkECache
     3.37 make
     3.12 ProbeTT
     2.91 bishop_mobility
     2.47 unmake
     2.02 check_legal
     1.84 see
     1.51 qsearch
     1.47 Rook
     1.07 f_in_check
```

462.libquatum

Invocation: 462-libquantum-1397-8
```
    % time name
    37.96 quantum_toffoli
    21.41 quantum_cnot
    21.02 quantum_sigma_x
     8.98 quantum_swaptheleads
     5.53 __umoddi3
     2.26 quantum_state_collapse
     1.47 quantum_gate1
```

464.h264ref

Invocation: 464-h264ref-foreman-ref1
```
    % time name
    25.69 SetupFastFullPelSearch
    21.43 FastFullPelBlockMotionSearch
    11.94 SATD
     6.99 FastPelY_14
     6.64 SubPelBlockMotionSearch
     3.73 SetupLargerBlocks
     3.60 dct_luma
     3.28 UMVLine16Y_11
     2.51 FastLine16Y_11
     1.27 UMVPelY_14
     1.18 BlockMotionSearch
```

Invocation: 464-h264ref-foreman-ref2
```
    % time name
    22.02 SetupFastFullPelSearch
    13.47 FastFullPelBlockMotionSearch
    10.47 SATD
     6.93 dct_luma
     5.65 SubPelBlockMotionSearch
     5.32 FastPelY_14
     4.20 biari_encode_symbol
     3.03 SetupLargerBlocks
     2.86 UMVLine16Y_11
     2.32 FastLine16Y_11
     1.56 getNonAffNeighbour
     1.43 Mode_Decision_for_4x4IntraBlocks
     1.26 OneComponentChromaPrediction4x4
     1.15 UMVPelY_14
     1.08 dct_chroma
```

Invocation: 464-h264ref-sss-ref
```
    % time name
    26.36 SetupFastFullPelSearch
    10.58 FastFullPelBlockMotionSearch
    10.14 SATD
     7.63 dct_luma
     6.29 FastPelY_14
     5.71 SubPelBlockMotionSearch
     3.34 SetupLargerBlocks
     2.67 FastLine16Y_11
     1.88 biari_encode_symbol
     1.76 getNonAffNeighbour
     1.69 Mode_Decision_for_4x4IntraBlocks
     1.37 dct_chroma
     1.34 OneComponentChromaPrediction4x4
     1.25 UMVLine16Y_11
     1.14 BlockMotionSearch
     1.13 store_coding_state
     1.05 get_mb_block_pos
     1.03 RDCost_for_4x4IntraBlocks
     1.00 find_sad_16x16
```

471.omnetpp

Invocation: 471-omnetpp-ref
```
    % time name
    19.52 cMessageHeap::shiftup
     7.58 cGate::deliver
     4.66 cSimulation::selectNextModule
     4.41 cModule::findGate
     3.95 cObject::setOwner
     3.94 EtherMAC::handleMessage
     3.85 cOutVector::record
     3.70 cSubModIterator::operator++
     3.31 cFileOutputVectorManager::record
     2.93 cMessageHeap::insert
     2.54 cArray::get
     2.54 cSimulation::setContextModule
     2.25 cMessage::operator=
     2.18 cSimpleChannel::deliver
     2.06 cObject::~cObject
     1.89 EtherMAC::printState
     1.50 cQueue::remove_qelem
     1.35 cMessageHeap::getFirst
     1.29 cSimpleModule::arrived
     1.14 cSimpleModule::sendDelayed
```

473.astar

Invocation: 473-astar-Biglakes
```
    % time name
    25.63 way2obj::releasepoint
    24.87 wayobj::makebound2
    12.12 regwayobj::isaddtobound
    11.02 regwayobj::makebound2
     9.44 regmngobj::getregfillnum
     4.73 way2obj::releasebound
     4.72 way2obj::isaddtobound
     2.93 way2obj::addtobound
```

Invocation: 473-astar-rivers
```
    % time name
    40.64 wayobj::makebound2
    25.82 way2obj::releasepoint
     9.92 regwayobj::makebound2
     9.23 regmngobj::getregfillnum
     5.10 way2obj::releasebound
     3.69 way2obj::addtobound
     2.89 way2obj::isaddtobound
     1.61 regwayobj::isaddtobound
```

483.xalan

Invocation: 483-xalan-ref
```
    % time name
    10.70 __gnu_cxx::__normal_iterator
     9.59 xercesc_2_5::ValueStore::isDuplicateOf
     9.50 xercesc_2_5::ValueStore::contains
     8.01 xalanc_1_8::XStringCachedAllocator::destroy
     6.32 xercesc_2_5::BaseRefVectorOf::elementAt
     4.98 xercesc_2_5::ValueVectorOf::size
     4.54 xalanc_1_8::ReusableArenaBlock::ownsObject
     3.85 xercesc_2_5::ValueVectorOf::elementAt
     2.84 xercesc_2_5::NameDatatypeValidator::compare
     1.90 xalanc_1_8::XStringCachedAllocator::createString
     1.63 xercesc_2_5::BaseRefVectorOf::elementAt
     1.54 xalanc_1_8::ReusableArenaBlock::blockAvailable
     1.52 xalanc_1_8::XalanDOMStringCache::release
     1.40 Xalanc_1_8::VariablesStack::findEntry
     1.34 xalanc_1_8::FunctionSubstring::execute
     1.20 Xalanc_1_8::XalanDOMString::equals
     1.14 xalanc_1_8::XPath::executeMore
     1.10 xalanc_1_8::XalanReferenceCountedObject::removeRef
          erence
     1.06 xalanc_1_8::ElemTemplateElement::executeChildren
```