ハイパフォーマンス
コンピューティング 57−11
（1995. 8. 24）

スケーラビリティに基づく 高並列計算機のパフォーマンス予測

古市 実裕、永松 礼夫、出口 光一郎
東京大学 工学部 計数工学科
〒113 東京都文京区本郷 7-3-1
E-mail :{furu,nag,deguchi}@meip7.t.u-tokyo.ac.jp

並列計算機のスケーラビリティは、計算機の性能向上を語る上で重要な概念であるが、その定量的評価やそれに基づく性能比較は容易ではない。これまで、並列計算機のスケーラビリティを評価するいくつかの手法が提案されている。しかし、それらの手法は、効率がプロセッサ数と問題サイズに関して単調であるという仮定を置いており、現実の局面で通用しない場合もある。本稿では新しいスケーラビリティの解釈を提案する。この解釈に基づく評価法は、特定のプロセッサ数と問題サイズにおける効率を測定し、パフォーマンスを特徴づけるいくつかのパラメータ (キャッシュサイズ, バンド幅,MIPS 値など) を用いてそれらを補間し、性能評価尺度の全体的な挙動を予測するものである。

Performance Estimation for Highly Parallel Computers Based on Scalability Concept

Sanehiro FURUICHI, Leo NAGAMATSU, Koichiro DEGUCHI
Faculty of Engineering, University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo, 113 Japan

Scalability of parallel computer systems is an important concept for considering the performance improvement of the system. But it is not easy to evaluate the scalability itself quantitatively or to compare performances based on it. Some evaluation techniques of the scalability of parallel computers have been proposed. But those are based on the assumption of monotonic feature of efficiency with respect to the processor number and the problem size, and sometimes not useful in real situations. We propose a new interpretation of the scalabitliy, where we firstly measure the performance for some pairs of the processor number and the problem size, then estimate whole performances inter/extra-polating them based on some system parameters such as cache size, bandwidth, MIPS and so on.

1 はじめに

並列計算機の性能を語る上で、プロセッサや相互結合網、入出力ポート等の単体の絶対性能の他に、それらの相互作用の結果であるシステム全体のスケーラビリティという概念がよく使われる。通常、プロセッサ台数を増加させた場合、通信オーバヘッド等の要因により、効率が低下し、コストに見合う(プロセッサ数に比例した)パフォーマンス向上が得られにくくなる。プロセッサ台数を増加させてもパフォーマンスが向上し続ける計算機を、スケーラビリティのよい計算機と呼ぶ。

スケーラビリティという概念は、定性的な意味で用いられることが多く、その定量的評価方法は曖昧なままであったが、最近、スケーラビリティに基づいて並列計算機を定量的に評価するいくつかの方法が提案されている[1, 2]。しかし、それらの手法は、プロセッサ数と問題サイズに対して効率が単調に変化するという仮定をおいており、現実の局面では通用しない場合もある。本稿では、計算機の性能評価尺度を、プロセッサ数と問題サイズの2者をパラメータとする関数で表現することで、それらを一般化する方法を提案する。この手法によれば、パフォーマンスを特徴づけるいくつかのパラメータを事前に知ることができると、プロセッサ数と問題サイズの任意の組合せに対するシステムの性能が予測できる。また、その結果からスケーラビリティを読みとれる。簡単な並列プログラムを例に、本評価手法の有効性を示す。

2 従来の性能評価手法

これまで、並列計算機システムにおける、プロセッサ数を増加させた時の性能向上を評価する様々な方法が提案されてきた。従来は、多くの場合、一つのアルゴリズムについて、問題サイズを固定のまま、プロセッサ数だけをパラメータとして、1台の時と比べた速度向上率(fixed-size speedup)を用いて性能評価を行なっていた。

しかし、一般の多くの問題がそうであるように、並列アルゴリズム中に逐次部分が存在すれば、この方法では、いずれは性能向上の限界に達することが、Amdahl の法則[3]として知られており、問題サイズを固定した評価の方式は実際的ではない。

それに対して、実際の場面では、プロセッサ数を増加させた時には、実行可能な問題サイズも増加させることができ、より高い性能が引き出せる。こういった状況の評価をするために、処理時間一定となるように問題サイズを増加させた時の速度向上率(fixed-time speedup)[4]や、各プロセッサで処理に要するメモリ容量を一定に保つように問題サイズを増加させた時の速度向上率(memory-bounded speedup)[5]などが、新たな評価方法として提案されている。

これらの方法は、いずれもプロセッサ1台で逐次処理した場合を基準にしているが、1台でのパフォーマンスが測定不能な場合には評価が行なえない。この欠点に対して、問題サイズから実行時間のオーダーが見積もれると仮定して、問題サイズで正規化することで、逐次実行時間を用いずに、効率の振舞いを表現するサイズ相対効率[6]が提案されている。

しかし、これらの方法でも、特定のプロセッサ台数における特定のアルゴリズムのパフォーマンスを基準にしているため、同じ問題を解く異なるアルゴリズムを同一の基準から比較できないという点では、並列計算機の性能評価法として不充分である。

さらに、これまでの方法は、あるプロセッサ数に対して特定の問題サイズでしか評価しないため、並列計算機と並列アルゴリズムのある一面の振舞いしか読みとることができず、計算機システム全体の性能について、充分な情報を得られなかった。

そこで、本稿では、実際に並列アプリケーションを動かす場合には、状況に応じて問題サイズは変化し、それによって性能も大きく変化するという点に注目し、パフォーマンスを2つのパラメータ(プロセッサ数と問題サイズ)を持つ関数として表すという一般化による評価方法を提案する。

3 効率の意味とその振舞い

3.1 効率の定義

並列計算機システムの性能評価尺度として効率を用いる。ただし、先に述べた理由から、プロセッサ数と問題サイズの両方をパラメータとして効率を求める必要がある。ここでは、効率 Eを、問題サイズ W、プロセッサ数 Nの関数として、以下のように定義する。

$$E(N,W) = \frac{S(N,W)}{S_{ref}} \tag{1}$$

ただし、$S(N,W)$はプロセッサあたりの平均速度で、実行時間 $T(N,W)$を用いて、

$$S(N,W) = \frac{W}{NT(N,W)}$$

と表される。また、S_{ref}は基準速度であり、何らかの意味でそのシステムでの「一番よい」性能である。問題サイズとは、その問題を解くのに要する計算コストであり、そのアルゴリズムで必要な演算回数であると定義した。ただし、後述の例では、乗算と加算のコスト比を考慮し、乗算回数のみを問題サイズとした。

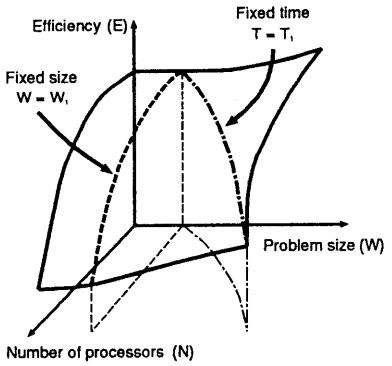

図 1: プロセッサ数と問題サイズの関数として表現された効率 (効率曲面)

効率は、プロセッサ数と問題サイズの両方をパラメータとする関数となる。N軸方向にプロセッサ数、W軸方向に問題サイズをとり、各点に対して E 軸方向にその点の効率をプロットすると、図1のように3次元空間の曲面の形で効率が表現できる。以後、この曲面を効率曲面と呼ぶ。

3.2 従来の性能評価方法との関連

効率を式(1)のように定義した場合、基準速度を何にとるかに応じて、効率の振舞いも変わってくる。

例えば、逐次実行時の速度を基準にする方法は従来からもよく用いられている。サイズ固定速度向上率(fixed-size speedup)[3]では、効率 Eは

$$E(N) = \frac{T(1, W_1)}{N\,T(N, W_1)} \tag{2}$$

と定義される。これは式(1)において、基準速度を逐次実行時の速度にとり、問題サイズをある値 W_1に固定した場合に相当する。図1の効率曲面上では、$W = W_1$の面による断面の形で表現される。

また、実行時間固定速度向上率(fixed-time speedup)[4]を用いた場合の効率 Eは、

$$E(N) = \frac{W_N}{N\,W_1} \tag{3}$$

と定義される。ただし、W_Nは、

$$T(N, W_N) = T(1, W_1)$$

を満たす問題サイズである。これは、式(1)において、逐次実行速度を基準にとり、実行時間を一定の値にするように問題サイズを増やしていく場合に相当する。図1では、実行時間をある一定の値 T_1 に固定するような (N, W) の組からなる曲面による効率曲面の断面に相当する。

3.3 効率とスケーラビリティ

スケーラビリティの考え方の一つに、*iso-efficiency* [2]がある。これは、問題サイズを増やすと、計算時間に対するオーバヘッドの比が減少し、効率は向上するという考えに基づいている。プロセッサ数増加時に効率を一定に保つように問題サイズを増やし、その増やし方によってスケーラビリティを判断する。効率をある一定値に維持するのに必要な問題サイズをプロセッサ数Nの関数として、

$$W = f_E(N) \tag{4}$$

と表す。ただし、効率の基準はその問題サイズでの逐次実行時間であり、

$$E = \frac{T(1, W)}{N\,T(N, W)} = const.$$

を満たす。式(4)を *iso-efficiency* 関数と呼び、この関数の挙動から、スケーラビリティを読みとる。例えば、$f_E(N)$ が指数関数的であるならば、効率維持に必要な問題サイズは大規模システムでは非常に大きくなり、スケーラビリティは悪いといえる。また、$f_E(N)$ が線形関数ならば、大規模システムでも実現可能な問題サイズなので、スケーラビリティはよいと判断できる。慣例では、$E = 1/2$ の場合について評価する。

これに対して、*iso-speed* [1]という概念は、問題サイズを増加させれば速度も向上するという考え方に基づき、プロセッサ数増加時にプロセッサあたりの平均速度 $S(N, W)$ を維持するのに必要な問題サイズで並列計算機のスケーラビリティを評価するものである。ここでは、プロセッサ数を N_1から N_2に増やした時のスケーラビリティ$\Psi(N_1, N_2)$ を、

$$\Psi(N_1, N_2) = \frac{W_1/N_1}{W_2/N_2} \qquad (N_2 > N_1) \tag{5}$$

と定量化した。ただし、W_1、W_2は、

$$S(N_1, W_1) = S(N_2, W_2) = const.$$

を満たす。通常は、$N = 1$、$W = \infty$ の時の速度(これを漸近速度と呼ぶ)の 1/2 の値を維持すべき平均速度として評価を行なう。

どちらの方法も、図2に示したように、効率曲面のある一部分(通常はE軸方向最大値の 1/2 の位置)を等高線

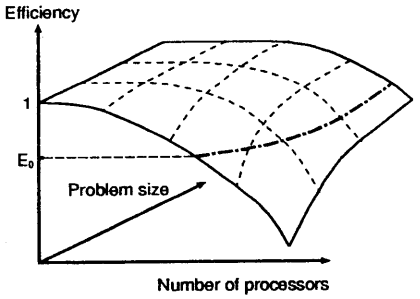

図 2: *iso-efficiency* に基づくスケーラビリティ評価

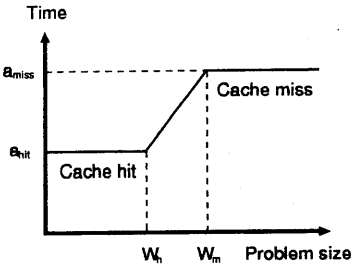

図 3: キャッシュ・ヒット/ミスする場合の計算時間

沿いになぞることで、システム全体のスケーラビリティを表現する方式である。両者の違いは、効率の基準の選び方だけである。

両者とも、効率の曲面は単調であり、曲面のある一部分から、システム全体のスケーラビリティを判断できることを前提としている。しかし、実際の並列計算機で並列アルゴリズムを実行する場合は、効率曲面は単調になるとは限らず、アーキテクチャやアルゴリズムの様々な要因によって、複雑な形状をするのが一般的である。我々の方法では、効率 Eを、プロセッサ数 Nと問題サイズ Wの関数として一般化するので、効率曲面が単調でない場合でも、スケーラビリティを表現することができる。

4 効率の 3 次元曲面表現

効率の基準速度は、異なるアルゴリズムの振舞いを同一の視点から比較可能にするため、プロセッサの計算能力 (FLOPS) にとることにする。効率の上限値は 1 で、これはプロセッサの計算能力を全て使い切ったことを示している。

一般に、Nが小さい時には、計算で必要なメモリが全部同時にキャッシュに乗らなくなることの効果、また、Nが大きい時には、プロセッサ間相互の通信が増大することによる影響で効率 Eは低下する。そこで、ある問題サイズ Wについて、特定のプロセッサ数 Nで効率 Eが極大となることが期待される。広い範囲の N、Wについて、効率 $E(N,W)$ の挙動がわかれば、そのような Nと Wの組合せを見つけ、並列計算機システムを有効に使うことができる。

しかし、すべての N、Wの組合せに対して効率 Eを測定し、その挙動を知ることは不可能である。そこで、この効率 Eの振舞いを、パフォーマンスを特徴づけるいくつかのパラメータから予測し、その予測値を用いて性能を評価する方法を提案する。本稿では、パラメータとして、キャッシュサイズ (C) と、キャッシュヒット・キャッシュミス時の計算時間 (a_{hit}/a_{miss})、通信メッセージサイズ (L)、通信回数 (m) を用いる。これらのパラメータはいずれも、逐次実行、あるいは小規模システムでの実行による測定や、プログラムコードの解析などから知ることができる。

実行時間に基づいてモデルを考えると、各要因の和として表現できるため理解しやすい。並列プログラムの全実行時間 Tを、計算部分 T_cとそれ以外のオーバヘッド部分 T_oに分割し、

$$T = T_c + T_o$$

とする。プログラムコードの解析などから、各プロセッサでの問題サイズ W が推測できれば、計算時間 T_cは、

$$T_c = a\,W$$

と見積もれる。ただし、a は単位問題サイズあたりの実行時間であり、この値は、問題サイズの大きさに応じて、データがキャッシュに入るときと入らない時とで異なる値を示す。キャッシュの振舞いは、図 3のようにモデル化した。W_hは全データがキャッシュヒットするような問題サイズの上限値、W_mは全データがキャッシュミスを生じる問題サイズの下限値である。

一方、オーバヘッド T_oは、プログラムと計算機アーキテクチャの相互作用に依存するので、容易に見積もることはできないが、いくつかの仮定をすることで、近似的にモデル化できる。

分散メモリ型マルチプロセッサでは、オーバヘッドは主にメッセージ通信による。通信オーバヘッドはメッセージサイズや通信回数、転送距離、ネットワークバンド幅などの関数として表現できる。ここでは通信サイズ Lと通信回数 m から、

$$T_o = b\,m\,L$$

と見積もれるとした。

5 性能評価の例：行列積問題

提案方法の適用例として、$n \times n$ 行列の乗算を Nプロセッサで並列実行した場合について述べる。行列を各プロセッサに N分割して配置し、負荷は各プロセッサで均一になるようにした。各プロセッサは、部分行列の乗算と隣接のプロセッサとの通信を繰り返しながら、全行列乗算を実行する。問題サイズ (乗算回数) は全体で n^3 となる。

5.1 予測モデル

各プロセッサは、$n \times \frac{n}{N}$の行列乗算を N 回行なうので、計算時間 T_cは、

$$T_o = a \times \frac{n^3}{N^2} \times N = an^3/N \qquad (6)$$

とおける。

また、キャッシュの効果によって、係数 a は変化するが、ここでは図3のモデルに従うとした。小規模システムで問題サイズとキャッシュの効果の関係を実測した結果から、全データがキャッシュヒットする問題サイズの上限値 W_hは、メモリ使用量がキャッシュサイズの約90%の時であり、また、全データがキャッシュミスを生じる問題サイズの下限値 W_mは、メモリ使用量が W_hの約2倍となる場合であることがわかっている。

一方、通信オーバヘッド T_oは、サイズ n^2/Nのメッセージを $N-1$ 回通信するので、

$$\begin{aligned} T_o &= b \times \frac{n^2}{N} \times (N-1) \\ &\approx bn^2 \qquad (N \gg 1) \end{aligned} \qquad (7)$$

と見積もった。

5.2 測定環境

上述のプログラムを並列計算機AP1000上で実行し、効率を求めた。AP1000は各プロセッサ間を結ぶT-net(トーラスネット)以外に、同期用のS-net、放送用のB-net[7]があるが、T-netのみを用い、更に転送距離を一定にするためセル構成を1次元トーラスに指定して実験した。

効率の推測モデルに用いた各パラメータを表1に示す。いずれも小規模システム構成での実測に基づく。

基準速度 S_{ref}		2.78 (MFLOPS)
1ステップの計算時間 a	cache hit	0.71 (μs)
	cache miss	1.20 (μs)
キャッシュサイズ C		256 (Kbyte)
通信バンド幅 b		0.5 (μs/byte)

表 1: AP1000 における測定パラメータ

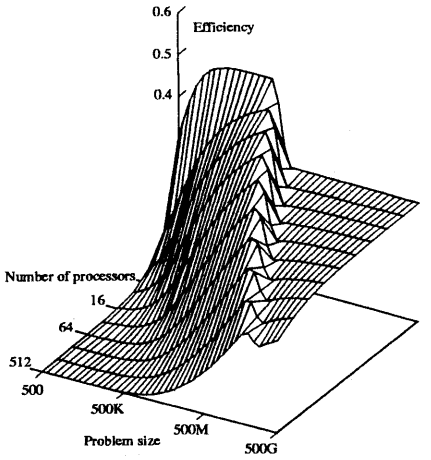

図 4: 行列乗算プログラムの効率 (モデルからの推測)

5.3 効率の3次元表示

行列乗算プログラムの実行で得られた効率の振舞いは、図4のように効率曲面で表示できる。スケーラビリティを評価しやすいように、等高線表示を図5に示す。

実際の効率と推測値との比較のため、図6にはいくつかの断面を示す。キャッシュの振舞いやオーバヘッドがほぼ正しく予測されていることがわかる。

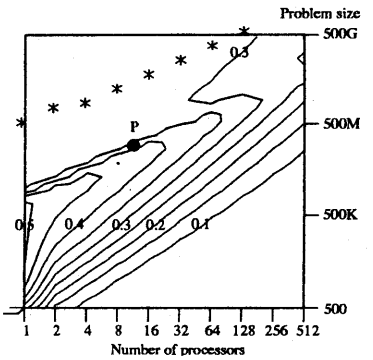

図 5: 効率 (モデル) の等高線表示

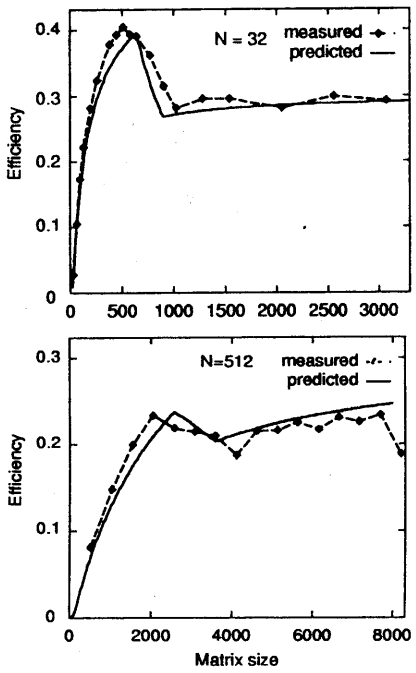

図 6: 実測値とモデルからの推定値

予測モデルでは、メモリ容量による問題サイズの制限を考慮していないが、実環境ではメモリ使用量が1プロセッサあたり16Mbyteという制限を越える領域(図5中の*印)は、実行不可能となる。

5.4 効率の振舞いとスケーラビリティ

効率を等高線表示すると、各線の方向がスケーラビリティを示すことになる。図5により、効率維持に必要なプロセッサ数と問題サイズとの関係が等高線の向きで判断できる。曲面中に尾根や部分的な凹凸が存在する場合、スケーラビリティは一様ではなくなる。

図5の場合、尾根線以外には特徴的な凹凸は存在しない。尾根以外の部分では、等高線の傾きは1なので、効率維持に必要な問題サイズは、プロセッサ数に対して線形に増加すると考えられる。また、図中P点近傍の、尾根の縁の部分では、傾きが約1/2となり、効率維持に必要な問題サイズの増加率はキャッシュ効果により、小さいことがわかる。

全体的に見て、傾きが2以上になる箇所はほとんどないことで、スケーラビリティがよいことがわかる。このプログラムは、各プロセッサの負荷が均一であり、また、隣接セルとしか通信を行なわない。このことからも、比較的スケーラビリティがよいと推測され、実験の結果とも一致している。

6 まとめ

並列計算機システムのスケーラビリティは、台数を増やした場合に計算能力をどれだけ有効利用できるかの指標であり、計算機性能の総合的評価に有用である。ところが、一般には効率の振舞いは複雑で、スケーラビリティを簡単に表すことは困難であった。

本稿では、効率をプロセッサ数Nと問題サイズWをパラメータに持つ関数と考え、3次元空間の曲面として表現し、その全体的な挙動を見ることでシステムのスケーラビリティを求める方法を提案した。これは*iso-efficiency*、*iso-speed*の概念を一般化したものとなっている。

また、(N,W)の全ての組合せについて効率を測定することなしに、逐次あるいは小規模システムでの効率だけから、全体の挙動を推測する方法を提案した。

効率を等高線表示すると、等高線の向きからスケーラビリティを知ることが容易になり、最適な(N,W)の関係を知ることができる。

計算時間とオーバヘッドの見積もりは、行列積の例ではプログラムコードの解析だけから比較的簡単に実現できたが、複雑な並列プログラムでは容易ではない。通信形態などが特定できず、時間見積もりが困難な場合には、部分的シミュレーションの併用などが考えられる。

参考文献

[1] Xian-He Sun and Diane T. Rover. Scalability of parallel algorithm-machine combinations. *IEEE Transaction on Parallel and Distributed Systems*, Vol. 5, No. 6, pp. 599–613, 1994.

[2] V.Kumar and V.Singh. Scalability of parallel algorithms for the all-pairs shortest path problems. *Proc. of Conf. on Parallel Processing*, pp. 136–140, 1990.

[3] G.Amdahl. Validity of the single-processor approach to achieving large scale computing capabilitiies. *Proc. AFIPS Conf.*, pp. 483–485, 1967.

[4] J.Gustafson. Reevaluating Amdahl's law. *CACM*, Vol. 31, pp. 523–533, 1988.

[5] Xian-He Sun and Lionel M. Ni. Scalabe problem and memory-bounded speedup. *J. Parallel Distrib. Computing*, Vol. 9, pp. 27–37, 1993.

[6] 佐藤三久, 関口智嗣. 並列計算機システムのスケーラビリティについて. 情報処理学会研究報告 HPC49-2, pp. 9–16, 1993.

[7] 堀江健志, 石畑宏明, 清水俊幸, 池坂守夫. 高並列計算機AP1000のアーキテクチャと性能評価. 電子情報通信学会技術研究報告 CPSY91-46, pp. 173–180, 1991.