White-Box Benchmarking

Emilio Hernández[1] and Tony Hey[2]

[1] Departamento de Computación,
Universidad Simón Bolívar, Apartado 89000, Caracas, Venezuela,
emilio@usb.ve
[2] Department of Electronics and Computer Science,
University of Southampton, Southampton SO17 1BJ, UK
ajgh@ecs.soton.ac.uk

Abstract. Structural performance analysis of the NAS parallel benchmarks is used to time code sections and specific classes of activity, such as communication or data movements. This technique is called *white-box benchmarking* because, similarly to white-box methodologies used in program testing, the programs are not treated as black boxes. The timing methodology is portable, which is indispensable to make comparative benchmarking across different computer systems. A combination of conditional compilation and code instrumentation is used to measure execution time related to different aspects of application performance. This benchmarking methodology is proposed to help understand parallel application behaviour on distributed-memory parallel platforms.

1 Introduction

Computer evaluation methodologies based on multi-layered benchmark suites, like Genesis [1], EuroBen [2] and Parkbench [3] have been proposed. The layered approach is used for characterising the performance of complex benchmarks based on the performance models inferred from the execution of low-level benchmarks. However, it is not straightforward to relate the performance of the low-level benchmarks with the performance of more complex benchmarks.

The study of the relationship between the performance of a whole program and the performance of its components may help establish the relevance of using low-level benchmarks to characterise the performance of more complex benchmarks. For this reason we propose the structural performance analysis of complex benchmarks. A benchmarking methodology, called *white-box benchmarking*, is proposed to help understand parallel application performance. The proposed methodology differs from standard profiling in that it is not procedure oriented. Partial execution times are not only associated to code sections but also to activity classes, such as communication or data movements. These execution times may be compared to the results of simpler benchmarks in order to assess their predictive properties. The proposed methodology is portable. It only relies on MPI_WTIME (the MPI [4] timing function) and the availability of a source code preprocessor for conditional compilation, for instance, a standard C preprocessor.

Timing Method. The proposed timing method is simple enough to be portable. It is based on the combination of two basic techniques: *incremental conditional compilation* and *section timing*. Incremental conditional compilation consists of selecting code fragments from the original benchmark to form several kernels of the original benchmark. A basic kernel of a parallel benchmark can be built by selecting the communication skeleton. A second communication kernel can contain the communication skeleton plus data movements related to communication (e.g. data transfers to communication buffers). By measuring the elapsed time of both kernels, we know the total communication time (the time measured for the first kernel) and the time spent in data movements (the difference in the execution time of both benchmarks). The net computation time can be obtained by subtracting the execution time of the second kernel from the execution time of the complete benchmark. Not every program is suitable for this type of code isolation, see [5] for a more detailed discussion. Section timing is used on code fragments that take a relatively long time to execute, for example, the subroutines at the higher level of the call tree. Three executable files may be produced, the communication kernel, the communication plus data movements kernel and the whole benchmark. Optionally, if information by code section is required, an additional compile-time constant has to be set to include the timing functions that obtain the partial times. The use of MPI_WTIME allows us to achieve code portability. Two out of these three benchmark executions are usually fast because they will not execute "real" computation, but only communication and data movements. Section 2 presents an example of the use of the methodology with the NAS parallel Benchmarks [6]. In section 3 we present our conclusions.

2 Case Study with NAS Parallel Benchmarks

The NAS Parallel Benchmarks [6] are a widely recognized suite of benchmarks derived from important classes of aerophysics applications. In this work we used the application benchmarks (LU, BT and SP) and focused on the communication kernels extracted from these benchmarks. The main visible difference between these communication kernels comes from the fact that LU sends a larger number of short messages, while SP and BT send fewer and longer messages. This means that the LU communication kernel should benefit from low latency interconnect subsystems, while BT and SP communication kernels would execute faster on networks with a greater bandwidth.

2.1 Experiments

Several experiments with the instrumented version of the NAS benchmarks were conducted on a Cray T3D and a IBM SP2. For a description of the hardware and software configurations see [5]. Several experiments were conducted using the white-box methodology described here. These experiments are also described in detail in [5].

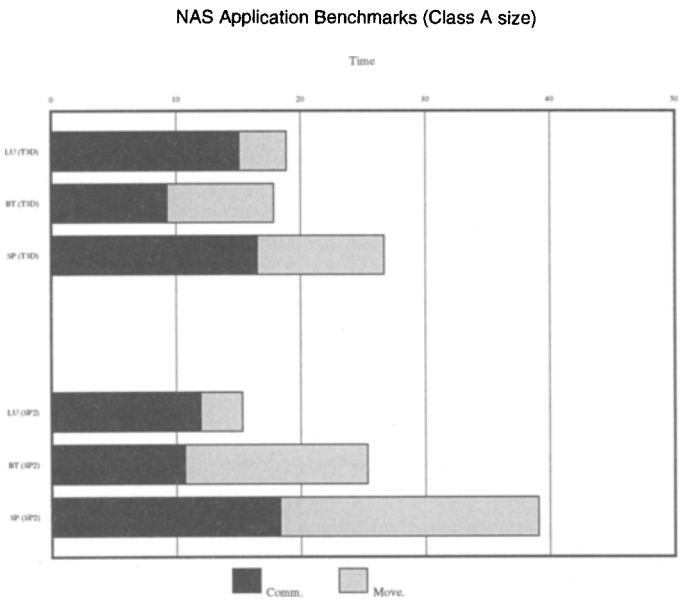

Fig. 1. Communication time and communication-related data movement time on T3D and SP2 (16 processors).

Figure 1 compares communication overhead in the T3D and the SP2 for LU, BT and SP. The communication kernel of LU runs marginally faster on the SP2 than on the T3D, while SP and BT communication kernels execute faster on the T3D. The execution of the communication kernels indicates that communication performance is not substantially better in the SP2 or the T3D. The main difference, in favour of the T3D, is the time spent in data movements related to communication, rather than communication itself.

Measurements made with COMMS1 (the Parkbench ping-pong benchmark), slightly modified for transmitting double precision elements, show that the T3D has a startup time equal to $104.359\mu sec$ and a bandwidth equal to 3.709 *Mdp/sec*, while the SP2 has a startup time equal to $209.458\mu sec$ and a bandwidth equal to 4.210 *Mdp/sec*, where *Mdb* means "millions of double precision elements".

As mentioned above, the LU communication kernel should run faster on low latency networks, while BT and SP communication kernels would execute faster on networks with a greater bandwidth. The observed behaviour of LU, BT and SP, seems to contradict the expected behaviour of these benchmarks, based on the COMMS1 results. Apart from bandwidth and startup time, many other factors may be playing an important role in communication performance, which are not measured by COMMS1. Some of these factors are network contention, the presence of collective communication functions, the fact that messages are sent from different memory locations, etc. In other words, it is clear from these experiments that communication performance may not be easily characterized

by low level communication parameters (latency and bandwidth) obtained from simpler benchmarks.

3 Conclusions

White-box benchmarking is a portable and comparatively effortless performance evaluation methodology. A few executions of each benchmark are necessary to get the information presented in this article, one for the complete benchmark and the rest for the extracted kernels. A benchmark visualisation interface like GBIS [7] may easily be enhanced to incorporate query and visualisation mechanisms to facilitate the presentation of results related to this methodology.

Useful information has been extracted from the NAS parallel benchmarks case study using white-box benchmarking. Communication-computation profiles of the NAS parallel benchmarks may easily indicate the balance between communication and computation time. Communication kernels obtained by isolating the communication skeleton of selected applications may give us a better idea about the strength of the communication subsystem. Additionally, some behaviour characteristics can be exposed using white-box benchmarking, like load balance and a basic execution profile.

This methodology may be used in benchmark design rather than in application development. Programs specifically developed as benchmarks may incorporate code for partial measurements and kernel selection. A description of the instrumented code sections may also be useful and, consequently, could be provided with the benchmarks. The diagnostic information provided by white-box benchmarking is useful to help understand the variable performance of parallel applications on different platforms.

References

1. C. A. Addison, V.S. Getov, A.J.G. Hey, R.W. Hockney, and I.C. Wolton. The genesis distributed-memory benchmarks. In J. Dongarra and W. Gentzsch, editors, *Computer Benchmarks*, pages 257–271. North-Holland, 1993.
2. A. van der Steen. Status and Direction of the EuroBen Benchmark. *Supercomputer*, 11(4):4–18, 1995.
3. J. Dongarra and T. Hey. The PARKBENCH Benchmark Collection. *Supercomputer*, 11(2-3):94–114, 1995.
4. Message Passing Interface Forum. The message passing interface standard. Technical report, Univeristy of Tennessee, Knoxville, USA, April 1994.
5. E. Hernández and T. Hey. White-Box Benchmarking (longer version). Available at `http://www.usb.ve/ emilio/WhiteBox.ps`, 1998.
6. D. Bailey *et. al.* The NAS Parallel Benchmarks. Technical Report RNR-94-007, NASA Ames Research Center, USA, March 1994.
7. M. Papiani, A.J.G. Hey, and R.W. Hockney. The Graphical Benchmark Information Service. *Scientific Programming*, 4(4), 1995.