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

2005－HPC－103 (27)
2005／8／5

効率の良い広域分散対応のタスク並列スクリプト言語の実現

高木 祐志[†] 西川 雄彦[†] 大野 和彦[†]
佐々木 敬泰[†] 近藤 利夫[†] 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 を開発している．MegaScript は，独立したプログラムをタスクとして扱い，複数のタスクを並列に実行する．実行制御はマスターホスト上のスケジューラが行うが，集中制御方式をとっているため大規模化する際のボトルネックとなっている．本論文では，MegaScript を広域分散環境上で効率良く動作させるための通信機構とランタイムについて述べる．

Widely-distributed Implementation of Task Parallel Script Language MegaScript

YUJI TAKAGI,[†] TAKEHIKO NISHIKAWA,[†] KAZUHIKO OHNO,[†]
TAKAHIRO SASAKI,[†] TOSHIO KONDO[†]
and HIROSHI NAKASHIMA[††]

We are developing a task parallel script language MegaScript for megascale computing. MegaScript regards independent programs as tasks, and executes them in parallel. Single scheduler controls all tasks in current implementation, but it will cause large overhead in megascale environment. In this paper, we describe the design of the communication and runtime for widely-distributed environment.

1. はじめに

近年，遺伝子解析や気象／災害のシミュレータなどでは複雑な数値計算を要するため，1Pflops 以上の計算能力が期待されている．2005 年 2 月現在，世界最速である米 IBM のスーパーコンピュータでは，13684CPU を使って 70Tflops を達成している．しかし，スーパーコンピュータの設置には膨大なコストが必要になるという問題点があり，スーパーコンピュータの延長で Pfops を実現するのは現在のところ極めて困難である．このためコモディティな技術を用いた「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」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，それらの資源を効率よく利用する実行システムが必要である．そこで，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[1] を開発している．

現在の MegaScript 処理系[2][3] は，並列計算に参加するホストが増加するにつれ，通信の負荷が制御を行うホストに偏り，性能低下を引き起こしてしまう．さらに，全ての計算ホストが制御ホストと互いに直接通信し合える環境でないと正常に動作しないという問題点を抱えている．

これらの問題を解決するには，現在の MegaScript の動作モデルを変更し，さらに処理系の通信機構を設計しなおす必要がある．

以後，第 2 章で MegaScript 言語の概要を述べ，第 3 章で MegaScript 言語の基本機能が処理系でどのように実装されているかを説明する．第 4 章で現在の MegaScript の問題点に触れ，その解決策を提案する．第 5 章で提案手法の設計を述べ，第 6 章で予備評価を行い，最後にまとめる．

2. タスク並列スクリプト言語 MegaScript

2.1 MegaScript の概要

メガスケールの並列性を持つプログラムを一から記述するのは非常に困難である．その解決策として，逐次プログラムや部分問題を並列化した小規模な並

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

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「多重並列モデル」が考えられている．MegaScript は，この多重並列モデルを意識して設計されているプログラミング言語である．

現在の MegaScript 処理系は，マスターホスト 1 台とそれ以外のスレーブホストとで構成されたマスター／スレーブモデルで動作する．マスターホストは全てのスレーブホストの制御を一括して管理する集中管理方式によって並列計算を実現している．

2.2 タスクとストリーム

MegaScript では，並列実行単位を「タスク」と呼ぶ．タスクは独立性の高いモジュールであり，ユーザは処理の主要部分を別プログラムで用意する．MegaScript プログラムではこれらの実行プログラムをタスクとして定義し，生成・実行する方式を取る．また，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している．

タスク間通信の際に使用する論理的な通信路を表すために，「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することが可能であり，タスクの標準出力をストリームの入力端に接続することにより，ストリームにデータを流すことができる．ストリームを流れるデータは，ストリーム出力端に接続しているタスクに標準入力として受け渡される．ストリームの入力端に複数のタスクが接続されたときはストリームに流れるメッセージは行単位で非同期にマージされ，ストリームの出力端に複数のタスクが接続されたときは接続している全てのタスクに対してマルチキャストが行われる (図 1)．

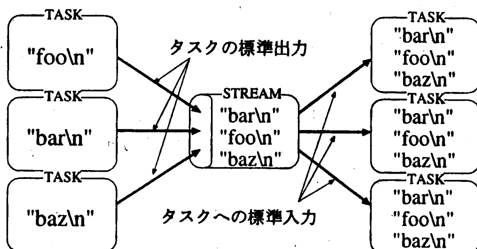

図 1 ストリームの振る舞い
Fig. 1 Behavior of Stream

2.3 スケジューラ

定義したタスクやストリームをどのホスト上に配置・生成するかは，ユーザが意識し MegaScript プログラム中に明示する必要はない．これらは MegaScript ランタイム上に存在するスケジューラが自動的に配置戦略や生成タイミングを決定し，効率の良い実行処理を行う．

2.4 プログラミング

MegaScript はオブジェクト指向言語 Ruby[4] を拡張した言語である．Ruby で提供されている基本機能はそのまま使用することができるため，必要に応じて複雑な処理を記述することが可能である．MegaScript ではさらに，独自に用意したランタイム API を提供し，非常に柔軟で強力な並列処理の記述が可能である．

MegaScript では，タスク・ストリームを表すクラスを提供している．ユーザがタスク・ストリームを扱う際にはこれらのクラスのインスタンスを作成し，メソッドを呼び出すという形で行う．タスク・ストリームを用いた基本的なプログラミングの流れを以下に示す．

- タスク・ストリームの定義
  並列に動作させる実行プログラムをタスクとして定義し，インスタンスの生成を行う．また，それらのタスクを接続するためのストリームの定義とインスタンス生成を行う．
- タスク・ストリームの接続
  定義したタスク間でデータの受け渡しができるように，タスク間をストリームで接続する．これにより，論理的なタスクネットワークの構造が形成される．
- タスク・ストリームのスケジューリングキューへの登録
  定義されたタスクやストリームをスケジューリング待ちキューへ登録する．
- スケジューリング
  MegaScript スケジューラを呼び出し，スケジューリング待ちキューに登録されているタスク・ストリームのスケジューリングを行う．スケジューラは登録されているタスク・ストリームの情報から，効率の良いホスト配置を自動的に選択し，タスクの実行を指示する．

3. MegaScript 処理系

3.1 タスクとストリームの扱い

図 2 を用いて，処理系でのタスクとストリームの実現方法の説明を行う．タスクプログラムが標準出力に対してデータの出力を行うと，ランタイムが出力データをパイプにより受け取り (1)，接続先のストリームと，そのストリームが設置されているホストをそのタスクに対応するタスククラスのインスタンスから参照する (2)．その後，ストリーム設置ホストのランタイムを経由し，ストリーム処理機構へデータが受け渡さ

れ (3), 最後に, 目的のタスクの標準入力へデータが渡される (4).

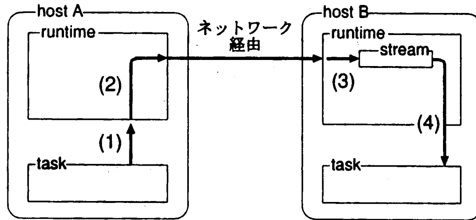

図 2 ランタイムのタスク間通信処理
Fig. 2 Communication between Tasks in Runtime

このようにして, タスク・ストリームが処理系で実装されている.

3.2 ランタイム間の制御

ランタイム間で行われる制御はすべて, システムメッセージを相手ホストへ送信することにより行われている. ある特定のホスト上のランタイムが, 他ホストのランタイムの制御を行いたい際には, その制御内容をシステムメッセージに記し, 相手ホストに送信する. システムメッセージを受信したランタイムは, そのメッセージから制御内容を確認し, 内容に沿った処理が実行される. また, タスク間通信にもシステムメッセージが利用されている.

3.3 通信機構

システムメッセージの通信は非同期に行われているため, 常にシステムメッセージの受信を確認する必要があり, 受信専用のスレッドがランタイムの実行中に並行に動作している.

MegaScript 処理系では, システムメッセージのホスト間通信に MPI[5] を利用している. MPI はスレッドアンセーフであり, ブロッキング受信を行っている最中はシステムメッセージの送信を行うことができない. また, 同様にシステムメッセージの送信中に受信関数を呼び出すこともできない. このため, メッセージの送信や受信を行う処理では排他制御を行い, 同時に送信処理と受信処理が行われないようにしている. さらに, 上記の理由で受信スレッドではブロッキング受信が行えないために, ポーリングによる受信処理を行っている.

4. 現在のランタイムの問題点と解決案

4.1 問題点

我々が考えている現在の MegaScript ランタイムの問題点は以下の二点である.

一つ目の問題点はマスターホスト上で動作するスケジューラが他ホストへのタスク配置や制御を集中管理していることである. この方式では制御用のメッセージがマスターホストに集中するため, スレーブホストの増加に伴い, マスターホストがシステム全体のボトルネックとなる.

次の問題点は全ての計算ホストがマスターホストに対して直接接続しなければならないことである. MegaScript 処理系の通信機構として使用している MPI は, 通信の際に指定された通信プロトコルによってダイレクトにメッセージ送信を行う. しかし, 近年のネットワーク環境ではセキュリティのために, あるネットワーク内のマシンに対して, ネットワーク外からの通信を特定のゲートウェイのみに限定しているケースも考えられる. そのため, マスターホストから直接コネクションを貼ることができないホストに対しては並列計算に参加できないという問題が発生し, せっかくの計算資源も利用できない状況に陥ってしまう.

4.2 解決案

これらの問題点を解決するためには, マスターホスト上のスケジューラの通信負荷を分散させる動作モデルへの変更が必要であり, それによってコネクションがマスターホストに集中しない処理系の実現も可能になる.

MegaScript で想定している実行環境は, 単一ドメイン内に接続されているクラスタやスタンドアローンのマシンの集合体が, 複数の外部ドメインの集合体と連携しあい, 並列計算を行うというものである. 多数の計算マシンを保有するドメイン内では, LAN 内のネットワーク接続を階層化しているケースが多い.

これらにより, MegaScript の実行システムにも, 単一ドメイン内のホストの集合体 (以後, ホストグループと呼ぶ) を単位とした階層構造を利用するのが最も適していると考え, MegaScript 処理系の実装方針に採用する. 階層構造をうまく利用し, 効率の良い実行システムを構築するためには階層型スケジューラと階層型通信機構の連携が重要になる.

5. 広域分散化手法

5.1 階層型スケジューラの設計

MegaScript 処理系のスケジューリングによる負荷分散を行うために, 広域分散環境上で効率良く動作するスケジューラを現在開発中であり, 以下にその概要を示す.

各ホストグループには代表ホストを設定し, そのホスト上でスケジューラが動作する. 代表ホストはホス

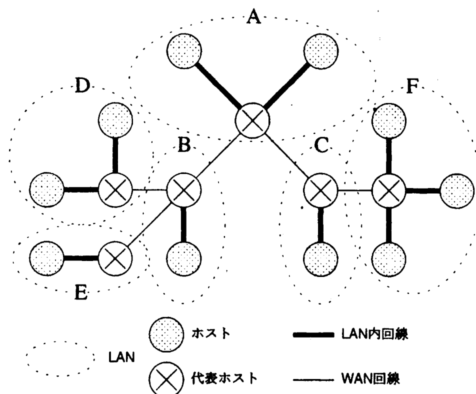

図 3 実際のネットワーク構造
Fig. 3 Network Structure

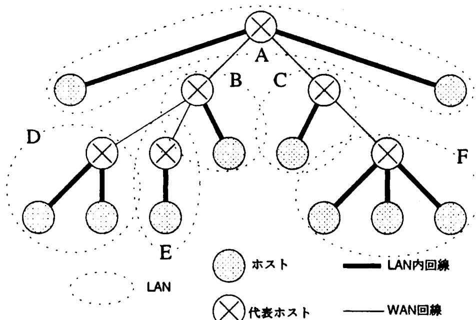

図 4 提案手法の全体像
Fig. 4 Example of Hierarchical Structure

トグループ内のホストおよび下階層の代表ホストを統括し，タスクを配置する．スケジューラは，タスク間通信を出来る限り同一ホスト内に閉じ込め，遅延の大きいWAN経由の通信が少なくなるようにタスクのスケジューリングを行う．以上のような階層構造を持つ論理ネットワークを，広域分散環境上に動的に構築し，スケジューラ負荷を分散しやすい環境を実現する．

例えば，図3のようなネットワーク構造を持つ複数のホストグループを取り扱うときは，処理系では図4のようなネットワークとして認識を行う．(図3のA-Fが，図4のA-Fと対応する)

5.2 階層型通信機構の設計

全てのタスク間通信を同一ホスト内に閉じ込めるようなスケジューリングを行うと，タスク配置が特定のホストグループに偏ってしまう危険性があるため，ある程度のホスト間通信は必要不可欠となる．

並列計算に参加するマシンの情報については，各ホストグループの代表ホストが管轄するホスト情報のみを保持し，管理を行うことで負荷分散を行うことが望ましい．

この設計だと，マスターホストは実行中の全ての計算マシンのリストを保持しないことになる．よって，全ホストのMegaScript処理系の起動処理には，マスターホストのランタイム起動後に動的に並列計算参加マシンのプロセスの追加を行う必要がある．しかし，MPIはプロセスの動的追加には対応していないため，MPIを用いたランタイムの実装では，階層構造による論理ネットワークを構築するのが困難である．

この問題を解決するには通信ライブラリによる支援が必要であり，通信機構を検討し直す必要がある．

5.3 通信機構の改良

前節で述べた通り，MPIを用いた通信機構では，階層構造を構築するMegaScriptランタイムの実装は困難であり，これらの問題を解決するための通信ライブラリに切り替える必要がある．これらの要件を満たすものとして，MegaScriptの通信機構にPhoenix[6]を採用する．Phoenixは，動的プロセス参入が可能なグリッド環境対応のメッセージパッシングライブラリである．Phoenixでは，ルーティング機能をサポートしており，直接コネクションを貼れない環境においても，下層でメッセージのフォワードが行われる．

Phoenixはスレッドセーフで設計されているため，受信スレッドがブロッキング受信を行っている最中にもデータの送信が可能である．よって，MegaScript処理系のポーリング受信をブロッキング受信として実装しなおすことで，MegaScript処理系で行っているポーリング受信処理や排他処理により発生するオーバーヘッドの緩和が期待できる．

5.4 階層型ランタイムのタスク間通信

階層構造型のMegaScriptでは，スケジューラが各ホストグループの代表ホストごとに設置される．マスターホストが階層構造用に作成したスケジューリング結果に対してサブマスターのスケジューラが再スケジューリングを行う．さらにその下階層にホストグループが存在する時にはこれが繰り返され，下層までタスクが転送される．

このとき，ドメイン間を跨ぐタスク間通信を行うタスクが存在したとすると，送信元がタスククラス内の情報として受け取った送信先ストリームは，各代表ホストが行うスケジューリングにより下層まで転送されているため，保持する情報と，転送先で実際に配置されたホストが一致しない問題が発生する．

この問題を解決するため，接続ストリームのホスト解決機構を導入する．例として図5のホストAが，接

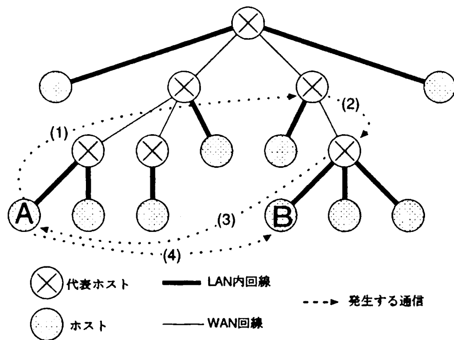

図 5 ストリーム設置ホスト解決機構
Fig. 5 Host Resolution Mechanism

続先ストリームを保持するホスト B に対して通信を行いたい際に，どのようにホスト解決が行われるかを説明する．タスク情報として渡されるストリーム ID の保持ホストには，特定のドメインの下階層に配置されている代表ホストが上位層のスケジューラにより指定されている．実際に通信が必要になった際には，タスク情報に指定されている代表ホストに対して，通信先のストリーム保持ホストの問い合わせ依頼を行う (1)．問い合わせ依頼を受け取った代表ホストは，スケジューリングの履歴より，さらに下層の代表ホストに問い合わせを行う (2)．そして，実際にストリーム配置を確定させたスケジューラまで問い合わせが伝達されると，そのスケジューラがストリームのホスト番号をダイレクトに依頼ホストまで伝達し (3)，ホスト番号が解決される．解決されたホスト番号を用いて，送信先ストリーム ID に対してメッセージの送信を行う (4)．このようにして，どのホスト上のストリームに対してもタスク間通信が可能になる．

6. 予備評価

6.1 評価内容

MegaScript 処理系に階層型通信機構を実装するにあたり，ランタイム間の通信に Phoenix を用いた処理系の実装を行い，MPI と Phoenix を切り替えられるようにした．実装前の通信性能と比較するため，通信が頻繁に行われる評価用アプリケーションを用いて実行時間の比較を行った．

また，Phoenix によるポーリング受信とブロッキング受信の性能比較も行い，ポーリング受信とブロッキング受信でどの程度性能に影響がでるのかを確認した．

評価用のアプリケーションには，数バイトの文字列を交互に送信するピンポン型通信プログラムを採用し，通信回数を変更させて実行時間の比較を行った．

6.2 評価環境

評価用の実行環境には，以下の表 1 に示すスペックのクラスタを利用した．

PC	DELL PowerEdge800
OS	Fedora Core 3(kernel-2.6.11)
CPU	Pentium4 2.8GHz
Memory	512MB ECC Registered
Network	Gigabit Ethernet

表 1 評価用 PC 性能
Table 1 PC Performance for Evaluation

また，評価用のクラスタにインストールされているソフトウェアは以下のとおりである．

- gcc 3.4.2
- MPICH 1.2.6
- ruby 1.8.2
- Phoenix 1.0

なお，Phoenix を用いた処理系では，メッセージの通信プロトコルとして SSH を用いた実装を行っている．

6.3 評価結果

ピンポンの通信回数として，0 回，100 回，500 回，1000 回，5000 回をそれぞれ 10 回ずつ実行し，その平均の実行時間の算出を行った．0 回の評価を取るのは，MegaScript の初期化処理と終了処理にかかるオーバーヘッドを測定するためであり，実行結果からこの時間を差し引くと，通信に費したおおよその時間を求めることが出来る．

実行時間の測定結果を表 2 と図 6 に示す．括弧内の数字は，実行時間から 0 回ピンポンの実行時間を差し引いた値である．

6.4 考察

図 6 は，表 2 の結果をグラフで表したものである．この図を見ると，ピンポンの通信時間はおおよそ通信回数と比例しているのが確認できる．グラフの傾きが大きいほど，単位通信あたりのオーバーヘッドが大きいことを意味する．

まず，MPI とポーリング受信の Phoenix で実行結果を比較する．実行時間はいずれも MPI のほうが短く，Phoenix では通信時間に関して 7，8 倍ほどの大きな性能低下が見られた．これは，Phoenix では SSH を用いた通信を行っており，その影響で MPI よりもオーバーヘッドが大きく発生したと思われる．また，Phoenix のスレッドセーフの設計によるオーバーヘッドも原因の 1 つと考えられる．

次に，ポーリング受信の Phoenix とブロッキング受信の Phoenix で実行結果を比較する．ランタイム

Number	0	100	500	1000	5000
MPI (polling)	1.56	2.11 (0.55)	4.59 (3.03)	7.12 (5.56)	28.59 (27.03)
Phoenix (polling)	7.27	11.02 (3.75)	29.04 (21.77)	51.29 (44.02)	228.22 (220.95)
Phoenix (blocking)	7.17	7.59 (0.42)	9.51 (2.34)	11.76 (4.59)	30.39 (23.22)

表 2 ピンポンプログラムの実行時間
Table 2 Execution Time of Ping-Pong Program

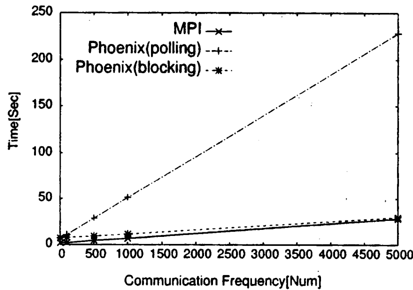

図 6 ピンポンプログラムの実行時間
Fig. 6 Execution Time of Ping-Pong Program

の初期化処理・終了処理にかかるオーバーヘッドは双方ほとんど変わらないことが確認できる．しかし，通信時間を比較すると 9 倍ほどの著しい性能差が確認できた．この差は，ポーリング間隔によるオーバーヘッドと考えられる．

最後に，MPI とブロッキング受信の Phoenix で実行結果を比較する．ランタイムの初期化・終了処理のオーバーヘッドの違いのため，実行時間はいずれも MPI のほうが短いが，通信時間を見ると，Phoenix のほうがわずかに性能が向上しているのがわかる．これは，使用している通信プロトコルでは MPI のほうが高速ではあるが，MPI でのポーリング受信で発生するオーバーヘッドのほうが通信時間に与える影響が大きかったと考えられる．

以上により，通信ライブラリを変更したことによる実行時間の影響を考察すると，ランタイムの初期化処理・終了処理にはある程度のオーバーヘッドが発生してしまうが，通信性能に関しては実装前よりも向上していることが分かった．

7. おわりに

現在の MegaScript 処理系の問題点を指摘し，その解決策として既存のマスター／ワーカーモデルを改良した階層型モデルの設計を行い，それに付随する問題点とその解決策の提案を行った．そして，これら提案手法を MegaScript 処理系に実装するにあたり，Phoenix ライブラリによる通信機構をランタイム上に実装し，実行時間比較による通信性能の評価を行った．

今後は，実際に提案手法に基づく階層型の通信機構を実装し，階層型のスケジューラと連携して，階層構造に対応した MegaScript 処理系の実装を行う．

謝辞 本研究は，科学技術振興事業団・戦略的基礎研究「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) 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) 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, 情処研報 2003-HPC-95, pp. 119–124 (2003).
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 向けランタイムシステム, 情処研報 2004-HPC-99, pp. 7–12 (2004).
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).
5) S.Pacheco, P.: *Parallel Programming with MPI*, Morgan Kaufmann Publishers (1997).
6) Kenjiro, T., Toshio, E., Kenji, K. and Akinori, Y.: Phoenix : a Parallel Programming Model for Accommodating Dynamically Joining/Leaving Resources, *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming(PPoPP 2003)*.