卒業論文

題目

# ファイルアクセスを伴う実用プログラムを用いたタスク並列スクリプト言語MegaScriptの性能評価

指導教員

大野 和彦 講師

2011年

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

後藤 友成 (407814)

# 内容梗概

近年，環境・気象シミュレーションや科学技術計算などの分野で Pflops レベルの計算能力が必要とされてきている．しかし，メガスケール規模の並列性を持つプログラムを一から記述することは非常に困難である．そこで，そのような大規模並列プログラミング環境を容易に実現するため，タスク並列プログラミング言語 MegaScript が開発されている．

今までの MegaScript は標準入出力間をストリームと呼ばれる通信路で接続しており，ファイル入出力に対応していなかった．現在の MegaScript にはより多くのアプリケーションに対応できるようファイル入出力を扱う機能が提案・実装されている．しかし，現段階ではファイルアクセスの機能はシミュレーションによる動作検証しか行うことができない．

そこで，新たに拡張されたファイルアクセスの機能を実用プログラムを用いて評価を行うにあたり，本研究では，Network File System(NFS) を使用してファイル入出力の評価を行った．評価には，CG 作成，遺伝子解析，天文画像の合成の 3 種のアプリケーションを用い，データ転送量によってどのような差が出るか調べた．

その結果，CG 作成，遺伝子解析のようなデータ転送量が比較的小さなアプリケーションでは NFS を用いた場合でも通信時間の割合が少なく通信による実行時間の低下はあまり見られなかった．一方，天文画像の合成のようなデータ転送量が大きなアプリケーションでは NFS を用いた場合，通信時間の増加することがわかった．そのためデータ転送量が多い場合，MegaScript のファイルシステムを使用することにより実行時間の短縮を見込めることがわかった．

# Abstract

Recently processing ability of computer over Pflops is being required in some fields such as environmental simulation and scientific technical calcuration.

However, programming with the mega-scale parallelism up from nothing is very difficult. Therefore, we are developing “task parallel script language MegaScript” in order to realization large-scale parallel programing environment like that.

Until now, MegaScript could only connect standard I/O called ”Stream”, therefore MegaScript hasn’t could file I/O. However now, MegaScript can File I/O to correspond many applications. But only file access function can use Operation verification to simulation.

So before, evluation of file access using practical programs,this time file I/O evaluated using Network File System(NFS).

We use following programs: making computer graphics, gene analysis and synthesis of astronomy images for a evaluation.

As a result, programs of few data transfer like computer graphics and gene analysis wasn’t too take time. But programs of much data transfer like synthesis of astronomy is was too take time.

Therefore,in case we execute much data transfer application,if we use new file system in MegaScript, we can expect reduction of communication time.

# 目 次

# 図 目 次

# 表 目 次

# 1　はじめに

遺伝子解析のような生命工学や，環境・気象などのシミュレーションに代表される科学技術計算において，1Pflops 以上の計算能力が必要とされてきている．しかし，スーパーコンピュータの延長でこれを実現するのは，莫大なコストと巨大な施設が必要であるため極めて困難である．そのため，100 万台規模のプロセッサを用いたメガスケールコンピューティングの研究が行われている．メガスケールコンピューティング環境では，性能の異なる計算機やネットワークの集合としてシステムを構築するため，それらのリソースを効率よく利用する実行システムが必要である．また，メガスケールの並列性をもつプログラムを一から記述するには，高度な技術や多大な人的・時間的コストが必要となる．そこで，メガスケールコンピューティングを容易に実現する言語として，タスク並列スクリプト言語 MegaScript[1, 2] が開発されている．

## 1.1　研究目的

MegaScript は既存プログラムを容易に並列化する言語であるが，今までのタスク間通信は標準入出力のみに対応しており，ファイルを用いたプログラムを扱えないなどの制約があった．しかし現在の MegaScript に

はより多くのアプリケーションに対応できるようファイル入出力を扱う機能が提案・実装されている．しかし，現段階ではファイルアクセスの機能はシミュレーションによる動作検証しか行うことができない．そこで，新たに拡張されたファイルアクセスの機能を実用プログラムを用いて評価を行うにあたり，本研究では３種の実用的なプログラムを用いてNetwork File System(NFS) の評価を行い，MegaScript のファイルアクセス機能にどれ程度有用性があるか評価を行う．

### 1.2　本論の構成

まず，第２章では MegaScript の概要について述べ，第３章で評価の詳細を説明する．そして，第４章で評価を行い，第５章で考察し，最後に第６章でまとめる．

## 2　MegaScript の概要

本研究で評価の対象とするタスク並列スクリプト言語 MegaScript について本章で説明する．

## 2.1 MegaScript の概要

大規模なプログラムをユーザーが一から記述することは非常に困難である．そのため，逐次プログラムを並列化した小規模な並列プログラムを組み合わせることにより大規模並列性を引き出す「2 階層並列モデル」が考えられている．MegaScript は大規模プログラミングを実現可能とし，広域分散した複数の並列計算機を含む環境下で，それらの資源を効率よく利用し動作することを目指している．MegaScript は，逐次プログラムや小規模な並列プログラムのような独立性の高い外部プログラムを「タスク」として扱い，複数のタスクを制御して大規模の並列実行を行うタスク並列スクリプト言語である．また，これらのタスクの標準入出力間を「ストリーム」と呼ばれる通信路で接続している．さらに，ユーザー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 を拡張する形で設計されている．

## 2.2 タスク

タスクは MegaScript の並列実行単位である．タスクの実体は C 言語や Java，Ruby 等の任意の言語で作成された独立性の高い外部プログラムであり，内部の処理に MegaScript は関与しない．

## 2.3 ストリーム

ストリームはタスク間で通信を行うための論理的な通信路である．あるタスクの標準出力の内容を，ほかのタスクの標準入力に流し込むことで，タスク間でデータの送受信を行う．一つのストリームの入出力にそれぞれ複数のタスクを接続することで，一対多，多対多などの通信を簡潔に実現できる．簡単な MegaScript プログラムの例を図 2.1 に挙げる．この記述により，図 2.2 のようなタスクネットワークが生成される．今回の例で，は Task a と Task b で得られた結果をストリームを通して Task c に送っている．

```
include MegaScript
#タスク・ストリームの定義
task_a = Task.new("./solver1")
task_b = Task.new("./solver2")
task_c = Task.new("./collector")
stream = Stream.new
#タスク・ストリームの接続
stream.connect(task_a, IN)
stream.connect(task_b, IN)
stream.connect(task_c, OUT)
#タスク・ストリームの生成
task_a.create
task_b.create
task_c.create
stream.create
#スケジューラの呼び出し
schedule
```

図 2.1: プログラム例

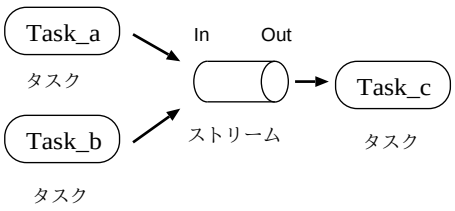


図 2.2: タスクネットワーク

## 2.4 メタプログラム

### 2.4.1 メタプログラム

MegaScript ではタスクに関する情報の記述方式としてメタプログラムを用いる．メタプログラムは元のタスクプログラムの制御構造と計算処理，入出力処理を抽象化して記述するものである．メタプログラムを図 2.3(a) のように記述すると，このタスクは図 2.3(b) のような処理を行うという情報を与えることができる．

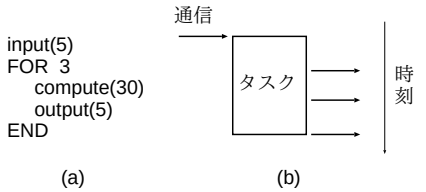


図 2.3: メタプログラムの表現

```
Class MyTask < Task
    def initialize(*arg)
        @exefile = "./compare_genomes"
        @arg = arg
    end
def behavior
    FOR @arg[1]
        input(7)
        compute(@arg[2])
        output(3)
    END
end
```

図 2.4: メタ情報を含むタスククラスの定義例

また，メタプログラムを与えるため，スケジューラの使用時には図 2.4 のように新たなタスククラスを定義し，メタ情報を与える．

### 2.4.2 メタ情報

処理系はメタプログラムからタスクごとのメタ情報を得る．メタ情報の内容は計算や入出力のコストである．これらのコストは静的に求まる

場合は数値となるが，静的に求まらない場合，例えばタスクの実行時引数によってループ回数が決まる場合や，入力がある度に処理を行うようなタスクの場合は，関数として表される．スケジューラはメタ情報を利用してスケジューリングを行う．

## 2.5 処理系

MegaScript の処理系は主にトランスレータ，ランタイムから構成される (図 2.5)．MegaScript プログラムを処 理系に渡すと，まずトランスレータがメタプログラム部分を解釈し，メタ情報ファイルを生成する．残りの Ruby コード部はそのままランタイムに渡される．ランタイムは API クラス定義を組み込んだ Ruby インタプリタとスケジューラ，およびメタモデルから成る．インタプリタは Ruby コードを実行し，タスクやストリームなど API クラスのオブジェクトを生成する．スケジューラはインタプリタから適時呼び出され，タスクやストリームの配置ホストを決定し，それらのホストにプロセスなどの生成を指示する．メタモデルはメタ情報を基に，実行中のプログラムが生成するタスクネットワークを抽象モデル化したものであり，実行中に得られた動的情報によって適時補正される．スケジューラはこのモデルを用いて各タスクの計算/通信コス

トを求め，戦略を決定する．

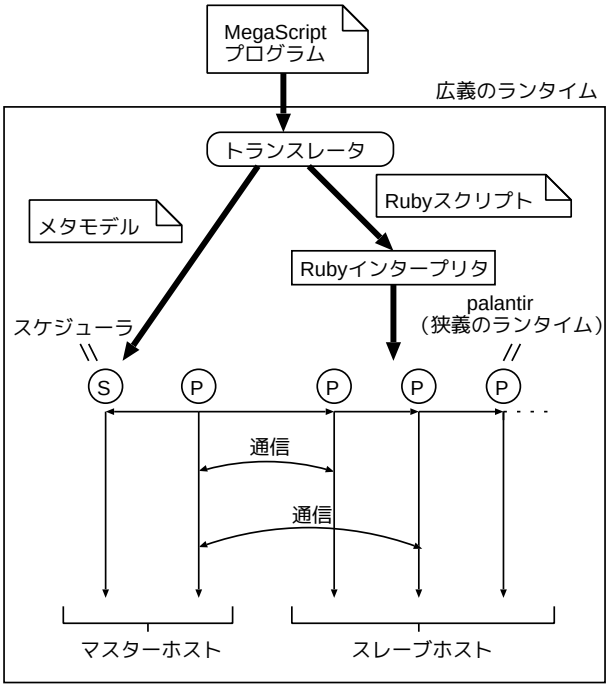


図 2.5: 処理系の概要

### 2.5.1 トランスレータ

まずトランスレータが MegaScript プログラムを受け取ると，メタプログラム部分を解釈してメタ情報の集合であるメタモデルを生成する．残りの Ruby スクリプト部分はランタイムに渡し，メタモデルはスケジューラへ渡される．

### 2.5.2 ランタイム

ランタイムはMegaScriptの本体と言える部分であり，タスクの生成・配置やタスク間通信などの並列実行機構を提供する．また，Rubyインタプリタを呼び出しRubyスクリプトを実行する．

### 2.5.3 スケジューラ

スケジューラはタスクの配置戦略や生成タイミングを決定する．スケジューラはトランスレータが生成するメタモデルから得られるメタ情報と，ランタイムから得られるタスクネットワークの接続情報をもとにスケジューリングを行ってタスクの配置を決定し，その結果をランタイムに伝える．

### 2.5.4 タスクとストリーム

マスターホスト上のマスタープロセスでは，スケジューラがタスク $T_i$ をスレーブホスト $H_j$ 上で 起動することを決定すると，$H_j$ 上のスレーブプロセスに $T_i$ の実行ファイル名や実行時引数などを送信する．そして，スレーブプロセスはタスクプログラムを子プロセスとして実行する．このときシステムコールpipe()およびdup2()を利用して両プロセス間で2本のパイプを接続し，タスクプロセスが自ホスト上のランタイムプロセ

スと双方向通信できるようにしている．一方，スレーブホスト上のランタイムに対しては，他ホストからの通信や自ホスト上のタスクプロセスからの出力が，非同期に発生する．これをイベントドリブンで処理するため，以下のスレッドを生成する．

**受信スレッド**　他ホストからメッセージを監視し，受信スレッド処理を行う．

**入力端ストリームスレッド**　システムコール select() によりタスクプロセスが書き込むパイプすべてを監視し，タスクの出力データを読み込む．

**出力端ストリームスレッド**　タスクプロセスが読み出すパイプすべてを保持し，必要に応じてタスクに送るデータを書き込む．

# 3　評価内容

## 3.1　評価用プログラムの詳細と選択理由

MegaScript とそのファイルアクセスの機能を評価するために，いくつかの実用的なプログラムや，それらを複数組み合わせたスクリプトを用意した．

### 3.1.1 プログラム概要

評価に用いたプログラムの選定に当たっては，まず，遺伝子解析やマルチメディア処理のような実用性の高いものであることとした．また，プログラム自体の実用性が高いだけでなく，MegaScript で並列実行するメリットが充分に大きいものである必要もある．その上で，データ転送量に違いのある以下の 3 種のプログラムを選んだ．

1. CG 作成プログラム

   CG(コンピュータ・グラフィック) を作成するプログラムを用いて，一枚の高精細な CG の作成を行う．[3]

2. 遺伝子解析プログラム

   2 本の遺伝子配列を比較し，一致する部位を探索する．

3. 天文画像の合成プログラム

   複数枚の天文 (FITS) 画像から，1 枚の大きな天文画像に合成する．

### 3.1.2 プログラムの詳細

1. CG 作成プログラム

   映画などで使用されるコンピュータグラフィックスでは，プロセッサの速度が向上により複雑で高品質なレンダリング技術が使えるよ

うになったため，ここ十数年，逆にレンダリング時間が増加している．したがって，レイトレーシング法という手法によって 3DCG を作成するプログラム POV-Ray[4] のバージョン 3.6 を用いて，高精細な CG を作成するプログラムを用意した．

レイトレーシング法とは 3DCG 作成手法の中でも計算量が多く，複雑な高品質画像を作り出すことの出来る手法の一つである．レイトレーシング法では,「光線がどのように反射，屈折するのか，ある物体にぶつかったときどうのような影響を受けるのか」といったことを光線を追跡しながらその都度計算していき，これを繰り返すことで色や光の強さなどを決定して画像を作り出す．ただし，画像から出た光線の中で目まで到達するものはその一部であり，光源のほうから追跡を始めると，ほとんどの光線の計算が無駄になってしまう．そこでレイトレーシング法では，通常目から発射された光線を追跡していく (図 3.6)．レイトレーシング法の実際の処理手順は以下のようになる．

- 視点からスクリーンの全ての画素に向かって視線をのばす．
- 視線が物体にぶつかれば，屈折光と反射光の 2 つの光線にわかれる．それぞれの光線の角度や強さは，物体の材質によって異

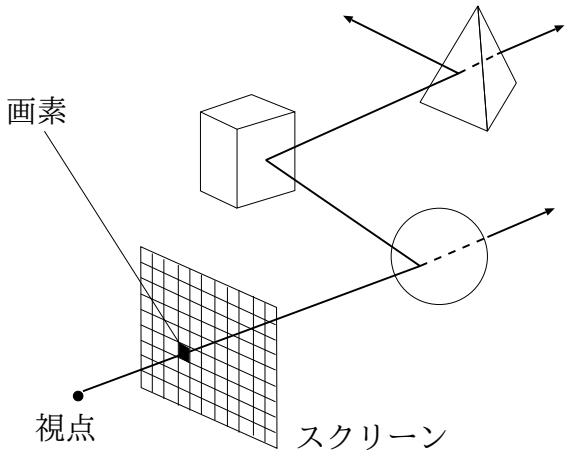


図 3.6: レイトレーシング法

なる．ぶつからなければ，背景色になる．

- それぞれの光線について，上記の作業を繰り返し，全ての影響を足し合わせた色を決める．

画像のレンダリングには，あらかじめ用意しておいたシーンファイルを読み込むことで行う．シーンファイルはカメラ，光源，物体などの位置や，物体の形，色を記述したテキストファイルである．今回は POV-Ray 付属の iortest を使用した．この POV-Ray を用いて横 4000，縦 3000 ドットの高精細な一枚の PNG 形式の画像を作成する．並列化するために画像を横方向にスライスし，分割してレンダリングするが，これを最終的に一つの画像にするために画像処理ソフト ImageMagick[5, 6] の crop という画像を結合するプログラム

を使用する．並列化について詳しくは後述する．

2. 遺伝子解析プログラム

遺伝子解析プログラムにはBLAST[7] を並列化したプログラムを作成し，使用した．BLASTは，バイオインフォマティクスで最も広く使われているプログラムの一つであり，DNA の塩基配列あるいはタンパク質のアミノ酸配列のシーケンスアライメント (問い合わせ配列) を行うためのアルゴリズムを指し，またそのアルゴリズムを実装したプログラムである．BLAST アルゴリズムを実行する際には，2 つの情報が必要である．クエリシーケンス（ターゲットシーケンスともいう）とシーケンスデータベースである．BLAST アルゴリズムは，シーケンスデータベース中のシーケンス断片と類似するクエリシーケンス中のシーケンス断片を見つけ出す．多くの場合，クエリシーケンスは，シーケンスデータベースと比べてデータ量が非常に小さい．例えば，クエリシーケンスが千個程度のヌクレオチド（核酸塩基）であるのに対し，シーケンスデータベースは数億のヌクレオチドのデータをもっている場合がある．

BLAST アルゴリズムは，クエリシーケンスとシーケンスデータベースとの間で，高い閾値でシーケンスアライメントを行う．

BLAST アルゴリズムの実行は，概念的には次に述べるように 3 段階に分かれている．

・ 第 1 段階: クエリシーケンスを短い固定長データに分割した断片で，シーケンスデータベースを厳密に検索する．この固定長データの長さを W（ワード）とする．例えば，W = 3 で，クエリシーケンス AGTTAC に対してデータベース中に ACTTAG というシーケンスが存在した場合，BLAST アルゴリズムは，TTA という部分データが両シーケンスで共有されていると認識する．W のサイズは，特に指定しない場合，ヌクレオチドでは 11，アミノ酸では 3 である．

・第 2 段階: BLAST アルゴリズムは，クエリシーケンスと，部分データを共有すると認識したデータベース中のシーケンス群に対して，ギャップを考慮しない単純なアライメントを行う．このギャップを考慮しないアライメントは，第 1 段階の W の長さの固定長での検索処理を，アライメントのスコアが高くなるように両方向に W（ワード）のサイズを拡張した処理である．この段階ではヌクレオ

チド（核酸塩基）やアミノ酸の挿入や欠損は考慮しない．先述の例では，AGTTAC と ACTTAG はそれぞれ TTA を共に含んでおり，ギャップを考慮しないアライメントは次のようになる．(図 3.7).

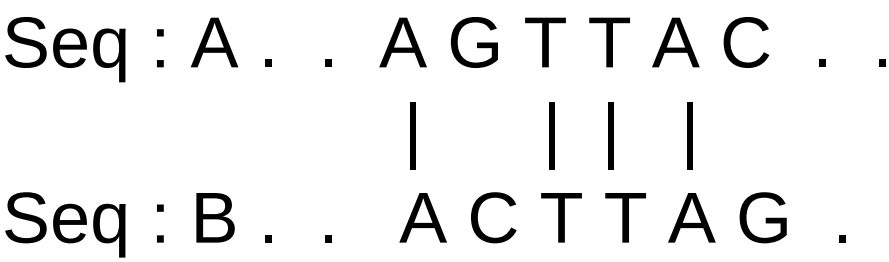


図 3.7: アライメント

第 2 段階で，ギャップを考慮せずに，スコアの高いアライメントが行えた場合，データベースのシーケンスデータは第 3 段階の処理対象となる．

・第 3 段階: BLAST アルゴリズムは，Smith-Waterman アルゴリズムの変形版アルゴリズムで，クエリシーケンスとデータベースのシーケンスの間で，ギャップを考慮したアライメントを行う．アライメントを行った後，統計的に有意なアライメント群がユーザに示される．

今回は BLAST に含まれているプログラムの中から塩基配列のクエリシーケンスを，6 フレームで翻訳して，アミノ酸配列に翻訳され

た塩基配列データベースから類似するシーケンスを検索するプログラムを使用した.

3. 天文画像の合成プログラム

天文画像の合成プログラムにはMontage[8] を用いている．Montageとは複数枚の画像を一枚の天文 (FITS) 画像に合成するプログラムである．１枚の画像を生成するには４つのステップを要する.

・ステップ１　画像の再投影

ステップ１では入力画像の再投影を行う．再投影の前に入力画像のメタテーブルを作成し，そのメタテーブルを使用し入力画像の再投影を行う.

・ステップ２　背景のモデリング

重なり合う画像間の背景をするため，重複する各ペアの差を調べ，差分画像のセットを作成する.

・ステップ３　背景のマッチング

差分画像のセットを使用して各画像に適用される修正テーブルを作成する.

・ステップ４　画像の結合

背景がマッチングされた画像を結合し，１枚の天文画像を作成する．

大まかな流れを (図 3.8) に示す．

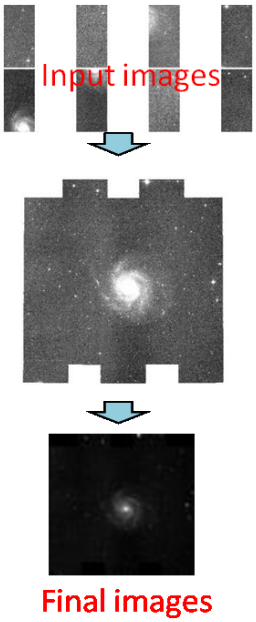


図 3.8: 天文画像合成の流れ

## 3.2　各プログラムの並列化手法

### 3.2.1　CG 作成プログラム

タスクは以下の 2 種類ある．

(a) CG 作成タスク…分割画像の作成

POV-Ray を使用する．並列実行のために，一枚の画像を水平方向で等分割してレンダリングし，それぞれをタスクとする．

例えば，4000 × 3000 ドットの画像を 3 つのタスクで分割してレンダリングする場合，タスク 1 は縦 1〜1000 ドット，タスク 2 は縦 1001〜2000 ドット，タスク 3 は縦 2001〜3000 ドットの範囲をそれぞれ処理する (図 3.9 左).

(b) 画像結合タスク

3a で分割して生成された画像を結合して一枚の画像にする (図 3.9 右).

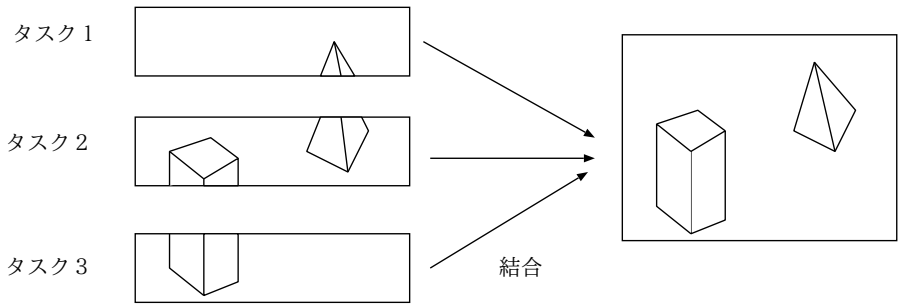


図 3.9: 静止画の分割レンダリングと結合

### 3.2.2 遺伝子解析プログラム

タスクは以下の 3 種類ある.

(a) 分割タスク…分割データベースの作成

データベースを等分割する (図 3.10 右).

(b) 検索タスク

blastall を実行し，(a) で分割されたデータベースと比較対象の配列を比較し，類似性を検索する．(図 3.10 中央).

(c) 結合タスク

(b) で検索された結果を結合する．(図 3.10 左).

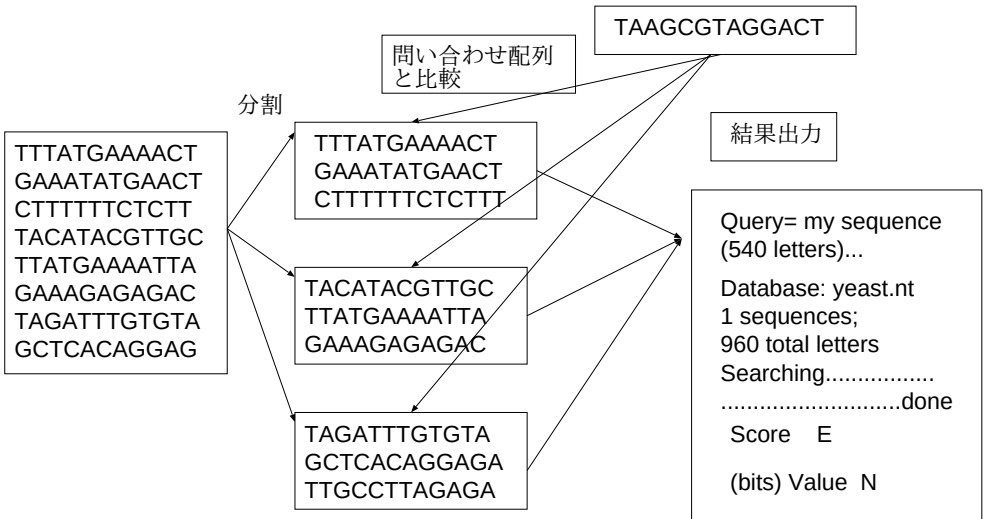


図 3.10: 遺伝子解析の流れ

### 3.2.3 天文画像の合成プログラム

天文画像の合成には以下の流れを順次行う．処理の流れを (図 3.11) に示す．今回並列化を行った部分は入力画像を結合用の画像に変換する mProjectPP のみであるが，他には各画像の背景の差を調べる mDiffFit，各画像の背景を除去した画像を作成する mBackGround

の部分が並列化可能である．

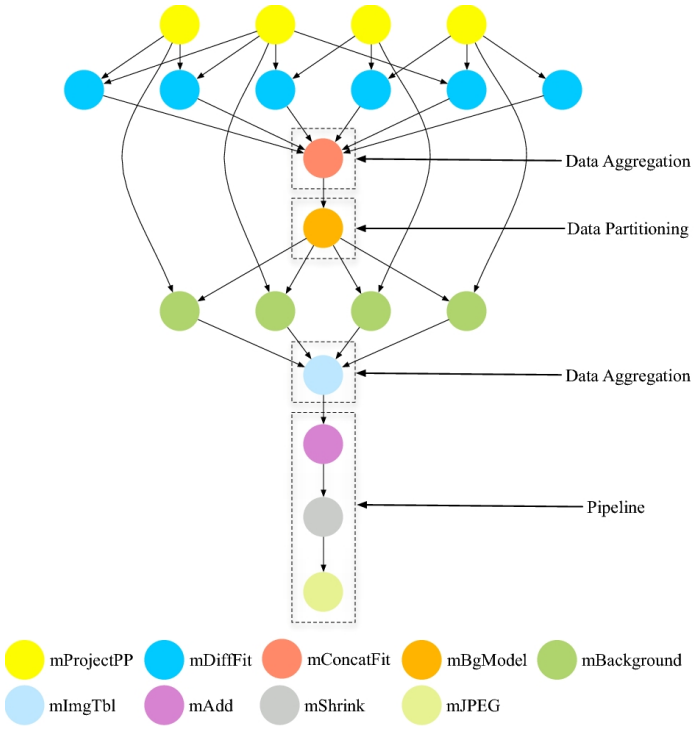


図 3.11: montage-workflow

# 4 評価結果

## 4.1 評価方針

今までの MegaScript にはファイルアクセスの機能は実装されていなかったが，現在の MegaScript ではタスクの出力ファイルを実行ホストのローカルディスクに一時的に保存し，その後，必要に応じてホスト間で転送することによってファイル入出力を扱う機能を新しく実装している．しかし，実用プログラムを使用して評価を行う状態ではない．従って今回は

表 4.1: 実行環境

| | PC クラスタの性能 |
|---|---|
| プロセッサ | Pentium4 2.8GHz |
| メモリ | 512MB |
| OS | Vine3.2(kernel-smp-2.4.31) |
| 通信ライブラリ | Phoenix 1.0 |

Network File System(NFS) を使用してファイルアクセスの評価を行った．MegaScript の実行時間と各タスクの実行時間の差を通信時間とし，各プログラムの通信時間の割合がどの程度占めるのかを調べた．評価環境を表 4.1 に示す．

## 4.2 評価結果

各評価プログラムの評価結果を表 4.2 に示す．CG 作成については台数効果による実行時間は４台あたりまでは大幅に減少できたが，それ以降はあまり減少は見られなかった．通信時間の割合は台数と共に多少増加したが大幅には増加しなかった．

遺伝子解析については台数効果による実行時間は８台まで大幅に減少した．通信時間の割合は並列台数を増やすにつれて増加したが，実際の通信時間はそれほど増加しない結果となった．

天文画像の合成については部分的に並列化を行ったため台数効果によ

表 4.2: アプリケーションの実行時間と通信時間の割合

| CG 作成 | | | | |
|---|---|---|---|---|
| ホスト数 | 1 | 2 | 4 | 8 |
| 逐次／ MegaScript(秒) | 1087.4 | 620.8 | 376.2 | 283.1 |
| タスクの計算時間の割合 (%) | 98.4 | 97.2 | 95.2 | 93.6 |
| 通信時間の割合 (%) | 1.6 | 2.8 | 4.8 | 6.4 |

| 遺伝子解析 | | | | |
|---|---|---|---|---|
| ホスト数 | 1 | 2 | 4 | 8 |
| 逐次／ MegaScript(秒) | 377.1 | 195.3 | 104.5 | 62.0 |
| タスクの計算時間の割合 (%) | 94.6 | 92.8 | 92.3 | 85.9 |
| 通信時間の割合 (%) | 5.4 | 7.2 | 7.7 | 14.1 |

| 天文画像の合成 | | | | |
|---|---|---|---|---|
| ホスト数 | 1 | 2 | 4 | 8 |
| 逐次／ MegaScript(秒) | 435.2 | 376.5 | 346.9 | 347.0 |
| タスクの計算時間の割合 (%) | 82.9 | 80.1 | 78.5 | 73.7 |
| 通信時間の割合 (%) | 17.1 | 19.9 | 21.5 | 26.3 |

る実行時間はあまり減少しなかった．通信時間は逐次実行でも通信時間の割合が高く並列台数が増えるにつれて通信時間の割合が大幅に増加した．

# 5 考察

評価結果から，CG 作成の実行時間が 4 台あたりで台数効果による減少があまり見られなかった

これはデータ量が，小さく分割して作成する画像の計算量が各タスク

毎に大きく違うからである．通信時間に関しては扱うファイルの容量が比較的小さいため，並列台数が増加しても通信時間の占める割合はあまり増加は見られなかった．

遺伝子解析の実行時間は並列台数が増加するにつれてある程度実行時間の減少が見られた．これはデータベースを等分割しており，各タスクの計算量が等しく終了時間が近いためだと考えられる．通信時間に関しては，CG 作成と同じくデータ量は小さいが CG 作成より通信時間の割合は増加した．これは各タスクの計算量が等しく同時に通信が発生し NFS サーバにアクセスが集中してしまうからだと考えられる．しかし CG 作成に比べて実行時間が短いので多少の増加は無視できるほどである．

天文画像の合成については実行時間は部分的にのみ並列化を行っているためあまり台数効果は得られなかったが，まだ並列化可能な部分がいくつかあるため改善の余地がある．通信時間に関しては他の２つのプログラムに比べて通信時間の割合が高かった．これはデータ転送量が大きく，ワークフローが複雑なためファイルアクセスの回数が多いためであると考えられる．

CG 作成，遺伝子解析のようなデータ転送量が比較的小さいプログラムでは NFS を使用した場合でも通信時間が実行時間の大きな問題とならな

いことがわかった．しかし，天文画像の合成のようなデータ転送量が大きく，ファイルアクセスを頻繁に行うプログラムでは通信時間の増加が大きな障害となることがわかった．

# 6　まとめ

NFS を使用する場合，データ転送量の大きくファイルアクセスを頻繁に伴うアプリケーションでは通信時間の増加が大きな障害となった．そのため，MegaScript 独自のファイルシステムを使用することにより通信時間の削減することが非常に重要である．MegaScript 独自のファイルシステムが使用可能になれば，画像を扱うアプリケーション，テキストファイルを使用するアプリケーション等，様々なアプリケーションを今後 MegaScript で並列化できるようになる．

今回の評価では NFS を用いたファイルアクセスの評価を行ったため，実際に MegaScript のファイルアクセスの機能を使用して評価を行うことが今後の課題である．また，今回は各ホストの性能が同一の環境で評価を行ったが，このような PC クラスタを複数ネットワークで繋いだような，大規模且つ不均質な環境での評価も今後行っていく必要がある．

# 謝辞

本研究を行うに当たって，日頃からご指導，ご助言いただきました大野和彦講師に深く感謝いたします．また，多くの助言をして下さった近藤利夫教授，佐々木敬泰助手，温かい心配りを頂いた田中みゆき事務員，そして研究室の方々に心よりお礼を申し上げます．

# 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] 大塚保紀，大野和彦，中島浩，:タスク並列スクリプト言語 MegaScript によるタスク動作モデル記述，情報処理学会研究報告 2003-HPC-95，pp.113-118(2003)．

[3] 萩原 大造：実用プログラムによる MegaScript 処理系のスケジューリング性能評価，卒業論文，三重大学 (2007)．

[4] 小室日出樹：POV-Ray ではじめるレイトレーシング 改訂二版，アスキー (2005)．

[5] ImagaMagick: Convert, Edit, and Compose Images

`http://www.imagemagick.org/`

[6] ImageMagick

`http://mechanics.civil.tohoku.ac.jp/soft/node43.html`

[7] Blast Help

http://blast.ncbi.nlm.nih.gov/Blast.cgi

[8] Montage

http://montage.ipac.caltech.edu/

# A　プログラムリスト

# B　評価用データ