様式Ｃ－１９

科学研究費助成事業（科学研究費補助金）研究成果報告書

平成２４年　５月２５日現在

機関番号：１７１０２
研究種目：若手研究（B）
研究期間：2009～2011
課題番号：21700036
研究課題名（和文）　並列言語 CAF 向け動的通信最適化技術の開発

研究課題名（英文）　Runtime Optimization Technologies for Communications of CAF

研究代表者
南里　豪志（NANRI TAKESHI）
九州大学・情報基盤研究開発センター・准教授
研究者番号：70284578

研究成果の概要（和文）：
並列コンピュータ上で動作する並列プログラムでは、別々に動作する複数のプロセス同士が通信を行いながら実行を進める。この通信の手順について、実行中の状況に応じて最適なものを選択することにより所要時間を減らし、効率の良い並列処理を可能とする動的通信最適化技術について提案、実装した。実験の結果、提案手法によって効率よく最適な手法が選択できていることを確認した。

研究成果の概要（英文）：
Parallel programs on parallel computers invoke communications among processes to progress in their executions. In this project, a technique that choose the appropriate algorithm of such communications according to the status of the program at runtime is introduced, so that it can reduce the time for communication and enable efficient execution of the programs. Results of experiments showed the effectiveness of the proposed technique.

交付決定額

（金額単位：円）

	直接経費	間接経費	合　計
２００９年度	1400000	420000	1820000
２０１０年度	700000	210000	910000
２０１１年度	700000	210000	910000
年度			
年度			
総　計	2800000	840000	3640000

研究分野：情報通信
科研費の分科・細目：情報学・ソフトウェア
キーワード：並列処理、動的最適化

１．研究開始当初の背景

並列プログラムの実行効率を向上させるための重要な技術として,通信最適化に関する技術が広く研究されてきた.

しかしながら, 計算機が大規模化, 複雑化するにつれて,プログラム実行前の静的な情報だけでは十分な最適化が行えないことが分かってきた.

これに対し, 2006 年に米国フロリダ州立大学の Faraj らが, MPI プログラムについて, 実行中の性能情報から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を開発した.

一方, MPI より抽象度の高い並列プログラミング手法として, 近年 CAF や UPC といった並列言語が提案されている.しかし,

これらの並列言語まで対象とした動的通信最適化技術はほとんど研究されていない。

２．研究の目的

並列プログラムにおける通信を、実行時の状況に応じて動的に最適化する技術を研究開発することにより、効率の良い並列処理を実現する。

３．研究の方法

まず、並列プログラムで頻繁に用いられる通信パターンを解析する。次に、その通信パターンを効率的に実装するアルゴリズムを、プロセス数やメッセージサイズ、プロセスの計算機上での配置などに応じて用意する。それらのアルゴリズムについて、実行時の状況に応じて自動的に選択する動的最適化技術を開発し、実験によって効果を検証する。

４．研究成果

並列プログラムで用いられている通信パターンについて調査したところ、Broadcast や Reduction のように、全部のプロセスが参加してデータのコピーや集約を行う通信が頻繁に用いられており、性能にも大きく影響することが分かった。そこで、これらの通信のうちでも最も通信量が多い Alltoall と呼ばれる通信パターンについて、実行時にアルゴリズムを動的に選択する技術を研究開発した。

本研究で提案する動的アルゴリズム選択手法では、各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、ランク配置によるアルゴリズムの性能への影響を計測した。

この計測の環境として、理化学研究所の RICC(RIKEN Integrated Cluster of Clusters)を用いた。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、プロセス数が 128、256 のそれぞれの場合について、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１、２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。

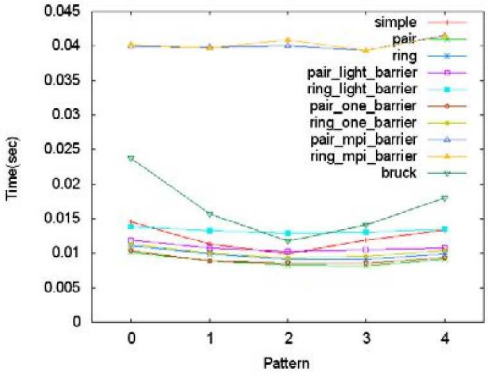

図 1　プロセス配置パターン毎の各アルゴリズムの所要時間（128 プロセス、16KB）

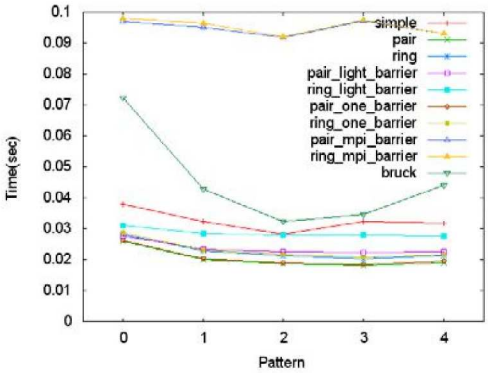

図 2　プロセス配置パターン毎の各アルゴリズムの所要時間（256 プロセス、16KB）

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。

A)　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 Nij 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 + (NLij * 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３、４に、メッセージサイズが 64KB のときの、2 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(DYNNOGROUP)、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。

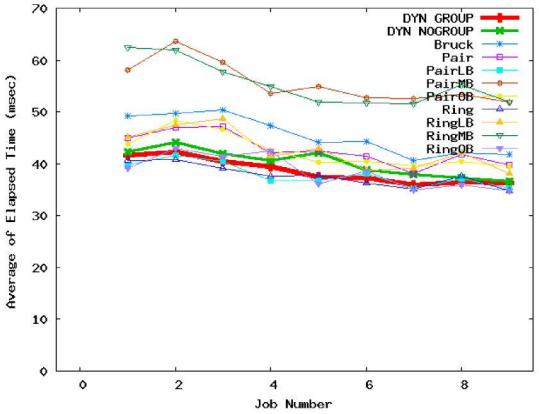

図 3　Alltoall 通信の平均所要時間

(64KB、2 プロセス x 32 ノード)

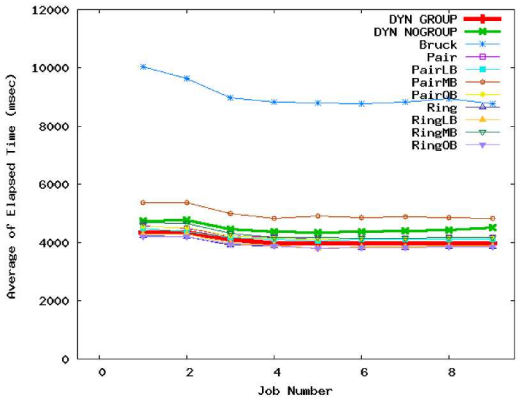

図 4　Alltoall 通信の平均所要時間

(64KB、8 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５、６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。

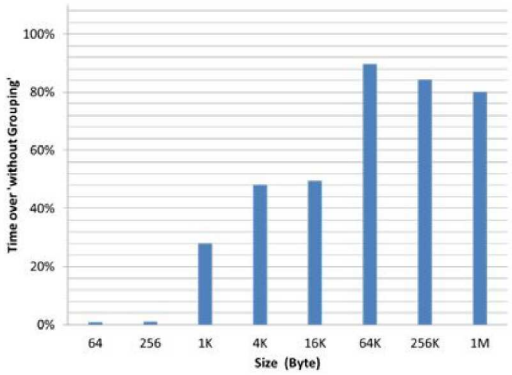

図 5　アルゴリズム絞り込みを行わない場合に対する所要時間の比率（Learning フェーズ）2 プロセス x 32 ノード

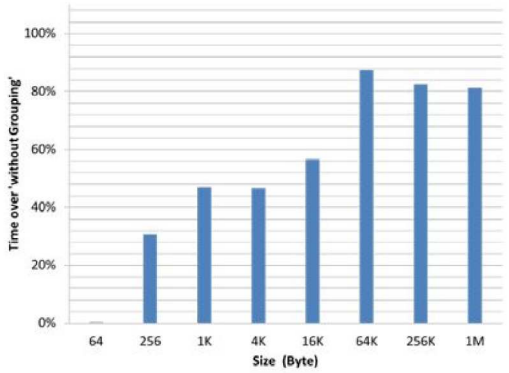

図 6　アルゴリズム絞り込みを行わない場合に対する所要時間の比率（Learning フェーズ）8 プロセス x 32 ノード

これらの結果より、本研究で提案した手法が実行時の状況に応じて最適なアルゴリズムを効率よく選択できていることが確認できた。このように、プログラムの実行前の情報および実行中の情報を活用した通信アルゴリズムの最適化技術は国内外でも事例が少ない。今後、本手法によって大規模な並列計算機における通信の効率化が図れると期待できる。

５．主な発表論文等
（研究代表者、研究分担者及び連携研究者には下線）

〔学会発表〕（計　8 件）

[1] 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).

[2] 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.

[3] Takeshi Nanri and Motoyoshi Kurokawa, Effect of Dynamic Algorithm Selection of All-to-All Communication on Environments with Unstable Network Speed, 2011 International Conference on High Performance Computing & Simulation, 2011.07.

[4] Yoshiyuki Morie, Takeshi Nanri, Ryutaro Susukita and Koji Inoue, "A Method for Predicting a Penalty of Contentions by Considering Priorities of Routing among Packets on Direct Interconnection Network", in Proceedings of the International Joing Conference on Computational Sciences and Optimization 2011, 2011.04.

[5] Yoshiyuki Morie, Takeshi Nanri and Motoyoshi Kurokawa, Task Allocation Method for Avoiding Contentions by the Information of Concurrent Communications, Proceedings of IASTED PDCN 2011, 2011.02.

[6] Kenichiro Kusaba, Takeshi Nanri and Seiji Fujino, Runtime Load-balancing Technique for Sparse Matrix-Vector Multiplication, International Workshop on Innovative Architecture, 2010.

[7] Hyacinthe Nzigou Mamadou, Takeshi Nanri, and Kazuaki Murakami, A Robust Dynamic Optimization for MPI Alltoall Operation, 18th International Heterogeneity in Computing Workshop, 2009.

[8] Hyacinthe Nzigou Mamadou, Feng Long Gu, Vivien Oddou, Takeshi Nanri, Kazuaki Murakami, A Dynamic Solution for Efficient MPI Collective Communications, International Workshop on HPC and Grid Applications ,2009.

６．研究組織
(1)研究代表者
南里　豪志（TAKESHI NANRI）
九州大学・情報基盤研究開発センター・准教授
研究者番号：70284578