特別研究報告

題目

λ コンピューティング環境構築のための Globus Toolkit を用いた MPI ライブラリの実装と評価

指導教官
村田 正幸 教授

報告者
井本 舞

平成 17 年 2 月 17 日

大阪大学 基礎工学部 情報科学科

平成 16 年度 特別研究報告

λコンピューティング環境構築のための Globus Toolkit を用いた MPI ライブラリの実装

井本 舞

内容梗概

近年，ネットワーク接続された複数の計算機を用いて大規模な科学技術計算を行うグリッド計算に関する研究開発が盛んに行われている．グリッド計算環境で分散計算を実行する場合，現状ではノード計算機間の通信には TCP/IP が用いられている．しかし，TCP/IP を用いたパケットを単位としたデータ交換では，パケット損失やパケット処理に要するオーバーヘッドの影響が大きく，大規模計算で必要な大量データの共有や交換を行うには十分な性能を得ることは難しい．

そこで各ノード計算機に光ファイバを直結し，さらに近年研究開発が活発に行われている WDM (Wavelength Division Multiplexing) 技術を適用して波長パスをノード計算機間の高速な通信チャネルとして活用するλコンピューティング環境をわれわれの研究グループでは提案している．波長パスを利用することにより，ユーザに対して高速かつ高信頼な通信パイプを提供することが可能になる．さらに，波長パスを用いて，例えば仮想的にノード計算機をリング状に接続することによって，分散計算を行うノード計算機間でのデータ交換，共有が可能になる．

本報告では，WDM 技術を利用してグリッド計算を高速に行うλコンピューティング環境の実現形態として，WDM 技術に基づくフォトニックネットワークである AWG–STAR システムを用いてグリッド計算環境を構築した．すなわち，グリッド計算のデファクト標準となっている Globus Toolkit をミドルウェアとして導入できるように，分散計算のための MPI (Message-Passing Interface) ライブラリである MPICH–G2 を AWG–STAR システム上で動作可能とした．そのために，AWG–STAR の共有メモリシステムを利用できるメッセージパッシング手法を提案し，実装している．

さらに，Globus Toolkit に基づいた MPI アプリケーションを実行し，構築したシステムが正常に動作することを確認し，また，実現システムにおける分散計算の性能を評価した．その結果，AWG–STAR を用いた共有メモリ上のデータ交換の性能は，共有メモリへのアクセス回数，データサイズに大きく影響されることが明らかになった．これは，グリッド計算をより高速に実行するλコンピューティング環境の設計に指針を与えるものである．

主な用語

λコンピューティング，分散計算，AWG–STAR，Globus Toolkit，共有メモリアクセス手法

目 次

図 目 次

表 目 次

1 はじめに

近年，グリッドコンピューティング環境の構築に対する期待が高まり，活発に研究開発が行われている．グリッドコンピューティングは，広域に分散した計算機やストレージ，さらには観測機器，さまざまなデバイスなど，多くの資源を，ネットワークを利用して統合的に接続し，ひとつの大規模な仮想計算機として機能させるインフラストラクチャを構成する技術である．このグリッドコンピューティングによって構築された仮想計算機を用いることにより，単体の計算機では解くことが難しい大規模な科学技術計算を行う，高性能観測機器から発生する大量のデータを高速に処理する，大規模データをリアルタイムに計算しながら可視化を行う，などより高度な処理能力を得ることができると期待されている．

グリッドコンピューティングを実現するためには，地理的，組織的に分散したさまざまなコンピューティングに関する資源を動的に共有し，協調動作させることが重要な課題となっている．Global Grid Forum [1] という標準化団体においては，アプリケーションや運用まで視野にいれたグリッドコンピューティングに関する関連技術の標準化を行っている．また，Globus Alliance によって，グリッドコンピューティング環境を構築するための Globus Toolkit と呼ばれるミドルウェアが開発されている [2]．特に，Globus Toolkit は世界中で広く使われ，デファクト標準になっている．

一方，グリッドコンピューティングを実現するために重要となる下位層の高速ネットワーク技術として，現在光伝送技術を用いた研究が活発に進められている．特に高速ネットワークを実現する手段として光の波長を用いて多重化を行う WDM (Wavelength Division Multiplexing) 技術が研究の中心であり，1000 波を利用できる新たな WDM 技術の研究も進められている [3]．また WDM を利用してインターネットの高速化を実現する IP over WDM ネットワークの研究もさかんに行われている．しかしながら，現状のネットワークにおいてはルーティングを行う際に，光信号を電気信号に変換し，もう一度光信号にかえる処理を行っており，光の高速性を損ねてしまう．そのため，WDM 技術以外のさまざまな光技術を下位レイヤの通信技術とする GMPLS (Generalized Multi-Protocol Label Switching) [4] と呼ばれるインターネットルーティング技術や，フォトニックネットワークの真の IP 化を実現するために光領域でパケット交換を行うフォトニックパケットスイッチの研究も行われている [5-11]．しかしながら，これらの技術は情報を扱う細粒度として IP パケットを扱い，ネットワーク上でそのパケットをいかに高速に運ぶかを研究開発の目標としている．そのため，このようなパケット交換技術に基づいたアーキテクチャをとる限り，個々のコネクションに対する高品質通信の実現は困難である．

光ネットワークを用いた高速な分散環境システムを目標とするミドルウェアとして OptIPuter がある [12]．OptIPuter は地球規模での光ネットワークによるグリッド環境を構築す

るために現在取り組まれているプロジェクトである．OptIPuter では，ノード計算機までが光ファイバで接続され，各ノード計算機のアプリケーションレベルでネットワーク資源の発見，配置，調整を行い，動的にノード計算機間の専用光パスを設定し，高速転送プロトコルを利用して巨大なデータを送信することを可能としている．また，様々な資源管理を行うためのミドルウェアとして QUANTA が開発されている．しかしながら，OptIPuter も現在のインターネット技術をベースとしており，情報転送の粒度としてパケットを用いるために，先にあげたようなパケット処理に関する問題が生じる．

そこで，我々の研究グループでは各計算機を接続している光ファイバを専用の通信路として利用し，WDM 技術を用いた高速な通信チャネルとして活用する λ コンピューティング環境を提案している [13-15]．λ コンピューティング環境においては，グリッド計算の通信を従来の TCP/IP を用いて実現するのではなく，あらかじめ設定した波長パスを利用することにより高速かつ高信頼な通信を実現することができる．すなわち，グリッドシステムを構成するフォトニックネットワーク上にデータ通信用の仮想チャネルをメッシュ状に張ることにより，高速チャネル上での分散計算が可能となる．また，λ コンピューティング環境を構成するネットワーク機器および計算機群を結ぶ仮想リングを想定し，仮想リング上の波長を高速な共有メモリとして利用することも可能である．その結果，広域分散システムにおける共有メモリと通信チャネルの区別の必要がなくなり，コンピュータ間の高速なデータ交換が可能になる（図 1）．

本報告では，高速かつ高信頼な通信を可能とする λ コンピューティング環境に，グリッドコンピューティング環境を構築するための世界標準ミドルウェアとなっている Globus Toolkit を導入することによって，高速な分散計算環境を提供することを目指す．すなわち，Globus Toolkit により構築されたグリッド計算環境の下位層に λ コンピューティング環境におけるフォトニック技術を利用することにより，分散計算を行いたいユーザは，従来の利用法を変えることなく高速なコンピューティング環境を利用することが可能となる．具体的には，ユーザは，Globus Toolkit を利用して MPI (Message Passing Interface) を用いた分散計算アプリケーションのジョブを投入し，分散計算が λ コンピューティング環境上で実行される．

また，本報告では，λ コンピューティング環境を構築するために，各ノード計算機上に存在する共有メモリを高速にアクセスする手法を実装する．実際には，日本電信電話株式会社フォトニクス研究所が開発している「情報共有ネットワークシステム (AWG–STAR)」を用いる [16-18]．この AWG–STAR システムは，AWG ルータと波長可変光源ををベースとした同的な波長ルーティングを利用し，複数端末ノード計算機の共有メモリを共有する，多対多マルチキャストシステムである．各ノード計算機は波長可変光源を通じて光ファイバにより AWG (Arrayed Waveguide Grating) と呼ばれるルータに接続され，物理的にはスタートポロジを，論理的にはリングトポロジを形成している．また，各ノード計算機は共有メモ

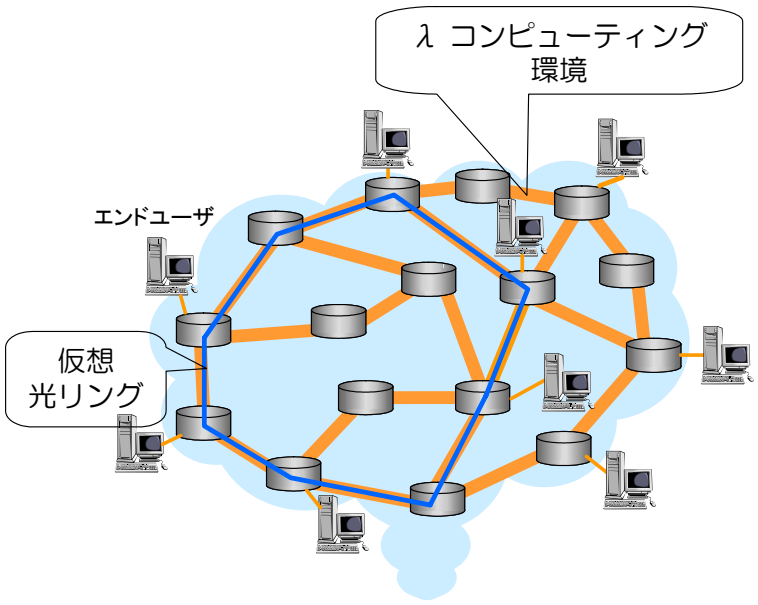

図 1: λコンピューティング環境

リボードを搭載しており，共有メモリボード上のデータは，AWG–STAR 上でリングネットワークを構成している全ノード計算機で同一のものになるよう設計されている．

本報告では，AWG–STAR システムを用いたλコンピューティング環境を構築し，Globus Toolkit によるミドルウェアを実装した上で，MPI による分散計算アプリケーションを動作させることによりλコンピューティング環境における分散計算性能を明らかにする．

以下，2 章ではグリッドコンピューティングの概要と MPI について説明する．3 章では本報告で用いた AWG–STAR システム，4 章において AWG–STAR システムにおける MPI ライブラリの実装方法について述べる．5 章では，構築したλコンピューティング環境において分散計算を行った場合の性能を評価する．最後に 6 章で本報告についてのまとめと今後の課題について述べる．

2　グリッドコンピューティング環境における分散計算手法

2.1　グリッドコンピューティング環境構築ミドルウェア Globus Toolkit

グリッドコンピューティングは，画像処理や遺伝子解析，地球環境シミュレーションなど 1 台の計算機では実用的な時間内で解を導出するのが難しい問題を解決するための手段として考案され，発展している．グリッドコンピューティング環境では，広域に分散した複数の計算機をネットワークで結び，CPU やデータストレージなどの計算機資源を共有し，あたかも 1 つの高性能計算機のように利用できるようにすることを目標としている．例えば，個々の計算機が高性能なものでなくとも，複数台の計算機を協調して動作させ，処理を行うことによって高性能計算機と同等の処理を行うことができるようになる．さらには，複数台のスーパーコンピュータを接続することにより，今までできなかった大規模な計算が可能になる．

グリッドコンピューティング環境において機種の異なる複数の計算機を接続し資源を共有するためには，各計算機間で統一されたグリッドコンピューティング環境構築ミドルウェアが必要となる．グリッドコンピューティング環境構築ミドルウェアとして，現在世界のデファクト標準となっているのが Globus Toolkit である [2, 19, 20]．Globus Toolkit は米国のグリッドプロジェクトである Globus Alliance が作成したグリッドコンピューティングのためのツールキットであり，グリッドコンピューティングに必要な機能の中で，リソース情報の収集，ジョブの管理，データの管理，そしてセキュリティと通信の技術を提供している．Globus Toolkit は主に UNIX OS の異機種混合環境で動かすことができ，個々の計算機の CPU，メモリ，ネットワーク，ファイルシステムなどのサービスの種類に依存しない均一のインターフェースを提供している．

本報告では λ コンピューティング環境上で分散計算を行うために必要なセキュリティとジョブ管理を Globus Toolkit に行わせる．本節では Globus Toolkit が提供するセキュリティとジョブ管理の概要を説明する．

2.1.1　Globus Toolkit におけるセキュリティ

Globus Toolkit は認証，認可，通信内容保護のための GSI (Grid Security Infrastructure) を提供している．GSI は Globus Toolkit におけるコンポーネントの基礎となる機能で，GSI のレイヤーの上にその他のコンポーネントが載ることにより，その他のコンポーネントのセキュリティを保証する．認証には X.509 証明書を用いた公開鍵暗号方式を採用しており，グリッドコンピューティング環境上の全てのユーザとサービスは証明書を利用して真偽を識別している．図 2 に示すように，ユーザクライアントとサーバは，それぞれ認証局 (CA;

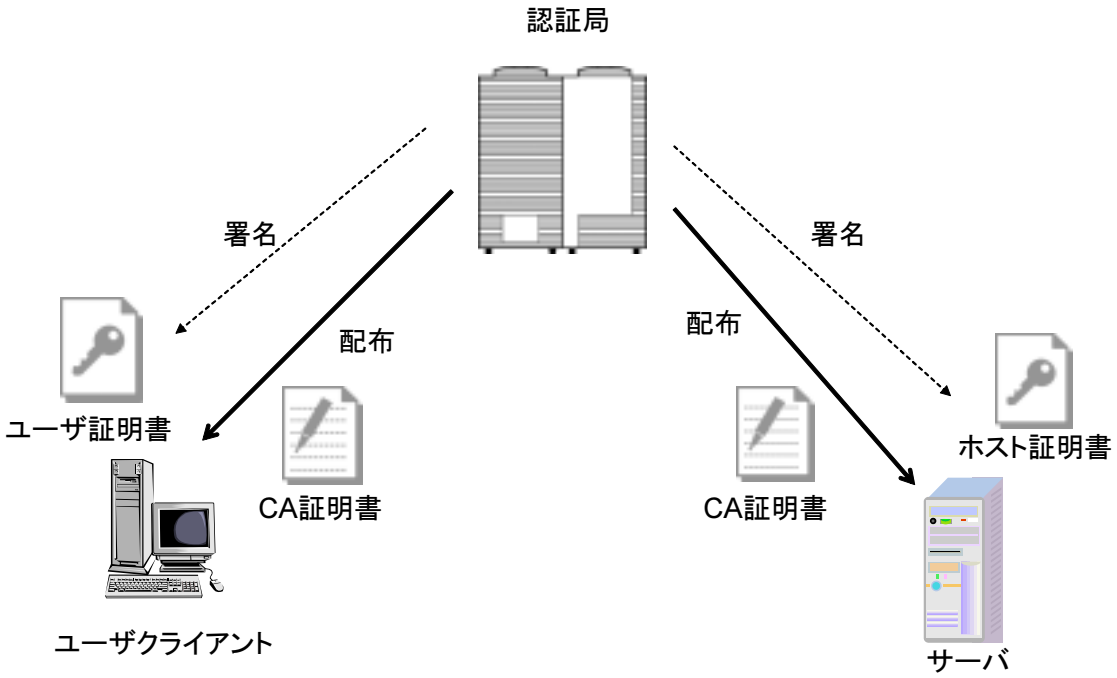

図 2: 認証局による認証

Certification Authority) から署名された証明書と CA 証明書を持っている．

ユーザがサービスを受けるためには，図 3 に示すように，認証局がユーザ証明書とユーザの秘密鍵から期限付きのプロキシ証明書を作成する．プロキシ証明書を作成するために，ユーザは秘密鍵のパスワードを入力する必要があり，プロキシ証明書を作成することがグリッドコンピューティング環境へログインすることに相当する．一度パスワードを入力すればプロキシ証明書の期限内は再度パスワード入力の必要性がなく，グリッド環境におけるシングルサインオンを実現している．ユーザのプロキシ証明書が作成され，ユーザがグリッド環境にログインすると，次にユーザとホストサーバとの相互認証を行う．相互認証に成功した後，ユーザはサービスを受けることができる．

GSI による認可には，各サーバにおけるマッピングファイルが用いられている．マッピングファイルには，ユーザ名とそのユーザに対する権限が記されており，各サーバやユーザごとに異なった権限を付加することができる．

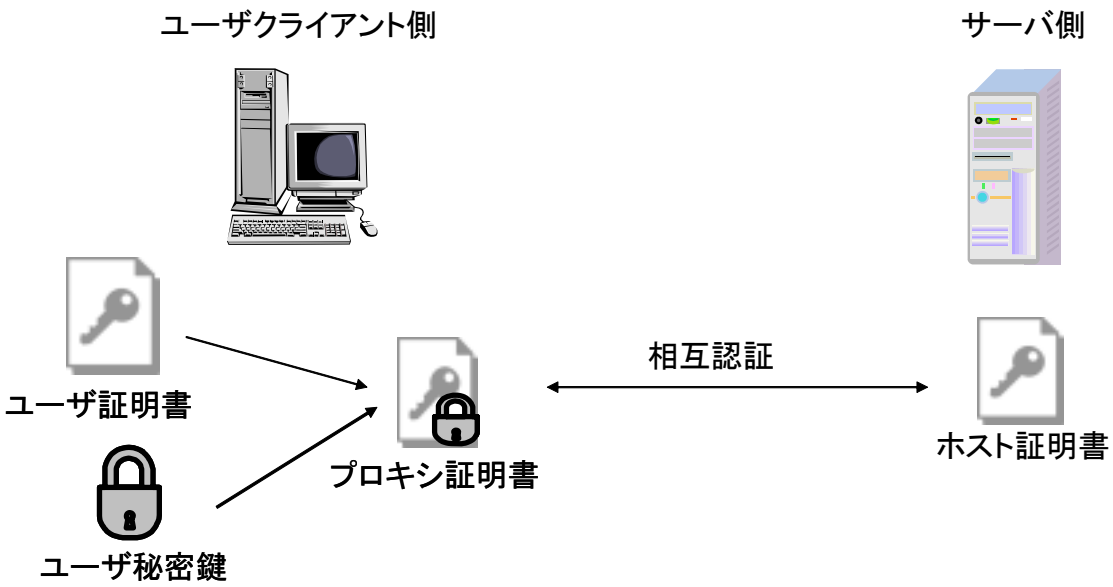

図 3: 証明書による相互認証

2.1.2 Globus Toolkit におけるリソースとジョブ管理

Globus Toolkit はリソースとジョブ管理のために GRAM (Globus Resource Allocation Manager) を提供している．GRAM はジョブを実行するためのリクエストとリモートシステムのリソースを使用するための標準的なインターフェースを提供している．GRAM のアーキテクチャにおいては分散計算アプリケーションで使用される Gatekeeper と Job Manager が主要なコンポーネントである．Gatekeeper はユーザクライアントとなる計算機からのジョブ要求を待ち受けるサーバであり，Job Manager は Gatekeeper によって生成され，実際の資源管理方法に従ってジョブを生成するプロセスである．

ユーザがユーザクライアントを通じてジョブ実行命令を出すと，ユーザクライアントのローカルマシンにおいて GRAM クライアントが生成される．GRAM クライアントはジョブ実行ノードへのリクエストを RSL (Resource Specification Language) という Globus Toolkit 特有の言語を用いてテキストファイルに出力し，ジョブ実行ノードの GRAM サーバに対して要求を出す．

GRAM サーバでは Gatekeeper が GRAM クライアントからのアクセスを待っており，アクセスがあるとユーザクライアントとユーザの認証を行う．認証に成功すると Gatekeeper はジョブ要求である RSL ファイルを受け取り，Job Manager を生成する．Job Manager は RSL ファイルに基づいてプロセスを生成し，ジョブを実行する．

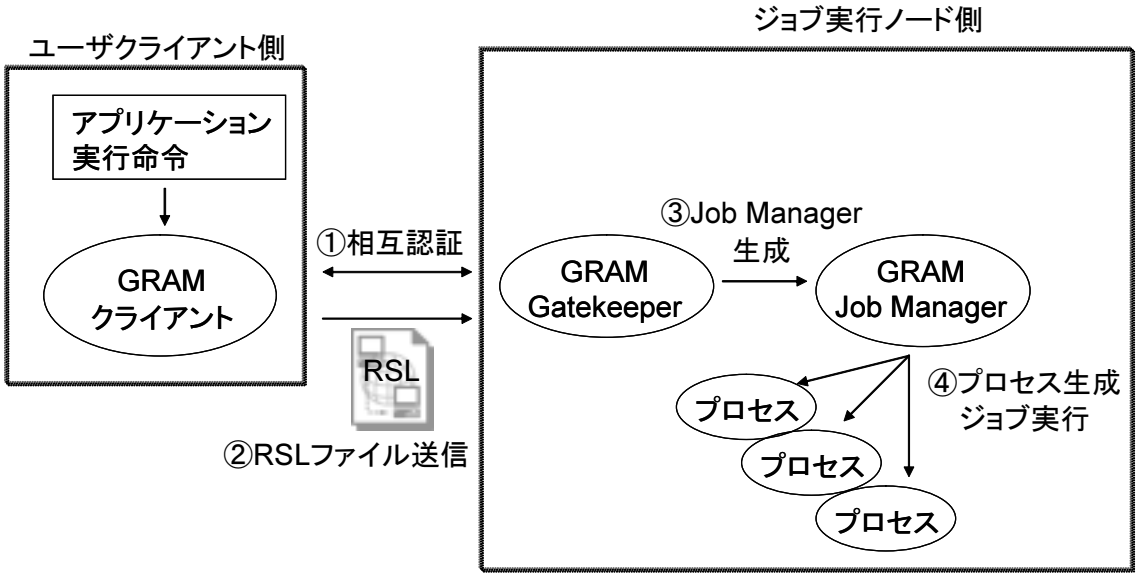

図 4: GRAM によるジョブ実行の流れ

2.2 分散計算のための MPI ライブラリ MPICH–G2

一般に，分散計算を行うシステムは，2 種類に大別できる．ひとつは，1 台の計算機に複数のプロセッサを搭載し，演算処理を行うためのシステムである．この方式では，全てのプロセッサがアドレス空間を共有しており，全プロセッサから単一の物理アドレスによりアクセスされる．このようなモデルとして UMA (Uniform Memory Access) があげられる．もうひとつは，ネットワークを通じて複数の計算機を接続して演算を行うシステムである．この方式では，全てのプロセッサが同一のアドレス空間を共有する場合と，各プロセッサが互いに独立したアドレス空間のメモリを持つ場合とがある．前者は，全てのプロセッサが同一のアドレス空間を共有するため，他のプロセッサのメモリへのアクセスはバスやネットワークなどの結合網を経由してアクセスすることができる．例として NUMA (Non-Uniform Memory Access) モデルがあげられる．一方，後者は，ネットワークを通じたメッセージパッシングによって計算を進め，それぞれの計算機のメモリは，全体の共有メモリとしては機能しない．メッセージパッシングを行うことにより，計算に必要なデータを共有し，他のプロセスと協調して動作することができる．このようなモデルとして NORA (NO Remote Memory Access) モデルがあげられる．

共有メモリを持たない NORA モデルの分散計算において，各プロセス間でデータの共有や同期を行うために必要なデータ交換の標準インターフェースとして考案，策定されたのが

MPI (Message-Passing Interface) である．MPI は C または Fortran から呼び出される関数群であり，分散計算を行うために必要なデータ交換の仕様を定めたものである． すなわち，MPI は API の仕様を定めただけであるため，実際の処理をどのように行うかは実装によって異なる．MPI の代表的なライブラリとして LAM や MPICH があげられる．本報告で対象とする Globus Toolkit では，MPI ライブラリとして MPICH をベースとした MPICH–G2 [21] が用いられている．

MPICH はアメリカのゴードン国立研究所が開発した MPI ライブラリである．移植しやすい実装になっているため，世界中のほとんどの並列計算機上で利用することができる．MPICH では，さまざまな計算機への移植を容易にするためのアーキテクチャとして ADI (Abstract Device Interface) という通信を抽象化したインターフェースを定義しており，ADI ではデバイスによって異なった実装をすることができる．

MPICH–G2 は MPICH のひとつの実装方法であり，MPICH のソースコードのうち，ADI の実装に Globus Toolkit の通信を行う API を利用している． 図 5 に MPICH–G2 の実装の関係を示す．MPI の実装で定められた “MPI_” で始まる MPI 関数は，その関数の中から “MPID_” で始まる ADI 関数を呼び出している．この MPI 関数の実装までは，MPICH のデバイスによらず共通である．ADI 関数の実装はデバイスによって異なっており，MPICH–G2 の実装においては ADI 関数関数の中から Globus Toolkit が提供する通信の API を呼び出している．

また，MPICH–G2 ではジョブ管理に Globus Toolkit の GRAM を利用している．GRAM によって，計算を行うリモートのジョブ実行ノード計算機上のプロセス生成と管理を行っている．図 6 に MPI アプリケーションが Globus Toolkit 上で実行される概念を示す． ユーザはあらかじめ，ジョブを実行するノード計算機および各ノード計算機で実行するプロセス数を指定しておく．mpirun コマンドにより MPI アプリケーションの実行命令が出されると，GRAM クライアントにより RSL ファイルが生成され，各ジョブ実行ノード計算機の Gatekeeper，Job Manager を通してプロセスが生成される．各プロセスは MPI アプリケーションの計算を実行し，アプリケーション内でデータ交換を行う MPI 関数が呼ばれると，Globus Toolkit が提供する通信の API を用いてデータ交換を行う．

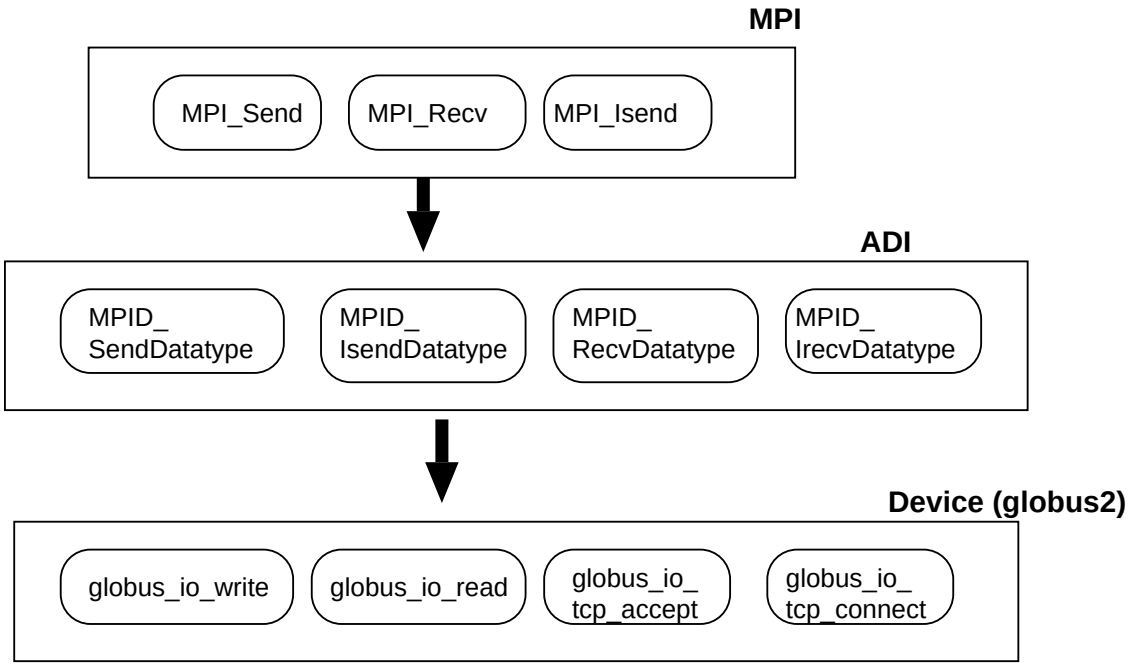

図 5: MPICH の階層

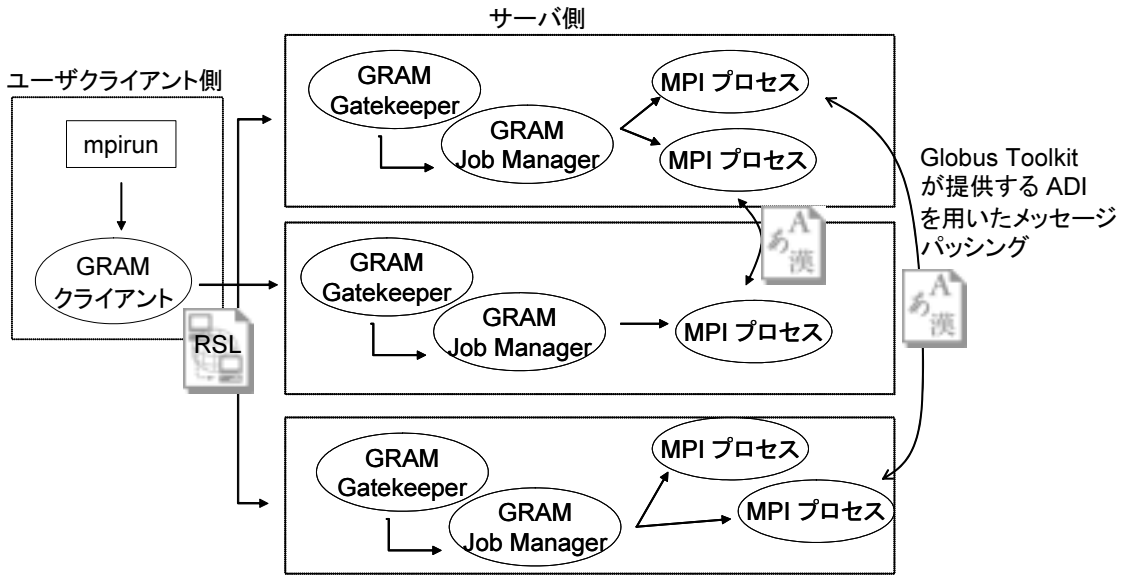

図 6: Globus Toolkit におけるジョブの実行方法

3　AWG–STAR システムの概要

本報告では，λコンピューティング環境構築のためのひとつの手法として，AWG–STAR システムを利用する．本章では AWG–STAR システムについて説明する．

3.1　AWG–STAR システム

AWG–STAR システムは，日本電信電話株式会社フォトニクス研究所により開発されたシステムであり，WDM 技術によるデータ転送と AWG ルータによる波長ルーティング技術によって実現された情報共有ネットワークシステムである [16-18]．AWG ルータは波長に基づいたルーティングを行っており，電気信号に変換せず光信号をそのまま処理するため，高速なネットワークを構築することができる．また，各ノード計算機は，共有メモリボード上に，全てのノード計算機で同一のデータを保持する共有メモリを持ち，AWG および WDM を利用して構成された高速な光リングネットワークを利用することによりノード計算機間で共有メモリ上のデータ交換をリアルタイムに行うことができる．従来のシステムでは，データ共有するためには何らかの明示的なデータ転送が必要であったが，AWG–STAR システムでは共有メモリに書き込まれたデータは光リングネットワークに送出され，全ノード計算機の共有メモリが自動的に更新される．AWG–STAR システムを用いることにより，共有メモリ上のデータ共有は，共有メモリに書き込む処理によりハードウェアがバックグラウンドで行うため，高速に実行される．他ノード計算機が更新したデータの取得は，AWG–STAR システムを通じて共有メモリに配信され，自動的に更新されるため，共有メモリから読み込むことにより実現できる．

本報告では，λコンピューティング環境の構築に AWG–STAR システムを用い，分散計算を行うことを考える．AWG–STAR システムの利用方法により，2 つのモデルが考えられる．ひとつは，共有メモリ上に計算対象となるデータを載せ，各ノード計算機が共有メモリとの間でデータの読み込み，書き込みを行うモデルである．この場合，共有メモリは全ノード計算機間で共有されるため，データ更新時にデータが光リングネットワークを 1 周するのを待つ必要があり，そのための遅延が生じる．もうひとつは，共有メモリに計算対象となるデータを載せるのではなく，主に AWG–STAR システムを高速チャネルとして利用するモデルである．すなわち，各ノード計算機の主メモリにデータを載せ，計算を行い，情報交換が必要な場合にのみデータを共有メモリ上に書き込み，他ノード計算機に高速に伝送するものである．本報告では，MPI ライブラリを用いた分散計算を対象にしており，親和性が高く，また共有メモリアクセスにおける遅延時間を軽減できるため，後者のモデルを対象とする．

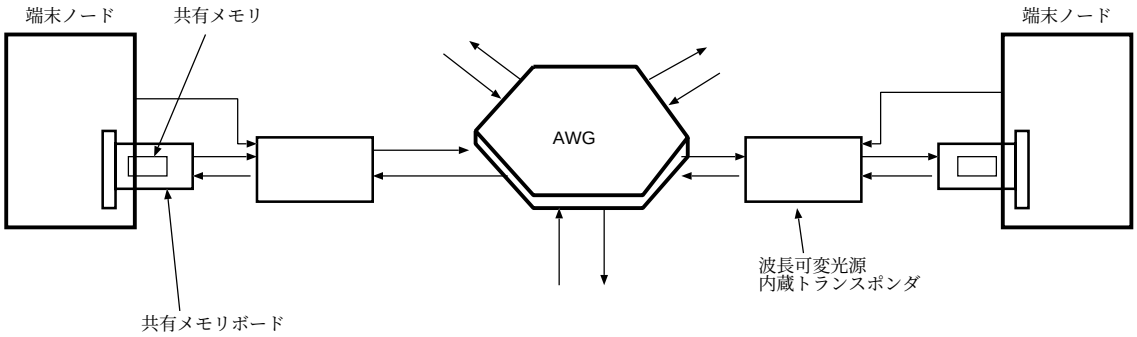

図 7: AWG–STAR システムの構成

表 1: 共有メモリボードの仕様

光インターフェースの伝送速度	2.152Gbps
ノード計算機の 1 回あたりの転送データ量	1KByte
ノード計算機でのフレーム転送処理遅延	500ns
共有メモリへの書き込み	60MBytes/s
共有メモリから読みだし	67MBytes/s

3.2 AWG–STAR システムの構成

AWG–STAR システムの概略図を図 7 に示す．このシステムでは，各ノード計算機は波長可変光源を通じて光ファイバにより AWG ルータに接続され，物理的にはスタートポロジを，論理的にはリングトポロジを形成し，光リングネットワークを形成している（図 8）．AWG–STAR はシングルモード光ファイバを使用している．AWG–STAR システム上の全ノード計算機は，共有メモリボードを搭載し，共有メモリはこのボード上にある．以降，特に断らない限り共有メモリは共有メモリボード上のものを指す．表 1 に共有メモリボードの仕様を示す [22].

また，AWG ルータは波長に基づいた動的なルーティングを行うルータである．AWG ルータは 32 個の入力ポートと 32 個の出力ポートを持っており，入力ポートに入力された光はその波長によって出力ポートが決定され出力される．波長の割り当て例について表 2 に示す．例えば，入力ポート 2 に波長 58 の光が入力されれば出力ポート 1 に出力される．ただしこの数値は AWG ルータのために定められている独自の波長番号である．

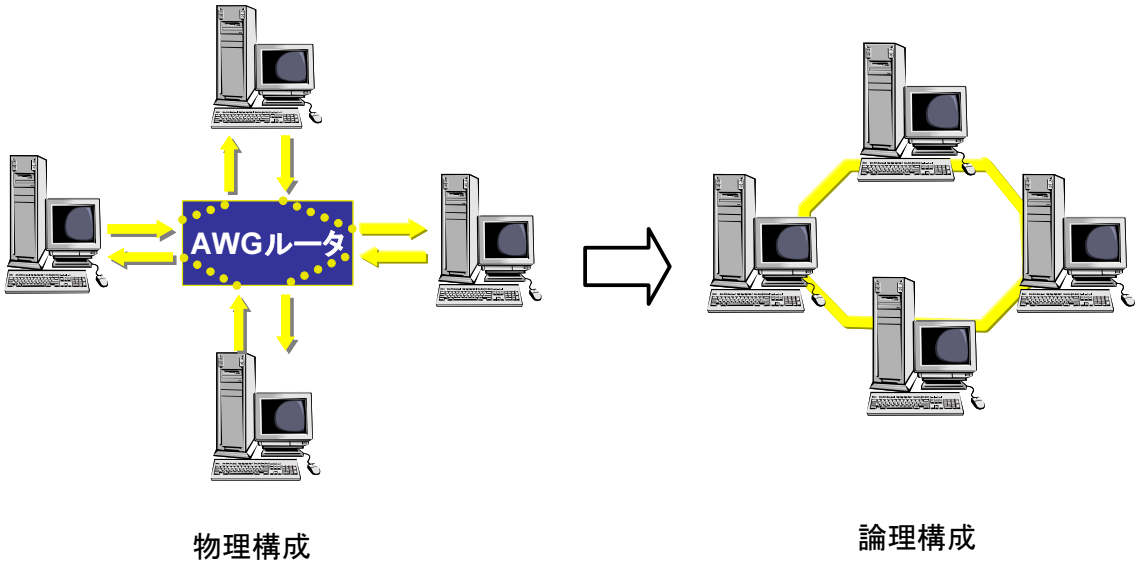

図 8: 光リングネットワークの構成

表 2: AWG ルータの入出力ポートと波長の対応

出力 / 入力	ポート 1	ポート 2	ポート 3
ポート 1	56	58	60
ポート 2	58	60	62
ポート 3	60	62	64

3.3 共有メモリへのアクセス

共有メモリへのアクセス手法には 2 通りある．ひとつは共有メモリボードの機能を用いた DMA (Direct Memory Access) アクセスであり，もうひとつはポインタを用いたアクセスである．これらは，共有メモリの先頭からのオフセットもしくは直接アドレスを指定することでアクセスが可能である．

共有メモリは共有メモリボード上にあり，計算機とは PCI バスで接続されている．そのため，共有メモリへのアクセスには，共有メモリへの書き込みもしくは読みだし時間に加えて PCI の転送時間が必要になり，ローカルメモリへアクセスする場合よりも遅延時間が大きくなるという問題がある．

共有メモリからのデータの取得については，自ノード計算機の共有メモリから読み出すた

め，光リングネットワークの通信路に負担をかけない．一方，共有メモリへの書き込みに際しては光リングネットワークへのアクセスが発生する．

3.4 高速チャネルにおける通信手法

光リングネットワーク上では，常にひとつのトークンが流れており，各ノード計算機はそのトークン上に更新を行ったデータに関する送信フレーム（アドレス，データ，制御コード，CRC）を付加することにより通信を行う．共有メモリの更新には 2 つの場合がある．ひとつは自ノード計算機の共有メモリに書き込む場合であり，もうひとつは他ノード計算機からの共有メモリの更新情報を受信した場合である．

1. 自ノード計算機の共有メモリに書き込む場合
   この場合，まず共有すべきデータを自ノード計算機の共有メモリに書き込み，その後トークンが回ってきた際に，送信したいフレームをトークンに付随している送信フレームの最後尾に付加し，次のノード計算機にトークンを転送する．リングを 1 周し，トークンが再度回ってきたら，先ほど付加した送信フレームを削除する．図 9 に自ノード計算機の共有メモリに書き込む場合の動作モデルを示す．

2. 他ノード計算機からの更新情報を受信した場合
   トークンが回って来た時にトークンに付加されている他ノード計算機の送信フレームを確認する．他ノード計算機の更新データがトークンに付随していれば，データを読み込み，自ノード計算機の共有メモリを更新し，次のノード計算機に向けてそのままトークン，データを送出する．図 10 に他ノード計算機からの更新データを受信した場合の動作モデルを示す．

図 11 にデータ更新時の動作を示す．あるノード計算機において共有メモリへの書き込みがあり，共有メモリ上のデータが更新された場合，そのノード計算機は光リング上を流れているトークンを待ち，トークンに更新データを付加する．他ノード計算機はトークンに付随している更新データを受け取り，自ノードにおける共有メモリ上のデータを更新する．トークンがリングを一周して再度回ってきたら，自ノードが付加した更新データを削除する．

3.5 AWG–STAR における遅延時間

各ノード計算機の共有メモリを利用するには，ローカルメモリにアクセスする以上に遅延時間を要する．その要因は PCI バスを経由する遅延時間とデータ共有を行うための遅延時間である．すなわち，ローカルメモリのデータを共有メモリボードに転送する遅延時間と，

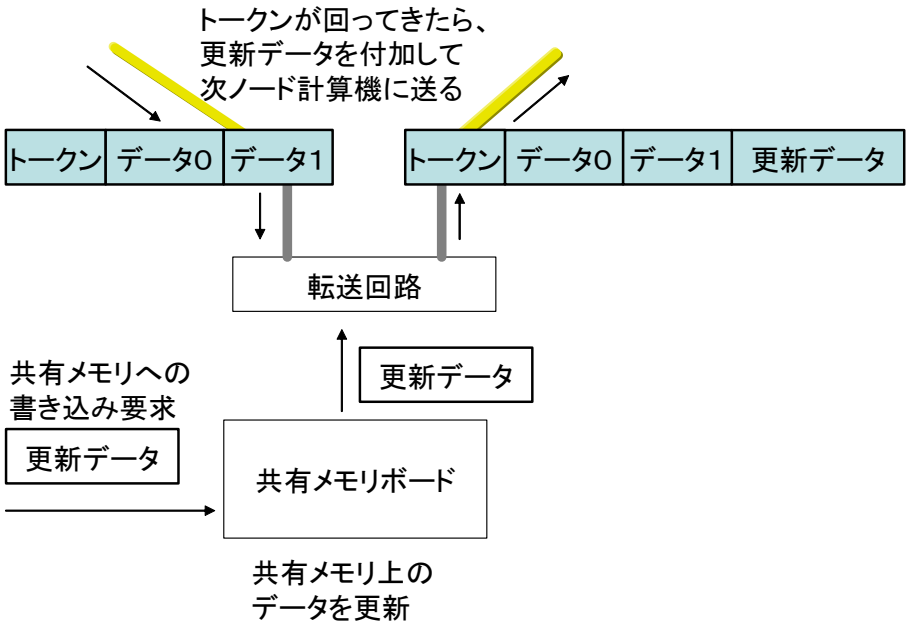

図 9: 自ノード計算機の共有メモリに書き込む場合

データを全ノード計算機が共有するために，少なくともデータが光リングネットワークを一周回する時間である．

すなわち，PCI バスを経由したデータアクセス速度は，表 1 に示した通りである．また，データ共有のための遅延の要因は 2 つある．ひとつは各ノード計算機上の共有メモリボードにおける転送処理遅延である．つまり，3.4 節に述べた，他のノード計算機から転送されてきたフレームを処理し，次のノード計算機に転送するために処理時間を必要とする．具体的には送信フレームの追加と削除，共有メモリへの反映のためにおよそ 500ns の時間を必要とする．もうひとつはリングネットワーク 1 周にかかる光ファイバによる伝搬遅延であり，5ns/m となっている．

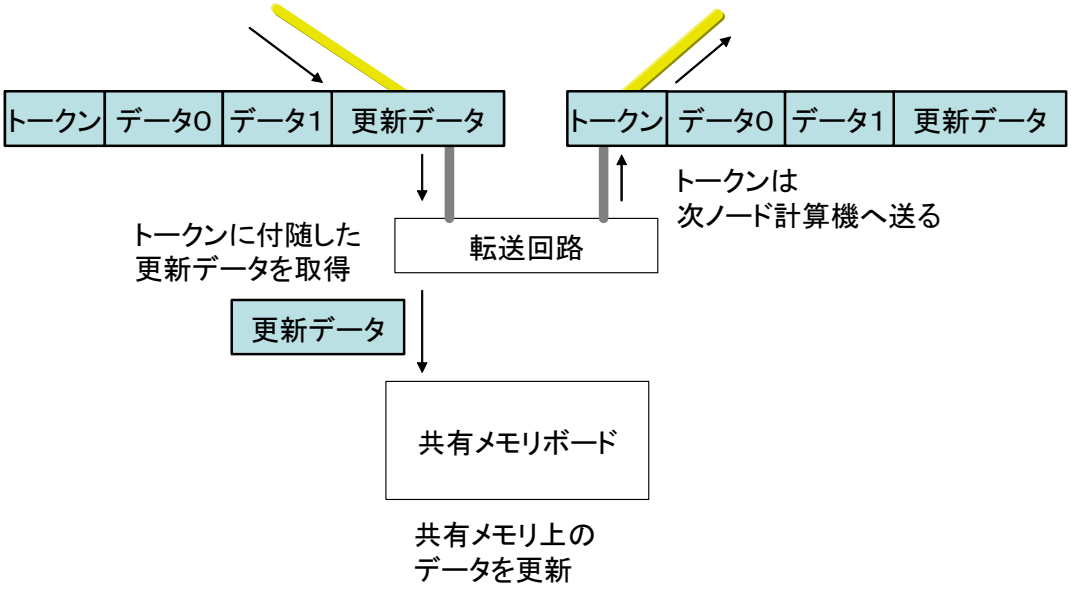

図 10: 他ノード計算機の更新情報を受信した場合

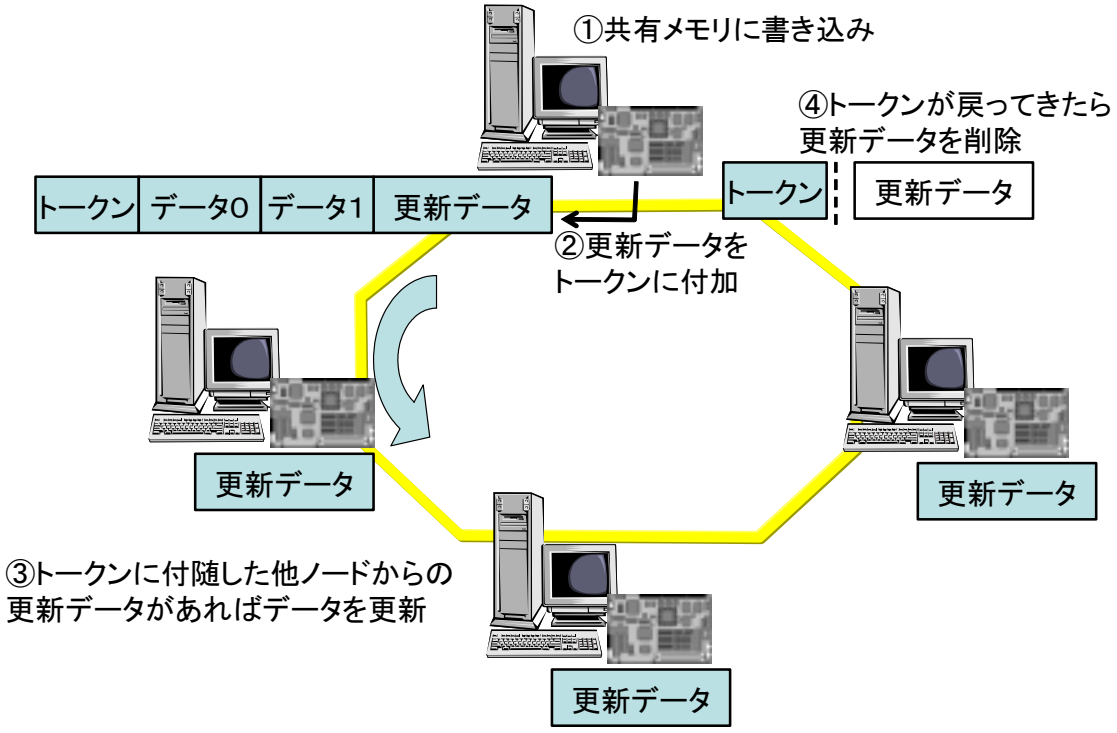

図 11: データ共有の流れ

4　共有メモリを用いたメッセージパッシング手法の提案

本章では AWG–STAR システムを用いた λ コンピューティング環境において，共有メモリを用いて行うメッセージパッシング手法および実装方法を提案する．実装するにあたって，MPICH–G2 の実装方法を参考にしたため，まず MPI の基本動作と MPICH–G2 におけるメッセージパッシングの方法について説明し，その後 MPICH–G2 を AWG–STAR システムに適用した場合の共有メモリを用いたメッセージパッシング手法について述べる．

4.1　MPI アプリケーションの基本動作

MPI アプリケーションは，1 つまたは複数の MPI プロセスから成る．各 MPI プロセスは，そのアプリケーションで実行される MPI プロセスの総数に応じてランクという非負整数値をもち，異なる MPI プロセスでは異なる値をもつ．例えば，あるアプリケーションで実行される MPI プロセスの個数が n 個の場合，各 MPI プロセスは $[0, 1, \cdots, n-1]$ のいずれかのランクを割り当てられる．以下，MPI アプリケーションにおいて実行される MPI プロセスを，単にプロセスと呼ぶ．

プロセス間でのデータ共有や，プロセス間での同期などは，メッセージパッシングで実現されている．メッセージパッシングを行うためには，メッセージデータを送信するプロセスがメッセージ送信関数を呼び，また，メッセージデータを受信するプロセスがメッセージ受信関数を呼ぶ必要がある．以下，前者のプロセスを送信プロセス，後者のプロセスを受信プロセスと呼ぶ．

4.2　MPICH–G2 におけるメッセーパッシング手法

MPICH–G2 におけるメッセージパッシングは TCP/IP を用いて行われ，通信のための API は Globus Toolkit が提供している．ただし，Globus Toolkit が提供する通信のための API は，基本的にはソケット通信の API と同様であるので，簡単のためにここではソケットにより説明する．図 12 に示すように，各プロセスは全ての他プロセスとの通信ソケットを確立しており，メッセージパッシングはソケットを用いて行っている．以下，実際に通信ソケットから受信したデータを受信データと呼び，受信関数から呼ばれた受信したいデータを受信要求と呼ぶ．

各プロセスでは，データを送信するために送信先プロセスごとに送信キューを持っている．各送信キューは実行するアプリケーションプロセスとは独立した異なるスレッド上で動いており，送信キューは通信ソケットを用いて送信先プロセスへデータを送信する．図 13 に送信側の動作モデルを示す．

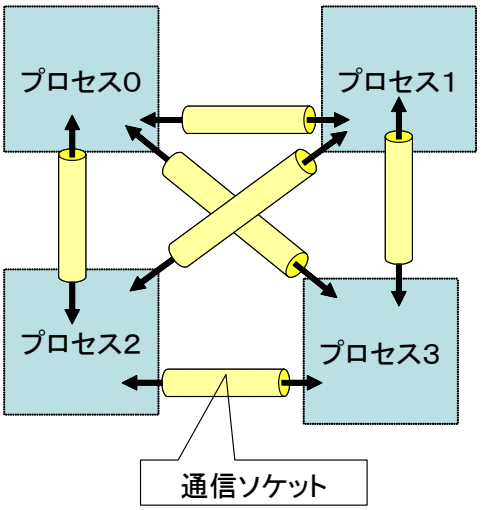

図 12: プロセスと通信ソケット

また，各プロセスではデータを受信するために，unexpected キューと posted キューと呼ばれる 2 種類の受信用キューを持っている．通信ソケットでデータを受信するタイミングと，プロセス上で受信関数が呼ばれるタイミングが前後するため，これらのキューを使い分けることでタイミングの差を吸収し，受信データと受信要求の対応付けを間違うことなく行うためである．受信関数が呼ばれたにも関わらず受信データを受信していない場合，データを受信するまで受信要求をバッファリングしておく必要がある．データを受信するまで受信要求をバッファリングするためのキューが posted キューである．逆に，通信ソケットからデータを受信したにも関わらずそのデータを受信するべき受信関数が呼ばれていない場合，受信関数が呼ばれるまで受信データをバッファリングしておく必要がある．受信関数が呼ばれるまで受信データをバッファリングするためのキューが unexpected キューである．受信データと受信要求は，データに付加されたヘッダの情報を元にして，要求に一致するデータであるかどうかを判定する．以下にこれらのキューがどのように利用されているか，データを受信したときと受信関数が呼ばれた場合の双方について述べる．

- データを受信したとき

  posted キューを検索して一致する受信要求がすでにあるか調べる．一致する受信要求があった場合，その受信要求を posted キューに挿入した受信関数の受信データとし，受信要求を posted キューから削除する．posted キュー内に一致する受信要求がなかった場合，受信データを unexpected キューに挿入する．図 14 にデータ受信時の動作を示す．

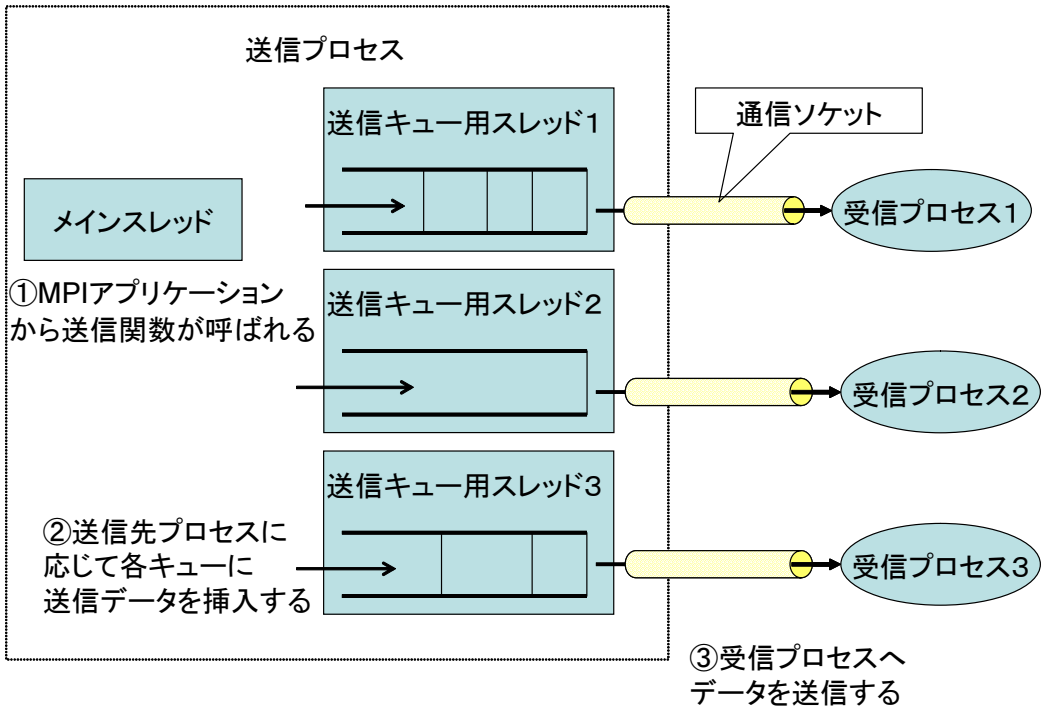

図 13: 送信側動作

- 受信関数が呼ばれたとき

  unexpected キューを検索して一致する受信データをすでに受信しているか調べる．一致する受信データがあった場合，そのデータを受信関数の受信データとし，受信データを unexpected キューから削除する．unexpected キュー内に一致する受信データがなかった場合，受信要求を posted キューに入れる．図 15 に受信関数呼び出し時の動作を示す.

4.3 AWG–STAR システムを利用した MPI ライブラリの実装方法の提案

MPICH–G2 におけるメッセージパッシングの手法に基づいて，AWG–STAR システムにおける共有メモリを用いた MPI ライブラリの実装方法を提案する．MPICH–G2 における通信の部分を改変することで，Globus Toolkit の API を利用せず，共有メモリを用いたメッセージパッシングを行う．ただし，今回の実装では，それぞれのノード計算機上ではひとつの MPI プロセスを動作させることを前提としている．これは AWG–STAR システムでは，共有メモリボードを複数のアプリケーションやプロセスで共有することを想定していないた

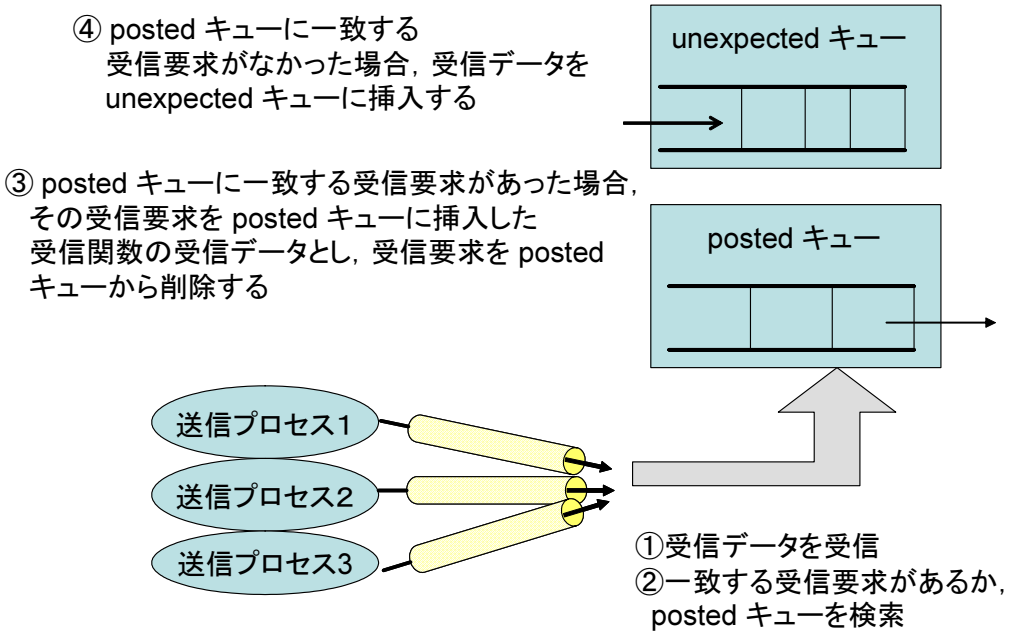

図 14: 受信側動作 (データ受信時)

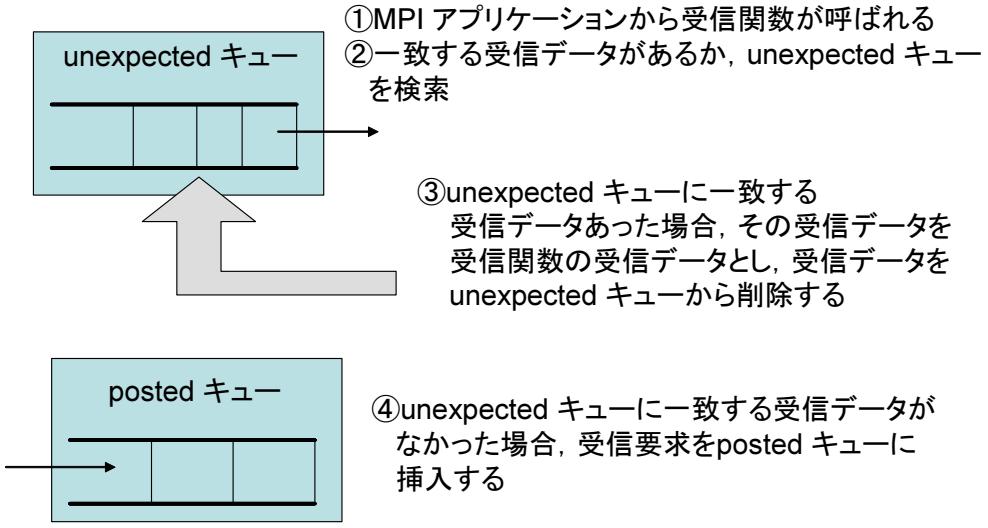

図 15: 受信側動作 (受信関数呼び出し時)

めである．従って，共有メモリの動的な割り当て機構も装備していないため，共有メモリエリアの利用方法は事前に決めておく必要がある．それを解決するための，共有メモリボードへアクセスやメモリの割り当てなどを行うメモリマネージャの実装は，今後の課題である．

AWG–STAR システムを用いてメッセージパッシングを実現する場合，MPICH–G2 における実装と異なる点は，MPICH–G2 では送信プロセスのローカルメモリ上で実装されている送信キューを共有メモリ上で実装する点である．また，MPICH–G2 における実装と同様の点は，受信プロセスのローカルメモリ上で実装されている unexpected キューと posted キューの実装方法と使用方法である．ただし，データを受信する際は，通信ソケットからデータを受信するのではなく，共有メモリからデータを読み込むという方法をとる．

まず，MPICH–G2 では送信プロセスのローカルメモリ上で実装されている送信キューを，共有メモリ上で実装する方法を述べる．各プロセスが，自プロセスも含めた全てのプロセスへデータを送信するためのキューを持つ必要があるため，プロセス数を n とすると，共有メモリの領域を n^2 個に分割し，各送受信プロセス間ごとの送信キュー用領域に割り当てる．図 16 に共有メモリの利用方法を示す．送信プロセスでは送信関数が呼び出されると，送信データを共有メモリ上に書き込む．共有メモリ上にデータを書き込む際，前のデータが書き込まれているアドレスの次から書き込まむ必要があるため，前のデータが書き込まれている最後尾のアドレスを送信プロセスのローカルメモリ上に保持している．送信キューを共有メモリ上に実装することにより，MPICH–G2 ではローカルメモリ上の送信スレッドとネットワーク上の通信ソケットに分かれていたデータ送信の作業を，ひとつの作業としてとらえることができる．なお，各送受信プロセス間に割り当てられた送信キューの領域を越えてデータを送信キューに挿入しようとした場合，新しいデータは送信キューの先頭アドレスから書き込まれる．

次に，受信プロセスが共有メモリからデータを読み込む場合を考える．この場合，受信プロセスは，送信プロセスが共有メモリにデータを書き込んだことを知る必要がある．そのために，AWG–STAR システムが提供するシグナルの機能を用いる．AWG–STAR システムが提供するシグナル機能とは，あるノード計算機が他の指定したノード計算機に対してシグナルを送り，シグナルを受けるノード計算機はどのノード計算機からのシグナルなのかを知ることができる機能である．今回の実装ではひとつのノード計算機上でひとつのプロセスを動かすことを前提にしているのため，ノード計算機を特定することでプロセスを特定することができる．

図 17 に共有メモリを用いたデータ送受信の送信プロセスと受信プロセスの動作遷移の様子を示す．送信プロセスは送信するデータを共有メモリ上に書き込んだ後，受信プロセスが動くノード計算機に対してシグナルを送信する．各プロセスは常に他のプロセスからのシグナルを待ち受けている．プロセスがシグナルを受けた場合，シグナルを送信したノード計

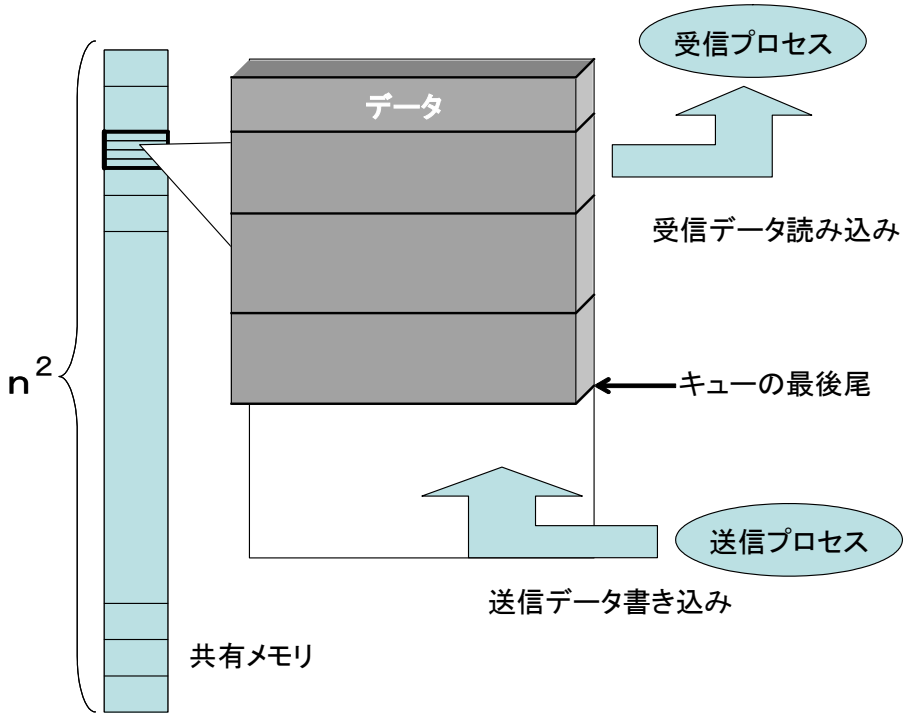

図 16: 提案手法における共有メモリの利用方法

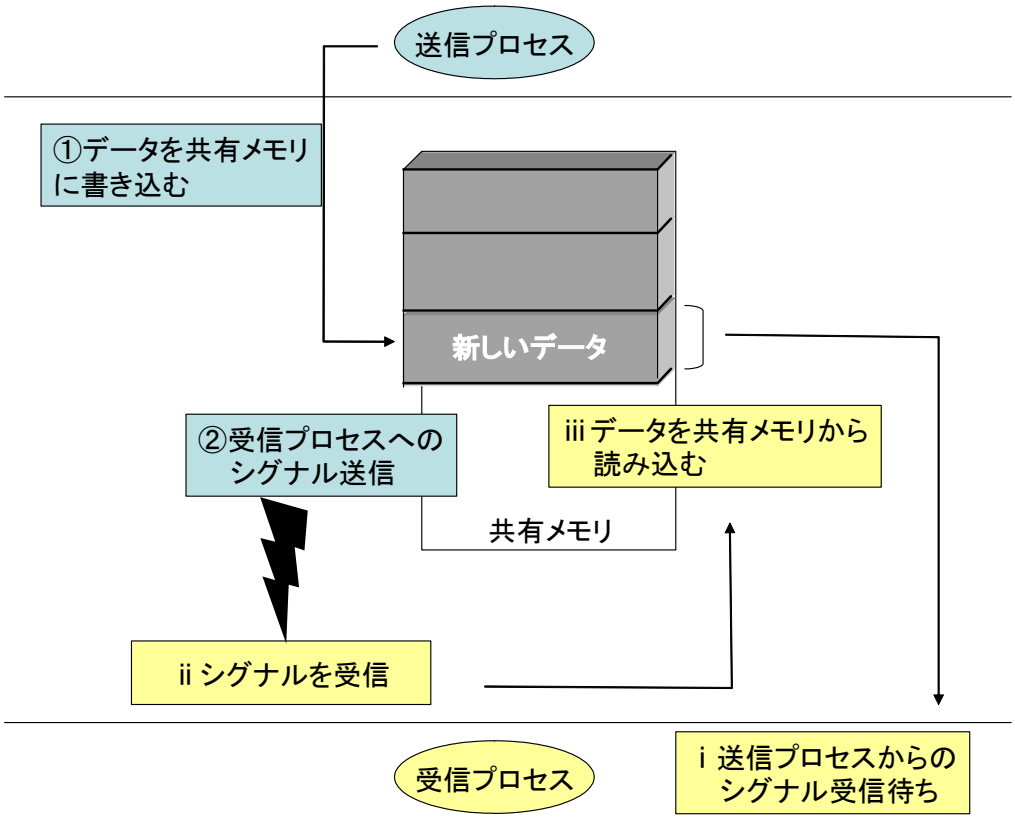

図 17: 送受信プロセス間のシグナリング

算機上で動くプロセスからデータを受信する必要があることを知り，共有メモリからデータを読み込む．共有メモリから読み込んだ受信データは，MPICH–G2 と同様の動作をして，posted キューに入れられるか，あるいは，一致する受信要求を発行した受信関数の受信データとなる．

4.4 MPITCH–G2 と提案方式の工数の検討

MPICH–G2 による MPI アプリケーションの実行方法と，AWG–STAR を用いた提案方式による MPI アプリケーションの実行方法を図 18 に示す．プロセスを動かすジョブ実行ノードに対する認証とジョブ実行要求送信は Globus Toolkit の API を用いて行う．MPI アプリケーションを動作させメッセージパッシングを行う際は，MPITCH–G2 を用いた場合は，ローカルメモリから OS のシステムコールを通じてソケットバッファにコピーされ，次にソケットバッファから NIC のメモリ上に転送され，NIC 内で MAC フレームを生成し，イーサネットを通じて転送される．AWG–STAR を用いた場合は，ローカルメモリから PCI バ

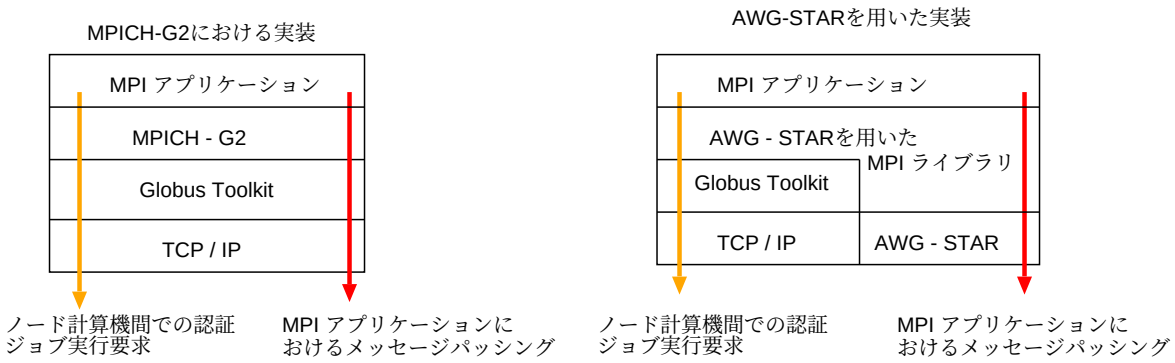

図 18: MPICH–G2 と提案方式の比較

スを通じ，共有メモリボードに転送，バッファされ，ブロック分割の後，ネットワークに送出される．AWG–STAR を用いた構成では，メモリコピーとパケット生成にかかるオーバーヘッドが少なく，高速なデータ交換が可能であると考えられる．

5　MPI アプリケーションによる評価

本章では，並列アプリケーションのベンチマークである姫野ベンチマーク [23] を MPI アプリケーションとして動作させ，計算速度を測定することにより，AWG–STAR を用いて構成した共有メモリシステムとそのメモリアクセス手法の性能を評価する．また，MPICH–G2 上でも同様のアプリケーションを走らせて比較対象としている．

5.1　実験システム環境

評価に用いた計算機の仕様を表 3 に示す．実験に使用した計算機の台数は 1 台から 4 台の範囲で行い，全て同じ性能の計算機を用いた．今回の実験では，ノード計算機数に応じて光リングの長さを変えている．具体的には，ノード計算機数を N とすると，光リングネットワークの長さは $10N$m としている．MPICH–G2 を利用する際はネットワークとして 100Mbps の Ethernet を用いている．

5.2　評価に用いるアプリケーションプログラム

姫野ベンチマークは，理化学研究所の姫野龍太郎氏が非圧縮流体解析コードの性能評価のために考案したベンチマークで，ポアソン方程式をヤコビの反復法で解く場合に主要なループの処理速度を計るものである．ヤコビ法は領域分割法を用いることで，三次元配列の領域をプロセス数で等分割し，並列化を行うことが可能である．例えば，1 つの問題を 4 つのプロセスで並列計算を行なう場合，三次元配列を 4 つに等分割する．図 19 に示すように，問題を分割する際は複数通りの分割方法がある．今回ベンチマークとして走らせた問題サイズはあらかじめ姫野ベンチマークで設定されている問題サイズを変更している．

5.3　アプリケーションプログラムによる評価

5.3.1　MPICH–G2 との比較による評価

AWG–STAR システムを用いたシステムで実装した MPI ライブラリでの処理速度と，Ethernet を使用している MPICH–G2 での処理速度の比較を行った．図 20 に 2 プロセスでアプリケーションを実行したときの処理速度を，図 21 に 4 プロセスでアプリケーションを実行したときの処理速度を示す．横軸には問題サイズ，縦軸には処理速度 (MFLOPS) をとっている．問題サイズの大きさと一度に交換するデータの大きさは比例関係にあり、問題サイズの大きさとデータの交換回数は反比例関係にある．

表 3: 実験に用いた計算機の仕様

CPU	Xeon 2.80 GHz
メインメモリ	SDRAM 512MB
1 次キャッシュ	512KB
2 次キャッシュ	512KB
NIC	Intel PRO/100MT
PCI バス	64 bit/66MHz
PCI 転送速度	533MBytes/sec
OS	Redhat Linux 7.3
コンパイラ	gcc 2.96
グリッド環境ミドルウェア	Globus Toolkit 2.4

図 20，図 21 で示されているように，AWG–STAR を用いた MPI アプリケーションでは，問題サイズが小さいときは処理速度が遅い．これは，問題サイズが小さいときは共有メモリへのアクセスが頻繁に発生するため，共有メモリボードへのアクセス遅延が大きなボトルネックになっているからである．しかし，問題サイズが大きくなり，共有メモリへのアクセス回数が減少すると，共有メモリボードへのアクセス遅延のボトルネックも減少するため，処理速度が速い．また，問題サイズが大きいときは一度に交換するデータサイズが大きいため，パケット処理が必要な MPICH–G2 に比べて AWG–STAR を用いた場合の方がよい性能が得られる．

5.3.2 プロセス数による評価

同じ大きさの問題サイズを，異なるプロセス数で実行したときの処理速度を図 22 に示す．1 プロセスと 2 プロセスでの処理速度を比べた場合，問題サイズが $129 \times 129 \times 257$ より小さいときは，2 プロセスで実行したときよりも 1 プロセスで実行したときの方が処理速度が速い．これは，1 プロセスで実行する際はメッセージパッシングを行わないため，メッセージパッシングのオーバーヘッドがないためである．しかし，問題サイズが大きくなるにつれて 1 プロセスと 2 プロセスの処理速度の差は少なくなり，問題サイズが $161 \times 161 \times 321$ のときは 2 プロセスで実行した方が処理速度が速い．この理由は，1 つのプロセスに与えられた領域が大きくなるにしたがって，処理速度が遅くなるからである．また，2 プロセスと 4 プロセスでの処理速度を比べた場合，2 プロセスで実行した方が処理速度が速い．1 プロセスと 2 プロセスを比較した場合と同様に，2 プロセスで実行した方がメッセージパッシング

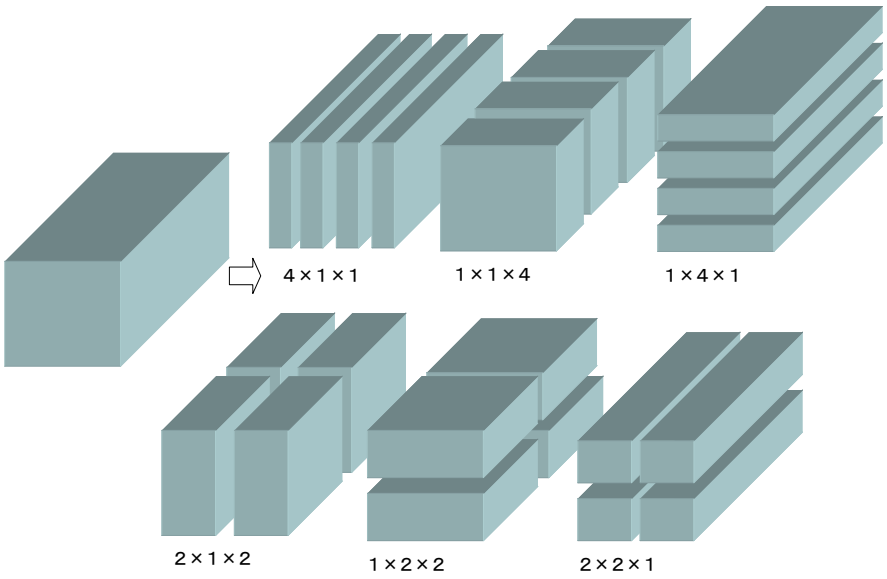

図 19: 姫野ベンチマークにおける 4 プロセスへの問題分割例

の回数が少ないからである．ただし，問題サイズが $193 \times 193 \times 385$ 以上のとき，問題サイズが大き過ぎるために 1 プロセスでは実行不可能であった．同様に，2 プロセスで問題サイズ $225 \times 225 \times 449$ を実行することもできなかった．つまり，1 つのプロセスでは計算できないような大きな問題サイズの場合も，複数の計算機を利用し分散計算することにより結果を得ることができることがわかる．

5.3.3 問題分割の違いによる評価

並列計算において交換をするデータサイズは，おおよそ問題分割により境界を共有している面積の大きさに比例する．そのため，実行するアプリケーションでは，同じサイズのデータを繰り返し交換するが，問題分割の違いによって同じ問題サイズであっても交換するデータサイズが異なる．図 23 に，アプリケーションの問題分割を方法を変えたときの処理速度の変化を示し，表 4 に，問題サイズ $97 \times 97 \times 193$ のときの交換するデータサイズを示す．単位は Byte，16 進数表記である．なお，問題分割で $4 \times 1 \times 1$ と $1 \times 4 \times 1$ を同じとみなし，$2 \times 1 \times 2$ と $1 \times 2 \times 2$ を同じとみなしたため，問題分割 $4 \times 1 \times 1$ と $1 \times 2 \times 2$ を省いている．交換するデータサイズが大きい，問題分割が $1 \times 4 \times 1$ や $2 \times 2 \times 1$ の場合は，処理速度が遅くなっている．

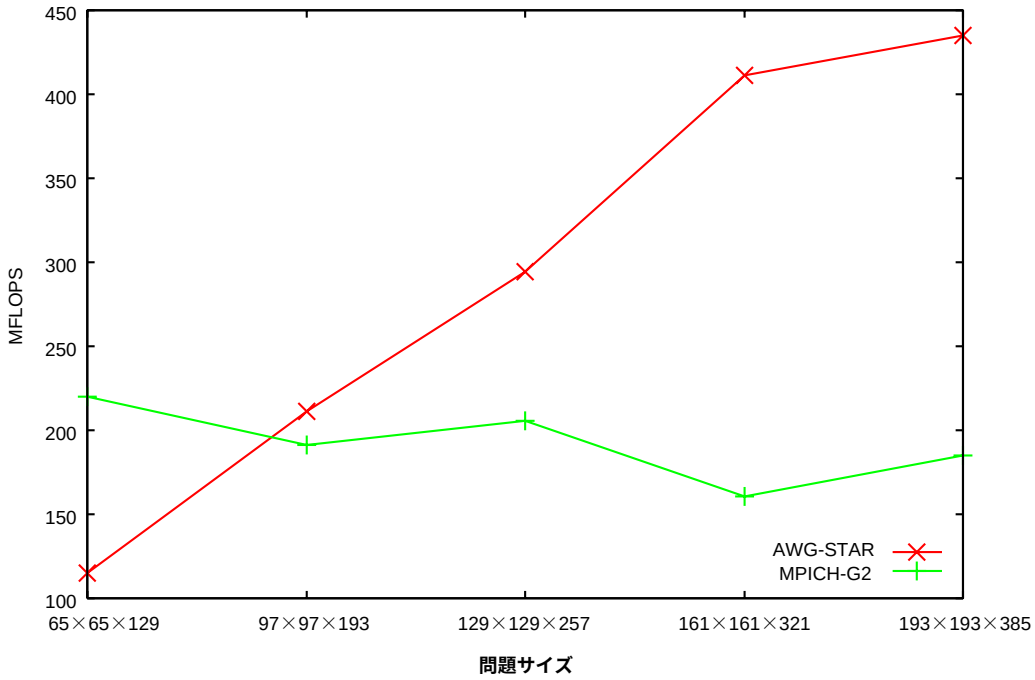

図 20: 2 プロセスで動作させたときの処理速度

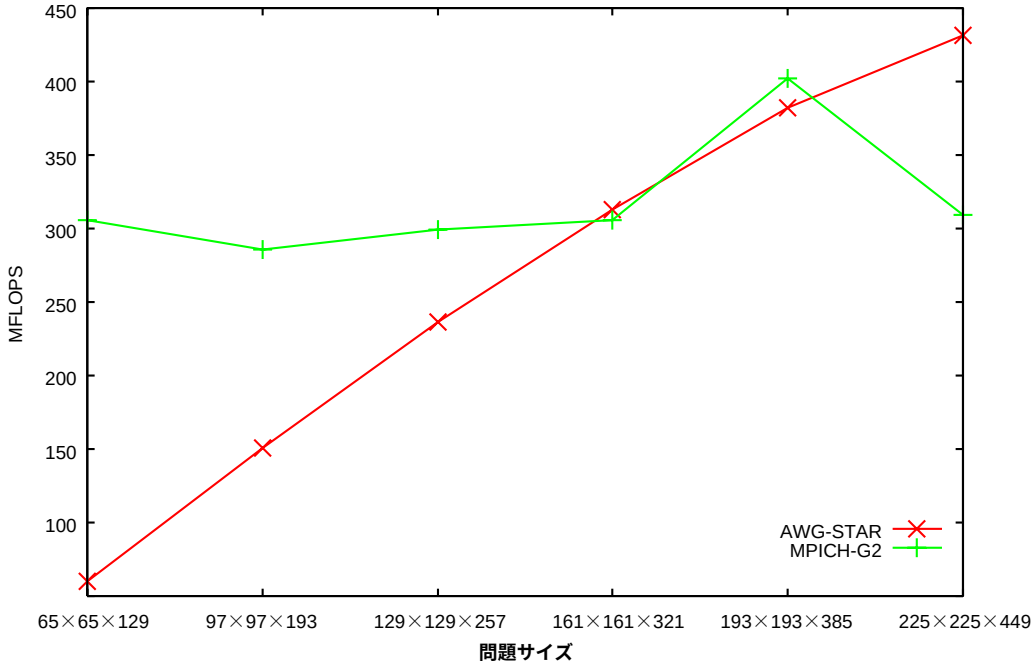

図 21: 4 プロセスで動作させたときの処理速度

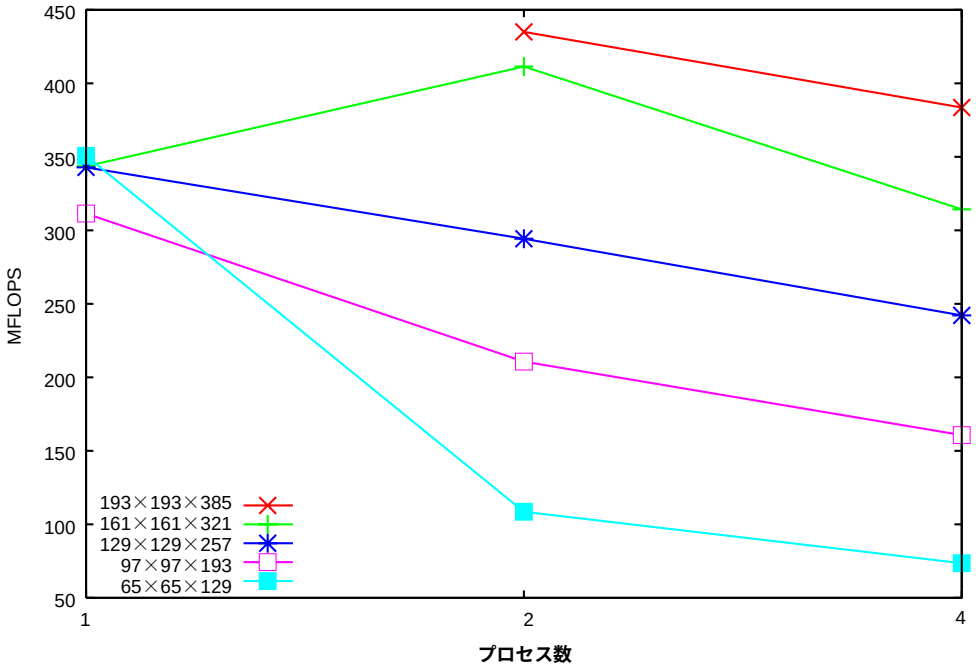

図 22: プロセス数の違いによる処理速度の変化

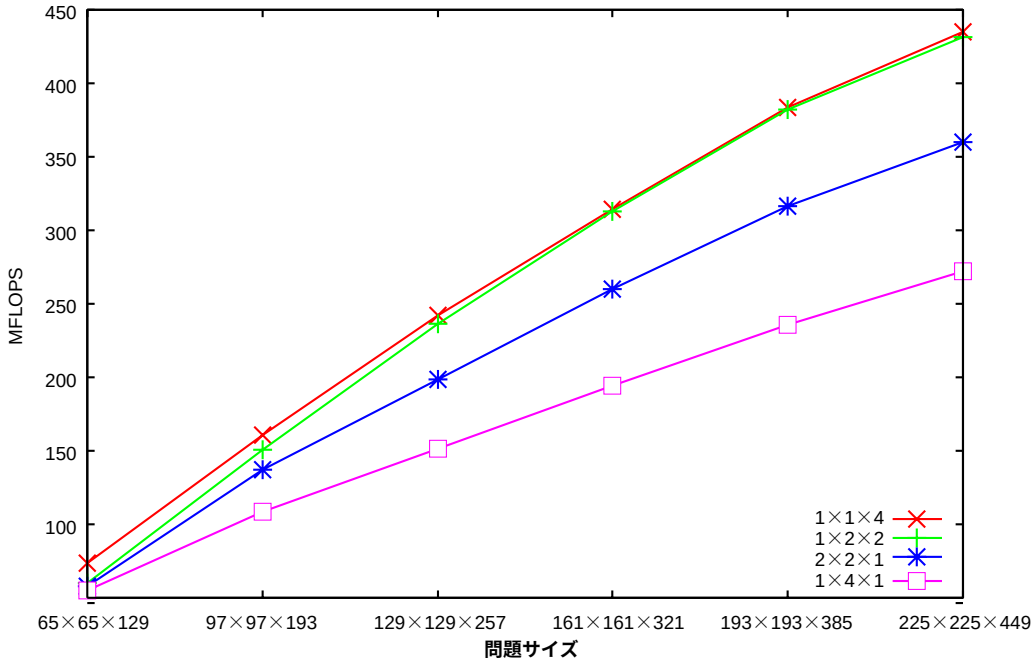

図 23: 問題分割の違いによる処理速度変化

表 4: 問題分割による交換データサイズの違い

問題分割方法	データサイズ
$1 \times 1 \times 4$	0x9000
$1 \times 2 \times 2$	0x9180,0x4980
$2 \times 2 \times 1$	0x9300
$1 \times 4 \times 1$	0x12000

6　おわりに

本報告では，λコンピューティング環境の実現形態のひとつとして WDM 技術に基づくフォトニックネットワークである AWG–STAR システムを用いてグリッド計算環境を構築し，グリッド計算環境でデファクト標準となっている Globus Toolkit をミドルウェアとして実装するため，分散計算を行うための MPI ライブラリの実装手法を提案した．

さらに，MPI アプリケーションを実行することにより，構築したシステムが動作することを確認し，構築システムにおける分散計算性能を評価した．その結果，AWG–STAR システムに基づく共有メモリシステムをλコンピューティング環境として利用する場合，共有メモリへのアクセス回数と，共有メモリへ書き込むデータサイズが性能に影響を与えることが分かった．

今後の課題として，共有メモリへの動的なメモリの割り当てがあげられる．共有メモリの動的な割り当てが可能となれば，より柔軟で効率のよい MPI ライブラリの実装が可能となる．また，1 台のノード計算機上で複数の MPI プロセスや他の共有メモリを用いたアプリケーションを実行させた際に競合しない機能を持たせることも今後の課題となる．

謝辞

本報告を終えるにあたり，御指導，御教授いただいた大阪大学大学院情報科学研究科の村田正幸教授に深く感謝致します．また，終始，御指導，御助言を頂いた大阪大学サイバーメディアセンターの馬場建一助教授に深く感謝致します．本報告において，多大な御協力を頂いた日本電信電話株式会社フォトニクス研究所の岡田顕氏，大阪大学大学院情報科学研究科の藤本典幸助教授に心からお礼申し上げます．

また，日頃から適切なご助言を頂いた大阪大学大学院情報科学研究科の若宮直紀助教授，牧一之進助手，大阪大学サイバーメディアセンターの長谷川剛助教授，笹部昌弘助手，大阪大学大学院経済学研究科の荒川伸一助手に心から感謝致します．

また，本報告のためにいろいろとお世話して頂いた中本博久氏，谷口英二氏に厚く御礼申し上げます．最後に，日頃から御協力を頂いた村田研究室および中野研究室のみなさまに心から御礼申し上げます．

参考文献

[1] "Global Grid Forum." available at `http://www.gridforum.org/`.

[2] "The Globus Aliance." available at `http://www.globus.org/`.

[3] M. Murata and K. Kitayama, "A 1,000-channel WDM network can resolve network bottleneck," in *Proceedings of the 7th Asia-Pacific Conference on Communications (APCC 2001) (Tokyo)*, pp. 113–116, Sept. 2001.

[4] E. L. Berger, "Generalized multi-protocol label switching (GMPLS) signaling functional description," *IETF RFC3471*, Jan. 2003.

[5] T. Yamaguchi, K. Baba, M. Murata, and K. Kitayama, "Packet scheduling for WDM fiber delay line buffers in photonic packet switches," in *Proceedings of OptiComm 2002*, vol. 4874, pp. 262–273, July 2002.

[6] K. Baba, R. Takemori, M. Murata, and K. Kitayama, "A packet scheduling algorithm for the 2x2 photonic packet switch with FDL buffers," in *Proceedings of 28th Europian Conference on Optical Communication 2002 (ECOC2002)*, Sept. 2002.

[7] S. L. Danielsen, B. Mikkelsen, C. Joergesen, T. Durhuus, and K. E. Stubkjaer, "WDM packet switch architectures and analysis of the influence of tuneable wavelength converters on the performance," *IEEE Jounal of Lightwave Technology*, vol. 15, pp. 219–227, Feb. 1997.

[8] S. L. Danielsen, C. Joergesen, B. Mikkelsen, and K. E. Stubkjaer,, "Analysis of a WDM packet switch with improved performance under bursty traffic conditions due to tuneable wavelength converters," *IEEE Journal of Lightwave Technology*, vol. 16, pp. 729–735, May 1998.

[9] D. Hunter, M. C. Chia, and I. Andonovic, "Buffering in optical packet switches," *IEEE Journal of Lightwave Technology*, vol. 16, pp. 2081–2094, Dec. 1998.

[10] K. L. Hall and K. A. Rauschenbach, "All-optical buffering of 40-gb/s data packets," in *IEEE Photonic Technology Letters*, vol. 10, pp. 442–444, 1998.

[11] T. Yamaguchi, K. Baba, M. Murata, and K. Kitayama, "Scheduling algorithm with consideration to void space reduction in photonic packet switch," *IEICE Transactions on Communications*, vol. E86-B, pp. 2310–2318, Aug. 2003.

[12] T. DeFanti, M. Brown, J. Leigh, O. Yu, E. He, J. Mambretti, D. Lillethun, and J. Weinberger, "Optical Switching Middleware for the OptIPuter," *IEICE Transaction on Communication*, vol. E86-B, Aug. 2003.

[13] H. Nakamoto, K. Baba, and M. Murata, "Shared memory access method for a λ computing environment," in *Proceeding of IFIP TC6 / WG6.10 First Optical Network and Technologies Conference(OpNeTec)*, pp. 210–217, Oct. 2004.

[14] 谷口 英二, "λコンピューティング環境のための共有メモリシステムの実装と評価," **大阪大学 特別研究報告**, 2004.

[15] E. Taniguchi, K. Baba, and M. Murata, "Implementation and evaluation of shared memory system for establishing lambda computing environment," Tech. Rep. 255, IEICE, Aug. 2004.

[16] Y. Sakai, K. Noguchi, R. Yoshimura, T. Sakamoto, A. Okada, and M. Matsuoka, "Management system for full-mesh WDM AWG–STAR network," in *27th European Conference on Optical Communication, 2001*, vol. 3, pp. 264–265, Sep 2001.

[17] A. Okada, H. Tanobe, and M. Matsuoka, "Dynamically reconfigurable real-time information-sharing network system based on a cyclic-frequency AWG and tunable-wavelength lasers," in *in Proceedings of ECOC2003*, sept 2003.

[18] 岡田顕，田野辺博正，松岡茂登, "波長ルーティング技術を用いたダイナミックに再構成可能な情報共有ネットワーク," **電子情報通信学会技術研究報告** *(IN2003-332)*, vol. 第103巻, no. 692号, pp. 423–427, 2004.

[19] 日本アイ・ビー・エム システムズ・エンジニアリング株式会社, **グリッド・コンピューティングとは何か**. ソフトバンク パブリッシング株式会社, 2004.

[20] 田中 良夫, "JSPP 2002 チュートリアル Globus Toolkit." available at `http://www.jp.grid.org/tech-info/JSPP02-tutorial.ppt`.

[21] "MPICH-G2." available at `http://www3.niu.edu/mpi/`.

[22] 日本電信電話株式会社フォトニクス研究所, **情報共有ネットワークシステム説明書**.

[23] “Himeno benchmark xp - 姫野ベンチとは.” available at http://accc.riken.jp/HPC/HimenoBMT/.