Vol.2011-HPC-130 No.1
2011/7/27

時間モデルを用いた並列処理の性能評価
- 並列化部に隠れた並列オーバーヘッド -

折居茂夫†

ブラックボックスアプローチによる並列処理時間のモデル化方法を提案してきた．この方法は並列処理システムの情報を使わないので，システム中で生じる未知の並列オーバーヘッドをモデル化できる可能性がある．例として 2 つの並列処理システムの並列化部に隠れている並列オーバーヘッドをモデル化し，それを使って並列性能を評価する．

Parallel Performance Evaluation Using Time Model
- Parallel Overhead Hiding in Parallelized Parts -

SHIGEO ORII†

A time modeling method using a black box approach has been proposed to make up a time model of parallel processing systems. There is a possibility that unknown parallel overheads of parallel systems, which occur everywhere in the system, can be modeled because the method does not use any knowledge of the parallel processing systems. For examples, parallel overheads hiding in parallelized parts are modeled and evaluated using the time models.

1. はじめに

並列処理の性能は，扱う問題の種類，問題を記述したプログラム，それを動かす並列計算機また，実際に実行するときに扱う問題の大きさ，プロセッサ数等に依存して大きく変化する．従って条件によっては著しく低い性能で計算機システムを実行してしまう可能性がある．このような状況は並列処理の規模の大きさに関わらす生じる可能性があるが，ペタフロプスの計算能力を持つ数万台規模のプロセッサを持つ並列計算機システムについては，高速に処理するという観点だけではなくこのような膨大な資源の有効利用という観点から低い性能で計算しないことが重要である．これゆえ並列性能評価方法の研究は益々重要になっていると考える．

資源の有効利用という観点から性能評価を考えると，並列効率をその指標とすることがまず考えられる．そこで高並列処理に適用した並列効率メトリックを提案した[1]. この並列効率メトリックは実際の時間測定により決定することができるが，トライアンドエラーを繰り返してこれを求める方法以外に，シミュレータにより予測した処理時間を用いる方法が考えられる[2]. またモデル化された処理時間を用いることが考えられる．そこで処理時間を式化してモデル化する方法に着目し，その方法を提案してきた[3, 4]. 提案した方法は，一種のブラックボックスアプローチを用いた方法で，プログラムの内部構造，計算機の内部構造を解析してモデル化しなくても処理の時間をモデル化することができる．提案したモデル化方法ではまた，並列化部の処理時間中プロセッサ数の増加とともに減らない処理時間及び，増加する処理時間を並列オーバーヘッドとしてモデル化できる．

並列化部中の並列オーバーヘッド，一般にはあまり考慮されないが，高並列処理では大きな資源の無駄遣いにつながる可能性がある．またこのオーバーヘッドは発生場所や原因が不明な場合が多いので，まず処理の全体の性能に寄与するか否かを明確にすることが重要と考える．このような課題に対し，並列処理システムの内部構造を用いないブラックボックスアプローチは有効であると考えられる．

2 章ではブラックボックスアプローチを用いた並列処理時間のモデル化方法を紹介する．3 章では並列化部に隠れた並列オーバーヘッドの事例を示す．始めに並列化された多重ループに隠れた並列オーバーヘッドがモデル化できることを示す.次にコンパイラ制御行によって並列化された部分に隠れた並列オーバーヘッドがモデル化できることを示す.

2. 並列処理時間のモデル化方法

図 1 のように並列処理システムをブラックボックスと考え，その出力から得られた

† 富士通 (株)
Fujitsu Ltd.

情報で並列処理時間をモデル化する方法[3,4]を以下にまとめる．

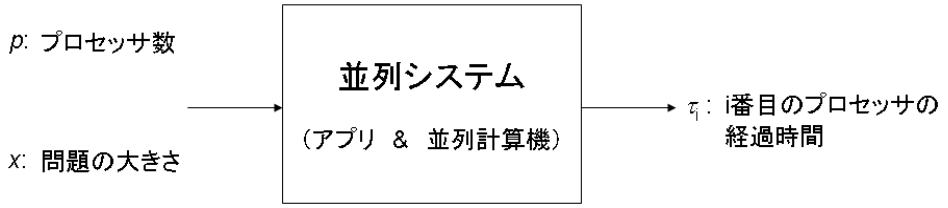

図 1　1 出力並列処理システム
Figure 1　One-output parallel processing system.

図 1 はある並列システムに対し，プロセッサ数 p と問題の大きさ x を入力した場合，出力として各プロセッサ i の経過時間 τ_i が得られることを想定している．紹介するモデル化方法は，これらの情報のみから並列処理システムの時間モデルを構築する．モデル化する時間は $\tau = Max_{i=1}^{p}(\tau_i)$ で，ロードインバランスでばらつく τ_i の中で一番長い経過時間即ちウォールクロック時間を念頭にモデル化することし，プロセッサ数と問題の大きさの関数としてモデル化し，式(1)でモデル化できると仮定した．ここに a は並列化部の時間中 p に対して反比例して減少する時間で，X は並列オーバーヘッドである．

$$p \cdot \tau \equiv a + p \cdot \mathrm{X} \tag{1}$$

ここで a の値に近い定数 Γ を導入し，式(1)の両辺から Γ を引いて変形すると式(2)を得る．

$$\frac{1-\varepsilon_p}{\varepsilon_p} \equiv \frac{1}{\Gamma}\left(a - \Gamma + p \cdot \mathrm{X}\right) \tag{2}$$

ここに $\varepsilon_p = \dfrac{\Gamma}{p \cdot \tau}$ で，τ は測定値であるので式(2)の左辺は測定値より求まる．

ここで式(2)の左辺が多項式で近似できると仮定して式(3)を得る．

$$\frac{1-\varepsilon_p}{\varepsilon_p} \equiv c_0 + p \cdot |c_1| + p \cdot \sum_{j=2} c_j (p-1)^{j-1} \tag{3}$$

式(2)と(3)を比較すると a と X について式(4)，(5)の関係を得る．ここで c_1 は逐次処理時間に対応する.しかし他項に比べて小さくなる場合等，フィッティングにおいて負の値になる場合があるため絶対値を取った．

$$a = \Gamma \cdot (1 + c_0) \tag{4}$$

$$X = \Gamma \cdot \left(|c_1| + \sum_{j=2} c_j (p-1)^{j-1} \right) \tag{5}$$

問題の大きさ x を固定し，プロセッサ数 p を変えて τ を測定し，その値を式(3)でフィッティングして $c_0, c_1, c_2 \cdots$ を決定すると，式(4)と(5)，式(1)より p を変数とした処理時間 τ のモデルを構築することができる．尚前回の報告[3, 4]では Γ 値にあるプロセッサ数と問題の大きさのときの Σ γ_i の値を用いた．

フィッティングに式(3)を用いる利点は 3 つある．第 1 に測定値 τ のみからモデルが構築できること．第 2 に処理中の全ての並列オーバーヘッドが X としてモデル化されること．第 3 にフィッティングする時間に $p \cdot \tau$ を用いており，τ 以外で生じる τ_i の変動を考慮しなくてすむことにある．一方欠点としては，多項式でフィッティングできる並列オーバーヘッドに, 適用できる並列システムが限定されてしまうことである．

これを改善するため図 2 のように新たに並列化部の経過時間 γ_i を測定し, a を式(4)からではなく τ_i とは独立に決定することを提案する．

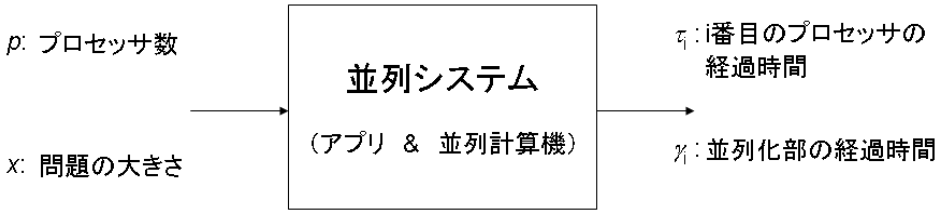

図 2　2 出力並列処理システム
Figure 2　Two-output parallel processing system.

測定した γ_i は式(6)のようにモデル化し，フィッティングにより a を決定する．ここに χ_{inv} は並列化部に隠れている並列オーバーヘッドである．

$$\sum_{i=1}^{p}\gamma_i / p \equiv a / p + \chi_{inv} \tag{6}$$

求めた a を $\Gamma = a$ として式(2)に代入すると式(7)を得る．

$$\frac{1-\varepsilon'_p}{\varepsilon'_p} \equiv \frac{1}{a}\left(p \cdot \mathrm{X}\right) \tag{7}$$

ここで $\varepsilon'_p = \dfrac{a}{p \cdot \tau}$ で，並列効率メトリック[1]を正確に表す量となる．

式(3)と同様，式(7)を式(8)のように多項式展開する．

$$\frac{1-\varepsilon'_p}{\varepsilon'_p} \equiv c_0 + p \cdot |c_1| + p \cdot \sum_{j=2} c_j (p-1)^{j-1} \tag{8}$$

式(7)と比較すると並列オーバーヘッドとして式(9)を得る．

$$X = a \cdot \left(\frac{c_0}{p} + |c_1| + \sum_{j=2} c_j (p-1)^{j-1} \right) \tag{9}$$

負の冪の項を持った式(9)により，式(5)でモデル化できなかった並列オーバーヘッドをモデル化できる．図 3 に X=Log(p)の例を示す．図 3 (a)は Log(p)の数値を生成して左辺とし式(8)でフィッティングした結果である．Fitting 1 は j=0, 1, 2 項を用いたフィッティング，Fitting 2 は j=0, 1, 2, 3 項を用いた．図 3 (b)は Log(p)値を点で，p･Log(p)を Eq. (8)でフィッティングして得た X のモデルを線で示す. 図中 Model 1 は Fitting 1 に，Model 2 は Fitting 2 に対応する時間モデルの線である. 図は Model 2 が Log(p)の点をほぼ結べることを示す.

X が式(4)のように通常の多項式で近時できる場合は，図 1 のように τ_i のみから時間のモデル化が可能である．近似できない場合は，図 2 の τ_i と γ_i を用いた式(6)，(8)によるモデル化によりモデル化できる場合がある. 式(8)における c_0 が他項と比べて無視できる場合は X のモデルは通常の多項式となる．

本論文では以下，上記で提案したモデル化方法を用い，モデル化により並列化部に隠れた並列オーバーヘッドを評価できようになることを例示する．これは問題の規模を固定して議論できるので，前回述べた多項式の係数を問題の大きさでモデル化する方法[4]の紹介は本論文では割愛する．

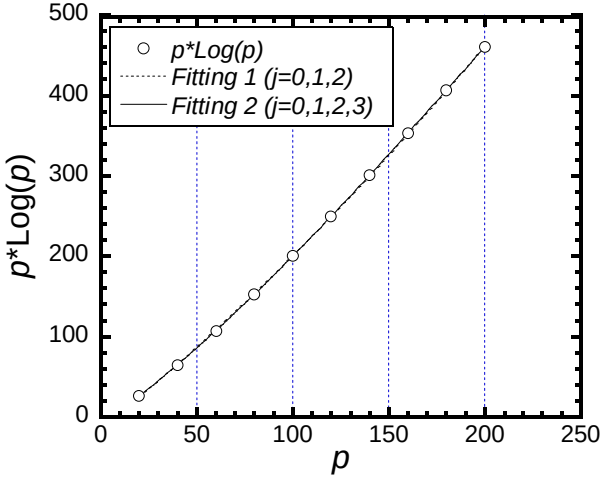

図 3 (a)　X=Log(p)に対する式(8)を用いたフィッティング結果
Figure 3 (a) Fitting results using Eq. (8) for X=Log(p).

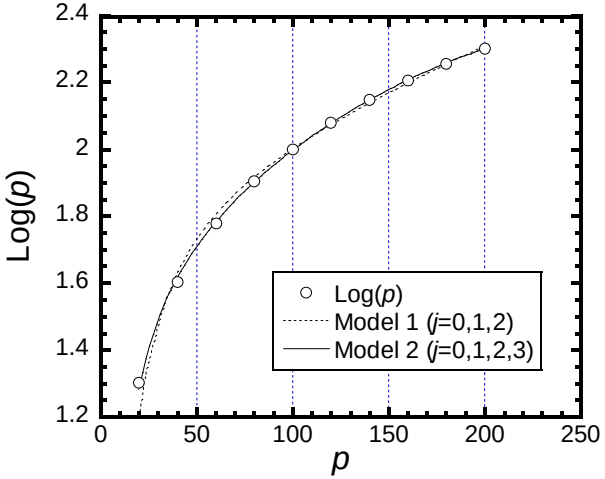

図 3 (b)　X=Log(p)に対する式(8)を用いた時間モデル
Figure 3 (b) Time models from Eq. (8) for X=Log(p).

3. 並列化部に隠れた並列オーバーヘッドのモデル化

3.1 並列化された多重ループに隠れた並列オーバーヘッド

図 4 は分子動力学シミュレーションの力の計算を Intel Paragon で並列化した MD コードの分子間力計算の一部である[5]. ここに粒子数 *n*, プロセッサ数 *p*= nnode である. 遮蔽距離を設けて計算量を削減しているので，各プロセッサが計算する粒子数は予め計算して icol に，その粒子番号は itab に格納されている．計算した各粒子に働く分子間力は fx，fy に蓄えられ，tgdsum [6]で全プロセッサの総和を行う.

このような 2 重ループの並列処理のモデル化はプログラムの構造を解析でモデル化するホワイトボックスアプローチによるモデル化では，do271 の回転数，データアクセスの時間モデルに種々の仮定が必要であった[7]が，図 2 のように MD コードと Paragon から成る並列処理システムを想定すると簡単にモデル化することができる.

```
      subroutine force(rcut2)
      dimension tmp(n)
      ・・・
      do 270 i=mnode+1,ipmax,nnode
         iii=iii+1
         xi=x(i)
         yi=y(i)
      do 271 k=1,icol(iii)
         j=itab(k,iii)
      ・・・
271 continue
270 continue
      call tgdsum(fx,n,twp)
      call tgdsum(fy,n,twp)
      ・・・
      return
      end
```

図 4　分動力学プログラムの分子間力計算の並列化部
Figure 4 Parallelized part of force computation of a molecular dynamics code [5].

ここでは図 4 の force の処理時間を並列処理システムの時間 τ とし，force と tgdsum の時間の測定値[5]から時間モデルを構築した例を示す． 図 5 (a)に示した *a*=153.0 は，Do 270 の実測値を γ として式(6)を用いて求めた結果である.

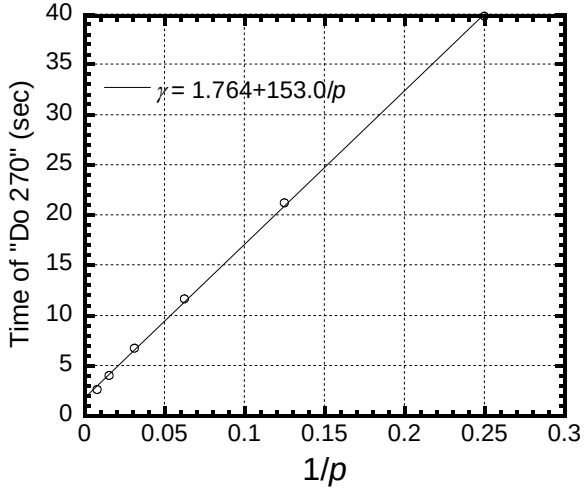

図 5 (a) force 計算の *a* 値の決定
Figure 5 (a) Estimation of a value of *a* for “force” computation.

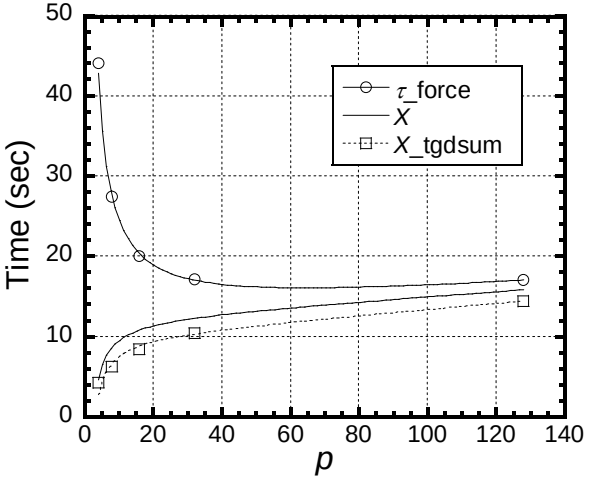

図 5　(b) Paragon における force 計算の測定時間とモデルの時間
Figure 5 (b) Times measured and modeled of “force” computation for Paragon.

これ用いて ε'_p を計算して式(8)でフィッティングし，時間モデル X = 153.0*(1/p - 0.2043/p + 0.08023 + 0.0001941*(p-1))を得る．また tgdsum を式(8)でフィッティングし X_tgdsum=153.0*(-0.1962/p+(0.06613+0.0002322*(p-1)))を得る．図５(b)に測定値とモデルの比較を示す．図は force の測定時間がモデル化とよく一致することを示す．モデル化した並列オーバーヘッド X と四角形で示した tgdsum の測定値を比べると約 2 秒大きく，並列化部の Do 270 中にどこで発生したか判らない並列オーバーヘッドが約 2 秒の逐次処理として存在することがわかる．この tgdsum の測定値は Log(p)に比例する．そこで式(８)を用いて tgdsum の測定値のモデルを作り測定値と比較した．結果を図５(c)に示す．丸で示したこの測定値をフィッティングした結果を実線で示す．点線は Log(p)でフィッティングした結果である．図はこの実線は Log(p)に比例する測定点を式(8)でフィッティングして結ぶことができることを示している．

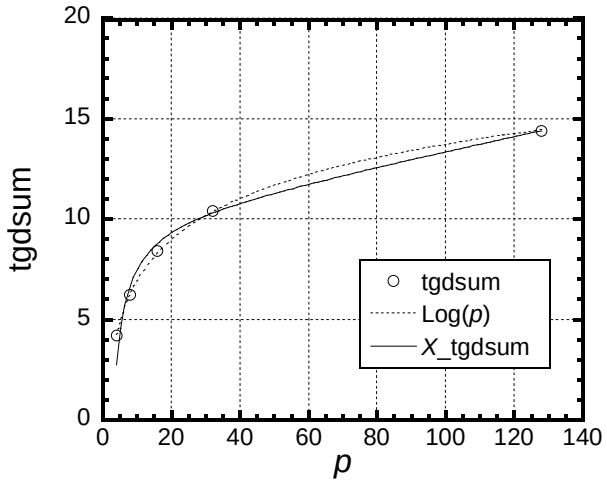

図５ (c) tgdsum の測定時間とモデルの時間
Figure 5 (c) Times measured and modeled of “tgdsum”.

3.2 コンパイラ制御行で並列化されたループに隠れた並列オーバーヘッド

図６は 3.1 節で述べた MD コードの force の部分を VPP300 の VPP Fortran のコンパイラ制御行を使って並列化したプログラムである[7].

```
      subroutine force(rcut2)
      parameter(n=7200)
      parameter(npe=10)
!xocl processor pe(npe)
!xocl subprocessor pes(npe)=pe(1:npe)
! xocl index partition ip=(pes,index=1:n,part=cyclic)
      Common/winter/itab(499,n)
! xocl local itab(:/ip)
       ・・・
! xocl spread do/ip
      do 270 i=1,n-1
       ・・・
! vocl loop,novrec
      do 271 k=1,icol(i)
       ・・・
  271 continue
      fx(i)=fx(i)+fxi$
      fy(i)=fy(i)+fyi$
  270 continue
! xocl end spread sum(fx),sum(fy)
      return
      end
```

図６ VPP Fortran で並列化された分動力学プログラムの分子間力計算
Figure 4 Parallelized force computation of a molecular dynamics code forVPP Fotran [7].

図中の! xocl end spread sum(fx),sum(fy)は 1 行で書かれているが，fx と fy に関するプロセッサ間の総和で，並列オーバーヘッドとなっている．この総和時間は VPP Fortran の知識がないと測定することができないが，！ xocl spread do/ip と! xocl end spread sum(fx),sum(fy)を時計で挟んで測定し，2 章で紹介したモデル化方法を用いると，総和時間とそれ以外の並列化された dio270 の中に隠れている並列オーバーヘッドとしてモデル化することができる．

図６に MD コードを VPP300 で実行した結果を示す．図中丸は force の測定値，三角は総和計算の測定値である．黒の実線は式(8)を用いて求めた時間モデル τ=640.9・(1/p +0.02830)で，測定値とモデルが良く一致していることを示す．尚 c_0 は小さい値なので零と置いてモデル化した．赤の実線はモデル化した並列オーバーヘッドで，X=18.1(=640.9･0.02830)とプロセッサ数に関係ない定数である．実測値は p=14 のとき X~11 で，これは並列オーバーヘッドの約 40%がプロセッサ間の総和以外のところで生じていることを意味する．モデル化により，今まで検知できなかったこの並列オーバ

ーヘッドが並列性能に寄与しており，プロセッサ数を増やして性能を向上することに対して大きな阻害要因であると評価することができる．

図 6 において明るい青の四角は並列効率メトリック ε'_p の値である．それを結ぶ曲線はモデルから得た ε'_p である．また p=14 までの測定値に基づいて作成した時間モデルによる性能予測として，ε'_p=0.5 のプロセッサ数は p=35 で，そのときの τ=30 秒を得る．このように時間モデルを構築すると，処理時間，効率とそれに対するプロセッサ数を定量的に評価できるようになる．

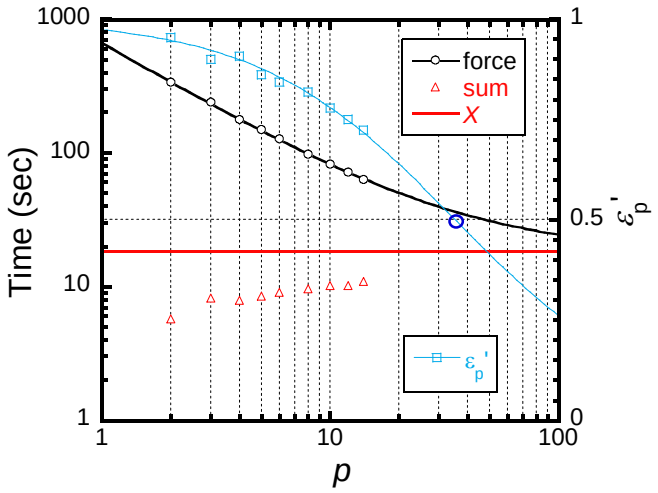

図 6　VPP300 における force 計算の測定時間とモデルの時間
Figure 6 Times measured and modeled of “force” computation for VPP300.

4. おわりに

これまで提案してきた図 1 の並列処理システムの処理の時間をモデル化する方法を紹介し，今回，図 2 による並列処理システムのモデル化方法を提案した．このモデル化方法により捕らえられた，今まで検知が難しかった並列化部に隠れた並列オーバーヘッドが，並列処理の性能に影響を与えることを 2 つの並列処理システム，MD コードと Paragon，MD コードと VPP300 によって例示した．VPP300 のコンパイラ制御行に適用できたので，OpenMP 等コンパイラ制御行を使ったマイクロタスクの並列化においてこのモデル化方法が適用できる可能性があると考える．

今回提案した方法でモデル化した Log(p)の時間モデルは外挿ができない．提案したモデル化方法が実際の並列性能評価でどのように使えるかは，今後の研究課題であると考える．

謝辞　富士通株式会社テクニカルコンピューティング･ソリューション事業本部の奥田基エグゼクティブアーキテクトの援助に感謝します．

参考文献

1)　S. Orii: Metrics for evaluation of parallel efficiency toward highly parallel processing, Parallel Comput, 36 (2010) 16-25.
2)　井上弘士：計算機シミュレータ BSIM, NSIM によるスーパーコンピュータの性能予測及び性能解析,学際大規模情報基盤共同利用・共同研究拠点 第 2 回シンポジウム 講演予稿，10-IS05, (2011).
3) 折居茂夫：高並列処理におけるスケーラビリティ評価方法，情報処理学会研究報告, 2009-HPC-121, Vol.2009, No.28, pp.1-6 (2009).
4) 折居茂夫：高並列処理におけるスケーラビリティ評価方法（Ⅱ），情報処理学会研究報告, 2010-HPC-126, Vol.2010, No.48 (2010).
5) 折居茂夫，分子動力学コードの段階的並列化手法, JAERI-Data/Code 96-023 (1996).
6) 大田敏郎，疎結合スカラ並列計算機のグローバル総和の高速化, JAERI-Data/Code 96-009 (1996).
7) 折居茂夫：数値計算のための並列計算機性能評価方法，情報処理学会論文誌，Vo.39, No.3 (1998).