Vol.2012-HPC-135 No.1
2012/8/1

階層型MegaScriptランタイムの通信方式

仲 貴幸[1]　松本 真樹[1]　大野 和彦[1]　佐々木 敬泰[1]　近藤 利夫[1]

概要：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，独立した逐次・並列の外部プログラムをタスクとして扱い，複数のタスクを並行・並列に実行する．タスクの標準入出力をストリームと呼ばれる仮想的な通信路と接続してタスク間の連携を実現している．従来の MegaScript ランタイムは集中管理型のマスター・スレイブモデルによる実装が行われていた．しかし計算ホスト数の増加につれて，マスターホスト 1 台にスケジューリング負荷や計算ホストの制御負荷が集中しスケーラビリティに欠けてしまうことが問題となっていた．そこで，マスターホストの負荷を分散させるため階層型動作モデルによるランタイムの提案を行い，実装を進めている．しかし，従来のタスク間通信方式を階層化した MegaScript ランタイムにそのまま適用すると効率的な通信を行うことが出来ない．そこで，本論文では階層化に対応した効率的な通信方式の提案を行う．

キーワード：分散コンピューティング，並列コンピューティング，コンピュータクラスタ，通信方式

Communication Scheme for Multi-Layered MegaScript Runtime

Takayuki Naka[1]　Masaki Matsumoto[1]　Kazuhiko Ohno[1]　Takahiro Sasaki[1]　Toshio Kondo[1]

Abstract: We are developing a parallel programming language MegaScript for mega-scale computing. MegaScript regards independent sequential/parallel programs as tasks and executes them in parallel. The current implementation is based on a centralized control architecture which consists of a master host and slave hosts. The former controls and manages the latter, and the latter execute tasks. However, the architecture lacks scalability because the master host will be the bottleneck for the large number of slave hosts. Although we have proposed and developing a new implementation based on a hierarchical model, which reduces the load of the master host, its communication is still inefficient. Therefore, we propose an efficient communication scheme for the hierarchical model.

Keywords: Distributed Computing，Parallel Computing，Computer Cluster，Communication Scheme

1. はじめに

近年，大規模計算の需要が増大する一方で，単一プロセッサでの性能向上は頭打ちになりつつあり，並列処理への期待が高まっている．特にコストパフォーマンスやスケーラビリティの面で大きな利点があるため，PC クラスタの利用に注目が集まっている．

大規模な並列アプリケーションを作成する手法の一つとして，複数の独立したプログラムをタスクとして組み合わせるワークフローが使われている [1], [2], [3]．ワークフローは，各タスクの独立性が高いため既存のソフトウェアを再利用しやすいことやタスク間のデータフローを非循環有向グラフ (DAG) の形で記述できるなどの利点を持つ．また，ワークフローでは一般にタスクやタスク間通信の粒度が大きく，大規模な計算資源を安価に供給できる広域分散環境との相性がよい．このため，天文学・物理学などの科学技術の分野において，大規模な問題を解く手段として盛んに利用されている [4], [5]．

そこで我々は大規模ワークフローを容易に記述できるタスク並列スクリプト言語 MegaScript の開発を進めている [6], [7], [8], [9], [10]．従来の MegaScript ランタイムは

[1] 三重大学大学院 工学研究科
Mie University

集中管理型のマスター・スレイブモデルによる実装が行われていた．しかし実行環境が大規模化し計算ホスト数が増加するにつれて，マスターホスト 1 台にスケジューリング負荷や計算ホストの制御負荷が集中してしまい，スケーラビリティに欠ける問題があった．そこで，マスターホストの負荷を分散させるためにホストの管理・制御やスケジューリングを行うサブマスターホストを新たに導入し，階層的にホストの管理を行う階層型動作モデルによるランタイムの提案を行い実装を進めている [11]．しかし従来のタスク間通信の通信方式を階層化した MegaScript ランタイムにそのまま適用すると効率的な通信を行うことが出来ず，性能が大幅に低下する．そこで，本論文では階層化に対応した 3 つの通信効率化手法を挙げ，一対一通信や一対多通信，多対一通信といった通信パターン毎に最適な手法を組み合わせた通信方式の提案を行う．

以下，2 章で MegaScript の概要を説明し，3 章で MegaScript ランタイムについて詳細を述べる．4 章で階層化に対応した効率的な通信方式について提案し，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

MegaScript はワークフローモデルに基づく粗粒度並列言語である．ユーザは処理の主要部を別プログラムとして用意し，MegaScript プログラム中でタスクとして生成・実行する．各タスクは PC クラスタなどの複数の計算ホスト上で並行・並列に実行され，ストリームと呼ばれる仮想的な通信路を介してタスク間の連携を実現している．MegaScript プログラムでは，タスクやストリームから構成されるワークフローやタスクの実行に必要な情報等を記述する．MegaScript ランタイムはこれらの情報を解析し，スケジューリング結果に従って各タスクを指定された計算ホストで実行する．

ここで未知の病原菌の DNA 配列群を既知の病原菌データと比較して特定を行い，またそれらの相互比較を行うワークフロー例 (**図 1**) を示す．まず未知の DNA 配列群に対し *BLAST* [12] や *FASTA* [13] を使用し既知のシーケンスデータベースとの相同性検索を行う．その結果を利用して *ClustalW* [14] においてマルチプルアライメントを行い，樹形図を得る (図 1 の太枠内)．これを複数のシーケンスデータベースで行い，シーケンスデータベースごとに得られた樹形図を *ClustalW Panel* [14] で読み込み相互比較を行う．

2.2 タスクとストリーム

タスクは MegaScript の並列実行単位である．タスクは独立性の高い逐次プログラムや並列プログラムのような外部プログラムであり，それらのプログラムの内部処理に

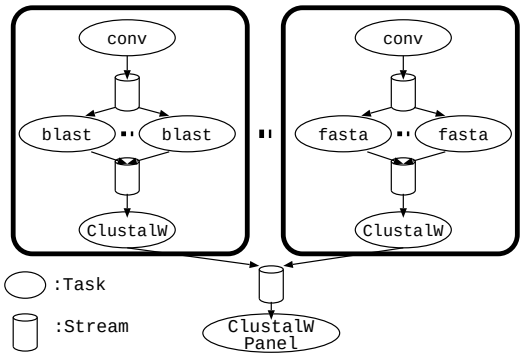

図 1 ワークフローの例
Fig. 1 Workflow Example

MegaScript は関与しない．

ストリームはあるタスクの標準出力の内容を他のタスクの標準入力に流すための仮想的な通信路である．1 つのストリームの入出力端にそれぞれ複数のタスクを接続することで，一対多，多対多などのタスク間通信を実現する．入力端に接続した入力側タスク群の出力が非決定的にマージされ，出力端に接続した出力側タスク群にマルチキャストされる (**図 2**)．現実装では標準入出力のテキストストリームのみ扱い，行単位でメッセージとしている．

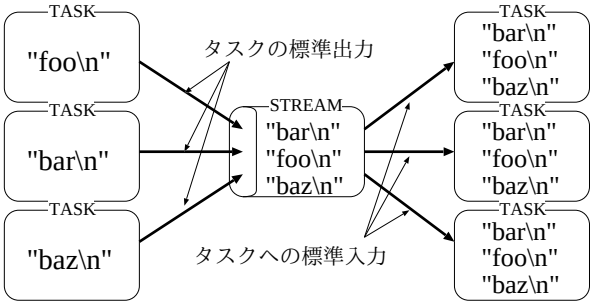

図 2 ストリームの振る舞い
Fig. 2 Behavior of Stream

2.3 動作モデル

従来の MegaScript ランタイムでは，動作モデルとして計算ホストの管理・制御に集中管理型のマスター・スレイブモデルを採用していた．しかし実行環境が大規模化し計算ホスト数が増加するに伴い，マスターホスト 1 台にスケジューリング負荷や計算ホストの制御負荷が集中しスケーラビリティに欠けてしまう問題があった．そこで我々はマスターホストの負荷を分散させ，より高いスケーラビリティを実現するためにホストの管理・制御やスケジューリングを行うサブマスターホストを新たに導入し，階層型動作モデルによるランタイムの提案を行い実装を進めている．

2.4 スケジューラ

大規模ワークフローを構築するアプリケーションを実行

する際，タスクの各ホストへの配置によって実行性能は大きく変化する．MegaScript ランタイムではタスクスケジューラを用意しており，スケジューラはワークフローの構造を解析し，各タスクの実行順や実行ホストを決定する．

階層型動作モデルでは，マスターホスト及びサブマスターホスト上でスケジューリングが行われ，スケジューリング結果に基づき各タスクやストリームをそれぞれ直下のホストに割り当てる．以降階層的にスケジューリングを繰り返し行い，スレイブホストまでタスクやストリームを割り当てる．

3. MegaScript ランタイム

MegaScript ランタイムは使用するホスト上にそれぞれランタイムプロセスを起動する (図 **3**(a))．ランタイムプロセスはそのホスト上で実行されるタスクプロセスを子プロセスとして生成し (図 3(b))，またストリームの管理，タスクスケジューリングや別ホスト上のランタイムプロセスとの通信を行う．異なるホスト上のタスク間通信は各ホスト上に起動したランタイムプロセスを介して行われる．

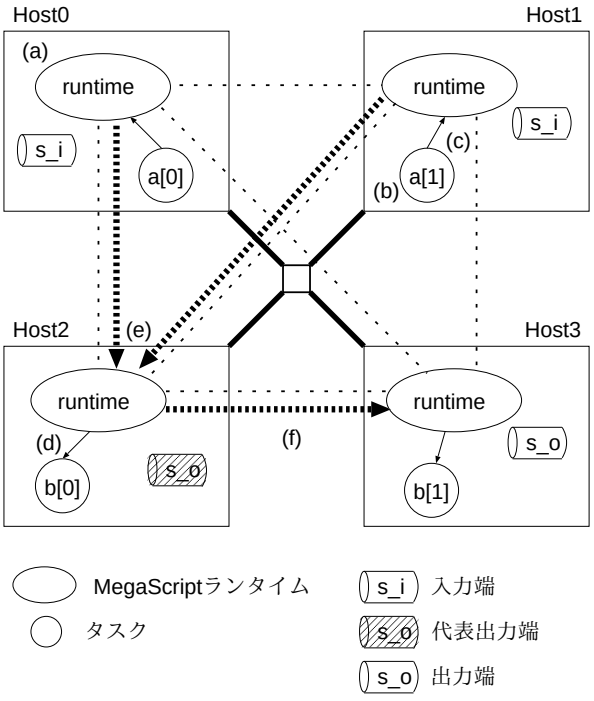

図 3　MegaScript ランタイムの実装の概要
Fig. 3　Implementation of MegaScript Runtime

3.1 タスク間接続とワークフロー

MegaScript ではタスクとストリームから構成されるワークフローを使用している．ストリーム情報からタスクとの接続関係が得られ，タスク間通信の種類を判別することが可能である．またワークフローにおいて，パラメータスイープのように同じプログラムを異なる条件で多数実行する場合，それらのタスクはワークフロー上で同一ストリームに接続されており，似た性質を持つことが推測される．このことから，同一ストリームに接続されたタスクの性質，すなわち総通信回数や総通信量は類似したものになると考えられる．

3.2 タスク間通信

ランタイムにおけるタスク間通信について説明する (図 3)．用いたワークフローは図 **4** で，タスク a[i] を入力側タスク，タスク b[i] を出力側タスクとしており，またそれらをストリームで接続している．タスクプロセス生成時，ランタイムプロセスはタスクプロセスと双方向パイプを用いて接続する．入力側タスクを実行するとき，ランタイムプロセスは図 3 の (c) のようにタスクの標準出力をパイプを通じて取得する．逆に出力側タスクを実行するときは， 図 3 の (d) のようにホスト間通信により受け取ったメッセージをランタイムプロセスからパイプを通じてタスクに与える．

次にストリームの実装の概要について説明する．MegaScript ランタイムではプログラム中で定義された 1 つのストリームに対し，入力端と出力端をそれぞれ生成する．入力端は入力側タスクを実行するホスト毎に 1 つずつ配置され (図 3 の s_i)，出力端は出力側タスクを実行するホスト毎に 1 つずつ配置される (図 3 の s_o)．出力端のうち 1 つはランタイム内で代表出力端として扱われる．

ストリームを流れるメッセージは図 3 の (e) のように一度代表出力端に集められメッセージのマージが行われる．代表出力端にはそれ以外の出力端の配置先ホストの情報がランタイムより与えられており，それをもとに図 3 の (f) のようにメッセージの転送を行い，タスク間通信を実現している．

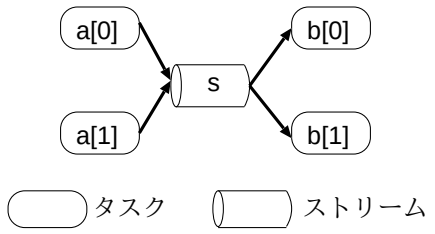

図 4　タスク間通信を行うワークフロー例
Fig. 4　Inter-task Communication Example

3.3 動作モデルとネットワーク

現在実装しているランタイムでは，最初にランタイムを起動したホスト上でマスタープロセスが起動し，そのプロセスが利用可能な直下のホスト上にサブマスタープロセスあるいはスレイブプロセスを起動する．サブマスタープロセスは利用可能な直下のホスト上にサブマスタープロセス

あるいはスレイブプロセスを起動する．図 5 を例に説明すると，最初にホスト A 上でランタイムを起動したとするとマスタープロセスが立ち上がる．マスタープロセスが起動したホスト A は次に，直下に管理するホストが存在するためホスト B 群上にはサブマスタープロセス，タスクを実行する計算ホストとして使用するためホスト C 群上にはスレイブプロセスを起動させる．同様にサブマスタープロセスが起動したホスト B 群はホスト D 群上にサブマスタープロセス，ホスト E 群上にスレイブプロセスをそれぞれ起動させる．また，サブマスターホスト単位で使用するホストをグループ化 (以後ホストグループと呼ぶ) しており，ホストグループを階層的に管理することで階層型動作モデルを実現している．

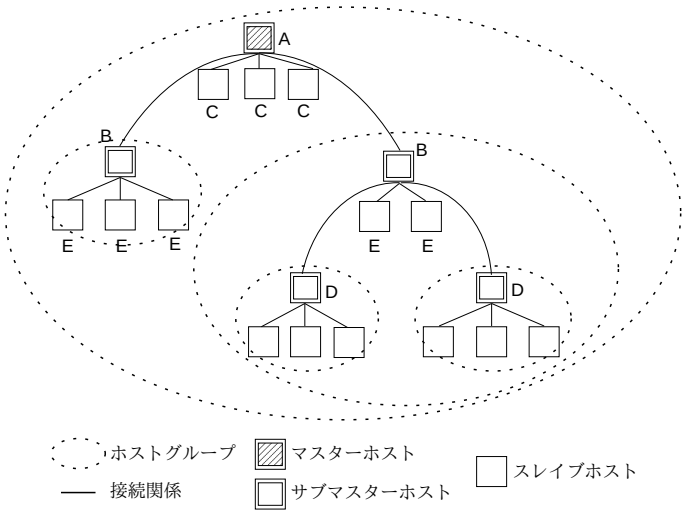

図 5　階層型 MegaScript ランタイムの動作モデル
Fig. 5　Hierarchical Model of MegaScript Runtime

MegaScript ランタイムは論理レベルでモデル構築を行っているため，論理ネットワークと物理ネットワークには差異が存在する．例えば，図 6 においてホスト H1 から H2 へ通信が発生するとき，物理レベルではネットワーク n1，n2，n3 とスイッチ s1，s2 を介して通信が行われる．しかし，論理レベルで同様の通信は仮想的な通信路 r1 を介して行われる．また，ファイアウォールや NAT などで直接通信が不可能なときやホストを経由して通信を行う方が効率的な通信が可能な場合が存在する．そのため，効率的なホスト間通信を行うには論理レベルでのルーティングが必要となる．

4. 提案手法

MegaScript におけるタスク間通信は一対一，一対多，多対一，多対多の 4 種類に大きく分類できる．タスク間通信では，3.2 節で述べたように入力側タスクの出力するメッセージは出力端の代表である代表出力端に一度集められマージされる．その後，代表出力端を持つホストからそれ以外の出力端を持つホスト群にメッセージを転送している．このことからタスク間通信をホスト単位の視点で考えると，多対多のタスク間通信は多対一と一対多のホスト間通信の組み合わせに置き換えることができる．したがって，タスク間通信は一対一，一対多，多対一の 3 種類のホスト間通信で構成されているとみなせる．

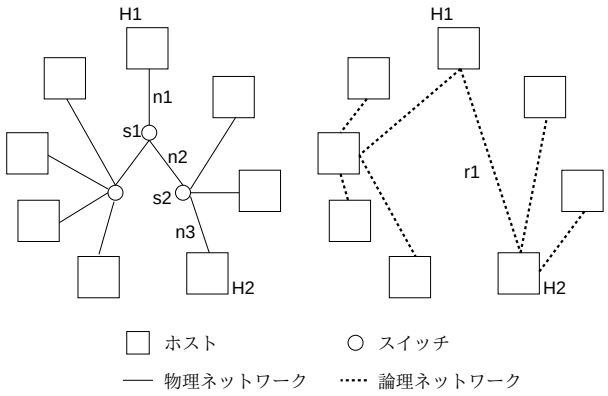

図 6　物理ネットワーク (左) と論理ネットワーク (右)
Fig. 6　Physical Network and Logical Network

4.1 通信効率化手法

4.1.1 メッセージのマルチキャスト

マルチキャストは一対多のホスト間通信，すなわち代表出力端を持つホストから出力端を持つホスト群へメッセージを転送するときに適用する．代表出力端を持つホストから出力端を持つ各々のホストに一つずつメッセージ通信する場合，出力端を持つホスト数が多いと転送側ホストに負荷が集中してしまう．そこで，ホストグループ内に出力端を持つホストが一台以上あるときはそのホストグループの代表ホストであるサブマスターホストにメッセージを転送する．メッセージを受け取ったサブマスターホストは，そのホストグループ内の出力端を持つすべてのホストへメッセージの転送を行う (図 7)．そのため，ホストグループ内に配置された出力端が多いほど異なるホストグループ間の通信回数やメッセージ量が削減でき，通信の効率化が見込める．

しかし，マルチキャストは本来必要のないホストを経由するため直接通信に比べ通信回数が増大している．そのため，出力端の配置やネットワーク性能によっては直接通信するほうが高効率となることがある．効果的な性能向上を目指すには，状況に応じて転送手法を切り替える必要がある．手法の切り替えにはタスクやストリームの配置情報やホスト間のネットワーク性能が利用できると考えられる．

4.1.2 ルーティング選択

ランタイムの階層化によって大規模なホスト群を管理することが可能となったが，ホスト数の増加に伴いホスト間通信の際に経由しなければならないホスト数もまた増大している．そのため，通信対象のホストと効率的に通信を行

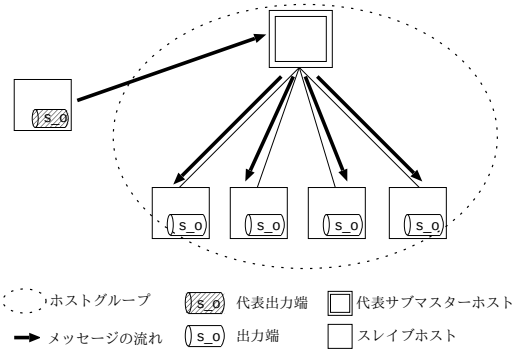

図 7　マルチキャスト手法
Fig. 7　Scheme of Multi Cast

うためには経由ホストの選択が重要となる．ルーティング選択手法はホスト間通信を行うとき，ランタイム全体に対して可能な限り通信回数や通信量を削減し，効率的なメッセージ通信を行うことができるようにメッセージ転送先を決定する手法である．

ルーティング選択には大きく分類して以下の４パターンが存在する (**図 8**).

パターン１ 自ホストから対象ホストに直接通信

パターン２ 自ホストから自ホストの先祖にあたるサブマスターホスト群 (例えばホスト H にとってはホスト F，E) やマスターホストを経由して対象ホストに通信

パターン３ 自ホストから対象ホストの先祖にあたるサブマスターホスト群やマスターホストを経由して対象ホストに通信

パターン４ 自ホストから自ホストの先祖にあたるサブマスターホスト群やマスターホスト及び対象ホストの先祖にあたるサブマスターホスト群やマスターホストを経由して対象ホストに通信

ルーティング選択にあたり，タスクとストリーム間の接続の種類やホスト間の直接通信の可否，通信性能などの静的情報をパラメータとして用いる．また，一定期間内に発生した通信の回数とサイズなどの動的情報もパラメータとして扱う．中でも特に静的情報がルーティング選択に大きな影響を与える．

また，ホスト間に通信を阻害するファイアウォールなどが存在しないとき経由ホストを省略することが出来る．**図 9** において，ホスト A から J に通信が発生した場合を例に説明する．ホスト E にファイアウォールが存在し，ホスト E より下位のホストへ外部から直接通信できない場合，ホスト A から J への直接通信は不可能である．このとき通信効率を考慮しない最も単純なルーティングは ACDEGJ となる．しかしホスト A から J に直接通信することは出来ないが，ホスト A から E へ直接通信は可能である．ホスト E から J も同様に直接通信が可能なため，通信効率を考慮したルーティングは AEJ となり，ホスト C，D，G との通信を省略することが出来る．このように経由ホスト数を

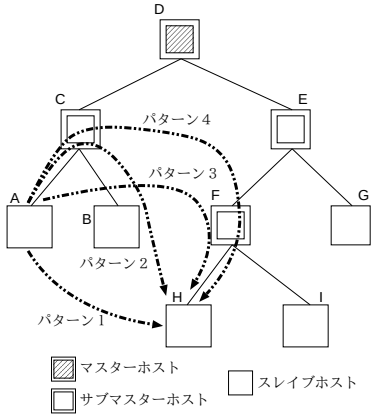

図 8　ルーティングパターンの例
Fig. 8　Routing Example

削減することによって通信の効率化が期待できる．

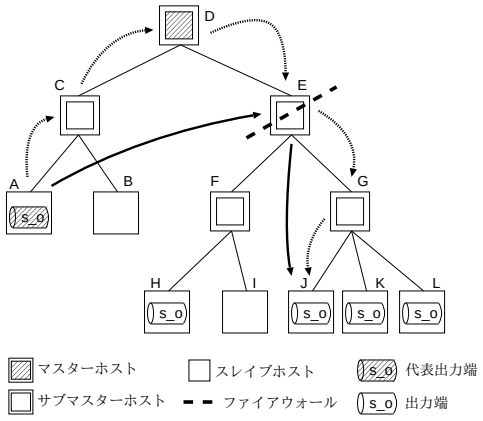

図 9　ルーティングにおける経由ホストの省略
Fig. 9　Omitting Transmitting Hosts

4.1.3　通信バッファリング

転送するメッセージのサイズが小さいとき，そのまま転送するのではなくメッセージを貯めておいて，ある一定のメッセージサイズになるようにメッセージのバッファリングを行い一度に転送する．通信バッファリングを行うことによって，ホスト間で発生する通信回数を削減することが可能である．

例えば，極めて小さいメッセージ通信が大量に発生するタスクが存在するとき，通信バッファリング手法を適用することによって，ホスト間の通信回数を大幅に削減することができる．また，異なるホストグループ間で通信するときは WAN などの低速なネットワークを経由するため，通信回数を削減することで通信の効率化が見込める．また，バッファリングを行うことで個々のメッセージに必要なヘッダを一つにまとめることが可能なためメッセージの実データの割合が高まり，ランタイムから見たバンド幅は向上する．しかし，レイテンシは低下し後続タスクの実行が遅れるという問題がある．そのため，バンド幅とレイテンシを両立可能なバッファリングサイズを決定する必要が

ある．

バッファリングサイズについて述べる．ここで，バッファリングサイズが 0 のときバッファリングは行われず，無限大のときすべてのメッセージがバッファリングされる．このとき通信バッファリングはホスト間通信の種類に依存せず適用することができる．バッファリングサイズはルーティング選択で決定した経由ホスト数，ホスト間のネットワーク情報や一定期間内に発生した通信の回数とサイズを総合的に評価し決定する．

4.2 通信効率化手法の適用

タスクとストリーム間の接続関係やホスト間の直接通信の可否などのネットワーク情報をもとに 4.1 節で述べた通信効率化手法をホスト間通信のパターンに合わせて適用する．

4.2.1 一対一のホスト間通信

対象ホスト間の直接通信が可能な場合，ルーティングはパターン 1 を選択する．これは一対一のホスト間通信において直接通信が経由ホストが存在せず最も効率的であるからである．

対象ホストとの直接通信が不可能な場合，パターン 1 の次に経由ホスト数が少ないパターン 2 ・ 3 を選択する．また，自ホストとマスターホスト間及びマスターホストと対象ホスト間にファイアウォールなどが存在するときパターン 2 ・ 3 は通信ができない可能性がある．そのときはパターン 4 を用いて通信を行う．

4.2.2 一対多のホスト間通信

マルチキャストを効率的に利用することを考慮し，出力端を持つホストの所属するホストグループの代表であるサブマスターホストにメッセージ転送を行う．メッセージを受け取ったサブマスターホストは子孫に出力端を持つすべての直下のホストにメッセージをマルチキャストする．

一対多のホスト間通信におけるマルチキャストを図 9 を例に説明する．このとき，代表出力端を持つホスト A は入力側タスクのメッセージを受信していたとする．ホスト A は出力端を持つホスト H，J，K，L の所属するホストグループ群の代表であるサブマスターホスト E にメッセージを転送する．そしてメッセージを受け取ったホスト E は子孫に出力端を持つすべての直下のホスト F，G にメッセージをマルチキャストする．メッセージを受け取ったホスト F，G もまた同様に出力端を持つすべての下位ホスト H，J，K，L にメッセージをマルチキャストする．

4.2.3 多対一のホスト間通信

多対一のホスト間通信では直接通信の可否はルーティングに大きな影響を与えず，また，マルチキャストを利用することが出来ない．一方，通信バッファリングを有効に利用することは可能である．まず最初に，入力側タスクを持つホストは所属するホストグループの代表であるサブマスターホストにメッセージ転送を行う．メッセージを受け取ったサブマスターホストは通信バッファリングを行う．そして，代表出力端を持つホストに直接通信が可能であればそのまま通信を行いメッセージを転送する．直接通信が不可能であれば 4.1.2 項で述べたルーティングを行う．

また，3.1 節で述べた類似する性質を持つと考えられるタスクが存在する場合，過去に実行されたタスクの総通信回数や総通信サイズなどの実行情報をルーティング選択や通信のバッファリングに利用することができる．

5. おわりに

本論文では，現在実装中の階層型 MegaScript ランタイムにおけるタスク間通信の効率的な通信方式を提案し，各通信パターンに合わせた提案手法の適用について検討した．

今後の課題としては，タスクの総通信回数や総通信量を記述したメタ情報やプログラム実行中に得られるタスクの通信傾向などの動的情報を利用した通信方式を開発する．また MegaScript ランタイムに提案手法の実装を行い，大規模環境下において実アプリケーションで性能評価を行う必要がある．

参考文献

[1] K. Taura, T. Matsuzaki, M. Miwa, Y. Kamoshida, D. Yokoyama, N. Dun, T. Shibata, C. S. Jun and J. Tsujii.: *Design and Implementation of GXP Make – A Workflow System Based on Make. eScience*, IEEE International Conference on, 214-221, (2010).

[2] E. Deelman, G. Singh, M. Sua, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob and D. S. Katz.: *Pegasus: A framework for mapping complex scientific workflows onto distributed systems*, Scientific Programming. 219-237, (2005).

[3] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz and I. Foster.: *Swift: A language for distributed parallel scripting*, Parallel Computing. (2011).

[4] E. Deelman, D. Gannon, M. Shields, and I. Taylor.: *Workflows and e-science: An overview of workflow system features and capabilities*, Future Gener. Comput.Syst., 25:528-540, (2009).

[5] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau and J. Myers.: *Examining the challenges of scientific workflows*, Computer, 40:24-32, (2007).

[6] 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, 73-76, (May 2003).

[7] 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, In 情処研報 2004-HPC-99, pages 7-12, July, (2004).

[8] 湯山 紘史, 大塚 保紀, 西里 一史，大野 和彦, 中島 浩: タスク並列スクリプト言語 MegaScript によるタスクモデルの記述手法, 先進的計算基盤システムシンポジウム SACSIS2004, 135-136, (May 2004).

[9] 阪口 裕輔, 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レベル

機能拡張機構, 情報処理学会論文誌 コンピューティングシステム, Vol. 47 No. SIG 12(ACS 15), pp. 296-307, (September 2006).

[10] K. Ohno, A. Mita, M. Matsumoto, T. Sasaki, T. Kondo, H. Nakashima: *Efficient Implementation of Large-scale Workflows based on Array Contraction*, Proceedings of the 22th IASTED International Conference on Parallel and Distributed Computing and Systems –PDCS 2010–, pp. 153-162, (November 2010).

[11] 西川雄彦, 高木祐志, 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 MegaScript ランタイムの広域分散化, 先進的計算基盤システムシンポジウム SACSIS2005, 251–252, (2005).

[12] S. Altschul, W. Gish, W. Miller, E. Myers and D. Lipman: *Basic local alignment search tool*, *Journal of Molecular Biology*, Vol. 215, pp. 403–410, (1990).

[13] D. J. Lipman and W. R. Pearson: *Rapid and sensitive protein similarity searches.*, *Science*, Vol. 227, pp. 1435–1441, (1985).

[14] J. D. Thompson, D. G. Higgins and T. J. Gibson: *CLUSTAL W:improving the sensitivity of progressive multiple sequence alignment through sequence weighting*, position-specific gap penalties and weight matrix choice, *Nucleic Acids Reseach*, Vol. 22, pp. 4673–4680, (1994).

[15] 松本 真樹，片野 聡，佐々木 敬泰，大野 和彦，近藤 利夫，中島 浩：非均質環境における適応型スケジューリング手法の提案と評価，電子情報通信学会論文誌，Vol.J93-D, No.6, pp.693–704, (2010).