11-IS04

負荷バランスや通信性能が予測困難な状況を想定した集団通信アルゴリズムの動的選択技術に関する研究

黒川　原佳（理化学研究所）

概要
バッチ型で利用する共同利用大規模並列計算機において、充填率を考慮したスケジューリングを行う場合、割り当てられる計算ノード間の距離が実行の度に変化するため、従来の静的な手法では最適な集団通信アルゴリズム選択を行えない。本研究課題では、集団通信アルゴリズムの動的選択技術について、各アルゴリズムの性能予測モデルを用いたアルゴリズム絞り込みによる低オーバヘッドの実装を提案し、効果を検証している。今年度は、昨年度提案したアルゴリズム絞り込みの手法を改良し、動的最適化技術のオーバヘッドを低減できることを確認した。

1. 研究の目的と意義

本課題では、大規模並列計算機上で、プロセスの配置によらず最適な集団通信アルゴリズムを選択する動的最適化技術について研究開発する。並列計算機をバッチ型で利用する場合、利用者から申請されたジョブがジョブスケジューラによって計算ノードに割り当てられる。この時ジョブスケジューラは、予め設定されたスケジューリングポリシーに従って、プロセスの配置を決定する。従来、多くのジョブスケジューラは、計算ノードをジョブクラスと呼ばれる大小の連続ブロックに分割し、それらのジョブクラス単位でプロセスを割り当てていた。これに対し、理化学研究所の RICC(RIKEN Integrated Cluster of Clusters)のメタスケジューラでは、出来るだけ空いている計算ノードが少なくなるように、計算ノードの位置に関係なくプロセスを配置する。その結果ジョブの充填率が向上するため、並列計算機全体のスループットも向上することが期待できる。一方、個々のジョブにおけるプログラム中の通信の性能は、そのジョブに割り当てられた計算ノードの相対的な位置関係によって大きく影響を受ける。例えば計算ノード同士の距離によって通信遅延時間が変動する。さらに、距離が遠い場合、他の通信と経路が競合する通信衝突が発生し、通信帯域幅が低下する可能性が高くなる。

特に、並列プログラムにおける全プロセスによる総和の計算や全対全のコピー等の、集団通信と呼ばれる通信では、この転送性能の低下が大きな問題となる。集団通信には複数の実装アルゴリズムが用意されており、状況に応じて最適なアルゴリズムを選択して使用する。従来は、事前に十分な調査を行うことにより、使用するプロセス数と転送するメッセージサイズから最適なアルゴリズムを選択している。しかし通信性能が安定しない環境では、同じプロセス数、メッセージサイズでも事前の調査時とは最適なアルゴリズムが変化する可能性が高い。そのため、実行時の状況に応じて適応的にアルゴリズムを選択する仕組みが必要となる。

そこで本課題では、集団通信の実装アルゴリズムを実行中に試しながら、最適なものを選択する動的最適化技術の研究開発を行っている。この最適化技術は、プログラム中で繰り返し呼ばれる集団通信について、その最初のほうの繰り返し時に実装アルゴリズムを変えながら実行することにより、実行時の状況における各アルゴリズムの性能を取得し、その情報に基づいて最適なアルゴリズムを選択するものである。昨年度の JHPCN 課題により、RICC においてこの手法が有効であることを示した。また、この手法におけるオーバヘッドを低減するため、選択対象のアルゴリズムを絞り込む技術を取り入れたが、昨年度の段階では十分な効果が得られなかった。そこで本年度は、このア

ルゴリズム絞り込み技術を改良し、RICC で効果を検証する。さらに、実際のアプリケーションで問題となる、負荷バランスによる各アルゴリズムの性能変動への対応についても研究する。

2. 当拠点公募型共同研究として実施した意義

(1) 共同研究を実施した大学名と研究体制

大学名： 九州大学

研究体制：

・黒川原佳（理化学研究所）

大規模並列計算機における集団通信アルゴリズム動的最適化技術の効果検証

・南里豪志（九州大学）

通信性能が不安定な環境での集団通信の動的最適化技術の研究開発

・稲富雄一（九州大学）

並列アプリケーション OpenFMO における集団通信の性能評価

(2) 共同研究分野

大規模情報システム関連研究分野

(3) 当公募型共同研究ならではという事項など

スーパーコンピュータの運用に携わる２つの組織の研究者間で情報や知見を共有でき、実際の計算機利用に即した技術の研究開発を行うことができる。

3. 研究成果の詳細と当初計画の達成状況

(1) 研究成果の詳細について

A) 高並列での OpenFMO の負荷バランス評価

今回の研究で対象アプリケーションとした OpenFMO について、高並列での負荷バランスを計測して評価した。

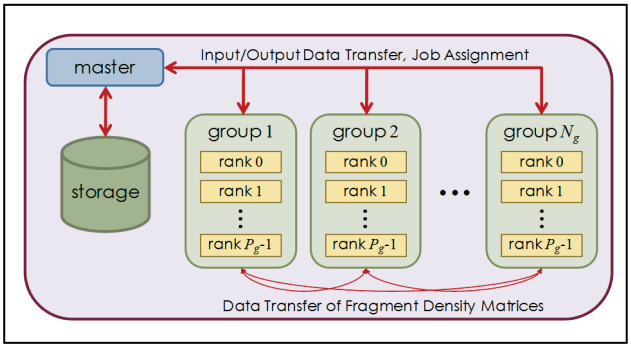

図 1 並列 FMO プログラムの基本構造

フラグメント分子軌道（FMO）法では、大きな分子を 20～40 原子程度の小さなフラグメントに分割して、フラグメントとフラグメントペアに対する小規模な電子状態計算を行うことで、巨大分子全体の電子状態（エネルギー、電荷分布など）を近似する。計算すべきフラグメントやフラグメントペアの数が多いこと、ならびに、各フラグメント（フラグメントペア）に対する電子状態計算を独立に行うことが可能であること、などから、FMO 計算は、マスタープロセスが複数のワーカープロセス群（グループ）に計算すべき小規模電子状態計算を割り当てるマスター - ワーカー型の実行形態に向いている。また、各小規模電子状態計算も並列処理が可能であるため、並列 FMO プログラム OpenFMO は、図 1 に示すように、2 段階並列化した構造を持つ。各グループ内での負荷均等化も重要であるが、今回は、各ワーカーに負荷が均等に割り当てられているかどうかを調べるために、大規模並列実行時のプロファイルを取得した。

使用した計算機は RICC の超並列 PC クラスタ部で、128 並列（32 プロセス、4 スレッド/プロセス）を 1 つのグループとした 63 グループによる並列処理を行った。また、MPI として OpenMPI（version 1.4.3）、コンパイラとして Intel C++ compiler（version 11.0）を用いた。計算対象としたのは、FMO の計算対象として最も需要が高いと思われる大きさ（数 100～1000 フラグメント）を持つアデノウィルス KNOB ドメイン（PDB ID=1nob、フラグメント数 576）で、基底関数は量子化学計算で広く用いられてい

る 6-31G(d)を利用した。この入力に対する計算を行って、グループあたりに割り当てられる小規模電子状態計算の数が少なく、負荷バランスが採りにくいと思われる、Self-Consistent Charge（SCC)ループ処理 1 回分の負荷バランスの様子を調べた。その結果を図 2 に示す。これを見ると、1 回の SCC ループ処理におけるグループ間での負荷のばらつきは小さく、負荷均等化が上手く行えていることが分かった。また、OpenFMO の現状のコードでは、数 10 回繰り返して行う各 SCC ループが終了する度に同期をしているが、複数回の SCC ループを非同期に実行することも可能であるため、非同期実行のコードを組み込めば、さらに精度の高い負荷均等化を行うことができる。

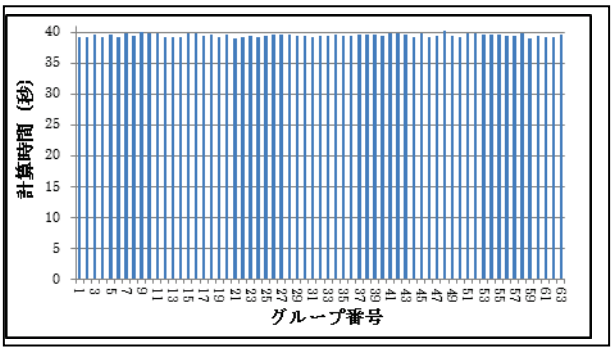

図 2　グループ間の負荷バランスの様子

今回の結果から、OpenFMO を用いた並列 FMO 計算では、各グループに均等に負荷を割り当てられることが明らかとなった。当初、高並列では負荷の割り当てにばらつきが生じて集団通信の性能に影響を与えると予想していたが、OpenFMO に関しては大きな問題とならないことが分かった。そのため本年度は、集団通信アルゴリズムの性能予測精度向上に注力し、負荷バランスを考慮したアルゴリズム選択は、それを必要とするアプリケーション選定を含め、今後の課題とした。

B) 集団通信アルゴリズムの性能へのランク配置の影響

本研究で提案する動的アルゴリズム選択手法では、各集団通信アルゴリズムについて、トポロジとランク配置に応じた性能予測を行う。この、ランク配置を考慮する重要性を検証するため、予備調査として、ランク配置による集団通信アルゴリズムの性能への影響を計測した。

実験で用いるランク配置のパターンは、RICC のインターコネクトネットワークのトポロジを考慮して決定した。RICC のインターコネクトネットワークは 2 階層のスイッチ群による Fat Tree トポロジで構成されており、下位の 52 台の Leaf Switch が上位の 2 台の Upper Switch と 2 本ずつ、合計 4 本のリンクで接続されている。この Leaf Switch に 20 台ずつの計算ノードが接続され、合計で 1024 ノードの PC クラスタを構成している。各ノードには一意に番号が付けられており、0 番目の Leaf Switch には 0～19、1 番目の Leaf Switch には 20～39 というように、Leaf Switch 内に連続した番号のノードが配置されている。一方、各 Leaf Switch から Upper Switch への 4 本のリンクには、0～3 の番号が付けられている。このトポロジにおける Leaf Switch を跨る通信では、Leaf Switch から Upper Switch へのリンクのうち、その通信の宛先ノードの番号を 4 で割った余りの数字と一致する番号のリンクを経由して、目的のノードにメッセージが送信される。

このようなトポロジでは、プロセスが配置されるノードの位置によって各リンクの使用頻度に偏りが生じる。そこで、この偏りによる影響を検証するため、ランク配置として以下の 5 パターンを用い、プロセス数を 64、128、192、256 と変化させ、各 Alltoall アルゴリズムの所要時間を計測した。

0）4 の倍数のノード番号のみにプロセスを配置
Leaf Switch と Upper Switch の間のリンクを、Leaf Switch あたり 1 本のみ使用する。各 Leaf Switch 内のプロセス数は、192 プロセスまでは 4、256 プロセスでは Leaf Switch 数が不足するので 5 とし、ノード番号の小さいものから順に 4 の倍数の番号を持つノードに配置する。

1）2 の倍数のノード番号のみにプロセスを配置
Leaf Switch と Upper Switch の間のリンクを、

Leaf Switch あたり 2 本使用する。各 Leaf Switch 内のプロセス数は、192 プロセスまでは 4、256 プロセスでは 5 とし、ノード番号の小さいものから順に 2 の倍数の番号を持つノードに配置する。

2）各 Leaf Switch の最初の 4 ノードにプロセスを配置

Leaf Switch と Upper Switch の間のリンクを、Leaf Switch あたり 4 本使用する。256 プロセスでは、各 Leaf Switch の最初の 5 ノードにプロセスを配置する。

3）各 Leaf Switch の最初の 8 ノードにプロセスを配置

Leaf Switch と Upper Switch の間のリンクを、Leaf Switch あたり 4 本使用する。パターン 2）に対して使用 Leaf Switch 数が半分となり、リンクあたりプロセス数が 2 倍となる。

4）各 Leaf Switch の最初の 16 ノードにプロセスを配置

Leaf Switch と Upper Switch の間のリンクを、Leaf Switch あたり 4 本使用する。パターン 2）に対して使用 Leaf Switch 数が 1/4 となり、リンクあたりプロセス数が 4 倍となる。

メッセージサイズを 16KB とし、各プロセス数で各アルゴリズムの所要時間を計測した結果を、図 3～図 6 に示す。各図で横軸はプロセス配置パターンの番号、縦軸は所要時間である。また、それぞれの線が、各アルゴリズムの所要時間である。

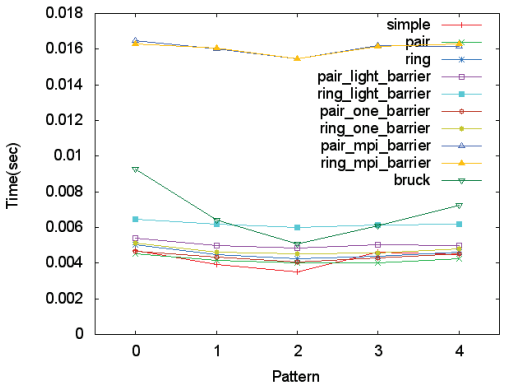

図 3 プロセス配置パターン毎の各アルゴリズムの所要時間（64 プロセス、16KB）

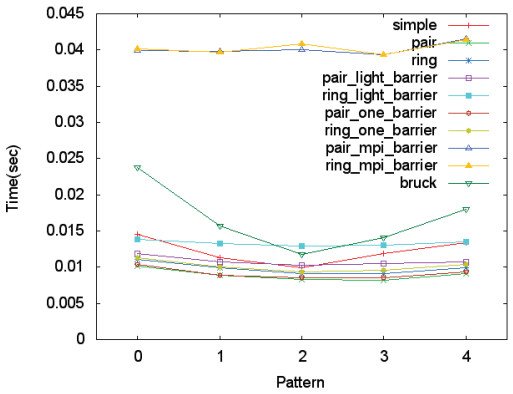

図 4 プロセス配置パターン毎の各アルゴリズムの所要時間（128 プロセス、16KB）

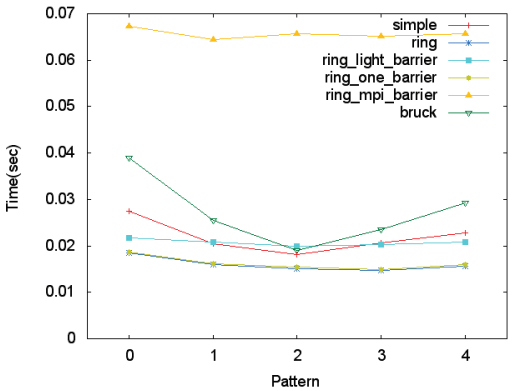

図 5 プロセス配置パターン毎の各アルゴリズムの所要時間（192 プロセス、16KB）

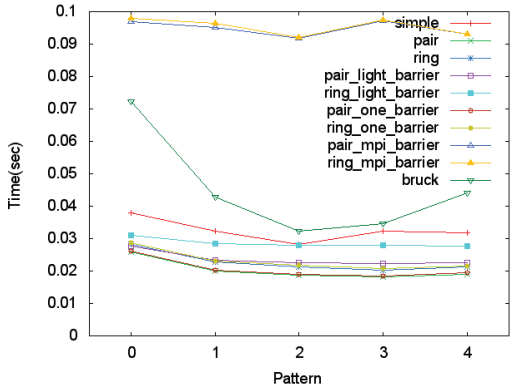

図 6　プロセス配置パターン毎の各アルゴリズムの所要時間（256 プロセス、16KB）

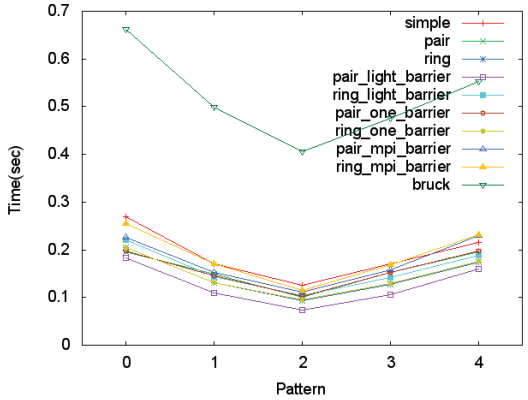

図 7　プロセス配置パターン毎の各アルゴリズムの所要時間（64 プロセス、1MB）

図より、16KB では bruck のパターン毎の性能の変動が大きいことが分かる。パターン 0 やパターン 4 では最速のアルゴリズムに対して 2 倍以上の時間を要しているのに対し、パターン 2 では最速のアルゴリズムに近い時間となっている。これは、プロセス数 P に対して、他のアルゴリズムが 1 回あたり 16KB の一対一通信を P-1 回繰り返して Alltoall 通信を実現しているのに対し bruck は 1 回あたり 16KB*P/2 の一対一通信を $\log_2 P$ 回繰り返しているため、プロセス配置パターンによるバンド幅の変化に影響されたためである。なお、リンクあたりのプロセス数が同じであるパターン 0 と 4 やパターン 1 と 3 で所要時間が異なるのは、パターン 3 と 4 は、パターン 0 と 1 に比べ、同じ Leaf Switch 内に配置されるプロセス数が多く、Leaf Switch をまたぐ通信の数が少ないためである。

一方、メッセージサイズが 1MB の場合の 64 プロセスでの各アルゴリズムの所要時間を図 5 に示す。このように、メッセージサイズが大きくなると、他のアルゴリズムでもプロセス配置パターンによって性能の変動が見られるようになる。

この調査により、プロセスが割り当てられたノードの位置によって有効なバンド幅が変動し、その影響でアルゴリズムの優劣が変化することが確認できた。

C) ランク配置に応じた集団通信アルゴリズム選択技術の提案と評価

前項の調査により、プロセスが割り当てられたノードの位置によって有効なバンド幅が変動し、その影響でアルゴリズムの優劣が変化することが確認できた。この結果を受け、プロセスが割り当てられたノードの位置とトポロジを考慮したアルゴリズム選択技術を提案する。

本手法では、まずノードの配置とトポロジの情報に基づいて候補アルゴリズムを絞り込み、その後、各候補アルゴリズムを実際に試して最速のものを選択する。アルゴリズムの絞り込みには、各アルゴリズムの性能モデルを用いる。

本稿では、提案手法の実装例として、RICC のトポロジを対象とした動的アルゴリズム選択による Alltoall 通信関数を実装する。この関数は、実行時のランク配置とトポロジ情報にもとづいて各 Alltoall アルゴリズムの所要時間を予測し、候補アルゴリズムを絞り込む。

各アルゴリズムの性能予測には、性能モデルを用いる。基本的に Alltoall 通信は、参加する全プロセス間で全対全通信を行う。そこで今回作成する Alltoall 通信関数では、全プロセスが

同時に通信を行う際の平均帯域幅を計算し、その値を用いて各アルゴリズムの性能を予測する。この平均帯域幅は、あるプロセスと他の全プロセスの間で通信を行う際の個々の通信の帯域幅の平均値とする。ここで個々の通信の帯域幅としては、それぞれの通信時に経由する各リンクの基本帯域幅を、そのリンクを同時に使用する他のプロセスによる通信の数の期待値で割った値を用いる。今回対象とする RICC のトポロジでは、Leaf Switch と Upper Switch の間のリンクにおいて、同時に使用するプロセス数が方向によって異なる。Leaf Switch から Upper Switch への方向は、その Leaf Switch から他の Leaf Switch への通信のうち、宛先ノード番号を 4 で割った値がそのリンク番号と一致するものが使用する。一方 Upper Switch から Leaf Switch への方向は、他の Leaf Switch からこの Leaf Switch への通信のうち、宛先ノード番号を 4 で割った値がそのリンク番号と一致するものが使用する。このうち、今回は、Upper Switch から Leaf Switch への方向の平均帯域幅を用いる。この平均帯域幅は Leaf Switch と Upper Switch の間のリンク毎に変動するため、全てのリンクについて計算した後、全リンクで最も小さい値を、システム全体の通信を律速する平均帯域幅として用いる。

次に、各リンクの平均帯域幅の算出方法を示す。ここで、ノード数を Nnode、ノード内プロセス数を Pn、i 番目の Leaf Switch を LSi、i 番目の Leaf Switch 内と Upper Switch の間のリンクのうち j 番目のものを ULij、プログラムが使用するノードのうち LSi に配置されたものの数を Ni、そのうちノード番号を 4 で割った値が j であるものを NLij とする。また、今回はノード内、Leaf Switch 内、Leaf Switch 間、それぞれ基準となる帯域幅は一定値 B として、平均帯域幅を計算する。

まず、LSi のノードの一つに割り当てられたプロセスが他の全プロセスから受信する場合の各通信の帯域幅を見積もる。ノード内のプロセスから受信する場合の帯域幅は、他の通信に妨げられないと仮定し、B のままとする。一方、Leaf Switch 内のノード間通信は、Leaf Switch からノードへの 1 本のリンクをノード内の全プロセスの受信で共有するため、平均帯域幅は B/Pn とする。また、Leaf Switch を跨ぐ通信の受信では、同じ Leaf Switch 内の各プロセスの通信のうち、同じリンクを経由して他の Leaf Switch から受信するものが 1 本のリンクを共有する。このプロセスが配置されたノードの番号を 4 で割った余りが j である場合 ULij を使って受信するので、同時に同じリンクを使って受信する LSi 内のプロセス数の期待値 RCVij は以下で計算できる。

$$RCVij = 1 + (NULij * Pn - 1) * (Nnode - Ni) * Pn / (Nnode * maxn - 1)$$

これらより、ULij を経由して受信する LSi 内のプロセスについて平均帯域幅 BRij を、以下で計算する。

$$BSij = B/((Pn - 1 + (Ni-1) * Pn * Pn + (Nnode - Ni) * RCVij) / (Nnode * Pn - 1))$$

これを、0≦i<使用スイッチ数、0≦j<4、の各 i、 j で計算し、その最小値をシステムの平均帯域幅とする。

各アルゴリズムの性能は、上記で得られた平均帯域幅をそれぞれの性能モデルに適用して予測する。今回の実装に用いた Alltoall の各アルゴリズムの性能モデルを以下に示す。

Algorithm	Model
Simple Spread	(P-1)*L+(P-1)*M*B
Ring	(P-1)*(L+M*B)
Ring with One Barrier	(P-1)*(L+M*B)+(L*$\log_2$P)
Ring with MPI_Barrier	(P-1)*(L+M*B)+2*(L*(P-1)*$\log_2$P)
Ring with Light Barrier	(P-1)*(L+M*B)+L*(P-1)
Pair	(P-1)*(L+M*B)
Pair with One Barrier	(P-1)*(L+M*B)+(L*$\log_2$P)
Pair with MPI_Barrier	(P-1)*(L+M*B)+2*(L*(P-1)*$\log_2$P)
Pair with Light Barrier	(P-1)*(L+M*B)+L*(P-1)
Bruck	L*$\log_2$P+P*M*B*$\log_2$P/2

これらのモデルは、Hockney モデルによる一対一通信の性能モデルをもとにしている。Hockney モデルでは、個々の一対一通信の所要時間を、遅延時間 L とバイト当たりの所要時間 B、すなわち帯域幅の逆数を用いて L + M*B と表す。こ

れを、各アルゴリズム内の一対一通信に適用して、性能モデルを作成した。

このモデル自体は、通信の衝突による影響が考慮されていない。しかし、前述の通り帯域幅をプロセスの配置に応じて調整することにより、衝突の影響を加味した所要時間を見積もることができる。

なお、一対一通信の性能モデルとしては、他にLogGPやPLogP{Parameterized Log-P)などが提案されている。特にPLogPは、メッセージサイズの変化に伴う実効帯域幅の変動を表現でき、より高い精度での性能予測が行えると期待できるため、今後適用を検討する。

今回作成したAlltoall通信関数の実装では、まず事前に基準となる帯域幅と遅延時間を計測した。これは、2プロセス間の一対一通信を繰り返して計測した結果を用いた。

また、トポロジ情報は事前に調査し、その結果を性能予測手法に反映させた。具体的には、Upper SwitchとLeaf Switchの間のリンクの構成、およびLeaf Switchにまたがる通信の経路選択ポリシーを調べ、それに基づいて帯域幅を調整する性能予測モデルを作成した。

一方、ランク配置情報の取得は、Alltoall関数の最初の呼び出し時に行う。これを実行開始時に行うこともできるが、今回はAlltoall通信関数の試験実装が目的であったため、この関数内で初回呼び出し時にランク配置情報の取得を行っている。

このランク配置情報と、事前に作成していた性能予測を用いて、最初の呼び出し時にアルゴリズムを絞り込む。今回の実装では、用意されているアルゴリズムのうち、最も所要時間が短いと予測されたものに対して、2倍以上の所要時間がかかると予測されたアルゴリズムを、アルゴリズム選択の候補から除外する。なお、このアルゴリズムを除外する閾値は、今後、システムの性能安定性や性能予測モデルの精度を確認しながら調整する予定である。

候補アルゴリズムを絞り込んだ後の最速アルゴリズムの選択にはSTAR-MPIを用いる。STAR-MPIの処理は、以下の2つのフェーズに分けて行われる。

Learningフェーズ：
集団通信が呼ばれると、選択候補のアルゴリズムのうちの一つを用いて実行し、結果を返す。その際所要時間を計測し、記録する。全ての候補アルゴリズムについて規定回数の計測が終了するまで、各集団通信呼び出しに対してこのフェーズを実行する。全ての候補アルゴリズムの計測が完了すると、それらの所要時間の記録から最も高速なアルゴリズムを選出し、次のMonitoringフェーズで使用するアルゴリズムとする。なお、アルゴリズムの選出に用いる各アルゴリズムの所要時間としては、全プロセスの所要時間の平均値を用いる。

Monitoringフェーズ：
Learningフェーズで選出されたアルゴリズムを用いて集団通信を行う。このフェーズはLearningフェーズが終了してアルゴリズムが選択された後、各集団呼び出しに対して実行する。実際の計算環境では実行中に状況が大きく変化することも考えられるため、定期的に集団通信の所要時間とLearningフェーズで得られた所要時間を比較する。もし、計測した時間がLearningフェーズ時の所要時間から大きく変動した場合、他のアルゴリズムの方が高速となった可能性があるので、再度Learningフェーズに入り、アルゴリズムを選択する。

今回実装した、動的アルゴリズム選択を行うAlltoall通信関数のRICCにおける性能を検証するため、Alltoall通信を200回呼び出すプログラムを用いて所要時間の計測を行った。このプログラムはSTAR-MPIのベンチマークプログラムとして提供されているものである。

このプログラムを、プロセス数とメッセージ

サイズを変えながら RICC のメタジョブスケジューラに投入した。RICC において用いられているメタスケジューラは、出来るだけ空いている計算ノードが少なくなるようにジョブを割り当てて、システムにおけるジョブの充填率を上げることにより、総合的な処理速度の向上を図っている。そのため、同じプロセス数のジョブであっても、割り当てられるノードの位置に規則性が無い。その結果、通信遅延の変動や通信衝突の影響により、通信性能が不安定となる。従来の MPI ライブラリの集団通信関数で用いられている静的なアルゴリズム選択手法では、このような環境では最適なアルゴリズム選択ができない。一方、提案手法や STAR-MPI では、実際に計算機上でアルゴリズムを試すため、状況に応じたアルゴリズム選択を行うことができる。

図 6～図 9 に、メッセージサイズが 64KB のときの、1 プロセス x 32 ノード、2 プロセス x 32 ノード、4 プロセス x 32 ノード、8 プロセス x 32 ノードのそれぞれのプロセス数での Alltoall 通信の平均所要時間を示す。X 軸はジョブの番号、Y 軸は所要時間である。示されている所要時間は、アルゴリズムを絞り込んだ動的アルゴリズム選択(DYN GROUP)、アルゴリズムを絞り込まない動的アルゴリズム選択(DYN NOGROUP)、および、Bruck(Bruck)、Pairwise(Pair)、Pairwise with Light-Barrier(PairLB)、Pairwise with MPI-Barrier(PairMB)、Pairwise with One-Barrier(PairOB)、Ring(Ring)、Ring with Light-Barrier(RingLB)、Ring with MPI-Barrier(RingMB)、Ring with One-Barrier(RingOB)の各アルゴリズムの所要時間である。

これらの結果より、提案手法によってほぼ最適なアルゴリズムが選択され、常に最速に近い性能が得られていることが分かる。また、アルゴリズムを絞り込まない場合との比較より、アルゴリズムを絞り込むことによる性能向上が得られることが分かる。

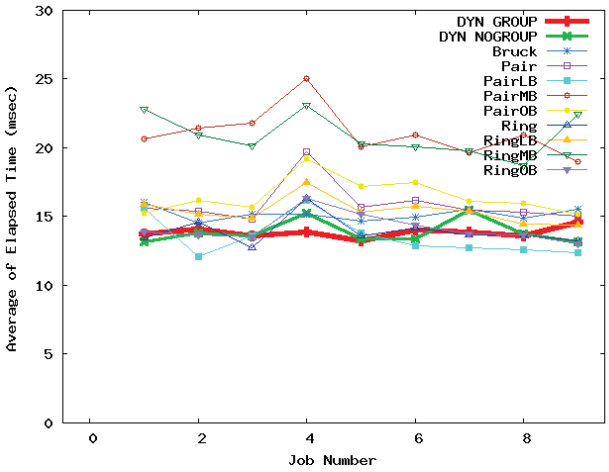

図 8　Alltoall 通信の平均所要時間
64KB、1 プロセス x 32 ノード

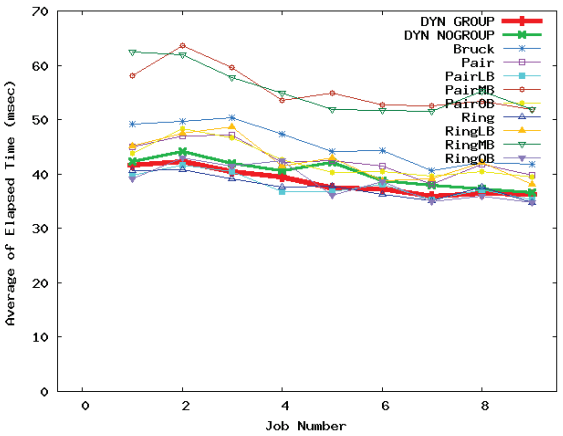

図 9　Alltoall 通信の平均所要時間
64KB、2 プロセス x 32 ノード

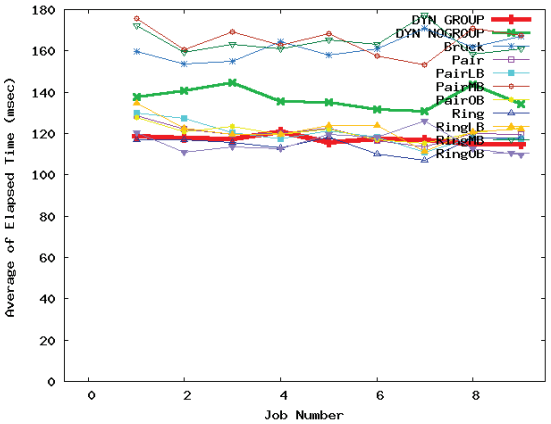

図 10　Alltoall 通信の平均所要時間
64KB、4 プロセス x 32 ノード

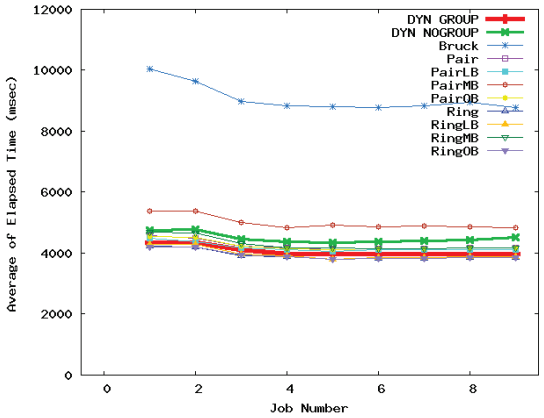

図 11　Alltoall 通信の平均所要時間
64KB、8 プロセス x 32 ノード

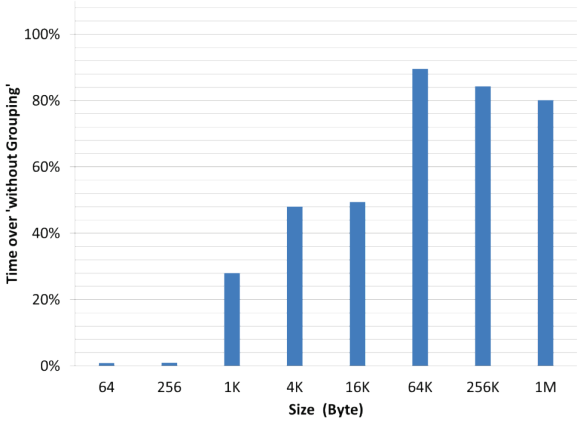

図 13　アルゴリズム絞り込みを行わない場合に対する所要時間の比率（Learning フェーズ）
2 プロセス x 32 ノード

さらに、アルゴリズムを絞り込むことによる効果を検証するため、Learning フェーズの所要時間の比率を図 10～図 13 に示す。ほぼすべての場合で、アルゴリズムを絞り込むことによって Learning フェーズの所要時間を短縮できている。

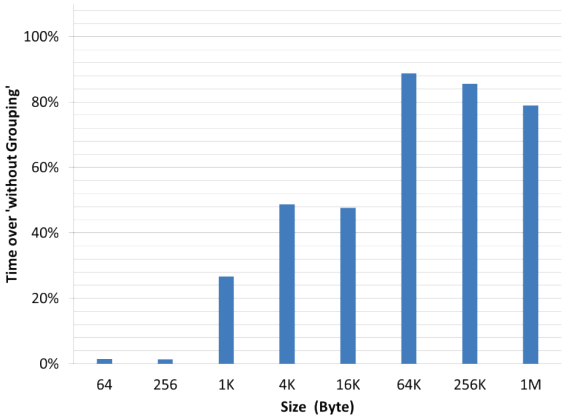

図 12　アルゴリズム絞り込みを行わない場合に対する所要時間の比率（Learning フェーズ）
1 プロセス x 32 ノード

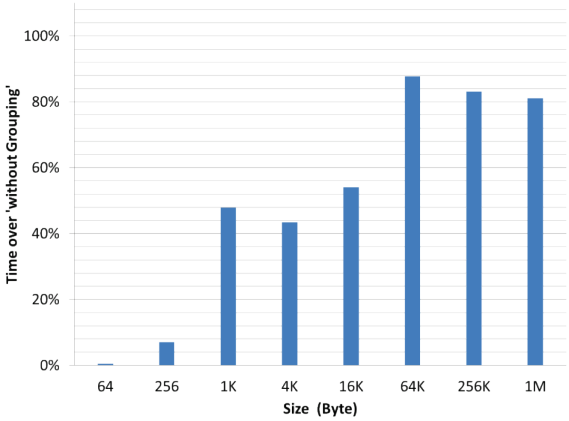

図 14　アルゴリズム絞り込みを行わない場合に対する所要時間の比率（Learning フェーズ）
4 プロセス x 32 ノード

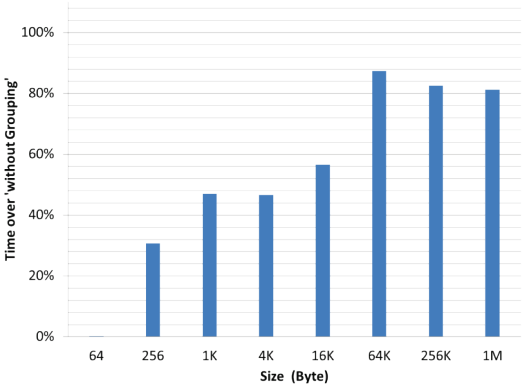

図 15　アルゴリズム絞り込みを行わない場合に対する所要時間の比率（Learning フェーズ）
8 プロセス x 32 ノード

(2) 当初計画の達成状況について

本年度は、動的アルゴリズム選択技術のオーバヘッド低減を目的とし、効率の良いアルゴリズム絞り込みに向けて調査および研究を行った。調査により、ランク配置がアルゴリズム性能の優劣に大きく影響することを明らかにした。さらに、ランク配置を考慮したアルゴリズム性能予測技術を提案し、この技術を用いた遅いアルゴリズムを選択肢から排除することによって、動的アルゴリズム選択におけるオーバヘッドの低減を実現できた。

なお、もう一つの目的であったアプリケーションの負荷バランスに応じたアルゴリズム選択技術については、対象アプリケーションとした OpenFMO が、予想に反して高並列でも良好な負荷バランスを示したため、アルゴリズムの再選定が必要となり、今後の課題とした。

4. 今後の展望

これから導入事例が増えると予想される多次元トーラスや Dragon Fly によるインターコネクトに向けて提案手法を改良するとともに、他の集団通信への適用も予定している。

5. 研究成果リスト

(1) 学術論文（投稿中のものは「投稿中」と明記）

(2) 国際会議プロシーディングス

(3) 国際会議発表

Takeshi Nanri and Motoyoshi Kurokawa, ″Effect of Dynamic Algorithm Selection of Alltoall Communication on Environments with Unstable Network Speed″, International Workshop on Autonomic and High Performance Computing 2011, Jul 2011.

Takeshi Nanri and Motoyoshi Kurokawa, ″Efficient Runtime Algorithm Selection of Collective Communication with Topology-Based PerformanceModels″, The 2012 International Conference on Parallel and Distributed Processing Techniques and Applications, Jul 2012(To appear).

(4) 国内会議発表

南里豪志、黒川原佳、“ランク配置に応じた集団通信アルゴリズム動的選択技術の提案”，第 133 回ハイパフォーマンスコンピューティング研究会、2012 年 3 月.

南里豪志、“通信ライブラリにおける実行時自動チューニング技術”、第 3 回自動チューニング技術の現状と応用に関するシンポジウム、招待講演、2011 年 12 月 5 日

(5) その他（特許，プレス発表，著書等）