# 等同效率用於分析線性擴展記憶空間之平行電腦的延展性之探討

**林作俊**

國立宜蘭技術學院電子工程系副教授

## 摘 要

研究顯示給予一個可延展平行電腦夠多的資料量，它的運算效率可以維持在一個常數。這樣的特性，對於在有限的時間內解決運算複雜且資料量龐大的問題提供了美好的前景。在[1,2]中，作者提出等同效率函數用以分析平行電腦的延展性。然而他們的分析中忽略了記憶體對延展性的影響。在此篇論文中，我們以等同效率函數為基礎，提出在有限的記憶資源中分析平行電腦延展性的方法。我們所獲得的結論如下：當平行電腦之每個運算模組的記憶體容量固定時且該平行電腦的執行效率必須維持在一個常數時，此平行電腦之運算模組的數量不一定可以無限制的擴展。

關鍵詞：延展性、等同效率、平行電腦、記憶空間

# On the Isoefficiency Analysis of Parallel Computers with Linearly Scale-up Memory Space

**Cho-Chin Lin**

Associate Professor, Department of Electronic Engineering, National Ilan Institute of Technology

## Abstract

A scalable parallel machine may maintain its efficiency constant by increasing the amount of input data. Thus, it has the opportunity to challenge the applications of large scale. In [1,2], isoefficiency function has been proposed as a measure of scalability. However, their analysis ignored the amount of memory in a parallel machine. In this paper, the isoefficiency function of a parallel machine with memory constraint is proposed. Our result shows if the amount of memory of a parallel machine is linearly proportional to the number of processing nodes and the efficiency needs to be maintained at a desired level, then, the maximal number of processing nodes of the system may be bounded. This implies that the amount of available memory also affects the scalability of the system.

**Key Words**：Scalability, Isoefficiency, Parallel Computer, Memory Space

# I. Introduction

Since computers were developed, they have been employed to assist human being in handling daily events. Many of the applications concerning the issues of human welfare and the science leading to a better living environment need a large volume of computations. For example, the goal of improving atmospheric modeling resolved to a 5-km scale and providing timing result is believed to require 20 TFLOPS of performance [4]. In addition, several applications require petascale computing was also identified in the report of US President's IT Advisory Committee [7]. Examples are:

- Climate and environmental modeling
- 3D protein molecule reconstruction
- Design of advanced aircraft
- Intelligent planetary spacecraft
- Severe storm forecasting
- Real-time medical imaging
- Nuclear weapons stewardship
- Molecular nanotechnology

Before the technologies of non-conventional computing paradigms, such as quantum computation or DNA computing [8,9], become mature, von Neumann model [10] has been the major stream of building a sequential computer. However, we know that most powerful sequential computers of today cannot meet the computational requirement needed to implement the applications [12]. Thus, it is obvious that substantial progress in computing technology is still necessary for developing high performance computing platform in order to make a serious attack on the applications.

The advance in the microprocessor and memory technologies impacts the speed of a computer. The performance of a microprocessor is advancing at a rate of 50 to 100% per year [5,11]. Today, the state-of-the-art microprocessors can have computation speed up to hundreds of MFLOPS [12]. In addition to that, memory capacity is increasing at a rate comparable to the increase in capacity of DRAM chips: quadrupling in size every three years [6]. Current personal computers or workstations use hundreds of Mbytes. It seems that substantial progress has been achieved in sequential computer technology. However, the performance of the computer still cannot meet the applications of increasing complexity. Thus, scalable architectures, which employ parallel processing technology to accelerate the processing speed, have been proposed to meet the computational requirements.

Scalable architectures have the opportunity to challenge the applications of large scale. One of the architectural solutions for achieving scalability is the parallel machine of distributed memory. It is formed by combining essentially complete processing nodes through a network. A complete processing node consists of memory modules and a number of processors. Those nodes communicate by exchanging data through the network. In general, high speedup can be achieved by appropriately partitioning the data into several sets and mapping the sets to each of the processing nodes according to the computational characteristics of the applications. MPMD and SPMD are the programming paradigms employed in the parallel machines [14]. Standards for supporting such programming paradigms, for examples, are PVM and MPI.

To capture the performance behaviors of parallel machines, various performance models have been proposed based on different focus. Three performance models based on speedup metrics are: Amdahl's law, Gustafson's Law, and Sun and Ni's Law [13]. Amdahl's Law states that the maximal speedup achievable by a parallel computer is bounded by the ratio of the sequential portion to the parallel portion in a parallel program. It implies that there is an upper bound on the speedup of a parallel computer running an application of a fixed input size. However, there are many scientific problems concerning the precision of the results instead of accelerating the time needed to derive the results. In general, the more precise the result is, the larger the input size will be. Gustafson's Law states that substantial speedup can be achieved by increasing the input size of a problem to fully utilize the processing nodes. Sun and Ni's Law states that the amount of memory space of a parallel machine puts an upper bound on the speedup of the parallel computer. The scalability of a parallel algorithm on parallel architectures is used to measure how easily the linear speedup can be achieved as the number of processing nodes increases. In general, mapping a parallel algorithm to parallel architectures leads to various speedups tendencies. It implies that the evaluation of the performance of a parallel computer should consider the interaction between parallel algorithms and parallel architectures. However, the above metrics do not consider the effect of various mapping techniques. Thus, isoefficiency is proposed for scalability analysis in [2,3]. It has been used to show that a parallel computer may maintain its efficiency constant by increasing the amount of input data for a parallel algorithm. However, their analysis ignored the available memory in a computer. In this paper, the isoefficiency function of parallel architectures with linearly scale-up memory is analyzed. Our result shows that although the isoefficiency function for a parallel architecture exists, the efficiency may not be

maintained at a desired level when the number of processing nodes is increased beyond a particular limit. It implies that the scalability measurement should consider memory size of a processing node as well as the mapping techniques.

In this paper, background related to isoefficiency measurement is given in Section 2. In Section 3, the definition of expansion range is given and a methodology for deriving the expansion range is also proposed. Some interesting observations and discussions are given in Section 4. Conclusions are made in Section 5.

# II. Background

In [1,2], the authors stated that the scalability of a parallel algorithm on a parallel architecture is a measure of its capability to effectively utilize an increasing number of processing nodes. Thus, they developed a scalability metric called isoefficiency function. The function denoted as $f(p)$ relates the problem size to the number of processing nodes $p$. The function illustrates the tendency in increase of problem size in proportion to the number of processing nodes used in an application. The relationship between the isoefficiency function of an algorithm-architecture pair and some performance measures has been shown in [2]. In the following, we denote the term *system* to be an algorithm-architecture pair. The isoefficiency function can be used to measure how easily a system is able to achieve linear speedup as the number of processing nodes increases. Assume the isoefficiency of system $S_1$ is $f_{S1}(p)$ and the isoefficiency of system $S_2$ is $f_{S2}(p)$. If $f_{S1}(p)=\Omega(f_{S2}(p))$, then the scalability of system $S_1$ is no better than that of system $S_2$. Figure 1 illustrates this. In Figure 1, Systems $A$, $B$, and $C$ have isoefficiency functions $f_A(p)=p$, $f_B(p)=p\log p$, and $f_C(p)=p^2$. We can observe that as the number of processing nodes is increased, the amount of data size needed by system $C$ to maintain efficiency at a desired level is larger than systems A and B. The isoefficiency function of a system depends on the computation time, the communication time and the synchronization time. In general, the isoefficiency function can be computed by using the following equation:

$$E = \frac{T^*(n)}{p \times T(A, p, n)} \qquad (1)$$

where

- $E$ is the efficiency of the system.
- $T^*(n)$ is the amount of computation time taken by an optimal sequential algorithm with input size equal to $n$.
- $T(A, p, n)$ is the computation time of a parallel algorithm $A$ on a parallel computer with $p$ processing nodes. The algorithm $A$ is not necessary to be an optimal algorithm.

Since $T^*(n)$ is the amount of computation time taken by an optimal sequential algorithm, there should be no redundant work performed by $T^*(n)$. However, many parallel algorithms need to perform more steps of computations than an optimal algorithm does in order to reduce the number of parallel computation steps. Thus, the order of magnitude of $T^*(n)$ should be the lower bound of $p \times T(A, p, n)$. It implies that isoefficiency may not exist for certain systems.

However, in many situations, we do not consider asymptotically optimal algorithm in program design. One of the examples is Strassen's Algorithm [18]. The algorithm, which is asymptotically the most efficient algorithm for multiplying $n \times n$ matrices to date, has time complexity $\Theta(n^{\log 7})$. It is much better than *normal* matrix multiplication algorithm, which has time complexity of $\Theta(n^3)$. However, the *normal* matrix multiplication algorithm is the skeleton used for most parallel matrix multiplication design. It implies that when we consider isoefficiency function for scalability measurement, the base algorithm for comparison is not necessary to be an optimal algorithm. Some operations, which are unnecessary for an optimal sequential algorithm such as making several copies of a shared datum, may be performed in a parallel algorithm. Although it takes several steps to perform such operations, the execution time may be reduced. Let $T(1,n)$ be the time taken by a single processing node system to simulate the computations performed by a parallel system. Inter-processor communication and synchronization do not appear in the computations of the system with single processing node. Thus, the computations performed in $T(1,n)$ time units are pure computations. An alternate definition of efficiency which uses $T(1,n)$ and $T(A, p, n)$ is defined as follows:

$$E = \frac{T(1, n)}{p \times T(A, p, n)} \qquad (2)$$

The isoefficiency function derived by using Equation (2) shows how well the pure computations of an algorithm can be paralleled. The alternate definition of efficiency uses $T(1,n)$ as a base. Thus, we denote the isoefficiency function derived by this definition as the absolute isoefficiency function. The advantage of taking the absolute isoefficiency function as a scalability measure is as follows: for a parallel algorithm which can solve a problem in a small period of time by using larger amount of operations (than those performed by an optimal sequential algorithm), the isoefficiency function still exists. In contrast to the absolute isoefficiency function, the definition in Equation (1) is called relative isoefficiency function. In the following analysis, we use absolute isoefficiency function instead of relative isoefficiency function to measure the scalability of a system.

In [2], the authors also defined the Memory Overhead Factor (MOF) for an algorithm-architecture pair. The memory overhead factor is the ratio of the total memory required by all the processing nodes to the amount of memory required for the same problem size on a single processor. Note that the problem size is measured based on the amount of computations needed to be performed by the system to derive the result.

# III. Isoefficiency with Memory Constraint

In the design of a parallel algorithm, mapping data and program partitions to processing nodes of a parallel system is an important issue. Employing an appropriate mapping strategy can enhance the performance of the parallel system. Two types of computations, global computation and local computation, have been mentioned in [16]. Global computation uses distributed local memories as a large shared memory in which the data is distributed and shared. The computations are referred to as local computations, when the memory within a processing node is only used by the node during computations. For some applications, the communication time and the synchronization time can be reduced by distributing several copies of the same program or data to each of the processing nodes and concurrently perform the computations. That is, we may be able to reduce the communication and synchronization overhead of a distributed memory system by using local computation instead of global computation. The isoefficiency function of such a system may improve. However, the number of copies may increase in proportion to the number of processing nodes. It may occur that the memory of the system can be used up quickly due to multiple copies of data or programs. This implies that we need to consider the memory size constraint of a parallel computer, when the isoefficiency function of the system is computed. The following equation is used to illustrate the fact given by the above observations.

$$MEM_{arch}(p) \quad \geq \quad MEM_{alg}(n) \qquad (3)$$

where,

- $MEM_{alg}(n)$ is the amount of memory needed by the algorithm *alg* to implement an application. It is a function of input size $n$.
- $MEM_{arch}(p)$ is the amount of available memory space provided by the parallel architecture *arch*. It is a function of the number of processing nodes $p$ of a parallel computer.

If we combine Equations (2) and (3) then we see that the number of processing nodes of a system should be bounded. If the number of processing nodes exceeds this bound, then it is impossible for the system to maintain constant efficiency. We call this range the *expansion range* of a parallel system.

In the followings, we use all-pairs-shortest-path problem as an example to illustrate that the amount of memory of a parallel machine can affect the scalability of a system. In our example, all-pairs-shortest-path problem is solved by Parallel Floyd Algorithm (checkerboard version) running on a hypercube parallel computer. Assume there are $n$ cities to be considered. Thus, the input to the computer is the relative distances of the $n$ cities. The information of the distances between any pair of cities is stored in a matrix of size $n^2$. In the algorithm, the matrix is partitioned to $p$ squares of size $(n/p^{1/2}) \times (n/p^{1/2})$ and each square is mapped to one of the $p$ processing nodes. It is easy to derive the parallel running time of the algorithm is

$$T_p = \Theta\left(\frac{n^3}{p}\right) + \Theta\left(\frac{n^2}{\sqrt{p}}\log p\right)$$

The first term is the computation time and the second term is the communication time. By employing Equation (2), we can derive that the absolute isoefficiency function is $\Theta(p^{1.5}(\log p)^3)$. Interested readers should refer to [3] for details. Since the information of relative distances between all pairs of the cities is stored in a matrix of size $n^2$, the amount of memory needed by the algorithm is as follow:

$$MEM_{alg}(n) = \Omega(n^2) \quad (4)$$

Assume each node of the hypercube computer has memory size of $m$ units. Since the memory space increases in linear proportion to the number of processing nodes of the parallel computer, the amount of memory space provided by the parallel computer is as follow:

$$MEM_{arch}(p) = \Theta(mp) \quad (5)$$

According to Equations (3), (4) and (5), we have,

$$mp = \Omega(n^2) \quad (6)$$

The work size $W$ is defined as the amount of computation needed to derive the result of a problem. Thus, the $W$ of the shortest path problem is $n^3$. Then, we have the following equation:

$$W=\Theta(p^{1.5}(\log p)^3).$$

It implies that $n^2=\Theta(p(\log p)^2)$. By substituting $n^2$ in (6), we have:

$$p = O(2^{\sqrt{m}})$$

The equation suggests that linear speedup can be achieved by expanding the system from a single processing node to $p$ processing nodes of $m$ memory units, where $p = O(2^{\sqrt{m}})$. If we use more than $p$ processing nodes, then efficiency cannot be maintained constant (i.e. linear speedup is not possible) due to insufficient memory space.

# IV. Observations and Discussions

Based on the above analysis, we consider the expansion range of the parallel systems listed in [3]. The results are shown in Table 1. From Table 1, we can observe some interesting results. Some systems with good isoefficiency function may have smaller expansion range; however, some systems with poor isoefficiency function have larger expansion range. For example,

Floyd Checkerboard on Cube (System $S_1$):

— isoefficiency function $f_{S1}(p) = \Theta(p^{1.5}(\log p)^3)$.

— Expansion range is $O(2^{\sqrt{m}})$

Dijkstra Source_Parallel on Cube (System $S_2$):

— isoefficiency function $f_{S2}(p) = \Theta((p\log p)^{1.5})$.

— Expansion range is $O\left(\frac{m^2}{(\log m)^3}\right)$

The isoefficiency function of System $S_1$ is poorer than that of System $S_2$. However, the expansion range of System $S_1$ is larger than that of System $S_2$. Speedup of a system is equal to efficiency $\times$ number of processing nodes. Assume Systems $S_1$ and $S_2$ need to maintain efficiency at some desired level. Then, System $S_1$ has a larger expansion range and can achieve higher speedup.

# V. Conclusion and Future Direction

In this paper, we have shown that although the isoefficiency function of a system exists, it cannot maintain constant efficiency for any number of processing nodes exceeding its expansion range. This paper also shows that a system that needs to maintain efficiency at some desired level but has poorer isoefficiency function may have larger expasion range that leads to a higher speedup. In the future, we will extend our scalability analysis to the hetrogeneous clusters in which the memory may scale up non-linearly.

# Acknowledgment

This research was supported by the National Science Council, Taiwan, ROC, under Grant NSC91-2213-E-197-003.

# References

1. Anshul Gupta and Vipin Kumar (1992), "Analyzing Performance of Large Scale Parallel Systems," *Technical Report* 92-32. *Department of Computer Science, University of Minnesota*.
2. Vipin Kumar and Anshul Gupta (1991), "Analyzing Scalability of Parallel Algorithms and Architectures," *Technical Report. June, Dept. of Computer Science, University of Minnesota*.
3. Vipin Kumar and Vineet Singh (1991), "Scalability of Parallel Algorithms for the All-Pairs Shortest-Path Problem," *Journal of Parallel and Distrubuted Computation* 13, pp. 124-138.
4. Howard J. Siegel et al. (1992), "Report of the Purdue Workshop on Grand Challenges in Computer Architecture for the Support of High Performance Computing," *Journal of Parallel and Distributed Computing*.
5. IEEE Symposium Record – Hot Chips IV (1992) August.
6. J. L. Hennessy and D. A. Patterson (1990), Computer Architecture – A Quantitative Approach, Morgan Kaufmann.
7. Jack J. Dongarra and David W. Walker (2001), "The Quest for Petascale Computing," *IEEE Computing in Science and Engineering*, vol. 3, no. 3, pp. 32-39.
8. R. Hughes (2001), "Quantum Computation," *IEEE Computing in Science and Engineering*, vol. 3, no. 2, pp. 26.
9. Albert Y. Zomaya, James A. Anderson, David B. Fogel, Gerard J. MilBurn, and Grzegorz Rozenberg (2001), "Nonconventional computing paradigms in the new millennium: a roundtable," *IEEE Computing in Science and Engineering*, vol3, no. 6, pp. 82-99.
10. David A. Patterson and John L. Hennessy (1994), Computer Organization & Design: The Hardware/ Software Interface, Morgan Kaufmann Publishers.
11. Sriram Vajapeyam and Mateo Valero (April 2001), "Early 21st Century Processors," *IEEE Computer*, pp. 47-50.
12. J. J. Dongarra (2002), "Performance of Various Computers Using Standard Linear Equations Software, (Linpack Benchmark Report)," *University of Tennessee, Computer Science Technical Report*.
13. Kai Hwang and Zhiwei Xu (1997), Scalable Parallel Computing: Technology/Architecture/Programming, McGRAW-HILL International Editions.
14. Barry Wilkinson and Michael Allen (1999), Parallel Programming: Technology and Applications Using Networked Workstations and Parallel Computers, Prentice Hall.
15. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein (2001), Introduction to Algorithms, 2nd Edition, The MIT Press.
16. X. -H Sun and Lionel M. Ni (1990), "Another View on Parallel Speedup," *In Proceedings of Supercomputing* '90, pp. 324-333.

91 年 09 月 09 日投稿
91 年 09 月 30 日接受

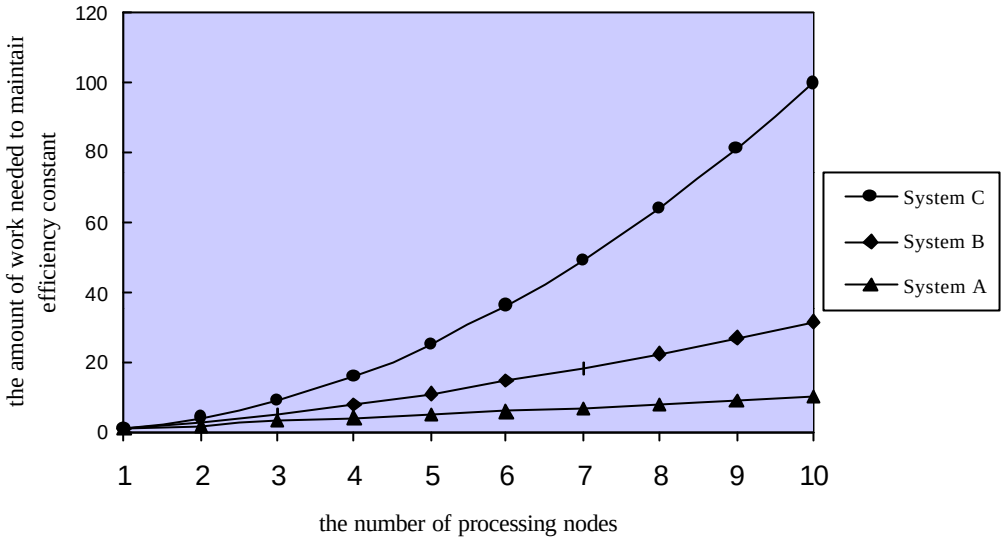


Fig 1　The isoefficiency functions of Parallel Systems *A*, *B*, and *C*.

Table 1　Comparison of the expansion of various algorithm-architecture pair.

| Base | Parallel Variant | Architecture | Isoefficiency | MOF | Expansion Range |
|---|---|---|---|---|---|
| Dijkstra | Source-Partition | Hypercube, | $p^3$ | $p$ | $O(m^{0.5})$ |
| Dijkstra | Source-Parallel | Hypercube | $(p\log p)^{1.5}$ | $n$ | $O(m^2/(\log m)^3)$ |
| | | Mesh | $p^{1.8}$ | $n$ | $O(m^{1.25})$ |
| Floyed | Stripe | Hypercube | $(p\log p)^3$ | 1 | $O(m/(\log m)^2)$ |
| | | Mesh (SF) | $p^{4.5}$ | 1 | $O(m^{0.5})$ |
| | | Mesh (CT) | $(p\log p)^3$ | 1 | $O(m/(\log m)^2)$ |
| Floyed | Checkboard | Hypercube | $p^{1.5}(\log p)^3$ | 1 | $O(2^{\sqrt{m}})$ |
| | | Mesh (SF) | $p^3$ | 1 | $O(m)$ |
| | | Mesh (CT) | $p^{2.25}$ | 1 | $O(m^2)$ |
| Floyed | Pipeline | Hypercube, | $p^{1.5}$ | 1 | $\infty$ |