Defence Science Journal, Vol. 54, No. 3, July 2004, pp. 361-377

Numerics of High Performance Computers and Benchmark Evaluation of Distributed Memory Computers

H.S. Krishna and K.P. Singh
Aeronautical Development Agency, Bangalore-560 017

ABSTRACT

The internal representation of numerical data, their speed of manipulation to generate the desired result through efficient utilisation of central processing unit, memory, and communication links are essential steps of all high performance scientific computations. Machine parameters, in particular, reveal accuracy and error bounds of computation, required for performance tuning of codes. This paper reports diagnosis of machine parameters, measurement of computing power of several workstations, serial and parallel computers, and a component-wise test procedure for distributed memory computers. Hierarchical memory structure is illustrated by block copying and unrolling techniques. Locality of reference for cache reuse of data is amply demonstrated by fast Fourier transform codes. Cache and register-blocking technique results in their optimum utilisation with consequent gain in throughput during vector-matrix operations. Implementation of these memory management techniques reduces cache inefficiency loss, which is known to be proportional to the number of processors. Of the two Linux clusters-ANUP16, HPC22 and HPC64, it has been found from the measurement of intrinsic parameters and from application benchmark of multi-block Euler code test run that ANUP16 is suitable for problems that exhibit fine-grained parallelism. The delivered performance of ANUP16 is of immense utility for developing high-end PC clusters like HPC64 and customised parallel computers with added advantage of speed and high degree of parallelism.

Keywords: Bandwidth, benchmark, cache, communication link, fast Fourier transformation, granularity, iteration, Linux cluster, latency, multigrid, Mflops, machine parameters, memory, PIM, matrix, refinement, distributed memory computers, high power scientific computations, parallel iterative method

1. INTRODUCTION

The two emerging fields of the twenty-first century, computational fluid dynamics and DSP, most researched in recent times, demand machines which must achieve Tflops performance and provide extensive memory storage' of about 1 GB/h. Often, the desired accuracy in numerical simulation of fluid flows may well be less than 10^{-8}. In the design of digital filters, it is desirable to represent data in the largest word length or bits to minimise the quantisation effect, which is proportional to the number of bits of finite length register. High performance computers, in particular, multiple instruction multiple data (MIMD) parallel machines meet these requirements to a great extent and provide a cost-effective solution to the problem. The selection of a computing facility suitable for a particular application depends on memory bandwidth and computing power or speed. Over the years, improvement in speed of computers

Revised 8 April 2003

has been largely due to the development of pipeline architecture, vector processing, supercomputing, parallel and massively parallel computing progressively in that order. Software parallel programming concepts, such as virtual processors, reduction operation, strip mining, and dependency analysis are important efficient parallelisation techniques which have largely resulted in improvement in hardware utilisation through hierarchical memory structure and memory management. The hardware architectural consideration also plays a crucial role in the implementation of these techniques as for instance, in fast Fourier transform (FFT) computations. Vector architecture requires codes of long vector length, whereas RISC/cache-based architecture requires small data length for efficient cache reuse. Today's high power computing resources[2] for distributed memory machine's include parallel numerical software, such as highly tuned linear algebra kernels-BLAS, LAPACK, PBLAS, ScaLAPACK, and ATLAS; parallel iterative method (PIM) solvers; and communication libraries like MPI-LAM, Petsc, Essl and Pessl (for IBM machines).

Benchmarking[3] of computers is useful in estimating performance of an application program on a particular architecture, exploring relative computing capabilities of several computer systems, determining their portability and architectural limitations. The choice of suitable benchmark depends on the available library. Parallel benchmarks should evaluate both fine- and coarse-grained parallelism. In addition, parallel benchmarks should apply the aforementioned programming concepts to actually simulate the process of computation for realistic or delivered performance.

In this paper, exhaustive benchmark tests that include standard, application, kernel, and synthetic benchmark codes are presented for distributed memory parallel computers. These codes find complete bound on computing time of algorithms, measure computational complexity, memory bandwidth and operation count independent of machine or programming language, either through recursive looping to increase computational intensity or through data reuse. Wall clock time (get_time_of_the_day in ns or us) is the single most important parameter for timing all operations.

The computing platforms benchmarked for performance comprise five workstations- DecAlpha, Linux, SunUltral, IrisS, and Iris2; a serial computer IBM RISC1 RS 6000 and four distributed memory parallel computers-PACE32, ANUP16, HPC22 and HPC64. Some special features of workstations are DecAlpha's various levels of optimisation to increase the speed of computation only at the expense of accuracy and IrisS's SDRAM, which is expandable up to 4 GB for extensive storage application.

2. DISTRIBUTED MEMORY PARALLEL COMPUTERS

2.1 ANUP16 Hardware Architecture

It is a customised parallel machine made up of Linux pentium III processors. Its cluster-based MIMD parallel architecture allows scalable computing. The nerve centre of this machine is the Fast Ethernet Intel 510T switch that connects 16 PCs in a cluster as shown in the Fig. 1. Some of the special features of this LAN technology switch are scalable stacking technology at reduced up-front cost, redundancy of stacking modules to eliminate single-point failure for switch-to-switch connections, stacks up to seven switches, 10/100 Mbps throughput per port in 24-port switch, and backplane scalable performance up to 14.7 Gbps (2.1Gbps/switch X 7) bandwidth. Communication is by the message passing between the master and the slave processors or nodes, and among nodes themselves. The master processor is also backed by the 200 GB Intel server. It supports MPICH parallel library and a defacto standard library called Anulib developed at the Bhabha Atomic Research Centre, Trombay. A monitor connected to slave nodes through sharer gives the status of processes

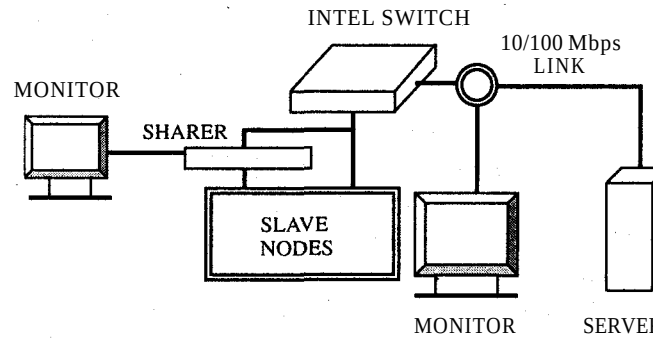

Figure 1. Block diagram of Linux cluster

The Machar code available online at Netlib repository and Berkeley's Spara program have been used for diagnosis of machine parameters. The single floating-point precision numbers for all the 32-bit machines mentioned here were found to be F(2, 24, -126, 128, T). From this, one may interpret underflow and overflow to be 1.1754951 X 1Q-^{38} and 3.402823466 X 10^{38}, floating-point precision (eps or machine epsilon) of 1.1920929 X 1Q-^{7} or machep (quantisation step size of digital filters), the relative spacing between the machine number (exact) and its succeeding number as 10~^{23}. In short notation, the double floating-point precision numbers for all computers are F(2,53, -1022,1024, T).

The other machine parameters of interest to computational fluid dynamics code developers, machine zero and clock granularity or resolution as determined from programs[3] Machar and Tickl respectively have been listed in the Table 1.

Machine zero is a measure of compiler accuracy and clock granularity is the smallest time interval measured by the clock. Machine zero is required to set error bounds for residual convergence of CFD codes. The apparent resolution of the clock as measured by Ctest of Perftest bench suite and Tickl is 0.95 us. The timings of elementary arithmetic operations[7] for all high performance computers are tabulated in Table 1. The time, T in nano seconds for an assignment (=) operation is shown in italics and all other arithmetic operations are expressed as the ratio of assignment operation. These values are useful in calculating Mflops (T~'/10^6) from operation count of the test code. Table 1 also shows values of vectorisation parameter /?$_M$ and memory-access bandwidth for read (/?) and write (\V) operations in MB/s. These parameters determine intranode communication rate and its granularity.

Table 1. Machine parameters of high performance computers

Parameter	ANUP	HPC	DEC	LINUX	SUN	IRIS2	IRIS3	RISC1	PACE
Processor/ model	P-III Katrnai	P-III C.M.	ST30	P-II Deschutes	Ultra Spare	MIPS R4400	MIPS R 10000	RS 6000	SPARC 10HS
Year	2000	2002	1999	1999	1993	1992	1993	1988	1995
Clock (MHz)	550.0	1O02.0	500.0	448.05	200.0	250.0	195.0	41.6	60.0
Cache (KB)	512	256	2048	512	512	512	512	256	256
RAM (MB)	512/256	2/1GB	512	505	512	128	128	64	- 64
Compiler (C)	GCC 2.96	GCC 2.96	Digital UnixV4	GCC 2.7.2.3	SunOS 5.5.1	MIPS Pro	MIPS Pro	Version 1.3.0.43	Sun R 4.1
EPS ,10"	2.22044	2.22044	2.22044	2.2204	2.2204	2.2204	2.2204	2.2204	2.2204
CKG.10^5	0.01298	0.9468	0.119	0.01362	0.119	1.0728	1.0728	1.0728	0.119
MZ.10^{20}	5.4201	5.4201	11102.2	5.4201	11102.0	11102.2	11102.2	11102.0	11102.0
flc.	142.0	195.0	32.0	75.0	58.0	31.0	81.0	46.0	-
R/W(MB/s)	440/260	790/475	327/469	369/281	62/68	116/123	54/51	-	-
+Linpack, (Mflops)	48.6	56.3	69.3	35.3	8.9	16.6	26.3	23.2	-
+MGSOR (s)	10.25	5.63	13.0	14.48	33.5	51.9	-	81.2	77.0
Operations (=)TimeT.10'°s = : + : - : / : *	81 1:2:2 :25:2	46 1:2:2 :26:3	3 1:28:28 :222:28	54 1:4:4 :49: 44	504 1: 1: 1 :2:1	40 1:4:4 :36:8	48 1:2:2 : 13:2	252 1:2:3 :4:3	604 1:2:2 :2:2

*Double precision, + Optimisation, P: Pentium, EPS: Machine epsilon, R/W: Memory read/write

4. THROUGHPUT

Iterative convergence and mesh refinement form the basis of most computational fluid dynamics solution techniques. CFD codes employing time-dependent technique involve large number of iterations for the solution to converge and attain steady-state values. While grid generation technique is aimed at obtaining smooth and ever so fine grid, the purpose of flow solver development is to compute flow variables in minimum time through faster iterative convergence of solution.

The difference between successive stages of approximation of a physical variable, which is usually pressure or density in CFD computations, is called the residual. The solution by FDM, FEM or FVM methods of governing differential equations of fluid flow, heat transfer, structural dynamics, control systems, computational electromagnetics (CEM), electric circuits and aeroacoustics, is obtained either by reducing or by factoring into linear system of the form $Ax = b$, where A is the global coefficient matrix, x is the solution vector, and b is the global matrix of source terms, which is zero in most cases. Invariably, such linear systems are readily solved by well established linear algebra matrix methods. These methods function as convergence acceleration devices to speedup the computation process by driving the residual to machine epsilon or zero. All linear systems solution methods fall under the following two categories: (i) multi-level method and (ii) iterative methods.

Multigriding is the most popular multi-level method because it is also the fastest iterative convergence technique known. Its lower bound complexity is *O(N).* For illustrating the efficacy of this method, Fivol program[8] has been chosen. Fivol is a freeware finite volume elliptic grid generator that solves Laplace equation in cartesian coordinates. It applies SOR technique for smoothing the solution. This code is modified to work on a 2-level multigrid V-cycle. The bar chart in Fig. 3 shows the relative speeds or work in Mflops of the various machines as computed by Fivol-multigrid (FivolMV) program for iterative residual convergence to the same floating-point precision of 10^{-8}. A number of multigrid codes are available at Mgnet website, and the execution

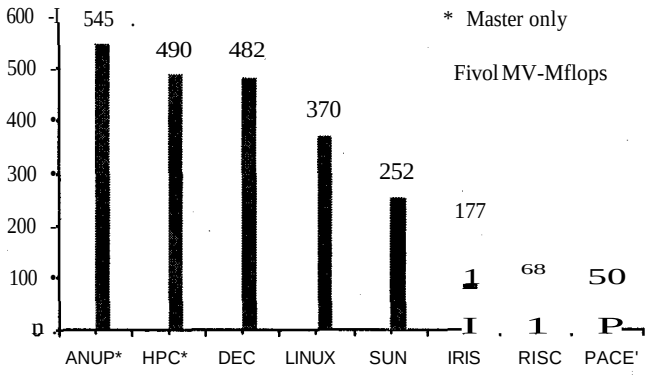

Figure 3. Computing power of various types of high performance computers.

time for one such code, Mgsor is presented in Table 1 for 2-level V-cycle multigrid with SOR smoothing on non-red-black grid. It compares the relative speed of the machines in serial computation.

Iterative methods applied to linear algebra problems are broadly classified into direct or stationary methods and non-stationary or parallel iterative methods[9](PIM). In direct methods, matrix ordering determines solution time and storage requirement, and cost depends on the order of matrix storage. The popular direct iterative method of solution of linear systems like block Jacobi, LU, Cholesky, and variants of Gaussian methods are suitable for small problem sizes, require more storage space and execution time when compared to parallel iterative methods.

The throughput in Mflops as a function of process grid of LU factorisartion routine, as obtained from Scalapack test suite run are shown in Table 2 for HPC64. It shows that the effect of domain or matrix decomposition on throughput in fine-grained parallelism, parallel iterative methods, more commonly known by the generic name as Krylov solvers, are preferred for large-scale problems that involve sparse matrices computations. Laplacian and Poisson elliptic solvers for incompressible flow solution in aircraft aerodynamics, implicit Euler unsteady and time-dependent solution, implicit Navier-Stokes methods, all lead to the solution of sparse matrix.

For over a decade now, at the Innovative Computing Laboratory, University of Tennessee, USA, Dongarra's team[10] has carried out intensive research in sparse linear algebra to develop parallel preconditioned

Table 2. Throughput of ScaLAPACK LU routine for HPC64

PxQ	1x64	64 x 1	8x8	4x16	4x4
100x100	6.1	1.5	5.4	9.2	17.1
500x500	36.9	7.96	32.6	57.2	127.0

for parallel iterative method and smart libraries for partial automatic choice of iterative method and preconditioners, load-balancing techniques based on matrix structure, since it often reflects the physical structure and changes in differential coefficients, and block structuring of matrices based on balancing number of rows or zeros for enhanced performance. Their effort culminated in the development of three widely used linear algebra packages, particularly suited for dense matrices: Liftpack foi*vector computers, Lapack for shared-memory computers, and ScaLAPACK for distributed-memory machines. In all these packages, the non-zero elements of sparse matrix are stored by indirect addressing through pointers, either in compressed row storage format or compressed column storage format. To obtain good load balance, usually square block scattered or block-cyclic decompositions is employed. Block partitioning of matrices ensures efficient cache and data reuse.

The BLAS2 linear algebra package ($O(N^2)$) implements these techniques in a highly tuned manner. The structured matrices are more rapidly solved than the unstructured ones. Standard test matrices may be obtained online from Matrix Market. Since each node of distributed memory cluster computers have gigabyte memory storage, handling of sparse matrices is not a constraint. Storage or retrieval of data is in column-major order for Fortran and row-major order for C in stride one input/output operation.

The prominent Krylov solvers for CFD applications based on non-stationary methods, such as conjugate gradient, GMRES, and QMR, generate a sequence of orthogonal vectors of residuals during each iteration. Minimisation of gradients of these quadratic functional residuals (by least-square technique in GMRES and QMR) yields solution of linear systems. In recent times, Newton-Krylov-Schwarz method"'[13] has been applied in nonlinear implicit CFD solver for full potential and 3-D time-dependent Euler computations. PIM optimisation[14] procedure comprises quadtree data structure adaptation to improve locality of reference, overlapping nearest neighbour communication with computation, excepting inner product and norms that induce global synchronisation, A perfect or linear speedup is the upper limit to parallel scalability while that for numerical scalability is given by linear relationship between computing power and problem size. Together, these ensure that optimum speedup or perfect scalability is obtained without sacrificing numerical accuracy. Scalability then implies that the solution time be constant when both the problem size and the number of processors increase in a fixed ratio. This is illustrated in Fig. 4 for the execution time obtained from PIM Krylov solver, Psparslib[15] test run on HPC64 and ANUP16 process grid (P ' Q) for two problem sizes of 100 ' 100 and 200 ' 200 on each processor. Further confirmation of scalabilty is shown in Table 3 for typical CFD problem of over million mesh points, class B bench test run of NAS parallel benchmark-LU factorisation routine. Both ANUP16 and HPC64 scale linearly up to 16 nodes with constant high value of throughput of about 40 Mflops per processor that is comparable to any of the other high performance

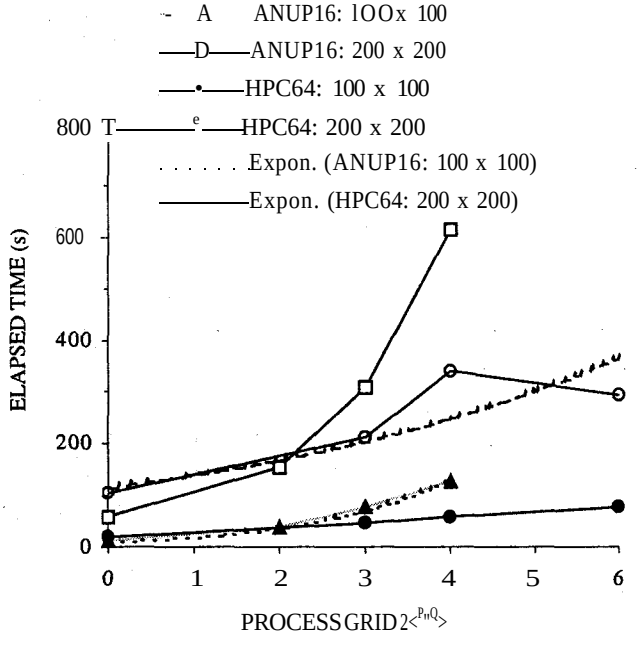

Figure 4. Parallel and numerical scalability

Table 3. Throughput of NAS-LU benchmark

P	Elapsed time (ET)	Speedup (S)	Efficiency (E)	Mflops/P
ANUP16, class B, 102^3, 250 iterations				
1	11402	1.0	-	43.8
2	5877	1.9	97	43.0
8	1612	7.1	88	38.7
16	863	13.0	82	36.1
HPC64, class B, 102^3, 250 iterations				
1	14173	1.0	-	35.2
2	7246	1.9	98	34.4
8	1756	8.1	100	35.5
16	991	14.0	89	31.4
32	1234	11.0	36	12.6
64	593	24.0	37	13.1

computers, such as Intel Paragon, CRAY, etc. But beyond 16 nodes, machine utilisation of HPC64 is poor as evidenced by sharp decline in efficiency *(E);* computation-to-communication ratio (CC); and speedup (5). The difference between Fig. 4 and Table 3 for high-end HPC64 can be reconciled by observing that the former is a coarse-grained problem whereas the latter (LU) is fine-grained problem. The parallel iterative methods (PIMs) are preferred to multi-level methods due to their modular nature that enables these to be used as black box routines[16] with little or no modification to the existing programs. Convergence histories of PIMs aid as stopping criterion if iteration count could be predetermined for a given error bound or machine epsilon. The number of iterations for uniform convergence or rapid convergence depends on the Krylov subspace dimension and the preconditioner used.

In any computational fluid dynamics computation, it is always desirable to seek finer resolution of grid in the flow field region in order that an accurate representation of the flow region be obtained, more so in the regions where large gradients in the flow variables are expected, such as that occurring at the leading and trailing edges of airfoil, flow at the convex corner, etc. But computational power in terms of memory and speed of the computer, time, and cost put a limit on the resolution of the grid. The effect of refinement is illustrated by general unsteady Euler solver (GUES) code run for NASA F3 forebody at Mach number 1.7. GUES is a 3-D time-marching, finite-difference code based on MacCormack's predictor-corrector explicit scheme for flow field solution in cylindrical coordinates *(z, Q, r).* The log-linear plot of throughput versus grid points in Fig. 5 shows stark contrast for axial (Gl, G2, and G3) and azimuthal (G4 and G5) refinements, mainly because of linear topology and that communication dominates in cross-flow direction. The machine-code dependencies require that isogranularity plots be linear for perfectly scalable

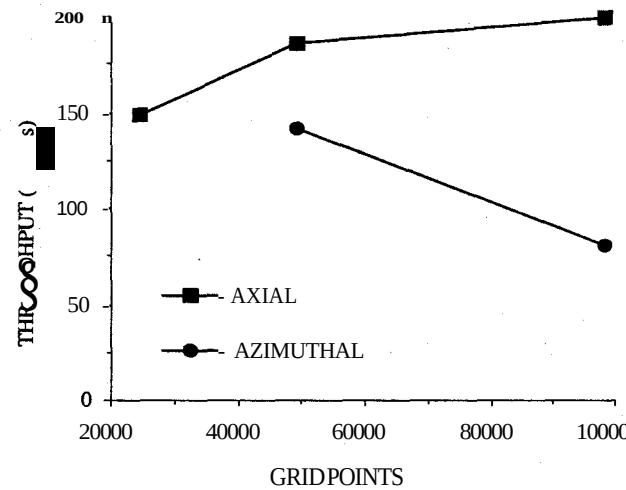

Figure 5. Effect of grid refinement (PACE32)

code. This stipulation is satisfied by low-end machines (up to 8 nodes) only in Fig. 6.

A linearly scalable program on various architectures ensures ease of portability. The parallel balance point for PACE32 is located at 16 processors. The delivered performance of PACE32 is about 10-20 per cent of its peak performance. Benchmark based on execution times and throughput have been refuted of late, as psuedo work measures by Gustafson in his HINT commercial benchmark. He contends that the performance should be quantified in terms of absolute measure- quality improvement per second (QUIPS): The performance curve, on which HINT

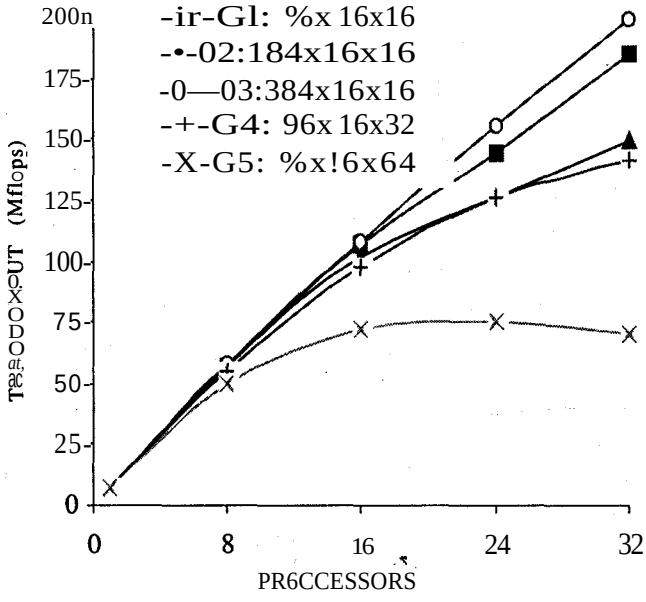

Figure 6. Isogranularity plot for scalable performance

measurement is based, has resemblence to time-processor product curve (measure of goodness of parallel implementation) and also the learning curve of a neural network.-

5. MEMORY MANAGEMENT

The programs and data that reside in the main memory are often shuffled to vary the amount of memory in use. The data itself may be copied to other memory locations, such as cache, registers, auxiliary, and external storages by the operating system softwares. In any case, the locality of reference determines the speed of access to data stored on a particular location. The various test suites used for benchmark study in Table 4 and the block diagram of processor components in Fig. 7 present an overview of the bench test plan.

5.1 Hierarchical Memory

The memory of ANUP16 processor ranging from fast but small capacity to slow but large capacity storage units consists of registers (CPU), cache, main memory (RAM), and external storage in magnetic tapes (4 mm). The cache of ANUP16 is located between CPU and the main memory (RAM) and its access time is close to processor logic clock cycle time. It stores program under

Table 4. Standard benchmark codes used in performance evaluation

Code	Output	Component test
Genesis	Low-level metrics:	
(i)Polyl,2, Rinfl	Rm R~.^{n}i/2	Cache
(ii) Tick 1 & 2	Timer error	CPU
LLC suite		
(i) Cachebench	Memory bandwitdth	Cache, memory
(ii) Mpbench	Bandwidth, MPIcalls	Internode link
BLAS suite		
(i) Sblat 1,2, 3	Iteration count	Memory, cache
FFT-0216	ET(s) and FFT	Cache, memory
CMU TP suite	ET(s) and FFT	Cache, memory

execution and temporary data in use. The access time ratio between the cache and the main memory varies from 1 to 7 as reported in the literature.

Hierarchical memory offers the advantage of highest average speed. Prior to execution of a code, parameters necessary to select the best program transformation in looping operation are unknown. Under these circumstances, advantages of spatial and temporal locality of references are explored to improve program performance. Also, array dimensions may be subject to alterations during run time. Efficient utilisation of memory hierarchy by the compiler by minimising cache miss, page fault, or altering main memory access pattern are known to play an important role in reducing execution time substantially. The most effective method to deal with the problems that often arise in linear algebra computations, has been the blocked-memory access or strip mining in large vector operations. Blocking exploits, to a high degree, the memory hierarchy. Many present-day high performance computers are designed to allow reuse of data transferred to faster level of memory hierarchy, as often as possible, before these are restored to slower level. In matrix computation, a (n x m) matrix is decomposed or partitioned into pair-wise disjunct matrices or submatrices. These submatrices are then copied into contiguous memory area in the highest memory hierarchy, cache. These

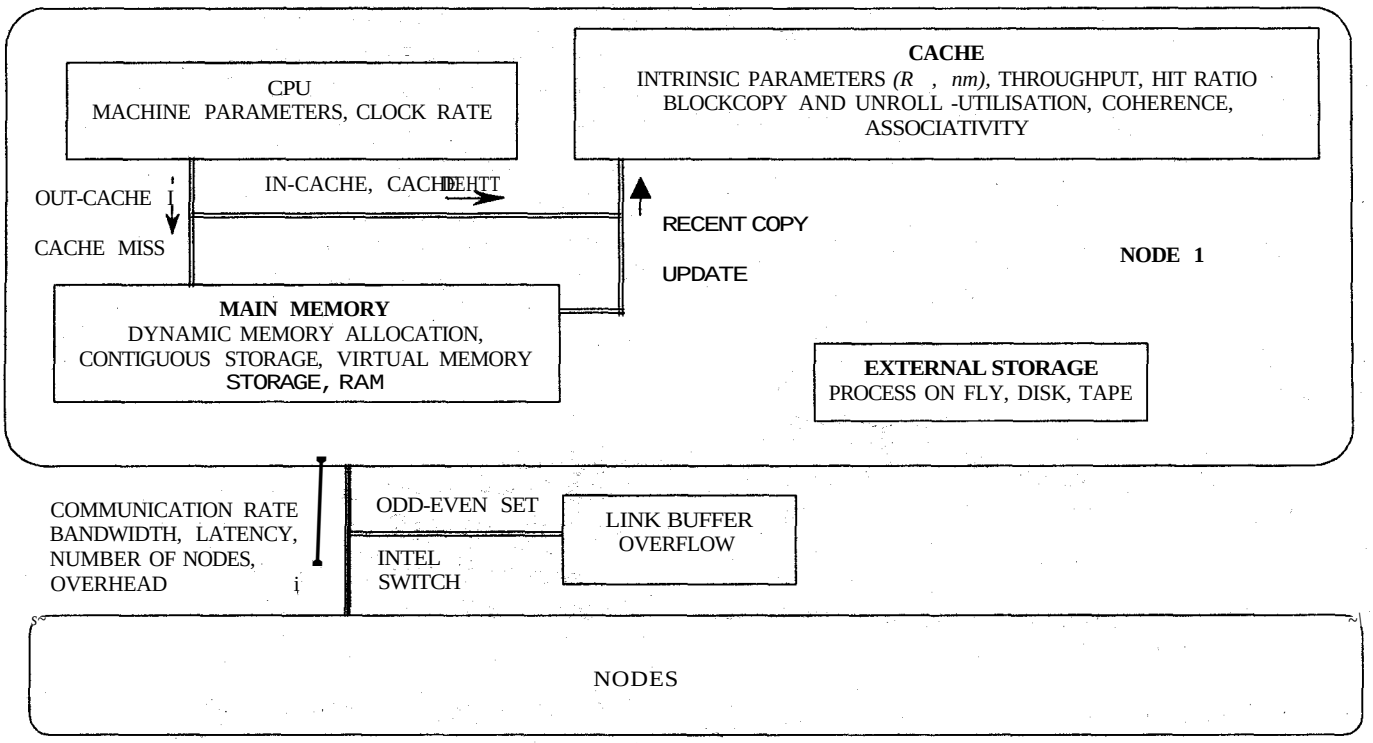

Figure 7. Block diagram of processor components and their functionality

operations may be further hastened by operand registers that store programs as well as intermediate results. An example of software implementation of this idea is the block-copying algorithm[6] for matrix multiplication of matrices $A_{.k}$ and B_{kJ}, which is given below :

```
do i, =1, n, nb
 do i2 =1, n, nb
  do j{ = z'2, MIN(n, i2+nb - 1)
   doj2= it, MIN(n, i^nb - 1)
    A(j-i2+l, ;2-z,+1) = A(jy 7,)
    do kt = I, n, nb
     dok2 = kv MlN(n, k^+nb -1)
       do £3 = iy MIN(n, i2+nb -1)
       do k4 = iv MlN(n, it+nb -1)
       C(k4, k2) = C(k4, k2) +
                   A(k3- i2+i. kr ;1+1) *
                   B(ky k2)
enddo
```

The ANUP16 execution time for the above algorithm computed by synthetic code BLKCP is shown in Fig. 8. As the block size *nb* is decreased from 200 to 100, there is a large drop, about one-fifth of the computation time, and any further reduction in *nb* results in little or no saving in computation time. In general, this is true, for empirically determined good block sizes are found in the range 32 to 256. Block size depends on the computation-to-communication ratio of the system problem size, hence it is identified as a tuning parameter. Small caches must transact with small block sizes, usually less than 8 KB, to increase cache hit ratio.

Another efficient loop transformation method, that is both machine- and compiler-dependent, is loop unrolling in reduction operation on a vector. It may be recalled that unrolling outer nested loops increases locality of reference. The code segment that implements loop unrolling of dot product of two matrices is given below:

$$C_{i+d,j+d} - \mathcal{C}_{i+d,j+d} + A_{i+d,k} * B_{kj+d}; \quad .7 = 1, \text{m} > u;$$

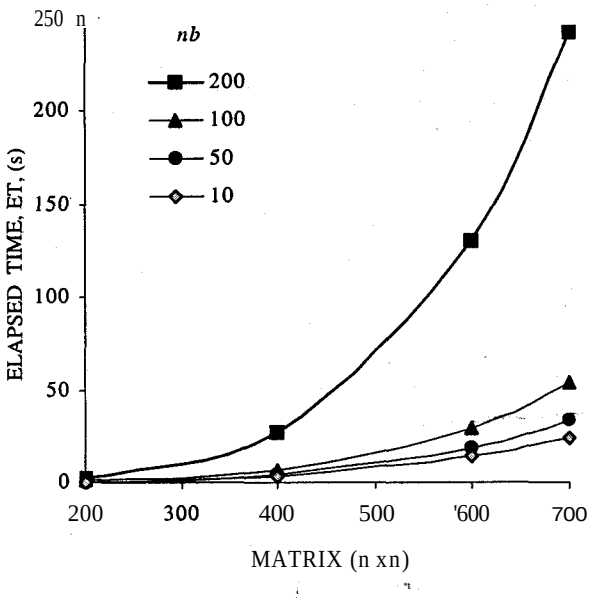

Figure 8. Blocking copying by varying block size *nb* of submatrices.

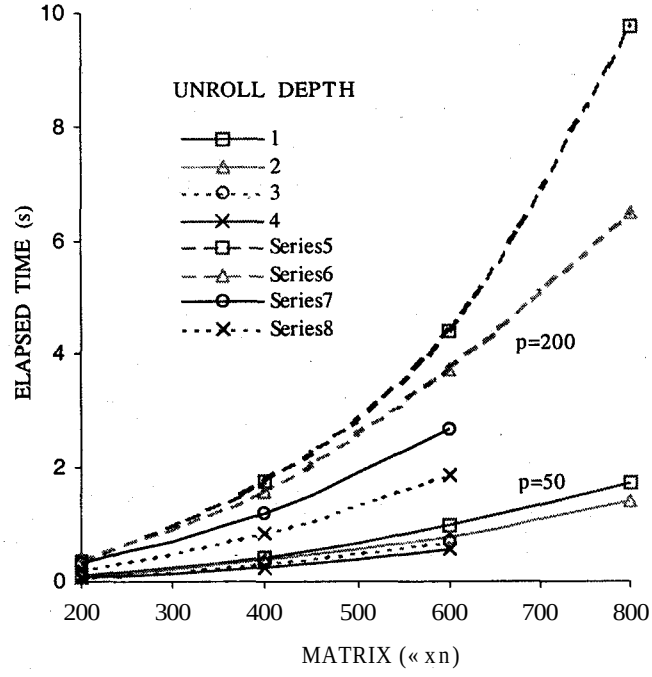

Figure 9. Loop optimisation by increasing depth of unroll

$i = I, n, u$; $k = I, p, I$; $d = 0, 1$; $u = 1, 2, 3, 4$.

The elapsed time (ET) for square matrices (m = n) are graphed in Fig. 9. For a given innermost block sizep, the computation time reduces uniformly as the depth of unroll u is increased from 1 to 4. Loop interchanges by permuting *ijk* variants in six different ways show deterioration in performance when the matrices (data) exceed cache size (n>200) or main memory (n> 1600).

Blocking and unrolling techniques have been implemented in the standard numerical library BLAS level 2 and BLAS level 3 and LAPACK. BLAS level 2 and BLAS level 3 is a collection of linear algebra subroutines that implements various levels of memory hierarchy. The BLAS level 1 subroutine *Saxpy* carries out completely unblocked vector-vector operation and has computational complexity *O(ri).* The BLAS level 2 subroutine *Sgemv* performs matrix-vector operations (double-nested loop) of $O(n^2)$ complexity and the BLAS level 3 subroutine *Sgemm* performs matrix-matrix operations (triple-nested loop)of complexity <9(n^3). SBLAT level 2 and SBLAT level 3 programs regulate the amount of data processed during each run so that large problems run for fewer iterations to maintain the run time for each problem size nearly constant. A test run of SBLAT suite, required 3460 and 17496 computational calls by subroutines *Sgemv* (BLAS level 2) and *Sgemm(BLAS* level 3) respectively on both ANUP and HPC computers. Computational electromagnetics (CEM) technique based on frequency domain approach of method of moments solves discretised integral equation of impedance matrix Z and unknown vector of current amplitudes / for one or more excitations of voltage, V *(ZI = V).* In the determination of radar cross section of aircraft by method of moments, linear algebra kernels of BLAS level 2 and BLAS level 3 are used to speedup the solution of the above matrix equation.

5.2 Cache Utilisation

As already noted, reuse of data greatly improves processor performance. Performance losses in parallel computers grow proportionately as the number of nodes, and therefore, efficient cache utilisation becomes a vital problem. The characterisation[17] of cache memory is based on the assumption of linear timing model and infinite bandwidth. The two important intrinsic parameters related to the machine performance, $R^{\wedge}$ and $F_{\,,,}$ (or $n_{\,\_\,}$) due to Hockney[2-17] give a

measure of pipeline effect and scalar computation. The R_n is the throughput in Mflops of the machine obtained for an infinite or excessive long calculation and n_m is the computational intensity at half of its sustained performance. A single parameter may be defined as the ratio of these two measures to obtain intranode communication time as $t_{nm}=(F+F_{l/2})/R_{aa}$. Two programs, Polyl and Poly2 of Genesis bench suite, evaluate a polynomial by Homer's rule to determine in-cache and out-cache performance. These two programs differ in only one respect. The coefficients of the polynomial in Poly2 are ten-times larger than those in Polyl so that the former tackles a larger problem of size in excess of cache memory. Figure 10 shows a comparison of these performances for both the master and the slave nodes of ANUP16. The negative gradient of t_c (shown by thick lines in Fig. 10) suggest that the intrinsic performance parameter /?_ increases with computational intensity since the two parameters are inversely related to each other. HPC, however, with marked steeper gradient has higher $/?_M$ value compared to its counterpart ANUP, which implies that HPC yields better performance for large- or coarse-grained problems.

A C-language benchmark code Cachebench of Llcbench test suite with dynamic memory allocation and compiler optimisation was test run on both ANUP 16 and HPC22 to determine the memory access patterns. The bandwidth in MB/s for compiler optimised read and write test cases have been compared in Fig. 11 with that of distributed memory computer, CRAY T3E. The greater bandwidth exhibited for write operations by CRAY while executing small vector lengths ($< 2^{15}$) is due to its constant data reuse. The read operation of ANUP 16 shows larger bandwidth (MB/s) than the write operation because of prefetching of data. Moreover, the write operation is plagued by replacement policy (write-through and write-back) and write buffering. The inadequecy in bandwidth of HPC is made up, to a large extent, by its faster clock rate, a space-time apportionment. The intrinsic performance parameters generated by Rinfl code of Genesis test suite and tabulated in Table 5 corroborates the explanation of Fig. 10.

All DSP techniques employ the FFT kernels.

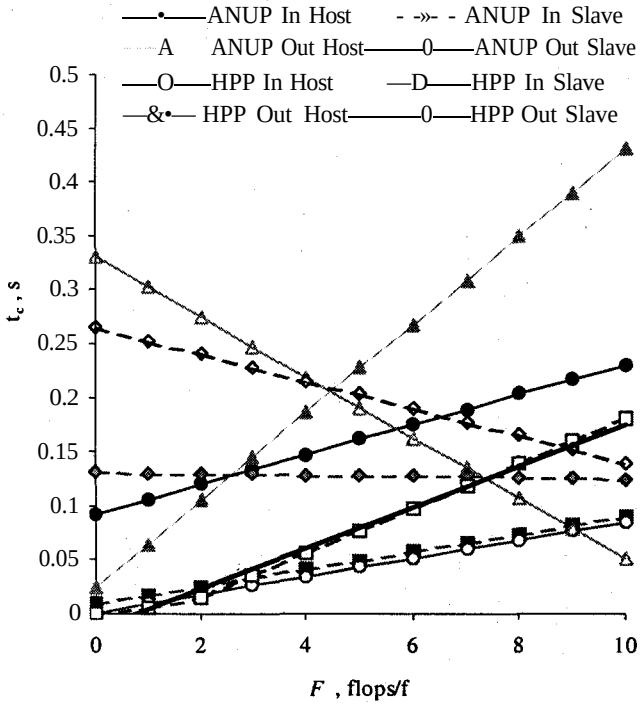

Figure 10. Intranode communication time

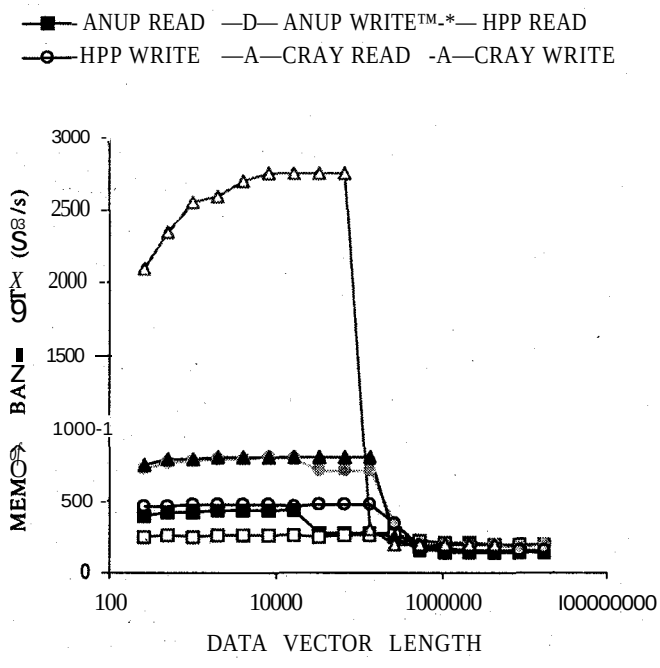

Figure 11. Memory read and write operations

Table 5. Selected values of intrinsic vectorisation parameter /?_ of MIMD computers

ANUP MASTER			ANUP SLAVE			HPC MASTER			HPC SLAVE		
Length	7L,	R_a	Length	&.	R_n	Length	/Z»	R_n	Length	/L,	R_a
<80	142	21.1	<100	100	10	<80	109	20.5	<80	110	28.3
>300	24.7	204	>500	11.8	206	>300	25.9	111	> 300	27.9	111

The FFT butterfly graph is the basic block for counting and bitonic sorting networks. The FFT method of solution of Poisson equation on hypothetical parallel random access machine(PRAM) is the fastest method with asymptotic lower bound complexity of 0(logAO computations. Ever since Cooley-Tukey proposed a numerical solution of FFT in the mid-sixties, many faster and efficient methods have been developed. The decimation in time Cooley-Tukey algorithm first reorders input data bit reversal method and then performs twiddle factor multiplication. The faster Tukey-Sands decimation in frequency algorithm reverses these two computational steps. These algorithms require n-passes through data to compute *2n* points. A more efficient Radix-8 algorithm requires *n* passes only. Bailey[18] gave a 4-step algorithm in which a (nl x *nl)* matrix of stored data undergoes n_l simultaneous $n_2$point FFT prior to multiplication by twiddle factor, then transposes to *n2* x nl matrix, and finally performs «2 simultaneous *nl* point FFT. As an improvement of this algorithm* a 4-step architecture-adaptive parallel algorithm that requires only three passes was suggested . In this method, the matrix transposition, which is communication-intensive, is globally adapted to interconnection network of parallel computers and the phase rotation or twiddle-factor scaling is locally adapted to a single processor for full cache utilisation. The most efficient FFT algorithm known todate is the optimal bit reversal COBRA algorithm[19]. To perform FFT on large data file, array dimensions too must be large, and during matrix transposition and bit reversal, performance degrades due to limited, small memory cache. Cache associativity depends on computer architecture and Murphy's law requires that cache lines be distinct between source and destination for the same associativity set. As a feasible solution to this problem, wide cache has been used in modern microprocessors.

The CMU benchmark TPSUITE is a collection of FFT, radar, and imaging software. Its FFT1 code applies Bailey's 4-step algorithm to exploit task parallelism by performing FFT through matrix transpose, scaling, and butterfly computation. The timing for these steps were found to be 49 per cent, 16 per cent, and 34 per cent of the total computing time, respectively, for the data vector length considered above. The same FFT 1 code is architecturally adapted for HPC22, as explained in previous paragragh, to perform computation in parallel. Figure 12 shows the throughput in Mflops (= $5Mog_2W$ / ET) for Cooley-Tukey, FFT1 serial and parallel, and Radix-8 FFT codes. The cache data reuse is clearly evident for Radix-8 algorithm because of unrolling.

Particle tracing[20], a scientific visualisation technique, applies cell caching method to reduce number of cache misses. The field values within each cell are defined by cell-local interpolating function. The vertices of the cell are copied into small buffers

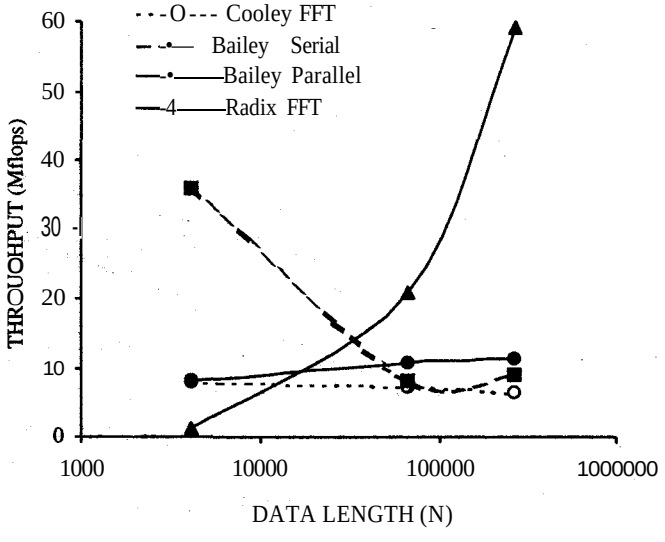

Figure 12. Cache utilisation by FFT codes

Table 6. Elapsed time/iteration for coarse-grained problems

Parallel computer	Node P	Euler code	ET(s) GCM 131x51x113	ET(s) GCC 51x25x67
PACE32	8	Original	37.6	4.54
	8	Modified	32.5	-
	16	Original	32.1	3.33
	16	Modified	20.0	-
HPC22/ (HPC64*)	8	--	7.35(7.10*)	0.55(1.74*)
	16	--	6.13(5.42*)	0.41 (1.52*)
ANUP16	8	~	6.69	0.91
	15	~	5.50	1.14

to support the fly calculation of flow variables. The position and vector field samples are loaded into contiguous memory locations in column-major order (Fortran) for caching the data. This method has been applied to the successive-lerp interpolation algorithm and the factored version of volume method. Also in virtual memory computers, the small data length is preferred for reasons mentioned above.

5.3 Buffer Overflow & External Storage

Buffer is a low capacity storage device connected to the communication links. It temporarily stores data when the links are busy. A large problem involving intensive computation may cause buffer overflow and congestion in the links. Single instruction multiple data (SIMD) machines like PACE32 can be economically utilised in terms of computational time when it is required to tackle problems with data parallelism. A data-parallel job permits single operation to be applied to all the elements of its data structure simultaneously. The odd-even process-exchange algorithm given below is used to minimise the time taken in internode communication across overlapping boundary planes in aircraft multiblock Euler solution[4].

do parallel np = 1, P

iodd=MOD(np, 2); ieven = 1 - iodd;

if (ieven.eq. 1) then

odd nodes send data and even nodes receive

else

even nodes send and odd nodes receive

enddo parallel

The amount of data transferred across overlapping node boundary is 314 KB, which far exceeds the PACE32 communication buffer capacity of 256 KB. At any given instance, every node either sends or receives the data but not both, thereby reducing the data transferred to the link. It may be observed in Table 6 that there is a reduction of about one-third of elapsed time per iteration for modified code that incorporates odd-even process exchange when compared with that of the original program. A similar problem arises in the FDM/FVM solution of Euler/potential flow equations that use central differencing scheme. The values at grid points in the stream-wise direction change sign alternately, which is referred to as odd-even decoupling. Here again, the odd and even numbered sets of grid points may be solved for parallel implementation.

External data storage devices, such as magnetic tapes and magnetic disks are used to store large volume of waveform data signals or images to compute Fourier transform. In real-time multimedia information systems, huge data in uncompressed form must be sorted on the fly to prevent loss of information content upon compression. Such data can be processed on the fly by Singleton algorithm[21] which computes FFT in two stages of bit-reversed computing pass and permutation pass. The ANUP16 parallel computer has a 4 mm tape drive with a speed of 900 in/s and maximum access time of cassette of less than 4 min. A built-in routine of system calls, a make file of command line arguments or environment variables in C to perform FFT on the fly is necessary to run Fourfs code[21]. As a test run, this code took 12.5 s for data file of 0.25 MB size. The conversion from ASCII to binary data file format usually results in 30 per cent reduction in size (Lempel-Ziv algorithm). The database standard, such as CGNS reduces overhead costs arising from file translation and multiple data sets in various formats and also aids in information retrieval.

CGNS has been implemented[22] in commercial CFD codes- NUMECA, CFL3D, OVERFLOW, WIND, PLOT3D, to name a few, and MIT's V3 visualisation software.

6. COMMUNICATION RATE

The 10/100 Mbps Intel 51OT and 410T switches link the different nodes of ANUP16 and HPC22 to the master. The Fast Ethernet switch has its performance enhanced through increased raw bandwidth and reduced traffic congestion. An important feature of scalable stacking technology (SST) is the chassis-based switch that enables it to be managed as a single device. The scalable stacking technology is controlled by simple network management protocol (SNMP)-supported Linux operating system. Each of the 16 ports has a 10/100 Mbps link connected to it. The rate of internode data transfer determines the communication rate of the parallel computer. All parallel computers are designed to minimise the time taken for this process.

The complexity of parallel computation log *P*, is characterised through four parameters-bandwidth *(W),* latency (L), overhead (O) and number of processors(P). Bandwidth rated in Mbps is the amount of data in megabits communicated by link per unit time (s), latency is idle or start-up time of processor before the message is sent. Since network capacity is finite and almost (L/W), messages can be transferred from one machine to another at any given time, the assumption made in Hockney's intrinsic parameter calculation, that latency is constant or bandwidth is infinite, is highly idealistic[23].

The MPI library communication calls, 8 in number, tested on HPC22 using Mpbench program of Llcbench suite for increasing message size have been plotted in Fig. 13. A comparison with CRAY T3E (broken line) shows an order of magnitude higher bandwidth for CRAY, the effect of cache is also evident as all curves tend to level off with the increase in traffic. Roundtrip, measured in transactions per second, shows better utilisation of link for small data size, which is actually twice that of other cases, but its performance declines when the link becomes busy. The actual internode communication rates of ANUP16 and HPC22 are determined by Banlat code using ANULIB library (Alib) in ping-

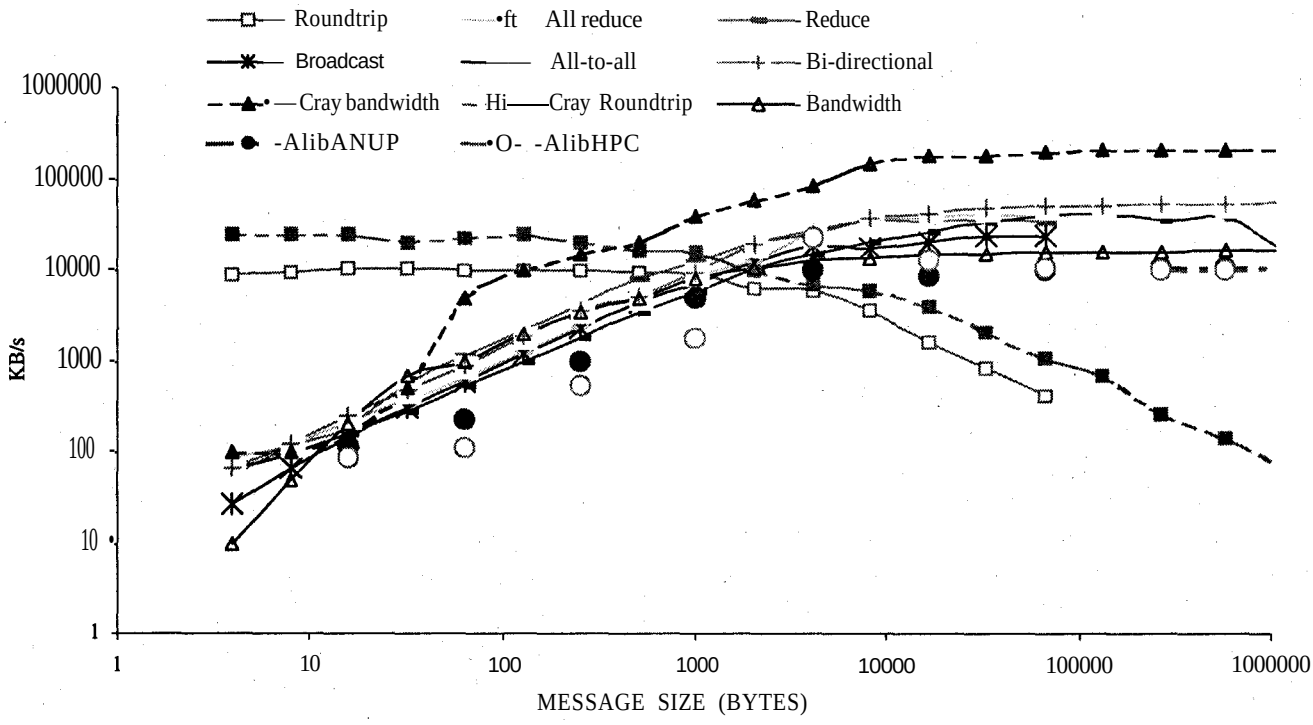

Figure 13. Intel switch performance for MPI/Anulib (Alib) communication calls

pong test.

In a one-to-all communication, message is broadcast by the master to all the eight slaves in the subgroup and the message returned by a single processor is received by the master. The measured bandwidth by ANULIB plotted in Fig. 13 is 10-15 per cent lower than that of MPI LAM communication calls, which means that the latter is slightly faster than the former. Latency is determined by C-benchmark code Ctest and the overhead is found through repeated measurement in the above test with Banlat code. The measured maximum and average latency are 26.00 ^is and 0.95 us (950 ms) respectively and the communication overhead varied from 0.22 ns to 0.44 ns.

In Fig. 14, the inverse relationship of latency in contrast to bandwidth, is evident with exponential rise in latency when the message size exceeds that of cache(10^4). Iteration, however, has negligible effect on latency. Finally, the inverse of barrier communication rate of Intel 510T, 410T and 3Com switches are presented in Table 7 as a comparative measure of switch performance during synchronisation.

6. APPLICATION BENCHMARK-A COARSE-GRAINED CFD PROBLEM

A coarse-grained process performs large number (millions) of arithmetic operations before communication takes place. A measure of granularity of a process is the computation-to-communication ratio. To determine the performance of MIMD machines for a CFD problem, two cases of flow simulation for combat aircraft configuration at Mach number 0.9 and angle of attack of 5° are performed. In the first case, the tailless aircraft covered by 51 x 25 x 67 (roll-x, pitch-y and yaw-z axes) mesh points (GCC grid of 9 blocks and memory 2MB) and in the second, a more dense mesh of 131 x 51 x 114 points (GCM grid of 142 blocks and 9MB disk space) generated over the aircraft carrying a close-combat missile are solved by Jameson's finite volume, 3-D Cartesian, time-dependent Euler multi-block solver. The elapsed time per iteration for these computation are recorded in Table 6.

It has been observed that ANUP16 is faster

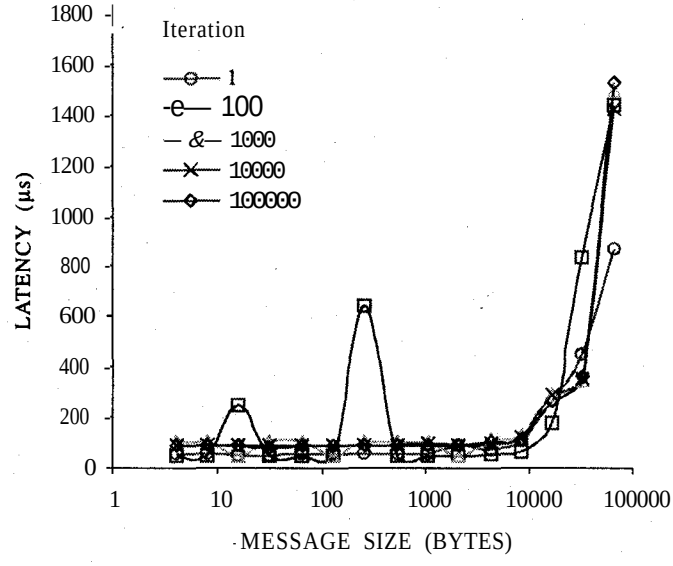

Figure 14. Variation of latency of HPC22 with message size

than HPC22 for computationally-intensive GCM grid and the reverse is true for the fewer-points GCC grid, the reason being that GCM grid comprises 142 logical blocks whereas GCC grid is made up of barely 9 logical blocks. The cells inside any given block are structured but the blocks themselves may be unstructured. Thus, division into greater number of logical blocks or fine grain dominated by communication appears to favour ANUP16. The MIMD machines are at least four times faster than SEVID PACE32 machines despite easing congestion in communication link of the latter by odd-even process exchange (modified code).

8. CONCLUSION

A complete bench test procedure for parallel computers is established. As a thumb rule, a balanced parallel architecture should yield equal number of MB, Mbps, and Mflops. To realise it in practice for a range of applications, performance data from

Table 7. Global synchronisation time per barrier of links

Computer	ANUP16	HPC22	HPC64
Processors	16	22	64
F.E. switch	S510T	J410T	3Com
Time barrier	1633 us	1000 us	3450 us

bench test would be required. It is found that for efficient memory management through locality of reference, block size of 50 to 100 and depth of unroll up to 4 are required. The vectorisation parameters /?, «, and intranode communication time determined from cache operation or memory bottleneck indicate that fine-grained problems yield optimum performance of ANUP16.

Coarse-grained parallelism is desirable for efficient parallelisation of distributed memory machines. Generally in practice, this is achieved by increasing the number of processors as in HPC64. Scalability and application benchmark tests revealed that HPC64 optimum performance may be obtained for coarse-grained problems. A single node cache performance of HPC64 is found to be almost the same as that of CRAY-T3E. Roughly, the computing power of ANUP16 is at least ten times faster than the serial RISC1 machine and at least three times faster than that of DecAlpha workstation.

REFERENCES

1. del Rosario, J.M. & Choudhary, A.N. High performance input/output for massively parallel computers: Problems and prospects. *IEEE Computer*, March 1994, 59-68.

2. Dongarra, J.J.; Meuer, W.H. & Strohmaier E. Top 500 supercomputing sites, Ed. 13. University of Tennessee, UT-CS-99-434, November 11, 1999.

3. Addison, C.A.; Getov,VS.;Hey, A.J.G.; Hockney, R.W. & Wolton, I.C. The GENESIS of distributed memory benchmarks. *In* Computer benchmarks, edited by J. Dongarra and W. Gentzsch. North-Holland, 1993. pp. 257-71.

4. Krishna, H.S. & Singh, K.P. Performance analysis and efficient parallelisation of CFD codes; *In* IIT Kharagpur Golden Jubilee Celebration, Procedure of Workshop on CFD, December 5-7, 2001. IIT Kharagpur.

5. Deshpande, S.M. High performance computing (HPC) in CFD: An Indian perspective. Invited Lecture. *In* Proceedings of Sixth International Conference on HPC in Asia-Pacific Region, HPC-Asia-2002. Bangalore, 16-19 December 2002. Vol. 1. pp.129.

6. Ueberhuber, C.W. Numerical computation: Methods, software and analysis. Vol. 1. Springer, 1997. 267p.

7. Krishna, H.S. & Singh, K.P. Scalable performance of beowulf clusters. *In* Proceedings of Sixth International Conference on High Performance Computing in Asia Pacific region, HPC-Asia-2002, Bangalore, 16-19 December, 2002. Vol. 1. pp. 153-56.

8. Fletcher, C.A.J. Computational techniques for fluid dynamics. Vol. 1. Springer-Verlag.

9. Demme, J.W.; Heath M.T. & van der Vorst, H.A. LAPACK working note 60, parallel numerical linear algebra. University of Tennessee. UT-CS-93-192, August 1993.

10. Dongarra, J. Innovative Computing Laboratory, University of Tennessee, Knoxville. Report No. ICL-2001.

11. Cai, X.; Keyes, D.E. & Venkatkrishnan, V. Newton-Krylov-Schwarz: An implicit solver for CFD. NASA Report ICASE-1995-87.

12. Jameson, A. & Caughey, D.A. How many steps are required to solve the Euler equations for steady compressible flow: *In* Search of a fast solution algorithm. AIAA Paper No-2001-2673.

13. Sharov,D.; Luo, H. & Baum, J.D. Implementation of unstructured grid GMRES+ LU-SGS method on shared memory: Cache-based parallel computers. AIAA Paper No-2000-0927.

14. Tian ruo Yang (Ed). Parallel numerical computations with applications. Kluwer Academic Publications, 1999. ISBN 0-7923-8588-8.

15. Saad, Y.; Lo, G.C. & Kuznetsov, S.; PSPARSLIB users manual: A portable library of parallel iterative solvers. University of Minnesota, Dept of Computer Science, Minneapolis, MN, 1998.

16. Eijkhout, V. Overview of iterative linear system solver packages. University of Tennessee, UT-CS-98-411, July 1998.

17. Getov, V.S. Performance characterisation of the cache memory effect. *Supercomputer*, 1995, 62, XI(4), 31-49.

18. Bailey, D.H. FFTs in external or hierarchical memory. 1990, *Journal of Supercomputing*, *4*, 23-35.

19. Gatlin, K.S. & Carter, L. Memory hierarchy considerations for fast transpose and bit reversals. *In* Proceedings of HPCA, Orlando, PL, 9 to 13 January 1999.

20. Hultquist, J.P.M. Improving the performance of particle tracing in curvilinear grids. AIAA Aerospace Sciences Meeting, Reno, NV, January 1994. AIAA Paper No-94-0324.

21. Press, W.H. *et al.* Numerical recipes in FORTRAN 77: *In* The art of scientific computing. Cambridge University Press, 1986. pp. 525 & 881.

22. Poirier, D.M.A. *et al.* Advances in the CGNS database standard for aerodynamics and CFD. AIAA Aerospace Sciences Meeting, Reno, NV, January 12, 2000. AIAA 2000-0681.

23. Petrini, F. Latency and bandwidth requirements of MPP: FFT as a case study. Elsevier, June 1999.

Contributors

Mr H.S. Krishna obtained his BE(Mech Engg) from the Bangalore University and ME(Aerospace Engg) from the Indian Institute of Science (IISc), Bangalore. Presently, he is working as Scientist/Engineer D at the Aerodynamic Group of Aeronautical Development Agency(ADA), Bangalore. His areas of expertise include: Heat transfer, computational fluid dynamics, and high performance computing.

Dr K.P. Singh obtained his BSc(Mech Engg) from the Banaras Hindu University; MTech (Mech Engg) from the Indian Institute of Technology, Kanpur and PhD(Aerospace Engg) from the IISc, Bangalore. Presently, he is Group Director, Aerodynamic Group at the ADA, Bangalore. His areas of research include: Development of CFD Euler codes, aerodynamic design of launch vehicle, CFD tools and their application for design and development of aircraft, and parallel computing.