Performance Analysis of Distributed Applications by Suitability Functions

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

To be published in: *Proceedings of the MPPM Conference, Berlin,*
IEEE Computer Society Press, 1993.

Abstract

A simple programming model of distributed-memory message-passing computer systems is first applied to describe the couple architecture/application by two sets of parameters. The node timing formula is then derived on the basis of scalar, vector and communication components. A set of suitability functions, extracted from the performance formulae, are defined. These functions are applied as an example to the performance analysis of the 1-dimensional FFT benchmark from the GENESIS benchmark suite. The suitability functions could also be useful for comparative performance analysis of both existing distributed-memory systems and new architectures under development.

1 Introduction

One of the main purposes of benchmarking analysis is to create a relatively realistic picture of the application performance. Commercially available supercomputers are usually advertised with their peak performance figures, which do not provide much information about the calculation speed for a particular application. Very often the real performance may be less by a factor of 10 and even worse from the available peak figures. In order to obtain more realistic performance results a number of benchmark suites have been developed, covering different aspects and types of parallel and distributed systems. Usually the assessment procedure is based on representative computationally intensive kernels and also full application codes. A problem arises when trying to compare the performance results of one benchmark on different machines. The discussion about the performance losses is still open to determine a qualitative measure for *suitability* of the application problem to the computer system.

Most existing distributed applications represent homogeneous problems. Their main feature is that each member of the underlying data domain is associated with a similar calculation time [2]. In this paper we concentrate our attention on *uniformly distributed applications.* These are load-balanced homogeneous applications placing equal number of members from the data domain in each node.

2 Programming Model

Let us consider a programming model of distributed memory message-passing systems with both scalar and vector processors in every node. In the most popular view, each node architecture consists of a communication element (CE), capable of handling message-passing functions; a local memory (LM), containing both data and program codes; a scalar processing element (SPE), performing the scalar operations; a vector processing element (VPE) and a vector cache memory (VCM), dedicated for a fast execution of vector operations (Fig. 1). Typical examples of such a system are the SUPRENUM computer [4], the iPSC/860, the Meiko CS-1 and CS-2, etc.

Taking the worst case assumption that there is no overlap between scalar operations, vector operations and communications, the total execution time (T) is a sum of the scalar execution time (T^s), the vector execution time (T^v) and the time for communications (T^c).

$$T = T^s + T^v + T^c \tag{1}$$

Every application and distributed-memory system, which conform to the above programming model can

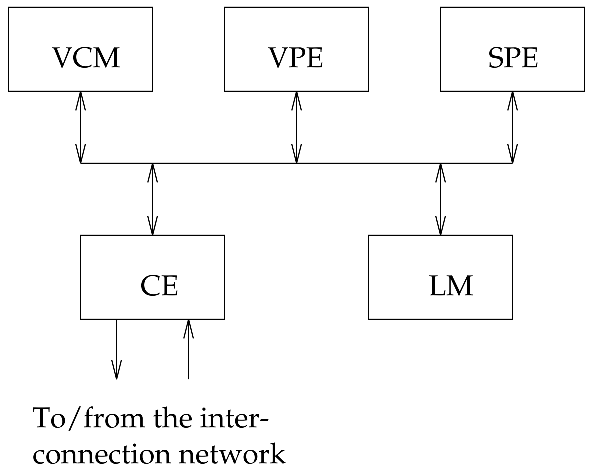

Figure 1: Node architecture of distributed-memory computers (CE - communication element, LM - local memory, SPE - scalar processing element, VPE - vector processing element, VCM - vector cache memory)

be presented by two sets of parameters [1]:

- $S(r^s_\infty, r^v_\infty, r^c_\infty, n^v_{1/2}, n^c_{1/2})$ - the computer system's parameters;
- $A(s^s, s^v, s^c, n^v, n^c)$ - the application's parameters.

The system's description is given in terms of the asymptotic performance (r_∞) and half-performance ($n_{1/2}$) parameters [8], whilst in the application's description s stands for the total amount of scalar, vector or communication work to be done and n is the average vector or message length respectively. The amount of work is measured appropriately by either the number of floating-point operations performed, or by the number of words transferred.

3 Timing Analysis

The scalar processing time depends on the total number of scalar floating-point operations (s^s) divided by the scalar computational rate (r^s_∞).

$$T^s = \frac{s^s}{r^s_\infty} \tag{2}$$

Because of the vector start-up time, the vector processing time is the sum of two terms. The first is the total number of floating-point operations executed in all vector instructions in the benchmark, divided by the asymptotic vector performance (r^v_∞). The second term is the total number of vector instructions and therefore start-ups ($q^v = s^v/n^v$), times the vector half-performance length and again divided by r^v_∞.

$$T^v = \frac{s^v}{r^v_\infty} + \frac{q^v n^v_{1/2}}{r^v_\infty} \tag{3}$$

The communication time may similarly be represented by two components arising from transmission time and start-up time:

$$T^c = \frac{s^c}{r^c_\infty} + \frac{q^c n^c_{1/2}}{r^c_\infty} \tag{4}$$

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 ($q^c = s^c/n^c$) 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.

The total node calculation time (T^n) is therefore

$$\begin{aligned} T^n = T^s + T^v &= \frac{s^s}{r^s_\infty} + \frac{s^v}{r^v_\infty} + \frac{q^v n^v_{1/2}}{r^v_\infty} = \\ &= \frac{s^s}{r^s_\infty} + \frac{q^v(n^v + n^v_{1/2})}{r^v_\infty} \end{aligned} \tag{5}$$

where $n^v (= s^v/q^v)$ is the average vector length.

If $n^v \to \infty$, we may neglect the effect of $n^v_{1/2}$, and $T^n \to T^n_\infty$, where

$$T^n_\infty = \frac{s^s}{r^s_\infty} + \frac{q^v n^v}{r^v_\infty} = \frac{s^s}{r^s_\infty} + \frac{s^v}{r^v_\infty} = \frac{s^n}{r^n_\infty} \tag{6}$$

In the last equality of (6) we have introduced the total arithmetic in the node as $s^n = s^v + s^s$ and the node asymptotic performance (r^n_∞) for this application which is the weighted harmonic mean

$$r^n_\infty = \frac{s^n}{\dfrac{s^s}{r^s_\infty} + \dfrac{s^v}{r^v_\infty}}. \tag{7}$$

[1]In these parameters we are using superscripts not as powers, but to distinguish scalar, vector and communication components.

4 Suitability Functions

The distributed-memory benchmark performance, R_B, is defined as a function of the problem size (N) and the number of processors (p) using the flop-count, $F_B(N)$, for the benchmark under consideration [9].

$$R_B(N,p) = \frac{F_B(N)}{T(N,p)} \tag{8}$$

Assuming that there is no redundant code being performed by the p processors, then the benchmark floating operation count $F_B = s^n p$. The flop-count and the total benchmark time can be substituted in (8) and after some transformations one can obtain

$$R_B(N,p) = \frac{r_\infty^n p}{1 + \frac{r_\infty^n}{r_\infty^v} \frac{s^v}{s^n} \frac{n_{1/2}^v}{n^v} + \frac{r_\infty^n}{r_\infty^c} \frac{s^c}{s^n} \left(1 + \frac{n_{1/2}^c}{n^c}\right)} \tag{9}$$

Here (9), the numerator contains the node peak performance (r_∞^n) as a function of the scalar/vector ratio of the application (7). As it was shown in [3] this formula may be used to represent the fit of the application to the architecture. Let us define two functions

$$\alpha^{n/v}(N,p) = \frac{r_\infty^n}{r_\infty^v} \frac{s^v(N,p)}{s^n(N,p)}$$

$$\alpha^{n/c}(N,p) = \frac{r_\infty^n}{r_\infty^c} \frac{s^c(N,p)}{s^n(N,p)} \tag{10}$$

The function $\alpha^{n/v}$ represents the suitability of the vectorised fraction of the application code to the vector/scalar ratio of the system. This function is always ≤ 1 and our wish is to make it as close to 1 as possible in order to ensure an efficient utilization of the vector processor. The function $\alpha^{n/c}$ may be used for measuring the fit of the application to the architecture in terms of calculation/communication ratio. Here s^n/s^c is the application's calculation/communication ratio or calculation intensity, while r_∞^n/r_∞^c is the corresponding calculation/communication ratio of the architecture. The best fit (but not the best performance) is when $\alpha^{n/c} = 1$. If $\alpha^{n/c} \ll 1$, then $R_B \approx R_\infty$, 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 $\alpha^{n/c} \gg 1$, then the performance dramatically decreases as a result of the very intensive communications leaving only a very short time for the main calculation work. Because of their use in the above way, we refer the functions α as *suitability functions*.

The node asymptotic performance (r_∞^n) is however a derivative parameter which depends on both computer system and application. Substituting for r_∞^n in the denominator of (9) one can now define three functions, which may be used for measuring the suitability of the application to the distributed-memory system under consideration. These new functions depend only on the prime system's and application's parameters.

$$\alpha^{v/s}(N,p) = \frac{r_\infty^v}{r_\infty^s} \frac{s^s(N,p)}{s^v(N,p)};$$

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

$$\alpha^{c/v}(N,p) = \frac{r_\infty^c}{r_\infty^v} \frac{s^v(N,p)}{s^c(N,p)}. \tag{11}$$

After some transformations one can derive the relations between the functions $\alpha^{n/v}$ and $\alpha^{n/c}$ on the one hand and the functions $\alpha^{v/s}$, $\alpha^{c/s}$ and $\alpha^{c/v}$ on the other hand.

$$\alpha^{n/v}(N,p) = \frac{1}{1 + \alpha^{v/s}}$$

$$\alpha^{n/c}(N,p) = \frac{1}{\alpha^{c/s} + \alpha^{c/v}} \tag{12}$$

5 Performance Analysis of 1-dimensional FFT

The distributed version of 1-dimensional FFT benchmark from the GENESIS benchmark suite [5] is 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, which conforms to our programming model and one may apply the suitability functions for performance analysis of the 1-dimensional FFT benchmark.

The performance analysis is based on the nominal operation count for the butterfly phase of the FFT algorithm – $5N \log N$ real floating-point operations. The distributed implementation of the butterfly phase

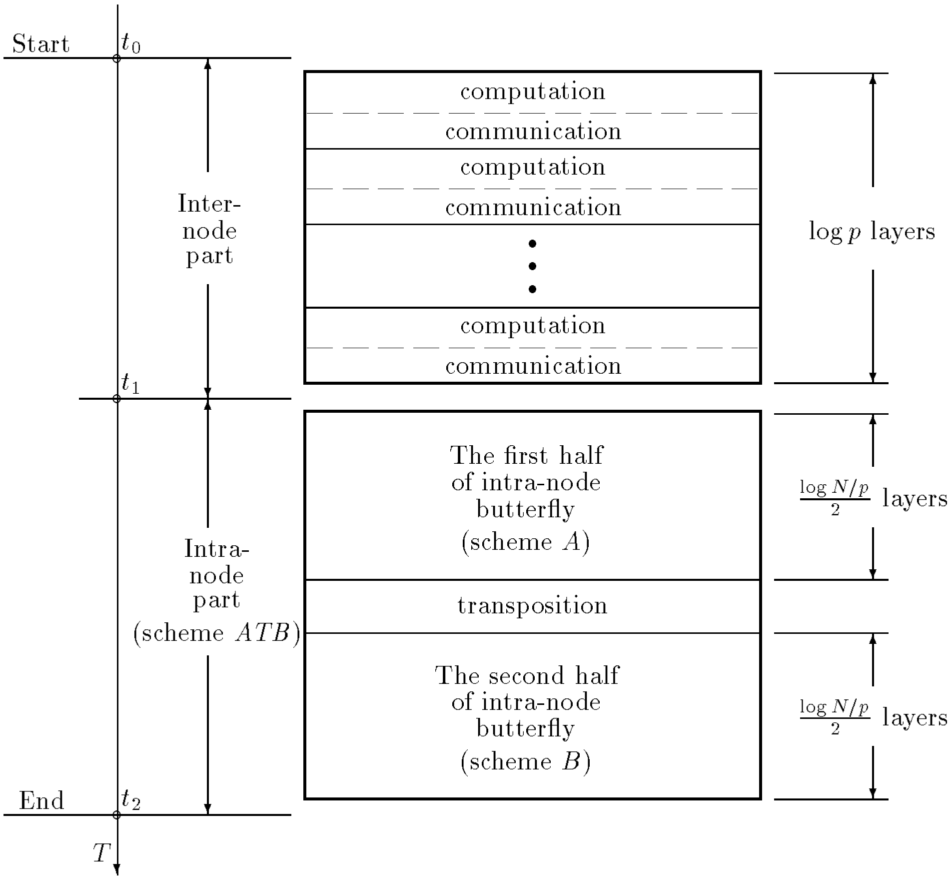

Figure 2: The FFT butterfly layering per node

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 [8]. 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 the vectorised version, a transposition is performed in the middle, bringing the implementation to scheme ATB.

In the definitions of suitability functions the parameters s^c and n^c have two arguments – N and p. In the case of 1-dimensional FFT N is the total number of data-points to be transformed and p is the number of processors. One can obtain the following expressions for the total amount of communication work to be done (s^c) an the average message length (n^c):

$$s^c(N,p) = \frac{N \log p}{p}; \qquad n^c(N,p) = \frac{N}{p}. \tag{13}$$

On the basis of the analysis above the following values for s^v and n^v can also be formulated:

$$s^v(N,p) = \frac{5N\log N}{p};$$

$$n^v(N,p) = \frac{N \log N}{2p\left(\log p + 2\left(\sqrt{\frac{N}{p}} - 1\right)\right)}. \tag{14}$$

The code of FFT1 does not have a scalar component ($s^s = 0$) and $\alpha^{n/v} = 1$ because $s^n = s^v$ and $r_\infty^n = r_\infty^v$. Thus, the vector performance formula of uniformly distributed applications is:

$$R_B^v(N,p) = \frac{r_\infty^n p}{1 + \frac{n_{1/2}^v}{n^v(N,p)} + \alpha^{n/c}(N,p)\left(1 + \frac{n_{1/2}^c}{n^c(N,p)}\right)} \tag{15}$$

$$R_B^v(N,p) = \frac{r_\infty^v p}{1 + \frac{2n_{1/2}^v p \log p}{N \log N} + \frac{4n_{1/2}^v p \left(\sqrt{\frac{N}{p}} - 1\right)}{N \log N} + \frac{r_\infty^v \log p}{5 r_\infty^c \log N} + \frac{r_\infty^v n_{1/2}^c p \log p}{5 r_\infty^c N \log N}} \qquad (16)$$

The benchmark performance can then be derived on the basis of (9) and after some transformations one can obtain the corresponding performance formula for the vectorised version of the 1-dimensional FFT (16). In this formula the performance is a function of N and p. All other parameters can be substituted by their measured or estimated values. For example, the SUPRENUM computer provides the following values for the system's parameters when running the FFT1 benchmark: $r_\infty^s = 0.188$ Mflop/s; $r_\infty^v = 3.5$ Mflop/s; $r_\infty^c = 0.6$ Mword/s; $n_{1/2}^v = 225$; $n_{1/2}^c = 4000$. The measurement results from the distributed benchmark version showed a very good fit to the predicted values by the above formula [3].

Table 1: Values of the suitability function $\alpha^{n/c}$ as calculated for the distributed memory 1-dimensional FFT benchmark

Problem size (N)	$p=2$	$p=4$	$p=8$	$p=16$
$4K$	0.097	0.194	0.292	0.389
$16K$	0.083	0.167	0.250	0.333
$64K$	0.073	0.146	0.219	0.292
$256K$	0.065	0.130	0.194	0.259

The suitability function $\alpha^{n/c}$ can now be calculated on the basis of (13), (14) and the system's parameters of the SUPRENUM computer.

$$\alpha^{n/c}(N,p) = \frac{r_\infty^v}{r_\infty^c} \frac{\log p}{5 \log N} \qquad (17)$$

The values of this function for four problem sizes of FFT1 are shown in Table 1 and plotted against the number of processors on Figure 3. The results show an underloaded communication subsystem in comparison to the calculations performed for the 1-dimensional FFT benchmark. This reduces slightly the negative influence of the half-bandwidth message length parameter ($n_{1/2}^c$), which is a result of the long message start-up time. The poor start-up time mean that high performance could only be achieved on problems with a coarse granularity and by code using relatively few, long messages, which fortunately is the case with the FFT1 benchmark. Obviously, the effect of underloaded communications decreases with the increase of the number of processors and the function $\alpha^{n/c}$ is slightly below 1 for the maximum number of processors within one cluster ($p = 16$). It is difficult to predict the balance of a multicluster system because we have not taken into account during the analysis the effect of the intercluster communications on SUPRENUM.

6 Discussion

As one can see from the definitions (10) the α-functions depend on the problem size (N) of the application and on the number of processors (p). From this point of view they can give additional information about the *scalability* of a given pair architecture/application in terms of both the problem size and the number of processors.

Any uniformly distributed application can be represented as a n-dimensional solid [1]. Parallelizing and tuning the application can be viewed as slicing, dicing, cutting, etc. operations over the application's solid. These operations are somewhat similar to the decomposition attributes defined in the Fortran D language [7] and also in the High Performance Fortran language specification [6]. The monitoring of the suitability functions could also be useful when performing *optimization* procedures to decide which one from the above operations to apply in order to achieve the best possible performance (Fig. 4).

7 Concluding Remarks

A number of suitability functions (α) are defined, which can be used for assessing the suitability of a given problem to a particular distributed-memory system, which may be both an existing installation or a new architecture under development. These functions could also be useful for the comparison of different versions of one application and also when performing optimization procedures to find out the best possible domain decomposition.

The list of α-functions is not limited to those shown above. In order to create a more detailed picture of

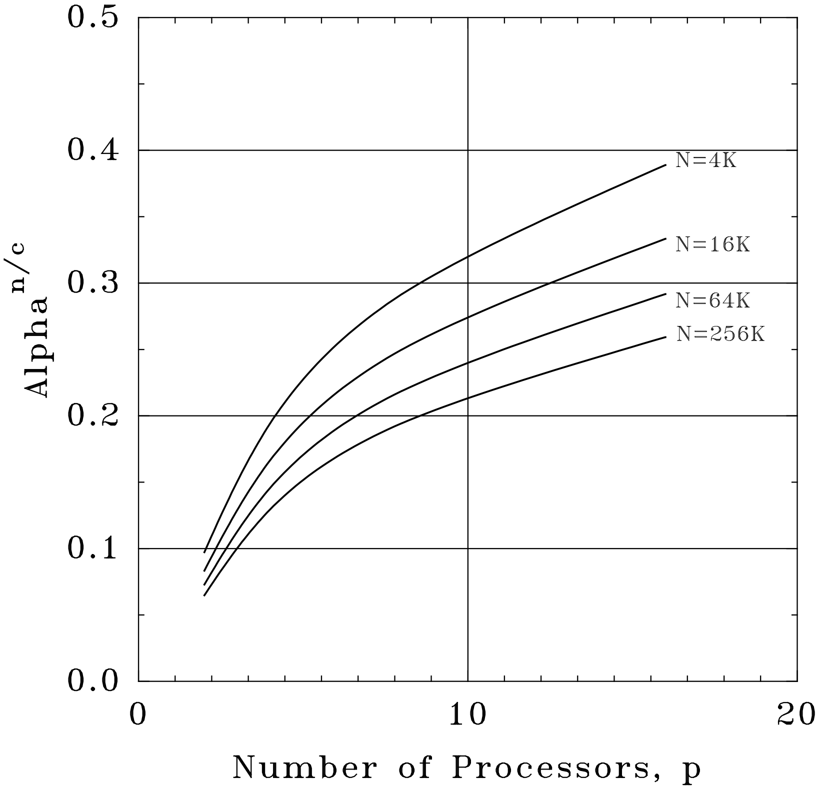

Figure 3: Suitability of the distributed 1-dimensional FFT benchmark to the SUPRENUM for different values of transform length, N represented by the function $\alpha^{n/c}$.

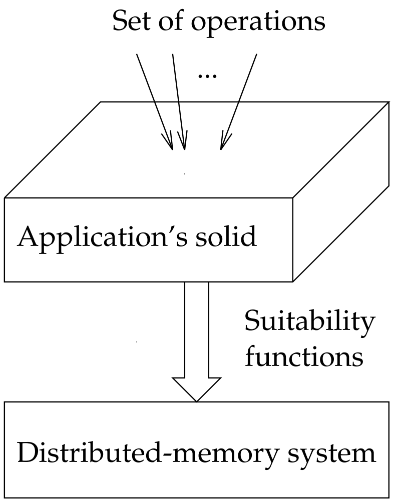

Figure 4: Tuning the application's decomposition by the suitability functions

suitability properties of a particular application, new functions can be defined, representing, for example, the influence of the memory bandwidth, the variable communication throughput, etc. It should also be mentioned that, in general, different architectures will have different sets of α-functions.

References

[1] L. Carter, *Private communication,* IBM T.J. Watson Research Center, Yorktown, May 1992.

[2] G. C. Fox, M. A. Johnson, G. A. Lyzenga, S. W. Otto, J. K. Salmon, D. W. Walker, *Solving Problems on Concurrent Processors 1: General Techniques and Regular Problems,* Prentice Hall, 1988.

[3] V. S. Getov, "1-Dimensional Parallel FFT Benchmark on SUPRENUM," In: D. Etiemble and J.-C. Syre (Eds.), PARLE'92 Parallel Architectures

and Languages Europe, *Lecture Notes in Computer Science,* Vol. 605, pp. 163-174, 1992.

[4] W. K. Giloi, "SUPRENUM: A Trendsetter in Modern Supercomputer Development," *Parallel Computing,* Vol. 7(3), pp. 283-296, 1988.

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

[6] High Performance Fortran Forum, *High Performance Fortran Language Specification, Version 1.0,* Technical report CRPC-TR-92225, Rice University, Houston, May, 1993.

[7] S. Hiranandani, K. Kennedy, C.-W. Tseng, Compiling Fortran D for MIMD Distributed-Memory Machines, *Communications of the ACM,* Vol. 35(8), pp. 66-80, 1992.

[8] R. W. Hockney, C. R. Jesshope, *Parallel Computers 2: Architecture, Programming and Algorithms,* Adam Hilger, 1988.

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