領域分割法におけるサブドメインのノード内最適自動マッピング

伊東　聰[†]　後藤和哉[††]　小野謙二[†]

各計算機ノードがマルチコアプロセッサを複数搭載した階層的な並列環境において，異なる通信経路による通信性能の違いと，領域分割型の並列計算における通信パターンを考慮し，サブドメインのCPU/コアへのマッピングを通信時間が最小になるよう最適化した．Intel Xeon 大規模 PC クラスタにおけるベンチマークテストの結果，通信時間の短縮を確認した．また，通信速度を実測したところ，異なる通信経路での通信性能の違いが確認された．今回利用したテスト環境はノード間の理論的通信性能は均一であるが，多数のジョブが同時実行されている環境での性能測定である．このことから，通信距離が均一な環境であっても，集合運用時において本手法の有効性が示された．

Automatically optimized core mapping for subdomains of domain decomposition methods on multicore parallel environment

Satoshi Ito,[†] Kazuya Goto[††] and Kenji Ono[†]

On hierarchical parallel environment with multicore processors, mapping of subdomains on CPU/cores were optimized considering the difference of communication speed of different communication paths and the communication pattern of parallel applications based on domain decomposition methods. We tested our method on massively paralleled Intel Xeon PC cluster and confirmed that it reduced communication time in some benchmark tests.

1. 緒　　言

連続体シミュレーションにおいて主流である領域分割型並列計算では，領域間通信のコストが並列効率を低下させる主要因である．通信量の最小化と均一化に関しては，metis に代表されるグラフ分割ツール[1)2)]を利用した効果的な分割法が提案されている．

ところが，マルチコアに代表されるようにプロセッサが多階層化されたため，階層間での通信性能に著しい格差が生じている．加えて，Cray 社製スーパーコンピュータ Jaguar[3)] や京速コンピュータ「京」[4)] など，超大規模計算機では全ノード/プロセッサが均一な通信距離を確保できないネットワーク構成をとる傾向にある．

また，マルチコア環境において各コアにメモリとコントローラが付随する共有メモリ型アーキテクチャ（NUMA など）では，スレッドコンテキストの切り替えにより自コアで利用するデータが付随するメモリ上に存在しない可能性がある．このような環境では，スレッドバインディングを用いてプロセスが初期割り付けコアから移動することを抑制することができ，計算性能向上が見込めることが知られている．

このように，計算機アーキテクチャの主流が多階層化・複雑化している現在，高効率な並列計算を実現するためには，CPU，メモリの階層構造やそれらを結ぶバスのバンド幅などに合わせた並列計算の最適化が不可欠である．また，大規模計算機は集合運用されることが一般的であり，バッチシステムを用いて多数のジョブがスケジューリングされ，同時実行される．このような環境では実行環境（割り当てノードやその配置など）は計算実行時まで不明であり，他ユーザとの競合状況により実行時の通信性能に大きな差が生じる．

このような背景から，本研究では計算実行時に割り当てられた計算機リソースの情報（プロセッサ階層構造，通信性能など）を取得し，リソースに合わせた部分領域の最適マッピング方法を開発・検証する．本研究ではその第１段階として，全計算ノード間の通信距離が均一な場合について，各ノードへの部分領域割り当てについて検討する．

† 独立行政法人　理化学研究所　次世代計算科学研究開発プログラム
Computational Science Research Program, RIKEN.
†† 株式会社ヴァイナス　プロジェクト推進部
Project Development Dept., VINAS Co., Ltd.

2. 計算領域のコアへの最適自動マッピング

グラフ分割ツールによって分割された計算領域には，領域ごとにユニークに決まる番号が割り振られる．MPI を用いた領域分割型並列計算では，MPI ランクと同一番号の計算領域を担当させることが一般的である．しかし，この領域番号は隣接する領域間で近い値をとるとは限らないため，場合によっては通信距離の遠いコアに配置される可能性がある．

サブドメイン（部分領域）の最適配置を行うためには（1）各ノードに対してコア数分の“隣接する”サブドメインの割り当て，(2) コア配置に合わせたサブドメインの割り当ての 2 段階の配置が必要である．

ここでは以下の方針に従って最適配置を行う．まず利用する計算機のハードウェアリソース（次節参照）を自動的に取得し，各コアの接続状態を示すグラフを作成する．次に，サブドメインの隣接関係を示すグラフを作成し，グラフ分割ソフトを用いてサブドメインの分割（グループ化）を行う．最後に，グループ化されたサブドメインをコア接続グラフを用いて最適配置する．

2.1 ハードウェアリソースの取得

様々なアーキテクチャと OS 環境においてハードウェアリソースの情報を取得するために，本研究では Portable Hardware Locality （hwloc）[5] を利用する．hwloc は OpenMPI プロジェクト内のサブプロジェクトの一つであり，マルチコア環境の階層的ハードウェア構成を幅広い OS，アーキテクチャにおいて取得可能である．hwloc で取得できるハードウェア構成としては以下の物が含まれる．

- NUMA 型メモリのノード構成とサイズ
- プロセッサソケット
- プロセッサコア
- プロセッサスレッド（ハイパースレッディング CPU）

2.2 プロセッサ接続グラフの構築

hwloc の API を通じてハードウェア情報を取得すると，内部では図 **1** に示すような階層的ハードウェア構成を表現するツリー状のデータ構造が構築される．この図の最も上のレベルがマシンを表しており，最も下のレベルが実際にプロセスを割り当てる論理プロセッサである．ここから得られる論理プロセッサ同士の論理的距離に応じて分割された計算領域を割り当てれば，並列計算を効率的に実行できる．本研究では論理プロセッサの距離をプロセッサ接続グラフの重みとして表現する．具体的には以下の通りである．

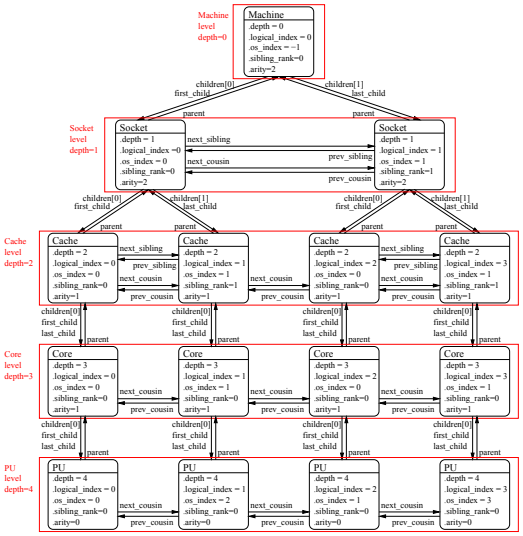

図 **1** hwloc 内部の階層的ハードウェア構成の表現（5) より引用）．
Fig. 1 Diagram of hierarchical hardware configuration (from 5)).

論理プロセッサを頂点とし，1 台の計算機ノードに含まれる全プロセッサ数を頂点数とする完全グラフ（全頂点が互いに接続されているグラフ）を作成する．次にエッジに付与する重みを決定する．hwloc が与えるツリーデータ構造において，最上位のシステムをゼロとし階層が下がるに従って 1,2,…と増えていく深さ d を定義する．ノード内の二つの論理プロセッサは，ツリーデータ構造の階層を遡って行くことで必ず共通の親を見つけることができる．共通の親が下の階層であるほど，この二つの論理プロセッサの距離は近くなる．そこで，この二つのプロセッサ間の重みを共通の親の深さ d を用いて 10^d とする．

2.3 領域接続グラフの作成

計算領域の割り当てを決めるにあたっては，

- 各計算機ノードへの計算領域の割り振り
- 各計算機ノード内での計算領域の論理プロセッサへの割り付け

の 2 段階の作業が必要である．ここで重要なことは，相対的に遅い通信経路を用いる通信，すなわちノードをまたいだ通信量を小さく抑えることである．これは領域分割法における通信量最小化と全く同じ問題である．よって，グラフ分割により解決可能である．ここでは計算領域を頂点とし，通信を必要とする領域同士をエッジで接続する．さらに，各エッジには通信量に応じた重みを付与する（図 **2**）．

2.4 計算領域のグループ化とコアへのマッピング

2.3 節で作成した領域接続グラフを用いて分割された計算領域をグループ化し，各計算ノードへ割り当

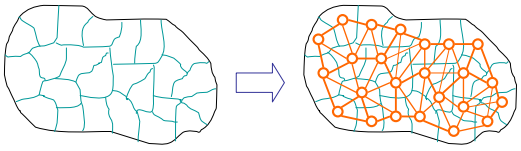

図 2　領域分割された計算領域および領域接続グラフ.
Fig. 2　Decomposed alanysis domain and domain connectivity graph.

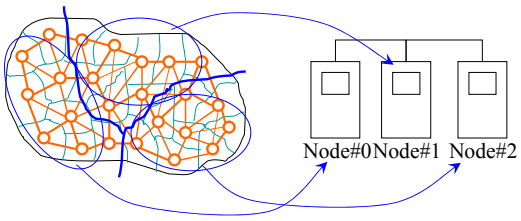

図 3　領域接続グラフの分割および計算機ノードへの割り当て.
Fig. 3　Decomposition of domain connectivity graph and assignment of each subgraph to computer node.

てる（図 **3**）．グラフ分割ソフトとして，本研究では Scotch[2] を用いる．Scotch ではグラフ分割の他にマッピング機能も提供しており，次のステップのコアへの割り付け決定で利用するためである．本来ならば，グループ化した計算領域の各ノードへの割り付けもノード間の通信性能に応じてノード内と同様に割り付けするべきであるが，本研究ではノード間の通信性能はすべて均一と仮定しているため，ここではノードへの割り付けについては考慮しない．

グループ化された計算領域の各プロセッサへの割り付けは Mercier ら[6] の手法をもとに実装した．まず，対象となる計算領域のみの領域接続グラフを 2.3 節の領域グラフから抽出する．このグラフを Scotch を用いて 2.2 節で作成したプロセッサ接続グラフにマッピングすることで，各計算領域を割り付けるべきプロセッサが決定される．

3. 実行時の通信速度測定

本章ではマルチコア大規模計算機環境における通信性能の測定方法について述べる．その目的は，集合運用下における通信性能の測定値と理論値の違いを確認すること，また，本研究の結果（通信時間の最小化）を評価するためである．また，本研究では行われていないグループ化されたサブドメインの各計算ノードへの割り当てを行う際には，ここで測定された通信性能がノード接続グラフにおける各エッジへの重みになる．

3.1 バンド幅とレイテンシの測定方法

1 組の MPI プロセス間の通信速度の測定は，1 対 1 通信（point-to-point）で双方向にデータを送受信することで行う．いくつかの異なるサイズのデータでそれぞれ複数回通信時間を測定する．データサイズと通信時間の関係を最小二乗法により直線にフィッティングし，直線の傾きの逆数をバンド幅，データサイズ 0 に対応する通信時間をレイテンシとして算出する．

表 1　各ステップで各ノードが通信するノードの決め方の例（ノード数が 8 の場合）.
Table 1　Example of order of nodes that each node communicates with at each step (the number of nodes is eight).

Node ID	Step 1	2	3	4	5	6	7
0	1	2	3	4	5	6	7
1	0	3	2	5	4	7	6
2	3	0	1	6	7	4	5
3	2	1	0	7	6	5	4
4	5	6	7	0	1	2	3
5	4	7	6	1	0	3	2
6	7	4	5	2	3	0	1
7	6	5	4	3	2	1	0

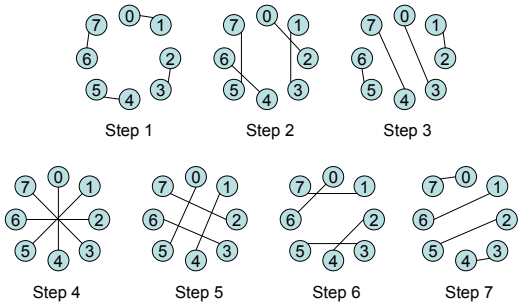

図 4　ノード数が 8 の場合（表 1）の各ステップでの通信パターン.
Fig. 4　Communication pattern at each step when the number of nodes is eight (表 1).

3.2 並列測定アルゴリズム

(1) ノード間計測の並列化

ノード間通信速度の測定にあたっては，すべてのノードの組について通信速度を測定する必要がある．ノード数を n とすると，組み合わせ数は ${}_nC_2 = n(n-1)/2$ となるため，大規模な並列計算では，ノードの組み合わせ数が膨大になり，計測の並列化が必須となる．

1 対 1 の通信は，競合しない組み合わせを複数同時に行うことが可能である．たとえば，ノード数が 8 の場合，各ステップで各ノードが通信するノードを**表 1** のように決めることができる．この場合の各ステップでのノード間の通信パターンは**図 4** のようになる．

この例における決め方を一般化すると，ノード i がステップ k で通信する相手は，$j = i \hat{} k$ で求められる（ただし，$\hat{}$ はビットごとの排他的 OR 演算子）．こ

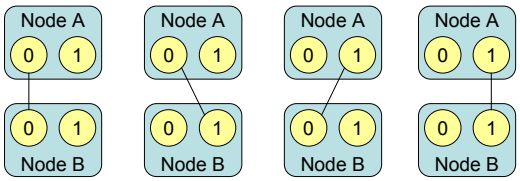

図 5 ノードあたり 2 コアの場合のノード間通信時間の計り方（全 4 通り）.
Fig. 5 Possibilities of measuring communication time between two nodes when each node has two cores.

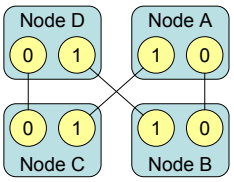

図 6 複数コア同時利用により複数ステップの計測を行う例.
Fig. 6 Example of simultanious measurement of communication time between two nodes by using multiple cores.

の方法に従うと，n が偶数の場合，1 ステップで最大 $n/2$ 組の同時計測が可能であり，また，全計測に必要なステップ数は，$2^{m-1} < n \leq 2^m$ となる m を用いて，$2^m - 1$ となる．
なお，このアルゴリズムは，ノード内のコア間通信速度の計測についても適用可能である．

(2) ノード間の通信速度の測定は各ノードの 1 つずつのコアを使用

ノードあたりのコア数を n_c とすると，ある 2 つのノード間の通信時間の計測に使うコアの組み合わせは n_c^2 通りある．たとえば，ノードあたり 2 コアの場合，通信時間の計り方は**図 5** に示す 4 通りある．計測に使うコアの違いによる通信時間の違いは，ノード内のコア同士の通信時間と同程度と考えられる．このため，ノード間の通信速度と比較して，ノード内の通信速度が十分に高速である場合，計測に使うコアの違いは無視できる．ここでは，多くの実行環境において，ノード内の通信速度は，ノード間の通信速度よりも十分高速であると想定し，n_c^2 通りのうち 1 通りのみ計測することとする．なお，実際の通信速度の関係については，実測を行い確認する．

(3) 複数コアの同時利用

ノード内の各コアは同時にそれぞれ異なるノードとの通信速度を測定することが可能である．たとえば，各ノードのコア 0 が Step 1 の通信を行い，各ノードのコア 1 が Step 2 の通信を行う，といったことが可能である（**図 6** 参照）．このため，n_c ステップ分の計測を同時に行うことが可能となる．全計測に必要なステップ数が，$2^{m-1} < n \leq 2^m$ となる m を用いて $2^m - 1$ とあらわされることから，ceil$((2^m - 1)/n_c)$ ステップで全計測が可能である（ただし，ceil は引数の値を下回らない最小の整数を求める関数）．
なお，同時に n_c 個のノードとの通信速度を測定した場合，バンド幅の理論的な最大値は，ノード間通信のバンド幅の $1/n_c$ となる．

4. 計 測 結 果

4.1 最適自動マッピング

ベンチマークプログラムに最適自動マッピングアルゴリズムを組込み，通信時間を測定した．本ベンチマークプログラムは，ポアソン方程式で記述される 3 次元熱伝導問題を差分法によって離散化し，得られた連立一次方程式を Jacobi 法で求解するものであり，並列化には，V-Sphere フレームワーク[7] を用いている．

今回のアルゴリズムでは，ノードへ割り当てるサブドメインのグループ化（ノード間最適割り当て）と，ノード内での割り当て最適化を行っている．両者の効果を分けて観測するため，ノード間の最適割り当てのみを行い，ノード内の最適割り当ておよびバインドを行わないケースについても計算を行うこととした．

本ベンチマークプログラムは直交格子を用いているため，グラフ分割ではなく各軸方向の領域数を指定することで計算領域の分割が行われる．そのため，物理的に隣接する計算領域が同じノードに割り当てられるようになっており，ノードあたりのコア数が少ない場合や，領域数が少ない場合には，既に最適に近い割り当てとなっている．領域数が増加すると，既存の割り当てでは 1 ノードに割り当てられる計算領域が x 軸方向に 1 列に細長く並ぶため，最適な割り当てと比較するとノードを越えた通信量はやや大きくなるが，同じノードに割り当てられた計算領域が隣接している状況は変わらない．しかし，非構造格子を用いる場合（有限要素法など）や，粗密のある格子を用いる場合には，よりランダムな割り当てが行われる場合が多いと考えられる．このため，ランダムな割り当てでの計算も行い，結果を比較した．

したがって今回は，(1) 最適割り当てを行わない場合（Original），(2) 最適割り当てを行った場合（AFC），(3) ノード間の最適割り当てのみを行い，ノード内でのプロセッサへの割り当ておよびバインドは行わなかった場合（AFC-NB），(4) ランダムな割り当てを行った場合（Random），の 4 ケースにつ

表 2　RICC の仕様詳細
Table 2　Specification of RICC.

CPU	Intel Xeon 5570 (2.93GHz, Quadcore) x 2/node
メモリ	12GB/node
総ノード数	1024 nodes (8192 cores)
実行環境	ジョブスケジューラで管理され，プロセスは予めコアにバインドされる．
コンパイラ	Intel Compiler 11.1
MPI	OpenMPI 1.3.3
ネットワーク構成	InfiniBand X4 DDR トポロジー : Fat-tree bisection bandwidth: 240GB/s
ノード間通信性能	2.0GB/s
CPU 間通信性能	25.6GB/s
メモリバンド幅	32GB/s (プロセッサあたり)

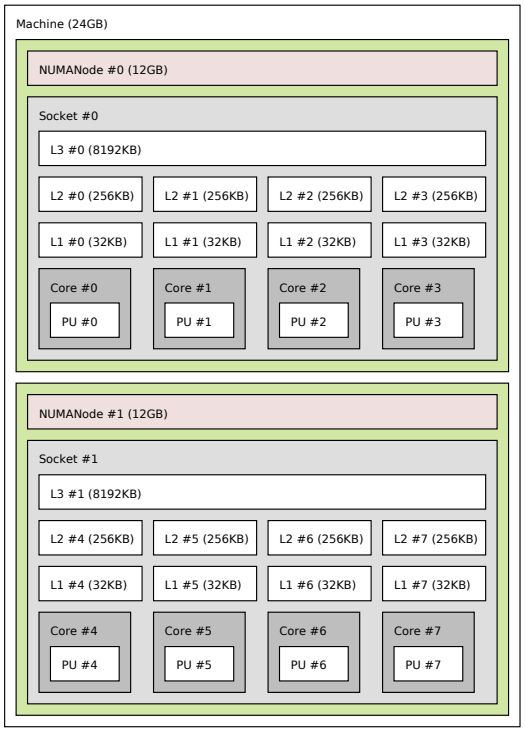

図 7　RICC の各ノードのハードウェア構成.
Fig. 7　Hardware configuration of each node of RICC.

いて計算時間を比較する．

使用した計算機環境は，理化学研究所のスーパーコンピュータ RICC の超並列 PC クラスタである．RICC の仕様詳細を**表 2** に示す． また，hwloc により生成される RICC の各ノードのハードウェア構成図を**図 7** に示す．

以下に示す 2 種類のウィークスケーリングを行い，性能を測定した．

Case 1　解析領域全体のアスペクト比を一定として，ウィークスケーリングした場合．このケースは，同一の物理現象の解析を，解析規模を変えて解析する場合に対応する．各サブドメインのアスペクト比は分割数に応じて変化する．解析領域全体，および，サブドメインの格子点数は**表 3** に示す通りである．

表 3　Case 1 のウィークスケーリングにおける解析領域全体，および，サブドメインの格子点数.
Table 3　Number of grids of whole domain and each subdomain at Case 1.

#Cores	Grids (whole)	Grids (subdomain)
32	3251×813×813	813x407x203
64	4096×1024×1024	1024x256x256
128	5161×1290×1290	645x323x323
256	6502×1625×1625	813x406x203
512	8192×2048×2048	1024x256x256

表 4　Case 2 のウィークスケーリングにおける解析領域全体，および，サブドメインの格子点数.
Table 4　Number of grids of whole domain and each subdomain at Case 1.

#Cores	Grids (whole)	Grids (subdomain)
32	4096×512×1024	1024x256x256
64	4096×1024×1024	1024x256x256
128	8192×1024×1024	1024x256x256
256	8192×1024×2048	1024x256x256
512	8192×2048×2048	1024x256x256

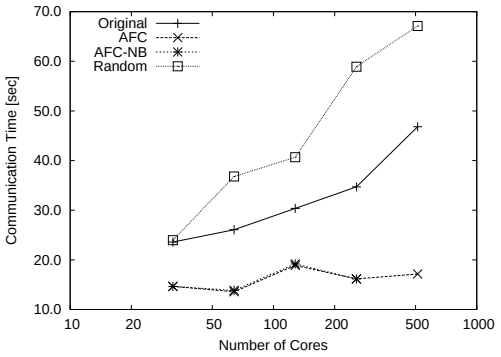

図 8　Case 1 のノード間通信時間.
Fig. 8　Total communication time in Case 1.

Case 2　領域分割後の各サブドメインのアスペクト比を一定として，ウィークスケーリングした場合．この場合，解析領域全体の形が変化するが，各サブドメインにおける節点数と通信量の比が一定となる．解析領域全体，および，サブドメインの格子点数は**表 4** に示す通りである．

RICC における結果を**図 8**，**図 9** に示す．Case 1, Case 2 ともに，Original および Random では，領域数の増加とともに通信時間が増大していくが，AFC および AFC-NB では通信時間の増大はごく僅かである．AFC は，Original に対して最大 2.7 倍，Random

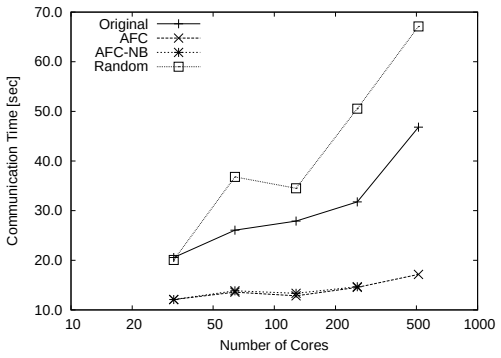

図 **9** Case 2 のノード間通信時間.
Fig. 9 Total communication time in Case 2.

に対して最大 3.9 倍の高速化が観測され，最適割り当てによる通信時間短縮の効果が大きいことが確認された．一方，AFC と AFC-NB の間に優位な差は観測されなかった．したがって，今回数値実験に用いた環境では，高速化の主な要因はノード間の最適割り当てによるものであると言える．

4.2 通信速度測定

RICC において通信時間の実測を行った．通信データのサイズは，4.1 節のベンチマーク計算における通信量の平均である約 1.5MB を中心として，前後 5 点（各 0.667, 0.8, 1.0, 1.25, 1.5 倍）を用い，各点の計測を 7 回行った．計算されたバンド幅とレイテンシを，同一 CPU 内のコア間，ノード内の CPU 間，ノード間に分けて集計し，それぞれ平均と標準偏差を計算した．

測定の結果を**表 5** に示す．ノード間（inter-node）の通信性能については，ノード数が 2 の場合にノード間通信のバンド幅が最大となっており，約 2.2GB/s と，ほぼ理論性能通りの実効性能が観測されている．また，ノード数の増加にともない，平均速度が遅くなっている．これは，同時に通信速度を測定するノード数が増加するためである．ノード数が大きい場合，ノード間通信バンド幅の理論性能の 1/8 である 256MB/s がコアあたりの理論性能となるが，実際にはそれよりもやや遅くなっており，速度のばらつきも大きくなることがわかった．RICC のノード間通信性能は理論上均一とされるが，この結果はノード間の通信距離が必ずしも均一ではないことを示唆しており，最適割り当てにおいて，ノード間の通信距離を考慮することにより，さらに計算機リソースを有効利用できる可能性があることを示している．

今回の最適自動マッピングでは，ノード間，同一ノード内の CPU 間，同一 CPU 内のコア間の通信速度の大小関係について，ノード内同一 CPU 内のコア間が最も速く，ノード間が最も遅いと仮定した．これについて，実測結果では，2 ノード（16 コア）の時にノード間の通信速度がノード内の CPU 間の通信よりも速くなっているが，4.1 節のベンチマーク計算を行った 4 ノード（32 コア）以上では，今回仮定した通りの大小関係が認められた．これは，RICC において最適自動マッピングの効果が大きく表われたことを裏付けている．また，ノード間の通信速度の測定において，ノード内の通信速度は十分に高速であると考え，測定に用いるコアの違いを無視したが，RICC の実行環境では，8 ノード以上使用する場合には，この方法で妥当な結果が得られていると考えられる．

5. 結　　論

本研究では，マルチコア大規模並列計算機環境における領域分割型並列計算法の，サブドメインのコアへの最適マッピング法について提案した．ハードウェア情報に基づいた計算機の階層構造グラフ，およびサブドメイン間の通信量を考慮した重み付きグラフの二つのグラフを利用し，コアへの最適なサブドメインマッピングを実現した．Intel Xeon 大規模 PC クラスタ（RICC）におけるテスト結果から，本手法により並列計算時の通信時間を大幅に短縮可能であり，大規模計算時の並列性能向上を見込めることが分かった．

大規模並列環境における実行時の通信速度を並列処理により高速に実測するアルゴリズムを開発し，RICC で測定を実施し，通信性能を調査した．その結果，RICC においては，通信経路の違いによる通信性能の違いが明確に存在していることが分かった．このことが，最適マッピングによる通信時間の短縮が可能となった主な要因であると考えられる．また，RICC のノード間通信は，理論上均一とされているが，ノード数が大きい場合には必ずしも均一ではないことが分かった．

1 章で述べたように，本研究ではノード間の通信性能は均一という仮定での最適マッピング法を提案している．今後主流となる非均一通信性能環境に対しては，グループ化したサブドメインの接続グラフを作成し，そのグラフを計算機ノードグラフへマッピングすることで非均一性を考慮した最適割り当てを行い，通信時間を短縮することができると考えられる．均一通信性能環境においても，ジョブスケジューラで管理された集合運用の場合には，今回観測されたように，実質的に非均一となっていることがあるため，非均一性を考慮した最適割り当てが有効になると考えられる。

謝辞　本研究は，文部科学省　最先端・高性能汎用

表 5 通信速度の実測結果.
Table 5 Measured communication bandwidth and latency.

#Nodes	#Cores	Intra-Node								Inter-Node			
		Intra-CPU				Inter-CPU							
		Bandwidth [MB/sec]		Latency [μsec]		Bandwidth [MB/sec]		Latency [μsec]		Bandwidth [MB/sec]		Latency [μsec]	
		Avg	Stdev	Avg	Stdev	Avg	Stdev	Avg	Stdev	Avg	Stdev	Avg	Stdev
1	8	2752	9	18	2	1270	15	24	4	-	-	-	-
2	16	2724	9	17	1	1247	8	17	5	2227	-	0	-
4	32	2740	12	19	2	1420	9	33	3	655	14	77	22
8	64	2781	13	24	2	1314	13	24	5	186	19	150	149
16	128	2767	15	24	2	1449	4	25	2	159	16	152	176
32	256	2773	11	25	2	1361	16	32	4	130	44	65	118

スーパーコンピュータの開発利用「次世代生命体統合シミュレーションソフトウェアの研究開発」の支援を受け行われたものである．数値計算は，理化学研究所情報基盤センターの RIKEN Integrated Cluster of Clusters （RICC）を用いて行った.

参 考 文 献

1) Karypis G. and Kumar V.: Parallel Multilevel k-way Partitioning Scheme for Irregular Graphs, *Supercomputing* (1996).
2) *Scotch 5.1.8*, http://www.labri.fr/perso/pelegrin/scotch/
3) http://www.nccs.gov/computing-resources/jaguar/#XT5-6-Core-Upgrade
4) http://www.nsc.riken.jp/index_j.html
5) *Hardware Locality (hwloc) 1.0.2*, http://www.open-mpi.org/projects/hwloc/
6) Mercier G. and Clet-Ortega J.: Towards an Efficient Process Placement Policy for MPI Applications in Multicore Environments, *Europvm/mpi 2009*, Espoo, Finland (2009).
7) 小野謙二, 玉木剛, 野田茂穂, 岩田正子, 重谷隆之: オブジェクト指向並列化クラスライブラリの開発と性能評価, 情報処理学会論文誌：コンピューティングシステム, Vol.48 No.SIG 8 (ACS 18) (2007).