The Development of Parkbench and Performance Prediction

Tony Hey and David Lancaster

Electronics & Computer Science
University of Southampton
Southampton SO17 1BJ, U.K.
(djl@ecs.soton.ac.uk)

Abstract

We present a brief overview of the development of benchmarks for parallel performance analysis and show how a set of widely accepted parallel benchmarks has emerged from the Parkbench initiative. At the lowest level, basic node parameters are captured by the LINPACK benchmark and the Genesis Communication benchmarks which give information on message-passing latency and bandwidth. At kernel and application level, results are widely available for the NAS Parallel Benchmarks. A new release of the Genesis communications benchmarks is described which addresses the important issue of the effect of memory hierarchy on message transfers.

The remainder of the paper is concerned with the way in which benchmark results can be used to predict performance of full applications. Three studies are considered: the PERFORM estimation tool and WHITEBOX benchmarking are research projects indicating some possible directions for progress, while INTREPID is the basis of a commercial system for scheduling meta-applications based on performance models for the components.

1 Introduction

Benchmarks have progressed a long way from simply generating single figures of merit for a kind of consumer guide to the "best computer". Nowadays there is strong need for performance prediction, for example for the scheduling necessary for the Information Power Grid (IPG)[1] and other distributed systems. An important requirement for such scheduling is the ablility to rapidly and reliably extrapolate the behavior of applications as their size is scaled up and they are run on larger systems. Such performance modeling needs measured machine parameters as input. These are the parameters that are ascertained by low-level benchmarks: for example, communication latencies and bandwidth curves are the kind of parameters vital in modeling any parallel process. Performance on modern machines is also strongly dependent on how well the code utilizes the memory hierarchy, and low-level benchmarks to measure parameters that describe the memory hierarchy will also be needed.

Much work has yet to be done in developing appropriate performance models, but here we present some of the benchmarking techniques that will be used to measure the input parameters of these models. We attempt to place this discussion in context by providing a short overview of the development of parallel benchmarking. We illustrate how the Parkbench initiative has adopted techniques learned from the historical development. Then, concentrating on the low-level class, we present a new release of Genesis low-level communications benchmarks that are intended to be used for precisely the purposes outlined above.

In the final section we discuss performance modeling, mainly concentrating how modeling is implemented in INTREPID[2], a present day scheduler for meta-applications. It is interesting to see the level of complexity required even with very simple models, and it is important to recognise the need for rapid scheduling decisions and for modeling without disclosure of the source code. Although results reported for INTREPID are encouraging, certain features such as the memory hierarchy are not treated well and we report on some research projects, PERFORM[3] and WHITEBOX[4] benchmarking, which attempt to do better.

2 Overview of Benchmark Development

We give a brief overview of the development of benchmarks for parallel performance analysis, emphasizing some of the lessons that can be learned for future work.

2.1 Single Processor Benchmarks

Whetstone: An early attempt to simulate a typical scientific workload using portable codes was the Whetstone benchmark[6] devised in 1972. The benchmarks were synthetic, consisting of artificial kernels intended to represent the computationally intensive part of certain scientific codes. The timing of this benchmark on different machines was compared and reported in terms of a unit called Whetstone instructions per second.

Livermore Loops: The Livermore Fortran Kernels [7] are a set of 24 loop kernels extracted from operational codes used at Lawrence Livermore National Labs. These became widely used and publicized the concept of the Mflop unit. For scientific codes, speed measured in floating point operations per second is useful, and has less dependence on the instruction set than simply measuring the total number of instructions emitted per second. The Livermore loops also assign a nominal flopcount to common floating point operations and allow measurements to be made in the absence of a hardware counter.

SPEC: In 1989 a set of machine vendors agreed on a set of real programs and run conditions to use as common benchmarks. The strong backing from vendors has been important in the success of SPEC, as has been the wide dissemination of results. The contents of the suite have evolved in time and presently SPEC95[8] contains 8 integer and 10 floating point codes. The precise content of the suite has been subject to some criticism and presently SPEC95 is regarded as more relevant to the workstations than supercomputers (though there are now benchmarks from the SPEC High Performance-Group[9]).

2.2 Simple Performance Modeling

Amdahl's law: Amdahl places an upper bound on the speedup (the ratio of the speed on P processors to the speed on one) of a code by decomposing the runtime into a fraction that can be parallelized efficiently and an inherently serial fraction. The maximum speedup occurs when the parallel part of the code dominates the runtime. However, Amdahl's assumption that the decomposition is fixed is now known to be incorrect. Typically the fraction of the runtime that is inherently serial depends on the problem size, and the best speedups are found for large codes. This law is therefore no longer considered a "show-stopper" for parallel computing, and indeed speedups in excess of 90% for a 1000 processor system have been observed with the IFS weatherforecasting code[5].

Hockney Parameters: The efficiency of computations on vector machines depends strongly on the vector length n. The simplest model for the computational time is linear assuming some fixed overhead. In terms of Hockney parameters[10] the runtime is given by $t = (n + n_{1/2})/r_\infty$. The parameter r_∞ measures the asymptotic performance and $n_{1/2}$ is the vector length needed to achieve half this performance. Performance models of this type have been used in attempts to understand kernel (for example FFT) performance[10], but the memory hierarchy in more recent cache based machines is more difficult to model. We need to find abstractions as simple and useful as Hockney's to understand and predict performance on parallel and distributed systems.

2.3 Parallel Benchmark Suites

Perfect Club: For this benchmark suite[11], 13 complete application codes running on shared memory vector systems were chosen. Unfortunately the complexity of the codes meant that it was difficult to infer conclusions about the architectures. It was also hard to quantify the effort that went into optimizing the codes for different platforms (though the benchmarks have been useful for other purposes[12]).

GENESIS: These distributed memory benchmarks[13] were based on an Esprit project starting in 1988. The codes used PARMACS message passing

and Fortran 77 and introduced the idea of a hierarchy of benchmark levels. A new version of these codes based on MPI is described later in this paper.

NAS: The original 1991 NAS benchmarks[14] were "paper and pencil", only defining the algorithms and leaving implementation up to the users. Several important algorithms used in the CFD work at NASA were abstracted to form 5 kernels (EP, MG, CG, FT, IS) and 3 simulated applications (LU, SP, BT). Fortran codes were later provided to clarify the algorithms, and following the NAS group's involvement with the Parkbench initiative, these have now been made available in both serial and parallel form. Results are collected and published on the NAS web site.

LINPACK: is a simple kernel benchmark[15] based on solving a dense system of linear equations. To accommodate improved processors, over time the size of the problem has increased from 100×100, and the test can also be used to measure how performance varies as a function of problem size. This code is quite straightforward, compilers finding it easy to optimize, and it is used to classify machines for the "Top500" list[16].

Euroben: was initially intended to provide a very detailed assessment of a single scalar or vector node and was then extended to shared memory parallel machines. The benchmark[17] is based on three modules that at the simplest level test basic operations, memory conflicts and bottlenecks and at higher levels test progressively more elaborate algorithms. The Euroben tests have recently been used for a very detailed analysis of the Origin 2000[18]

RAPS: The Real Applications on Parallel Systems[19] consortium is composed of both vendors and European scientific centers, mainly working on weather and climate modeling. The suite consists of large-scale application codes, either the commercial production codes themselves or benchmark versions. These codes are used as procurement benchmarks.

3 Benchmarking Techniques

As benchmarking techniques have developed, there has been a corresponding increase in the sophistication with which benchmarks are used. This is not always good, and there are many methods that can be used to inflate performance figures[20]. Indeed, in the early days these tricks gave both benchmarking and parallel computers a bad reputation, and it was to place it on a more firm and scientific footing that the Parkbench initiative was

conceived. This initiative has incorporated many valuable lessons from the development outlined above, for example the recognition that more than one parameter is needed to model performance. We have identified three aspects that seem particularly relevant at present: the benchmark content, the importance of the memory hierarchy and the need for wide dissemination of results. Below, we expand on the first two of these, the need for the third is self evident.

3.1 Benchmark Content

As has been apparent from the very earliest days of benchmarking, there is a major distinction between low-level synthetic benchmarks and larger tests that represent real production codes. Based on the work of Genesis, a hierarchical classification of benchmarks has been widely adopted. This hierarchy consists of:

- Low-Level codes: Measuring the basic machine properties; both of single nodes and of communications interconnections.
- Kernels: Computationally intensive kernels testing some widely used scientific algorithm.
- Compact Applications: Compact application codes including I/O, but designed to be run easily.

The compact application benchmarks attempt to capture the behavior of other features of production codes other than the computational kernel. For example: I/O and its interaction with computation. As such, these benchmarks are closer to real applications and can help identify bottlenecks that may be observed in full production codes. These codes can also provide useful support in the procurement process along with factors such as reliability and support, but in no way take over the role of the main procurement benchmarks which must be large production codes chosen appropriately for each site. RAPS is an example of how cooperation at this level can be beneficial to both procurement sites and vendors.

Low-level and kernel benchmarks are more finely honed instruments intended to test specific performance characteristics. These benchmarks do not play significant roles in procurement but are necessary for researchers

to understand the basic limitations of a given architecture. Although there are many similar benchmarks in the low-level class and there is rarely much leeway in how to write the essential core of any particular low-level test, it is important for researchers to know exactly what the test is doing and precisely how the measurements are taken. A common standard set of tests is therefore very desirable. It is vital that these low-level tests be immediately portable, that the precise significance of the timing measurements be clear and that the results be widely and publicly available.

3.2 Memory Hierarchy

With the retreat from vector architectures and the increasing use of commodity components in the High Performance market, the memory hierarchy has become ever more important. Single node performance is critically dependent on the compiler having a good understanding of the cache behavior. Although it has always been clear that benchmarks test the compiler as much as the architecture, the increasing complexity of the memory hierarchy and the subtlety of the code rearrangements needed have placed greater emphasis on the compiler than ever before.

These difficulties are quite apparent at the serial level and there are many techniques to write explicit codes that despite having higher overall instruction counts, have better performance due to improved cache usage. The blocking of matrix-matrix multiplication is a well known example.

At the parallel level the memory hierarchy continues to have a significant effect. Cache effects can easily be seen when communicating non-contiguous data. In fact, halo exchange of multidimensional data frequently occurs in many scientific applications and if the cache is not used appropriately significant under-performance can result. In more recent implementations of MPI, this problem can be addressed automatically using the derived data types.

4 Parkbench

The Parkbench initiative of 1992 was an attempt by a group of users and vendors to bring some order and honesty to parallel benchmarking. The objectives were to gain some wide acceptance for an agreed set of paral-

lel benchmarks and to make the codes and results publicly available. The methodology[21] is based on some of the principles deduced from past benchmarking practice as outlined above. The Parkbench suite consists of a hierarchy of tests as explained earlier, and results have been available on the Parkbench web site[22]. One interesting innovation was the reporting of measurements of performance for both unoptimized and optimized codes. The magnitude of the difference provides useful information about the "ease of use" of the machine.

Users have now broadly converged to a standard selection of tests in the suite. At the node level this consists of the LINPACK test, possibly augmented by Euroben tests. Common agreement on suitable codes to test the memory hierarchy are still missing. Kernel codes are provided by the NAS serial and parallel benchmarks and low-level communications codes are based on the Genesis benchmarks.

Recently, discussion in the Parkbench Group has focussed on improving the low-level Genesis codes and some problems that have been reported, mainly related to the complexity and ease of use of the suite. To address these issues a new version of low-level Genesis communications benchmarks has been written and this will now constitute the low-level codes in the new release of the Parkbench suite.

4.1 New Low-Level Genesis Communications Benchmarks

The design philosophy of these low-level codes has strongly aimed at simplicity to address the concerns mentioned above. Simplicity aids portability and allows the timing measurements to have a clear significance. The requirement of portability is vital for low-level benchmarks where much of the interest lies in comparison of results on many different machines. There is little freedom in how to implement the core of a low-level benchmark, but small differences can occur, so it is important to be able to inspect this core part of the code and thereby have a precise understanding of how the measurements are made. This again is aided by simplicity.

These low-level codes include pingpong and other tests and have been written in Fortran 77 using MPI message passing. None of the benchmarks consists of more than a couple of hundred lines of code.

- `Comms-PingPong1`: Standard message pingpong with only one message propagating at a given time (simplex) using MPI_SEND – MPI_RECV pairs.
- `Comms-PingPong2`: Interleaved message pingpong in which two messages can be travelling in opposite directions at the same time (duplex) using MPI_SENDRECV.
- `Comms-PingPong3`: Comms-PingPong1 with data touching on every receive.
- `Comms-Strided1`: Sending strided data in a pingpong pattern from a matrix using a DO-LOOP.
- `Comms-Strided2`: Sending strided data in a pingpong pattern using the MPI_VECTOR datatype.
- `Comms-Allgather`: Determining the network saturation bandwidth using the collective MPI_ALLGATHER call.
- `Comms-Synch`: Synchronization by MPI barriers.

The tests are described in detail at the Genesis web site[23] where the code may also be downloaded. The pingpong tests are quite standard while the tests based on sending strided data allow the effect of the memory hierarchy on communication to be probed. `Comms-PingPong3` also investigates memory hierarchy by forcing data up through the caches and checking the receive buffer at each step. The results of the tests that involve the memory hierarchy depend much more strongly on the compiler optimisations that the results from the other tests. `Comms-Allgather` considers more than two processors and further tests investigating the collective operations are still to be added to the suite. The Genesis web site also features a searchable database of results obtained from these benchmarks on a variety of machines. We encourage new results to be submitted to the database.

The general pattern of use is to supply test parameters via an input file. The parameter values are not chosen automatically so they are under complete control of the user. Examples of the fixed format input files containing suitable parameter choices are provided with the distribution, but it is often helpful to make some quick initial runs covering a smaller parameter space

to get a feel for the results to be expected in a longer run. There are separate input files for each test.

Output files, containing the benchmark timings are in a format that can be directly submitted to the central repository, but they are ASCII and are easy to interpret.

We suggest that the tests be run several times to ensure that the results are stable. Non-repeatability can be due to a the system not being dedicated, but even for dedicated machines it is endemic for modern operating systems due to their complexity. By varying the parameters in the input file, the test can be made to take longer and this tends to average out fluctuations on short time scales leading to more stable results. The tests themselves contain a warmup section intended to reduce this problem.

We decided to separate the measurement and analysis stages because it is important to check that a particular model for the data is reasonable before attempting to fit it. The benchmark results are therefore in the form of raw timing measurements. Plots are generated automatically when results are submitted to the database, but we feel that this is no substitute for individual analysis. We have provided some simple script tools using gnuplot however we encourage any form of analysis such as the PICT[24] fitting tool that is also available at the web site.

4.2 Results

All results that have been submitted are publicly available on the database. Presently results are available for the Cray T3D, Cray T3E, Fujitsu VX4, Sun Enterprise 10000 and Origin 2000. Results on the IBM SP2 and CS2 in different configurations can also be viewed. An interesting baseline measurement is provided by results on a cluster of DEC alpha workstations running Linux and fast Ethernet.

As an example of the strided tests, we show in figure 1 a plot of the results from `Comms-Strided1` and `Comms-Strided2` on an Origin 2000. The much better performance when strided data is sent using the MPI_VECTOR derived datatype rather than using a simple DO-LOOP indicates that the MPI implementation is making more effective use of the memory hierarchy. The fall in the DO-LOOP curve at large message size can be ascribed directly to the finite cache size. The peak bandwidth seen with `Comms-Strided2` is similar to that observed with contiguous data in `Comms-PingPong1`.

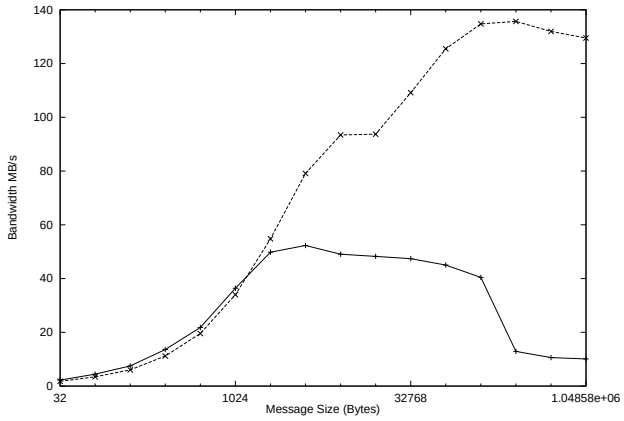

Figure 1: Bandwidth from `Comms-Strided1` (bottom line) and `Comms-Strided2` (top line) tests on an Origin 2000.

4.3 A Future for Parkbench?

It is suggested that a standard set of benchmark codes has now emerged and forms the basic Parkbench Suite. These are:

- Low-Level: LINPACK and New Genesis Communications benchmarks.
- Kernels: NAS Parallel Kernels.
- Compact Applications: NAS compact Applications.

Further work will be done in improving the quality and coverage of the Parkbench web sites at both Knoxville and Southampton. Now that there is a substantial set of results from the low-level tests, detailed analysis can be performed and both the techniques of analysis and the conclusions will be made available on the web.

The suite will continue to evolve. Certainly the low-level Genesis communications benchmarks will be enlarged to include more tests of collective operations. We have exposed the need for a standard serial test probing the memory hierarchy on a single node, for example a test with "tiling" compiler codes. A final possibility is to include HPF versions of the kernel class benchmarks.

5 Performance

In the preceding sections we have described how benchmarking has developed to the sophisticated level represented by the present Parkbench suite. Unfortunately the use of this benchmarking data in estimating the performance of large (or even moderately sized) codes is less advanced. In the absence of reliable modeling techniques low level simulation techniques have provided quite accurate performance figures, but these methods are time-consuming, require full disclosure of the source code and are machine specific. For the purposes of the IPG and other environments where it is essential to be able to make scheduling decisions rapidly, simulation is not appropriate. Below we describe several approaches investigated at Southampton that are intended to speed up the process and to avoid the need for source code which is often commercially sensitive. The PERFORM estimation tool and WHITEBOX benchmarking are research projects indicating some possible directions for progress. Our third example, INTREPID, is the basis of a commercial system for scheduling meta-applications based on performance models for the components.

5.1 PERFORM Estimation Tool

Low-level simulation provides a full description of the effect of the memory hierarchy that is missing in naive approaches to performance modeling based on total operation count, but is notoriously slow. PERFORM[3] is an attempt to preserve the good representation of the memory hierarchy by speeding up the low-level simulation technique using a method based on execution driven simulation and program slicing to analyse code. These techniques avoid full simulation of the complete code and select only the most significant parts, for example: load and store operations that take a variable amount of time to complete are fully simulated, whereas the time for arithmetic operations is based on simple operation counts. The most significant time saving choice is to only simulate the first few iterations of a loop until the complete loop timing can be accurately predicted. A further saving in both time and memory usage is to use program slicing to remove from the code all variables that do not directly affect the control flow.

Another difference from traditional low-level simulation is that the simulation engine is not tied directly to one particular machine. Operations

are represented in terms of a set of abstract machine instructions and a set of machine specific parameters is used to describe a particular machines in terms of this abstract model.

Execution driven simulation with program slicing certainly presents a much faster performance estimation method than low level simulation, and the results on a variety of simple codes such as the Livermore Loops are encouraging in terms of accuracy. The method still requires access to the source code.

5.2 WHITEBOX Benchmarking

WHITEBOX[4] benchmarking is an attempt to derive a useful level of profile information without excessive instrumentation of source code. Rather than profile the code in terms of the individual function calls, the code is divided into sections representing communication, data movement, and computation. These sections are timed by a method known as incremental conditional compilation.

This technique allows the communication pattern to be determined and one would expect to be able to model this using results from low-level communications benchmarks. It is an important lesson that this is not a trivial procedure: a study of NAS LU, BT and SP parallel codes indicates that their communication cannot be simply modeled in terms of two processor pingpong benchmark figures. It is precisely for such reasons that the other Genesis communication benchmark tests address the memory hierarchy and communication involving multiple processors and collective MPI calls.

While this approach certainly reduces the extent to which the source code must be disclosed, it can still provide detailed information. The most important conclusion is that substantial work will have to be put into relating performance to the results from low-level benchmarks.

5.3 INTREPID

To demonstrate the state of present day performance prediction in practical applications we describe INTREPID[2], a scheduler for meta-applications. INTREPID has a fuller knowlege of the applications than ordinary platform management systems, and is designed to interface with commercial systems

such as LoadLeveler and LSF. The performance modeling contained in INTREPID is very simple and phenomenological and consists of two parts: a general timing model and a specific load model for each application. The timing model is based on a simple fixed functional form with a small number of parameters describing the machine and other parameters describing the load imposed by the application. The bulk of the work involves the load model which must be created separately for each application to determine how the application load parameters depend on inputs such as problem size. This approach can be viewed as the opposite extreme to the low-level simulation technique and potentially does not need detailed knowledge of the source code. This example will show that despite the simplicity of the performance model, a complicated structure is still required in order to put it to use.

INTREPID has been developed though a series of EU funded projects as a manager of resources for meta-applications. Meta-applications are single applications consisting of a variety of tasks running on a distributed, non-dedicated system. The system is composed of a set of machines that might possibly contain individually parallel machines. INTREPID makes a critical path analysis of the meta-application under consideration and schedules it on the system in the most efficient way. To enable it to do this scheduling, a model for the performance of each task on any of the machines comprising the system is necessary.

The general timing model that fulfills this requirement is based on a fixed functional form for the time taken $T(P)$, on a machine with P processors with application parameters that depend on the particular task. It is hard to imagine that any more simple form than the following would capture essential features:

$$T(P) = t_{start} + \frac{L_{cpu} f_{cpu}}{R_{cpu}} + \frac{L_{cpu}(1 - f_{cpu})}{P R_{cpu}} + \frac{L_{io} f_{io}}{R_{io}} + \frac{L_{io}(1 - f_{io})}{P R_{io}} + \frac{L_{comm}(P)}{R_{comm}} \tag{1}$$

The rate parameters R_{cpu}, R_{io} and R_{comm} are determined by standard benchmarks and depend only on the hardware of the processors as does P the number of processors in the machine. The other parameters, known as application parameters, $L_{cpu}, L_{io}, L_{comm}$ as well as f_{cpu} and f_{io} are parameters characterising the application. The L's are loads, whereas the f's describe, in Amdahl form, the fraction of the code that is intrinsically serial. The final

parameter t_{start} depends on both machine and application and represents the time needed for license authentication etc.

Having identified the application parameters required, the performance modeling enters the second phase of the application load model which for each task determines the dependence of the L and f application parameters in terms of input. First, some knowledge of the application is needed to choose a small number of parameters that characterise the input, describing the particular problem being solved. In most cases there is some parameter describing the "problem size", for example the number of mesh points in a finite element code, and there may be further parameters such as the number of time steps needed. The essence of the performance modeling is then reduced to identifying these parameters and determining how the quantities $L_{cpu}, L_{io}, L_{comm}, f_{cpu}$ and f_{io} depend on them. A separate model must be constructed for each application.

In the context of the EU projects, this modeling procedure has been carried out for a variety of commercial application codes:

- CALIFE, a CFD application from Bertin.
- The finite element statics NASTRAN solution sequence from MSC.
- PARNASO, a non-linear structural analysis code from ENEL.
- AS-THETIS, an electromagnetics analysis code from AeroSpatiale.
- SIMAID, a multi-body dynamics code for robotics from CEIT.

It is important to stress that the full source code is not necessary to derive models for these applications because this can be done by fitting curves based on timing data. On the other hand, this is time-consuming and not very accurate, and the more that is known about the application the better will be the estimates. Despite the simplicity of the model, it is remarkable that performance estimates for the applications listed above lie within 20% of the measured values. Successful full scale tests of INTREPID have been made on up to 100 machines located in a variety of sites throughout Europe.

Although the performance models are extremely simple, INTREPID represents a full present day scheduler. This should be well understood both in terms of the accuracy with which it can predict performance and of the time constraints under which it operates, in order to see how more sophisticated

approaches, such as those considered in PERFORM and WHITEBOX can contribute.

6 Conclusion

We have given an overview of the development of benchmarking and have illustrated how the lessons learned have been incorporated into the present day Parkbench suite. This consists of:

- Low-Level: LINPACK and New Genesis Communications benchmarks.
- Kernels: NAS Serial and Parallel Kernels.
- Compact Applications: NAS Compact Applications.

As a result of this effort, good data now exists for low-level benchmarks on many parallel systems. The immediate challenge for performance analysis is to understand the behaviour of kernel benchmarks and small applications in terms of the measurements taken from low-level benchmarks and machine parameters. In the longer term this knowledge must be used for prediction of full application performance on modern platforms such as the IPG. Already, the Legion and Globus projects[1] have discussed the implementation of resource management in their respective environments. Progress in incorporating performance modeling will come from projects such as INTREPID and POEMS[25]. We have emphasised some of the important requirements, including speed of performance prediction and the need for modeling that does not require source code disclosure.

Acknowledgments

The authors would like to acknowledge assistance from Alistair Dunlop, Mark Papiani, Dennis Nicole, Kenji Takeda and Ivan Wolton.

References

[1] I. Foster and C. Kesselman (eds.), *The GRID: Blueprint for a New Computer Infrastructure*, Morgan Kaufman, San Francisco (1998).

[2] Contact the Parallel Applications Centre, University of Southampton:

`http://www.pac.soton.ac.uk`

[3] A.N. Dunlop, and A.J.G. Hey, *PERFORM - A Fast Simulator For Estimating Program Execution Time*, J. of Perf. Eval. and Modeling of Comp. Sys. (PEMCS), (1997).

[4] E. Hernández and A.J.G. Hey, *White-Box Benchmarking*, (1998), available at:

`http://www.usb.ve/emilio/WhiteBox.ps`

[5] G. Hoffmann, Private Communication.

[6] H.J. Curnow, B.A. Wichmann, *A Synthetic Benchmark*, Computer Journal **19**, 1, (1976) 43-49.

[7] F.K.McMahon, *The Livermore Fortran Kernels: a Computer Test of Numerical Performance Range*, Lawrence Livermore National Lab., Technical Report UCRL-53745, 1986.

[8] R. Weicker, J. Reilly, *SPEC FAQ*,

`http://npwww.epfl.ch/bench/SPEC.FAQ.html`

[9] R. Eigenmann and S. Hassanzadeh, *Benchmarking with Real Industrial Applications: The SPEC High-Performance Group*, IEEE Comp. Sc. & Eng., **3**, (1996) 18–23.

[10] *The Science of Computer Benchmarking*, SIAM, Philadelphia (1996)

[11] M. Berry et. al., *The Perfect Club Benchmarks: Effective Performance Evaluation of Supercomputers*, Int. J. Supercomputer Applications, **3**, 1,(1989) 5-40.

[12] R. Eigenmann, J. Hoeflinger and D. Padua, *On the Automatic Parallelization of the Perfect Benchmarks*, EEE Trans. of Prl. and Dist. Sys., **9**, 1, (1998) 5–23.

[13] C.A. Addison, V.S. Getov, A.J.G. Hey, R.W. Hockney and I.C. Wolton, The GENESIS Distributed Memory Benchmarks, Computer Benchmarks, J.J Dongarra and W. Gentzsch (Eds), Advances in Parallel Computing, **8**, Elsevier BV, Amsterdam, 1991.

[14] D.H. Bailey et. al. *The NAS Parallel Benchmarks*, Int. J. Supercomputer Applications, **5**, 3,(1991) 63-73.

[15] J.J. Dongarra, *Performance of Various Computers Using Standard Linear Equation Software*, Report CS-89-85, Univ. of Tennessee, Knoxville, Nov. 1996.

[16] Top 500 results are available from the University of Mannheim and also from netlib:

`http://www.netlib.org/benchmark/top500/top500.list.html`

[17] A.J. van der Steen, *The Benchmark of the EuroBen Group*, Parallel Computing, **17**, (1991) 1211-1221.

[18] Aad J. van der Steen, Ruud J. van der Pas, *Benchmarking the Silicon Graphics Origin 2000 System*, Dept. of Computational Physics, Utrecht Univ., Technical Report **WFI-98-2**, 1998.

[19] The RAPS consortium is organized by Pallas Gmbh.

`http://www.pallas.de/raps.html`

[20] D.H. Bailey, *Twelve ways to fool the masses when giving performance results on parallel computers*, Supercomputer, **45**, VIII-5, (1991) 4-7.

[21] R. Hockney, M. Berry (Eds.), *Public International Benchmarks for Parallel Computers* , Parkbench Committee Report No 1, Scientific Programming, **3**, (1994) 101-104.

[22] Parkbench web site:

`http://www.netlib.org/parkbench/`

[23] Southampton Low-level communications benchmarks using MPI and Fortran 77. Available with documentation and results database at:

`http://gather.ecs.soton.ac.uk`.

[24] See the original site at:

`http://www.minnow.demon.co.uk/pict`.

[25] E. Deelman, A. Dube, A. Hoisie, Y. Luo, R.L. Oliver, D. Sundaram-Stukel, H. Wasserman, V.S.Adve, R. Bagrodia, J.C. Browne, E. Houstis, O. Lubeck, J. Rice, P.J. Teller and M.K. Vernon, *POEMS: End-to-end Performance Design of Large Parallel Adaptive Computational Systems.* Available at:

`http://www.cs.utexas.edu/users/poems`