Network Enabled Server の World-wide Grid における性能

田中良夫[†1] 建部修見[†1] 寺西慶太[†2]
二方克昌[†3,†4] 合田憲人[†3]
関口智嗣[†1] 原辰次[†3]

Network Enabled Server(NES) システムは Grid において簡単なプログラミングインタフェースを提供するグローバルコンピューティングシステムである．本稿では NES システムを用いて日本，米国間に渡る World-wide Grid で分散 / 並列計算を行なった際の性能を報告し，得られた知見から NES システムの設計 / 実装に関する指針を述べる．World-wide Grid では通信の方向によって通信性能に違いが生じたり，通信性能が安定しないなどの現象が見られ，大規模行列を引数としてとる数値計算のようにクライアントとサーバ間で大量のデータの送受信が必要となり，通信性能が全体の性能に対して大きな影響を与えるようなアプリケーションに対してはネットワークバンド幅の予約や通信性能を予測するシステムの利用が必要である事が分かった．また，数値最適化問題のようなタスク並列性のある，比較的細粒度な計算に関しても Network Enabled Server は有効なプログラミングインタフェースを与え，そのようなアプリケーションを考慮した機能を提供することにより，十分高い性能を期待できる事が分かった．

Performance of a Network Enabled Server on a Real World-wide Grid

Yoshio Tanaka,[†1] Osamu Tatebe,[†1] Keita Teranishi,[†2]
Yoshiaki Futakata,[†3,†4] Kento Aida,[†3] Satoshi Sekiguchi[†1]
and Shinji Hara[†3]

A Network Enabled Server (NES) is considered to be a good candidate as a viable Grid middleware, offering an easy-to-use programming model. In this paper, we report the performance of parallel and/or distributed applications utilizing a NES system on a World-wide Grid. We also describe the guideline of design and implementation of NES systems. On the World-wide Grid, communication performance is not stable and it depends on the direction of the communication. Network reservation and network performance forecast systems are required for NES especially for such applications as numerical computations which take large-scale matrix arguments and incur large data transfers between the client and the server, and the communication performance impacted on the overall performance. On the other hand, NES can be an effective programming model for task-parallel and relatively fine-grained applications such as numerical optimization problems.

1. はじめに

Network Enabled Server(NES) システムはクライアントからサーバに対してタスクの実行を依頼するいわゆる RPC 型のグローバルコンピューティングシステムである[1]．NES はグローバルコンピューティング環境 (**Grid**[2]) で関数，ライブラリを遠隔実行するための簡単なプログラミングインタフェースを提供し，アプリケーションプログラマは Grid でのプログラム開発や既存のプログラムの Grid への移植を容易に行なうことができる．また，NES システムは Nimrod[3] などのパラメータサーベイを行なうシステムや Program Solving Environment(PSE) などの下位レイヤとして使用することもでき，Globus[4] や Legion[5] などの，より低レベルな機構を提供するグローバルコンピューティングシステムとアプリケーションを橋渡するミドルウェアとして有望な形態である．

Grid は実際の応用計算に利用される段階に入ってい

†1 電子技術総合研究所 (現 産業技術総合研究所)
The Electrotechnical Laboratory (National Insitute of Advanced Industrial Science and Technology)
†2 ペンシルバニア州立大学
The Pennsylvania State University
†3 東京工業大学
Tokyo Institute of Technology
†4 現在日本アイ・ビー・エム株式会社勤務
IBM Japan, Ltd.

るが，Gridにおける標準的なプログラミングモデルはまだ確立されていない．今後ますますGrid環境の整備が進むと考えられ，Grid上で高い開発効率と実行時性能を提供するプログラミングモデルの確立は急務である．NESはGridでの標準的なプログラミングモデルのひとつとして有望であり，その確立に向けてプログラミングに関するガイドラインの作成，GridでのRPCとして必要な機能の洗い出しおよび性能調査による有効性の検証が必要である．

NESの性能に関しては，既存のNESシステムの性能がひとつの目安となる．NESはGridにおけるプログラミングモデルとして非常に注目されており，既にNinf[6]，NetSolve[7]，Punch[8]やCORBAを用いたGridシステム[9]などのいくつかのNESシステムが研究，開発されており，これらのシステムは研究段階から実用段階に入りつつある．しかし，報告されている性能はLANの中でのものやシミュレーションによるもの，あるいはGridの場合でもたかだか日本国内や米国内といった“domestic”な環境におけるものであり，日・米・欧をまたぐような実際の広域Grid(**World-wide Grid**)での性能報告はほとんどない．Gridはネットワーク性能等の点でLAN環境とは全く異るものであり，また，シミュレーションによる評価でも実環境をどの程度正確に反映することができているかといった点において疑問が残る．国際会議Supercomputing(SC) 2000でのNetchallenge[10]やSC 2001で予定されているSC Global[11]のように，近年World-wide Gridにおいてグローバルコンピューティングの試みが行なわれるようになっている．日米をまたぐようなWorld-wideな環境でのグローバルコンピューティングとしては，大阪大学の超高圧電子顕微鏡を米国から遠隔利用するデモンストレーションがSC'98で行なわれている．また，SC 2000のNetchallengeは日欧米間でのデータの転送性能に重点を置いたデモンストレーションである．しかし，高性能計算の実アプリケーションを用いたWorld-wide Gridの性能に関する報告はない．

我々はNESシステムを用いたプログラムをWorld-wide Gridで実行し，その性能を調査した．アプリケーションの実装方法や得られた性能などの今回得られた知見をもとに，NESの有効性を検証することができる．現在Global Grid Forum(GGF)[12]において世界中の研究者がグローバルコンピューティングシステムの標準化について議論を進めている．2001年1月の時点でGGFには10のワーキンググループ(WG)が存在し，プロトコル，API等の標準化に関して議論を行なっている．それらのWGのひとつであるAdvanced Programming Models(APM) WG[13]はGridにおけるプログラミングに関するWGであり，Gridにおけるプログラミングモデルの確立を目指している．我々はAPM WGの任務としてNESによるプログラミングに関するガイドラインの作成や有効性の検証を行なっており，本研究はその活動の一環となる．

我々は代表的な数値演算ライブラリであるScaLAPACK[14]と，数値最適化問題のひとつであるBMI固有値問題をNESシステムのひとつであるNinfを用いて実装し，SC 2000において，会場である米国，ダラスと日本の電総研および東工大との間で実行し，その性能を測定した．本稿ではその際得られたデータをまとめ，NESのWorld-wide Gridにおける性能を報告する．また，得られた知見を元に，NESおよび他のGridシステムの開発者に対して設計や実装技術に関するガイドラインを与える．次節では，実験環境および実験の概要を述べる．3節では今回実験に用いるNESシステムであるNinfの概要を述べる．4節と5節ではScaLAPACKおよびBMI固有値問題の概要，Ninfによる実装方法および実験結果を示し，最後に6節で実験結果のまとめとNESシステムの設計 / 実装に関する指針を述べる．

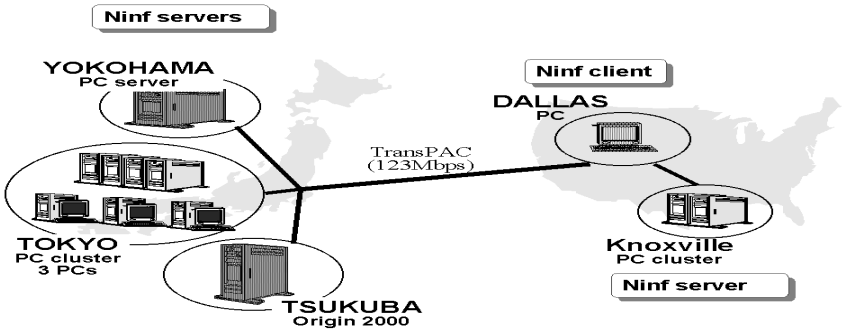
2. 実験環境と概要

我々は2000年11月に米国，テキサス州ダラスで開かれた高性能計算に関する国際会議SC 2000において研究成果の展示を行ない，その機会を利用して実験を行なった．本節では実験環境および実験の概要について述べる．

実験はSC2000の会場であるダラスにあるDallas Convention Centerと，日本のつくば市にある電総研，東工大の大岡山キャンパス(東京都)，長津田キャンパス(横浜市)および米国，テネシー州ノックスビルにあるテネシー大学の間で行なった．**図1**に実験環境を示す．ダラスの会場と電総研の間の経路情報として，tracerouteコマンドの出力を**図2**に示す．なお，双方向とも同じ経路を通っている事を確認している．経路の概要を**図3**に示す．また，ダラスの会場と東工大との経路は米国〜日本間は電総研との経路と同じであり，日本国内がAPANからWIDE経由となっている．

米国と日本の間はAPAN/TransPAC(123Mbps)によって接続されている．ただし，今回の実験はネットワークバンド幅の占有利用は行なわなかった．ダラス会場と電総研のftpコマンドによるファイル転送の際のスループットは1〜2Mbps程度であった．

今回はダラスの会場にPCを1台置き，このPC上で動くクライアントから電総研のOrigin 2000，東工大の



Client

system	CPU	location
PC	Pentium II (333MHz)	Dallas

Servers

system	CPU	location
Origin 2000	MIPS (195MHz, 16CPU)	ETL/Tsukuba
PC 1	Pentium III (500MHz)	TITECH/Tokyo
PC 2	Pentium III (500MHz)	TITECH/Tokyo
PC 3	Pentium III (450MHz)	TITECH/Tokyo
TITECH Cluster (4 nodes)	Pentium III (800MHz)	TITECH/Tokyo
PC Server	Pentium III Xeon (550MHz)	TITECH/Yokohama
UTK Cluster (2 nodes)	Dual Pentium III (550MHz)	UTK/Knoxville

図 1 実験環境
Fig. 1 Experimental testbed

```
traceroute to hpc.etl.go.jp (192.50.75.16), 30 hops max, 38 byte packets
 1  r792-rtr (140.221.159.254)  0.458 ms  0.357 ms  0.349 ms
 2  core-1-swrtr-2-b (140.221.130.29)  0.211 ms  0.144 ms  0.134 ms
 3  abilene-scinet (140.221.130.90)  5.469 ms  4.997 ms  5.033 ms
 4  kscy-hstn.abilene.ucaid.edu (198.32.8.62)  20.924 ms  20.903 ms  20.889 ms
 5  ipls-kscy.abilene.ucaid.edu (198.32.8.6)  57.080 ms  30.022 ms  29.991 ms
 6  st-abilene.startap.net (206.220.240.205)  34.727 ms  35.281 ms  34.608 ms
 7  tpr-atm3-0-7.jp.apan.net (203.181.248.242)  195.162 ms  195.006 ms  194.603 ms
 8  tppr-atm2-0-3.jp.apan.net (203.181.248.233)  200.412 ms  201.142 ms  200.277 ms
 9  im-tyx-02-fddi0-0.inoc.imnet.ad.jp (202.241.2.48)  195.728 ms  195.571 ms  195.725 ms
10  im-tyx-51-fa6-0-0.inoc.imnet.ad.jp (202.241.2.166)  179.817 ms  179.766 ms  180.045 ms
11  im-tbc-04-atm4-0-03.enoc.imnet.ad.jp (202.241.10.89)  199.775 ms  200.054 ms  200.300 ms
12  im-tbc-01-fa4-0.enoc.imnet.ad.jp (202.241.1.177)  199.754 ms  199.905 ms  199.480 ms
13  202.241.*.* (202.241.*.*)  200.812 ms  200.342 ms  201.781 ms
14  * * *
15  * * *
16  hpc.etl.go.jp (192.50.75.16)  200.747 ms *  200.728 ms
```

図 2 ダラスの会場から電総研への traceroute
Fig. 2 Traceroute from Dallas to ETL

両キャンパスにある PC および PC クラスタやテネシー大学の PC クラスタ上で動くサーバに対して計算要求を依頼するという形で実験を行なった．実験に際し，東工大の PC は占有状態であった． Origin 2000 とテネシー大学の PC クラスタは占有していないが実験中は他のユーザがほとんどいない状態であった．実験に用いたアプリケーションは代表的な数値計算ライブラリである ScaLAPACK と数値最適化問題のひとつである BMI 固有値問題であり，これらを NES システムのひとつである Ninf を用いて実装した．引き続き，Ninf の概要と，これらのアプリケーションの概要，Ninf による実装方法および実験結果について述べる．

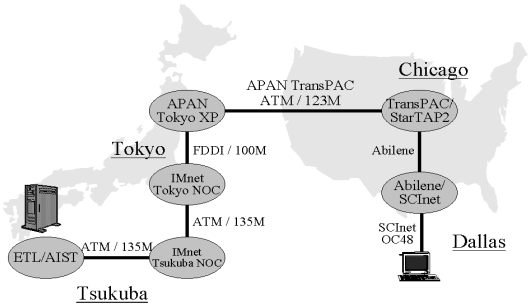


図 3 ダラスの会場と電総研との接続経路
Fig. 3 Network connection between Dallas Convention Center and ETL

3. Ninf の概要

Ninf は電総研で開発されている NES システムである．今回我々は Ninf を用いて World-wide Grid での NES の性能を測定した．本節では Ninf の概要を簡単に述べる．

Ninf はクライアント / サーバモデルに基づいており，クライアントからリモート計算機上のライブラリ (リモートライブラリ) を呼び出す仕組みを提供している．リモートライブラリはクライアントに通知するための引数情報を含む Ninf Executable と呼ばれる実行形式により実装される．クライアントがリモートライブラリを呼び出す場合，まずサーバに計算要求が送られる．計算要求を受け取ったサーバはライブラリ名から Ninf Executable を検索し，実行する．Ninf Executable のスタブルーチンはサーバやクライアントとの通信や引数のマーシャリング等を行なう．

既存のライブラリを Ninf サーバ上で利用可能なリモートライブラリとして作成する場合，(1) ライブラリ関数の呼び出し情報を Ninf IDL(Interface Description Language) と呼ばれる言語で記述する．(2) Ninf IDL コンパイラで IDL ファイルをコンパイルし，スタブルーチンを作成する．(3) スタブルーチンと関数本体をリンクし，Ninf Executable を作成する．(4) Ninf サーバを動かし，作成された Ninf Executable を登録する．といった手順を踏む．

また，Ninf は Ninf クライアント上でのリモートライブラリ呼び出し (Ninf 呼び出し) のための API として *Ninf_call()* を提供している．*Ninf_call()* は次のように用いる．

```
Ninf_call(name, args....);
```

Ninf_call() の第 1 引数には，リモートライブラリの名前を記述する．実際に呼び出されるサーバは，URL の記法に従って指定，環境変数による指定，スケジューラによる選択などの方法によって決定される．*Ninf_call()* はブロッキングであり，リモート計算機での計算が完全に終了し，計算結果を回収するまでリターンしない．要素ルーチンの実行と引数の送信を行ない，計算の終了を待たずにリターンする非同期呼び出し関数として *Ninf_call_async()* が提供されている．

Ninf クライアントは実行時に Ninf サーバから動的に引数情報を獲得するため，Ninf はクライアント作成時には IDL ファイルを記述する必要がないという特徴を持つ．これはクライアントがリモートライブラリを呼び出す際に，(1) クライアントからサーバに引数情報を要求する．(2) サーバからクライアントに引数情報を送る．(3) 受け取った情報に基づいて，クライアントからサーバに引数を送る．という手順を踏んでいるからである．このように Ninf では呼び出し開始時に引数情報のやりとりが行なわれるが，引数情報として送受信されるデータ量はわずかであり，引数のデータサイズが大きい場合にはこのプロトコルによるオーバーヘッドは無視できる．非均質な環境では Sun の XDR データフォーマットにデータを変換してクライアントとサーバはデータをやりとりするが，このようなデータ変換は非均質な環境では Ninf に限らず必要となる．他に特に特殊なプロトコルを利用している部分はなく，Ninf を用いて測定した実験結果は一般的な NES の性能として考えることができる．

4. ScaLAPACK

NES はリモートライブラリの呼び出しによって遠隔地にある高性能計算機を容易に利用する手段を提供しており，その対象アプリケーションのひとつに数値計算があげられる．我々は ScaLAPACK を用いて Grid での NES による数値計算の実行性能を測定した．本節では ScaLAPACK の概要および Ninf 化，Grid での性能について述べる．

4.1 ScaLAPAK の概要と Ninf 化

ScaLAPACK[14] は，連立一次方程式解法，固有値解法などの線形代数ライブラリ LAPACK を分散メモリ対応に拡張した並列数値ライブラリである．それぞれのアルゴリズムはメモリ階層間のデータ移動が少なくなるブロックアルゴリズムに基づいている．ScaLAPACK は基本線形代数通信ライブラリ BLACS を通信層として用い，また基本線形代数ライブラリ BLAS を BLACS を用い並列分散拡張した Parallel

BLAS (PBLAS) を利用し記述されている．ライブラリのソースは netlib[15] から入手することができる．ScaLAPACK は，密行列，帯行列解法だけではなく，反復固有値解法の PARPACK， ARPACK，疎行列直接解法の CAPSS， MFACT および疎行列反復解法の前処理の ParPre を含んだ形で構成されている．

ScaLAPACK では，行列は二次元プロセスグリッドの分散メモリにブロックサイクリック分散により配置される．この分散行列を表現するために配列デスクリプタを利用する．配列デスクリプタは密行列，帯，三角行列， out-of-core 密行列の三種類用意され，それらはデスクリプタのデータ型により区別される．密行列のデスクリプタは，プロセスグリッド，行列全体の大きさ，ブロックサイズ， $(0,0)$ 要素を持つプロセス，プロセッサローカルなリーディングディメンジョンからなる．ScaLAPACK のそれぞれのライブラリの引数は，基本的には LAPACK に基づいており，配列の引数に関してだけ，配列，配列デスクリプタ，部分行列の先頭位置と変更される．

ScaLAPACK を並列計算機上で Ninf Executable として実行させるためには， Ninf スタブ関数において，行列を分散させ，配列デスクリプタを作成し ScaLAPACK を呼び出す必要がある．そこで， Ninf クライアントから送られる引数から ScaLAPACK のライブラリを呼び出す ScaLAPACK wrapper を作成した．ScaLAPACK wrapper では rank 0 のプロセスで一度すべての引数を受取り，スカラーは全プロセスにブロードキャスト，配列はブロックサイクリック分散に再分散させ，配列デスクリプタを作成し， ScaLAPACK のそれぞれのライブラリを呼び出している．この ScaLAPACK wrapper により ScaLAPACK の Ninf IDL では配列分散を意識しない LAPACK と同様のインターフェースとすることができ，並列プログラミングをすることなく，並列ライブラリを利用することができる．図 4に対称正定値行列解法の `pdposv` と対称標準固有値解法の`pdsyev` の Ninf IDL を示す．`ninf_pdposv` などが ScaLAPACK wrapper となっている．現在の実装では rank 0 のプロセスで全配列データを受け取っているため，全配列サイズの上限が rank 0 のプロセスのメモリサイズとなってしまう．パイプライン的に rank 0 プロセスが配列を分散させることにより，このメモリ制限を回避することができるが，今回はこの部分は未実装である．

並列計算機の実行状況に応じて，使用プロセッサ，使用プロセス数を変化させることも可能であるが，今回の実装では使用プロセス数は一定としている．使用プロセッサは， PC クラスタの場合ロックファイルにより排他的に利用できるようになっている． Ninf 化された ScaLAPACK ライブラリを利用する場合，クライアントとサーバとの間で引数行列や計算結果の行列を送受信する必要があるため，通信性能が全体の性能に大きな影響を与えることが予想される．

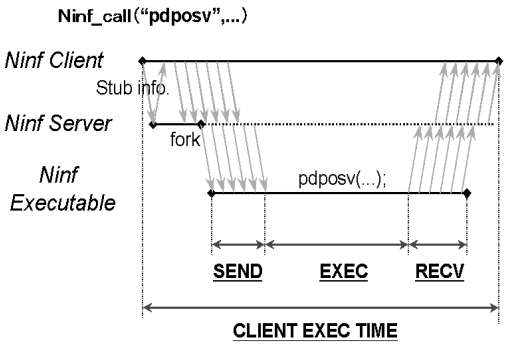


図 7 Ninf 呼び出しの時間測定のタイミング
Fig. 7 Ninf call performance measurement

4.2 実験および実験結果

我々は電総研の Origin 2000 およびテネシー大学の PC クラスタ上で ScaLAPACK ライブラリをリモートライブラリとして呼び出す Ninf Executable を作成し，Ninf サーバを稼働させた．ライブラリの実行に際しては， MPI を用いて Origin 2000 では 4 台， PC クラスタでは 2 台のプロセッサが利用されるようにサーバのコンフィグレーションを設定した．一方， Ninf 化された ScaLAPACK の各ライブラリを呼び出すクライアントを実装し，実験では引数行列のサイズを 200, 500, 700, 1000 の 4 通りに設定して各クライアントを実行した．実験は**表 1** に示す **LAN 環境**， **Domestic Grid**，および **World-wide Grid** の 3 通りのクライアント / サーバの組み合わせについて行なった．

コレスキー分解の`pdposv` と対称固有値解法の`pdsyev` における Ninf 呼び出しの性能を**図 5** から**図 9** に示す．図 5 と**図 6** は`pdposv` において引数行列のサイズをそれぞれ 500 および 1000 に設定して Ninf 呼び出しを行なったもの，**図 8** と図 9 は`pdsyev` において引数行列のサイズをそれぞれ 500 および 1000 に設定して Ninf 呼び出しを行なったものの結果である．いずれにおいても， LAN 環境および World-wide Grid では 24 回，Domestic Grid では 12 回の Ninf 呼び出しを行なった．

ここで，**クライアント実行時間**を「クライアントが *Ninf_call()* を呼び出してから，計算結果を回収し，*Ninf_call()* からリターンするまでの経過時間」と定義する． Ninf サーバはクライアントからの要求を受け

```
Define pdposv(IN char uplo, IN int n, IN int nrhs,
              INOUT double a[n][lda:n], IN int lda,
              INOUT double b[nrhs][ldb:n], IN int ldb,
              OUT int info[1])
"Linear Equation Solver for Symmetric Positive Definite Matrix."
Backend "BLACS"
Shrink "yes"
Calls "C" ninf_pdposv(uplo, n, nrhs, a, lda, b, ldb, info);

Define pdsyev(IN char jobz, IN char uplo, IN int n,
              INOUT double a[n][lda:n], IN int lda, OUT double w[n],
              OUT double z[n][lda:n], IN int ldz, OUT int info[1])
"Solver for Symmetric Standard Eigenvalue Problem."
Backend "BLACS"
Shrink "yes"
Calls "C" ninf_pdsyev(jobz, uplo, n, a, lda, w, z, ldz, info);
```

図 4 pdposv および pdsyev の Ninf IDL
Fig. 4 Ninf IDL for pdposv and pdsyev

表 1 クライアント / サーバの組み合わせ
Table 1 Client and servers used in experiments

	クライアント	サーバ
LAN 環境による実験	電総研の PC	電総研の Origin 2000
Domestic Grid による実験	ダラスの PC	テネシー大学の PC クラスタ
World-wide Grid による実験	ダラスの PC	電総研の Origin 2000

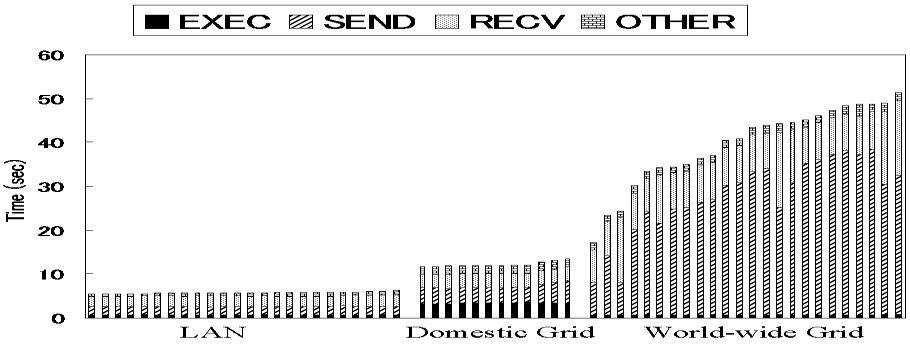


図 5 pdposv, 行列サイズは 500
Fig. 5 pdposv, Matrix size is 500

付けると子プロセスを生成し，生成されたプロセスが Ninf executable を実行する．クライアントと Ninf executable との間のデータ転送は Ninf サーバが中継する☆．グラフには**図 7** に示すように，クライアント実行時間 (CLIENT EXEC TIME) を (1) サーバ計算機上で実際にライブラリの実行にかかった時間 (EXEC), (2)Ninf executable がデータの受信を開始してから完了するまでの時間 (SEND), (3)Ninf executable がデータの送信を開始してから完了するまでの時間 (RECV), (4) その他の時間 (クライアント実行から EXEC, SEND, RECV の合計を引いた時間，OTHER), にブレークダウンして示している．いずれのグラフも LAN 環境での結果，Domestic Grid での結果，World-wide Grid での結果をすべてクライアント実行時間の短い順に並べて表示している．また，各ライブラリ，引数行列のサイズ，実行環境ごとの，クライアント実行時間の最小値，最大値，平均値，標準偏差を**表 2** に示す．

これらの結果より，LAN 環境では安定した性能が得られていることが分かる．また，Domestic Grid でも LAN 環境ほどではないものの，安定した性能が得られている．しかし World-wide Grid では性能の差が大きく，いずれの場合においても，最悪の場合は最良の場合の 2 倍〜3 倍程度の時間がかかってしまうという現象が

☆ 現在開発中の Ninf ではクライアントと Ninf executable が直接データを送受信する設計になっている

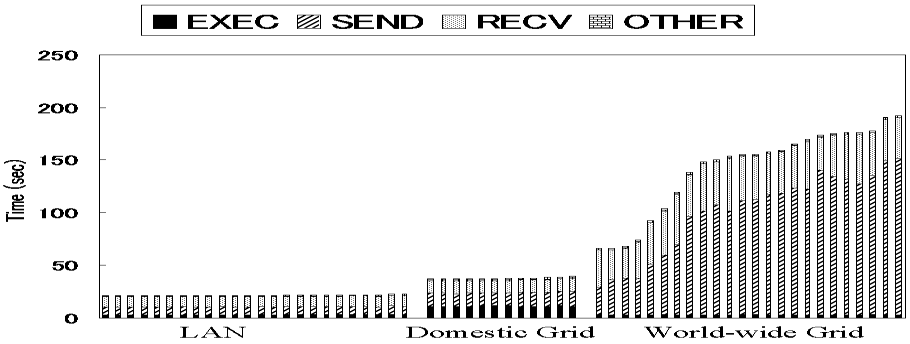


図 6 pdposv，行列サイズは 1000
Fig. 6 pdposv, Matrix size is 1000

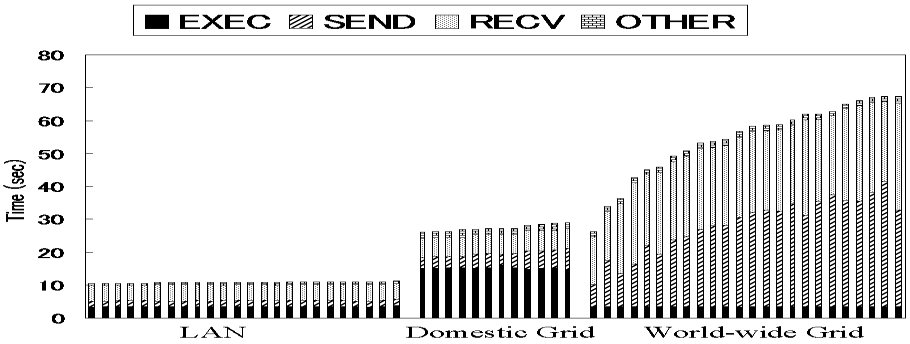


図 8 pdsyev，行列サイズは 500
Fig. 8 pdsyev, Matrix size is 500

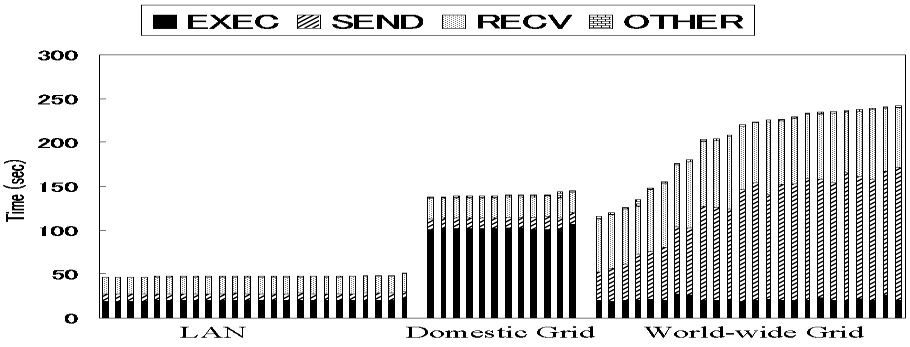


図 9 pdsyev，行列サイズは 1000
Fig. 9 pdsyev, Matrix size is 1000

見られる．標準偏差からも World-wide Grid が突出して性能のばらつきがあることが分かる．グラフより，この性能のばらつきはデータ転送の時間の振れに起因していることが読み取れる．

表 3 に各ライブラリ，引数行列のサイズについて，Ninf 呼び出しの際にクライアントとサーバとの間で転送されるデータ量 (KB 単位．少数点以下四捨五入) を，**表 4** に World-wide Grid におけるデータ転送の際のス

表 2 クライアント実行時間の最小値，最大値，平均値および標準偏差
Table 2 Minimum, maximum, average and standard deviation of client execution time

ライブラリ	引数サイズ	実験環境	最小値	最大値	平均値	標準偏差
pdposv	500	LAN 環境	5.46	6.16	5.73	0.19
		Domestic Grid	11.69	13.57	12.18	0.57
		World-wide Grid	17.20	51.32	39.48	8.88
	1000	LAN 環境	20.76	22.28	21.42	0.39
		Domestic Grid	36.76	39.13	37.60	0.73
		World-wide Grid	66.48	192.02	141.90	40.35
pdsyev	500	LAN 環境	10.48	11.23	10.82	0.20
		Domestic Grid	25.95	29.12	27.36	0.96
		World-wide Grid	26.37	67.27	54.31	10.97
	1000	LAN 環境	46.46	51.06	47.47	0.88
		Domestic Grid	138.32	144.71	140.09	2.01
		World-wide Grid	115.37	241.89	199.78	42.65

最小値，最大値，平均値の単位は秒

表 4 World-wide Grid におけるクライアント / サーバ間のスループットの最小値，最大値，平均値および標準偏差
Table 4 Minimum, maximum, average and standard deviation of bandwidth on World-wide Grid

ライブラリ	引数サイズ	方向	最小値	最大値	平均値	標準偏差
pdposv	500	上り	53.83	273.84	91.08	58.71
		下り	119.23	274.77	221.08	48.56
	1000	上り	53.77	302.76	107.65	70.42
		下り	161.99	281.47	203.51	30.18
pdsyev	500	上り	51.05	287.00	94.03	52.91
		下り	121.32	269.21	166.24	33.58
	1000	上り	51.58	245.47	93.85	54.79
		下り	188.82	287.58	220.13	22.54
全体		上り	19.58	303.47	93.89	56.78
		下り	53.10	287.58	187.36	45.76

最小値，最大値，平均値の単位は KB/ 秒

表 3 クライアント / サーバ間のデータ転送量
Table 3 Data size transferred between the client and the server

ライブラリ	引数サイズ	上り	下り
pdposv	500	2035	2035
	1000	7977	7977
pdsyev	500	1957	3918
	1000	7820	15648

単位は KB

ループットの最小値，最大値，平均値および標準偏差を示す．また，今回の実験で行なわれた World-wide Grid でのすべての Ninf 呼び出し[☆]におけるデータ転送のスループットも示す．ただし，ここでクライアントからサーバへの通信を**上り**，サーバからクライアントへの通信を**下り**と定義し，上りと下りそれぞれについてデータ転送のスループットを求めている．

表 4 において上りと下りのスループットを比較した場合，最大値はデータサイズに依存せずにほぼ同じ値が得られているが，最小値や平均値に関しては上りは下りの 1/2 から 1/3 程度の値となっている．つまり，最大性能としては上り下りともほぼ同じ性能を期待できるにもかかわらず，実際にはほとんどの場合において，上りは下りの半分以下の性能しか得られない事が分かる．このことは図 5 から図 9 において，上りと下りでデータサイズが同じである pdposv で上りが下りの 2 倍近く時間がかかっていることや，上りが下りの約 1/2 の大きさのデータしか送らない pdsyev で上りと下りがほぼ同じ時間かかっていることからも読み取れる．また，標準偏差より World-wide Grid ではスループットの振れ (特に上りの性能の振れ) が大きく，このことが全体の性能の振れにつながっていることが分かる．

APAN の Network Operation Center が提供している TransPAC のトラフィック情報[☆☆]によると，TransPAC では実験を行なった 2000 年 11 月を含め，年間を通して上りは下りに比べて平均的に数倍程度のトラフィックがあることや，下りのトラフィックが比較的

☆ 全ライブラリ，全引数サイズの組み合わせによる Ninf 呼び出し．全部で 1508 回の Ninf 呼び出しがあった．

☆☆ http://www.jp.apan.net/NOC/tm/us/us.html

安定しているのに対し，上りのトラフィックは振れが大きい事が分かる．このトラフィックの違いが実験で上りのスループットが低いことや，標準偏差が大きいことの要因として考えられる．

5. BMI 固有値問題

NES は関数呼び出しレベルでの Grid の利用を可能としており，探索問題のようなタスク並列な並列性を持つアプリケーションへの利用が期待される．我々は BMI 固有値問題を用いて Grid でのタスク並列なアプリケーションの実行性能を測定した．本節では BMI 固有値問題の概要および Ninf 化， Grid での性能について述べる．

5.1 BMI 固有値問題の概要と Ninf 化

BMI(Bilinear Matrix Inequality) 固有値問題は，制御系設計の分野で扱われる数値最適化問題の一つであり，対称行列 $F_{ij} = F_{ij}^T \in \mathcal{R}^{m\times m}(i = 0,\cdots,n_x, j = 0,\cdots,n_y)$ が与えられる時， (1) 式に定義されるベクトル変数 x， y に関する双線形行列関数 $F : \mathcal{R}^{n_x} \times \mathcal{R}^{n_y} \to \mathcal{R}^{m\times m}$ の最大固有値を最小化する x， y を求めることを目的とする．

$$F(x,y) := F_{00} + \sum_{i=1}^{n_x} x_i F_{i0} + \sum_{j=1}^{n_y} y_i F_{0j} + \sum_{i=1}^{n_x}\sum_{j=1}^{n_y} x_i y_i F_{ij} \qquad (1)$$

本問題は NP 困難であることが知られており[16]，分枝限定法による解法が提案されている．我々は，藤岡らにより提案されている分枝限定法による BMI 固有値問題の解法[17] を Ninf を用いて実装した． Ninf による BMI 固有値問題の計算では，複数の Ninf サーバ上で並列計算を行う場合と単一の Ninf サーバ上で計算を行う場合の 2 種類の実行方式を用いた．

複数サーバ上での計算では，子問題の計算，具体的には解の下界値と上界値の計算を行う Ninf executable を開発し，各々の子問題を別々の Ninf サーバ上で並列に実行する．即ち， Ninf サーバ上では子問題の計算を行い， Ninf クライアント上では分枝操作，子問題の計算を行う Ninf executable の呼び出し (Ninf 呼び出し)，および限定操作を行う．**図 10** に複数サーバ上で計算を行う場合の Ninf クライアント上での処理手順を示す．

これに対して単一サーバ上での計算では全ての処理，即ち分枝操作，子問題の計算，限定操作を行う Ninf executable を開発し， Ninf クライアント上では 1 回の Ninf 呼び出しのみを行う． Ninf 化された BMI ライブラリを利用する場合，クライアントとサーバ間でのデータ転送量はさほど大きくない．このようなアプリケーションの場合，通信性能が全体の性能にどのような影響を与えるか，また，実際の広域環境でどのような性能で動作するかを調査する．

```
while (上界値 - 下界値 < 許容誤差) {
  分枝ノードの決定;
  サーバ数分の子問題の生成 (分枝操作);
  for (i=0; i<サーバ数; i++) {
    Ninf executable(子問題) の呼出;
  }
  下界値，上界値の更新;
  限定操作;
}
```

図 10 BMI 問題の解法における Ninf クライアントの処理手順
Fig. 10 Ninf client procedure for BMI problem

5.2 実験および実験結果

Ninf サーバには，東工大大岡山キャンパスおよび長津田キャンパスに設置された 8 台の PC を用いる (図 1 における PC 1, PC 2, PC3, TITECH Cluster および PC Server)．複数サーバ上での計算ではこれらすべてのサーバを利用し，単一サーバ上での実行では長津田キャンパスにある PC Server のみを用いた．

本実験では，ベクトル変数 x， y，および行列 $F_{i,j}$ の大きさが異なる 3 種類の問題を計算した．**表 5** にこれらの問題における x， y の次元および $F_{i,j}$ 行列サイズを示す．表 5 中の値および行列 $F_{i,j}$ の初期値は，制御系設計における実問題に則して決定されている．

表 5 問題サイズ
Table 5 Problem size

	x の次元	y の次元	行列のサイズ
問題 1	4	5	12
問題 2	2	6	24
問題 3	2	3	60

表 6 に，複数サーバ上で各問題を実行した場合の Ninf クライアント上での Ninf 呼び出しの回数と 1 回の Ninf 呼び出し当たりのサーバ上での平均計算時間 (計算時間 /Ninf 呼び出し) および平均通信時間 (通信時間 /Ninf 呼び出し) を示す．ただし，ここでの通信時間は上りと下りの通信時間の和である．藤岡らによる分枝限定法による BMI 固有値問題解法では，子問題の計算量および入力データサイズが $F_{x,y}$ の行列サイズに依存する．そのため，表 6 に示すように， $F_{x,y}$ の行列サイズの最も小さい問題 1 では， 1 回の Ninf 呼び出し当たりの計算時間および通信時間が他の問題よりも小さいことがわかる．一方，解の探索範囲，即ち生成される子問題数は x， y の次元数に依存するため，表 6 に示すように，問題 1 では Ninf 呼び出しの回数が他に比べて大きいこと

がわかる.

表 6 複数サーバ上での計算における Ninf 呼び出し当たりの計算・通信時間
Table 6 Computation and communication time per Ninf_call on multiple servers

	Ninf 呼び出し	計算時間 / Ninf 呼び出し	通信時間 / Ninf 呼び出し
問題 1	8673	0.14	0.40
問題 2	1529	0.40	0.97
問題 3	209	2.87	2.74

時間の単位は秒

次に，**表 7** に複数サーバおよび単一サーバ上での Ninf による BMI 固有値問題の実行時間 (経過時間) を示す. 表中，通信時間は，実行時間中の通信時間を示す. 表 7 より，問題 2 および問題 3 においては，複数サーバ上での実行時間が単一サーバ上での実行時間に比べて短縮されており， Grid での並列処理による実行時間短縮が確認できる. また，問題 1 に関しては複数サーバ上での実行時間が単一サーバの場合よりも増大していることがわかる.

表 6 に示すように，問題 1 では 1 回の Ninf 呼び出しあたりの Ninf サーバ上での計算時間が 0.14 秒と非常に短い上に， Ninf 呼び出しの回数が他の問題よりも多い. このため，問題 1 では Ninf サーバ上での計算時間に対する Ninf クライアント上で Ninf 呼び出しの起動オーバーヘッドが大きく，複数サーバ上での計算では性能低下が著しかったといえる.

表 7 Ninf による BMI 固有値問題の実行時間
Table 7 Execution time of BMI eigenvalue problem using Ninf

	複数サーバ		単一サーバ	
	実行時間	通信時間	実行時間	通信時間
問題 1	3581	441	2365	0.42
問題 2	834	186	1636	0.83
問題 3	247	74	708	1.50

単位は秒

6. 結果のまとめ

NES はサービス提供者がサーバを利用可能にしていれば，クライアント側ではローカルな計算機で実行される通常の関数呼び出しをリモート呼び出しの関数に置き換えるだけで遠隔地にある高性能計算機を利用するという手段を提供している. そのプログラミングの容易さは NES の特徴のひとつであるが，実行時にはクライアントからサーバへの引数の転送と，サーバからクライアントへの結果の通知が必要となる. ScaLAPACK を用いた実験より，クライアントとサーバ間でのデータ転送量が多いアプリケーションの場合， World-wide Grid では LAN 環境や Domestic Grid に比べて低くかつ不安定な通信性能の影響を大きく受けることが明らかになった. 今回実験を行なった World-wide Grid では TransPAC の上りのトラフィックが下りトラフィックの数倍程度あることや，その変動が大きい事が原因となっているが，TransPAC に限らず広域ネットワークの場合はこのような現象が起きやすい. 今後 World-wide Grid においても通信性能はどんどん高くなる事が予想されるが，LAN に対する相対的な通信性能や Grid 特有の通信性能の振れを考慮したシステムの設計，実装が必要である.

また，タスク並列なアプリケーションである BMI 固有値問題の分枝限定法による解法プログラムを用いて実験を行なった. 我々の実装方法では計算の粒度は引数として与えられる行列のサイズに依存するが. 今回与えたデータのように単一タスクの計算時間が数秒程度の比較的細粒度のタスク並列性を抽出する問題においても，複数の Ninf サーバによる Grid 計算によりプログラムの実行時間を短縮できることが確認された. しかしながら現状のアルゴリズムでは，対象とする問題の $F_{i,j}$ の行列サイズが小さい場合 Ninf クライアント上での Ninf 呼び出し起動オーバーヘッドが Ninf サーバ上での計算時間に対して相対的に大きくなり，複数サーバ上での性能が低下することも確認された. これは 1 回の Ninf 呼び出しにおいてサーバ上で計算される子問題数を適切に決定するアルゴリズムを開発することにより改善することができるが， NES システムとしてはこのような細粒度の呼び出しを考慮した機能を提供する必要がある. 例えば Ninf の場合は *Ninf_call()* が実行されて Ninf 呼び出しが行なわれるとサーバとの接続が確立され，サーバ上での計算が終わって結果がクライアントに返されるとその接続は切断されてしまう. プロセスが利用できるファイルデスクリプタ数の制限などを考えると，リモート呼び出しのたびに接続，切断を行なう実装が頑健で安全であるが，細粒度なタスク並列性のあるアプリケーションに高い性能を提供するためにはクライアントとサーバとの接続を切らずに何度も利用する方が望ましい. 今回の実験により， NES が比較的細粒度のタスク並列性を持つ問題に対しても有効なプログラミングインタフェースを提供することが分かった. このようなアプリケーションに対して高い性能を提供するような実装， API の提供を行なうことにより， World-wide Grid における実行性能の点においても十分期待でき，今後 Grid でのタスク並列の並列性を持つプログラムに対する標準的なプログラミングモデルとして NES が有望であることが示さ

れた.

現在 NES は，当初想定されていた「遠隔地の高性能計算機上でライブラリを実行する」という枠にとどまらず，広域に分散配置された多数のプロセッサ資源上でのタスク並列性を持つ比較的細粒度な問題の実行や，広域に分散配置されたペタバイト級の大規模データに対するデータ処理など様々な領域の問題に利用され始めている．今回得られた知見より，これらの幅広い利用に対して Grid において高い性能を提供するために，NES システムの設計 / 実装に関して次のような指針をまとめる．

- 今回の実験のように利用するサーバが決まっている場合は，あらかじめバンド幅の予約を行なうなどして高いスループットを確保する手段を提供する．
- スケジューラが複数のサーバの中から実行するサーバを動的に選択する場合には，Network Weather Service[18] などのネットワーク性能予測システムから得られる情報を利用して通信の方向を考慮した上でデータの転送時間を予測し，サーバ選択の戦略に利用する．
- 実行時 (リモート呼び出し時) ではなく，事前にデータをサーバ側に転送する仕組みを提供する．これにより実行時のデータ転送時間が削減され，実行効率が大幅に改善される．
- 広域に分散配置された大規模なデータ処理に対応すべく，「データを保持するサーバ」を探し出してそのサーバに対してリモート呼び出しを行なう機能を提供する．
- 細粒度なタスク並列性のあるアプリケーションに対する Grid における標準的なプログラミングモデルとして，クライアントとサーバとの通信路を切らずに再利用する実装や API の提供を行なう．

7. おわりに

実際の World-wide Grid における実験データは貴重である．今回得られたデータはおおむね予想できる範囲のものではあったが，今回得られたデータに基づいてシステムを適切に設計，実装することにより，World-wide Grid においても十分高い性能を期待できる事が分かった．我々は「Grid におけるリモートライブラリ呼び出しによるプログラミング」の標準化を行なおうとしている．今回得られた知見を元に，アプリケーションプログラマに対する情報 (プログラミングの方法等) や Grid の研究者に対する情報 (性能，実装技術およびそこから得られる標準化案) をまとめる予定である．

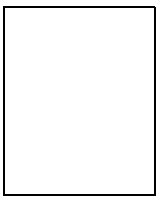
謝辞 本研究に際し，計算機の利用を快諾して頂いたテネシー大学の Jack Dongarra 教授に感謝いたします．また，様々なご助言を頂いた東京工業大学の松岡聡助教授，産業技術総合研究所の中田秀基博士をはじめとする Ninf チームの皆様と，研究をサポートして頂いた産業技術総合研究所大蒔和人情報処理研究部門長に感謝いたします．

参考文献

1) Matsuoka, S., Nakada, H., Sato, M. and Sekiguchi, S.: Design issues of Network Enabled Server Systems for the Grid. http://www.eece.unm.edu/~dbader/grid/.
2) Foster, I. and Kesselman, C.(eds.): *The GRID: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann (1999).
3) Abramson, D., Giddy, J. and Kotler, L.: High Performance Modeling with Nimrod/G: Killer Application for the Global Grid?, *International Parallel and Distributed Processing Symposium*, pp. 520–528 (2000).
4) Foster, I. and Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit, *Supercomputer Applications*, Vol. 11, No. 2, pp. 115–128 (1997).
5) Grimshaw, A. S., Wulf, W. A., French, J. C., Weaver, A. C. and Jr., P. F. R.: Legion: The Next Logical Step Toward a Nationwide Virtual Computer, Technical Report CS-94-21, UVa CS (1994).
6) Nakada, H., Sato, M. and Sekiguchi, S.: Design and implementations of Ninf: towards a global computing infrastructure, *Future Generation Computer Systems*, Vol. 15, pp. 649–658 (1999).
7) Casanova, H. and Dongarra, J.: NetSolve: A Network Server for Solving Computational Science Problems, *Supercomputer Applications and High Performance Computing*, Vol. 11, No. 3, pp. 212–223 (1997).
8) Kapadia, N. H., Fortes, J. A. B. and Brodley, C. E.: Predictive application-performance modeling in a computational grid environment, *Eighth IEEE International Symposium on High Performance Distributed Computing*, pp. 47–54 (1999).
9) René, C. and Priol, T.: MPI Code Encapsulating using Parallel CORBA Object, *Eighth IEEE International Symposium on High Performance Distributed Computing*, pp. 3–10 (1999).
10) : . http://www.mcs.anl.gov/sc2000_netchallenge/.
11) : . http://www.mcs.anl.gov/scglobal/.
12) : . http://www.gridforum.org/.
13) : . http://www.eece.unm.edu/~dbader/grid/.

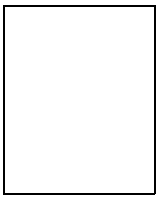
14) Blackford, L. S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J., Hammarling, S., Henry, G., Petitet, A., Stanley, K. and Walker, D.: *ScaLAPACK User's Guide*, SIAM (1997).
15) : . http://www.netlib.org/ http://phase.hpcc.gr.jp/mirrors/netlib/.
16) Toker, O. and Özbay, H.: On the NP-hardness of Solving Bilinear Matrix Inequalities and Simultaneous Stabilization with Static Output Feedback, *ACC*, pp. 2525–2526 (1995).
17) Fujioka, H. and Hoshijima, H.: Bounds for the BMI Eigenvalue Problem, 計測自動制御学会論文集, Vol. 33, No. 7, pp. 616–621 (1996).
18) Wolski, R., Spring, N. T. and Hayes, J.: The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing, Technical Report TR-CS98-599, UCSD (1998).

(平成年月日受付)
(平成年月日採録)



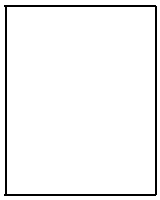
田中　良夫（正会員）

昭和40年生．平成7年慶応義塾大学大学院理工学研究科後期博士課程単位取得退学．平成8年技術研究組合新情報処理開発機構入所．平成12年通産省電子技術総合研究所入所．平成13年4月より独立行政法人産業技術総合研究所．現在に至る．博士(工学)．広域分散計算，グリッド技術，コンピューティングポータルに関する研究に従事．IC'99論文賞．ACM会員．



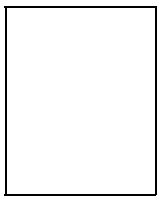
建部　修見（正会員）

昭和44年生．平成4年東京大学理学部情報科学科卒業．平成9年同大学大学院理学系研究科情報科学専攻博士課程修了．同年電子技術総合研究所入所．平成13年4月より独立行政法人産業技術総合研究所．並列数値アルゴリズム，並列計算機システム，広域分散計算の研究に従事．ハイパフォーマンスコンピューティングに興味を持つ．理学博士．日本応用数理学会，ACM各会員．



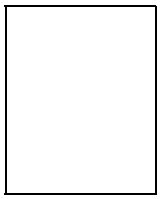
寺西　慶太（正会員）

てらにしけいた



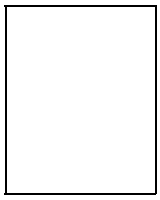
二方　克昌（正会員）

ふたかたよしあき



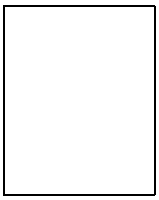
合田　憲人（正会員）

昭和42年生．平成2年早稲田大学理工学部電気工学科卒業．平成4年同大学大学院修士課程修了．平成8年同大学院博士課程退学．平成4年早稲田大学情報科学研究教育センター助手，平成8年同特別研究員．平成9年東京工業大学大学院情報理工学研究科助手，平成11年同大学院総合理工学研究科専任講師，現在に至る．博士（工学）．並列・分散計算方式，並列化コンパイラ，スケジューリング技術，グローバルコンピューティングの研究に従事．電子情報通信学会，電気学会，ACM，IEEE-CS各会員．



関口　智嗣（正会員）

昭和34年生．昭和57年東京大学理学部情報科学科卒業．昭和59年筑波大学大学院理工学研究科修了．同年電子技術総合研究所入所．平成13年4月より独立行政法人産業技術総合研究所．以来，データ駆動型スーパーコンピュータSIGMA-1の開発等の研究に従事．並列数値アルゴリズム，計算機性能評価技術，ネットワークコンピューティングに興味を持つ．市村賞受賞．情報処理学会，日本応用数理学会，ソフトウェア科学会，SIAM，IEEE各会員．



原　辰次（正会員）

1976 年 3 月東京工業大学大学院理工学研究科制御工学専攻修士課程修了．同年 4 月日本電信電話公社入社．1980 年 4 月長岡技術科学大学助手．1984 年 4 月東京工業大学工学部助教授．1990 年 9 月同学総合理工学研究科教授となり現在に至る．ロバスト制御，ディジタル制御などシステム制御理論，制御系設計支援環境の研究に従事．IEEE，計測自動制御学会，シミュレーション学会などの会員．