Comparative Performance Analysis of Uniformly Distributed Applications

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

Published in: *Proceedings Euromicro Workshop on PDP,*
IEEE Computer Society Press, January, 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 shown to be particularly useful for comparative performance analysis of both existing distributed-memory systems and new architectures under development.

1 Introduction

One of the main purposes of benchmark performance analysis is to give a relatively realistic picture of the performance of a given application. Commercially available supercomputers are usually advertised with their peak performance figures, which do not give much information about the performance of a given application. Very often the real performance may be less by a factor of 10 and even worse from the peak performance. 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. The assessment procedure is usually 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. In particular, the study of performance losses still requires the definition of appropriate functions to measure the suitability of a given application to any particular computer system.

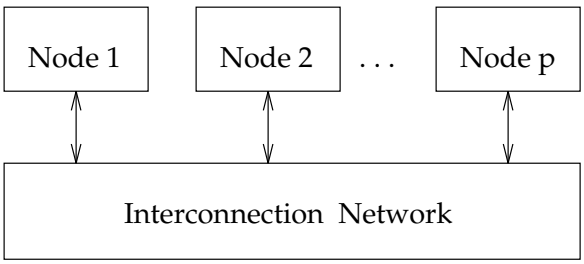

Figure 1: Distributed-memory computer system

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 [3]. In this paper we concentrate our attention on *uniformly distributed applications*. These are load-balanced homogeneous applications placing equal numbers of members from the data domain in each node.

2 Programming Model

Let us consider a programming model of distributed memory message-passing systems (Fig. 1) 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. 2). Typical examples of such a system are the SUPRENUM computer [5], the iPSC/860, the Meiko CS with i860 nodes [2], etc.

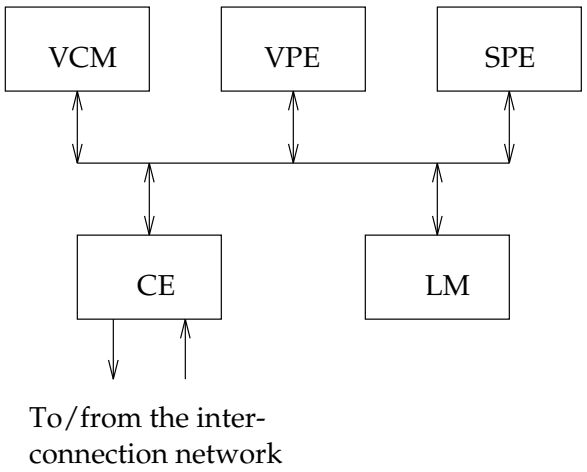

Figure 2: Node architecture of distributed-memory computers (CE - communication element, LM - local memory, SPE - scalar processing element, VPE - vector processing element, VCM - vector cache memory)

Taking the worst case assumption that there is no overlap between calculation and communication, the benchmark time (T) can be presented as a sum of the time for calculation and the time for communication. In the case of no overlap between scalar and vector operations the total execution time is a sum of the time for scalar operations (T^s), the time for vector operations (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 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 [7], 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.

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

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

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

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

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

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 [8].

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

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

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

As it was shown in [4] this formula may be used to represent the fit of the application to the architecture. Let us define two functions

$$\alpha^{n/v}(N,p) = \frac{r^n_\infty}{r^v_\infty}\frac{s^v(N,p)}{s^n(N,p)} \tag{10}$$

$$\alpha^{n/c}(N,p) = \frac{r^n_\infty}{r^c_\infty}\frac{s^c(N,p)}{s^n(N,p)} \tag{11}$$

The function $\alpha^{n/v}$ represents the suitability of vectorised fraction of the application 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. $\alpha^{n/c}$ may be used for measuring the fit of the application to the architecture in terms of calculation/communication ratio. 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 because 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*.

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.

$$\alpha^{v/s}(N,p) = \frac{r^v_\infty}{r^s_\infty}\frac{s^s(N,p)}{s^v(N,p)} \tag{12}$$

$$\alpha^{c/s}(N,p) = \frac{r^c_\infty}{r^s_\infty}\frac{s^s(N,p)}{s^c(N,p)} \tag{13}$$

$$\alpha^{c/v}(N,p) = \frac{r^c_\infty}{r^v_\infty}\frac{s^v(N,p)}{s^c(N,p)} \tag{14}$$

The new version of the performance formula is

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

5 Discussion

As one can see from the definitions (12- 14) 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.

Given an uniformly distributed application, one can represent it as a n-dimensional solid [1]. Parallelizing and tuning the application can be viewed as dicing, cutting, etc. operations over the application's solid. These operations are somewhat similar to the decomposition attributes defined in the Fortran D language [6]. The monitoring of the suitability functions could be useful when deciding which operations to apply in order to achieve the highest possible performance (Fig. 3).

6 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

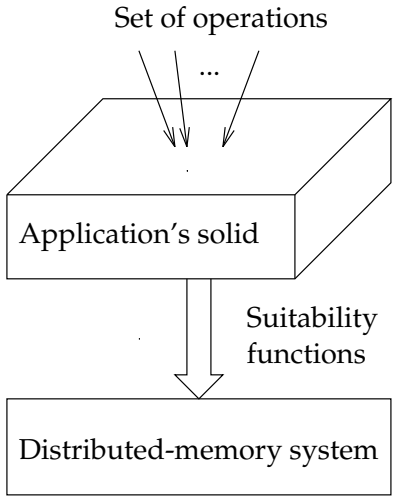

Figure 3: Tuning the application's decomposition by the suitability functions

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 more detailed picture of suitability properties of a given 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 different architectures will have in general different sets of α-functions. These questions are subject of future work.

References

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

[2] Dual i860 Element (MK096), *Information & Product News for Meiko Users,* Vol. 2, pp. 18-19, 1992.

[3] 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.

[4] 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.

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

[6] 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.

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

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