1-Dimensional Parallel FFT Benchmark on SUPRENUM

Vladimir Getov

Dept. of Electronics & Computer Science
University of Southampton
Southampton SO9 5NH, U.K.
e-mail vsg@ecs.soton.ac.uk

Abstract. A distributed memory vectorised 1-dimensional FFT benchmark is first presented. The performance results of this benchmark on SUPRENUM are given and discussed. A performance analysis of the distributed memory FFT benchmark is performed and Hockney's performance parameters (r_∞ and $n_{1/2}$) are then employed to derive the performance formula. This formula is shown to fit the experimental results very well. A generalization of the analysis for uniformly distributed applications is also discussed, as well as some important characteristics such as the calculation/communication ratio, the fit of the application to the architecture, the average message length and the average vector length.

Keywords: distributed memory FFT benchmark; performance formulae and parameters; SUPRENUM performance.

1 Introduction

FFT algorithms are largely used to transform physical processes from the time domain to the frequency domain (Fourier analysis) or from the frequency domain to the time domain (Fourier synthesis). The numerous applications on the one hand, and the large variety of numerical methods and algorithms [Swa87] on the other, have made them some of the most popular benchmarks for performance measurements and analysis.

The SUPRENUM computer has a MIMD/SIMD architecture [Gil88], providing a simultaneous process execution across the nodes with message-passing communications and a data parallelism within the nodes on the basis of a vector floating-point unit [Kam88]. The simple equivalent transformations from 1-dimensional to multi-dimensional FFT's and vice-versa have incontestable advantages for performance improvement in the SUPRENUM environment. Those advantages are expressed mainly in a significant decrease of communication intensity and also in an increase of vectorisation efficiency. However, it is particularly interesting to assess the SUPRENUM performance for a high level of parallelism and a relatively low level of vectorisation. This is the reason for concentrating our attention especially to 1-dimensional FFT and for including this benchmark in the GENESIS distributed benchmark suite [Hey91].

2 Benchmark Features

2.1 Calculation Efficiency

The Fourier transform comprises two main phases:

- the generation of twiddle factors;
- the calculation of harmonic amplitudes (butterfly phase).

The latter is commonly critical for the speed of computation. The first phase needs a smaller number of arithmetic operations and, furthermore, it may be precomputed and used many times for different problem sizes in some applications. The FFT algorithm offers an exceptional reduction of the number of operations for the calculation of harmonic amplitudes. The price paid for that reduction is the variable stride indexation for the classical FFT derivation, or the disordered bit-reversed form of the resulting output sequence for the 'in-place' FFT algorithms. The first option causes problems for both vectorisation and distribution. On the one hand, the distributed version of this option combines data from two processors and puts the result in a third processor which is clearly inefficient. Variable stride indexation, on the other hand, is inconvenient for the SUPRENUM computer because of significant performance losses in the vector floating point unit [Kam88]. Thus only the 'in-place' transforms are suitable for implementation on SUPRENUM, but unfortunately they have their output sequence in scrambled order. Most frequently the solution of this problem includes a bit-reversal of the input or the output sequence, which is actually a third component in the transformation evaluation. The bit-reversal comprises two steps:

- Preparation of a bit-reversed index vector. This calculation depends only on the problem size and the data allocation across the processors in distributed versions and may be precomputed for each different problem size, as was the case of the twiddle factors.
- Shuffle of input(output) sequence in accordance with the bit-reversed index vector, which can be done during the calculation of harmonic amplitudes.

Hence, the best calculation efficiency could be achieved if the bit-reversed index vectors and the twiddle factors are precomputed for every different problem size. In that case the time-critical part of 1-dimensional FFT consists of the butterfly phase and the shuffle of input (output) sequence. The latter may be neglected as it does not imply any arithmetic operations.

2.2 Communication Structure

The knowledge of the processor number is necessary in the distributed versions of the benchmark. The natural communication structure for the butterfly phase is a d-dimensional hypercube [Cha88]. Assuming there are $p = 2^d$ processors we define d directions labeled from 0 to $d-1$. Starting from the first processor, which is assigned number 0, the other nodes are numbered in the following fashion. Assume the current processor number is p_i and the processor we want to assign a number is in positive direction d_j from this one, then we assign it the number $p_i + 2^{d_j}$. This uniquely and consistently numbers all the processors. The benchmark uses a local communication control mechanism, performed by task-number dependent functions [Pea68, Pea77]. In our particular case the following functions are implemented:

Destination address = f(Task number, Transform level);
Message allocation = f(Task number, Transform level);
Twiddle factors offset = f(Task number, Transform level).

All distributed versions first implement the communication (inter-node) part of the butterfly phase in $\log p$ stages. This allows short vectors for twiddle factors to be employed in this part of the transform.

2.3 Size Parameters

The problem size is determined by the total number of data-points to be transformed (N). In the multi-node versions the 1-dimensional input sequence is distributed uniformly across the processors. Hence, for a given problem size there are several different time measurements for a different number of processors. Therefore the main benchmark's parameters that specify the problem are:

N - the total number of data-points to be transformed;
p - the number of processors.

It is possible to encounter a large variety of different input data assignments across the nodes in a distributed memory system [Law75]. The number of possible assignments depends on the number of nodes and is usually equal to $p!$. The communication intensity and therefore the overall performance depend on the choice of data assignment. Hence, it is worth trying to find an optimal solution to this problem. Probably the best initial data assignment for the 1-dimensional FFT is a bit-reversed order of input vectors (each of length N/p) across the nodes which allows communication to be avoided during the bit-reversal phase of the benchmark.

2.4 Versions

There are scalar (Fortran77) and vectorised (Fortran90) versions of this benchmark running on a single SUPRENUM node or distributed among the processors of the SUPRENUM machine. Both scalar and vectorised versions present variations of the classical derivation of the algorithm by Cooley-Tukey [CT65]. They all perform 'in-place' transforms which economize on memory, so allowing measurements in a wider range of problem sizes to be performed.
The benchmark uses the radix-2 FFT algorithm for length $N = 2^m$, which consists of m radix-2 stages. The first $\log p$ stages are retained for the inter-node (communication) part of the benchmark. At each stage of the second (intra-node) part of the transform, $N/2p$ radix-2 butterflies are computed by performing two nested `DO`-loops. Depending on the stage of the FFT computation, the length of the two `DO`-loops varies from $N/2p$ to 1 for the number of data points (i-dimension) and from 1 to $N/2p$ for the number of partial transforms (k-dimension). It is possible to swap the order of these `DO`-loops half-way through the transform to increase the

average vector length. The butterfly phase of all versions is implemented by 10 real floating point operations in the innermost loop. Following the notation in [HJ88] the development of the vectorised benchmarks is characterized by:

- Scheme A - vectorisation in i-dimension (the innermost `DO`-loop is over the number of data points, maintaining a stride of one);
- Scheme B - vectorisation in k-dimension (the innermost `DO`-loop is over the partial transforms, using a variable stride);
- Scheme AB - in that case the transform swaps scheme A for scheme B half-way, having a stride of one in the first half and a variable stride in the second half of the butterfly phase;
- Scheme TB - vectorisation in k-dimension (a stride of one is achieved by reshaping the input data vector);
- Scheme ATB - maintains a stride of one throughout the transform by means of a transposition in the middle.

3 SUPRENUM Performance

The results generated by the 1-dimensional FFT benchmark include the elapsed time and the performance of the butterfly phase. Timing measurements of the remaining transformation phases (bit-reversed index generation, twiddle factors calculation, bit-reversed shuffle) are also output. Only the benchmark performance (R_B) of the butterfly phase is presented here, as this is the significant component for many applications.

$$R_B(N,p) = \frac{F_B(N)}{T(N,p)} \tag{1}$$

Here F_B is the number of real arithmetic operations ($5N \log N$ for the 1-dimensional FFT) and T is the elapsed time. Performance figures in *Mflop/s* for various single node versions of the benchmark are given in Table 1. Scheme A is seen to be much more efficient than scheme B. The main reason for this appears to be the variable stride indexation in scheme B whilst scheme A uses a stride of one. If the problem size $N \geq 16K$, then scheme B and, as a result of that scheme AB as well, enter a performance degradation area (see Fig. 1). This performance degradation arises most likely because the maximal stride becomes larger than the volume of the cache vector memory ($7Kword$) [Kam88]. In the case of the 1-dimensional FFT benchmark $stride_{max} = N/2$. Hence, if $N \geq 16K$ the maximal stride is indeed beyond the address space of the cache vector memory and the performance degradation takes effect.

In order to overcome this problem a new scheme TB subroutine is introduced performing the transform by means of stride of one. The performance of scheme TB is then only marginally inferior to scheme A. Hence, it is the faster scheme A subroutine which is called an extra time if the number of layers in the intra-node part is odd. Furthermore, scheme A performs the computation in the inter-node part of the transform for distributed versions (see Fig. 2).

The increase in the performance of scheme AB is significant in comparison with schemes A and B. This increase can be attributed to the increase in average vector length for this version. Scheme AB also outperforms scheme ATB for smaller

Table 1. SUPRENUM performance in *Mflop/s* for the single node 1-dimensional FFT benchmark

Problem size (N)	Scalar version	Vectorised versions				
		Scheme A	Scheme B	Scheme AB	Scheme TB	Scheme ATB
128	0.120	0.215	0.126	0.605	0.179	0.546
256	0.125	0.264	0.148	0.948	0.213	0.931
512	0.127	0.304	0.168	1.422	0.243	1.324
$1K$	0.129	0.341	0.186	2.082	0.272	1.882
$2K$	0.129	0.376	0.203	2.428	0.299	2.327
$4K$	0.130	0.409	0.219	2.701	0.325	2.805
$8K$	0.130	0.444	0.235	3.409	0.352	3.453
$16K$	0.131	0.478	0.227	2.879	0.378	4.125
$32K$	0.131	0.512	0.215	2.632	0.405	4.378
$64K$	0.131	0.546	0.215	2.019	0.431	4.839
$128K$	0.132	0.580	0.215	2.333	0.457	5.014

transforms. However, as the problem size increases the efficiency of scheme AB declines, probably again due to the variable stride indexation in the B part of the algorithm. Hence scheme ATB becomes the most efficient for large transforms and the highest performance on a single node has been achieved with it - 5.014 *Mflop/s* for $N = 128K$ data-points.

The highest performance achieved on the SUPRENUM in Liverpool is 37.981 *Mflop/s* (scheme ATB, $1024K$ data-points distributed across 16 processors). The results in Table 2 show that the performance of the benchmark still increases fast enough as the number of processors increases. This gives us further interest to run the 1-dimensional FFT benchmark on larger configurations.

Table 2. SUPRENUM performance in *Mflop/s* for the distributed memory 1-dimensional FFT benchmark

Problem size (N)	Scalar version				Vectorised version (Scheme ATB)			
	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 2$	$p = 4$	$p = 8$	$p = 16$
$1K$	0.216	0.412	0.598	1.020	0.908	1.286	1.249	1.925
$2K$	0.236	0.421	0.729	1.394	1.586	1.722	2.572	3.587
$4K$	0.248	0.465	0.835	1.341	2.549	3.012	3.803	4.744
$8K$	0.253	0.492	0.932	1.667	3.244	4.638	5.788	7.032
$16K$	0.257	0.504	0.971	1.837	3.955	6.472	8.623	11.288
$32K$	0.258	0.511	1.001	1.934	5.163	6.781	11.692	16.297
$64K$	0.260	0.515	1.014	1.989	5.182	9.093	13.314	22.329
$128K$	0.260	0.517	1.023	2.017	6.534	10.347	17.718	24.199
$256K$	[a]	0.519	1.028	2.033	[a]	11.594	18.965	32.641
$512K$	[a]	[a]	1.032	2.045	[a]	[a]	20.871	34.284
$1024K$	[a]	[a]	[a]	2.053	[a]	[a]	[a]	37.981

[a] No results, because the problem size is too large to fit into the local memories.

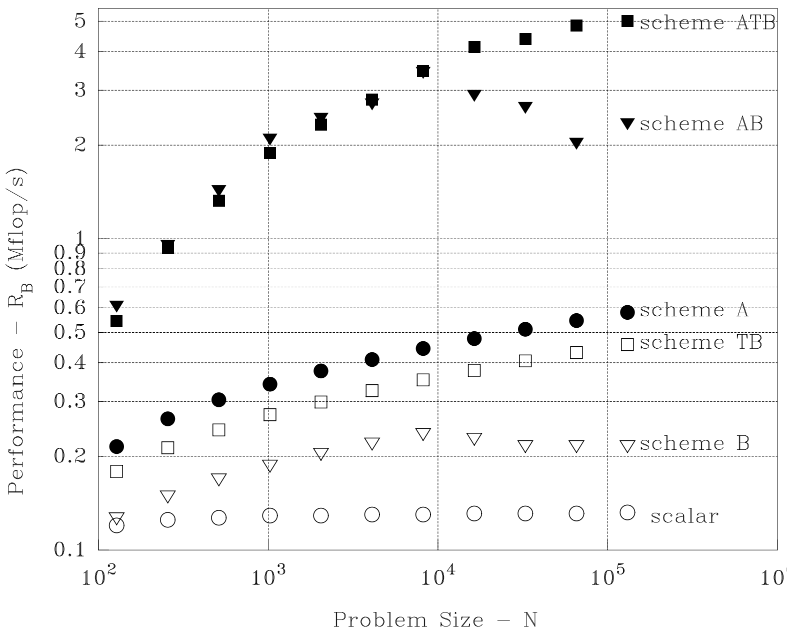

Fig. 1. Benchmark performance of the single node 1-dimensional FFT, $R_B(N)$, as a function of the problem size, N, for different vectorised versions.

4 Performance Analysis

The distributed versions of 1-dimensional FFT benchmark are implemented by a set of identical tasks which calculate the transform itself, and a master-task which performs the task-creation mechanism, some control functions, and the collection of results. We assume for simplicity that the master-task provides an ideal global synchronization, i.e. all of the 'workers' start and stop simultaneously. There is only one type of bidirectional communication between pairs of nodes throughout the transform which leads to the assumption of a full-duplex communication sub-system performing these communications simultaneously. Hence, neither overlaps nor gaps can occur between calculations and communications if all of the nodes have identical hardware performance and behaviour.

From the assumptions above, the time for the whole 1-dimensional FFT benchmark is equal to the time for a single task of it. Hence, only the elapsed time per node need be considered for the purpose of deriving the performance formula. The derivation is based on an operation count for the butterfly phase of $5N \log N$ real arithmetic operations. The following analysis deals only with this phase paying no attention to the bit-reversal and the twiddle factors generation as these can be neglected in many applications.

The butterfly phase comprises two main parts: an inter-node part and an intra-node part(Fig. 2). The inter-node part contains all the communications of the benchmark.

The communication range of every data-point halves from layer to layer. Thus, there is no more communication at the end of the first $\log p$ layers. The intra-node part follows the classical vectorised implementation of the 1-dimensional FFT algorithm [HJ88]. This part is divided into two halves (scheme A and scheme B) having each of them $\frac{\log N/p}{2}$ layers in order to obtain the maximum possible vector length. It should be pointed out that for maintaining a stride of one in the second half of vectorised version, a transposition is performed in the middle, bringing the implementation to scheme ATB. This solution raises substantially (by about 60%) the SUPRENUM performance of the intra-node part of the benchmark.

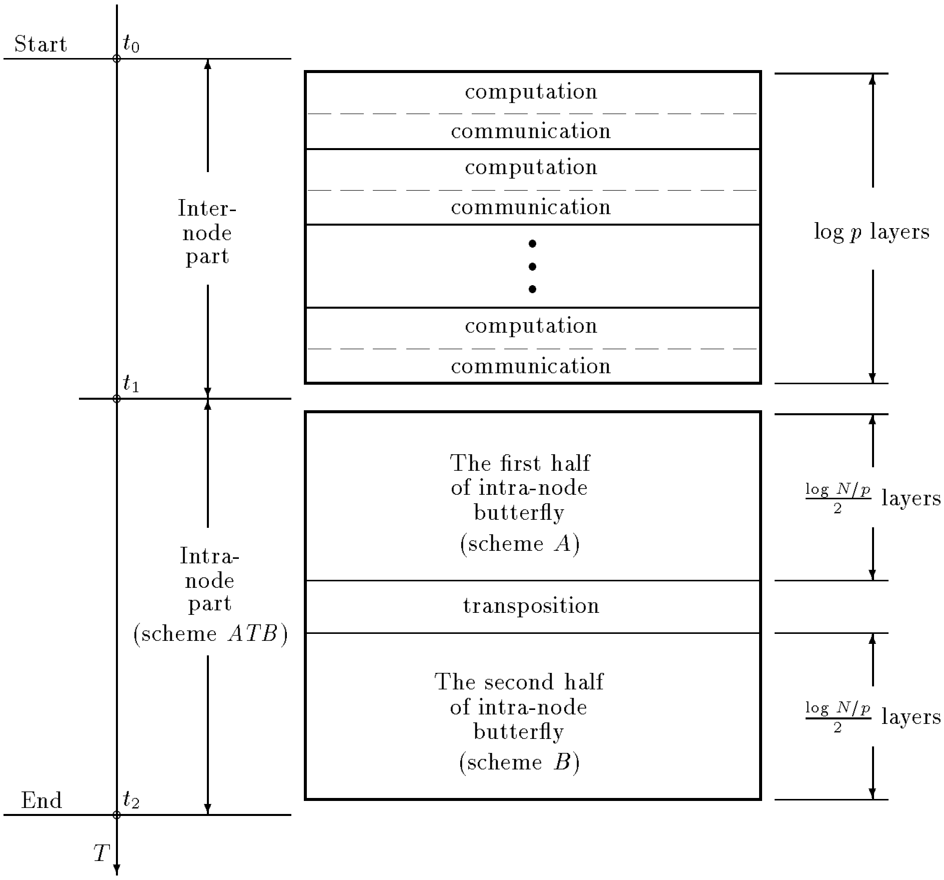

Fig. 2. The FFT butterfly layering per node

The benchmark time (T) is a function of N and p presented as a sum of the calculation time and the communication time [Hoc92], because no overlaps occur:

$$T(N,p) = T_{calc}(N,p) + T_{com}(N,p) \quad (2)$$

The communication time can be represented by two components - transmission time and 'start-up' time:

$$T_{com}(N,p) = \frac{s^c}{r^c_\infty} + \frac{q^c n^c_{1/2}}{r^c_\infty} \qquad (3)$$

The transmission time depends on the total amount of data to be sent per node (s^c) divided by the asymptotic communication performance (bandwidth) (r^c_∞). The 'start-up' time is given by the number of requests to send per node (q^c) multiplied by the communication half-performance parameter ($n^c_{1/2}$) and divided again by r^c_∞. In order to ensure compatibility with the calculation time formula we introduce similar measurement units. The measurement unit for s^c and $n^c_{1/2}$ is *word*, while r^c_∞ is being measured in *Mword/s*. Here the length of one word is equal to the length of the corresponding data type used in the calculation time formula. In our implementation one word is equal to 8 bytes (`DOUBLE PRECISION` data type).
In formula (3) s^c and q^c are functions of N and p. In the case of 1-dimensional FFT one can obtain:

$$s^c(N,p) = \frac{N \log p}{p}; \qquad q^c(N,p) = \log p. \qquad (4)$$

Substituting (4) into (3) one derives the communication timing formula of the 1-dimensional FFT benchmark:

$$T_{com}(N,p) = \frac{N \log p}{r^c_\infty p} + \frac{n^c_{1/2} \log p}{r^c_\infty} \qquad (5)$$

The calculation time for scalar code (T^s_{calc}) is given by:

$$T^s_{calc}(N,p) = \frac{5N \log N}{r^s_\infty p}, \qquad (6)$$

where r^s_∞ is the scalar theoretical performance per node. Thus, adding (6) and (5) the scalar timing formula of 1-dimensional FFT is:

$$T^s(N,p) = \frac{5N \log N}{r^s_\infty p} + \frac{N \log p}{r^c_\infty p} + \frac{n^c_{1/2} \log p}{r^c_\infty} \qquad (7)$$

The calculation time for vector code (T^v_{calc}) can be derived on the basis of the following formula:

$$T^v_{calc}(N,p) = \frac{s^v}{r^v_\infty} + \frac{q^v n^v_{1/2}}{r^v_\infty} \qquad (8)$$

where s^v is the number of floating point operations, q^v is the number of vector operations, and r^v_∞ is the asymptotic vector performance per node. On the basis of the timing analysis above one can obtain the following values for s^v and q^v:

$$s^v(N,p) = \frac{5N \log N}{p}; \qquad q^v(N,p) = 10\left(\log p + 2\left(\sqrt{\frac{N}{p}} - 1\right)\right). \qquad (9)$$

Thus, performing the corresponding substitutions in (8) and adding the communication time (5), the vector timing formula of the 1-dimensional FFT is:

$$T^v(N,p) = \frac{5N \log N}{r^v_\infty p} + \frac{10 n^v_{1/2}\left(\log p + 2\left(\sqrt{\frac{N}{p}} - 1\right)\right)}{r^v_\infty} + \frac{N \log p}{r^c_\infty p} + \frac{n^c_{1/2} \log p}{r^c_\infty} \qquad (10)$$

The benchmark performance can be derived on the basis of (1, 7, 10) and after some transformations one can obtain the corresponding performance formulae for scalar (11) and vectorised (12) versions of the 1-dimensional FFT:

$$R_B^s(N,p) = \frac{r_\infty^s p}{1 + \frac{r_\infty^s \log p}{5 r_\infty^c \log N} + \frac{r_\infty^s n_{1/2}^c p \log p}{5 r_\infty^c N \log N}}; \tag{11}$$

$$R_B^v(N,p) =$$

$$= \frac{r_\infty^v p}{1 + \frac{2 n_{1/2}^v p \log p}{N \log N} + \frac{4 n_{1/2}^v p \left(\sqrt{\frac{N}{p}} - 1\right)}{N \log N} + \frac{r_\infty^v \log p}{5 r_\infty^c \log N} + \frac{r_\infty^v n_{1/2}^c p \log p}{5 r_\infty^c N \log N}} \tag{12}$$

These formulae are functions of N and p. All other parameters can be substituted by their measured or estimated values for the SUPRENUM computer ($r_\infty^s = 0.188$ *Mflop/s*; $r_\infty^v = 3.5$ *Mflop/s*; $r_\infty^c = 0.6$ *Mword/s*; $n_{1/2}^v = 225$; $n_{1/2}^c = 4000$). Some of the measured values from the distributed benchmark versions together with the predicted values by the above formula are plotted on Fig. 3. The results show a very good fit of the performance formula (12) to the measurement results.

5 Performance Formula of Uniformly Distributed Applications

The expressions (11) and (12) have some fascinating properties, which give us the opportunity to continue to a further extent the performance analysis. In these formulae one can easily discover that

$$\frac{5 \log N}{\log p} = \frac{5 N \log N p}{p \log pN} = \frac{s^s}{s^c} \tag{13}$$

This is the application's calculation/communication ratio or calculation intensity, while r_∞^s / r_∞^c is the corresponding calculation/communication ratio of the architecture. It should be mentioned also that $N/p = n^c$ is the average message length of the application. Thus, the scalar performance formula of uniformly distributed applications is given by:

$$R_B^s(N,p) = \frac{r_\infty^s p}{1 + \frac{r_\infty^s}{r_\infty^c} \frac{s^c(N,p)}{s^s(N,p)} \left(1 + \frac{n_{1/2}^c}{n^c(N,p)}\right)} \tag{14}$$

In this formula we can define a function α^s:

$$\alpha^s(N,p) = \frac{r_\infty^s}{r_\infty^c} \frac{s^c(N,p)}{s^s(N,p)} \tag{15}$$

This function may be used for measuring the fit of the application to the architecture. The best fit (but not the best performance) is when $\alpha = 1$. If $\alpha \ll 1$, then $R_B^s \approx R_\infty^s$,

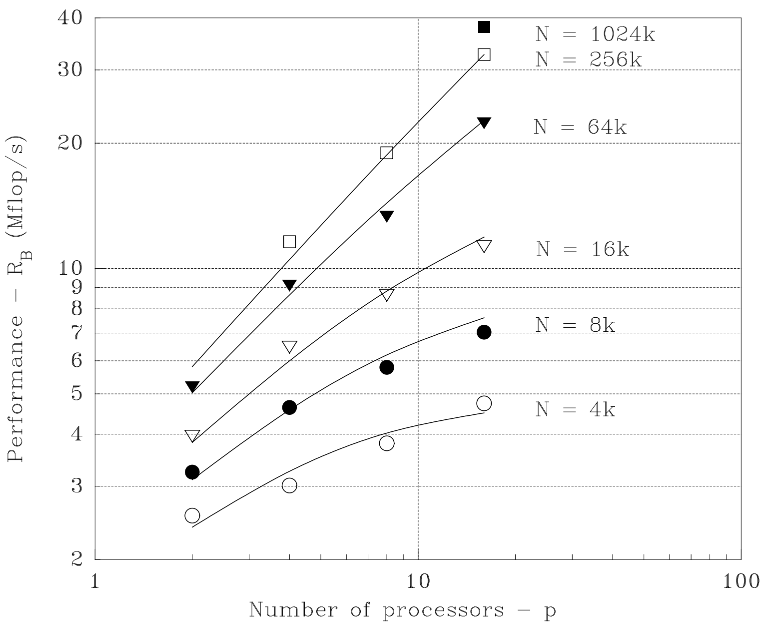

Fig. 3. Benchmark performance of the distributed 1-dimensional FFT, $R_B(N;p)$, as a function of the number of processors, p, for different values of transform length, N. The plot symbols are measured values, and the lines are the predicted values from the performance analysis (see text).

but the communication subsystem is underloaded as the application is uniformly distributed into almost independent tasks with a very low degree of communications between them. If $\alpha \gg 1$, then the performance dramatically decreases because of the very intensive communications leaving only a very short time for the main calculation work.

The vector performance formula of uniformly distributed applications follows a similar derivation. It has, however, one more fascinating function presenting the average vector length of the application, which in the case of 1-dimensional FFT is given by:

$$n^v(N,p) = \frac{N \log N}{2p\left(\log p + 2\left(\sqrt{\frac{N}{p}} - 1\right)\right)} \tag{16}$$

Thus, the vector performance formula of uniformly distributed applications is:

$$R_B^v(N,p) = \frac{r_\infty^v p}{1 + \frac{n_{1/2}^v}{n^v(N,p)} + \alpha^v(N,p)\left(1 + \frac{n_{1/2}^c}{n^c(N,p)}\right)} \tag{17}$$

Here again the function α^v shows how much the application fits the architecture:

$$\alpha^v(N,p) = \frac{r_\infty^v}{r_\infty^c}\frac{s^c(N,p)}{s^v(N,p)} \tag{18}$$

In both scalar and vector formulae the numerator contains the total peak performance (R_∞) as a function of the number of processors employed. For uniformly distributed applications with mixed scalar and vector code the number of floating point operations per node (s^n) is:

$$s^n(N,p) = s^s(N,p) + s^v(N,p) \tag{19}$$

In that case we can define the node peak performance (r^n_∞) as

$$r^n_\infty = \frac{s^n}{\dfrac{s^s}{r^s_\infty} + \dfrac{s^v}{r^v_\infty}}. \tag{20}$$

and finally, the general performance formula of uniformly distributed applications is:

$$R_B = \frac{r^n_\infty p}{1 + \dfrac{n^v_{1/2}}{n^v} + \dfrac{r^n_\infty}{r^c_\infty}\dfrac{s^c}{s^n}\left(1 + \dfrac{n^c_{1/2}}{n^c}\right)} \tag{21}$$

Here (21), the numerator contains the node peak performance (r^n_∞) as a function of the scalar/vector ratio of the application (20). This is in fact Amdahl's limit [Amd67] for the node asymptotic performance.

6 Concluding Remarks

The above formulae are derived on the basis of some assumptions and restrictions. They do not consider the importance of overlapping, being valid only for non-overlapped codes. Neither timing nor performance formulae include any additional calculation time caused by the transposition (see Fig. 2). In our case this time is negligible and may be omitted, because the transposition does not imply any arithmetic operations. In general, however, the effect of extra code must be taken into account in $T(N,p)$, while $F_B(N)$ should be kept unchanged for different versions of a given benchmark.

It is impossible to extrapolate the performance prediction for the SUPRENUM computer beyond 16 processors, because of the influence of the inter-cluster communication and the communication's hierarchy.

The performance formulae above do not take into account the effect of the memory bandwidth and the variable communication throughput. From that point of view they give only a rough idea about the performance of uniformly distributed applications. These restrictions are subject for future work.

7 Acknowledgments

The author wishes to thank his colleagues from the Concurrent Computations Group at the University of Southampton and in particular Prof. Roger Hockney for his continuous encouragements and fruitful discussions. Special thanks go to the staff of the computer centre at the University of Liverpool for their assistance in operating the SUPRENUM installation. The funding received from the ESPRIT project GENESIS is also gratefully acknowledged.

References

[Amd67] G.M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. 1967 spring joint computer conference. *AFIPS Conference Proceedings*, 30:483–485, 1967.

[Cha88] R.M. Chamberlain. Gray codes, Fast Fourier Transforms and hypercubes. *Parallel Computing*, 6:225–233, February 1988.

[CT65] J.W. Cooley and J.W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, **19**(90):297–301, 1965.

[Gil88] W.K. Giloi. SUPRENUM: A trendsetter in modern supercomputer development. *Parallel Computing*, **7**(3):283–296, July 1988.

[Hey91] A.J.G. Hey. The GENESIS distributed memory benchmarks. *Parallel Computing*, **17**(10 - 11):1275–1283, 1991.

[HJ88] R.W. Hockney and C.R. Jesshope. *Parallel Computers 2: Architecture, Programming and Algorithms.* IOP Publishing, Adam Hilger, Bristol and Philadelphia, 1988.

[Hoc92] R.W. Hockney. A framework for benchmark performance analysis. *Supercomputer 48*, **IX**(2):9–22, March 1992.

[Kam88] H. Kammer. The SUPRENUM vector floating-point unit. *Parallel Computing*, **7**(3):315–323, July 1988.

[Law75] D.H. Lawrie. Access and alignment of data in an array processor. *IEEE Trans. on Computers*, 24(12):1145–1155, December 1975.

[Pea68] M.C. Pease. An adaptation of the Fast Fourier Transform for parallel processing. *Journal of the ACM*, 15(2):252–264, April 1968.

[Pea77] M.C. Pease. The indirect binary n-cube microprocessor array. *IEEE Trans. on Computers*, 26(5):458–473, May 1977.

[Swa87] P.N. Swarztrauber. Multiprocessor FFTs. *Parallel Computing*, 5:197–210, July 1987.