分散メモリ計算機用 Ninf API の実現に向けて

小川宏高[†]　松岡聡[†]　中田秀基[††]
佐藤三久[†††]　関口智嗣[††]

ネットワーク数値情報ライブラリ Ninf (**N**etwork based **Inf**ormation library for High Performance Computing) は，広域に分散した計算資源や情報資源を利用した超分散並列計算の基盤を提供するソフトウェアシステムである．本稿では，この Ninf システム上で分散メモリ型並列計算機を計算資源として利用する場合に，フロントエンドマシンや I/O プロセッサ等の単一ノードで動作する Ninf サーバに計算情報が集中して律速となる問題を指摘する．この問題を解決するため，初期データ分散を記述するための共通の API を提供すると共に，クライアントとの接続を各ノードに対して適切に順次ハンドオフしていく機構の導入を提案する．また，ハンドオフ機構のコア部分を富士通の AP1000 上に実現し予備評価を行った結果，16% の性能向上を得た．

Ninf API for Distributed Memory Multiprocessors

HIROTAKA OGAWA ,[†] SATOSHI MATSUOKA ,[†] HIDEMOTO NAKADA ,[††]
MITSUHISA SATO [†††] and SATOSHI SEKIGUCHI[††]

To establish a basis for globally-distributed parallel computing in numerial computing, we are currently working on the Ninf (**N**etwork based **Inf**ormation library for High Performance Computing) software system. Using this system on distributed memory multiprocessors, Ninf core server runs on a single node such as a frond-end machine or an I/O processor. As a result, computation data concentrates on the node and easily become a bottleneck and even might exhaust its memory resource. To prevent this problem, we propose new common API for describing initial data distributions and mechanism to hand-off connection with Ninf-client to the target node successively. We preliminarily evaluate our hand-off mechachism on Fujitsu's AP1000. Results show that it improves the total execution times.

1. はじめに

計算機ネットワーク技術の発展に伴い，広域ネットワークの利用が促進され，様々な情報サービスがインターネットを通じてアクセス可能になった．これらのサービスの重要な特長は透明性である．即ち，時間や場所を問わず，あたかも共有された情報にアクセスするが如く，それらの情報を入手できるという点である．しかし，E-mail，FTP，World Wide Web 等の現在普及している情報サービスの多くはデータの共有に留まっている．一方，ネットワーク技術の進展に伴う高速化，バンド幅拡大によるギガビット級ネットワークの実現を視野に入れれば，広域ネットワークに接続された超高速計算機やディスクストレージを含む計算資源の共有もまた可能である．このように広域に分散した計算資源や情報資源を利用した超分散並列計算技術を Global Computing ないし World Wide Computing と呼ぶ．

Ninf (**N**etwork based **Inf**ormation Library for High Performance Computing)[1)] は科学技術計算向けの Global Computing を実現する基盤ソフトウェアであり，広域ネットワーク上に分散された計算資源や情報資源を使ったアプリケーションの記述を容易にする．科学技術計算向けの計算資源としてはよく使われる数値計算ルーチン等が利用でき，情報資源としては物理定数や過去の論文中の計算結果を参照する情報ベースが利用できる．

Ninf の基本システムは Client-Server モデルに基づく計算・情報資源利用を支援する．計算資源はリモート計算ホスト上のリモートライブラリとして実現され，クライアントは計算に必要なパラメータをサーバに送り，サーバは計算後，結果を送り返す．情報資源はリモート情報ホスト上の情報ベースとして実現され，クライアントは要求情報の指定を送り，サーバは情報のコンテンツ

[†] 東京大学工学部
Faculty of Engineering, The University of Tokyo
[††] 電子技術総合研究所
Electrotechnical Laboratory
[†††] 新情報処理開発機構
Real World Computing Partnership

を送り返す．この両者はインタフェースを共通化できるので，以降は計算資源についてのみ述べる．

Ninf Remote Procedure Call (Ninf RPC)2),3) はリモートライブラリの利用手続きを既存の言語のプログラマにも分かりやすい API として提供するものである．これを用いることでプログラマはネットワークプログラミングに煩わされることなく，Global Computing 環境での科学技術計算アプリケーションが書ける．

しかし，この Ninf RPC はサーバ側で単一のメモリ空間を仮定しているため，ベクトル型計算機や共有メモリ型計算機での利用には問題がないが，分散メモリ型計算機では，フロントエンドマシンや I/O プロセッサ等の単一ノードに計算情報が集中して大きく効率を低下させるという問題が生じる．

本稿では，この問題に対し，初期・最終データ分散を記述するための API を提供し，その情報を使ってクライアントのコネクションを適切な各ノードに対して順次ハンドオフしていくことによって，計算情報を分配・収集する機構を導入することで効率の低下を防ぐ方法を提案する．また，このハンドオフ機構を富士通の分散メモリ型並列計算機 AP1000 上に実現して，Linpack benchmark のプログラムを用いて予備評価を行った結果，16% の性能向上を得た．

2. Ninf RPC の概要

Ninf の基本システムは Client-Server モデルに基づく計算資源利用を支援する．計算資源はリモート計算ホスト上のリモートライブラリとして実現される．これを実現するために計算ホストでは Ninf サーバと呼ばれるサーバプロセスを設けると共に，ユーザにはサーバに接続するためのライブラリを提供して，クライアントが計算に必要なパラメータをサーバに送り，サーバの計算後，結果を受け取る手続きを簡素化する．

以下では Ninf RPC システムの構成要素について説明をする．

2.1 Ninf サーバ

Ninf サーバは Ninf 計算ホスト上で動作する．サーバ - クライアント間の通信の概要を図 **1** に示す．

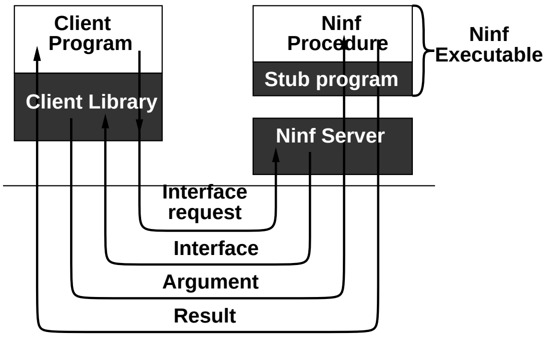

図 **1**　Ninf RPC の概要

Ninf リモートライブラリはネットワーク stub ルーチンとライブラリ本体をリンクした実行プログラムとして実現されており，サーバはこの実行プログラム群 (Ninf executables) の管理を司る．クライアントプログラムがライブラリを呼び出すと，Ninf サーバはその名前に関連付けされた Ninf executable を捜し出し，独立プロセスとしてフォークし，クライアントとの通信のための TCP/IP ストリームを渡し，Ninf executable のみで独立した通信を開始する．stub ルーチンは Ninf サーバとの通信手続きやクライアントの送り付けた引数のマーシャリング等を行い，ライブラリ本体への引き継ぎ作業を行う．Ninf executables のライブラリ本体は計算ホストで効率良く実行できるようにチューンされているのが通常である．

Ninf RPC の現在の実装では，クライアントとサーバ間の通信は信頼性を重視して TCP connection によって実現されている．また，異機種間利用を実現するために，データフォーマットとして Sun の XDR を採用している．

2.2 プログラミングインタフェース

Ninf クライアントのためのプログラミングインタフェースは `Ninf_call` だけである．`Ninf_call` の使用方法の説明として以下に行列積の例を示す．

```
/* 変数の宣言 */
double A[N][N],B[N][N],C[N][N];
/* 行列積 C=A*B ルーチンの呼び出し */
dmmul(N, A, B, C);
```

dmmul ルーチンが Ninf サーバで利用可能なら，クライアントプログラムでは dmmul 呼び出し部分を以下のように書き換えて Ninf ライブラリを利用できる．

```
Ninf_call("dmmul", N, A, B, C);
```

このようにクライアントではローカル関数を呼び出すのと同じように，関数の名前を指定して `Ninf_call` するだけでよい．`Ninf_call` は自動的に関数の arity を決定し，各引数の型を決定し，引数のマーシャリングを行い，サーバにリモート呼び出しし，結果を受け取り，結果を然るべき引数に格納するという一連の処理を行う．結果として，Ninf RPC のデザインではあたかもクライアントとサーバで引数が共有されているかのようにユーザには見える．

2.3 Ninf IDL

上記のプログラミングインタフェースを実現するためには，クライアントとサーバの間で関数に関する情報の共有ができていなければならない．しかも，広域に分散されて使われることを考慮すると，UNIX 標準の RPC のように，関数に関する情報の共有を静的にヘッダファイルとして与えるというのは現実的ではない．したがって，Ninf RPC ではクライアントがサーバにインタフェース情報を問い合わせ，その情報を利用する

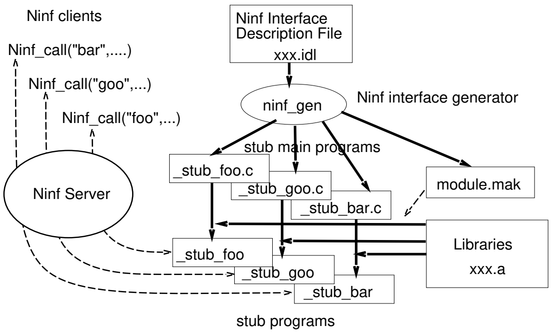

図 **2**　Ninf IDL とインタフェースジェネレータ

という動的手法を採用している．インタフェース情報はパラメータの数，引数の型，引数のアクセスモード(IN/OUT/INOUT)，引数のサイズ (Array などのサイズ)，引数のマーシャリング情報を含んでいる．

Ninf IDL (Interface Description Language) はこれらのインタフェース情報を記述する手段を提供する．Ninf ライブラリの提供者は予め IDL を使ってインタフェースを記述し，ライブラリを Ninf サーバに登録する手続きを踏む．

行列積のインタフェース記述は以下のように行う．

```
Define dmmul(long mode_in int n,
             mode_in double A[n][n],
             mode_in double B[n][n],
             mode_out double C[n][n])
"dmmul is matrix multiply for double precision",
/* このルーチンを含むオブジェクトモジュールの指定 */
Required "libxxx.o"
/* C の Calling convention を利用 */
Calls "C" dmmul(n,A,B,C);
```

`mode_in`, `mode_out` はアクセスモードの指定子である．引数として使われるベクタ・行列等のサイズ指定に使われる変数は，同じ引数の並びで `mode_in` で指定されている必要がある．また，このサイズ指定には単純な式も記述できる．その他，ライブラリの説明文，オブジェクトモジュールやリンクに必要なライブラリの指定，ライブラリの作成言語，リンク時オプション，別名等が記述できる．

図 2 に示すように，このインタフェース記述ファイルは `Ninf_gen` コマンドを使って stub ルーチンと Makefile に変換される．ライブラリ提供者は make を実行してライブラリのサーバへの登録を完了する．

3.　分散メモリ型並列計算機利用時の問題点

前述したように現状のサーバのインタフェースでは，Ninf executable は UNIX のプロセスとしてフォークされ，クライアントとの通信を独立して行う．このため，分散メモリ型並列計算機で利用する場合に以下に述べる問題が生じる．

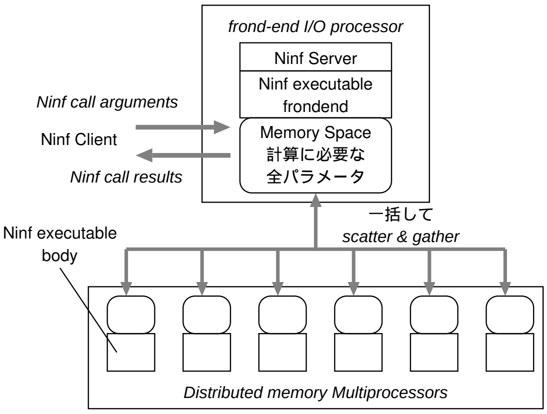

図 **3**　分散メモリ型並列計算機における Ninf server の構成

ベクトル型計算機や共有メモリ型並列計算機上でマルチプロセッサ対応の UNIX が動作している場合，共有メモリに Ninf RPC の引数を読み込み，単一プロセスをフォークしてその保護空間の中でマルチスレッドで並列プログラミングを行うことが可能であり，現状の Ninf の構造を変える必要はない．

一方，分散メモリ型並列計算機では，各プロセッサのメモリ空間内に適切にデータを配置した後に計算を開始する必要がある．従って，現状のシステムでは一旦フロントエンドマシンや I/O プロセッサ等の単一ノード上で動作するサーバ内のメモリ上に全てデータをバッファリングし，その上でさらに各ノードにメッセージ転送する必要がある．また，返値についても同様に単一ノードに回収，バッファリングし，クライアントに転送する必要がある (図 **3**)．

このような場合，サーバに負荷が集中してしまい，さらに引数データがサーバの実メモリサイズを越える場合は，大量のスワップが起きることにより，システム効率の低下が避けられない．結果として計算ホスト自体の性能を有効に利用できなくなってしまう．

4.　Ninf 基本システムへの拡張

前述の問題の対策として，分散メモリ型並列計算機における各ノードプロセッサへのデータの分配の仕様を Ninf IDL に記述できるように拡張する．フロントエンドマシン (I/O プロセッサ) と各ノードとの間の通信は，データ分散が IDL によって指定されていることによって，サーバのメモリリソースを大きく圧迫することなく，実施することができる．

以下では，IDL の拡張の詳細と 2 つのフロントエンド - ノード間通信戦略について述べる．

4.1　Ninf IDL の拡張

Ninf IDL を拡張して `Ninf_Call` 引数の読み書きモードが `mode_in`(read) の場合の初期データ分散方法，`mode_out`(write) の場合の最終データ回収方法，

`mode_inout`(read&write) の場合の分散・回収方法の指定が行えるようにする．

シンタックスは HPF4) の分散ディレクティブ `DISTRIBUTE` に類似のものを用いて拡張する．即ち，

```
Define sample(long mode_in int n,
              mode_in double A[n][n],...)
```

という例の行列 A を 2 つの次元についてブロック分割したい場合には，

```
mode_in distribute(BLOCK,BLOCK) double A[n][n]
```

と記述すると，配列を抽象的なプロセッサメッシュに配置することを意味する．サイクリック分割を指定する `CYCLIC`，分割しないことを指定する `*` や `BLOCK`, `CYCLIC` に指定できる引数とその意味についても同様である．

`distribute` が指定されている引数は指定された分散方法に従って，データがプロセッサに配置されるものとして，初期分散と回収を行う．

`distribute` が指定されていない引数は，`mode_in` の場合は全ノードに放送し，`mode_out` の場合は計算終了後に最初のノードから受け取る．また `mode_inout` の場合は全ノードに放送し，計算終了後に最初のノードから受け取る．

4.2 フロントエンド - ノード間のデータ通信戦略

4.2.1 buffering-base method

オリジナルではサーバ側で無条件にデータ格納用のメモリを確保しており，この領域を直接作業領域として使用しない分散メモリ計算機では意味がない．buffering-base method では，メモリを動的に確保する代わりに固定サイズの専用バッファを設ける．このバッファはリングバッファを用いて再利用されるので確保・解放手続きのオーバーヘッドはない．

クライアントから送信の場合，送られてくるデータが XDR でエンコーディングされているので，フロントエンドはこの表現を使用する計算機の表現に変換して，専用バッファに格納する．このバッファが一定量に達するとフロントエンド主導で各ノードに分配を行う．

逆に値を返す場合は，フロントエンドから各ノードにデータ要求メッセージを送り，ノードからのデータを受信してバッファに蓄える．受信データが一定量に達するとクライアントへの転送を開始する．

4.2.2 hand-off method

各ノードに分配されるべきデータが転送される場合にのみ，そのノードとクライアントが直接通信コネクションを張り，クライアントから直接ノードにデータが転送されるようにする．この際，クライアントとのコネクションは，フロントエンドの制御の下に，適切にノードからノードにハンドオフされていく．逆に値を返す場合も分配されたデータを同様に各ノードから直接返されるようにする．

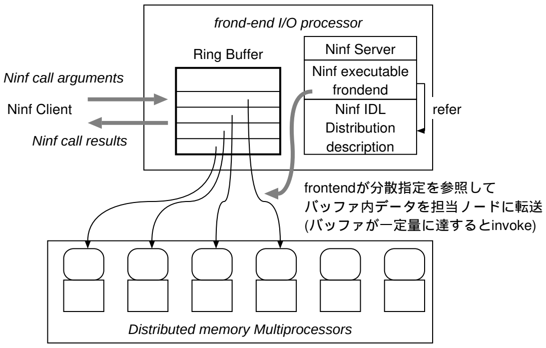

図 4 buffering-base method

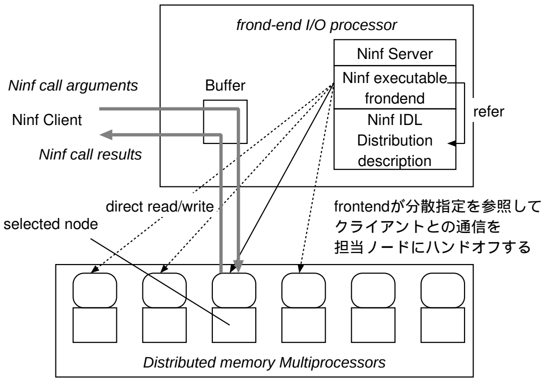

図 5 hand-off method

ワークステーションクラスタのように構成ノードプロセッサが外とのネットワークインタフェースを有している場合は，ノードからクライアントにコールバックすることによってサーバを経由せずに直接コネクションを樹立可能である．

富士通 AP1000 をはじめとする多くの分散メモリ型並列計算機のように構成ノードプロセッサが外とのネットワークインタフェースを有していない場合は，クライアントと直接通信できるのはフロントエンドだけである．したがって，フロントエンドはクライアントと接続するノードを選択・制御すると共に，パケットサイズ程度の少量のバッファを設けて両者の通信のリフレクタとして動作する (図 5)．

buffering-base method との大きな違いはデータ表現の変換等の処理をノード単位で行えるため，通信の多重化によってサーバ処理の負荷分散が実現できるという点である．

5. 評　価

Ninf 基本システムの富士通の分散メモリ型並列計算機 AP1000 への移植を行い，その上で前述した buffering-base method と hand-off method のコア部分を実装し，予備評価を行った．

本稿の時点では Ninf IDL への分散指定子

`distribute` の拡張は未実装である．従って両手法を用いた場合の効果についてのみ検討することにする．

5.1 評価環境

Ninf の計算サーバマシンとして富士通の分散メモリ型並列計算機 AP1000 (64PE) を利用した．AP1000 のフロントエンドマシンは SunOS 4.1.x ないし 5.x が運用されているので TCP/IP ベースで実装されている Ninf システムの移植は容易であった．クライアントマシンとしては SparcStation 20 502 を利用した．

比較のために，フロントエンドのデータハンドリング手続きとして以下の 3 つの方法を用意した:

オリジナル Ninf call の引数に渡される通信データのサイズに応じたメモリ領域を無条件で実行時に確保・解放する．通信データの XDR と SPARC の表現の間の変換処理はフロントエンドで行われる．

buffering-base method フロントエンド上に 1Mbytes 固定のリングバッファを用意し，バッファの 80% を超える量のデータが格納された時点でフロントエンドとノードの通信を invoke する．ノードへの送信データの場合は alignment を考慮し，ノードへ送るデータ全部が揃っていない部分については送信を見送り，残りが到着してから一括送信される．通信データの XDR と SPARC の表現の間の変換処理はフロントエンドで行われる．

hand-off method フロントエンド上に Ninf call の各引数の情報と各ノードの通信状況を表すテーブルを用意して，クライアントと通信するノードの管理 (選択・制御) を行う．通信自体は 16Kbytes のバッファを用意しておく．ノードへの送信の場合はノード側が受信完了判定を行い，完了するまでフロントエンドは順次クライアントからデータを受け取り，ノードに送信する．逆も同様である．通信データの XDR と SPARC の表現の間の変換処理は各ノードで行われる．

buffering-base method, hand-off method とも通信の二重化は必要と考えられるが現時点では未実装である．

評価には倍精度の LINPACK benchmark のプログラムを用い，対象とする行列は 200×200 から 1200×1200 の範囲で，LU 分解コア + 後退代入ルーチンについて測定した．この測定用ルーチンの Ninf IDL 記述を図 **6** に示す．

5.2 結果

図 **7** に SS20 単体および 3 つの方法で Linpack benchmark を実行した場合の実行性能を示す．また，実行時間の内訳を表 **1** に示す．

行列サイズが小さい ($n < 400$) 領域では buffering-base method，hand-off method ともに original に及ばない．特に hand-off method では手続き自体のオーバーヘッドのために効率が低い．

buffering-base method は単純に original のバッファ管理を効率良く行うだけであるため，サイズが大きい領域で若干上回るだけである．表 1 に示されるように通信データの XDR と SPARC の表現の間の変換のためにフロントエンド上で費す時間が双方とも大きいためと考えられる．

```
Module Aplinpack;
Library "aplinpack.o";

Define ap_dgefasl(
        mode_in distribute(BLOCK,BLOCK) \
                double a[n][lda:n],
        mode_in int lda,
        mode_in int n,
        mode_in int ipvt[n],
        mode_inout double b[n],
        mode_in int job)
"Gaussian elimination & Backward subst routine"
Calls ap_dgefasl(a,n,n,ipvt,b,job);
```

図 **6** 測定で用いた Ninf IDL

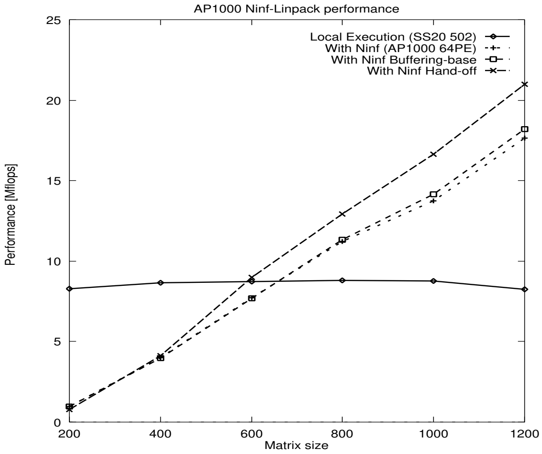

図 **7** Linpack benchmark による実行性能の比較

hand-off method では変換をノード上で行える．従って，ノードではデータを受け取り，コネクションが次のノードに渡った後，変換手続きを行うパイプライン実行が可能となり，フロントエンドでの処理が律速となることを防げる．AP1000 のようにノードプロセッサが SPARC 25MHz と比較的低速の場合でも 16% 程度効率が向上しており，ノードプロセッサがフロントエンドマシンに比べて十分速い場合には，この方法を採る利点はさらに大きいと考えられる．

6. 関連研究

Ninf に類似した Global Computing を実現するプロジェクトとして NetSolve[5] がある．NetSolve は我々の `Ninf_call` と同様，使いやすい API を提供するが，インタフェース記述言語を備えていないため，広域に分

表 1　実行時間の内訳

n=200	*client*	*server*	*node*	*calc*
original	5.7	2.7	1.6	0.27
buffering	5.7	2.7	1.6	0.27
hand-off	6.9	3.9	3.3	0.27
n=400	*client*	*server*	*node*	*calc*
original	10.7	7.6	3.4	0.95
buffering	10.8	7.7	3.5	0.95
hand-off	10.4	7.3	5.9	0.95
n=600	*client*	*server*	*node*	*calc*
original	18.7	15.5	6.1	2.43
buffering	18.8	15.7	6.2	2.43
hand-off	16.1	13.0	11.6	2.43
n=800	*client*	*server*	*node*	*calc*
original	30.5	27.3	10.4	5.04
buffering	30.2	27.0	10.5	5.04
hand-off	26.5	23.4	21.2	5.04
n=1000	*client*	*server*	*node*	*calc*
original	48.7	44.6	16.7	9.72
buffering	47.3	42.9	17.3	9.72
hand-off	40.2	36.6	34.2	9.72
n=1200	*client*	*server*	*node*	*calc*
original	65.5	62.1	25.0	16.6
buffering	63.4	60.3	25.9	16.6
hand-off	55.0	51.3	48.6	16.6

client	Ninf_call を呼び出してから完了するまでの時間
server	サーバが Ninf_call を受付けてから完了するまでの時間
node	各ノードでの処理時間 (計算時間を含む)
calc	各ノードでの正味の計算時間

散された計算資源のインタフェース情報の共有や保守が難しい．また，超並列機の運用は考慮されていない．

Legion[6)]プロジェクトは，広域分散システム上でのプログラミングを支援するオブジェクト指向言語 Mentat[7)]を使って，多くの計算資源を統合した仮想計算機を実現する．Mentat はプログラミング言語であるため，分散システム特有の最適化や機能が実現が容易である．例えば，特定のクラスのメソッドが自動的に負荷分散対象になる機能を使って，ユーザは分散処理部分を指定する．これに対し，Ninf は特定の言語を仮定しないため，ユーザにとっては既存のシステムとの連続性に優れるが，大規模な分散並列計算の実現には限界がある．

Fortran の並列拡張標準である HPF[4)] は，データ並列を支援するために配列の分散配置手法が整備されている点が特徴的である．HPF では Alignment，Distribute, Mapping の 3 段階で行う．Alignment は複数の配列の配置の相関を指定し，指定された配列間の操作は，コンパイラによって実行時の局所性が保証されて高速に行えることを期待する．Distribute は配列をユーザが指定する抽象的なプロセッサメッシュへ配置し，Mapping は実プロセッサへ配置する．本稿の IDL の拡張では Distribute 指定のみを導入したため，引数の複数の配列間の Alignment は指定できない．Alignment はコンパイラの最適化のためには不可欠だが，サーバとクライアントとの通信手続きを支援するインタフェース記述としては強力すぎると判断したためである．もちろん，alignment 指定を IDL に含めることは容易である．

7. おわりに

本稿では，分散メモリ型並列計算機において Ninf システムを効率良く運用するために，初期・最終データ配置を記述するための API を提供し，その情報を使って計算情報を各ノードに対して順次ハンドオフしながら分配・収集する機構を導入する方法を提案した．また，この機構を富士通の AP1000 上に実装して有効性について検討を行った．

今後は未実装である API への拡張を含めた，実装の安定性，完成度を向上させるとともに，他プラットフォームへの移植を予定している．

謝辞　本研究を進める上で討議頂いたお茶の水女子大学長嶋雲兵教授に感謝する．

参考文献

1) Sekiguchi, S. et al.: —Ninf—: Network based Information Library for Globally High Performance Computing, *Proceedings of Parallel Object Oriented Methods and Applications(POOMA)* (1996).

2) Nakada, H., Sato, M. and Sekiguchi, S.: Ninf RPC Protocol, Technical Report TR95-28, Electrotechnical Laboratory (1996).

3) Nakada, H. et al.: A Metaserver Architecture for Ninf: Networked Infromation Library for High Performance Computing, *IPSJ SIG Notes 96-HPC-60* (1996).

4) High Performance Fortran Forum: *High Performance Fortran Language Specification Version 1.1* (1994).

5) Casanova, H. and Dongarra, J.: NetSolve: A Network Server for Solving Computational Science Problems, Technical report, University of Tennessee (1996).

6) Grimshaw, A. S. et al.: Legion: The Next Logical Step Toward a Nationalwide Virtual Computer, Technical Report CS-94-21, University of Virginia (1994).

7) Grimshaw, A.S.: Easy to Use Object-Oriented Parallel Programming with Mentat, *IEEE Computer*, pp. 39–51 (1993).