8 How Fast is My Beowulf

David Bailey

One of the first questions that a user of a new Beowulf-type system asks is, "How fast does my system run?" Performance is more than just a curiosity for cluster systems. It is arguably the central motivation for building a clustered system in the first place—a single node is not sufficient for the task at hand. Thus the measurement of performance, and comparisons of performance between various available system designs and constructions, is of paramount importance.

8.1 Metrics

There are many different metrics for system performance, varying greatly in their meaningfulness and ease of measurement. Here are some of the more widely used metrics:

1. Theoretical peak performance. This statistic is merely the maximum aggregate performance of the system. For scientific users, this means the maximum aggregate floating-point operations per second, usually calculated as

$$P = N * C * F * R$$

where P is the performance, N is the number of nodes, C is the number of CPUs per node, F is the number of floating-point operations per clock period, and R is the clock rate, measured in cycles per second. P is typically given in millions of floating-point operations per second (Mflop/s) or in billions of floating-point operations per second (Gflop/s). For non-scientific applications, integer operations are counted instead of floating-point operations per second, and rates are typically measured in Mop/s and Gop/s, variantly given as Mip/s and Gip/s. For non-homogeneous systems, P is calculated as the total of the theoretical peak performance figures for each homogeneous sub-system.

The advantage of this metric is that is very easy to calculate, and, what's more, there is little disputing the result—the relevant data is in many cases publicly available. The disadvantage of this metric is that by definition it is unattainable by ordinary application programs. Indeed, a growing concern of scientific users, in particular, of parallel and distributed systems is that the typical gap between peak and sustained performance seems to be increasing, not decreasing.

2. Application performance. This statistic is the number of operations performed while executing an application program, divided by the total run time. As

with theoretical peak performance, it is typically given in Mflop/s, Gflop/s, Mop/s or Gop/s.

This metric, if calculated for an application program that reasonably closely resembles the program that the user ultimately intends to run on the system, is obviously a much more meaningful metric than theoretical peak performance. However, this metric is correspondingly harder to use, because one must first port the benchmark program to the cluster system, which may be a laborious and time-consuming task. Secondly, one must determine fairly accurately the number of floating-point (or integer) operations actually performed by the code. Along this line, one should ascertain that the algorithms used in the code are really the most efficient available for this task—otherwise one should use a floating-point operation count that corresponds to that of an efficient algorithm implementation, or else one's results can be questioned. One key difficulty with this metric is the extent to which the source code has been "tuned" for optimal performance on the given system—comparing results that on one system are based on a highly tuned implementation to those on another system, where the application has not be highly tuned, can be misleading. Nonetheless, if used properly this metric can be very useful.

3. Application run time. This statistic simply means the total wall-clock run time for performing a given application. One advantage of this statistic is that it frees the user from the need to count operations performed. Also, this avoids the potential distortion of using a code to assess performance whose operation count is larger than it needs be, due to its usage of an inefficient algorithm. In many regards, this is the "ultimate" metric, in the sense that it is precisely the ultimate figure of merit for an application running on a system. The disadvantage of this metric is that unless one is comparing two systems, both of which have run exactly the same application, it is hard to meaningfully compare systems based solely on comparisons of run time performance. Further, the issue of tuning also is present here—in comparing performance between systems one has to insure that both implementations have been comparably tuned.

4. Scalability. Users often cite "scalability" statistics when describing the performance of their system. This is usually computed as:

$$S = \frac{T(1)}{T(N)}$$

where $T(1)$ is the wall clock run time for a particular program on a single processor, and $T(N)$ is the run time on N processors. A scalability figure close to N means that

the program "scales" well—evidently the parallel implementation is very efficient, and the parallel overhead very low, so that nearly a "linear" speedup has been achieved.

Scalability statistics can often provide useful information. For example, they can help one determine an optimal number of processors for a given application. But they can also be misleading, particularly if cited in the absence of application performance statistics. For example, note that an impressive speedup statistic may be due to a very low value of $T(N)$, which appears in the denominator, but it may also be due to a large value of $T(1)$—in other words, an inefficient one-processor implementation. Indeed, it is a common experience of researchers working with parallel systems that their speedup statistic worsens when they accelerate their parallel program due to clever tuning. Also, it is often simply impossible to compute this statistic, because while a the benchmark test program may run on all or most of the processors in a system, it may require too much memory to run on a single node.

5. Parallel efficiency. A variant of the scalability metric is "parallel efficiency", which is usually defined to be $P(N)/N$. Parallel efficiency statistics near one are ideal. This metric suffers from the same potential difficulties as the scalability metric.

6. Percent of peak. Sometimes application performance statistics are given in terms of the percent of theoretical peak performance. Such statistics are useful in highlighting the extent to which an application is utilizing the full computational power of the system. For example, a low percentage of peak may indicate a mismatch of the architecture and the application, deserving further study to determine the source of the difficulty. However, a percent-of-peak figure by itself is not too informative—an embarrassingly parallel application can achieve a high percentage of peak, but this is not a notable achievement. In general, percent-of-peak figures beg the question: "What percentage of peak is a realistic target for a given type of application?"

7. Latency and bandwidth. Many users are interested in the latency (time delay) and bandwidth (transfer rate) of the inter-processor communications network, since the network is one of the key elements of the system design. These metrics have the advantage of being fairly easy to determine. The disadvantage is that the network often performs differently under highly loaded situations than the latency and bandwidth figures by themselves reveal. And, needless to say, these metrics

characterize only the network, and give no information on the computational speed of individual processors.

8. System utilization. One common weakness of the above metrics is that they tend to ignore system-level effects. These effects include competition between two different tasks running in the system, competition between I/O-intensive tasks and non-I/O-intensive tasks, inefficiencies in job schedulers, job start-up delays, etc. Thus some Beowulf system users have measured the performance of a system on a long-term throughput basis, as a contrast to conventional benchmark performance testing.

As you can see, there no single type of performance measurement, much less a single figure of merit, that is simultaneously easy to determine and completely informative. In one sense, only one figure of merit matters, as emphasized above: the wall clock run time for your particular application on your particular system. But this is not easy to determine before a purchase or upgrade decision has been made. And even if you can make such a measurement, it is not clear how to compare your results with the thousands of other Beowulf system users around the world, not to mention other types of systems and clusters.

These considerations have led many users of parallel and cluster systems to compare performance based on a few standard benchmark programs. In this way, one can determine if your particular system design is as effective (as measured by a handful of benchmarks) as another. Such comparisons might not be entirely relevant to your particular application, but with some experience you can find one or more well-known benchmark that tends to give performance figures that are well correlated with your particular needs.

8.2 Ping-Pong Test

One of the most widely utilized measurements performed on cluster systems is the ping-pong test, which means one of several widely available test programs that measures the latency and bandwidth of the inter-processor communications network. There are a number of tools for testing TCP performance including `netperf` and `netpipe` (see `www.netperf.org` and `www.scl.ameslab.gov/netpipe`). Ping-pong tests that are appropriate for application developers measure the performance of the user API, and are typically written in C and assume that the message passing interface (MPI) communications library is installed on the system. More details on downloading and running these are given in Section 11.10.

8.3 The Linpack Benchmark

The Linpack benchmark dates back to the early 1980s, when Jack Dongarra, then of Argonne National Laboratory in the USA, began collecting performance results of systems, based on their speed in solving a 100×100 linear system using Fortran routines from the Linpack library. While this size problem is no longer a supercomputer-class exercise, it is still useful for assessing the computational performance of a single-processor system. In particular, it is a reasonable way to measure the performance of a single node of a Beowulf-type system. One can obtain the Linpack source code, plus instructions for running the Linpack benchmark, from the `http://www.netlib.org/benchmark`.

More recently, Dongarra has released the "highly parallel computing" benchmark. This benchmark was developed for medium-to-large parallel and distributed systems, and has now been tabulated on hundreds of computer systems [1, Table 3]. Unlike the basic Linpack benchmark, the scalable version does not specify a matrix size. Instead, the user is invited to solve the largest problem that he/she can reasonably run on the available system, given limitations of memory. Further, the user is not restricted to running a fixed source code, as with the single-processor version. Instead, one is free to use just about any reasonable programming language and parallel computation library, including assembly-coded library routines if desired.

A portable implementation of the highly parallel Linpack benchmark, called the High Performance Linpack (HPL) benchmark, is available. More details on downloading and running the HPL benchmark are given in Section 11.10.3.

During the past ten years, Dongarra and Strohmaier have compiled a running list of the world's top 500 computers, based on the scalable Linpack benchmark. The current listing is available from the URL `http://www.top500.org`. Interestingly, one of the top-ranking systems is the ASCI Red system at Sandia National Laboratory in Albuquerque, NM. The ASCI Red system is a Pentium-based cluster system, although not truly a "Beowulf" system, since it has a custom-designed inter-processor network. With a "Rmax" rating of 2.379 Tflop/s (2.379 trillion floating-point operations per second), it currently ranks number three in Top 500 list (based on the June 2001 listing).

The Linpack benchmarks are fairly easy to download and run. The calculation of performance, once a timing figure is obtained, is very easy. The principal advantage, however, of these benchmarks is that there is a huge collection of results with which one can compare—it is very easy to determine how one's system stacks up against other similar systems.

The principal disadvantage of the Linpack benchmarks, both single-processor and parallel, is that they tend to over-estimate the performance that real-world scientific applications can expect to achieve on a given system. This is because the Linpack codes are "dense matrix" calculations, which have very favorable data locality characteristics. It is not uncommon for the scalable Linpack benchmark, for example, to achieve 30% or more of the theoretical peak performance potential of a system. Real scientific application codes, in contrast, seldom achieve more than 10% of the peak figure on modern distributed memory parallel systems such as Beowulf systems.

8.4 The NAS Parallel Benchmark Suite

The NAS Parallel Benchmark (NPB) suite was designed at NASA Ames Research Center in 1990 to typify high-end aeroscience computations. This suite consists of eight individual benchmarks, including five general computational kernels and three simulated computational fluid dynamics applications:

EP An "embarrassingly parallel" calculation—requires almost no inter-processor communication.

MG A multigrid calculation—tests both short- and long-distance communication.

CG A conjugate gradient calculation—tests irregular communication.

FT A 3-D fast Fourier transform calculation—tests massive all-to-all communication.

IS An integer sort—involves integer data and irregular communication.

LU A simulated fluid dynamics application, using the "LU" approach.

SP A simulated fluid dynamics application, using the "SP" approach.

BT A simulated fluid dynamics application, using the "BT" approach.

The original NAS Parallel Benchmark suite was a "paper-and-pencil" specification—the specific calculations to be performed for each benchmark were specified in a technical document, even down to the detail of how to generate the initial data. Some straightforward one-processor sample program codes were provided in the original release, but it was intended that those implementing this suite would utilize one of several vendor-specific parallel programming models available at the time (1990).

The original NPB problem set was deemed the "Class A" size. Subsequently some larger problem sets were defined: "Class B", which are about four times as large as the Class A problems, and "Class C", which are about four times as large as the Class B problems. The small single-processor sample codes are sometimes referred to as the "Class W" size.

Since the time of the original NPB release, implementations of the NPB using MPI and also OpenMP have been provided by the NASA team. These are available at the URL `http://www.nas.nasa.gov/Software/NPB/`.

As with the Linpack benchmark, the NPB suite can be used both to measure the performance of a single node of a Beowulf system, or the entire system. In particular, the "sample" or "Class W" size problems can be easily run on a single-processor system. For a Beowulf system with say 32 processors, the Class A size is an appropriate test. The Class B problems are appropriate for systems with roughly 32–128 processors. The Class C problems can be used for systems with up to 256 CPUs.

Unfortunately, the NASA research team that designed and championed the NPB suite has now almost entirely left NASA (note that the author, one of the designers of the NPB, is now at LBNL). As a result, NASA is no longer actively supporting and promoting the benchmarks. Thus there probably will not be a "Class D" problem size, etc. Further, NASA is no longer actively collecting results.

However, the NPB suite continues to attract attention from the parallel computing research community. This is because it is fairly widely recognized that the NPB suite reflects real-world parallel scientific computation, to a significantly greater degree than most other available benchmarks.

We recommend that users of Beowulf-type systems use the MPI version of the NPB suite. Instructions for downloading, installing and running the suite are given at the NPB web site.

This chapter was processed by LaTeX on June 27, 2001.

References

[1] Jack Dongarra. Performance of various computers using standard linear equations software. Technical Report Computer Science Technical Report Number CS-89–85, University of Tennessee, Knoxville TN, 37996, 2001. `http://www.netlib.org/benchmark/performance.ps`.

Index