A Framework for Benchmark Performance Analysis

Roger Hockney
Electronics and Computer Science Department
Southampton University, Southampton SO9 5NH, UK

Published in: *Supercomputer 48*, **IX**-2, (1992), pp.9-22.

A methodology is proposed for the sensible analysis of computer benchmark results, and illustrated with results from a parallelised particle/mesh (PIC) code, the Genesis LPM1 benchmark. The importance of choosing the correct performance metric is emphasised, and the temporal, simulation, benchmark and hardware performance metrics are defined. The use of speedup and Mflop/s as figures of merit are strongly discouraged.

1 Introduction

The conclusions drawn from a benchmark study of computer performance depend not only on the basic timing results obtained, but also on the way there are interpreted and converted into performance figures. The choice of the performance metric, may itself influence the conclusions. For example, do we want the computer that generates the most Mflop/s (or has the highest speedup), or the computer that solves the problem in the least time? It is now well known that high values of the first metrics do not necessarily imply the second property. This confusion can be avoided by choosing a more suitable metric that reflects solution time directly, for example either the temporal or simulation performance, defined below. This issue of the sensible choice of performance metric is becoming increasing important with the advent of massively parallel computers which have the potential of very high Mflop rates, but have much more limited potential for reducing solution time. The following methodology, may seem rather pedantic, but we feel it is important in benchmarking, as in any other field of science, to define clearly the units and symbols being used.

2 Time Measurement

The fundamental measurement made in any benchmark is the time to complete some specified task. All other performance figures are derived from this basic timing measurement. The benchmark time, $T(N;p)$, will be a function of the number of processors, p, and the problem size, here represented by the vector variable, N, which stands for a set of parameters characterising the size of the problem, e.g. the number of mesh points in each dimension, and the number of particles in a particle-mesh simulation. Benchmark problems of different sizes can be created by multiplying all the size parameters by suitable powers of a single scale factor, thereby increasing the spatial and particle resolution in a sensible way, and reducing the size parameters to a single size factor (usually called α).

We believe that it is most important to regard execution time and performance as a function of at least the two variables (N, p), which define a parameter plane. Much confusion has arisen in the past by attempts to treat performance as a function of a single variable, by taking a particular path through this plane, and not stating what path is taken. Many different paths may be taken, and hence many different conclusions can be drawn. It is important, therefore, always to define the path through the performance plane, or better as we do here, to study the shape of the two-dimensional performance hill. We show in the performance analysis that there may also be an optimum path up this hill.

3 Floating-Point Operation Count

Although we discourage the use of millions of floating-point operations per second as a performance metric, it can be a useful measure if the number of floating-point operations, $F(N)$, needed to solve the benchmark problem is carefully defined. Also, it is particularly important to make a clear distinction between the quantity of arithmetic performed (the floating-point operation count or work, with the unit symbols 'flop' or 'Mflop') and the rate at which these operations are performed (the performance, with the unit symbols 'flop/s' or 'Mflop/s'). This distinction is analogous to that made in mechanics between 'work' which has the unit 'joule', and 'power' which has the unit 'joule/s' or 'watt'. For this reason we follow the usage of [1, 2] and abbreviate floating-point operation(s) to flop, but never pluralise it with an added 's', in order to avoid any possible confusion with a performance rate (which is written flop per second or flop/s). Note that the 'p' is an integral part of the name for the unit 'flop', and does not mean 'per' which has the standard symbol '/' as in other physical units, e.g. cm/s etc. In this way we follow the normal conventions for naming international SI units in physics as recommended by the Royal Society [3], and avoid the ambiguity which can arise with such notations as Mflops (is this the plural of Mflop or is it a rate 'per second'?), and the popular MFLOPS and MFLOPs which are similarly open to misinterpretation. There is no good reason why computer scientists should not follow good international practice in naming their units.

For simple problems (e.g. matrix multiply) it is sufficient to use a theoretical value for the floating-point operation count (in this case $2n^3$ flop, for nxn matrices) obtained by inspection of the code or consideration of the arithmetic in the algorithm. For more complex problems containing data-dependent conditional statements, an empirical method may have to be used. The sequential version of the benchmark code defines the problem and the algorithm to be used to solve it. Counters can be inserted into this code or a hardware monitor used to count the number of floating-point operations. The latter is the procedure followed by the PERFECT Club [4]. In either case a decision has to be made regarding the number of flop that are to be credited for different types of floating-point operations, and we see no good reason to deviate from those chosen by McMahon [5] when the Mflop/s measure was originally defined. These are:

add, subtract, multiply	1 flop
divide, square-root	4 flop
exponential, sine etc.	8 flop
IF(X .REL. Y)	1 flop

We distinguish two types of operation count. The first is the nominal benchmark floating-point operation count, $F_B(N)$, which is found in the above way from the defining Fortran77 sequential code. The other is the actual number of floating-point operations performed by the hardware when executing the distributed multi-node version, $F_H(N)$, which may be greater than the nominal benchmark count, due to the distributed version performing redundant arithmetic operations.

4 Performance Metrics

Given the time of execution $T(N;p)$ and the flop-count $F(N)$ several different performance measures can be defined. Each metric has its own uses, and gives different information about the computer and algorithm used in the benchmark. It is important therefore to distinguish the metrics with different names, symbols and units, and to understand clearly the difference between them. Much confusion and wasted work can arise from optimising a benchmark with respect to an inappropriate metric. The principal performance metrics are:

4.1 Temporal Performance

If we are interested in comparing the performance of different algorithms for the solution of the same problem, then the correct performance metric to use is the *Temporal Performance*, R_T, which is defined as the inverse of the execution time

$$R_T(N;p) = T^{-1}(N;p) \qquad (1)$$

The units of temporal performance are, in general, solutions per second (sol/s), or some more appropriate absolute unit such as timesteps per second (tstep/s). With this metric we can be sure that the algorithm with the highest performance executes in the least time, and is therefore the best algorithm. We note that the number of flop does not appear in this definition, because the objective of algorithm design is not to perform the most arithmetic per second, but rather it is to solve a given problem in the least time, regardless of the amount of arithmetic involved. For this reason the temporal performance is also the metric that a computer user should employ to select the best algorithm to solve his problem, because his objective is also to solve the problem in the least time, and he does not care how much arithmetic is done to achieve this.

4.2 Simulation Performance

A special case of temporal performance occurs for simulation programs in which the benchmark problem is defined as the simulation of a certain period of physical time, rather than a certain number of timesteps. In this case we speak of the *simulation performance* and use units such as *simulated days per day* (written sim-d/d or 'd'/d) in weather forecasting where the apostrophe is used to indicate 'simulated', or *simulated pico-seconds per second* (written sim-ps/s or 'ps'/s) in electronic device simulation. It is important to use simulation performance rather than timestep/s if one is comparing different simulation algorithms which may require different sizes of timestep for the same accuracy (for example an implicit scheme that can use a large timestep, compared with an explicit scheme that requires a much smaller step). In order to maintain numerical stability, explicit schemes also require the use of a smaller timestep as the spatial grid is made finer. For such schemes the simulation performance falls off dramatically as the problem size is increased by introducing more mesh points in order to refine the spatial resolution: the doubling of the number of mesh-points in each of three dimensions can reduce the simulation performance by a factor near 16 because the timestep must also be approximately halved. Even though the larger problem will generate more Megaflop per second, in forecasting, it is the simulated days per day (i.e. the simulation performance) and not the Mflop/s, that matter to the user.

As we see below, benchmark performance is also measured in terms of the amount of arithmetic performed per second or Mflop/s. However it is important to realise that it is incorrect to compare the Mflop/s achieved by two algorithms and to conclude that the algorithm with the highest Mflop/s rating is the best algorithm. This is because the two algorithms may be performing quite different amounts of arithmetic during the solution of the same problem. The temporal performance metric, R_T, defined above, has been introduced to overcome this problem, and provide a measure that can be used to compare different algorithms for solving the same problem. However, it should be remembered that the temporal performance only has the same meaning within the confines of a fixed problem, and no meaning can be attached to a comparison of the temporal performance on one problem with the temporal performance on another.

4.3 Benchmark Performance

In order to compare the performance of a computer on one benchmark with its performance on another, account must be taken of the different amounts of work (measured in flop) that the different problems require for their solution. Using the flop-count for the benchmark, $F_B(N)$, we can define the benchmark performance as

$$R_B(N;p) = F_B(N)/T(N;p) \tag{2}$$

The units of benchmark performance are Mflop/s (benchmark name), where we include the name of the benchmark in parentheses to emphasise that the performance may depend strongly on the problem being solved, and to emphasise that the values are based on the nominal benchmark flop-count. In other contexts such performance figures would probably be quoted as examples of the so-called *sustained* performance of a computer. We feel that the use of this term is meaningless unless the problem being solved and the degree of code optimisation is quoted, because the performance is so varied across different benchmarks and different levels of optimisation. Hence we favour the quotation of a selection of benchmark performance figures, rather than a single sustained performance, because the latter implies that the quoted performance is maintained over all problems.

Note also that the flop-count $F_B(N)$ is that for the defining sequential version of the benchmark, and that the same count is used to calculate R_B for the distributed-memory (DM) version of the program, even though the DM version may actually perform a different number of operations. It is usual for DM programs to perform more arithmetic than the defining sequential version, because often numbers are recomputed on the nodes in order to save communicating their values from a master processor. However such calculations are redundant (they have already been performed on the master) and it would be incorrect to credit them to the flop-count of the distributed program. Using the sequential flop-count in the calculation of the DM programs benchmark performance has the additional advantage that it is possible to conclude that, for a given benchmark, the implementation that has the highest benchmark performance is the best because it executes in the least time. This would not necessarily be the case if a different $F_B(N)$ were used for different implementations of the benchmark. To this extent benchmark performance Mflop/s must be understood to be nominal values, and not necessarily exactly the number of operations executed per second by the hardware.

4.4 Hardware Performance

If we wish to compare the observed performance with the theoretical capabilities of the computer hardware, we must compute the actual number of floating-point operations performed, $F_H(N;p)$, and from it the actual hardware performance

$$R_H(N;p) = F_H(N)/T(N;p) \tag{3}$$

The hardware performance also has the units Mflop/s, and will have the same value as the benchmark performance for the sequential version of the benchmark. However, the hardware performance may be higher than the benchmark performance for the distributed version, because the hardware performance gives credit for redundant arithmetic operations, whereas the benchmark performance does not.

Assuming a computer with multiple-CPUs each with multiple arithmetic pipelines, delivering a maximum of one flop per clock period, the theoretical peak value of hardware performance is

$$r^* = \frac{floating - pointpipes/CPU}{clockperiod} \times numberofCPUs \tag{4}$$

with units of Mflop/s if the clock period is expressed in microseconds. By comparing the measured hardware performance, $R_H(N;p)$, with the theoretical peak performance, we can assess the fraction of the available performance that is being realised by a particular implementation of the benchmark.

Computer performance is conveniently presented both in tabular form and as graphs of $R(N;p)$ for constant N plotted against p on log/log axes. The use of logarithmic axes allows the inclusion of a wide range of processor number and performance on a single graph. Within one benchmark, it also permits the vertical axis to be interpreted as either time or performance by reading the appropriate scale. In this respect we follow the practice of CPEP [6], but we plot the performance the *right-way-up* with the highest performance at the top not the bottom. In such log/log performance graphs, the lines of perfect linear speedup (performance directly proportional to the number of processors) are diagonal straight lines.

4.5 Speedup and Efficiency

We do not favour the use of any of the popular performance metrics: speedup, efficiency or performance per node; because all these are either easily misinterpreted or obscure important effects. The speedup of a benchmark code is defined as the ratio of the p-processor temporal performance to the single-processor temporal performance. It is a very useful and convenient measure if we are concerned with the optimisation of a particular code in isolation, because its value can easily be compared with the maximum possible speedup, namely the number of processors being used. We can thereby assess how much of the potential hardware performance is being utilised. However benchmarking is to do with comparing the performance of different computers, and all the above three metrics are unsuitable for this purpose.

Speedup compares the performance of a code with itself, and might therefore be called an introspective, or even incestuous measure. Problems can therefore arise (see below), and incorrect conclusions can be drawn, if we try to use speedup to compare different algorithms on the same computer, or the same algorithm on different computers. This is because speedup is a relative measure (it is defined as the ratio of two performances), and therefore all knowledge of the absolute performance has been lost. Benchmarking, however, is concerned with the comparison of the absolute performance of computers, and therefore the use of a relative measure like speedup is not very useful, and can be positively misleading. For example, we do not wish to conclude that a computer with a large number of slow processors and therefore high value of speedup, is faster than another with fewer processors and therefore with a lower speedup, if in fact the reverse is the case, because the processors on the second computer are so much faster. Only by adopting absolute measures of performance with physical units involving inverse time, can one avoid this type of false conclusion.

Speedup is not even useful for comparing the performance of one algorithm with another on the same computer, because it is not necessarily true that the algorithm with the highest speedup executes in the least time (see, e.g. [7]). One can only be sure that this is the case if the single-processor temporal performance of both algorithms is the same, which is most unlikely. If the single-processor performances of the two algorithms are different and we compare the speedups of the two algorithms, then we are comparing the performance of the two algorithms measured in different units. This is like comparing the speeds of two cars, one measured in m.p.h. and the other in cm/s. Such a comparison has no validity either for cars or algorithms. Computers and algorithms can only safely be compared in terms of their absolute performance in solving a problem. The most unambiguous measure is the temporal (or simulation) performance, which is the inverse of the time of execution.

The benchmark performance per node might seem to be an attractive metric because it is an absolute measure which can be related directly to the hardware performance of node. However it has the major defect that it hides the point at which the absolute performance begins to decrease as the number of processors increases. If we plot benchmark performance against number of processors, this point is clearly visible as a maximum, however if the same data is plotted as performance per node, all we see is a very uninteresting monotonically falling line, and the important maximum has disappeared. The efficiency, which is defined as the speedup divided by the number of processors, is doubly condemned because it is a relative measure and hides the maximum. As an example of the proposed methodology we present below the complete analysis of the Local Particle-Mesh benchmark (LPM1) from the Genesis distributed memory benchmark suite.

5 Local Particle-Mesh Benchmark (LPM1)

This benchmark is the time dependent simulation of an electronic device using a particle-mesh or PIC-type algorithm. It is a two-dimensional (r, z) geometry with the fields computed on a regular mesh (33x75). The fields computed are E_r, E_z, and B_ϕ(into the mesh). The electron distribution in the device is represented by a group of 500 to 1000 simulated electrons each of which has its positions and velocity stored. The timestep proceeds as follows:

1. The particle coordinates are inspected, and the current of each particle is assigned to the mesh points in its neighbourhood.
2. The fields are advanced for a short timestep according to Maxwell's equations.
3. Each particle is accelerated according to Newton's Laws with the new field values obtained by inspecting the mesh values near each particle.
4. Injection takes place along the cathode of the device according to the value of the normal E-field.

This is called a local particle-mesh simulation because the timescale is such that only neighbouring (i.e. local) field values are required during the update of the field equations. In contrast a global simulation might require the solution of, say, Poisson's equation over the whole mesh. Because of the locality of data a geometric subdivision of the problem between the nodes of a transputer network (or other distributed-memory computer such as Suprenum) is natural. The existing distributed-memory implementation takes this approach, and performs a one-dimension domain decomposition. For p processors, the device is divided into p approximately equal slices in the z-direction, and each slice is assigned to a different processor. Each processor is responsible for computing the fields, and moving the particles in its region of space. When particles move out of this region, their coordinates are transferred to the correct neighbouring processor, which computes their subsequent motion. The timestep is such that a particle will move no further than to a neighbouring processor in one timestep. The processors are therefore configured as a chain, and during the timestep loop, communication is only required between neighbouring processors in the chain. The communication comprises an exchange of the edge values of the fields and currents, together with the coordinates of any particles that are moving between processors.

The standard benchmark is defined as the simulation of one nanosecond of device time using a (33,75) mesh and starting from an empty device. During this time electrons are emitted from the cathode starting at the left of the device, and the electron cloud gradually fills the device from left to right. After one nanosecond the device is about half filled with electrons. Larger problem sizes are introduced by using more mesh-points in the z-direction, and computing on a $(33, 75\alpha)$ mesh, where α is the problem-size factor (=1,2,4,8). It is found empirically that the total number of particles in the system at the end of one nanosecond grows somewhat faster than proportionality to α and is represented within a few percent by

$$N_{end} = 628\alpha^{1.172} \tag{5}$$

In order to produce a timing formulae we must make a simple model of the filling of the device. Since the number of mesh-points computed by each processor is approximately the same, the timing of the multiprocessor simulation will be determined by the time take by the processor containing the most particles. Since the device is filled from left to right this will be the leftmost processor in the chain, and will be called the critical processor. We assume that the critical processor fills at a constant rate until it contains $2N_{end}/p$ particles (which are its share of the particles present at the end of the simulation), and that the number of particles in the critical processor remains constant at this value for the rest of the benchmark run. The '2' in the above expression arises because only the first $p/2$ processors in the chain contain any particles at the end of the benchmark (i.e. the device only half fills). The average number of particles in the critical processor is then given by

$$N_1 = \frac{2N_{end}}{p}(1 - 1/p) \tag{6}$$

where the first term would apply alone if the critical processor was filled with particles from the beginning, and the second term is the correction taking into account that there are fewer particles during the filling process.

An inspection of the Fortran code shows that, per timestep, there are 46 flop per mesh point and 116 flop per particle in the most important terms of the operation-count. This leads us to define the nominal flop-count per timestep for the benchmark to be

$$F_B(\alpha) = 46 \times 33 \times 75\alpha + 116 \times N_{end}/2 \tag{7}$$

where the last term assumes the number of particles increases linearly from zero to N_{end} during the reference single-processor run.

Figure 1 shows the temporal performance in units of timestep per second as a function of the number of processors for four problem sizes on a 64-processor Parsys SN1000 Supernode.

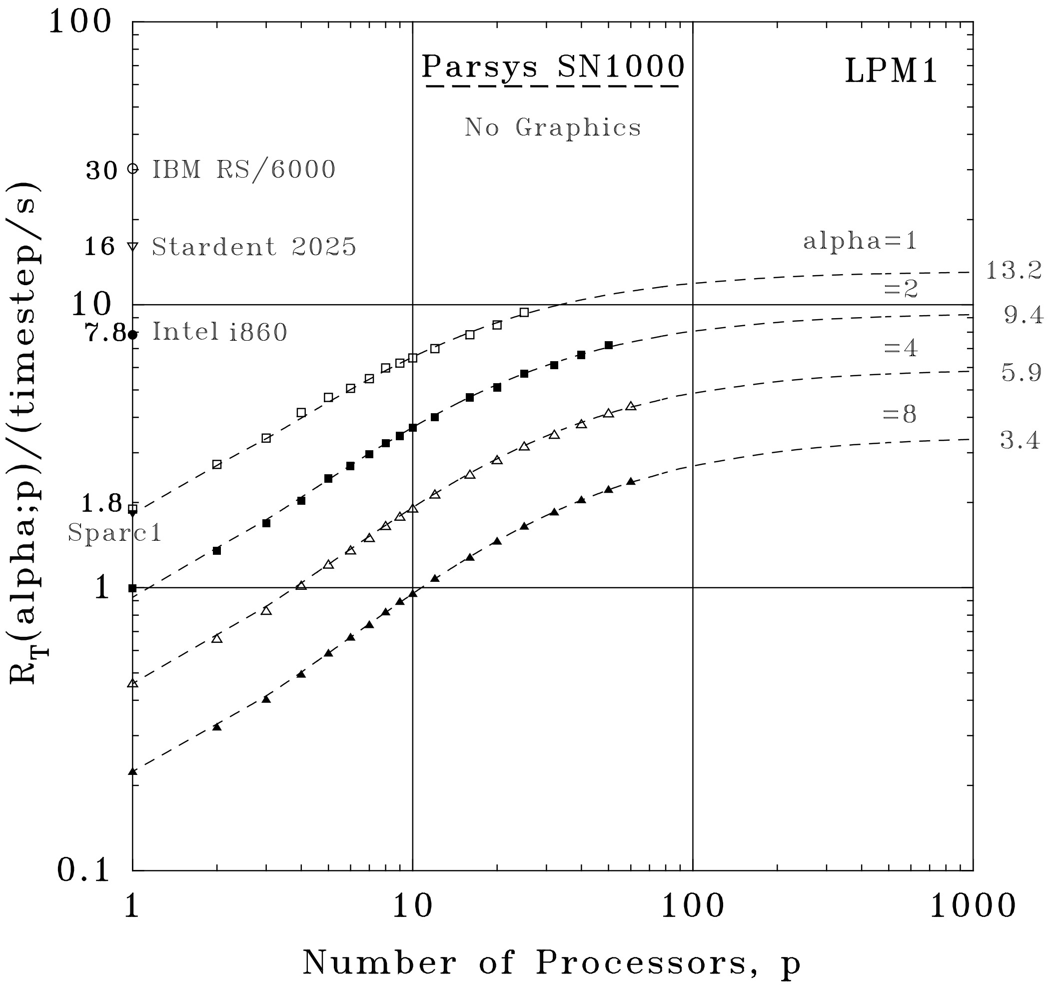

Figure 1: Temporal performance of the LPM1 benchmark for four problem sizes as a function of the number of processors used, on a Parsys SN1000 Supernode with 25 MHz T800 transputers. α is the problem-size factor.

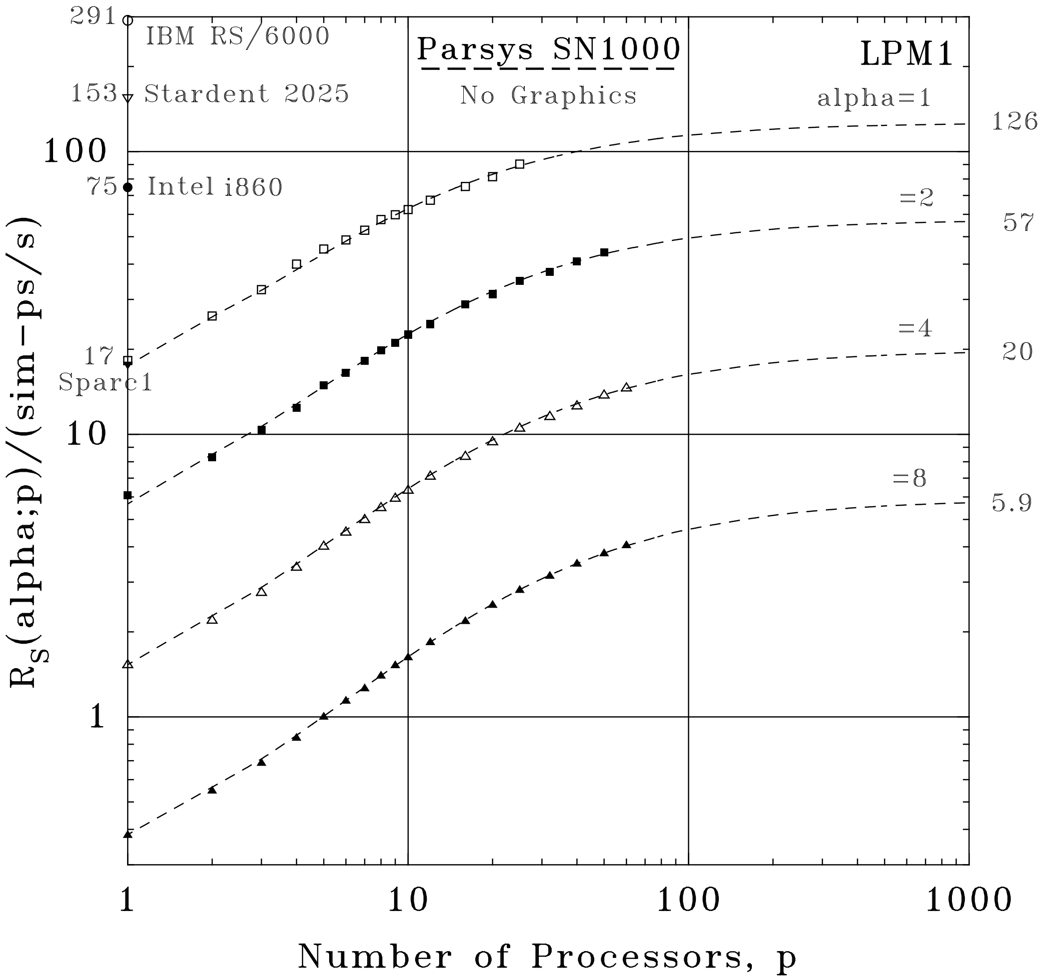

Figure 2: Simulation performance of the LPM1 benchmark for four problem sizes as a function of the number of processors used, on a Parsys SN1000 Supernode with 25 MHz T800 transputers. α is the problem-size factor. The timing data used is the same as Figure 1.

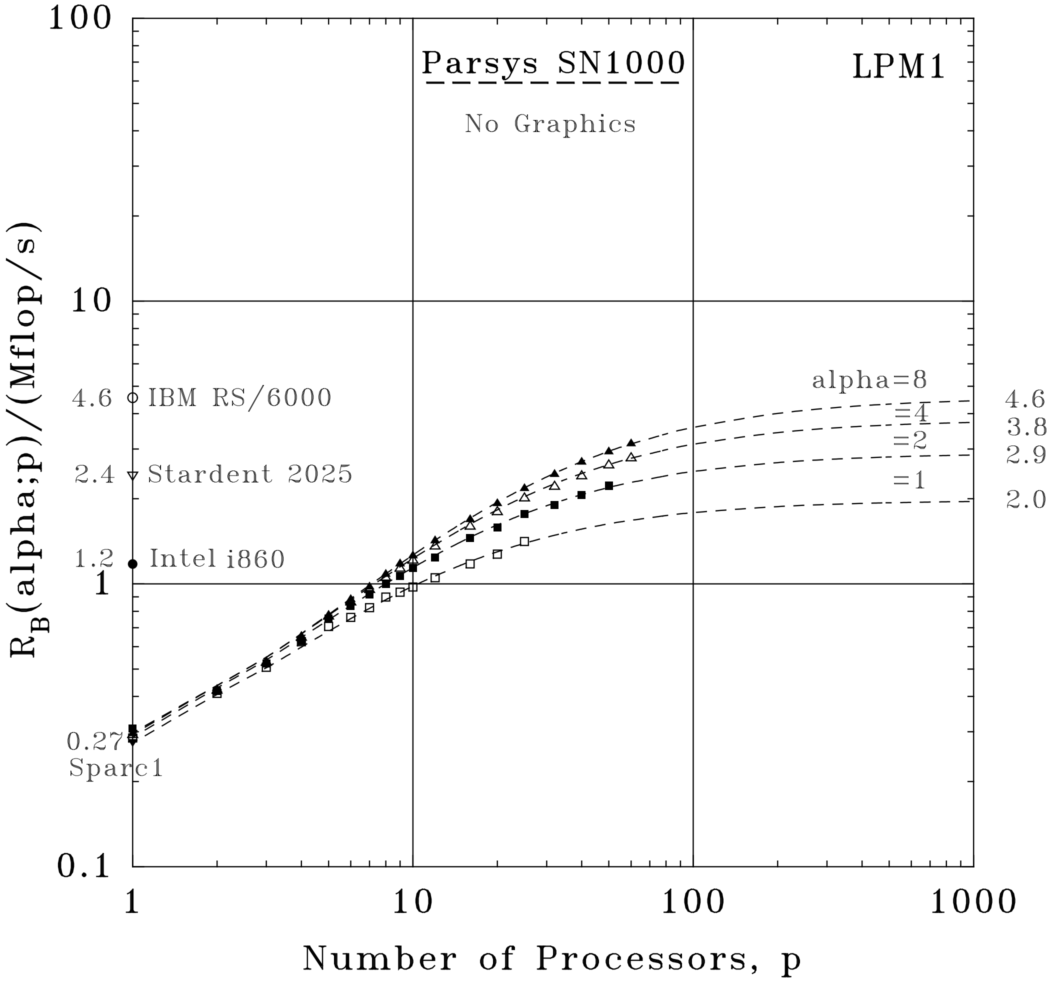

Figure 3: Benchmark performance of the LPM1 benchmark for four problem sizes as a function of the number of processors used, on a Parsys SN1000 Supernode with 25 MHz T800 transputers. α is the problem-size factor. The timing data used is the same as Figure 1.

Each node comprises a 25 MHz Inmos T800 transputer with 4 MByte of local memory. The problem size is increased by multiplying the number of mesh points in the z-direction by a factor $\alpha = 1, 2, 4, 8$. For simplicity there is no change to the number of mesh points in the r-direction, nor is there domain decomposition in the r-direction. That is to say that each processor computes all the r-points in its region of the z-direction.

The time of execution of the benchmark has been fitted within a few percent to the formula

$$T(\alpha; p) = a + b\alpha + \frac{c\alpha}{p}\left\{1 + 1.280\alpha^{0.172}(1 - 1/p)\right\} \tag{8}$$

where $a = 0.04532$s, $b = 0.03071$s, and $c = 0.3584$s. The first two terms containing a and b take account of the non-parallelised part of the code and arise from both communication and arithmetic. The last term which is proportional to c and inversely proportional proportional to the number of processors p, is from the parallelised part of the code which is subdivided between the processors. The two parts of the parallelised code are represented by the two terms within the braces. The first term (=1) comes from the field calculation on the mesh points. The complicated second term comes from an empirical fit to the number of particles in the critical processor that contains the most particles, and therefore determines the calculation time. The values of a, b, c are obtained by a least-squares fit (using the SigmaPlot plotting package) to the temporal performance which is given by

$$R_T(\alpha; p) = \frac{1}{T(\alpha; p)} \tag{9}$$

Figure 1 also shows on the y-axis the performance of a number of well-known single-processor workstations. The best of these is the IBM RS/6000 with a performance of 30 tstep/s, followed by the Stardent 2025 at 16 tstep/s. Next comes the Intel i860 with values obtained with the Fortran compiler from the Portland Group. These measurements were all made without graphics, and involved running the single-processor Fortran77 code without any changes. In contrast the distributed-memory code for the Parsys Supernode has taken many months to develop. It is clear that the Amdahl limit prevents the DM parallel SN1000 code from exceeding about 13 tstep/s, however many processors are used, and that this code runs slower than a single processor IBM RS/6000 or Stardent 2025. It is not believed that there is much scope for further significant parallelisation of the SN1000 code without changing the existing algorithm, therefore the only way to increase the performance of the SN1000 on this definition of the benchmark is to increase the speed of the processors and the communication. The recently announced Inmos T9000 transputer with about 10 times the processing and communication rates of the T800 transputer used in the SN1000, should achieve this and reach an asymptotic processing rate of about 80 tstep/s, provided that the startup time for communication is correspondingly also reduced by a factor of ten. However unless the graphics can similarly be speeded up, this processing rate could not be used in a problem requiring continuous graphical visualisation, because the time would be dominated by the slow graphical display and not by the fast calculation.

Figures 2 and 3 show the same performance data as figure 1 but plotted as simulation performance and benchmark Mflop/s performance. Both the temporal and simulation performance show that the smallest problem ($\alpha = 1$) is the fastest, and that the fall-off of performance with problem size is much faster for simulation performance because the timestep size decreases as the number of mesh points increases. However the benchmark Mflop/s is seen to increase with problem size, showing that this metric can be misleading, and does not indicate the actual performance of a simulation.

Figure 4 shows the performance results plotted in terms speedup. The speedup curves are similar in shape to those of the benchmark performance in Mflop/s but we have now

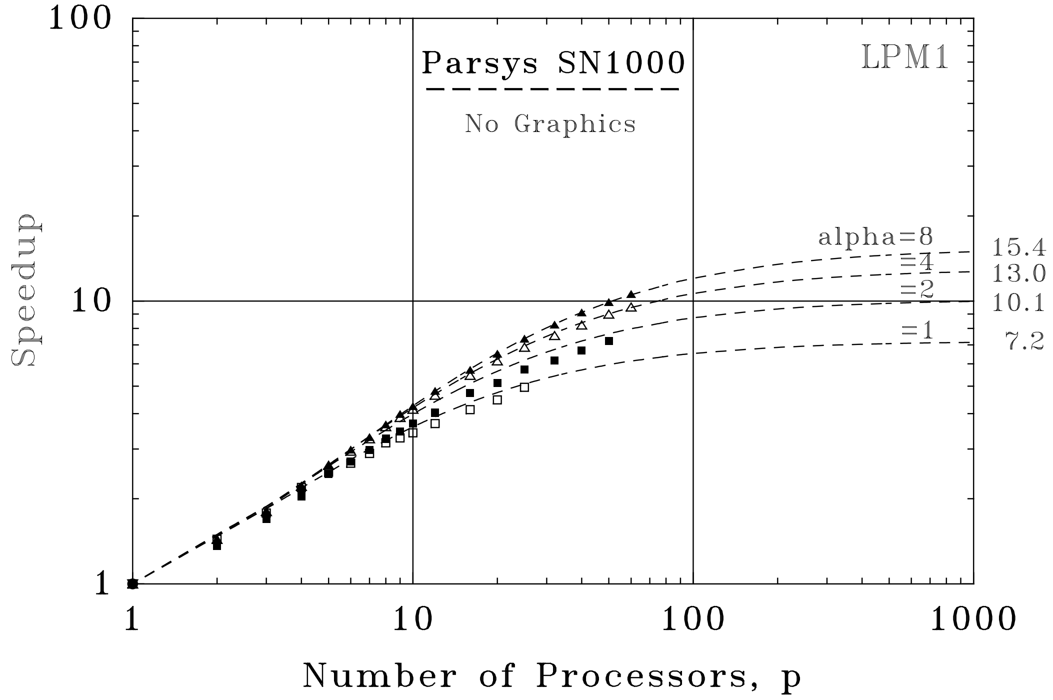

Figure 4: LPM1 benchmark results presented in terms of speedup. The timing data used is the same as Figure 1.

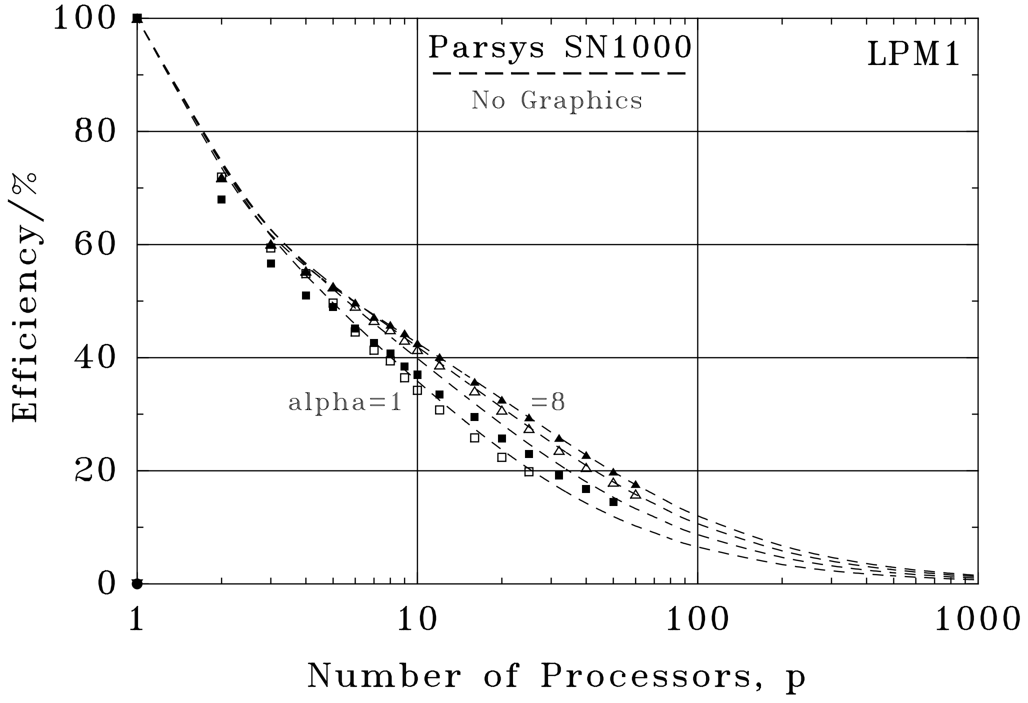

Figure 5: LPM1 benchmark results presented in terms of efficiency. The timing data used is the same as Figure 1.

lost all knowledge of the absolute value of the performance, and can no longer compare the performance with the other computers. The speedup, like the benchmark performance, credits the largest problem ($\alpha = 8$) with the highest performance, whereas we know from the temporal performance that the largest problem actually takes longer to run. Figure 5 shows the same results plotted in terms of the efficiency of calculation, that is to say the speedup divided by the number of processors. Again we have no comparison with other computers, and the largest problem is found to have the highest efficiency although it has the lowest temporal performance. We also find that the fastest problem (in sim-ps/s) has the lowest efficiency, so that it does not seem that efficiency is measuring what the user wants to know. This example clearly demonstrates why we do not think that benchmark results should be presented in terms of speedup or efficiency.

Acknowledgments

The author wishes to thank colleagues at Southampton University for their help in obtaining and interpreting these results, and in operating the Parsys Supernode SN1000: particularly Mark Debbage, Mark Hill, Nick Floros, Bryan Carpenter, John Merlin and Vladimir Getov.

References

[1] Hockney R.W., C.R. Jesshope, *Parallel Computers 2: Architecture, Programming and Algorithms*, (Adam Hilger/IOP Publishing, Bristol & New York, 1988). Distributed in the USA by the American Institute of Physics, c/o AIDC, 64 Depot Road, Colchester, VT 05445.

[2] *Supercomputer*, (ASFRA, Edam, Netherlands).

[3] Symbols Committee of the Royal Society, *Quantities, Units and Symbols* (The Royal Society, London, 1975).

[4] M. Berry, D. Chen, P. Koss, D. Kuck, S. Lo, Y. Pang, L. Pointer, R. Roloff, A. Sameh, E. Clementi, S. Chin, D. Schneider, G. Fox, P. Messina, D. Walker, C. Hsiung, J. Schwarzmeier, K. Lue, S. Orszag, F. Seidl, O. Johnson, R. Goodrum, J. Martin, The PERFECT Club benchmarks: effective performance evaluation of computers, *Intl. J. Supercomputer Appls.* **3**(3) (1989) 5-40.

[5] F. H. McMahon, The Livermore Fortran Kernels test of the numerical performance range, in: J. L. Martin,ed., *Performance Evaluation of Supercomputers* (Elsevier Science B.V., North Holland, Amsterdam, 1988) 143-186.

[6] P. Messina, C. Baillie, E. Felten, P. Hipes, R. Williams, A. Alagar, A. Kamrath, R. Leary, W. Pfeiffer, J. Rogers, D. Walker, Benchmarking advanced architecture computers, *Concurrency: Practice and Experience* **2**(3) (1990) 195-255.

[7] Z. Cvetanovic, E. G. Freedman, C. Notsinger, Efficient decomposition and performance of parallel PDE, FFT, Monte-Carlo simulations, simplex and sparse solvers, *Proceedings Supercomputing90* (IEEE, New York, 1990) 465-474.