社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－HPC－107 (23)
2006／8／1

タスク並列スクリプト言語MegaScriptのタスク動作モデルの検証

大村竜義[†] 北谷浩貴[†,☆] 中島浩[†,☆☆]

我々は，メガスケールコンピューティング向けの並列プログラミング言語として，MegaScriptを提案している．MegaScriptでは，ユーザがタスクの振る舞いをメタプログラムとして記述し，そのメタプログラムを解析することで，タスクの挙動や性能を示すメタモデルが生成される．

以前のメタモデルでは，MPIの集合通信を表現することが困難であったため．通信コストの産出に大きな誤差が生じ，その結果モデルの精度が低くなると言う問題があった．本論文では，メタプログラム，メタモデル，およびモデル解釈の各々について拡張・改良を行うともに，様々な実行環境下でのメタモデルでの精度の検証を行い，精度の高いモデルが構築できることを確認した．またある実行環境に対応するモデルを，他の実行環境に適応させる方法について，予備的な実験を行って課題を明らかにした．

Validation for the model of task performance of The MegaScript Task Parallel Script Language

TATSUYOSHI OHMURA,[†] HIROKI KITATANI[†,☆] and HIROSHI NAKASHIMA[†,☆☆]

We are pursuing researchs on our task-parallel script language named MegaScript for mega-scale computing. MegaScript has a unique feature named *meta-programming* by which a script programmer may give an abstract performance model of the tasks invoked from the script for efficient task scheduling and allocation.

This paper aims at improving the model accuracy especially for tasks with collective communications. We introduced a new framework to represent collective communications in a meta-program together with the model construction and interpretation mechanism for them. Our evaluation with three types of clusters shows that highly accurate models are constructed for all of them. We also conducted a preliminary experiment to adapt a model for a cluster to others to clarify the issues for the model adaptation.

1. はじめに

近年，数百Tera-FLOPSという計算能力を持つ数万台規模の並列計算機が登場している．しかし，地球レベルのシミュレーションやゲノム解析などの大規模かつ複雑な問題に対しては，数Peta-FLOPS以上の計算能力が必要であり，その性能を得るためには，100万大規模のプロセッサによる汎用メガスケールコンピューティングが必要になる．

しかし，従来からある専用並列計算機をメガスケール規模で構築・運用するためには，莫大なハードウェアを設置する巨大な施設や莫大な電力が必要となる．このため我々は，コモディティ技術を基にした「低電力化とモデリング技術によるメガスケールコンピューティング」を実現するための研究を行っている．

このメガスケール環境では，並列計算のプログラムは大規模化・複雑化してしまい，プログラムの記述することはもとより，その振る舞いを把握することが困難となる．このような極めて大きな規模の並列計算を容易かつ効率的に記述する方法として，我々は，タスク並列スクリプト言語MegaScriptを提案している.[1] MegaScriptでは，ユーザがタスクの振る舞いを抽象的に表現できるメタプログラムと，それに基づき特性を考慮したタスクスケジューリングを行う枠組みを用意している．

以前より我々は，このメタプログラムに基づき生成されるタスク動作モデルの精度の向上する研究を行っている．これまでの研究の結果，通信量が小さいあるいは通信パターンが単純なプログラムに対して，高精度のモデルが構築できることが明らかになっている.[2] しかし，複雑で大量の通信，特に集合通信を多用する並列タスクについては，モデルの精度が低いという問

† 豊橋技術科学大学
Toyohashi University of Technology
☆ 現在，(株) キューブシステム
Presently with Cube System Inc.
☆☆ 現在，京都大学
Presently with Kyoto University

```
class Pi < Task
  def initialize (*arg)
    @exefile = "./pi"
    @args = arg
  end

  def behavior
    FOR 0 .. @args[0]
      compute (2)
      IF 0.5
        compute (1)
      END
    END
  end
end
```

図 1　タスクの定義例

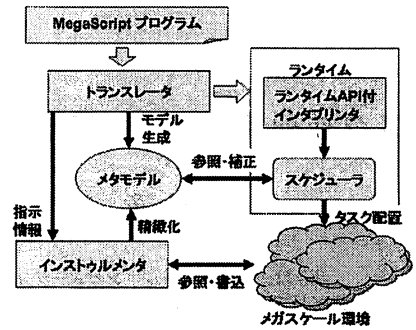

図 2　処理系の概要図

題があった．そこで我々は，メタプログラム，メタモデル，およびモデル解釈の各々について拡張・改良を行い，ライブラリや実行環境に対応したモデル解釈を行えるようにした．また，特定の環境で構築したメタモデルを用い，他環境下でのタスクの実行時間の推定を行った．

以下，本論文では，2 章で，本研究の背景となっているタスク並列言語 MegaScript システムについて述べる．3 章では，拡張したモデルについて述べる．4 章では，その評価について述べる．5 章では，他環境下でのタスクの実行時間の推定について述べる．最後に，6 章でまとめる．

2. タスク並列スクリプト言語 MegaScript

本章では，研究の背景である MegaScript について述べる．

2.1 概　　要

既存の並列処理ライブラリを用いて，大規模な並列プログラムを記述するのは非常に困難である．主な理由として，ユーザが並列性や粒度を考慮する必要があることと，実行環境を意識したプログラミングが必要になることがあげられる．そこで我々は，従来の枠組みで設計された並列または逐次のプログラムを下位層のタスクとし，それを多数組みあ合わせたタスク並列プログラムを上位層とする 2 階層並列プログラミングと，上位層記述のためのスクリプト言語 MegaScript を提案している．ユーザがタスクの特性を抽象的に記述する枠組みとしてメタプログラムを用意されている．この情報を元にスケジューラは，効率的なスケジューリングをすることができる．

2.2 タ ス ク

MegaScript の並列実行単位であるタスクは，外部プログラムとして与えられる．したがって，MegaScript のプログラムや処理系はタスク内部の動作には一切関与しない．また，タスクは**図 1** に示すように，タスククラスを継承する形で定義され，タスク情報は initialize メソッドと behavior メソッドに分けて記述される．さらに，複数のタスク間でデータのやりとりを行うために，「ストリーム」という論理的な通信路を設けている．これは，各タスクの外部から観測・操作可能な入出力である標準入出力を接続する方法で実現している。ストリームには，複数のタスクが接続可能で，マルチキャストやマージを行うことが可能である．

2.3 メタプログラム

効率的なスケジューリングをするためには，タスクの特性をあらかじめ知る必要がある．ユーザによるタスクの特性の記述方法として，タスクの計算量や通信量，必要なリソース情報などを直接記述する方法が考えられる．しかし，これらの情報だけでは，通信頻度やタイミングといったタスクの振る舞いを知ることができない．また，ユーザ自身がプログラムを解析し，コストを見積もる必要がある．この作業は，かなり困難な作業となる．

そこで MegaScript では，タスクの振る舞いに関する情報の記述方法として「メタプログラム」というスクリプトを用いる．メタプログラムは，図 1 に示すように．behaivior メソッド内に記述され，これを抽象実行した結果がタスクの特性を示すモデルとなる．このように，記述方式としてプログラムを用いているので，高い記述性があり，柔軟で詳細な記述が可能である．また逆に，以下に示すような実行時間を抽象的に表現する機能を用いて，プログラマの知識レベルに応じた概略的に記述することも可能である．

- 計算処理:compute()
- 標準入出力:input(), output()
- メッセージ通信:msgsend(), msgrecv()
- 制御構文:FOR, IF, PARALLEL

2.4 MegaScript 処理系

MegaScirpt の処理系の概要を**図 2** に示す．処理系は，トランスレータ，ランタイム，スケジューラ，インストゥルメンタから構成される．

トランスレータは，ユーザが記述したメタプログラムの実行解釈を行い，タスク動作モデルを生成する．

ランタイムは，基本的な並列実行機能をランタイムAPIとしてユーザに提供する．このAPIを用いて記述されたMegaScriptをランタイムが解釈し，メガスケール環境上でのタスクの配置やタスク間通信を実現する．また，システムリソースの状況をタスクの実行時間といった動的な情報収集も行う．

スケジューラは，ランタイム内部で動作しタスク配置戦略を決定する．スケジューラの動作は，静的スケジューリングと動的補正の2つからなる．スケジューラは，まずメタモデルと実行環境の情報をもとにスケジューリングを行い，タスクの初期配置を決定する．いくつかのタスクが終了すると，ランタイムから通知される動的情報をもとに，メタモデルの検証・修正／再構築を行い精緻化を行う．精緻化されたモデルをもとに再スケジューリングを行う．

インストゥルメンタは，高精度なモデルを構築する機構である．これは，メタプログラムに記述されたタスク内部情報取得指示関数に基づき，タスクプログラムにプロファイルコードを挿入する．それによって得られた情報を用いて，モデルを構築する．

3. MegaScriptのタスク動作モデル

本章では，メタプログラムから生成されるタスク動作モデルについて述べる．

3.1 基本概念

MegaScriptでは，効率の良いスケジューリングのために，スケジューラがタスクの特性を把握する必要がある．そのため，タスクのコスト予測モデルを用意する．これが，「タスク動作モデル」である．MegaScriptでは，このモデルの構築にメタプログラムを使用する．そのため，このモデルを「メタモデル」と呼ぶ．

メタモデルの実態は，メタプログラムからMegaScriptトランスレータによって生成され，スケジューラによって解釈実行されるプログラムである．この「メタモデル生成スクリプト」と呼ぶプログラムは，メタプログラムの抽象構文木に対応し，複数の構文木とノードから構成される．抽象構文木は，抽象化関数ごとに生成され，それぞれ関数名を名前として持つ．これを解析し得られる主なコストは，以下のようなメソッドを呼び出し取得することができる．

- comp_cost
  計算コストを取得するメソッド
- msgcomm_cost
  通信コストを取得するメソッド

3.2 集合通信モデル

メタプログラムに大まかに記述された計算や通信の挙動が，実際のタスクの実行挙動と大きくことなっていれば，モデルの精度が極めて低いものとなるのは当然である．計算については，並列タスクのアルゴリズムやプログラムについてある程度知識があれば，比較的正しい記述ができる．しかし，通信についてはこれらの知識だけでは十分ではないことがある．例えば，最も広く使われている並列計算・通信ライブラリであるMPIには，ブロードキャスト，all-to-allといった集合通信を行う関数が用意されている．しかし，これらが具体的にどのような通信を行うかはライブラリの実装に依存されている．したがって，メタプログラマに集合通信の具体的方法を要求することは困難であり，1対1通信のみで正しい通信アルゴリズムを記述することは期待できない．

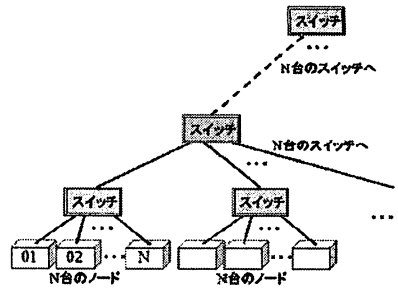

図3 多進木構成のネットワーク

そこで，最も広く使われている通信ライブラリであるMPIと，その代表的な実装であるMPICHを対象として，集合通信を適切にモデル化する方法について述べる．なお，本論文では，MPICH1.2.6を使用している．MPIで定義されており，並列計算でよく使用される集合通信は以下のものがある．

- MPI_Bcast
  特定のノード(ルートノード)が保持するデータを，他の全てのノードに送信する．
- MPI_Alltoall
  全てのノードから全てのノードに対して，宛先ノードごとに違うデータを配る．

3.3 ネットワーク形態による影響

集合通信では，ネットワーク構造を考慮しなければならないときがある．多数のノードを持つシステムでは，**図3**のように，スイッチを非終端ノードとする多進木構成のネットワークが用いられる場合が多い．このようなシステム上で，全ノード数に比例する量のデータ通信が起こる全対全通信では，上位層を経由する通信が時間的に集中して大量に生じるため，上位層がボトルネックとなる傾向が顕著となる．したがって，このような通信をモデル化する際には，通信アルゴリズムだけではなく，ネットワークの構造も考慮しなければならない．

3.4 集合通信関数モデル

本節では，主な集合通信関数においての，MPICHのアルゴリズムに基づく通信時間のモデルについて述べる．

3.4.1 MPI_Bcast

Bcast の通信時間を表現するコスト式 $Cost$ は，n をメッセージ長，p をノード数としたとき以下のように n の値に応じた計算式で表現される．なお，式中の α と β は，実行環境のネットワークのレイテンシとバンド幅の逆数を表現する定数であり，log の底は 2 である．

- $n \leq$ 12KB
  データを持っているノードが持っていないノードに送るという作業を繰り返すアルゴリズム binomial tree を用いており，通信コストは下式で表される．
  $Cost = \alpha\lceil \log p \rceil + \beta n \lceil \log p \rceil$
- 12KB $< n \leq$ 512KB
  binomial tree による scatter の後に，recursive doubling による all-gather を行うアルゴリズムを用いており，通信コストは下式となる．
  $Cost = 2\alpha\lceil \log p \rceil + 2\beta n \lceil \log p \rceil$
- $n >$ 512KB
  $n \leq$ 512KB と同様に scatter と all-gether により行うが，all-gather をリングアルゴリズムで実現するため，通信コストは下式となる．
  $Cost = \alpha(\lceil \log p \rceil + p - 1) + 2\beta n((p-1)/p)$

3.4.2 MPI_Alltoall

Alltoall の通信コストは，以下のようなモデル式で表される．

- $n \leq$ 256B
  Jehoshua Bruck のアルゴリズムを用いており．通信コストは下式で表される．
  $Cost = \alpha\lceil \log p \rceil + \beta(np/2)\lceil \log p \rceil$
- 256B $< n \leq$ 32KB
  isend と irecv を使い，全ノードがそれぞれのノードにデータを配る方法を用いているため，通信コストは下式となる．
  $Cost = \alpha(p-1) + \beta np$
- $n >$ 32KB
  ノード数が 2 のべき乗の場合は MPI_Sendrecv を用いた pairwise exchange で，また 2 のべき乗でない場合は $n \leq$ 32KB と同様の方法を用いる．したがっていずれの場合も通信コストは下式となる．
  $Cost = \alpha(p-1) + \beta np$

Alltoall については，ネットワークが非均質な階層構造である場合，上位層のボトルネックによる影響が現れる．そこで，この影響を考慮したモデルを構築する．

ネットワーク構造にボトルネックが生じる最も簡単な状況は，図 4 に示すように上位層の枝次数が 2 であり，下位層の枝次数が等しいような，2 レベルの木構造である．このような木構造は，中規模クラスタでは一般的なネットワーク構成である．

Alltoall 通信の過程で，このような 2 分割の木構造の

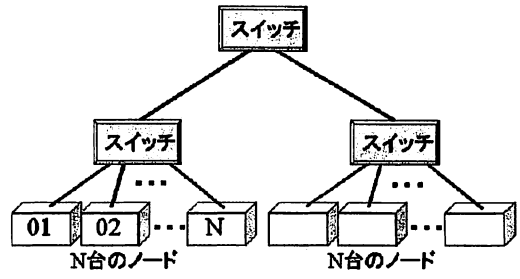

図 4 2 レベルの木構造ネットワーク

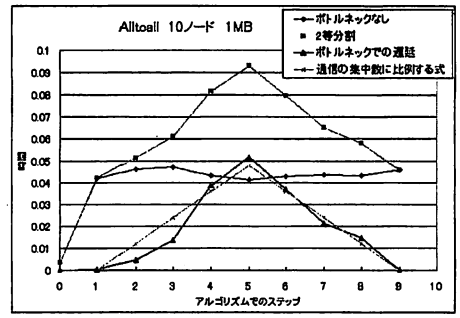

図 5 Alltoall のボトルネックでの遅延

ボトルネックがどのような影響を及ぼすかを，10 ノードの Pentium III(866MHz) を Giga-bit Ethernet で接続したクラスタを用いて，(a) 全ノードが 10 ポートのスイッチに直接接続される場合 (ボトルネックなし)，および (b)5 ノードずつが別個の下位スイッチに接続された場合 (2 等分割)，のそれぞれについて測定した．

図 5 の測定結果に示すように，Alltoall を構成する 10 ステップの中で，他のノードにデータを配送するステップ 1〜9 の所要時間は，ボトルネックのないフラット環境ではほぼ一定となる．一方，上位のスイッチとリンクがボトルネックとなる場合，それらを経由する通信ペア数に比例した時間が，フラット環境の通信時間に加えられることが分かる．この結果より，ボトルネックによる遅延が通信の集中に比例すると仮定すると，次の式が得られる．

- 2 等分割され中央にボトルネックのある環境での Alltoall
  $Cost$ = フラット環境での通信コスト + 混雑による遅延
  = フラット環境での通信コスト + 混雑数あたりの遅延時間 ×(ボトルネック通過通信回数 − a)
  $a = \begin{cases} p/2 & p = 2^k \\ p-1 & p \neq 2^k \end{cases}$

3.5 コストと実行時間

本節では，計算・通信コストといくつかの実行結果

から，高精度の性能モデルを構築する方法について述べる．

そもそも，MegaScript では，compute(10) で与えられる計算コスト 10 や，msgsend(20) で与えられる通信コスト 20 の意味は，メタプログラマが自由に決めることができる．すなわち，compute(10) の 10 とは 10 秒の計算なのか，あるいは 10 回の計算なのかといったことを設定でき，1 つのメタプログラム内で統一されていればよい．

また，compute(10) は計算量を，msgsend(20) は通信量を，それぞれ何らかの単位で表現したものであり，両者のコストの値はまったく別次元のものである．その一方で，スケジューリングの際には，1 つのタスクプログラム全体のコストとして何らかの値を参照する必要がある．そこで，別次元のコストを統合して扱う手段を考える．

ここでスケジューラにとって必要なタスクのコスト情報は，タスクの実行時間を何らかの単位で表現した値であり，計算・通信コストが実行時間に線形に寄与すると仮定すれば以下の式で表現することができる．なお，以下の式の a，b をスケーリング係数と呼ぶ．

実行時間 $= a \times$ 計算コスト $+ b \times$ 通信コスト

さて上記の式によって実行時間がモデル化できるとしても，2 つのパラメータ a と b の値を求めなければならない．これらはメタプログラマの意味づけに依存するため，タスクに独立な定数とすることはできない．一方，スケジューラは，一般にスケジューリングのためにメタモデルを利用するので，一つのタスクプログラムに対して，計算・通信コストが異なる値を持つような複数の実行を行うものと考える．そこで，計算・通信コストが異なる複数の実行履歴を用いて，a と b のパラメータを推定することとする．

具体的には，複数の異なるコストとそれらに対応する実行時間を，a と b を未知数とする過剰条件連立方程式とし，その近似解を最小二乗法によって求めることで，特定のタスクと a と b の値を推定する．

4. 評　　価

前章で述べた集合通信関数とネットワーク形態による影響を考慮したメタモデルを検証するために，様々な実行環境下で評価した．

4.1 評価環境

評価環境を，**表 1** に示す．全てのクラスタのネットワークは，全て Gigabit Ethernet で構成されている．また，クラスタ A のネットワーク形態は 10 ノードずつ 5 つのスイッチに接続され，その 5 つのスイッチをもう 1 段上のスイッチでまとめるというツリー構造になっている．そのほかのクラスタのネットワーク形態はスイッチ 1 つのフラットな環境である．

表 1　評価環境

	node	CPU	周波数	Memory
クラスタ A	50	Intel Pentium III	866MHz	512MB
クラスタ B	16	Intel Xeon	2.8GHz	1GB
クラスタ C	16	Efficeon TM8820	1GHz	512MB

4.2 評価結果

評価プログラムとして，NAS Parallel Benchmarks3.1 の FT プログラムを用いた．FT(3-D FFT PDE) ベンチマークは，3 次元偏微分方程式を FFT を用いて解くプログラムである．

FT では，3 次元配列のデータ交換に Alltoall が利用される．評価結果を**図 6**，**図 7**，**図 8** に示す．これより，モデルから得られる予測実行時間とプログラムの実実行時間の誤差は，最大 15%，平均 3% であり，様々な実行環境で十分に精度が高いことが分かる．また，クラスタ A のようなネットワーク形態による影響がある環境下でも十分に精度が高いことも分かる．

5. 他環境下における実行時間の推定

前述の最小二乗法による推定では，正確なモデルは得られるが，環境の違うクラスタが多くなればなるほど必要なデータ数が増加する問題がある．そこで，この問題を解決するために，特定の実行環境下での実行時情報とメタモデルのコストを用いて，他環境下での実行時間の推定を行う．

5.1 推定方法

3.5 節で述べた a，b の値が，特定の実行環境下では求まっていることを前提とする．そして，計算・通信コストを静的な実行環境の情報を元に補正し，実行時間の推定を行う．

計算時間は，計算量と CPU の周波数や FLOPS といった計算性能に関係している．そこで，計算コストを計算性能を元に補正を行う．特定環境下におけるベンチマークの結果 (FLOPS などの速度指標) を ρ_c とし，他環境下におけるベンチマークの結果を ρ'_c とすると，計算コストの補正値は，ρ_c/ρ'_c となる．

一方，通信時間は，ネットワークのバンド幅の影響が強い．そこで，通信性能を元に通信コストの補正を行う．特定環境下におけるベンチマークの結果 (バンド幅などの速度指標) を ρ_m とし，他環境下におけるベンチマークの結果を ρ'_m とすると，計算コストの補正値は，ρ_m/ρ'_m となる．以上より，他環境下での実行時間は次のように推定できる．

$$cost = a \times \text{計算コスト} \times \rho_c/\rho'_c + b \times \text{通信コスト} \times \rho_m/\rho'_m$$

5.2 評　　価

本節では，クラスタ A のメタモデルを元に，クラスタ B・C の a，b を推定し．前節で述べた推定法の評価を行う．計算コストの補正のために，姫野ベンチ

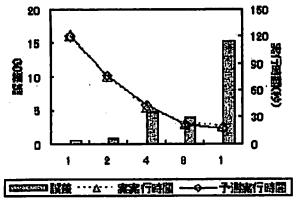

図 6 クラスタ A での FT の結果

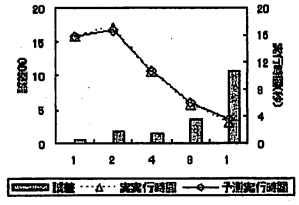

図 7 クラスタ B での FT の結果

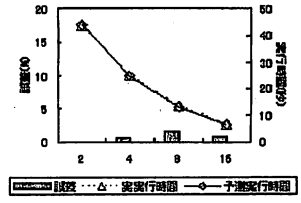

図 8 クラスタ C での FT の結果

表 2 性能係数の推定

	a	b
クラスタ B		
ベンチマーク推定値	2.23×10^{-9}	3.05×10^{-8}
最小二乗法推定値	1.48×10^{-9}	2.43×10^{-8}
クラスタ C		
ベンチマーク推定値	1.26×10^{-8}	6.36×10^{-8}
最小二乗法推定値	5.79×10^{-9}	2.76×10^{-8}

マークの結果を使用した．また，通信コストの補正のために，iperf の結果を使用した．

評価プログラムには 4 章と同じ FT を用い，8 ノードの性能推定値を最小二乗法による推定値と比較した．

まず，クラスタ A の a，b を最小二乗法による推定値は，$a = 1.12 \times 10^{-8}$，$b = 3.05 \times 10^{-8}$ となり．これに基づく実行時間の推定誤差は図 6 に示したように 0.4〜15.3% と小さな値になっている．

次に，前述のように $a' = a \times \rho_c/\rho'_c$, $b' = b \times \rho_m/\rho'_m$ としてクラスタ B と C の係数を推定した値（ベンチマーク推定値）と，それを 4 章の最小二乗法推定値とを比較したものを**表 2** に示す．なお最小二乗法による性能推定の誤差は，図 7，図 8 に示したようにクラスタ B では 10%以下，クラスタ C では 2%以下の小さな値である．

表 2 から明らかなように，ベンチマークを用いた a の推定値は過大に（性能が過小に）見積もられており，特にクラスタ C では 2 倍以上の大きな誤差が生じている．これはワークロードによって異なる，キャッシュサイズやメモリ性能が実行時間に与える影響を考慮できていないためであると考えられる．一方 b については，クラスタ B について比較的良い推定値が得られているが，クラスタ C では 2.5 倍程度という過大な（性能は過小な）見積もりとなっている．これはメモリ性能や I/O 性能など，ネットワークの基本的なバンド幅以外の要素が，通信パターンや通信量によって通信性能に与える影響を勘案できていないことを意味している．

以上のように今回の実験では，クラスタ B とクラスタ C の性能が過小に見積もられる結果となり，たとえば 8 ノードでの実行時間推定値はクラスタ B で実測値の 1.7 倍，クラスタ C では 2.3 倍となった．この程度の誤差は，たとえばスケジューリングの際の初期推定値としては許容できなくもないが，タスクの挙動を大きく見誤る可能性が高く，より精密な推定法を検討する必要性が明らかになった．

6. ま と め

本論文では，タスク並列スクリプト言語 MegaScript のタスク動作モデルの精度を向上するために，メタプログラム，メタモデル，およびモデルの解釈について拡張・改良を行った．MPI の各集合通信関数に対応したモデル式を立て，正確な通信コストを求めることができるようになった．また，ネットワーク構造の影響を考慮したモデルを作成した．そして，メタモデルからコストを解釈する方法として，最小二乗法による計算・通信コストの実行時間への変換法を提案した．検証の結果，非常に精度の高いモデルを生成できた．

しかし，精度の高いモデルを生成するためには，非常に多くのデータが必要である問題がある．これに対し，特定実行環境下におけるメタモデルと実行環境情報を元に，他の実行環境下でのスケーリング係数の推定を行った．メタモデルの計算コストと通信コストを計算・通信性能によって補正し，スケーリング係数の推定を行ったが，この方法では十分な精度が得られないことと，今後の課題としてより効果的な推定法が必要であることが明らかになった．

謝辞 本研究の一部は、科学技術振興機構・戦略的想像研究推進事業「低電力化とモデリング技術によるメガスケールコンピューティング」，および文部科学省科学研究費補助金 (特定研究，研究課題番号 18049039，「高性能計算の高精度モデル化技術」) による．

参 考 文 献

1) 大塚保紀，深野佑公，西里一史，大野和彦，中島浩: タスク並列スクリプト言語 MegaScript の構想，先進的計算基盤システムシンポジウム *SACSIS 2003*, pp.73–76 (2003).
2) 湯山紘史，津邑公暁，中島浩: タスク並列スクリプト言語 MegaScript における高精度実行モデルの構築．情処論 ACS, 46-SIG12(ACS11),pp.181-193(2005).