卒業論文

題目

MegaScript処理系における
アダプタ機構の実装及び評価

指導教員

大野和彦　講師

平成24年度

三重大学 工学部 情報工学科
計算機アーキテクチャ研究室

松嶋佑弦(408849)

内容梗概

近年，遺伝子解析や環境シミュレーションなど膨大な計算処理を要する分野において，Pflops 以上の計算性能が求められている．Pflops を超える性能を得るには，100 万台規模のプロセッサを用いた並列処理が不可欠である．しかし，既存の並列プログラミングモデルで大規模な並列処理をプログラミングすることは困難である．そこで，本研究室では大規模な並列処理を容易に実現するためのタスク並列スクリプト言語 MegaScript の開発を行っている．

MegaScript では独立性の高い外部プログラムをタスクとして扱い，ストリームと呼ばれる論理的な通信路でタスクの標準入出力をつなぎ合わせることで並列処理を実現している．このような実行モデルにおいて，高性能化のためにはアプリケーションや計算環境に応じて言語処理系やタスク毎のプログラムを変更，最適化する必要がある．しかし，処理系の改造はユーザにとって非常に煩雑であり，言語自体の拡張性，再利用性の低下を招く．これらの問題を回避するためにアダプタ機構が提案されている．

アダプタ機構は拡張用コードを処理系・タスクから分離して記述できるようにするためのフレームワークであり，ユーザが記述した拡張コードをアダプタとして実行時に処理系へと組み込み，イベント発生時にコールバックすることで処理の挙動を拡張する．しかし，現在の MegaScript 処理系には実装されておらず，また計算環境に応じた改造を行うための機能が存在しない．

そこで本論文では MegaScript 処理系にアダプタ機構を実装し，機能拡張・追加を行った．新たに指定時間の経過，メモリや CPU のリソース変動をイベントとしてアダプタを登録可能にしたことで，定期的な計算資源の確認や，ホストの性能変動時に動的スケジューリングの呼び出しなどが可能となった．

本機構を用いてホストの性能を取得することで発生するオーバーヘッドを評価した結果，アダプタの実行間隔が実用的な範囲内でのオーバーヘッドは全体の処理に対して十分小さく，許容範囲内であることが確認された．

Abstract

Many research fields, such as gene analysis and environment simulation, use high computation power over Pflops. To obtain such power, parallel computing using 1 million processors is required. Therefore, we are developing a task-parallel script language MegaScript for large scale parallel processing.

MegaScript executes processes in parallel by connecting tasks which are external programs. Streams are logical channels to connect standard inputs/outputs of tasks. Using MegaScript, various kinds of parallel computation models can be implemented by explicit description of task networks. Although MegaScript runtime and task programs should be specialized to the target application and the runtime for high performances, it is undesirable for portability. Adapter scheme solves these problems.

Adapter scheme is a framework for extension and optimization without modifying the runtime nor the task programs. Using Adapter scheme, the user can describe the code in user scripts to modify the runtime. However, this scheme is not implemented in current MegaScript, and cannot customize for execution environments.

Therefore, we implemented Adapter scheme in MegaScript runtime, and extended/improved it. We implemented a new timer adapter, which can be used to check computing resources at fixed intervals. We also implemented adapters for the event of changing computing resources. Thus, we can implement dynamic scheduling easily.

The evaluation of the overhead to obtain computing resources using our implementation shows enough performance for practical use.

目 次

図 目 次

表 目 次

1　はじめに

1.1　背景

近年，遺伝子解析や環境シミュレーションなど複雑な物理計算を要する分野において，Pflops 以上の計算性能が求められている．Pflops を超える性能を得るには，100 万台規模のプロセッサを用いた並列処理が不可欠である．しかし，既存の並列プログラミングモデルで大規模な並列処理をプログラミングすることは困難である．そこで，本研究室では大規模な並列処理を容易に実現するためのタスク並列スクリプト言語 MegaScript[1] を開発している．MegaScript はその性質上，アプリケーションや計算環境に応じて言語処理系やタスク毎のプログラムを変更，最適化する必要がある．しかし，処理系の改造はユーザにとって非常に煩雑であり，また言語自体の可搬性，再利用性が低下してしまう．

1.2　目的

処理系の改造を容易にし，かつ拡張用のコードを処理系から分離して記述するためにアダプタ機構 [2] が提案されているが，現在の MegaScript 処理系には実装されておらず，また計算環境に応じた改造を行うための機能が充実していない．

そこで，本論文ではMegaScriptの機能拡張及び最適化を容易にするアダプタ機構を実装し，機能拡張・追加を行った．アダプタ機構は処理系の動作をイベントとし，そこに拡張用コードであるアダプタを割り込ませることで処理の挙動を拡張する．新しく指定時間の経過をイベントとしてアダプタを登録できるようにしたことで，定期的にホストの性能取得が可能となり，またホストの性能変動をイベントとしてアダプタを登録できるようにしたことで動的スケジューリングの実装が容易となった．

1.3 論文の構成

本論文の構成は以下のようになっている．まず2章，3章でMegaScriptとアダプタ機構について述べる．4章で関連研究との比較を行い，5章で今回設計・実装したアダプタ機構の改良箇所について述べる．6章では改良を行ったアダプタ機構の評価結果を示し，最後に7章でまとめを行い，今後の課題について検討する．

2　タスク並列スクリプト言語 MegaScript

2.1　言語の概要

MegaScript は，100 万プロセッサ規模の並列処理を目的とする Ruby[3] ベースのスクリプト言語である．図 2.1 のようなワークフローを記述することで並列処理を実現できる．

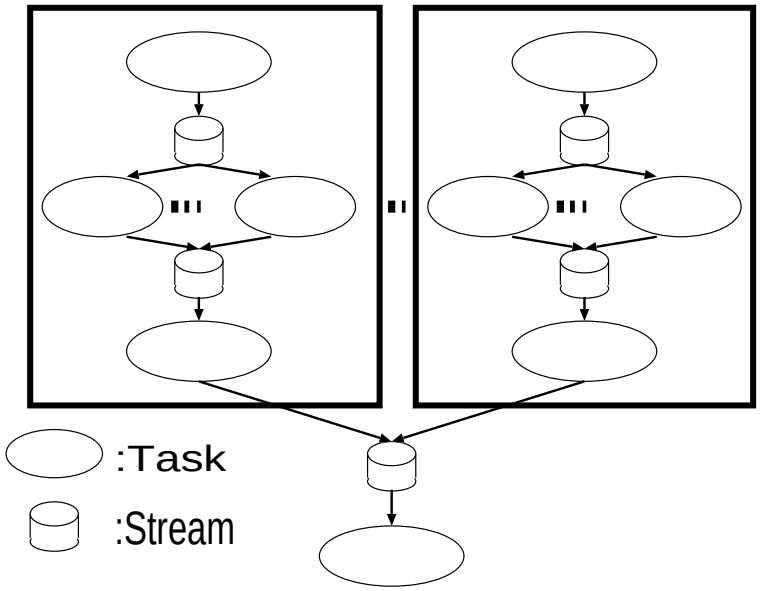

図 2.1: ワークフローの例

MegaScript では，独立性の高い外部プログラムをタスクとして利用し，ストリームと呼ばれる論理的な通信路でタスクの標準入出力をつなぎ合わせている．そして，依存関係のないタスクを複数の計算機で同時に実行することで並列処理を実現している．

このような実行モデルにおいて，高性能化のためには対象となるアプリケーションや計算環境の特性に応じて言語処理系やタスク毎のプログラムを変更，最適化する必要がある．一方で，記述の容易性やコードの再利用性を高めるために計算環境に依存せず，タスク間の独立性を確保することが望ましい．

2.2 タスク

MegaScript では処理の実行単位をタスクと定義している．タスクは独立した外部プログラムであり，ユーザは任意の言語でタスクプログラムを作成することができる．このため，既存のプログラムをタスクとして流用したり，処理内容に応じて記述言語を変えるといったことが自由に行える．また，MegaScript はタスク内部の処理に一切関与せず，タスク間の情報のやりとりは標準入出力を利用している．

2.3 ストリーム

ストリームは，タスクの標準出力の内容を他のタスクの標準入力に流し込むための通信路であり，MegaScript におけるタスク間通信を実現する．このストリームはテキスト通信に対応しており，改行を区切りとして 1 行を 1 つの不可分なメッセージとして扱う．

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することができ，一対多，多対多などの通信を簡潔に実現することができる．入力端に複数のタスクを接続した場合，メッセージは非決定的にマージされる．また，出力端に複数のタスクを接続した場合，メッセージはそれらのタスクにマルチキャストされる (図 2.2).

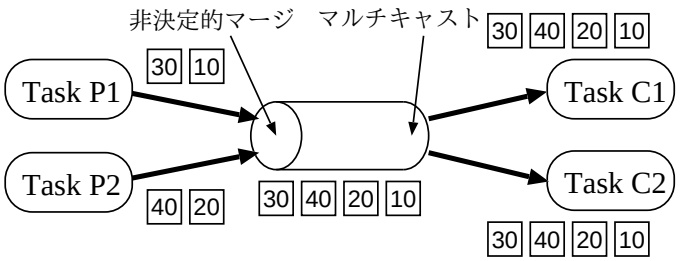

図 2.2: ストリームによる多対多通信モデル

2.4 API クラス

MegaScript では，タスクやストリームなど並列実行の基盤となる要素を組み込みの API クラスとして実現している．現在の MegaScript では以下の 4 クラスが提供される．

2.4.1 Task クラス

Task は MegaScript におけるタスクを表現・操作するためのクラスであり，1 オブジェクトが 1 つのタスクに対応する．オブジェクト生成時に引

数として利用するプログラムを指定する．

2.4.2 Stream クラス

Stream は MegaScript におけるストリームを表現・操作するためのクラスであり，1 オブジェクトが 1 本のストリームに対応する．通信を行うタスクは Stream の connect メソッドによりストリームへと接続する．引数には，接続するタスクオブジェクトとストリームの入力端 (IN) および出力端 (OUT) を接続先として指定する．

2.4.3 Host クラス

Host は抽出化された実行環境の各ホストを表現するためのクラスであり，1 オブジェクトが 1 つのホストに対応する．ホストオブジェクトは実行時に自ホストの情報を収集し，専用のメソッドを介して提供する．

2.4.4 Scheduler クラス

Scheduler はタスクスケジューリングを実現するためのベースクラスである．このクラスを継承してメソッドをオーバーライドすることで，任意のスケジューリング戦略を実現できる．

3　アダプタ機構

3.1　アダプタ機構の概要

アダプタ機構はアプリケーションや環境依存のコードをアダプタとして MegaScript 処理系へと組み込むための共通のフレームワークであり，実行毎に最適な処理の実現をサポートする．

ユーザは処理の最適化のためのコードをアダプタとしてユーザスクリプト内に記述し，これを実行時に処理系へと組み込むことで処理の挙動を拡張する．利用する計算環境に特化したシステムを構築するには，処理系のコード内から改造が必要な箇所を特定しコードを書き換え，コンパイルし直す必要がある．しかし，アダプタを利用することにより，処理系やタスクプログラムを変更することなく，実行環境に特化した機能の拡張や最適化のためのカスタマイズを簡潔に行える．図 3.3 にメッセージのフィルタリングを行うアダプタの使用例を示す．

ユーザは Adapter クラスを継承した独自クラスを定義し，callback メソッドをオーバーライドすることで拡張コードを作成する．各イベントに対応した変数に独自クラスを登録することでイベント発生時に拡張コードを実行できる．

```
class MyAdapter < Adapter
  def callback(msg)
    msg.gsub!(^.*:/,"")
    return msg
  end
end
task = Task.new("worker")
task.event_output = MyAdapter.new
```

図 3.3: アダプタの例

3.2 アダプタの記述をサポートする拡張機能

3.2.1 登録先オブジェクトの操作

タスクなどの処理系内部で管理されているオブジェクトをアダプタ側から操作するために，組み込みのメソッドとして partner が提供される．

partner はアダプタの登録先である API クラスのオブジェクトを指し示すポインタであり，アダプタ側からはアクセサとして利用することができる．partner を用いることで，登録先のオブジェクトに対する操作や各種情報の取得などを，処理系内部を隠蔽した状態でアダプタ側から行えるようになる．

3.2.2 アダプタチェーン

1 つのイベント発生に対し複数のアダプタを連続して呼び出すために，アダプタの配列オブジェクトを登録できる．

```
task.event_output = [Adapter1.new, Adapter2.new, ...]
```

イベントの発生時には，配列の先頭のアダプタからチェーン式にコールバックされ，引数には一つ前のアダプタの返り値が伝搬される．

出力イベントのメンバへアダプタの配列を登録した場合，アダプタ配列のコールバックは図 3.4 に示すように動作する．この機能を利用するこ

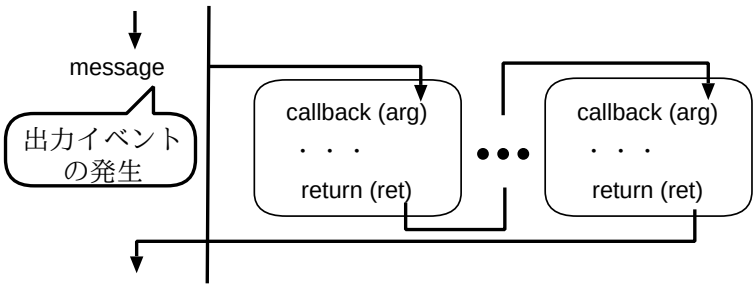

図 3.4: アダプタのチェーン式コールバック

とで，既存のアダプタを組み合わせて利用したり，複雑な機能を複数の単純なアダプタの集合として記述したりできる．

4　関連研究

大規模並列計算に対する既存アプローチとして，UNICORE[5]，オーガニックジョブコントローラ [6] などのワークフロー型システムが挙げられる．ワークフロー型システムはユーザのジョブを，タスク間の依存関係を表すワークフローとして与える．データの受け渡し時には，システム側でデータ変換処理を行うことでタスクの変更を最小限に抑えることができる．MegaScript ではアダプタを用いることで，タスク外部においてデータフローの不整合性や非効率性を解消できるだけでなく，ホストの動作拡張を行うこともできる．また，アダプタはユーザスクリプトに記述するだけで使用できるため，容易に MegaScript を拡張できる．

5　提案手法

5.1　設計

アダプタを登録する際には，partner にアダプタの登録先オブジェクトを登録する必要がある．また，今後アダプタ機構に機能が追加された場合，それらも登録を必要とする可能性がある．そこで，アダプタの操作を容易にするためアダプタの登録，追加，削除，入替用メソッドを各 API クラスに追加する．

広域分散環境において，計算環境のパフォーマンスを維持するためには各計算機の監視が必要となる．また，計算資源が変動した場合には適切な処置をしなければ，パフォーマンスの低下に繋がる．そこで，ホストに対するアダプタ機構を設計・実装する．定期的にホストの計算資源の監視をするための一定時間経過をイベントとした機能や，性能変動の際に処置を行うためのメモリリソースや CPU リソースの変動をイベントとした機能を新たに設計・実装する．これらのアダプタ機構を利用すれば，ホストの性能変動時に動的スケジューリングを実行するといった機能拡張を容易に組み込むことができる．また，アダプタを通して計算資源を監視するために，計算資源を計算する機能をホストに追加する．

5.2 実装

5.2.1 API クラスの拡張

アダプタから計算資源を確認するために，memused，cpuused メソッドを Host クラスに追加した．memused はホスト全体のメモリ使用率を，cpuused はホスト全体の CPU 使用率を計算するメソッドである．linux では top コマンドや free コマンドなど CPU 使用率やメモリ使用率を確認できるコマンドが存在するが，それらを短い間隔で実行しテキスト処理をしていては処理が重くなってしまう．従って，/proc ディレクトリ内の仮想ファイルを読み込むことでそれらの機能を実装した．

アダプタの登録や削除等を行うために表 5.1 のメソッドを Task クラス，Host クラスに追加した．これらのメソッドは各イベント毎に存在する．これらのメソッドを使用することで，ユーザはアダプタの拡張機能を利用するために必要な処理を意識せずにアダプタの操作が可能になる．registerAdapter は対応するイベントに登録されているアダプタをすべて削除し，新しく指定したアダプタを登録する．addAdapter は登録済みのアダプタを削除せずに指定したアダプタを追加する．対応するイベントにアダプタが登録されていた場合は最後尾に追加される．deleteAdapter は引数に配列の添え字を取り対応するアダプタを削除する．replaceAdapter

は引数に添え字とアダプタを取り対応するアダプタを新たに指定したアダプタと入れ替える．

表 5.1: アダプタ操作用メソッド

	メソッド名	引数
登録	register*Name*[1] Adapter	アダプタ (配列)
追加	add*Name*[1] Adapter	アダプタ (配列)
削除	delete*Name*[1] Adapter	整数値 (添え字)
入替	replace*Name*[1] Adapter	整数値 (添え字), アダプタ (配列)

5.2.2 アダプタ機構の実装

表 5.2 の通り，タスクやホストでの各イベントに対するアダプタ機構を実装した．このうちタスクに関するアダプタ機構は従来手法と同等の機能を有する．

`TimerAdapter` は一定時間毎に登録されたホストで実行されるアダプタであり，アダプタ登録時，またはクラス定義時にインスタンス変数 `interval` にミリ秒単位で実行間隔を指定する必要がある．`interval` をホストに登録させると複数の実行間隔で `TimerAdapter` の呼び出しを行いたい場合に，`interval` と `TimerAdapter` との関連付けが複雑になってしまう．そこで，アダプタ自身に実行間隔を保持させることで複数の `TimerAdapter` を登録した際にも各実行間隔でそれぞれの `TimerAdapter` を実行できる

[1] *Name* には表 4.2 のアダプタ名が入る

ように設計した．これにより，TimerAdapter と interval は 1 対 1 の関係が保証され，関連付けは容易になった．また，アダプタ登録とともに実行間隔を指定できるようになったことでアダプタ登録の単純化にも繋がっている．図 5.5 に 10 秒ごとにメモリの使用率を表示するアダプタの例を示す．

```
class MyTimer < Adapter
  def initialize(val)
    @interval = val
  end

  def callback()
    puts self.partner.memused
  end
end

host.addTimerAdapter(MyTimer.new(10000))
```

図 5.5: TimerAdapter の使用例

性能変動を感知するために CheckMemAdapter と CheckCPUAdapter を提供する．これらは内部で TimerAdapter を利用する．CheckMemAdapter は一定時間毎にホスト全体のメモリ使用率をチェックし，指定された上限値，下限値に対して比較を行う．この結果を MegaScript 処理系で受け取り，MemoryAdapter を呼び出す際の引数を決定する．CheckCPUAdapter は CheckMemAdapter と同様に一定時間毎にホスト全体の CPU 使用率を

チェックする．この結果からCPUAdapterの引数を決定する．どちらもオブジェクト生成時の引数に上限値，下限値，実行間隔をとり上限値，下限値は相対値 (%) で指定する．

MemoryAdapterはメモリリソースが変動した際に実行されるアダプタであり，CheckMemAdapter の結果から登録されたアダプタを実行する．オブジェクト生成時に配列型で二つのアダプタを登録することで一つ目が上限値を超えたとき，二つ目が下限値を下回ったときに呼び出すアダプタとして登録できる．同様に，CPU リソースが変動した際に実行されるCPUAdapterが存在する．これら二つのアダプタはそれぞれCheckMemAdapter，CheckCPUAdapterの結果を利用しているため，MemoryAdapter，CPUAdapterを登録時にTimerAdapterにそれぞれCheckMemAdapter，CheckCPUAdapterが登録されていなかった場合，自動的にそれらを TimerAdapter に登録する．

表 5.2: アダプタ一覧表

対応クラス	イベント	アダプタ名
Task	入力	InputAdapter
	出力	OutputAdapter
	終了	TerminateAdapter
Host	一定時間経過	TimerAdapter
	メモリリソース変動	MemoryAdapter
	CPU リソース変動	CPUAdapter

6　性能評価

6.1　評価環境

評価用の実行環境には AMD Athlon II X2 B24 Processor 800MHz，メモリ 2GB を搭載した計算機を 16 台使用した．使用したワークフローは乱数を 1 つ生成するタスクと入力の平均値を出力するタスクを 1000 対 1 で繋いだものである．

6.2　評価内容

`TimerAdapter` を用いてホストの性能取得を行うことで，MegaScript の性能が著しく低下してしまわないかを確認するため，アプリケーションの実行に要する時間を測定し，オリジナルとの比較を行った．アダプタの実行間隔によりオーバーヘッドは変化すると思われるため，`interval` を 2000，1000，100 と変化させてそれぞれの実行時間を計測した．表 6.3 に計測結果とオリジナルとの時間比を示す．

表 6.3: 処理時間比

interval(ms)	none	2000	1000	100
実行時間 (s)	5.812	5.971	6.089	6.190
比率	1.000	1.027	1.047	1.065

6.3 考察

表 6.3 からわかるように，アダプタ使用時は実行時間が増加した．しかし，オーバーヘッドは 5%程度に抑えられている．MegaScript のアプリケーションは長時間の計算処理を行うことが予想されるため，このオーバーヘッドは無視できる範囲であると言える．

7　おわりに

7.1　まとめ

本研究では MegaScript 処理系にアダプタ機構を実装し、機能拡張・改良を行った．指定時間の経過や計算資源の変動をイベントとしたアダプタを登録可能にしたことで定期的なホストの性能監視を可能とし，動的スケジューリングの実装を容易にすることができた．

また評価を行った結果，今回実装したアダプタ機構によるオーバーヘッドは実用上無視できる範囲であることが確認できた．

7.2　今後の課題

今回実装したアダプタ機構はホスト全体の性能変動をイベントとしている．今後は MegaScript が使用している計算資源の変動をイベントとしてアダプタを登録できるようにすることで，複数のユーザが同じホストを利用している際に別のアダプタを実行可能とすることが挙げられる．

謝辞

本研究を行うに当たり，終始御指導を賜った大野和彦講師，並びに多くの助言を頂きました近藤利夫教授，佐々木敬泰助教に深謝致します．最後に，日頃から様々な点でお世話になった研究室の皆様，田中事務員に厚くお礼申し上げます．

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 SACSIS2003, pp. 73-76 (2003).

[2] 阪口祐輔, 大野和彦, 佐々木敬泰, 近藤利夫, 中島浩: タスク並列スクリプト言語処理系におけるユーザーレベル機能拡張機構, 情報処理学会論文誌, Vol. 47, No. SIG 12(ACS 15), pp. 296-307 (2006)

[3] 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).

[4] Smith, B.C.: Reflection and Semantics in Lisp, Proc. of ACM Symp. on Principles of Programming Languages, pp. 23-35 (1984).

[5] Romberg, M.: The UNICORE Architecture - Seamless Access to Distributed Resources, Proceedings of 8th IEEE International Symposium on High Performance Distributed Computing(HPDC8), pp. 287-293 (1999).

[6] 上田晴康, 吉田武俊, 安里彰: ジョブ投入と待ち合わせの出来るジョブ制御スクリプト: オーガニックジョブコントローラの試作, 情報処理学会研究報告 2003-OS-94, pp. 99-106 (2003).

A　アダプタクラス・リファレンス

A.1　Adapter クラス

Class	Adapter
Inherited Class	Object
Class Methods	new
Instance Methods	initialize, callback
Instance Variables	@partner

A.2　対応イベントのメンバー一覧

- Task
  - @event_input (Task#event_input)

    タスクのメッセージ入力
  - @event_output (Task#event_output)

    タスクのメッセージ出力
  - @event_terminate (Task#event_terminate)

    タスクのプロセス終了
- Host
  - @event_timer (Host#event_timer)

指定時間の経過毎

- @event_cpu（Host#event_cpu）

  CPU リソースの変動

- @event_memory（Host#event_memory）

  メモリリソースの変動