ハイパフォーマンス
コンピューティング 55−11
（1995. 3. 9）

ネットワーク数値情報ライブラリ Ninf の通信方法の予備的考察

飯岡 美恵、新居 由佳子、長嶋 雲兵、関口 智嗣*、佐藤 三久*、松岡 聡**、細矢 治夫

お茶の水女子大学、* 電子技術総合研究所、** 東京大学

Email:{*g9120503, g9120529, umpei, hosoya*}*@is.ocha.ac.jp*, {*sekiguti, msato*}*@etl.go.jp*, *matsu@is.s.u-tokyo.ac.jp*

関口らの提唱したコンピュータネットワークを利用した数値情報ライブラリ Ninf のシステムのモデルであるクライアント－サーバ方式の実現方法の検討を行った。ネットワーク上でのメッセージパッシングの実現手段として UNIX 標準の TCP/IP と メッセージパッシングライブラリの PVM を考え、それぞれを用いてシステムを試作し、プログラミングの容易さおよび関数呼び出しの実行時間の比較・検討等を行った。TCP/IP を用いたプログラムは PVM を用いたプログラムに比べ約 10 倍程度高速であったが、様々なエラーチェック等のプログラミングが必要であり、労力が大きい。

Evaluation of performance and reliability of client-server system coded by TCP/IP and PVM for Ninf system.

Mie IIOKA, Yukako NII, Umpei NAGASHIMA, Satoshi SEKIGUCHI*, Mitsuhisa SATO*, Satoshi MATSUOKA**, Haruo HOSOYA

Ochanomizu University, *Electrotechnical Laboratory, **University of Tokyo

Email:{*g9120503, g9120529, umpei, hosoya*}*@is.ocha.ac.jp*, {*sekiguti, msato*}*@etl.go.jp*, *matsu@is.s.u-tokyo.ac.jp*

A prototype of a numerical information library system based on a high performance wide area network (Ninf) was developed to evaluate performance and reliability of communication using TCP/IP and PVM. Though client-server system constructed by TCP/IP is almost 10 times faster than that by PVM, programming cost is quite high because many overhead and error treatment should be coded by the user. On the other hand, client-server program is easily made using PVM.

1 はじめに — ネットワーク数値情報ライブラリ：Ninf —

コンピュータネットワークの発展に伴い、ネットワーク利用による情報提供サービスが盛んに行われるようになった。ネットワークを利用した情報提供サービスの利点は、情報の存在場所への非依存性・情報入手の即時性・情報保持の不必要性などである。これらは、情報入手の確実性が保証されれば保存法や保存場所を考慮する必要はないので、情報資源の仮想的共有化と言うことができる。

数値計算の分野における情報資源の仮想的共有化について考えてみる。この場合に必要となる情報資源には、数値定数や、特種関数・数値計算ライブラリなどにより計算された値が挙げられる。現在、これらの情報資源は文書やプログラムという静的な形で共有されている。そのため、ftp 等による情報入手や自分の計算機環境に合わせたプログラムインストールなど、ユーザが独自に配慮すべき点が多く、またプログラムのバージョンアップ等への対応も通常は遅れがちである。これでは理想的な情報資源の仮想的共有化には程遠い。

これに対し、情報資源を文書やプログラムという静的な形ではなく、数値情報そのものを動的な形で共有することを考える。こうすれば数値定数や関数計算値などの数値情報についてそれぞれ高精度なものを選択し、ネットワーク上でそれらを仮想的に共有することにより、数値データの入手やプログラムバージョンアップなどに対するユーザの配慮は減少するものと思われる。

そこで関口らは、高速ネットワーク上で数値情報を動的に共有することにより、数値計算における理想的な情報資源の仮想的共有化を行い、数値計算で必要な数値情報を常に最新かつ信頼性の高い状態で即時に入手することが可能なネットワーク数値情報ライブラリ Ninf (Network based information library for High Performance Computing) を提唱した[1] 。このシステムでは、数値定数や特種関数の必要な精度に応じた値などの数値情報提供のサービスを行う。インターネット上に複数のサーバを設定するので、遠隔地のユーザは Ninf ライブラリ関数を呼び出すだけでサーバに蓄えられた高精度な数値情報を入手できる。

現在では、Mathematica や Lotus 1-2-3 などのアプリケーションソフトの開発により、C や Fortran などのプログラミング言語を使用せずに簡単に数値計算を行うことが可能となりつつある。これをプログラミング言語の隠蔽と言うならば、Ninf を利用することにより計算機の隠蔽が可能となる、と言うことができる。計算機の存在場所についてはネットワークの充実に伴い既にその隠蔽が進んでいるが、Ninf システムでは実際に数値計算を行っているのがスーパーコンピュータであるのかワークステーションであるのかなど、計算機の種類までも隠蔽することを可能とすることを目指す。

この Ninf は、構造的にネットワーク上に置かれたクライアント－サーバシステムとなっている。そのため、クライアント－サーバ間のメッセージパッシングが必要不可欠となるので、本研究では Ninf 実現の第一歩として Ninf システムにおけるクライアント－サーバ間のメッセージパッシングに着目し、その実現方法についての予備的考察を行った。ネットワーク上でのメッセージパッシングを実現する方法としては、直接ソケットを取り扱い、非常に低レベルな部分からプログラミングを行う方法と、既存のメッセージパッシングライブラリを利用し、ネットワークの細部を考慮せずにプログラミングを行う方法の 2 通りが考えられる。そこで、ソケットを直接取り扱う方法として UNIX OS のワークステーションで標準として採用されているプロトコルである TCP/IP を用いた。また、メッセージパッシングライブラリを利用する方法として PVM [2] を用いた。それぞれを用いてクライアント－サーバシステムを試作し、プログラミングの容易さやユーザの関数呼び出しに要する時間の比較・検討等を行なった。

2 試作したシステムの概要

試作した Ninf システムは図 1 に示すように、大きく分けて Ninf サーバプログラム・ Ninf クライアントプログラムの 2 つのプログラムから成っている。Ninf サーバは数値情報の管理を行い、 Ninf クライアントはサーバとユーザプログラムとの仲介を行う。

ユーザプログラムは Ninf に対してライブラリ関数を呼び出し、数値情報の要求を行う。ユーザプログラムにおいて要求された数値情報に関する情報はクライアントを介してサーバへ渡される。サーバは受け取った要求に沿った処理を行い、その結果をクライアントを経由してユーザプログラムへ送る。

2.1 Ninf サーバプログラムでの処理

Ninf サーバプログラムでは定数検索と関数計算の実行の管理を行う。現在、サーバに蓄えられている数値情

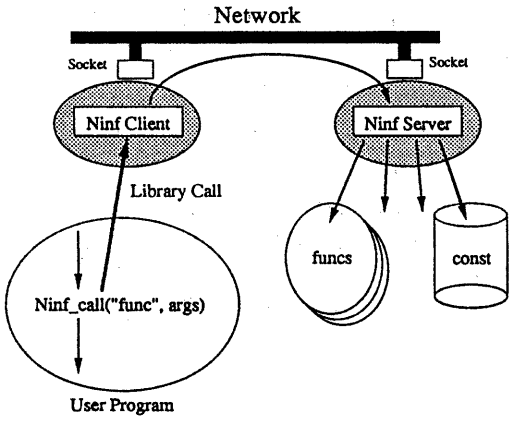

図 1　Ninf Client Server System

報はとりあえず定数約 100 種類、関数約 20 種類である。

物理定数などは、参考文献 [3] から採用し、関数は C の組み込み関数をデータベースに登録した。これらの数値情報のうち、定数情報は定数名とその値をファイルに保存した。定数値は 16 バイト用意されており、サーバは起動時にこのファイルから保存されている定数名とその値を char 型で読み込み、定数名をキーとして高さバランス木 (AVL-tree) [4] を作成する。今回試作したシステムでは数値情報の種類が少ないので、探索法の違いによるオーバーヘッドは無視できる。しかし、実際の Ninf システムでは例えば理科年表や化学便覧等のデータを全てサポートすることを考えると、高速なバックエンドデータベースが必須であると考えられる。

また関数情報に関しては、TCP/IP の場合には関数計算の実行可能ファイルへのポインタを保存しており、PVM の場合にはファイルに実行可能な関数名およびその関数計算の実行に必要な引数の個数を保存している。実際に関数計算を実行する際には、TCP/IP ではそのポインタへ制御を渡し、PVM ではファイルから読み込んだ情報をもとに関数計算を実行する子プロセスを生成する。

2.2　Ninf クライアントプログラムでの処理

Ninf クライアントプログラムは、ユーザプログラムとサーバとの仲介を行う。

まず、ユーザプログラム内でライブラリ関数呼び出しが行われると、クライアントはその関数の第一引数として与えられた定数名または関数名を解析し、定数名であるか関数名であるかの判断を行う。ユーザの要求が関数名であった場合には、続けてその関数計算に必要な個数の引数を解析する。そして、それらの情報をまとめてサーバへ送る。

サーバは、クライアントによって送られてきた情報をもとに定数の検索または関数を実行させ、数値情報をクライアントに返す。現段階では、定数検索の場合には char 型で、関数計算実行結果の場合には double 型で返される。その後クライアントは、サーバから送られてきた検索・実行結果を受け取り、正確な処理が行われたかどうかを調べる。これは、実際の検索・実行結果と同時に送られてくる文字列により判断する。この判断の結果、ユーザの要求にエラーが存在した場合には、ユーザプログラム側にエラー状態を表示すると同時にユーザプログラムへエラー状態を表す戻り値 -1 を返す。ユーザの要求にエラーが存在せず正確な処理が行われていた場合には、ライブラリ関数呼び出しの引数として与えられた、ユーザの要求する型に従って数値の型変換を行う。この時、ユーザが要求することのできる型は char・short・long・int・float・double の 6 種類である。これらはそれぞれ大文字または小文字での指定を可能とした。ユーザの要求した型が指定可能なものではなかった場合はエラーメッセージを表示し、ユーザプログラムにエラー状態を表す -1 を返す。ユーザが正しい型を要求した場合には、サーバから受け取った検索・実行結果をその型へ変換し、その変換結果をユーザプログラムへ渡して処理を終了する。このとき、正常終了を表す戻り値として 0 を返す。

2.3　ライブラリ関数：Ninf_call

ユーザプログラム内で呼び出すライブラリ関数名は Ninf_call とした。Ninf_call の仕様は以下の通りである。

1. 定数値
   ```
   status = Ninf_call("const_name",
   "type", &ans);
   ```
   const_name：定数名
   type：型名
   &ans：検索結果を格納する変数のアドレス

2. 関数の計算値
   ```
   status = Ninf_call("func_name",
   ```

```
args..., "type", &ans);
```

func_name：関数名
args：関数計算実行に必要な引数
type：型名
&ans：計算結果を格納する変数のアドレス

ユーザは Ninf_call の引数に要求する数値情報についての必要最小限の情報を与えるが、要求が定数検索であるか関数計算実行値であるかによって引数の個数が変化する。また関数計算実行値の場合でも、その関数の種類によってはいくつかの引数が必要である場合があるので引数の個数が変化する。そこで、Ninf_call の引数は可変長とした。

Ninf_call の第一引数にはサーバに検索・実行を要求する定数名または関数名を文字列で与える。第二引数以降には、要求が関数計算値である場合には、まずその関数を実行するために必要な引数を必要な個数だけ与える。次に、結果をどの型で入手するかの要求を与えるが、これは先程述べた 6 種類の型を文字列で指定することにより行う。そして最後に、サーバで実行した結果を格納する変数のアドレスを与える。定数検索を要求する場合には、要求する定数名の次に変換を要求する型名、そして最後に値を格納する変数のアドレスを与える。

TCP/IP と PVM を用いて上記の機能を持つシステムを試作し、その通信時間の測定・比較を行った。次にその方法および結果について説明する。

3 実行速度の測定

3.1 測定方法

試作したシステムを用いてデータの通信時間の測定を行なった。測定を行なった時間は CPU time・real time・system time の 3 種類である。CPU time は C の組み込み関数である times を用い、real time と system time は、UNIX のコマンドである time を用いた。times 関数は 1/60 秒の単位で、time は 1 秒単位で測定され、小数点第一位まで出力される。

測定にはお茶の水女子大学理学部情報科学科教育用システムの Toshiba AS4040 (SUN4 ipc, 24MHz, memory 24MB) 2 台を使用し、1 台をサーバに、もう 1 台をクライアントとした。

測定は本システム以外の job が走っていない環境で行い、ユーザプログラムで Ninf_call を 50 回 call するのを行うのにかかった時間を計測した。ユーザプログラムでは Ninf_call のエラーに対する処理は行なっていない。

測定を行なったユーザプログラムを図 2 に示す。

```
times();
Ninf_call ("pi","double",&ans);
Ninf_call ("avogadr","int",&ans);
        :          :          } 50 call
Ninf_call ("grav","float", &ans);
times();
```

図 2 測定に使用した user program

Ninf_call は数値定数を要求する場合と関数値を要求する場合に分けて測定した。Ninf_call を呼び出す場合、数値定数は現在蓄えている数値定数約 100 個から適当に選択した異なる 50 個を要求し、関数値はサーバ側に蓄えている関数約 20 個から要求回数が均等に、同様のデータを連続して要求しないように行った。また、数値データの型変換は double 型・float 型・long 型・int 型・short 型・char 型の中から、任意に選んだ。

CPU time は 50 回の Ninf_call の直前と直後の時間の差とし、times 関数から求めた。real time と system time は 50 回の Ninf_call にかかった時間とし、time コマンドから求めた。

以上の手順に従い、数値定数と関数値において測定を 10 回行った。その平均を求めた後、それぞれについて Ninf_call 1 回にかかる CPU time・real time・system time の平均時間を求めた。この測定結果を用いて、数値定数を要求する場合と関数値を要求する場合に測定した時間の平均を求た。

3.2 結果

結果を表 1 に示す。表 1 の constant と function は、それぞれユーザプログラムからの要求が数値定数、関数値である時のデータの通信時間を表し、mean は constant と function の測定値の平均を表したものである。

TCP/IP で作成したシステムおよび PVM で作成したシステムとも constant と function で大きな違いは見られなかった。これは、両者とも通信と通信のオーバーヘッドが時間の大部分を占めているためである。表 1 より、データ通信時間 (mean で) は TCP/IP を用

単位：sec

		CPU time	real time	sys. time
TCP/IP	constant	0.0038	0.05	0.004
	function	0.0043	0.05	0.005
	mean	0.0041	0.05	0.005
PVM	constant	0.0412	0.28	0.033
	function	0.0392	0.68	0.032
	mean	0.0402	0.48	0.033

consant： 数値データの通信時間
function： 関数値の通信時間
mean： 両者の平均

表 1　データ通信時間の測定結果

いて試作したシステムは PVM を用いたシステムより CPU time は約 10 倍、real time は約 9 倍、system time は約 6 倍短かいことが分かる。ただし、クライアント上での組み込み関数の実行は高々 10 μsec であるので、たとえ TCP/IP を用いたとしても、通信を用いた計算は 40 倍程度遅い。PVM ではさらにそれの 100 倍程度遅いことが分かった。

3.3　考察

TCP/IP と PVM を用いて試作したシステムについて、データの通信速度を測定した結果、実行時間では TCP/IP を用いて試作したシステムは PVM に比べ約 10 倍通信時間が短かかった。PVM ではデータの通信においてデーモンが間に入ったり、プロセスの起動などの様々な処理を行っている。それが今回試作したシステムにおけるデータ通信時間の測定結果に顕著に現われたと言える。

今回は、お茶の水女子大学内でデータ通信を行ったが、実際の Ninf システムの通信環境となる、異なるネットワーク間でのデータ通信時間の差について考えると、より多くの通信時間が必要となる。

たとえ TCP/IP を用いたとしても、簡単な関数計算を行うことを考えると、通信を用いた組み込み関数の計算の性能は遅すぎる。現在は実験的に C の組み込み関数が登録されているが、むしろ計算コストの大きな関数を登録すると Ninf の長所が出ると考えられる。しかしその場合、以下のような技術的課題が生じる。

- 引数や結果のデータセットがポインタを含むような複雑なデータ型の場合、異機種間でどのようにその違いを吸収するか。また、データ型の整合性をどう判断するのか。
- 引数や結果のデータセットが膨大な場合、どのように WAN 上を効率的に転送し、バッファリングするのか。
- 同じデータセットに対して複数回の演算を施す場合、コネクションをどのように張り続けるか。特にデータセットが大きい場合には、連続演算が可能であるのは必須である。
- サーバは受け付けられるデータセットの大きさに限りがあるとか、処理が重すぎると判断した場合など、どのようにクライアントに応答するのか、それに対してクライアントはどのように反応するのか（例えば、自動的に他のサーバに計算要求が委譲される、など）。
- クライアントはサーバがフェイルしたことをどのように判断するか。
- サーバが計算中にフェイルした場合、クライアントが同時にフェイルしないで再試行を含む別な手段で計算を続行するにはどのようにするか。特に膨大なデータセットがサーバに転送された後、複数回の演算を施す途中にフェイルした場合が肝心である。何らかのロールバックができるようにするのか。
- 複数の計算の要求がある場合、サーバはどのようにそれをスケジューリングすべきか。特にサーバが並列計算機の場合はどうするか。

他にもいくつかの課題が考えられようが、これらを解決するのが今後の研究の方向であると考えている。

4　プログラミングコスト

プログラミングコストにおいては、TCP/IP ではプリミティブな通信処理のみをサポートしているため、通信において低レベルな段階から全てプログラマが書かねばならず、プログラミングが複雑で量も多くなるという弱点がある。そのため、プログラミングコストは高い。一方、PVM ではプリミティブな処理は PVM 自身が機械的に処理するため、プログラマの配慮すべき点が少なくなり、プログラミングが簡単になる。

TCP/IP と PVM のプログラミングの複雑さの違いを例を挙げて説明する。図 3 は、TCP/IP を用いて試作したシステムのサーバプログラム中でデータをク

```
int buf_send( int soc, RESULT send_data)
{
        char *dpt;
        char rebuf[5];
        int    i, sizenum, num;

        sizenum = sizeof(send_data);
        dpt = (char*)(&sizenum);
        if((num = write(soc, dpt, sizeof(sizenum))) < 0){
                fprintf(stderr, "ERROR: BUF_SEND: WRITE \n");
                return -1;
        }
        for( ; ; )
                if(read(soc, rebuf, sizeof(rebuf, "yes", 3)) == 0)
                        break;
                else
                        return -1;

        dpt = (char*)(&send_data);

        if(write(soc, dpt, sizeof(send_data)) < 0){
                fprintf(stderr, "ERROR: WRITE <BUF_SEND> \n").
                return -1;
        }

        return 0;
}
```

図 3　TCP/IP のデータ送信用関数

ライアントへ送る処理を行っている関数 buf_send である。

図 3 の最後の write 関数が実際にデータを送信するものである。しかし実際の送信を行う前に、これから送信するデータのバイト数をクライアントへ送る作業を行っている。特に異種ネットワーク間で可変長のデータを送信する場合、通信途中でデータが途切れる可能性があるため、全てのデータを受け取ったかどうかをクライアントが判断しなければならないからである。PVM を用いたシステムではこのような作業は明示的に行う必要がなく、pvm_send という一つの関数を呼び出すだけでデータ送信が可能である。

このことから TCP/IP を用いてプログラムを作成する場合には、PVM 関数一つに相当する処理を行うための多くの手続きを記述しなければならないことが分かる。従って、PVM と同程度の信頼性を持つように TCP/IP を用いてプログラミングを行うのはコストが高く、Ninf ではクライアントプログラマに対してその差を吸収することを目標とする。

また、TCP/IP は低レベルな処理を取り扱うためエラー処理も低いレベルから行わなければならない。Ninf システムにおいては信頼のおけるデータ通信が必要であり、エラー処理が必須である。今後 TCP/IP のような低位のプロトコルを用いる場合は、ユーザにそのエラー処理のプログラミングコストを負担させるべきではなく、Ninf のライブラリが吸収すべきである。つまり、ライブラリプログラマが新たな関数や定数表をシステムに追加する場合は、エラー処理を全く意識しないか、または非常に簡便で統一的なエラー処理のプログラミングモデルが提供されるべきである。この場合、以下の技術的な課題がある。

- 低位のプロトコルによって異なるエラー処理をどのように統一的に扱うか。
- 性能をいかに維持するか。
- 性能の維持が困難な場合、どのようにユーザに様々なエラー処理に対する選択肢を与えるか。

これらは、先に述べた様々なフォルトへの対策と相まって、複雑なソフトウェア機構を要するが、それぞれの要素技術は分散システムにおいて培われたものも多い。今後は、それらをどのように Ninf の目的に従って取り入れ、あるいは改良し、さらに融合していくか、という点が研究上の課題となる。

PVM を用いると、プログラムを作成することは非常に簡単であった。以上のことから、Ninf システムにおいても信頼性の高いシステムの作成を容易にするのは PVM であるといえる。しかし、PVM を用いて通信を行なう場合、クライアントとサーバの両方で PVM が起動されている必要がある。現在、一般にクライアントとサーバの両方で PVM が常に起動されていることを想定するのは難しく、これが PVM を用いた通信の弱点となっている。その上、PVM がインストールされていない環境で Ninf システムを利用するには、Ninf システムの利用に際し、PVM のインストールおよび PVM のバージョンアップへの対応もユーザが独自に行う必要がある。また PVM を用いる場合、新たにユーザが Ninf の利用を開始するに当たり、サーバ側で新しいクライアントを登録し、クライアントとして Ninf を利用する許可を与えなければならない。

5　まとめ

ネットワーク数値情報ライブラリ "Ninf" を実現するにあたり、Ninf の通信モデルである、クライアント－サーバシステムの実現方法を考察した。通信方法として、TCP/IP を用いてプリミティブなクライアント－サーバシステムを作成することと、並列・分散処理用の

メッセージパッシングライブラリである PVM を用いて同様のシステムを作り上げることを行った。これらのうちどちらが Ninf システムの通信方法としてふさわしいかを判断するため、データ通信時間とプログラミングの容易さの点から両者を比較し、検討を行った。

TCP/IP と PVM の両方を用いて Ninf システムを試作し、データ通信時間の測定を行った結果、TCP/IP を用いて試作したシステムは PVM を用いて試作したシステムより、CPU time において約 10 倍程データ通信時間が短かかった。これは、PVM が通信を行う際にはデーモンが間に入ってきたり、プロセス起動の必要があるためである。

プログラミングする上では、TCP/IP はデータ通信の低レベルな処理を行うため処理が複雑になり、エラー処理などプログラミング時に配慮すべきことが多いと分かった。

6 今後の課題

本研究を進める課程で、いくつかの解決すべき課題が明らかになってきた。以下、簡単にそれらをまとめて提示する。

今後の課題として、クライアント－サーバシステムの実現方法を検討する上では、TCP/IP、PVM 以外のクライアント－サーバシステム構築の際に利用の考えられるライブラリや言語について性能評価を行い、Ninf の通信方法としてより適するものを選択する必要がある。現在そういったシステムを探す努力を行っている。

また、今回作成したシステムに関しては、様々なネットワーク間で TCP/IP と PVM を用いて試作したシステムについての性能調査を行うということが挙げられる。今回はお茶の水女子大学ネットワーク内でのデータ通信時間の測定を行ったが、実際には Ninf システムのデータ通信の行われる異種のネットワーク間で性能調査をする必要がある。

また、Ninf に登録する関数としては、現状のネットワークの性能を考慮すると計算コストが組み込み関数の 10 ～ 100 倍大きなものが有意義である。その場合には以下のような技術的課題が挙げられる。

- 複雑なデータ型の違いの吸収およびデータ型の整合性
- 大容量のデータセットの効率的転送およびバッファリング
- 同一データセットに対する複数演算でのコネクション維持
- サーバ上の計算機資源とユーザの要求する計算機資源の不一致時の処理
- クライアントへのサーバフェイルの通知方法および対処方法
- 複数の要求に対するサーバ上でのスケジューリング方法などの処理

等である。

また、新規関数などの追加に伴うエラー処理へのライブラリプログラマの配慮を最小限に押さえるため、エラー処理の統一化や性能維持方法、また性能維持が困難な場合の対処法などが挙げられる。

さらに、試作システムを作る上で明らかになった Ninf システム全体の課題として、多種類の数値データの格納方法を考察する必要がある。現在用いた数値データは非常に限られた種類のデータになっているが、現段階においても物理定数や電子密度等が分類されずに保管されており、検索が非効率である。検索を効率的に行うため、数値データは明確に分類する必要がある。しかし、実際のところ多岐の分野に使用される数値データも多く存在し、分類をどう行うかを決定するのは難しい問題である。検索を効率良く行うため、探索法の違いによるオーバヘッドを考慮した高速なバックエンドデータベースが必要であると考えられる。

また数値データには様々な単位を持つものが多く存在し、数値データをどのような単位で保存するか、数値データの単位指定の要求に応じるか、という問題も考えられる。単位指定による要求が可能であれば、ユーザが単位変換を行う必要がなくなる。しかし、単位要求に応じる場合、数値データの単位変換時に誤差が発生すると考えられ、高精度な数値データの供給が難しくなる。

また、有効桁数の要求に応じることのできるシステムが必要となる。一般に関数計算を行う上で、必要な有効桁数に適するアルゴリズムを用い計算することは、大変な労力を要する。有効桁数に適切なアルゴリズムを用いた計算を自動的に行い、計算値の有効桁を保証するシステムの実現は、この労力削減を可能とする。一方数値定数においては、数値定数自身に有効桁がある。数値定数使用時に有効桁数を知る必要があり、有効桁数を情報として与えることは非常に有用である。また、単位要求に応じた場合、数値定数の単位変換時に発生する誤差は有

効桁を情報として与えることにより、有効桁数の精度を保証し信頼性を高めることができる。

以上のような課題が明らかになったが、1つ1つ解決し、Ninfシステムを実現していきたい。

参考文献

[1] 関口 智嗣, 佐藤 三久, 長嶋 雲兵, "ネットワーク数値情報ライブラリ：- Ninf の設計 -", IPSJ SIG Notes, Vol.94, No.94-HPC-52, pp.127, (1994).

[2] Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., and Sunderam, V., *"PVM 3.0 user's guide and reference manual"*, Technical Report ORNL/TM-12187, Oak Ridge National Laboratory, (1993).

[3] B. D. Cullity 著, 松村 源太郎 訳, "新版カリティ X 線回折要論", アグネ, (1980).

[4] 石畑 清, "アルゴリズムとデータ構造", 岩波書店, (1989).