The GENESIS Distributed-Memory Benchmarks

C.A. Addison*, V.S. Getov, A.J.G. Hey, R.W. Hockney, I.C. Wolton
Department of Electronics and Computer Science
University of Southampton
Southampton SO9 5NH, U. K.

To appear in: J. Dongarra and W. Gentzsch (Eds.)
Computer Benchmarks, North-Holland

Abstract. The GENESIS distributed-memory benchmarks represent a significant step forward in the evaluation of distributed-memory MIMD systems. The initial version of this benchmark suite was augmented and modified in order to evaluate the SUPRENUM computer. This augmented suite has now been further modified to allow the benchmarks to be run on several important distributed systems without further code modification. This paper briefly describes the benchmark suite and outlines the methodology employed to run the benchmarks and interpret timing measurements. Some preliminary results from selected benchmarks running on the Intel iPSC/860 and the Meiko i860 Computing Surface are described and related to basic system parameters. The paper concludes with the possible future evolution of the suite.

1 Introduction

During the Esprit Project 2702 - GENESIS, a number of benchmark programs were developed to evaluate the performance of the German SUPRENUM distributed-memory (DM) computer compared to other parallel computers (including multiple-vector and shared memory architectures). Preliminary results for the so-called GENESIS pre-study have already been published in outline [7]. After modification and the addition of further codes the benchmark suite was used to assist in the evaluation of the SUPRENUM. More recently SUPRENUM-specific features have been removed from the code and the benchmarks now use the PARMACS macros [4] for code portability.

The GENESIS benchmarks are required to be written in standard Fortran77, with both a sequential (base) version and DM version being supplied. The distributed-memory versions are based on the use of the PARMACS macros which provide portability between several important machines. The normal arithmetic precision is 64-bits, although 32-bit codes may be included if there is good reason. Several of the benchmarks include a

*Centre for Mathematical Software Research, University of Liverpool, Liverpool, U. K.

version with explicit vector instructions expressed in Fortran90 syntax [14]. This version was important for the SUPRENUM computer and will be of further use as a basis for developing High Performance Fortran versions of the benchmarks.

Optimization of the benchmark codes is permitted – indeed encouraged – in order to show the full potential of the hardware, provided that the performance of the unmodified version is also published.

The Fortran77 sequential version of the benchmark defines the problem and algorithm to be used, and with minor modifications and tuning should run on vector computers and shared-memory (SM) parallel computers, using the currently available auto-vectorizing and auto-parallelizing (or auto-tasking) compilers. The DM version of the benchmark, on the other hand, is a fundamentally different program in which the work to be performed is explicitly distributed to multiple processors which may only communicate with each other by sending messages over communication channels using I/O (send/receive) statements. The translation of the sequential version to an efficient DM version without the help of software tools is usually a non-trivial task measured in man-months of effort. Currently a number of automatic software tools are being developed to ease this task, and it is expected that these will significantly reduce the labour involved. Nevertheless, significant human programming effort is still likely to be required. The value of the GENESIS benchmarks is that they reduce this work by providing distributed-memory message-passing (DMMP) programs for a number of important scientific problems.

Another important aspect of the GENESIS benchmarks is that they provide an approximate timing model which allows the variation of performance with both the problem size and the number of processors to be estimated. It is believed that this feature of the benchmarks will prove particularly valuable for predicting the performance of future computers, and in understanding how their performance scales.

A reader might well ask what is the justification for introducing yet another suite of benchmarks — after all the PERFECT Club [3], SPEC [17] and EuroBen [6] have already established substantial and growing suites of programs covering similar scientific applications. The reason is that none of the benchmarks mentioned above yet provide DMMP programs. Their programs are sequential and suitable for testing single CPUs, or SM multiple CPUs through the use of auto-tasking. They cannot be run on DMMP computers. The Numerical Aerodynamical Simulation organization at NASA Ames Research Centre has defined a set of 'pencil and paper' benchmarks [2] where the problem to be solved and the input data are specified but the benchmarker is free to choose the algorithm and may code these in either Fortran or C. Although this approach has merit, a considerable time is required to develop, debug and optimize such codes from scratch and as a result only application kernels can be treated in this way; no application codes are included. The only other DM benchmarks known to us are those from the CalTech Performance Evaluation Project (CPEP [13]), which indeed were assembled at about the same time as the GENESIS benchmarks and for similar reasons. Our work therefore has some overlap with that of CPEP. Their work covers, at present, a much wider range of computers (though there are no published results for more recent machines), but does not provide such a full timing analysis as that given here. At the present early state in the evolution of DM benchmarks we feel there is justification for this and other independent initiatives.

2 The Benchmarks

The GENESIS benchmarks have both sequential and DM versions written in standard Fortran77 supplemented by the use of the PARMACS macros [4] for message passing and task creation. Full documentation is supplied and in most cases includes a computation/communication analysis. The PARMACS macros developed at GMD and Argonne National Laboratories allow the same code to be run on a number of systems without modification of the code. These macros are currently supported on the Cray Y-MP, Intel iPSC/2 and iPSC/860, Meiko i860 and transputer based Computing Surfaces running CS Tools, nCUBE 2, PARSYTEC GC, SUPRENUM, networks of IBM RS/6000s and other UNIX workstations (on top of PVM) and networks of Suns (on top of CS Tools for Suns). Implementations for other machines are under development, these include an implementation for transputers that can be readily adapted to T9000 machines when they become available.

The benchmarks are divided into three levels according to the complexity/realism of the code.

1. *Synthetic code fragments*
   These are short codes designed to measure basic machine properties connected with vectorization, communication and synchronization.
2. *Application kernels*
   These codes are typically longer than the above and are the computationally-intensive kernels of a variety of different applications. These benchmarks are more complex to analyze but are more likely to come closer to modeling the behaviour of full applications programs.
3. *Full application codes*
   It is essential to verify whether the supposed 'application kernel' is a reliable guide to the performance of full applications programs running on parallel systems. Often for example there are non-parallelizable data distribution and collection phases in the full codes or perhaps the sheer size of real applications may result in memory and/or I/O bottlenecks. Thus the inclusion of full application programs is a necessary step in a complete evaluation of a parallel system.

The GENESIS benchmark suite contains the following codes in each category.

1. Synthetic code fragments:

   TICK1 Measurement of clock tick

   RINF1 Measurement of vectorization parameters r_∞ and $n_{1/2}$

   COMMS1 Unidirectional single message transfer ('pingpong' benchmark)

   COMMS2 Bidirectional exchange of two messages ('pingping' benchmark)

   SYNCH1 Barrier synchronization cost

2. Application kernels, or frequently used library subroutines:

    TRANS1 Transpose using naive algorithm

    TRANS2 Transpose using divide and conquer algorithm

    MATMUL1 Matrix multiply

    LU1 Matrix LU factorization

    QR2 QR decomposition using Householder transformations

    FFT1 One-dimensional FFT

    PDE1 Three-dimensional Poisson solver using parallel red-black relaxation

    PDE2 Two-dimensional multigrid Poisson solver using a range of fine and coarse grids

    QCD2 Conjugate gradient kernel from QCD

3. Full application codes:

    HELM1 Meteorology: pseudo-spectral Helmholtz solver

    HELM2 Meteorology: multigrid Helmholtz solver

    QCD1 Quantum chromodynamics (Monte-Carlo)

    MD1 Molecular dynamics (particle-particle)

    GR1 General relativity

    LPM1 Local particle-mesh (PIC–type code) simulation of an electromagnetic device

The current (May, 1992) release 2.0 of the suite contains the following benchmarks: TICK1, RINF1, COMMS1, COMMS2, SYNCH1, FFT1, PDE1, PDE2, QCD1, QCD2, MD1, GR1, LPM1. The codes are publicly available by anonymous `ftp` from Cornell University - `tesla.ee.cornell.edu`.

3 Methodology

The motivation behind the development of the GENESIS benchmark suite has been to assemble benchmarks for a range of important scientific and engineering problems, that could be used to evaluate the relative strengths and weaknesses of current DM machines and to enable predictions of performance on future architectures.

In assessing the strengths and weaknesses of a design we are not only interested in the optimum performance that can be achieved for a particular application, but also in the performance that can be achieved from the initial port of a code, the subsequent optimization that can be achieved, and how difficult it is to achieve this optimization. The last assessment must invariably be subjective and will depend on the skill of the benchmarker, as will the degree of optimization obtained. It is also important to assess how the performance varies with the number of processors for different problem sizes, in some cases optimum performance can only be achieved on a narrow range of problem sizes or perhaps for certain grid shapes in a grid based problem.

Many issues are raised in undertaking a comparative evaluation along the lines outlined above, not least of which is what should we adopt as an appropriate measure of *performance*. A few of these issues are discussed below, along with the practical guidelines adopted for running and interpreting the GENESIS benchmarks. More details are given in [1, 10].

The comparison of performance on different machines is an extremely complex and sensitive issue to handle properly, particularly because the basic characteristics of different DMMP computers differ widely. For instance, code that would deliver nearly optimal performance on a transputer array might deliver poor performance on SUPRENUM or the Intel iPSC/2. Similarly, if code written for the SUPRENUM exploited a complicated communications topology, performance on arrays of T8 transputers might be disappointing. As the GENESIS benchmarks are modified over time, it is hoped that it will become increasingly easier to obtain good performance from a wider range of DMMP computers.

The capability to estimate changes in benchmark performance with changes in system parameters such as communications start-up time and bandwidth is provided by the timing formulae which have been developed for most of the benchmarks. The timing formulae also allow performance of future systems to be estimated.

The following guidelines have been established for the running of the benchmarks and for interpretation of the results obtained.

1. Benchmarks generally should be run on a quiet system with only one user; this includes the host machine if communication with the host is a part of the benchmark. The equivalent of elapsed time should be measured.

2. The fundamental measurement made in any benchmark is the elapsed time to complete some specified task. All other performance figures (e.g. Mflop/s) 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 characterizing 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.

3. When a benchmark represents only a portion of an application code, task creation time must not be included in overall benchmark times. In this case, the total time is defined to be the maximum processor time measured after an initial synchronization of all tasks.

4. Benchmarks should not only measure the time of major modules, but whenever possible, they should also give the number of "useful" floating point operations (written flop) performed within that time as well. Mega-flop rates (written Mflop/s to avoid ambiguity, and to be consistent with the normal SI practice for naming units) can then be determined, which help to measure processor utilization. Times should be presented along with Mflop/s because the latter alone are open to abuse too easily.

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 [3]. 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 [15] 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

Two types of operation count are distinguished; 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. No credit is given for redundant operations in calculating benchmark performance.

5. Test results must be reproducible so that different evaluators can produce similar results. This has implications for routines that use random number generators.

6. Two general absolute measures of performance will be used:
the *temporal performance*, R_T, defined as the inverse of the elapsed time

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

and the *benchmark performance*, R_B, defined as the nominal standard floating-point operation count for the benchmark, divided by the elapsed time

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

The performance variation is studied as the number of processors increases with a fixed problem size, and as the problem size increases for a fixed number of processors. Scale-up, the performance when the problem size is scaled proportionately to the number of processors is of secondary importance and can be computed from sets of runs where either the problem size or number of processors is fixed.

7. The presentation of performance in terms of relative measures, such as speed-up or efficiency, are strongly discouraged as these can obscure detail, and lose all knowledge of the absolute value of of the computer's performance.

As mentioned earlier, optimization is actively encouraged, but the optimizations (and the resulting changes in performance) should be recorded. The idea of recording the time each optimization took to implement was viewed as desirable, but impractical. Optimizations may alter a program's performance so that similar, but not identical, results may be obtained. It is therefore important to have some easy way for a tester, who is not the developer, to see if a version is acceptably accurate. If each benchmark has a small set (say fewer than 10) of values that can be compared among different versions of the program, then a reliability test can be made automatically by computing the relative difference of these values when compared with a baseline version. A benchmark run would be acceptable only if these differences were all smaller than pre-defined thresholds.

Benchmarks must be specified by a low dimensional parameter space. This is necessary to obtain a complete picture of a benchmark's performance on a target machine with a "small" number of test runs. A benchmark would be run at pre-determined values within this parameter space in order to obtain a clear picture of how performance changes with varying parameters (such as problem size or number of processors). As part of the specification, the developer must also indicate the legal ranges for these input parameters, including how these parameter ranges are inter-related.

Standards pertaining to lower level evaluation details have also been established, such as the availability of machine readable timings to a particular number of significant figures, the use of particular performance metrics for comparisons and graphical display standards. The intention is that the results are presented in a form that allows readers to examine the data and make their own interpretations.

4 Use of the GENESIS Benchmark Suite in Evaluating Parallel Systems

The first release (1.0) of the GENESIS benchmark suite containing a subset of the present suite was used to evaluate the following wide range of machines with both SM and DM architectures for the GENESIS pre-study in June 1989:

SM Vector Supercomputers : ISIS, Cray-XMP, ETA-10

SM Multiprocessor : Alliant FX-8

DMMP with scalar nodes : nCUBE-1, Supernode, Meiko, iPSC/2

DMMP with vector nodes : iPSC/2, Amtek/Symult 2010

The results of this evaluation are contained in the final report of the working group [8] and summarized in [7]. The pace of change in parallel computation is such that all these machines have now been superseded as have some of the conclusions drawn at the time. However, the GENESIS benchmark suite has allowed us to monitor the changes in performance of different systems as technology has advanced.

One of the conclusions was that, at the time of study, a single vector supercomputer node could outperform DMMP systems on large vectorizable problems. DMMP systems with vector nodes were not impressive, rarely delivering more than 1 Mflop/s per vector node. The development and application of the benchmark suite during 1990 for the evaluation of the SUPRENUM machine and possible GENESIS architectures, demonstrated that supercomputer performance could be achieved with DMMP architectures. The best result was obtained for matrix multiplication with 870 Mflop/s from 64 SUPRENUM nodes on a large optimally shaped matrix. A highly optimized code for quantum chromodynamics achieved over 200 Mflop/s on 64 nodes. A two dimensional version of the red-black relaxation Poisson solver gave 264 Mflop/s on 64 nodes, whilst the multigrid Poisson solver ran at 160 Mflop/s.

Usually a high degree of code optimization is required to achieve such performances. The evaluation exercise confirmed a number of anticipated strengths and weaknesses of the SUPRENUM architecture. In particular, the fast access from the vector floating-point unit to vector cache memory but the long vector start-up time, the high communications bandwidth within a cluster but the long message start-up time. The poor start-up times meant that high performance could only be achieved on problems with a coarse granularity and by code using relatively few, long messages. Moreover, to achieve optimum performance it was necessary to make extensive use of non-portable SUPRENUM Fortran features. In rewriting the codes on the basis of PARMACS macros during 1991 most of the specific SUPRENUM optimizations have been lost, but it has still been possible to obtain good performance from the i860-based DM computers on which measurements have so far been made, without any optimization for the specific system.

4.1 Preliminary results on i860 systems

Caveats:

1. The results presented are from a short initial evaluation. There is still much scope for optimization of the codes and these results should be considered as representative of the performance that can be obtained from the first naive port of a code, rather than the optimum benchmark performance on the systems. A fuller evaluation is in progress.

2. The quoted benchmark measurements are for code based on the PARMACS macros, which introduces an extra overhead to communications above that incurred with the native communication calls. This extra overhead actually shows the *price of portability*, which one should pay in order to have a portable program across different DM computers. It can be measured with the communications benchmark, and the timing formulae supplied with the benchmarks allow estimation of the deterioration in performance due to this aspect of the use of PARMACS.

3. These results are presented as an example of how the benchmarks can be used to evaluate the performance of different systems. This performance is influenced by the compiler and message-passing software as well as the machine architecture. As software is being continually upgraded the results presented may not be representative of current system performance.

Table 1: Best single-node performance in Mflop/s on the Meiko MK096 and the iPSC/860 for various benchmarks

Computer System	FFT1	QCD2	PDE1	PDE2	QCD1	GR1
Meiko MK096	6.2	9.7	11.9	5.0	5.7	5.7
iPSC/860	4.7	6.7	4.6	4.0	3.4	2.4

The benchmarks have been run on two i860-based systems: the Intel iPSC/860 and the Meiko i860 Computing Surface. The Portland Group compiler was used on the iPSC/860, whilst the Greenhills compiler in association with the Pacific-Sierra vectorizer VAST-2/i860 was used on the Meiko. The iPSC/860 results were obtained on two different installations. the 8-node system at Southampton University Parallel Applications Centre and the 64-node system at Daresbury Laboratories. The Meiko measurements were made on a small 8-node demonstrator machine available to the GENESIS project which was supplied with the latest MK096 board design [5].

Good performance on a DM benchmark can only be obtained if the basic code of the computation performs well on a single node. Hence the first step in the evaluation was to measure the performance of the sequential code on a single node. In general it was found that the combination of the VAST-2 vectorizer with the Greenhills compiler on Meiko MK096 boards gave significantly better performance than the Portland compiler on the iPSC/860. Best performance was of course dependent on the problem size being large enough to allow efficient vectorization. Without vectorization the Portland compiler and iPSC/860 had the edge on performance. Table 1 shows the best vectorized single node performances obtained on both machines for benchmarks where results are available to make a comparison. Generally the maximum performance per node will be less than the best single-node value, and in some cases considerably less. An important factor which reduces performance per node for distributed programs is the overhead associated with communications from one node to another. The success of a DM computer depends on keeping the message start-up time low and providing a high asymptotic bandwidth between processors. If this is not done, one can easily find that time of execution is completely dominated by message communication time.

4.2 Communications Benchmark (COMMS1)

This benchmark measures the basic communication properties of a computer network by performing the 'pingpong' experiment between a neighbouring pair of nodes [9]. A message of varying length is sent and returned after the data has become available to the receiving user program. Half the time for this 'pingpong' exchange is recorded as the time to send a message from one node to a neighbour. Analysis of this time as a function of message length allows the calculation of the start-up time and the asymptotic bandwidth for a single message exchange between a neighbouring pair of nodes.

Because of the inconsistent terminology used to describe communications protocols (e.g. synchronous and asynchronous are often used with different and contradictory mean-

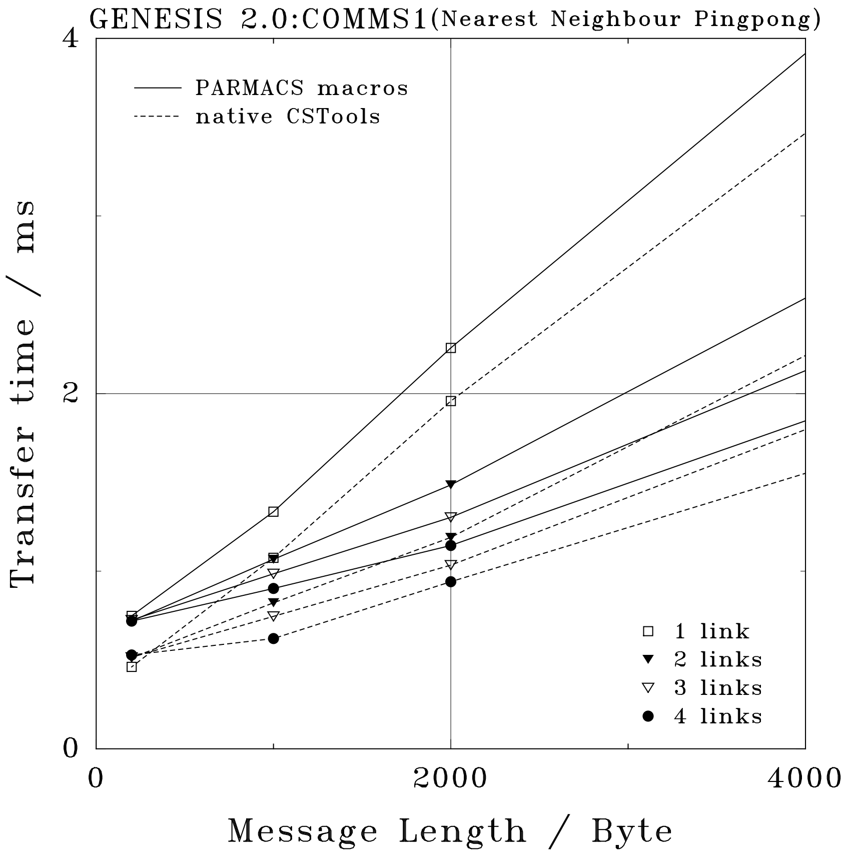

Figure 1: The time for a single message transfer, as a function of the message length on the Meiko MK096 Computing Surface

ings), it is necessary to state the communications language primitives that are used to implement the sending and receiving in the communications benchmark. In the PARMACS macros version (SEND) and (RECV) were used, in the native Intel version (CSEND) and (CRECV) were used, and in the native CS Tools version (CSNTX) and (CSNRX) were used.

The Meiko MK096 node architecture employs two transputers to handle communications between nodes. The configuration language used to load a distributed program allows the user some freedom to specify the number of transputer links connecting pairs of nodes. The message transfer time depends on the message length, the number of links and the level of implementation (PARMACS macros or CS Tools - see Fig. 1). Table 2 shows measurements of the asymptotic communications bandwidth and start-up time for both PARMACS macros and native CS Tools versions of the COMMS1 benchmark as a function of the number of transputer links connecting neighbouring processors. The overhead introduced by the PARMACS macros on the Meiko MK096 mainly effects the message start-up time. This overhead varies from 61% to 26% depending on the number of transputer links. The reduction of asymptotic bandwidth due to the the PARMACS macros implementation is about 10%.

On the iPSC/860, for messages longer than 100 byte, the communications parameters between neighbouring nodes from PARMACS macros were measured as 213 μs start-up time and 2.80 Mbyte/s asymptotic bandwidth. However, for messages of 100 byte or less

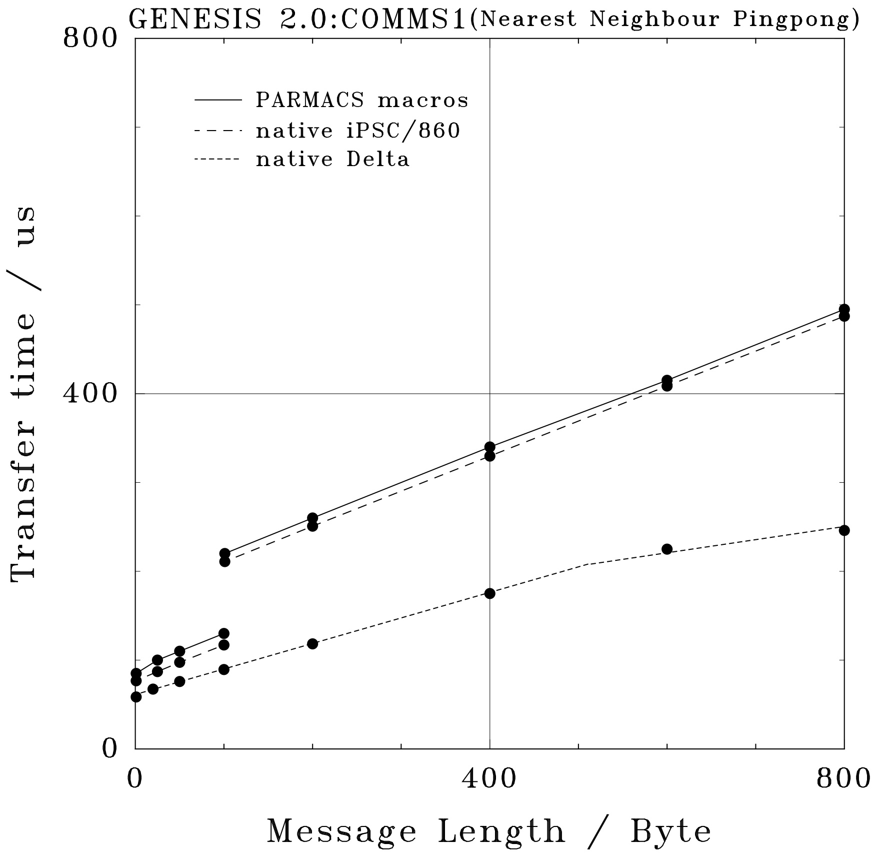

Figure 2: The time for a single message transfer, as a function of the message length on the iPSC/860 and the Touchstone Delta

the start-up time was only 85 μs (see Fig. 2). Using native Intel communications [11] the start-up time on the iPSC/860 was measured as 205 μs for long messages and 76 μs for short messages. These results show a relatively small overhead of about 4-10% in the communications start-up time for the PARMACS macros implementation on the iPSC/860. No difference was measurable in the asymptotic bandwidth. Fig. 2 also contains measurements on the machine using the native CSEND and CRECV primitives. Note that there is no longer a discontinuity in transfer time at 100 byte on the Delta machine, but the communications bandwidth is discontinuous at 512 byte.

Table 2: Asymptotic communications bandwidth in Mbyte/s and start-up time in μs as a function of the number of transputer links between processors on the Meiko MK096

Number of links	PARMACS macros		Native CS Tools	
	Bandwidth	Start-up	Bandwidth	Start-up
1	1.18	572.4	1.31	360.3
2	2.20	649.7	2.41	466.0
3	2.87	664.8	3.22	485.4
4	3.56	666.9	4.11	528.6

The most prominent feature of our present results is that the message start-up time on the Meiko using the PARMACS macros is significantly greater than for the iPSC/860. Both measurements were made from the PARMACS macros version of the benchmark but the extra overhead incurred in using the PARMACS macros, rather than the native communications calls, is significantly greater on the Meiko than the iPSC/860. The increased overhead with Meiko's CS Tools can be attributed to the relatively high level at which the PARMACS macros are implemented on top of CS Tools.

4.3 Conjugate Gradient Solver for QCD (QCD2)

This benchmark constitutes the kernel of SU(3) lattice gauge simulation with dynamical Kogut-Susskind fermions. The kernel consists of solving a large sparse system of linear equations by conjugate gradient iteration. This forms the core of several important algorithms for lattice gauge theory with fermions, and typically accounts for about 80% of the total execution time of a full application. The program provides for up-to three of the four dimensions of the space-time lattice to be distributed. Communications are required between nearest neighbours in each processor grid dimension, a global summation and a broadcast are also required at each iteration. Virtually all of the code is vectorizable.

Table 3 shows performance measurements for the distributed version of the benchmark on the Meiko. Table 4 shows the corresponding results on the iPSC/860. Figure 3 plots performance against number of processors, for different lattice sizes on both the Meiko and the iPSC/860.

Table 3: Performance results on the Meiko MK096 in Mflop/s for the QCD2 benchmark

Probl. size	2 Processors	4 Processors	8 Processors
4×4^3	8.14	6.47	6.34
16×4^3	11.8	13.6	16.9
4×8^3	14.6	18.8	-
8×8^3	15.3	21.1	35.4
16×8^3	14.9	22.8	40.7
32×8^3	-	-	43.8

Table 4: Performance results on the iPSC/860 in Mflop/s for the QCD2 benchmark

Probl. size	2 Proc.	4 Proc.	8 Proc.	16 Proc.	32 Proc.	64 Proc.
4×4^3	7.352	9.290	10.49	12.55	14.93	20.93
16×4^3	8.699	13.06	18.08	25.34	33.13	42.06
4×8^3	10.81	17.68	27.08	39.31	55.26	71.18
8×8^3	10.91	18.30	31.18	47.97	70.98	102.7
16×8^3	11.08	18.45	32.34	53.25	85.43	129.8
32×8^3	11.10	18.91	32.73	55.14	92.60	156.4
16×16^3	-	-	42.14	75.44	135.1	243.7

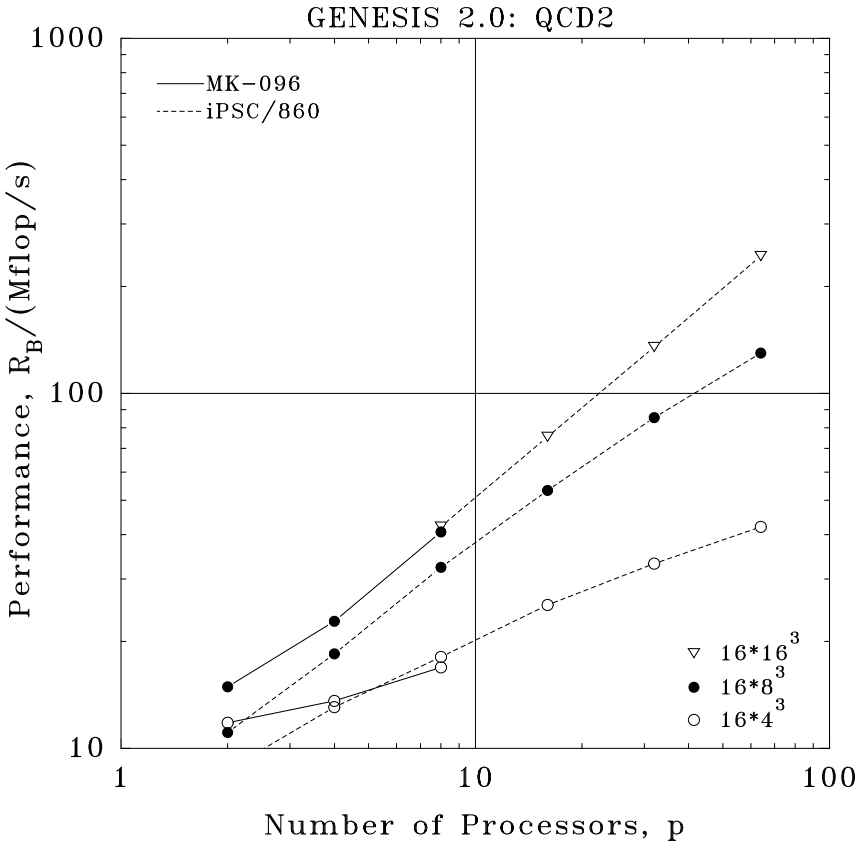

Figure 3: Performance of the Conjugate Gradient QCD Benchmark as a function of the number of processors, p, for different lattice sizes, N

The benchmark performance scales well with the number of processors as long as the problem size is large enough to permit good vector performance on each processor. This is evident in Figure 3 where performance on the iPSC/860 for the largest lattice, 16×16^3, increases linearly with the number of processors within the range of measurements. For smaller problem sizes a distinct flattening of the performance curve is evident as the number of processors is increased. This flattening is due to a reduction of the vector performance on each processor and to the increased relative importance of communications overheads as the local problem size on each processor decreases. As yet it has not been possible to run the benchmark on a large Meiko machine to investigate the scaling of performance with the number of processors.

Although a near linear increase in benchmark performance with number of processors can be achieved for suitable problem sizes, the gradient of the curve is limited by the overheads of communication. The effect of communication bandwidth could be examined on the Meiko by varying the number of transputer links connecting processors. By altering the standard configuration file produced by the PARMACS macros the minimum number of transputer links between processors could be increased from 2 to 3 to give an increase in asymptotic bandwidth from 2.20 to 2.87 Mbyte/s – see Table 2. With this optimization the performance for the 16×8^3 lattice on four processors was increased from 22.8 to 25.4 Mflop/s; further improvements can be expected from more substantial increases in bandwidth.

The performance of this benchmark could probably also be improved by increasing the degree of overlapping of communications with computation. The amount of overlap had to be decreased in the SUPRENUM version of the benchmark to allow the code to be run at all. It would be worth reverting to the previous communications structure for less temperamental machines. Further optimization in the basic single node performance could be obtained by optimizing vector shift routines used in the code. A combination of these optimizations should considerably increase the maximum performance (16×32^3 lattice size) of 325.0 Mflop/s obtained on 64 nodes of the iPSC/860. As an illustration of what can be achieved, it should be noted that a performance of over one Gflop/s has been reported for a highly optimized QCD code running on 64 Meiko i860 nodes in Edinburgh.

4.4 General Relativity Application (GR1)

This benchmark solves a system of hyperbolic PDEs, derived from general relativity, which describe the evolution of a gravitational field from an initial state. Although conceptually similar to solving the wave equation the equations are long and complicated. This application solves the axi-symmetric problem to reduce the problem to manageable size. Solution of the general problem requires three orders of magnitude more computer power and is likely to become of substantial interest as more powerful parallel machines are developed.

The present structure of the benchmark requires the problem size to scale with the number of processes, so results for constant problems sizes on different number of processes can not be presented. However, in this case the performance does not appear to be strongly dependent on the problem size within the measured range. Table 5 contains performance results for the GR1 benchmark on the Meiko MK096, while table 6 shows performance measurements for the distributed version of the benchmark on the iPSC/860. There is some loss of performance in going from the sequential version to the distributed code, but subsequently there is almost perfect speedup for up to 32 processors, with a performance of approximately 1.25 Mflop/s per node on the iPSC/860 and approximately 3 Mflop/s per node on the Meiko (up to 8 processors).

Table 5: Performance results in Mflop/s on the Meiko MK096 for the GR1 benchmark

1 Processor		4 Processors		8 Processors	
Probl. size	Performance	Probl. size	Performance	Probl. size	Performance
101×16	4.38	101×4	10.94	101×8	20.40
151×16	4.37	151×4	12.88	151×8	24.70
201×16	-	201×4	14.10	201×8	27.41
251×16	-	251×4	14.83	251×8	29.19

Table 6: Performance results in Mflop/s on the iPSC/860 for the GR1 benchmark

1 Processor		4 Processors		16 Processors		32 Processors	
Probl. size	Perfor-mance	Probl. size	Perfor-mance	Probl. size	Perfor-mance	Probl. size	Perfor-mance
101×16	2.43	101×4	4.97	101×16	20.4	101×32	40.5
151×16	2.16	151×4	4.76	151×16	19.4	151×32	38.9
201×16	2.11	201×4	4.82	201×16	19.5	201×32	39.1
251×16	2.14	251×4	4.78	251×16	19.5	251×32	39.1

5 Conclusions

The use of the PARMACS macros for task creation and message-passing within the benchmarks enables us to run the same basic benchmarks on many of today's leading distributed-memory machines without modification of the code. However to obtain optimum performance on a given system further optimization will always be required. The basic PARMACS version of the benchmark can be seen as measuring the performance obtainable from a first implementation of a code on a system. Subsequent optimization will then reveal how much further performance can be improved.

The effectiveness of benchmarks based on the PARMACS macros is dependent on the efficiency with which the PARMACS macros can be implemented on top of the message-passing layer of the system in question. This is illustrated by the situation with CS Tools where the current PARMACS macros implementation incurred significant overheads in message start-up time.

The latest release 2.0 of the benchmark suite has been used to make a preliminary evaluation of the iPSC/860 and Meiko i860 Computing Surface. Some initial observations can be made on the basis of the measurements obtained:

1. Good single node performance is essential to obtain high overall performance on the distributed benchmark. On the Intel i860 processor efficient vectorization is the key to achieving good performance from Fortran code. The combination of the Greenhills compiler with the VAST-2 vectorizer on the Meiko MK096 board generally gave superior single-node performance over the Portland compiler on the iPSC/860 for large problems.

2. The measured communications bandwidth between neighbouring nodes from PARMACS macros was similar on both machines, but the current overhead associated with using PARMACS macros on top of CS Tools results in a higher start-up time for the Meiko.

3. Both machines are currently not well balanced on most of the GENESIS benchmarks as regards speed of communication versus speed of computation. This shows up in relatively poor performance speed-up for multi-node measurements when compared with the single-node results. Improved communications on the next generation of machines should do much to improve this imbalance.

4. The preliminary results are for codes that have not been optimized in any way for the system under evaluation other than by use of compiler options. It is felt that significant improvements in performance on both machines can be achieved without investing substantial amounts of time.

5. In this preliminary evaluation, too few processors were available on the Meiko to estimate how well benchmark performance scales for large numbers of processors.

Evaluation of any computer, particularly a DMMP computer, requires careful, intensive work. It is our hope that the benchmarks that have evolved from the GENESIS project will make this task considerably easier.

6 Future Development

Version 2.0 of the GENESIS benchmark suite was released in May 1992. Since the original release of benchmarks from the GENESIS pre-study a further ten codes have been added to the suite and several of the original codes have been substantially improved. All distributed benchmarks are now based on the PARMACS macros.

Continued evolution of the suite is planned to take place in a new benchmarking venture called RAPS (Real Applications for Parallel Systems). The RAPS consortium plans to port a number of important production applications onto parallel computers with physically distributed memory. Current benchmarks have established that DMMP can achieve high performance on both compute-intensive kernels and on relatively compact research applications. As yet, however, it has not been adequately established that similar high performance can be maintained with large production codes. The RAPS consortium intends to address this issue.

The size of such codes means that it will only be possible to port a few representative production applications, so these will be complemented with a number of compact applications to increase the coverage of applications areas. There is also a need for a set of kernels that can assist in understanding and relating performance on the applications to the basic system parameters. While the production applications are being ported an initial version (0.0) of the RAPS benchmark suite was released in June 92. The existing GENESIS benchmarks suite are supplemented by a shallow water code for meteorology from the European Centre for Medium Range Weather Forecasting (ECMWF) and a general block-structured multigrid environment for PDE solution in two dimensions (LiSS package), developed by the Gesellschaft für Mathematik und Datenverarbeitung (GMD).

The initial RAPS 0.0 benchmarks are based on the PARMACS macros, however the development of international message-passing standards will be monitored and considered for adoption. The April 93 release of the RAPS benchmark suite will consist of the RAPS 0.0 codes plus the following new production codes:

FIRE This is a fluid flow code used in particular in automotive flow simulations. It was developed by AVL in Austria who are currently collaborating with BMW to produce a future parallel production version of the code.

IFS This is a global atmospheric simulation code which can be used for many different applications in meteorology and climatology. The benchmark code will be developed by ECMWF.

PAMCRASH This is an explicit nonlinear finite element code mainly used for the simulation of car crash situations in the design process for car bodies. The code is marketed and maintained by Engineering Systems International (ESI) and is currently used by many leading car manufacturers.

GEANT This code was developed at CERN to simulate the interaction of high-energy particle showers with detectors.

While release 0.0 codes are public domain benchmarks the production codes cannot be made public for commercial reasons. These codes will be made available to computer vendors under a restricted arrangement supervised by the independent software company PALLAS.

Current work on the GENESIS benchmark suite is concentrated on developing High Performance Fortran versions of the codes. The future development of the suite is planned to take place in coordination with other parallel benchmarking initiatives in the US and Europe.

References

[1] C. Addison, J. Allwright, N. Binsted, N. Bishop, B. Carpenter, P. Dalloz, D. Gee, V. Getov, A. Hey, R. Hockney, M. Lemke, J. Merlin, M. Pinches, C. Scott and I. Wolton. The GENESIS Distributed-Memory Benchmarks. Part 1: Methodology and General Relativity Benchmark with Results for the SUPRENUM Computer, *Concurrency: Practice and Experience*, **4**, to appear (1992).

[2] D.H. Bailey, J. Barton, T. Lasinski and H. Simon. The NAS Parallel Benchmarks, Technical Report RNR 91-002. NAS Applied Research Branch. NASA Ames Research Center, (1991).

[3] 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 and J. Martin. The PERFECT Club Benchmarks: Effective Performance Evaluation of Supercomputers, *Intl. J. Supercomputer Appls.* **3**(3) (1989) 5-40.

[4] L. Bomans, D. Roose and R. Hempel. The Argone/GMD Macros in FORTRAN for Portable Parallel Programming and their Implementation on the Intel iPSC/2, *Parallel Computing*, **15**(1-3) (1990) 119-132.

[5] Dual i860 Element (MK096), *Information & Product News for Meiko Users*, **2** (1992).

[6] A. Friedli, W. Gentzsch, R. Hockney and A. van der Steen. A European Supercomputer Benchmark Effort, *Supercomputer 34*, **VI**-6 (1989) 14-17.

[7] A.J.G. Hey. The GENESIS Distributed-Memory Benchmarks, *Parallel Computing*, **17**(10-11) (1991) 1275-1283.

[8] A.J.G. Hey and C.J. Scott. Report of the State-of-the-Art and Evaluation Work-package. Esprit-2 Project P2447 (GENESIS pre-study), June, (1989).

[9] R.W. Hockney. Performance Parameters and Benchmarking of Supercomputers, *Parallel Computing*, **17**(10-11) (1991) 1111-1130.

[10] R.W. Hockney. A Framework for Benchmark Performance Analysis, *Supercomputer 48*, **IX**-2 (1992) 9-22.

[11] R.W. Hockney and E.A. Carmona. Comparison of Communications on the Intel iPSC/860 and Touchstone Delta, *Parallel Computing*, **18**, to appear (1992).

[12] R.W. Hockney and C.R. Jesshope. *Parallel Computers 2: Architecture, Programming and Algorithms*, Adam Hilger/IOP Publishing, Bristol & New York, (1988).

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

[14] M. Metcalf and J. Reid. *Fortran90 Explained*, Oxford Science Publications/OUP, Oxford and New York, (1990) Chapter 6.

[15] 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.

[16] Proceedings 2nd International SUPRENUM Colloquium, 30 Sept.-2 Oct. 1987, Bonn, FRG. Special Issue (U. Trottenberg ed.), *Parallel Computing*, **7**(3) (1988).

[17] SPEC Benchmarks Suite Release 1.0, *SPEC Newslett.* **2**(3) (1990) 3-4. publ. Systems Performance Evaluation Cooperative, Waterside Associates, Fremont, California.