社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－HPC－107 (4)
2006／7／31

グリッド環境において計算結果を効率よく再利用するための通信量削減

松尾勝則[†] 田中裕也[†,☆] 川崎康博[†]
水谷泰治[††] 伊野文彦[†] 萩原兼一[†]

本稿では，プログラム内部で計算結果を再利用することを目的として，再利用のオーバヘッドを低減する通信量削減手法を提案する．ここでの再利用とは，同一計算の実行を回避することにより，グリッド資源を効率よく使用することを指す．提案手法は，計算結果を格納するデータベース（DB）内の属性値を最小化することにより，DB サーバおよび計算ノード間の通信量を削減する．最小化のために，ユーザが指定する初期の属性と等価なものをプログラムに対するデータ依存解析により列挙し，それらのうち最小のものを DB を操作するためのクエリに使用する．提案手法を医用画像処理に適用した結果，データサイズを 683MB から 52B まで削減できた．また，計算結果の再利用により，その実行時間を 9 分から最短で 27 秒に短縮できた．

Reducing Communication for Efficient Data Reuse in the Grid

KATSUNORI MATSUO,[†] YUYA TANAKA,[†,☆] YASUHIRO KAWASAKI,[†]
YASUHARU MIZUTANI,[††] FUMIHIKO INO[†] and KENICHI HAGIHARA[†]

This paper presents a communication reduction method that aims at minimizing overhead for data reuse during program execution. The data reuse here is designed to promote efficient use of grid resources by avoiding multiple execution of the same computation in a virtual community. Our method minimizes the data size of attributes, which are associated with computation products in a database (DB). To minimize this, the method performs data dependence analysis against the target program, and then lists attributes equivalent to initial attributes given by users. Among them, we select the minimum attributes to construct queries to operate the DB, reducing communication between the DB server and computing nodes. We also show some experimental results obtained using a medical application. The experimental results show that the proposed method successfully reduces the data size of a query from 680 MB to 124 B. Our reuse framework also accelerates the application from approximately 9 minutes to 27 seconds, achieving shorter responses with higher efficiency.

1. はじめに

近年，複数の組織間で計算資源を統合し，1 つの仮想的な高性能計算機環境として動作させるグリッド技術に関する研究が盛んである．多くの研究は，計算資源やデータを仮想組織内で共有することに主眼がある．

一方，計算資源の効率のよい利用を目指し，計算資源のみならず計算そのものを共有する試み[1)~5)]がある．具体的には，仮想組織内で計算結果を蓄積し，それらを計算の替わりに読みだすことで再利用する．

例えば，西川ら[1)]は特定のアプリケーションを対象とした再利用を実現している．具体的には，入力値が同一のジョブに対し，過去の計算結果を再利用できるものと判断し，仮想組織内における同一ジョブの再実行を回避する．そのために，計算結果および入力を属性として持つデータベース（DB）サーバを構築し，属性およびその値（属性値）を指定してクエリを作成することで，DB への登録や検索を実現する．

また，Deelman ら[2),5)]は再利用機構をワークフローシステムの一部として実現している．このシステムでは，プログラムを頂点，プログラム間のデータの流れを辺とする有向グラフでワークフローを表現する．

† 大阪大学大学院情報科学研究科コンピュータサイエンス専攻
Department of Computer Science, Graduate School of Information Science and Technology, Osaka University
†† 大阪工業大学情報科学部情報システム学科
Department of Information Systems, Faculty of Information Science and Technology, Osaka Instituted of Technology
☆ 現在，株式会社日立製作所
Presently with Hitachi, Ltd.

ユーザから与えられる有向グラフに対し，再利用により置き換えることのできる頂点および辺を除去し，有向グラフを簡略化したうえで計算資源に割り当てる．

このように，計算結果の再利用は，同一計算による計算資源の浪費を回避できる．したがって，仮想組織内で未実行の新しい計算に対し，計算資源を割り当てられる．しかし，既存の再利用機構はプログラム単位の計算結果を蓄積するため，プログラム内に同一計算が存在する可能性がある．その場合，プログラム内部の再利用によりさらなる効率化を図れる．

そこで本研究では，プログラム内の再利用を低オーバヘッドで実現することを目指し，DB サーバおよび計算ノード間の通信量を削減する手法を提案する．提案手法は，DB を操作するためのクエリが属性値を含むことに着目し，そのデータサイズを最小化することにより通信量の削減を試みる．その最小化のために，ユーザが指定する初期の属性と等価なものを有向グラフに対するデータ依存解析により列挙し，それらのうち最小のものをクエリに使用する．

以降では，まず計算結果再利用の原理を述べる（2 章）．次に，属性値のデータサイズ削減手法および属性の抽出手法を説明する（3 章）．その後，非剛体位置合わせ[6] を具体例として提案手法を評価する（4 章）．最後に本稿をまとめる（5 章）．

2. 計算結果再利用の原理

本章では，提案手法を説明するための前提知識についてまとめ，プログラム内部で再利用を実現するために解決すべき課題を挙げる．

2.1 再利用の前提

プログラムのフロー依存を有向グラフ DAG：$G = (V, E)$ で表す（**図 1**）．ここで，V は頂点の集合，E は辺の集合を表す．頂点を 1 つの代入文，辺を代入文間のフロー依存関係とする．以降，頂点 u を始点，v を終点とする辺を (u,v) と表記する．さらに，図 1 における頂点 4 への入力辺 R_3 および b のように，計算に直接的に入力される変数を**直接依存の変数**，a のようにフロー依存を介して間接的に入力される変数を**間接依存の変数**と呼ぶ．

次に，計算結果は DB で管理する（**図 2**）．DB に登録する属性は以下の 3 つである．

- 計算名
- 計算への入力
- 計算結果

ここで，計算名は仮想組織内で一意とする．このような DB が与えられたとき，入力と計算名の属性値をクエリで指定すれば計算結果を再利用のために取得できる．

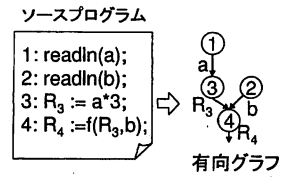


図 1 ソースプログラムを DAG で表した図

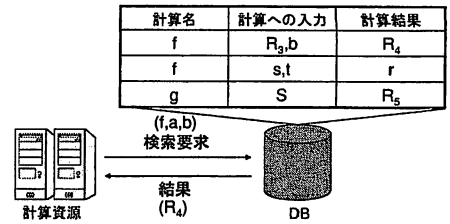


図 2 計算結果再利用のイメージ

最後に，本研究ではユーザがプログラムに関数呼出しを挿入することにより，再利用の対象範囲を指定することを想定している．

2.2 再利用の条件

ある計算の結果を再利用するとき，その計算は置き換え対象の計算と一致する必要がある．本節では，その判定条件を図 1 を用いて説明する．

再利用対象の計算を図 1 の頂点 4 とする．頂点 4 に対する入力は R_3 および b であり，再利用したい計算結果は R_4 となる．このとき，DB（図 2）に計算名 f，入力 R_3，b の属性値と計算結果 R_4 の属性値が登録されていれば，R_4 を再利用できる．ゆえに，R_4 を再利用するためには，計算名 f，入力 R_3，b の属性値が DB に登録されているデータと一致しなければならない．したがって，計算結果を再利用するための判定条件は下記となる．

- 計算名，入力の属性値が DB に登録されているデータと一致すること

2.3 再利用の手順

計算結果を再利用するためのプログラム実行前の手順および実行時の手順を示す．

- プログラム実行前
  - (1) ユーザがプログラム内の再利用対象の計算を開始行と終了行で指定する．
  - (2) プログラムを解析し，DAG：G および再利用対象の計算を表す頂点集合を求める．
  - (3) DB 上の計算結果を検索するために，属性

抽出アルゴリズム（3.2 節）で属性の候補集合を抽出する．

（4） 属性の候補集合の中でデータサイズ最小の属性と計算結果を保存する変数を引数として，再利用のための関数呼出しを挿入する．

（5） プログラムをコンパイルする．

- プログラム実行時

（1） 再利用対象の計算の実行前に，データサイズ最小の属性値を DB に転送する．

（2） DB 上の計算結果を検索する．計算結果がある場合，DB からそれを取得し，計算を省く．ない場合，実際に計算し，その計算結果を DB に登録する．

2.4 計算結果再利用の課題

プログラム内部における計算結果再利用に起因する課題は下記のとおりである．

- プログラム実行中に DB へ属性値を転送する．
- プログラム実行中に DB 上の計算結果を検索する．

属性値のデータサイズを削減すると，その転送時間が短くなる．また，検索対象のデータサイズも小さくなるので，計算結果の検索時間も削減できる．さらに，プログラムの実行時間を短くできる．したがって，オーバヘッドを削減するために，属性値のデータサイズ削減が必要である．

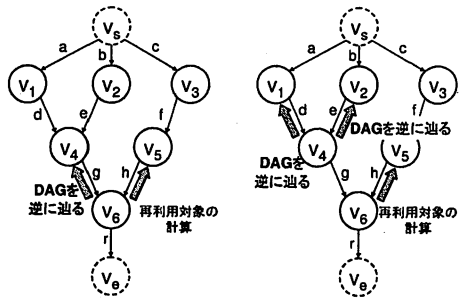
3. 属性値のデータサイズ削減と属性の抽出

本章では，提案手法について述べる．具体的には，属性値のデータサイズを削減する手法および，その手法を用いてデータサイズ最小の属性をプログラム内から抽出するアルゴリズムを説明する．

3.1 属性値のデータサイズ削減

データサイズの小さい属性を見つけるために，提案手法は**図 3** のように DAG を逆に辿り，間接依存する変数を求める．そして，その変数を用いて，再利用対象の計算に直接依存する変数を置換する．例えば，図 3 の変数 g は，**図 3(a)** に示すとおりに DAG を逆に辿ると g と間接依存の関係にある変数に d と e があることがわかる．よって，g はこの d，e を用いて v_4，d，e で置換できる．提案手法では，直接依存する変数だけではなく，間接依存する変数を検索の属性とする．また，さらに深く DAG を辿ることで他にも間接依存する変数を発見できるので，属性の候補を複数個生成できる．その中でデータサイズ最小のものを選択することで，属性値のデータサイズを削減する．

次に，提案手法の適用条件を説明する．提案手法を適用条件は下記のとおりである．



(a) 直接依存の変数抽出　　(b) 間接依存の変数抽出

図 3 DAG を逆に辿るイメージ

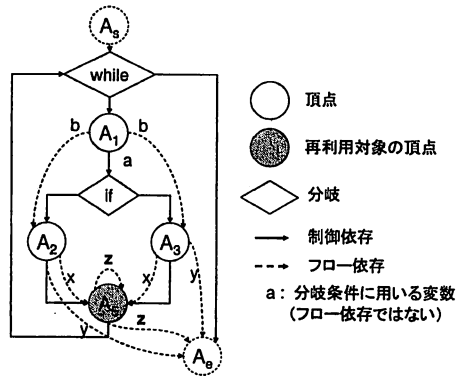


図 4 条件分岐および繰り返しを含むプログラム構造

- DB 上にある計算結果を出力するまでの処理内容と区間内の処理内容が同一であること

区間内に分岐があり，再利用対象の計算以外にフロー依存があれば，分岐条件の変数を属性に加える．また，ループ運搬依存の変数は間接依存の変数値の変遷を属性にする．

ここで**区間内**とは間接依存の変数を出力する計算と再利用対象の計算までの間を指す．処理内容が同一でなければならない理由は，区間内の処理内容が異なれば，後続の処理への入力が異なることもあるからである．特に，区間内に分岐および繰り返しが存在する場合，処理内容の表現を工夫する必要がある．

- 分岐が存在する場合

分岐条件で用いる変数を検索の属性に含めなければならないことがある．例えば，**図 4** において，変数 x はフロー依存の関係から間接依存の変数 b に置換できる．ここで，A_2 の処理実行前に，間接依存の変数 b で DB を検索すると A_3 の結果 x を

用いて計算した結果 z を取得できることがある．この場合は再利用できない．なぜならば，x は一致しても，A_2，A_3 の出力 y が一致するとは限らないからである．ゆえに，再利用対象の計算以外にフロー依存がある場合は処理内容を判別する必要がある．その判別には分岐条件の変数（図 4 の a）を属性に加えることで実現できる．

- 繰り返しが存在する場合

  ループ運搬依存の変数は処理内容を表現する間接依存の変数に置換する必要がある．例えば，図 4 では z がループ運搬依存の変数である．z を再利用する時，繰り返しで実行された処理内容を表さなければ後続の繰り返しで正しく計算 A_5 を実行できない．よって，z を間接依存の変数で表現するためには，その z を出力するまでの処理内容を表さなければならない．これは間接依存の変数値（図 4 の b）の変遷を保存することで実現できる．

3.2 属性抽出アルゴリズム

再利用対象の計算に直接依存する変数および間接依存する変数を用いて，検索の属性の候補を自動抽出するアルゴリズムを示す．属性の候補は DAG の辺を逆に辿ることで蓄積する．このアルゴリズムでは DAG の辺を根に到達するまで深さ優先で辿る．なぜなら，最適な属性候補を確実に得るためには根まで辿る必要があるからである．根まで辿った時，属性候補の抽出を終える．そして，抽出した属性の候補をプログラム開発者に提示する．

このアルゴリズムでは直接依存の属性と間接依存の属性を抽出する．以下，アルゴリズムの概略を説明する．このアルゴリズムの詳細は**図 5** である．

- 直接依存の属性の抽出（既存手法）

  直接依存の変数は DAG 中で再利用対象の領域外から領域内に向かう辺を列挙することで抽出する．つまり，DAG で再利用対象の領域内の全ての頂点について，領域外からの辺を列挙すれば直接依存の属性を抽出できる．よって，直接依存の属性はおおまかに下の 3 つの手順で抽出できる．

  （1） 再利用対象の領域外から領域内に向かう辺を列挙

  （2） 依存関係の印をマーク

  （3） 列挙した辺を属性候補に追加

  図 3(a) を直接依存の属性を抽出する例として説明する．まず，再利用対象の領域内の頂点 v_6 を終点とし領域外の頂点を始点とする辺を列挙する．その辺には (v_4, v_6) と (v_5, v_6) があり，これらが v_6 に直接依存する属性である．

```
入力： (1) DAG : G = (V, E)，制御依存グラフ
       (2) R (⊆ V)，再利用対象の領域内の頂点の集合
出力： (1) K，属性の候補集合
1: K = φ ; //属性候補を蓄積
2: P = φ ; //辺集合を蓄積
3: foreach ( 頂点 v ∈ R ) {
4:  v に訪問済みの印をつける ;
5:  foreach ( 辺 (z, v) ∈ E ) {
6:   if(z ∉ R){
7:    制御依存グラフをチェックし，依存の印をつける;
8:    if(辺 (z, v) != ループ運搬依存){
9:     スタック S に辺 (z, v) を push;
10:   }
11:   if((辺 (z, v)=分岐依存)∧
      (∃u(u! = v∧ 辺 (z, u) ∈ E)){
12:    分岐条件の変数を P に追加;
13:   }
14:   P に辺 (z, v) を追加
15:  }
16: }
17:}
18:K に P を加える ; //直接依存の属性候補を蓄積
19:while ( S! = φ ) {
20: S から辺 (x, v) を pop ; //再利用対象の領域拡大
21: P = φ ;
22: foreach ( 辺 (z, x) ∈ E ) {
23:  if(z ∉ R){
24:   制御依存グラフをチェックし，依存の印をつける;
25:   if(辺 (z, x) != ループ運搬依存){
26:    スタック S に辺 (z, x) を push;
27:   }
28:   if((辺 (z, x)=分岐依存)∧
      (∃u(u! = x∧ 辺 (z, u) ∈ E)){
29:    分岐条件の変数を P に追加;
30:   }
31:   P に辺 (z, x) を追加
32:  }
33: }
34: if ( P != φ ) {
35:  foreach ( クエリの候補 C ∈ K ) {
36:   C に含まれる辺 (x, v) を P に置換した
      属性を K に追加 ;//間接依存の属性候補を蓄積
37:  }
38: }
39:}
```

図 5 クエリの候補抽出アルゴリズム

- 間接依存の属性の抽出（提案手法）

  間接依存の属性はスタックに蓄積された辺を用いて，下記の手順で抽出する．

  （1） 再利用対象の領域を拡大

  （2） 新たに追加された領域内の頂点に直接依存する辺を列挙

  （3） 属性候補中の直接依存の辺を間接依存の辺に置換した候補を属性候補に追加

  これを繰り返せば，DAG の根まで辿ることになり，全ての間接依存の属性を抽出できる．

例えば図 3 では，まず，再利用対象の領域を v_4 まで拡大する．そして，v_4 に直接依存する属性 (v_1, v_4) と (v_2, v_4) を求める．最後に属性候補中の $\{(v_4, v_6), (v_5, v_6)\}$ の (v_4, v_6) を辺 (v_1, v_4), (v_2, v_4) と頂点 v_4 に置換し，属性候補に追加する．この一連の操作を繰り返すと根の v_s まで辿ることになり，全ての間接依存の属性を求められる．

4. 評価実験

医用画像処理の 1 つである非剛体位置合わせ[6] に対して提案手法を適用し，データサイズの削減効果および再利用による実行時間の短縮効果を調べた．

実験に用いた計算機環境は，16 台の PC をミリネット (2Gb/s) で接続した PC クラスタである．各 PC は，2.8GHz 駆動の Xeon プロセッサを 2 つ持ち，2GB の主記憶を装備する．ミリネット上の通信には MPICH-SCore[7] を用いる．これらに加え，DB サーバが 1 台あり，各 PC はイーサネット (1Gb/s) を介してサーバ上の DB を操作する．

非剛体位置合わせは，2 組の 3 次元画像を入力として，画像内の物理的な位置の対応関係を定める．この際，画像を序々に精細化し，階層的に位置を合わせる．各階層は，位置合わせを制御するためのパラメータを入力として必要とする．位置合わせ処理は，下記の理由により再利用が効果的である．

- 上記の PC クラスタにおいて 20 分程度を要し実行時間が長い．
- 階層ごとのパラメータの組合せに関してよいものが確立していないため，パラメータスイープに基づく位置合わせ結果の検証が望まれている．
- 粗い階層での計算結果を再利用すれば，上記の組合せを効率よく走査できる．

なお，画像サイズは 512 × 512 × 295 ボクセルであり，ファイルサイズは 1 画像当たり 148MB である．

4.1 データサイズ削減効果

再利用による実行時間の短縮を得るために，位置合わせにおいて実行時間の 98%を占める部分（**図 6** の 16 および 17 行目）を初期の再利用対象として指定した．**図 7** に，アルゴリズムに対応する DAG を示す．

まず，直接依存の属性を抽出する．16 および 17 行目に直接依存する変数は，パラメータ a，画像情報 b.img および位置合わせ結果 b.vec である．これら初期属性のデータサイズを**表 1** に示す．画像情報 b.img が 680MB ものサイズを占めている．

次に，直接依存の属性を間接依存のものに置き換える．データサイズの大きい b.img に着目すると，7 行目でファイル名 img1 および img2 を指定して読みだしていることから，これらを間接依存の属性として b.img と置き換えればよい．同様に，位置合わせ結果 b.vec について置き換え候補を挙げると，a，b.vec および b.img である．うち，b.img は記述の通りであり，b.vec はループ運搬依存の変数である．したがって，b.vec を置換するためには，ループ内部の制御フローの変遷を記憶するために a の変遷を属性とする．まとめると，初期属性 {b.img, b.vec, a} に対し，{img1, img2, a の変遷 } に置き換えることができる．

```
 1: readln(img1); // File name 1
 2: readln(img2); // File name 2
 3: readln(g1); // Parameter 1
 4: readln(g2); // Parameter 2
 5: readln(g3); // Parameter 3
 6: h = 0; // 階層
 7: b.img = load(img1, img2);
 8: while h<5 {
 9:     if h==0 {
10:         a = g1;
11:     } else if h==1 {
12:         a = g1 * g2;
13:     } else {
14:         a = g1 * g2 * g3;
15:     }
16:     b.vec = registration(a, b.img, b.vec);
17:     b.vec = resample(b.img, b.vec);// 階層上げ
18:     h += 1;
19: }
```

図 6 非剛体位置合わせアルゴリズム[6] の概要

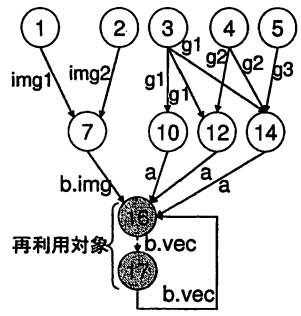


図 7 非剛体位置合わせの DAG

表 1 より，この際のデータサイズは 683MB から最小で 52B に削減できる．具体的には，680MB の b.img は 12B の img1 および img2 の 12B になり，階層ごとに 58KB から 3MB までデータサイズが増加する b.vec は img1，img2 および a の変遷，すなわち 20B から 52B に置き換わる．

4.2 実行時間の短縮効果

表 2 に，計算結果の再利用による実行時間の短縮効果を示す．この結果は，5 つの階層 L1 から L5 のう

表 1　階層 L1〜L5 の各々における属性値のデータサイズ

削減前						削減後					
属性	L1	L2	L3	L4	L5	属性	L1	L2	L3	L4	L5
b.img	680M					img1	6				
b.vec	58K	408K	3M			img2	6				
a	8					a の変遷	8	16	24	32	40
Total	680M	680M	683M			Total	20	28	36	44	52

表 2　非剛体位置合わせにおける再利用による計算時間の短縮効果

階層	階層ごとの再利用した時の実行時間（秒）					
	なし	L1	L1〜L2	L1〜L3	L1〜L4	L1〜L5
L1	2.4	0.1	0.1	0.1	0.1	0.1
L2	14.6	15.1	0.7	0.4	0.1	0.4
L3	85.1	94.8	85.8	0.7	0.3	0.6
L4	326.8	327.6	326.1	333.8	0.3	0.7
L5	91.4	92.5	90.7	93.3	95.6	0.7
その他	24.8	14.4	24.2	24.1	21.6	24.3
Total	545.1	544.5	527.6	452.4	118.0	26.8

ち，再利用対象を L1 のみからすべての階層 L1〜L5 まで徐々に広げたものである．表 2 では，実行時間の内訳を階層ごとに示しており，計算結果の読み書きに要する時間を含む．

本来は 545.1 秒を要する計算を，再利用範囲を広げていくことにより最短で 1/20 の 26.8 秒まで短縮できる．ただし，L3 までの再利用は時間短縮にほとんど貢献していない．したがって，時間短縮の効果を得るためには，L4 あるいは L5 までの再利用が必要である．つまり，L1〜L4 までのパラメータの組合せが同一であれば，それ以降の L5 におけるパラメータスイープを 1 回当たり 1/20 の時間で完了できる．

5. まとめと今後の課題

プログラム内部の計算に対し，低オーバヘッドな再利用の実現を目的として，DB サーバおよび計算ノード間の通信量を削減する手法を提案した．提案手法は，DB を操作するためのクエリが属性値を含むことに着目し，そのデータサイズを最小化することにより通信量の削減を試みる．その最小化のために，ユーザが指定する初期の属性と等価なものを有向グラフに対するデータ依存解析により列挙し，それらのうち最小のものをクエリに使用する．

提案手法を実アプリケーションに適用した結果，680MB 以上のオブジェクト変数を画像名および階層パラメータの値の変遷に置換することにより 52B に削減できた．また，約 9 分をかかる計算をイーサネット経由で計算結果を再利用することで最大で約 27 秒まで短縮できた．

今後の課題としては，一連の処理を自動化するツールの開発や実際の仮想組織における評価が挙げられる．

謝辞　本研究の一部は，科学研究費補助金基盤研究 (B) (2) (18300009)，特定領域研究 (17032007) および若手研究 (B) (17700060) の補助による．

参 考 文 献

1) Nishikawa, T., Nagashima, U. and Sekiguchi, S.: Design and Implementation of Intelligent Scheduler for Gaussian Portal on Quantum Chemistry Grid, *Proc. 3rd Int'l Conf. Computational Science (ICCS'03), Part III*, pp.244–253 (2003).

2) Deelman, E., Singh, G., Su, M.-H., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman, G. B., Good, J., Laity, A., Jacob, J.C. and Katz, D.S.: Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems, *Scientific Programming*, Vol.13, No.3, pp.219–237 (2005).

3) 増田慎吾，蟻川　浩，市川本浩，藤川和利，砂原秀樹：Grid Computing による科学技術計算のためのワークフローシステム，情報処理学会研究報告，2004-HPC-100, pp.31–36 (2004).

4) 廣安知之，三木光範，片浦哲平，谷村勇輔：Grid 環境における評価部に個体データベースを用いた遺伝的アルゴリズムの提案，情報処理学会研究報告，2002-HPC-91, pp.79–84 (2002).

5) Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Blackburn, K., Lazzarini, A., Arbree, A., Cavanaugh, R. and Koranda, S.: Mapping Abstract Complex Workflows onto Grid Environments, *J. Grid Computing*, Vol.1, No.1, pp.25–39 (2003).

6) Ino, F., Ooyama, K. and Hagihara, K.: A Data Distributed Parallel Algorithm for Nonrigid Image Registration, *Parallel Computing*, Vol.31, No.1, pp.19–43 (2005).

7) O'Carroll, F., Tezuka, H., Hori, A. and Ishikawa, Y.: The Design and Implementation of Zero Copy MPI Using Commodity Hardware with a High Performance Network, *Proc. 12th ACM Int'l Conf. Supercomputing (ICS'98)*, pp. 243–250 (1998).