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

2004－HPC－99　(2)
2004／7／30

タスク並列スクリプト言語MegaScript向けランタイムシステム

西 里 一 史† 大 野 和 彦†† 中 島 浩†

我々は，メガスケールコンピューティング向けの言語として「タスク並列スクリプト言語 MegaScript」を提案している．MegaScript では，SPMD のような既存の枠組みで並列化されたプログラムを「タスク」として扱い，複数のタスクを制御することによってタスク並列実行を行う．また，各タスクの標準入出力を論理的な通信路である「ストリーム」で接続することによってタスク間通信を行う．

本論文では，MegaScript のランタイムシステムを紹介する．MegaScript ランタイムは，ユーザに対して基本的なタスク並列実行機能をランタイム API として提供する．ランタイム API が提供する機能には，タスク・ストリーム定義，ストリーム接続，タスク・ストリーム生成許可，タスクスケジューリングがある．

The Runtime System for The MegaScript Task Parallel Script Language

Hitoshi Nishizato,† Kazuhiko Ohno†† and Hiroshi Nakashima†

We propose a task-parallel script language named MegaScript for megascale computing. A MegaScript program has a tow-tier parallelism; the lower is an ordinary parallelism such as SPMD and the upper is task-level parallelism in which thousands of lower level parallel tasks are involved. MegaScript manages the communication among tasks via logical stream.

In this paper, we present a runtime system for MegaScript. The runtime system provides functions such as task/stream definition, stream connection, management of task/stream generation and task scheduling for task-parallel execution.

1. はじめに

近年，地球シミュレータ[1] のような数十 TFLOPS もの計算能力を持つ数千台規模の並列計算機が登場している．しかし，ゲノム情報／生命工学などのライフサイエンス，環境・気象シミュレーションなどの複雑な物理系が絡み合う系，大規模な都市における地震や山火事などの災害シミュレーションなどでは 1PFLOPS 以上の計算能力が期待されている．ここで，PFLOPS 以上の性能を得るためには，100 万台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となる．

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・運用するためには膨大なハードウェアを設置する巨大な施設および膨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ベースとした「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遂行している．メガスケール環境はヘテロな計算機やネットワークの集合として構築されるため，それらのリソースを効率よく利用するためのプログラミング環境およびその実行環境が必要となる．そこで，我々はメガスケールコンピューティング向けの言語として「タスク並列スクリプト言語 MegaScript」を提案している[2]．

メガスケールの並列性を持つプログラムをゼロから記述するのは非現実である．そこで MegaScript では，既存の枠組みで並列化された比較的小規模 (10^3 スケール) の並列プログラムを実行単位 (タスク) とし，それらをパラメータサーベイや最適値探索のために並列実行する「多重並列モデル」を採用している．

また，前述の通りメガスケール環境は Grid と同様にノードやネットワークの性能が非均質である．よって，メガスケールの並列性を持つアプリケーションをメガスケール環境上で実行するためには，タスクの特性 (計算量・通信量等) と計算資源の状態を考慮して，タスクの配置や生成タイミングを制御するタスク

† 豊橋技術科学大学
Toyohashi University of Technology
†† 三重大学
Mie University

スケジューラが必要となる．しかし，タスクの計算量や通信量などの情報をタスクのソースプログラムの解析だけから取得するのは一般に困難である．一方で，MegaScript ユーザはタスクの挙動について何らかの知識を持っていることが期待できるが，その知識レベルは様々である．そこで，MegaScript ではユーザが持っているタスクに関する知識をスケジューラに提供するための枠組みとして「メタプログラム」を用意する．このメタプログラムは，ユーザの知識レベルに応じて大まかにあるいは詳細に記述可能なものとなっている．

このメタプログラムを解析して得られる情報は，コンパイラによるタスクの解析情報と統合され，タスクの静的な性能モデルに変換される．この静的モデルに，実行時のパラメータや実行後に得られる性能情報を加えて動的な性能モデルを生成する．MegaScript スケジューラは，この動的モデルに基づいたタスクの挙動予測の結果をタスクスケジューリングに利用する．

本論文では，MegaScript の動作基盤であるランタイムシステムの紹介を行う．以下，2 章では MegaScript の概要について述べ，3 章で MegaScript ランタイムが提供する API について述べた後，4 章でランタイム API の機能について述べる．最後に，5 章でまとめを行い，今後の課題について検討する．

2. MegaScript 概要

タスク並列プログラミング言語は記述容易性とともに，複雑な処理を記述できる能力が必要である．さらに，一般プログラマへの親和性および言語やライブラリの拡張性などを考慮し，オブジェクト指向スクリプト言語 Ruby[4] を拡張する形で MegaScript を設計・実装する．

2.1 タ ス ク

MegaScript の並列実行単位を「タスク」と呼ぶ．タスクは独立性の高い処理モジュールであり，逐次プログラムであっても並列プログラムであっても構わない☆．また，MegaScript はタスク内部の処理には一切関与しない．

タスクの定義は，MegaScript プログラム内で Task クラスを継承し，initialize メソッドと behavior メソッドを定義する形で行う．initialize メソッドにはタスクを実行するために必要な情報 (実行プログラム名・引数等) を記述し，behavior メソッドにはメタプログラムを記述する．図 1 にタスク定義の例を示す．なお，MegaScript で定義されるタスクのスケジューリングは，MegaScript スケジューラが全て自動的に行うことを原則としている．

```
class Pi < Task
  def initialize(*arg)
    @exefile = "./pi"
    @args = arg
    super
  end
  def behavior
    FOR 0..@args[0]
      compute(2)
      IF 0.5
        compute(1)
      END
    END
  end
end
```

図 1　タスク定義例
Fig. 1　Example of task definition

2.2 タスク間通信

複数のタスクが協調動作をして一つの問題を解くためには，それらのタスク間でデータのやりとりを行う手段を用意する必要がある．

MegaScript では，タスクは独立した実行プログラムであることを仮定しているので，そのプログラムを実行したプロセスに対し外部から観測できるデータ送受信路はファイル入出力しかない．さらに，ライブラリの差し換えなども行わないとすれば，プロセスを起動する MegaScript ランタイムから見えるデータ送受信路は，プロセスの標準入出力だけである．そこで MegaScript では，各タスクの標準入出力を接続する方法でタスク間通信を実現する．

ここで，我々が既に開発した (Perl)+[3] にならって，論理的な通信路を表す「ストリーム」という概念を導入する．ストリームの入出力端にはそれぞれ複数のタスクを接続できるものとする．MegaScript のタスク間通信は，このストリームを利用したストリーム通信によって行う．なお，アーキテクチャの違いなどを意識せずにタスク間で簡単に流せるのはテキストデータであるので，基本的に MegaScript のストリーム通信はテキストベースとする☆☆．そして，そのストリーム

☆ 現在，タスクは他言語で作成された逐次の実行プログラムを想定している．

☆☆ 将来的にバイナリデータも流せるようにする予定である．

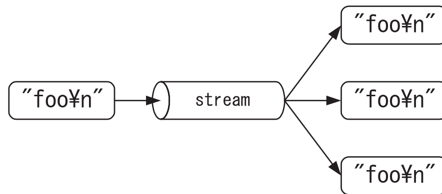

図 2　マルチキャスト
Fig. 2　Multicast via stream

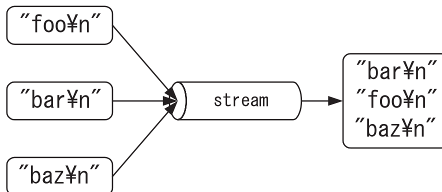

図 3　マージ
Fig. 3　Merge in stream

通信は基本的に行単位でバッファリングを行うものとする．

なお，ストリームの出力端に複数のタスクが接続された場合，そのストリームを流れるメッセージは，図 2 のように出力端に接続されているタスク全てにマルチキャストされる．

また，ストリームの入力端に複数のタスクが接続された場合，それらのタスクからそのストリームに入力されるメッセージは，図 3 のように一連のメッセージ列としてマージされる．ここで，図 3 では`"bar\n"`，`"foo\n"`，`"baz\n"`の順にマージされているが，実際は非同期にマージされるのでマージ結果は非決定的なものとなる．

2.3　メタプログラム

ユーザがタスクに関する情報を記述する方法の一つに，タスクの計算量や通信量，必要とする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タスクの概要を知ることができるだけで，通信頻度や入出力のタイミング，パラメータや実行時引数による振る舞いの変化を知ることができない．さらに，この方法はユーザ自信がプログラムを解析しコストを見積もる必要があるため，エンドユーザにとっては面倒で骨の折れる作業となる．

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」を採用している．メタプログラムは，記述方式としてプログラムを用いているため極めて高い記述性があり，柔軟で詳細な記述が可能である．また，メタプログラムでは

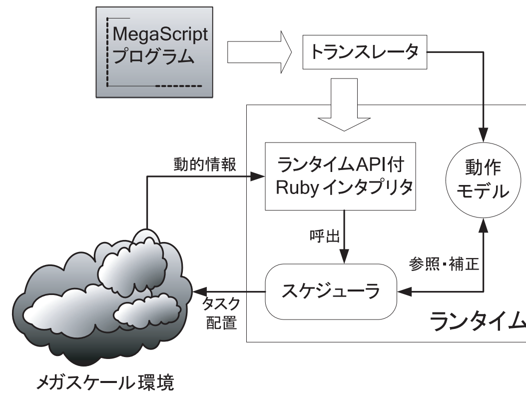

図 4　処理系の概要図
Fig. 4　Structure of execution system

抽象化表現を用いた記述が可能であるため，ユーザのタスクに関する知識レベルに応じた記述が行える．抽象化表現を用いることでタスクの大ざっぱな構造を簡略化した記述も可能である．ここで，抽象化する項目としては以下のものがある．

- 計算処理
- 標準入出力
- 制御構文
- メッセージ通信

なお，図 1 の behavior メソッド内のメタプログラムは，計算処理と制御構文の抽象化表現を用いて定義されている．

2.4　MegaScript 処理系

MegaScript 処理系の概要を図 4 に示す．処理系は主に，トランスレータ，ランタイム，スケジューラから構成される．

トランスレータは MegaScript のフロントエンドであり，ユーザが記述したメタプログラムの実行解釈を行い，タスクの動作モデル (メタモデル) を生成する．

MegaScript の実行基盤であるランタイムは，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．ランタイムは，ランタイム API を用いて記述された MegaScript プログラムを解釈し，分散環境上でのタスク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やタスクの実行時間といった動的な情報の収集も行う．

スケジューラは，ランタイム内部で動作しタスクの配置戦略や生成タイミングを決定する．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

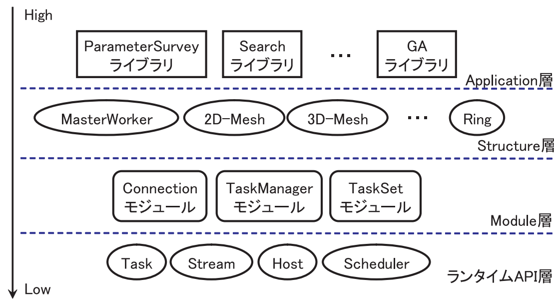

図 5 ライブラリモジュールの階層構造
Fig. 5 Hierarchical library module

配置を決定する．いくつかのタスクの実行が終了すると，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．スケジューラは精緻化を行ったメタモデルをもとに以後のスケジューリングを動的に補正する．

2.5 プログラミング

MegaScript ではランタイム API を利用した，低水準ではあるが非常に柔軟で強力な並列処理の記述が可能である．しかし，科学技術計算などを目的とするエンドユーザは並列処理の知識に乏しく，ランタイム API を使いこなすことは困難であると考えられる．そこで，MegaScript プログラミングをユーザフレンドリーなものにするために，図 5 のような階層構造を持ったライブラリモジュールを開発中である．この階層構造を用意することによって，並列処理に詳しいコアユーザから問題を解くことのみに興味があるエンドユーザまで幅広く利用可能となる．

ここで，Application 層でのプログラミングは図 1 のタスク定義に，

```
ParameterSuvey(Pi, 50000)
```

という一行を加えるだけで，ParameterSurvey ライブラリを利用した π のモンテカルロシミュレーションを行うことができる．このように Application 層でのプログラミングは，解きたい問題の種類が分かれば簡単に行うことができる．一方，ランタイム API 層でのプログラミングは，タスクとストリームの接続を一から記述しなくてはならない．次章では，ランタイム API について述べる．

3. ランタイム API

MegaScript ランタイムは図 5 に示したように，基本的なタスク並列実行機能を 4 つの Ruby API クラスとして提供している．これら 4 つのクラスにより，タスクやストリームの定義・接続・生成許可・タスクスケジューリングなどの機能を実現している．

以下，各クラスについて説明する．

3.1 Task クラス

Task クラスは，MegaScript におけるタスクを表現するクラスである．ランタイムは，Task クラス自身または Task クラスを継承して定義されたクラスのインスタンスを論理的なタスクとして取り扱う．生成許可は，`Task.create` メソッドをコールすることによって与えることができる．

3.2 Stream クラス

Stream クラスは，MegaScript におけるストリームを表現するクラスである．ランタイムは，Stream クラス自身または Stream クラスを継承して定義されたクラスのインスタンスを論理的なストリームとして取り扱う．生成許可は，`Stream.create` メソッドをコールすることによって与えることができる．さらに，Stream クラスはタスクとストリームを接続する `Stream.connect` メソッドを持つ．

3.3 Host クラス

Host クラスは，メガスケール環境で利用できるホストを表現するクラスである☆．物理生成直前のタスクやストリームを保持する生成待ちキューを持っており，主にスケジューラからの利用を想定している．このクラスはユーザによるインスタンス生成を許可しておらず，`Host#get_all` クラスメソッドによってのみ現在利用できるホストに対応する Host インスタンスの配列を取得することができる．

3.4 Scheduler クラス

Scheduler クラスは，タスクスケジューリングを実現するクラスであり，生成許可を与えられたタスクやストリームを保持するスケジューリング待ちキューを持っている．また，Scheduler クラスは `Scheduler.schedule` メソッドを持っており，これがコールされると予め与えられたスケジューリング戦略☆☆に従って，スケジューリング待ちキューからタスクやストリームインスタンスを取り出し配置するノードを決定する．配置するノードが決定したら，それらのインスタンスはそのノードに対応する Host インスタンスの生成待ちキューに挿入され，実際に物理生成される．ここで，ストリームを配置するノードとはストリームを流れるデータのシリアライゼーションを行うためのノードである．

☆ 現在はノード番号やホスト名といった情報しか持たせていないが，今後は実行時情報を組み込む予定である．
☆☆ デフォルトでは，均等配分を行うスケジューラが設定されている．

4. ランタイム API の機能

本章では，まずランタイム API を用いたプログラミングを紹介し，次にランタイム API がサポートするいくつかの機能について紹介していく．

4.1 ランタイム API を用いたプログラミング

ランタイム API を用いた基本的なプログラミングの流れは次のようになる．

(1) タスク・ストリームの定義
(2) タスクとストリームの接続処理
(3) タスク・ストリームへの生成許可
(4) タスクスケジューリング

例えば，一対一のタスク接続を行うプログラムは以下のように記述できる．

```
pro = Task.new("producer", 10)
con = Task.new("consumer")
stream = Stream.new
stream.connect(pro, IN)
stream.connect(con, OUT)
pro.create
con.create
stream.create
Scheduler.new.schedule
```

ここで，Task.new の第一引数はタスクに対応する実行プログラム名，第二引数はタスクに渡す実行時引数を設定する．また，IN 定数はストリームの入力端，OUT はストリームの出力端を表す．

4.2 タスクの実行時引数設定

タスクの実行時引数は，文字列，整数，浮動小数点数を並べて記述することで設定できる．また，タスクを実行する際のワーキングディレクトリは Task.working_dir インスタンス変数で設定できる．MegaScript は，実行時引数としてワイルドカード文字列を設定することができる．ワイルドカード文字列が設定されている場合，ランタイムはタスクを実行するワーキングディレクトリでワイルドカード展開を行った後，タスクに引数を引き渡す．なお，ワイルドカード展開して欲しくない場合はシングルクォートで括った文字列を設定すればよい．

また，同一引数の複数タスクの定義を行うために，Task.new_array メソッドが用意されている．しかし，引数がわずかにでも違う場合はタスク定義を行うために，ループを回さなければならない．これを解決するために，Task.new_array メソッドでの引数設定のための引数に，定義するタスク数と同じサイズの配列や範囲オブジェクトを設定可能にしている．設定例を以下に示す．ここで，Task.new_array の第一引数には定義するタスク数を設定する．

```
arg = "*.rb"
arg2 = ["'foo'", "'bar'", "'baz'"]
arg3 = 1..3
task_ary = Task.new_array(3,
            "sample", arg, arg2, arg3)
```

4.3 MegaScript 中からのストリームアクセス

タスクの動作確認や，先行タスクによって今後生成するタスクの数や種類が変わるといったプログラミングを行うためには MegaScript プログラム中でストリームにアクセスできる必要がある．これを行うために，ランタイムでは SELF 定数と Stream.open メソッドを用意している．

SELF 定数は MegaScript 自身を表す定数であり，プログラム中でアクセスしたいストリームに対してタスクの代わりに接続処理を行うことができる．その後，タスクスケジューリングを行った後に，そのストリームの open メソッドをコールすることによって，ストリームにアクセスできる IO オブジェクトを取得することができる．以下に，使用例を示す．

```
 …
stream.connect(SELF, IN)
 …
pout = stream.open
for i in 1..100
   pout.print i, "\n"
end
pout.close
```

4.4 タグ付きストリーム

パラメータサーベイ等のアプリケーションでは，複数のタスクの計算結果を集めるタスクが必要となる．しかし，MegaScript ではタスクの入力口は標準入力一本しかないため，多対一のストリーム接続を行う必要がある．この時，ストリームの入力端に接続された各計算タスクが連続した複数行の計算結果を出力した場合，ストリームにより各計算タスクの出力が非同期にマージされてしまい不都合が生じることになる．

そこで，MegaScript ではストリーム定義時または定義後に，ストリームの入力端に接続された各タスクの出力にタグを付加できる機能を用意している．タグとして指定できるものには以下のものがある．タグは 2 つ同時に設定することができ，順番も設定可能である．また，タグとタスクの出力するメッセージを分離するセパレータとして任意の文字列を設定することができる．なお，セパレータのデフォルト値は “:” で

ある．

- SENDER_ID
  入力端に接続されたタスクに割り振られたユニークなタスク ID
- SENDER_INDEX
  入力端に接続されたタスクに割り振られたタスク ID を 0 から始まる連続した番号で置き換えたもの

このタグ付け機能を用いてタグを付け，結果を集計するタスクとストリームの間にタグを処理するフィルタを挿入することによって基本的なパターンをサポートすることが可能になる．以下に，タグ利用例を示す．

- Stream.new(SENDER_ID)
  4:30284157
  2:17354268
  ...
- Stream.new([SENDER_ID,SENDER_INDEX], "#")
  4#1#30284157
  2#0#17354268
  ...

4.5 ストリームデータのまとめ送り

ランタイムはデフォルトで，入力端に接続されているタスクが出力するデーター行ごとに物理通信を行う．しかし，これまでの実験からある程度の大きさまでのメッセージなら，通信オーバーヘッドはメッセージサイズにかかわらず通信回数のみに依存することが分かっている．

そこで，ランタイムは Stream.packs インスタンス変数にまとめ送りする行数を設定することで，ストリームデータのまとめ送りをサポートする．なお，Stream.packs インスタンス変数のデフォルト値は 1[line] である．

4.6 ストリームの動的接続

探索問題等を扱うアプリケーションは，既存のタスクの要求に応じてストリームに動的にタスクを追加できる必要がある．よって，ランタイムは動的なストリームをサポートする必要がある．ここで，動的なストリームとはストリーム生成時に，接続されるべきタスクが全ては接続されていないストリームであるとする．逆に，ストリーム生成時に，接続されるべきタスクが全て接続されているストリームを静的なストリームと定義する．ランタイム API における静的なストリームと動的なストリームの切り替えインターフェースとして，Stream.dynamic インスタンス変数を用意しており，デフォルト値は *false* で静的なストリームを表す．この変数に *true* を設定することにより動的なストリームに切り替えることができる．

次に，動的なストリームのセマンティクスについて説明する．まず，動的なストリームの出力端に新たなタスクを接続した場合，そのタスクが接続される前にストリームを流れたメッセージ全てをそのタスクは受信する．そのため，動的なストリームはあといくつのタスクが接続されるか分からないので基本的にストリームを流れるデータのバッファリングを行う．また，これ以上新しいタスクが接続されないという保証がない限り，バッファの解放や出力端に接続されているタスクの終了処理が行えない．そこで，Stream.connect_up メソッドを用いて，明示的にこれ以上新たなタスクの接続が行われないことをランタイムに通知するメカニズムを用意している．

5. ま と め

本論文では，タスク並列スクリプト言語 MegaScript の概要について述べた後，MegaScript ランタイムシステムについて主にランタイム API を中心に紹介した．ランタイム API は，MegaScript の中でタスク並列実行をサポートする最もコアな部分である．

今後は，ランタイムの性能評価を行いシステムの妥当性を検証していく必要がある．また，システム情報を Host クラスに組み込み，その情報を MegaScript スケジューラから利用可能にする予定である．

現在 MegaScript ランタイムは，逐次の実行プログラムをタスクとして想定しており，かつ単一の PC クラスタを実装対象としている．そこで，タスクとして並列プログラムも扱えるようにし，かつ広域分散環境上で動作する MegaScript ランタイムを実現することも今後の課題となる．

謝辞 本研究の一部は，科学技術振興機構・戦略的創造研究推進事業「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」による．

参 考 文 献

1) Habata, S., Yokokawa, M. and Kitawaki, S., “The Earth Simulator System,” NEC Research & Development, Vol.44, No.1, pp.21-26, Jan. 2003.
2) 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 SACSIS2003, pp. 73-76 (2003).
3) 外崎由里子, 中田尚, 大野和彦, 中島浩: 並列スクリプト言語 (Perl)+の実装と設計, 並列処理シンポジウム JSPP’02, pp. 241-244 (2002).
4) まつもとゆきひろ, 石塚圭樹: オブジェクト指向スクリプト言語 Ruby, ASCII (1999).