卒業論文

題目

タスク並列スクリプト言語MegaScriptにおける バイナリストリームの設計と実装

指導教員

大野 和彦 講師

2013年

三重大学 工学部 情報工学科
コンピュータソフトウェア研究室

田端　美香 (410832)

内容梗概

近年，遺伝子解析や画像処理などの分野において大規模な物理計算への需要が高まっている．それに伴い，高速に処理が行える並列処理が必要とされているが，大規模な並列処理プログラムを記述するにはホスト管理やスケジューリングなどの専門的な知識が必要であり，高いプログラミング能力が求められてしまう．そこで我々は大規模な並列処理を容易に行うための言語として，タスク並列スクリプト言語 MegaScript を開発している．

MegaScript はタスク間の通信をストリームと呼ばれる仮想通信路が行っている．しかし，従来の MegaScript のストリームは，テキストデータの通信しか想定しておらず，画像や音声などのバイナリデータの通信を行うためには，プログラマがテキストデータに変換／復元する手間を要する．そこで，バイナリデータの通信を容易に行うことができるバイナリストリームを設計・実装し，評価を行った．

実装したストリームの有用性を示すために，実装したストリームと従来のストリームを用いてバイナリデータの転送時間を測定した．その結果，バイナリデータを容易に転送することが可能となり，転送時間を最大 70%削減できた．

Abstract

In recent years, demands to large-scale physical calculation increases in fields such as the image processing or gene analysis. The parallel computation that processing can perform fast is required with that. However, The programmer is also required deep knowledge, such as host management and scheduling, and high ability for large-scale parallel programming. Therefore, a task-parallel script language MegaScript is developed for the large-scale computing.

In MegaScript, virtual channel called stream communicate between tasks. However, existing stream of MegaScript suppose only communication of text data. The programmer needs trouble to encode/decode into text data to communicate of binary data such as images or voice. So I implemented binary stream which can communicate binary data easily.

目 次

図 目 次

表 目 次

1　はじめに

近年，遺伝子解析や画像処理などの様々な分野において大規模な物理計算への需要が高まっている．しかし，大規模な計算を行う並列処理プログラムを記述するにはホスト管理やスケジューリングなどの専門的な知識が必要となってしまう．そこで我々は大規模な並列処理を容易に行うための言語として，タスク並列スクリプト言語 MegaScript を開発している．本研究では MegaScript のストリームを改良することにより，より多様なアプリケーションを MegaScript で扱えるようにした．

本文の構成は以下のようになっている．第 2 章でまず MegaScript について述べ，第 3 章にバイナリストリームの設計と実装，第 4 章に評価を行い，最後に第 5 章にてまとめを行う．

2　背景

2.1　タスク並列スクリプト言語 MegaScript

2.1.1　基本設計

MegaScript は fig.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するプログラミング言語である．MegaScript では，任意の言語で書かれた独立したプログラムをタスクとして扱う．そのようにすることでプログラマはプログラムの再利用やプログラムに適した言語を用いることができる．そして，そのタスクの標準入出力をストリームが中継することにより，タスク間の通信を行う．しかし，MegaScript のストリームはバイナリデータの通信を想定しておらず，画像や音声などのデータの通信を行うには，バイナリデータを一度全て可読文字に変換してからストリームにデータを送らなければならない．

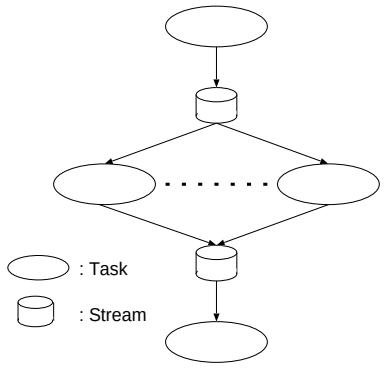

fig . 2.1: ワークフローの例

2.1.2 ストリーム

MegaScript におけるストリームとは，タスク間の仮想通信路である．ストリームはタスクの標準出力を別のタスクの標準入力へと繋ぐ．このときストリームは複数のタスクを接続することが可能である．複数のタスクがストリームの入力側に接続されている場合，タスクからの出力は非決定的に併合される．複数のタスクがストリームの出力側に接続されている場合，出力はそれらのタスクへマルチキャストされる．

また，従来のストリームはテキストデータの通信を想定しており，このようなストリームをテキストストリームと呼ぶ．

3　提案手法

3.1　概要

画像や音声処理などのアプリケーションを MegaScript で使用する場合，従来のストリームではバイナリデータのタスク間通信ができない．なぜなら，バイナリデータ中の ‘0a’ や ‘00’ などのバイトを改行文字やヌル文字であると認識し，データの欠落などが発生するからである．したがって，テキストストリームでバイナリデータの通信を行うためには，ユーザーがそれをテキストデータに変換する必要があり，非効率である．そこで，バイナリデータのタスク間通信を可能とするバイナリストリームの実現を考える．

ストリームは，ストリームの入力側に接続されたタスクの標準出力からデータを受け取り，出力側に接続されたタスクの標準入力へデータを転送する．テキストデータの場合，タスクはデータの末尾に改行文字を付加して標準出力する．テキストストリームではその改行文字を区切りとして，タスク間の通信を行う．バイナリストリームでは，任意のバイト列で構成されるバイナリデータを想定しているので，区切りの文字を設定できない．データを区切らないと，タスク側が入力の切れ目がわからなかったり，全てのデータが出力されるまで転送が開始されず，待ち時

間が発生してしまったりする．そこで，データサイズで区切ることを考える．

また，特定のバイトが含まれるデータを通信すると不具合が発生する．例えば，データの長さを取得する際，区切ったデータの末尾にヌル文字を付加して，それまでの長さを求めるが，データ中にバイト ‘00’ が存在するとそれをヌル文字と認識してしまい，以降のデータを無視する．このように ASCII コードの制御文字に相当するバイトがデータ中に含まれると，正常に通信ができない．そこで，そのようなバイトを別のバイトに置換する方法を考える．

3.2 実装

テキストストリームではタスクから標準出力されたデータを改行文字まで読み込み，通信を行う．バイナリストリームでは，データの中身ではなくサイズで区切るようにする．したがって，あらかじめ設定しておいた区切りサイズ分を一度に読み込む．標準出力内にデータがない状態で読み込みを行うとエラーが発生するので，終了条件として，データ読み込み後にそれが標準出力の末尾かどうか判定し，末尾なら読み込みを終了する．このようにすることで，一定サイズごとに区切ったタスク間通信が可能となる．また，テキストストリームは，末尾に改行文字を付

加してタスクに書き込むが，バイナリストリームは付加しないよう変更を行った．

特定のバイトを制御文字であると認識することで発生する不具合は，それらのバイトをタスクから読み込み後，別のバイトに置き換え，書き出す前に復元することで防いだ．置き換えるバイト列の簡単な例を fig.3.2 に示す．fig.3.2 のバイト列は末尾にヌル文字を付加してサイズ取得すると 4 バイト目が原因で 3 となる．したがって，‘00’ を別のバイトに置換することを考える．しかしながら，‘00’ を ‘90’ に置換すると，復元の際 6 バイト目の ‘90’ を ‘00’ にしてしまう．そこで，2 バイトのバイト列に置換する．fig.3.3 に示すように ‘00’ を ‘90 80’，‘90’ を ‘90 90’ と置換することで置換と復元を可能にする．しかし，画像データなどは ‘00’ というバイトが使用されることが多く，置換するとデータ量が増加し，性能低下に繋がる．そこで，‘00’ を 1 バイトに置換する．置換の対応を表した表を Table 3.1 に示す．

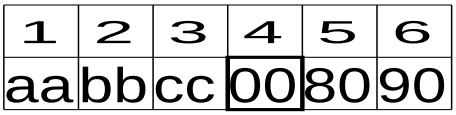

fig . 3.2: バイト列の例

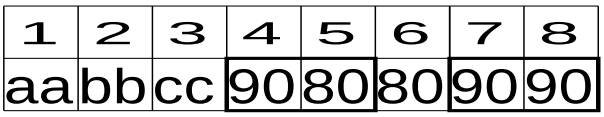

fig . 3.3: 置換の例

置換前	置換後
00	80
80	90 80
90	90 90

Table . 3.1: 置換の対応表

4 評価

4.1 評価方法

実装したストリームの有用性を示すために，バイナリデータをテキストストリームで転送したプログラムとバイナリストリームで転送したプログラムの置換後のデータ長と転送時間の評価を行った．このとき，データサイズを 1,2,4,8,16MB とした．バイナリストリームはデータの区切りサイズを 8KB としている．評価環境は Athlon II X2 3.0GHz，メモリ 2GB を搭載した計算機を使用した．

4.2 評価結果

Table 4.2 にデータサイズを変化させて測定した置換後のデータ長と転送時間を示す．このとき，テキストストリームの転送時間には，バイナリデータを Base64 を用いてテキストデータに変換し，復元する時間も含めている．同条件での評価のために，テキストデータに変換済みのものの転送時間を測定した．テキストストリームを用いた場合には転送の際，そのままデータ転送するものと，タスクが 8KB ごとに区切り文字である改行文字を挿入したものを測定した．

Table . 4.2: 置換後のデータ長 (MB) と転送時間 (秒)

データ (MB)		1	2	4	8	16
置換後の	テキスト	1.33	2.67	5.33	10.66	21.32
データ長	バイナリ	1.02	2.03	4.06	8.12	16.24
実行時間	テキスト	0.30	0.75	2.29	7.85	29.00
	バイナリ	0.15	0.29	0.74	2.40	8.65

Table . 4.3: テキストデータの転送時間 (秒)

データ (MB)		1.33	2.67	5.33	10.66	21.32
バイナリ		0.40	0.61	1.36	4.25	15.13
テキスト	区切りなし	0.43	0.77	2.15	7.37	27.76
	8KB 区切り	0.38	0.62	1.38	4.08	14.40

4.3 考察

Table 4.2 からわかるように，バイナリストリームを用いることで置換後のデータサイズの増加を抑えることができた．これは，テキストストリームでは可読文字以外を全て変換する必要があるのに対し，バイナリストリームではヌル文字と数種類の制御文字の置き換えのみで良いためである．また，転送時間を最大 70%削減することができた．これは，置換後のデータサイズの増加を抑えられたためと，置換・復元の時間を削減できたためであると考えられる．データ量が増加するほど削減率は高くなる．

また，Table 4.3 より，テキストストリームでも区切りを入れることで

転送時間は短縮できている．一度に転送するデータ量が大きいと，輻輳による遅延が起こるためではないかと考えられる．しかし，テキストストーリームでデータを区切って転送するためには，タスク側が区切らなければならず，また区切り文字である改行文字を削除する手間が発生する．

以上から，バイナリストリームは有用であると考えられる．

5　おわりに

本研究では，より多様なアプリケーションを MegaScript で扱えるようにするバイナリストリームを設計・実装し，評価を行った．その結果，本ストリームを用いることでバイナリデータの通信が可能になった．また，テキストストリームを用いるよりも転送時間を削減することができた．

今後の課題として，データの区切りサイズを，実行中に最適なサイズへと自動で調整することが挙げられる．また，データの置換を行わないよう実装できれば，置換の時間が削減でき，転送時間のさらなる削減が見込まれる．

謝辞

本研究を行うにあたり，ご指導，ご助言いただきました下さいました大野和彦講師，並びに多くの助言をいただきました山田俊行講師に深く感謝いたします．また，様々な局面にてお世話になりましたコンピュータソフトウェア研究室の皆様にも心より感謝いたします．

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 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] 大野 和彦, 大塚 保紀, 西里 一史, 阪口 裕輔, 高木 悠志, 中島 浩:タスク並列スクリプト言語 MegaScript ユーザマニュアル Ver0.05(2007).

[3] 西里　一史，大野　和彦，中島　浩：タスク並列スクリプト言語 MegaScript のランタイムシステムの設計と実装，情報処理学会研究報告，2003-HPC-95，pp.119-124(2003) .