# HMCS-G：グリッド環境における計算宇宙物理のためのハイブリッド計算システム

朴 泰祐[†1] 佐藤三久[†1] 小沼賢治[†2]
牧野淳一郎[†3] 須佐元[†4]
高橋大介[†1] 梅村雅之[†5]

我々は，グリッド環境上に分散した汎用並列計算機群と専用並列計算機を結合し，各種物理現象の複合シミュレーションを可能とする計算環境 HMCS-G を設計し，重力専用計算機 GRAPE-6 クラスタを中心とする計算宇宙物理学用のシステムを実装した．アプリケーションプログラミングの簡便さとシステム全体のスループット向上のため，グリッド RPC として OmniRPC を用い，HMCS-G の特性に合わせた効率的なシステムを実現した．各種ネットワーク環境における初期性能評価の結果，ある程度のネットワークバンド幅のもとで，HMCS-G は実アプリケーションの現実的な実行に耐える性能を提供できることが確認された．また，HMCS-G により，貴重な専用計算リソースである GRAPE-6 のようなシステムを，世界規模で共有することが可能となる．

# HMCS-G: Grid-enabled Hybrid Computing System for Computational Astrophysics

Taisuke Boku,[†1] Mitsuhisa Sato,[†1] Kenji Onuma,[†2]
Junichiro Makino,[†3] Hajime Susa,[†4] Daisuke Takahashi[†1]
and Masayuki Umemura[†5]

We have developed a hybrid computing system named HMCS-G which combines general purpose parallel systems and a special purpose machine for gravity calculation on computational grid environment. The prototype is implemented with GRAPE-6 gravity engine and OmniRPC as Grid-RPC with high portability and throughput. Through the preliminary performance evaluation on several network conditions, it is confirmed that the function and capability of HMCS-G can support actual application for multiphysics simulation. With HMCS-G, the utilization ratio of special purpose machines such as GRAPE-6 is also greatly enhanced through access from all over the world.

## 1. はじめに

計算宇宙物理学において，重力計算は最も基本的な計算であり，$N$ 個の粒子に対して $O(N^2)$ の計算量を必要とするため，大規模並列計算機やクラスタ等を用いたとしても，大量の $N$ に対する十分な計算速度を得ることは難しい．また，その演算は本質的に $N$ 対 $N$ の大量の通信を含むため，グリッド環境でこれを解くこともまた難しい．この問題を高速に解くために，いくつかの専用計算機が提案・開発されている．GRAPE（Gravity Pipeline）[7]はその代表的なものであり，10 年以上にわたって合計 6 世代の重力計算専用システムを作り続けてきた．GRAPE-6 はその最新版であり，1 ボードで 1 TFLOPS の重力計算性能を提供する．現時点で，最大の GRAPE-6 システムは東京大学にある 64 ボードのシステムで，64 TFLOPS のピーク性能を持ち，実アプリケーションでの実効性能は約 30 TFLOPS である[7]．一方，筑波大学計算物理学研究センターにも同様のボードを 8 枚用いた GRAPE-6

†1 筑波大学電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba
†2 筑波大学大学院システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba
†3 東京大学理学系研究科天文学専攻
Department of Astronomy, Graduate School of Science, The University of Tokyo
†4 立教大学理学部物理学科
Department of Physics, Faculty of Science, Rikkyo University
†5 筑波大学物理学系
Institute of Physics, University of Tsukuba

クラスタが実装され，利用されている．

銀河形成は宇宙物理学において最も重要な問題の 1 つである．これまで，この計算は重力のみを粒子間相互作用としてシミュレーションされてきた．しかし，この分野における最新の研究により，銀河形成には自己重力だけでなく，粒子からなるガス体の流体計算[4]，そして外部光源からの輻射の影響が無視できないことが分かってきた[5]．銀河の源はガスであり，個々の源はある範囲に広がった質量を持つ仮想粒子としてモデル化することができる．そして，それらの相互作用を連続した流体として処理すると同時に，仮想粒子間の自己重力作用も計算する必要がある．このようなシミュレーションを実現する最も効果的な手法の 1 つとして SPH（Smoothed Particle Hydrodynamics）がある．上述したように，もう 1 つの重要な要素は RT（Radiative Transfer：輻射）である．外部光源がガスに与える影響は，特に初期の銀河形成では大きい．よって，従来より詳細な銀河形成シミュレーションのためには，これらの作用すべてを含める必要がある[2]．

これまで GRAPE システムは，重力計算だけを超高速処理することを目的として設計されてきたため，GRAPE 単体ではこのような複雑な系をシミュレーションすることは不可能である．このため，我々は HMCS（Heterogeneous Multi-Computer System）と呼ばれる，ハイブリッド型並列計算機システムを開発してきた[1),2)]．HMCS は，RT 効果をともなう SPH 計算（以後，RT-SPH と呼ぶ）を処理する汎用計算機部（汎用 MPP またはクラスタ）と，重力計算を行う GRAPE-6 部からなり，それらを並列ネットワークで結合している．HMCS は 1 つのローカルサイトに汎用計算機部と GRAPE-6 システムの両者が導入されている限り，RT-SPH のような複雑な計算宇宙物理シミュレーションを実現するための理想的なシステムである．しかし，GRAPE-6 は TFLOPS 級の演算能力を通常の MPP やクラスタに比べはるかに低コストで実現できる専用計算機ではあるが，その特殊性ゆえこれを導入・維持できるサイトは限られている．さらに，数百万粒子規模の計算を行うためには，超高性能を持つ MPP や大規模クラスタを汎用計算機部として設けなければならない．たとえば，100 GFLOPS のピーク性能を持つクラスタを汎用計算機部として用いた場合，GRAPE-6 による重力計算の処理時間は，RT-SPH のそれの数十分の一で終わってしまう．

したがって，HMCS における汎用計算機部を多数のサイトにまたがるクラスタ（あるいは MPP またはベクトル機）に分散させ，それらの間で GRAPE-6 クラスタを共有するという使い方は，きわめて自然な発想である．ここでは GRAPE-6 クラスタは「重力計算サーバ」として位置づけられ，分散された汎用計算機部となるクラスタ類はクライアントという立場になる．我々はこのようなシステムを HMCS-G（‘G’ は Gravity と Grid の両者の意味）と呼ぶ．本論文では，HMCS-G のコンセプト・設計・実装，そして各サイトからのラウンドトリップ時間を基にした性能評価を行う．

## 2. HMCS

HMCS-G の設計説明に入る前に，オリジナルの HMCS について概説する[1),2)]．我々の HMCS プロトタイプは，2 種類の HPC プラットフォームから構成されている．汎用計算機部には 2,048 台の PU（Processing Unit）を持つ CP-PACS[3)]を用い，専用計算機部は 256 個の GRAPE-6 チップからなる GRAPE-6 クラスタ[7)]である．CP-PACS は 2,048 台の PU に加え，128 台の IOU（I/O Unit）を備えており，各々の IOU は独立した RAID-5 ディスクシステムを持ち，高速な並列入出力機能を実現している．GRAPE-6 の 1 ボードは 32 個の専用計算 ASIC を搭載し，N 体重力問題を超高速処理する．これらのボードを複数用いたクラスタは，そのボード枚数にほぼ比例する性能を提供する．2,048 PU の CP-PACS のピーク性能は 614.4 GFLOPS，8 ボードからなる GRAPE-6 クラスタのピーク性能は 8 TFLOPS である．

**図 1** に HMCS プロトタイプのブロック図を示す．8 枚の GRAPE-6 ボードのホストコンピュータとして，8 台の Linux PC（Pentium4）を用いている．CP-PACS の 16 台の IOU に設置された 100 base-TX イーサネット NIC から，16 本のリンクが 2 系統に分割され，2

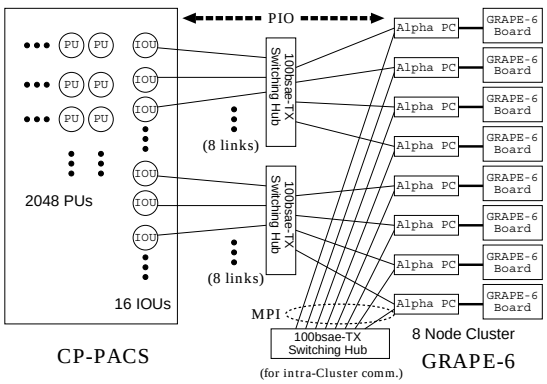



**図 1** 既存の HMCS のブロックダイアグラム
Fig. 1 Diagram of original HMCS.

台のスイッチングハブを通して 8 台の GRAPE-6 ホスト PC に接続されている．したがって，ネットワーク上の総リンク数は 24 本となっている．これらすべてを 1 つのスイッチに接続することは可能ではあるが，我々は HMCS の並列ネットワーク接続を，よりスケーラブルなものにする実験を含め，このような多系統接続形態をあえてとっている．

## 3. HMCS-G のコンセプトと設計

グリッドの大規模科学技術計算への利用ということについて，2 つの側面があると考えられる．1 つは，絶対的な総計算量を必要とする多数の問題に対し，リモートな資源の共有，簡便な利用方法を提供することによりその要求に応えるという貢献，もう 1 つは，GRAPE のような特殊システムを，実装されているサイト間の距離を意識することなしに，簡単に利用することを可能にすることにより，計算の質的な向上に貢献するということである．HMCS-G が目指すものは，後者にあたる．このコンセプトは，HMCS-G の現在のモデルである重力専用計算を含む複合計算に限らず，各種専用計算機に適用できるものである．

HMCS-G の基本的な発想は，計算宇宙物理学における，RT-SPH のような汎用計算部と重力専用計算部からなる大規模計算のうち前者に対応する処理を，複数のサイトに分散設置されている MPP またはクラスタに分散させ，専用計算機（GRAPE-6）をそれらの計算機からの重力計算リクエストを処理するサーバと位置づけることである．これを言い換えれば，GRAPE-6 という貴重な資源を，日本中あるいは世界中のクライアント計算機から簡単に利用できるようにすることにより，GRAPE-6 の有効利用を図るということになる．

GRAPE-6 だけを単体で利用し，重力相互作用のみに基づくシミュレーションを行っている場合は，その性能を十分に利用することが可能であり，多数のボードを追加することにより，全体の性能を向上できる．しかし，RT-SPH に代表される複合型計算を行おうとした時点で，多くの場合，その相対性能の差が大きいあまり，GRAPE-6 以外の汎用計算部分がボトルネックとなり GRAPE-6 の利用効率は低下してしまう．このため，同一サイトにある複数の汎用並列機（MPP またはクラスタ）から GRAPE-6 クラスタを多重利用することも当然考えられるが，それでもまだ性能比は埋まらないというのが現状である．したがって，より広範囲な分散環境における GRAPE-6 の共有という発想が生まれる．この場合，GRAPE-6 の遠隔アクセスということによって生じる通信遅延の問題は無視できない．しかし，汎用計算機部にはある程度の通信遅延隠蔽を行うアルゴリズムを実装することも可能であり[2]，そのことが致命的なボトルネックとはならない．むしろ，この通信遅延によって GRAPE-6 クラスタの利用率が落ちないようにすることが重要である．

以上のコンセプトに基づき，HMCS-G の設計を行う．GRAPE-6 は，各種の重力計算をサポートできるよう設計されている[6]．その基本的な能力は，3 次元空間上の $N$ 個の粒子（各々は独立な質量を持つ）に対し，$N$ 対 $N$ の完全相互作用を，定められた重力計算式に従ってパイプライン処理し，各粒子の加速度を出力するというものである．したがって，GRAPE-6 に対する入出力は，以下のように簡単なものとなる．

- 入力：総粒子数，および時間ステップと空間サイズのオーダ
- 入力：全粒子の質量・3 次元座標および前の計算ステップのポテンシャル等
- 出力：全粒子の 3 次元加速度とポテンシャル

各時間ステップにおいて，これらの入出力データがクライアントと GRAPE-6 の間で転送される．したがって，GRAPE-6 での計算を含むこれらの機能をグリッド環境に対応した RPC（Remote Procedure Call）として実装することは，きわめて自然である．

このグリッド上の RPC を用いた実装について，考慮しなければならない点がいくつかある．

**(1) マルチクライアント化**

既存の HMCS では，GRAPE-6 のサーバプロセスは汎用計算機部のプロセスと一対一の対応だけを想定しており，複数のクライアントを多重処理するようになっていない．従来のサーバプロセスをそのままにして，GRAPE-6 の複数のボードをボード単位で切り分け，空間的に多重化することも考えられるが，クライアントが要求する計算粒子数は一定ではないため，そのまま負荷をバランスさせるのは無理である．したがって，GRAPE-6 クラスタ全体で各々のクライアントの処理を並列化させ，さらにこれを時分割処理する形にサーバプロセスを設計し直す必要がある．

**(2) 通信遅延隠蔽**

GRAPE-6 の各ボードは，そのホストコンピュータ（Linux PC）の PCI バスを通じて接続された，特殊なハードウェアであり，ホストコンピュータ上のサーバプログラムは，GRAPE-6 ボードに対するすべてのアクセスを制御しなければならない．GRAPE-6 のドライバルーティンは非常に単純化されており，マルチスレッドによる制御に適していない．このため，複数のクライアントを個別に処理する複数スレッドを立ち上

げ，時分割処理するという方法はとれない．したがって，複数クライアントを単一スレッド（プロセス）で処理する必要が生じるため，もし 1 つのクライアントの通信遅延が大きいと，上述の入出力処理の最中，他のより近距離（時間的に）のクライアントが待たされることになってしまい，システムのスループットの低下を招く．このような通信遅延を隠蔽するために，各クライアントとサーバの間で十分な容量を持つなんらかのバッファリングを行う必要がある．

**(3) ローカルなクライアントの同時かつ優先的な処理**

既存の HMCS では，クライアントはサーバに対して直接 TCP/IP による通信を行っていた．我々は，GRAPE-6 を所有しているサイト内でのローカルな HMCS サービス（以後，これを HMCS-G と区別するため HMCS-L サービス，または単に HMCS-L と呼ぶ）は継続したい．すなわち，グリッド RPC を介したサービスのかたわら，これを介さない GRAPE-6 クラスタへの直接的なアクセスも同時に処理するようにしたい．その際，HMCS-L サービスを受けるローカルなクライアントが，外部クライアントからの低速通信の影響に巻き込まれることなく，高いスループットでの処理を継続できるようにする必要がある．

**(4) 粒子の一部に対する永続的な保持機能**

銀河形成シミュレーションにおいて，いわゆるダークマター（dark matter）の存在は，そうでない「見える」粒子と同様にきわめて重要な要素である．自己重力を含む RT-SPH において，ダークマターは RT-SPH 過程には貢献せず，重力計算にのみ影響を与える．したがって，RT-SPH 過程は通常物質のみで計算が行われるため，ダークマターの物理量は必ずしも汎用計算機部に送り返す必要はなく，重力計算部にそのまま保持すればよい．ダークマターの物質量は大きな割合を占め，たとえば我々の典型的なシミュレーションでは全粒子の半分がダークマターである．したがって，汎用計算機部との間でダークマターの物理量をやりとりせずに済ませられれば，入出力におけるデータ転送量を約半分に抑えることができ，通信時間を短縮することが可能となる．

我々は，RWCP[9]プロジェクトにおいて開発され，現在筑波大学で継続研究されている OmniRPC[10),11)]を，HMCS-G におけるグリッド RPC として利用することにした．このほかにも Ninf-G[14)]等のグリッド RPC があるが，OmniRPC はサーバ stub プロセスの永続性という特徴を持っており，上述した通信遅延隠蔽やダークマターの処理等に非常に適している．外部のクライアントからの通信は低速であることが予想されるため，stub プロセスがこれを中継することにより，実際の HMCS サーバとの通信は短時間で済ませ，他のクライアントの処理を妨げないようにすることが可能となる．また，サーバ stub プロセスを，従来の HMCS-L と同じプロトコルで GRAPE-6 サーバと通信するような形で実装し，クライアントの要求する処理を API 化すれば，HMCS-L との共存も自然な形で維持することが可能となる．**図 2** にこの様子を示す．

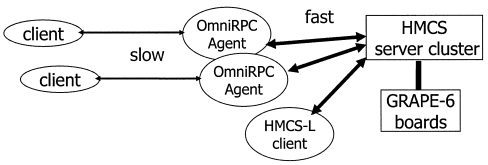



**図 2** HMCS-L プロトコルによるアクセスと OmniRPC 経由でのアクセスの関係

Fig. 2 Direct access with HMCS-L protocol and indirect access with OmniRPC to server cluster.

## 4. OmniRPC

本章では，HMCS-G において使用するグリッド RPC（Remote Procedure Call：遠隔手続き呼び出し）である OmniRPC について概説する．ここでは，OmniRPC の機能のうち RPC 機能として直接関係する部分のみについて述べる．このほかに OmniRPC には OpenMP プログラムからのシームレスな移行や，クラスタ対応の入出力機構等が備わっているが，それらの詳細については文献 10)，11) を参照されたい．

OmniRPC はその API と基本アーキテクチャを Ninf[13)]から継承している．クライアントと，遠隔手続きを実行する遠隔ホストとは，LAN や WAN 等で接続されている．遠隔ライブラリは，ネットワーク経由でのリクエストを受付ける stub ルーティンを main とする実行可能形式のサブルーティンとして実装される．このような実行可能形式プログラムを，remote executable（program）と呼ぶ．

ユーザは遠隔ライブラリを，ネットワークプログラミングの知識なしに呼び出すことができる．クライアントはその元プログラム中で長時間の計算を要する部分の処理を，遠隔ホスト（クラスタやスーパーコンピュータ）に依頼して処理することが可能となる．

`OmniRpcCall` は遠隔ルーティンを呼び出す最も単純なインタフェースである．例として，以下のようなインタフェースで書かれた，C 言語による単純な行列積プログラムを考える．

```
/* declaration */
double A[N][N],B[N][N],C[N][N];
```

```
....
/* calls matrix multiply, C = A * B */
dmmul(A,B,C,N);
```

もし dmmul ルーティンが遠隔ホストで利用可能ならば，クライアントはその遠隔ライブラリを OmniRpcCall を用いて以下のように呼び出すことができる．

```
/* call remote library */
OmniRpcCall("dmmul",A,B,C,N);
```

ここで，dmmul は遠隔ホストの executable に登録されたエントリ名であり，A，B，C，N はその引数である．このように，クライアントプログラム内では，あたかもそのルーティン呼び出しが単なるローカルな呼び出しであるかのように記述できる．OmniRpcCall はそのルーティンの実行可能性と，各引数の型および適切な渡し方を自動的に判別し，遠隔ホストに対する遠隔手続き呼び出しを実行し，結果を取り出した後，それを返り値の引数に正しくセットしてからクライアントに戻る．

クライアントプログラムのインタフェースをこのように単純化するため，クライアントの遠隔手続き呼び出しにおいて，遠隔ライブラリのすべての情報を実行時に参照する．その情報は，引数の数，その型とサイズおよびアクセスモード（読み出し/書込み）からなる．この情報を基に，RPC は引数の処理を行い，データの送受信処理を生成し，遠隔ホストとの通信を行う．この設計は，従来の RPC が行っているような，stub の生成をコンパイル時にクライアント側ですべて行う方法とは異なっている．

遠隔ルーティンのインタフェースは Ninf IDL（Interface Description Language）に基づいて記述される．以下は行列積ルーティンの IDL 例である．

```
Module MatrixMult;
...
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])
"... description ..."
/* Use C calling convention. */
Calls "C" dmmul(n,A,B,C);
```

mode_in と mode_out は引数が入力か出力かを示す．引数のサイズは，他の mode_in モードの引数を使って示すことも可能である（Fortran の整合配列と同等の機能と考えてよい）．この例では n の値が配列 A，B，C の大きさを決定している．

OmniRPC では，remote executable の永続性を明示的に利用するための API として OmniRpcHandle を提供している．ユーザは OmniRpcHandle によって与えられるハンドルを利用して，遠隔ホストで走っている特定の remote executable を手続き処理サーバとして指定することができる．

OmniRPC のもう 1 つの目的は，クラスタからグリッドまでのシームレスなプログラミング環境を提供することである．ユーザは OmniRPC を使って，同じプログラムをクラスタでもグリッドでも実行することができる．OmniRPC は遠隔プロセスの起動方法として，ローカル環境のために rsh，グリッド環境のために Globus[8]，そして任意の遠隔ホストのために ssh という，3 種類のオプションを提供している．特に ssh については，一般的なサイトでほとんどポート制限がかけられていないため，firewall 問題を回避できる．OmniRPC はこれを最大限に利用するため，ssh の port forwarding 機能を用いたクライアント・サーバ間の通信をサポートしている．我々は，グリッド上の資源の典型例は広範囲の地域に分散された「クラスタのクラスタ」であると考えている．ユーザは remote executable を実行するために，ローカルなジョブスケジューラに指示を与える．プライベートアドレスを持つクラスタを遠隔アクセスするために，遠隔ホストで起動される OmniRPC のエージェントプロセスがそれらの入出力要求を束ね，まとめてクライアントに伝達する．この機能により，クライアントは 1,000 台規模のノードを持つクラスタを単一のサーバとして利用することも可能となる．

## 5. HMCS-G の実装

### 5.1 HMCS サーバのマルチクライアント化

HMCS のグリッド RPC 化に先立ち，従来の HMCS サーバ（HMCS-L サーバ）をマルチクライアント化した．元々HMCS サーバは各々の PCI バスに GRAPE-6 ボードを 1 枚ずつ実装した，複数の Linux PC からなるクラスタで構成されている（現在は 8 ノード）．これに対し，クライアント側も一般的に複数の代表プロセス（以後，これらを client agent と呼ぶ）から通信が行われる☆．したがって，単純な一対一のクライアント・サーバ通信ではなく，複数対複数（それらの数も一致しない）の関係を処理する必要がある．このため，HMCS-L サーバは MPI による並列プログラムで

☆ クライアント側に MPP が想定されていることから，数百・数千のプロセスからの一斉通信を受けることは通信効率の点から非現実的であるため，client agent が通信をとりまとめることを前提としている．また，HMCS-L で利用されている並列ネットワークを有効利用するため，client agent の数は数個から数十個程度に調整される．

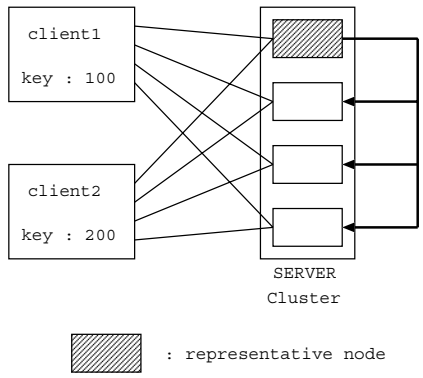



**図 3** HMCS-L サーバのマルチクライアント化
Fig. 3 Multi-client access to HMCS-L server.

実装されている．

HMCS-L サーバのマルチクライアント化では，このような並列 client agent が，さらに複数組存在するという状況に対応しなければならない（**図 3** 参照）．各 client agent はサーバに対して非同期にアクセスを行うため，単純に各サーバノードが TCP/IP における accept() および select() による受信処理を行っていては，GRAPE-6 ボード群を同一のクライアントのために利用することができない．したがって，この同期をとるために，サーバノードのうち最も番号の若いノードを代表ノードとし，他のサーバノードはこの代表ノードが選択したクライアントに対するサービスのみを実行する形で処理を行う（代表ノードのみが select() による非決定的なリクエスト受付を行い，他のノードは決定的な処理を行う）．

このほかに，client agent からの処理量に応じて，サーバノードを空間分割して効率的に処理する方法も考えられるが，全体的な処理のスケジューリングが複雑になるため，得策ではない．我々が想定する問題の大きさは，少なくとも数万粒子，多い場合は百万粒子単位での処理であり，8 ノードのサーバでの空間分割を行う必要はないと考えられる．したがって，全クライアントの中から，各クライアントを構成する client agent 群ごとにいっせいに処理を行う，いわゆる gang scheduling 的なサーバ処理を行うことにする．このスケジューリングは，前述したように，サーバノードの中で代表となるノードが実質的な処理受付を行い，他のノードはこのノードが決定した client agent 群のみの処理を受付けることにより，結果的に常に 1 つの client agent 群がその回の GRAPE6 処理を占有することになる．

また，従来の HMCS-L サーバでは単一クライアント（client agent 群）を対象としていたため，複数の入出力フェーズを分割せず，一括して扱っていた．しかし，マルチクライアント化においては，クライアントごとの通信遅延により，各通信フェーズの間に時間間隔が空くことが想定される．さらに，グリッド環境を想定した場合，通信途中，あるいは通信フェーズ間でクライアントが死んでしまい，処理が中断される可能性もある．このような状況を想定し，HMCS-L サーバの robustness を上げるため，通信フェーズの細分化と，すべての通信におけるタイムアウト処理を追加した．また，終了処理を行わず長時間にわたってアクセスが途切れたクライアントについても，処理を強制終了し管理テーブルの garbage collection を行う機能を追加した．

その他の追加機能として，クライアントの識別機能がある．複数のクライアントからの処理要求は TCP コネクションによって管理されるため，内部処理自体には識別子は特に必要ない．しかし，接続元別のアカウンティングやイベントログ情報のため，個々のクライアントに識別子を持たせることにした．識別子はクライアントのホスト IP アドレス（およびホスト名）と，整数のキー値の組合せである．後者については，クライアント側のライブラリによって作成され，現状ではその client agent 群の中のマスタノード（サーバ上の代表ノードに接続してくるノード）のプロセス ID が用いられている．これにより，同一ホストまたはクラスタ上で同時に複数の client agent 群が走っている場合でも，それらを一意に識別することが可能になる．また，後述する OmniRPC の agent stub プログラムは，実際のユーザアプリケーションのキー値を受け渡すようプログラムされている．これにより，サーバは従来の HMCS-L ベースのクライアントと HMCS-G ベースのクライアントを統一的に管理することができる．

以上の方針に従って，HMCS-L サーバをマルチクライアント化した[12]．サーバは GRAPE-6 クラスタに，一種の daemon として常駐する形になる．我々は特定のサービスポートをこれに割り当て，運用している．我々の firewall 内のマシンからのアクセスは直接受付けられるが，このポートは firewall を通過できない．したがって，外部からのアクセスには OmniRPC を用いた HMCS-G プロトコルでのアクセスが必要になる．

### 5.2 グリッド対応

HMCS-G の実装は，従来の HMCS-L 用ユーザ API ライブラリの代わりに HMCS-G 対応のそれを提供す

ることと，遠隔ホストで実行される stub プログラムを提供することで実現される．特に，後者についてはどのホストでこれを実行するかが問題となる．HMCS-L のサーバノードで直接これを起動することも可能ではあるが，firewall における Globus サービスのポート管理やローカルな HMCS-L クライアントに対するサービスの優先度を考慮し，一般的には他のホストでこれを実行することとした．このホストは我々のサイト内に存在し，Globus のアクセスポートが空くよう firewall に登録されている．また，OmniRPC の ssh によるアクセスも可能とするため，ユーザアカウント管理も行う．

ユーザに利用しやすい形でシステムを提供するため，我々は GRAPE-6 の API[6] とほぼ同様の考え方で HMCS-G の API を用意した．このライブラリは，3 章で述べた粒子データの入出力を記述することにより，OmniRPC の諸機能を用いて実際の RPC を実行するようプログラムされている．たとえば，粒子データをセットして計算を起動する以下のインタフェース

```
gg6calc(int np, double m[], double r[][3],
double f_old[], double pot[], int *error);
```

により，以下の処理が実行される．

```
OmniRpcExecCall(handle, "gg6_calc", np, m, r,
f_old, pot, error);
```

ここで，`OmniRpcExecCall` は `handle` によって指定される remote executable に対する処理要求を発行する．

stub プログラム（以後，これを server agent と呼ぶ）では，クライアントからの要求を受け，これを改めて HMCS-L プロトコルに従って GRAPE-6 サーバクラスタに送る．上述のサービスに対応する IDL は以下のように記述される．

```
Define gg6_calc(mode_in int np, mode_in double
m[np], mode_in double r[np][3], mode_in double
f_old[np], mode_in double pot[np], mode_out int
*error);

Calls "C" gg6_calc(np, m, r, f_old, pot, err);
```

関数 `gg6_calc` 内では，クライアントからの要求を実際に HMCS-L サーバに送る HMCS-L プロトコルの通信ライブラリが呼び出される．server agent は他の HMCS-L 上のローカルなプロセスと同様，GRAPE-6 サーバに近い場所で起動されるため，その通信は高速である．server agent は GRAPE-6 のローカルサイトにおける「門番」の役割と，データのバッファリングを行う．

**図 4** に HMCS-G の実行環境のブロック図を示す．

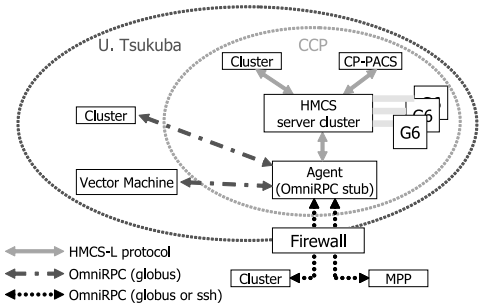



**図 4** HMCS-G における通信とプロセスの関係
Fig. 4 Diagram of processes and communications in HMCS-G.

前章で述べたとおり，OmniRPC における remote executable（ここでは server agent）を起動するには rsh, ssh そして Globus の GRAM[8]，の 3 種類の方法がある．現在の Grid 環境の標準である Globus を用いることは，これが実装されているサイトからのアクセスを容易にする．しかし，Globus を用いるためには，それが利用するポートを空けておく必要があり，firewall が設置されている場合にそのポートを特別に空けてもらう等の措置が必要となる．これに対し，ssh オプションによる OmniRPC では，ssh の port forwarding 機能を流用してデータを転送するため，このような心配が不要である☆．加えて，Linux クラスタでは ssh が標準的にサポートされているため，Globus 等のグリッドインフラストラクチャに不案内なアプリケーションユーザにとっても簡単に利用できるというメリットがある．我々のサイトでは，ssh と Globus のそれぞれに対応する server agent を個別に用意している．

## 6. 性能評価

### 6.1 実験環境

HMCS-G の性能を評価するため，我々はいくつかのサイトからの遠隔アクセスにおける性能評価を行った．ここでは実際の RT-SPH アプリケーションをクライアント上で走らせるのではなく，単に GRAPE-6 サーバに HMCS-G プロトコルでアクセスし，結果を受け取るという作業を繰り返すという，いわば ping-pong 転送的なベンチマークを実行した．実アプリケーションではないが，通信コスト等についてはそれと同等の処理を行うため，実質的な性能評価と見なすことができる．なお，本実験では通信性能に着目するため，8 ボードの GRAPE-6 のうち 1 枚だけを用い，1 つの client

☆ firewall を敷いているサイトでも，ssh によるアクセスは許可されているのが一般的である．

**表 1** CCP から各サイトへのネットワーク環境
Table 1 Network environment between CCP and other sites.

| | |
|---|---|
| 筑波大内 | 1 Gbps バックボーン + 100 Mbps 枝線 |
| 東工大 | 1 Gbps SuperSINET + 100 Mbps 枝線 |
| 立教大 | 商用プロバイダ（バンド幅不明） |

agent からの通信のみを受付ける形で評価を行った．

この実験には筑波大学，東京工業大学（東工大），立教大学の 3 つの大学が参加した．HMCS-G サーバ（HMCS-L サーバおよび server agent）は筑波大学計算物理学研究センター（CCP：Center for Computational Physics）に設置されている．以下の場所にある 5 種類のクライアントからのアクセスを処理した．

(1) CCP 内（HMCS-L プロトコル）
(2) CCP 外の筑波大内（OmniRPC/ssh）
(3) CCP 外の筑波大内（OmniRPC/globus）
(4) 東工大（OmniRPC/ssh）
(5) 立教大（OmniRPC/ssh）

**表 1** に CCP と他のサイトとのネットワーク接続状況を示す．CCP と東工大学術国際情報センター（GSIC）との間は，キャンパス間グリッド研究のための SuperSINET [15]が引かれており，両サイト間で 1 Gbps のバンド幅が専用線として利用可能になっている．立教大は大学自体が商用プロバイダでインターネットに接続されており，大学の出入口のバンド幅が 100 Mbps であるため，高い実効バンド幅は望めない．しかし，このような環境下での実測は興味深いものがあり，今回の実験に加えた．

### 6.2 基本的な round-trip 時間および通信性能評価

**表 2** に round-trip 時間の実測値を示す．各ケースとも，64 K（65,536）粒子のデータに対し，API で起動してからその結果が戻るまでの時間を計測している．ここにはサーバまでのデータ転送時間，重力計算処理時間，結果の転送時間が含まれる．表中，“r-t.” の欄は総時間，“comm.” の欄はそのうちデータ送受信にかかった時間（総時間から GRAPE-6 クラスタでの実処理時間 = 約 0.7 秒を引いたもの）を表している．“バンド幅” は，実転送データ量（この例では合計 5.24 MB）から計算した実効バンド幅である．実測は 100 回の通信を 1 セットとして，5 セットずつ行い，最良のセットを採用した．

まず，CCP 内での HMCS-L による処理では，通信は 100 Mbps Ethernet の性能をほぼ出しており，理想的な通信が行えている．これに対し，通信バンド幅としてはほぼ同じ条件と考えられる筑波大内からの HMCS-G アクセスでは，Globus オプションで性能が約半分に，ssh オプションでは約 1/3 に落ちている．いずれも OmniRPC を用いた server agent が介在することにより，通信オーバヘッドが生じていることが原因と考えられるが，ssh の方が Globus に比べ性能が落ちているのは，ssh でデフォルトに行われているデータの暗号化が原因と考えられる．東工大からのアクセスでは，実効バンド幅が約 25%まで落ちている．これは，SuperSINET ルータまでの経路が複数段の 1000 base-LX または 1000 base-SX スイッチを経由し，さらに筑波大・東工大に設置されたそれぞれのルータ間は数十 km を光スイッチを経由して MPLS 接続されているため，全体的な遅延時間が大きいことが原因である．実際，東工大のクライアントマシンから筑波大側の server agent を実行させるマシンまで ICMP ping を実行すると，2.3 msec の遅延時間が報告される．しかし，実際に RT-SPH のような重いクライアントを実行する際は，そちらの処理が実行時間の中では支配的となり，また 3 章でも述べたように，通信遅延隠蔽をアルゴリズムに含めることも可能であるため，この程度のバンド幅であれば許容範囲である．

**表 2** 64K 粒子に対する処理の round-trip 時間
Table 2 Round-trip time to process 64 K particles.

| サイト | 方法 | 時間 (sec) | | バンド幅 |
|---|---|---|---|---|
| | | r-t. | comm. | (MB/s) |
| CCP 内 | HMCS-L | 1.2 | 0.5 | 10.48 |
| 筑波大内 | globus | 1.7 | 1.0 | 5.24 |
| 筑波大内 | ssh | 2.1 | 1.4 | 3.74 |
| 東工大 | ssh | 2.8 | 2.1 | 2.50 |
| 立教大 | ssh | 9.8 | 9.1 | 0.58 |

これらに対し，立教大との通信では，実効バンド幅は 1 MB/s を下回ってしまった．これは，商用プロバイダによるインターネット接続ではやはりこの種のアプリケーション利用には不向きであることを示しており，ここまで大きい通信遅延はアルゴリズム的にも隠蔽が難しくなる．とはいえ，実際に 64 K 粒子に対する重力計算を通常の PC クラスタ等で行った場合，この程度の時間では到底済まない．GRAPE-6 が粒子対に対して行う演算は約 80 浮動小数点演算に相当し，64 K 粒子の対に対する総演算量 ($O(64\,K^2)$) は，約 350 GFLOP に相当する．ピーク性能 100 GFLOPS のクラスタを使ったとして，この種の計算に対する実効性能は，キャッシュが有効利用できず，通信オーバヘッドも大きい点から考えると，ピークの 1 割程度であろう．その場合，重力計算に 35 秒かかる計算にな

る．RT-SPH の計算自体は $O(N)$ のコストで済むため，クライアント側としてはより多数の粒子を扱いたい．仮に粒子数が倍になっただけで，重力演算時間は 100 秒を超えてしまう．したがって，たとえ 10 秒の round-trip 時間がかかったとしても，GRAPE-6 を遠隔利用する意義はあるといえる．

### 6.3 OmniRPCを介することによるオーバヘッド

前節では HMCS-G において OmniRPC を介した場合の ssh オプションと Globus オプションの差について延べたが，ここでは OmniRPC そのもののオーバヘッドが，特にリモート環境において与える影響を調べる．本来，HMCS-L で用いられるサーバへのアクセスポート番号は，一般的にファイヤウォールを通過しない特定の番号である．このため，たとえばインターネット越しに行われる立教大学からのアクセスを直接処理することはできない．そこで，ssh による明示的な port forwarding 機能の利用（ssh コマンドの '-R' または '-L' オプションの利用）により，クライアント側のローカルポートへのアクセスをサーバのポートに forwarding し，結果的に HMCS-L プロトコルによるサービスを実現させ，性能評価を行った．なお，この過程では ssh による暗号化等のオーバヘッドが含まれるが，元々OmniRPC の ssh オプションにおいても同様のオーバヘッドが存在するため，両者を比較することにより，かえって OmniRPC が介在する真のオーバヘッドを観察できると考えられる．

粒子数が変化した場合の性能について評価した結果を**図 5** に示す．ここでは，処理粒子数を 20,000 から 100,000 まで変化させ，クライアントから見た round-trip 時間（すなわちアプリケーションレベルで重力計算処理要求を出してからその結果が返るまでの時間）をグラフにしている．実行は各ケースごとに 20 反復し，その平均値を示している．汎例中，'RIKKYO' および 'TITECH' は，それぞれ立教大および東工大を示す．また，'OMRPC' および 'SSH' は，それぞれ OmniRPC（ssh オプション）を用いた場合と，ssh の port-forwarding を利用した HMCS-L プロトコルによる処理を示す．さらに，参考のために我々のローカルサイトからの HMCS-L プロトコルによる処理の場合の round-trip 時間（'LOCAL'）と，要求がサーバに届いてから実際に GRAPE-6 が処理をする正味のサービス時間（'SERVICE'）を添えた．このサービス時間であるが，すべてのケースについて測定した結果，粒子数が同じであればほぼ一定であり，有意な差は認められなかった（誤差は有効数字 3 桁目）．そこで，以降の性能評価でもこのサービス時間は粒子数にのみ依存し，クライアントのサイトやアクセス方法にはとらわれないものとして扱う．

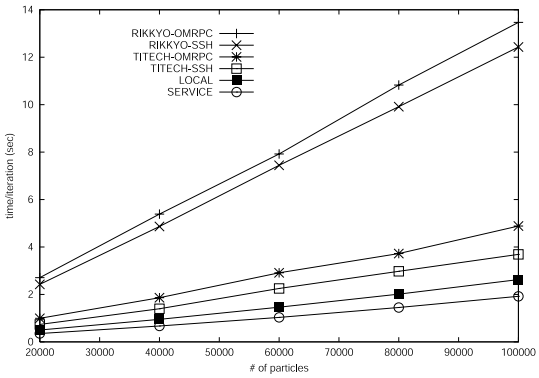



**図 5** OmniRPC を通した場合と ssh による port-forwarding の差
Fig. 5 Difference between cases with OmniRPC and with ssh port-forwarding.

まず，全体的に OmniRPC を介した場合は ssh port-forwarding による場合に比べ，若干性能が落ちていることが分かる．これは ssh port-forwarding では基本的にストリーム型のバッファリングが行われているのに対し，OmniRPC ではソフトウェアの処理層が厚いため，より大きなオーバヘッドが生じているためである．しかし，その差はそれほど大きくなく，東工大の場合で round-trip 時間で約 30%増し，バンド幅の狭い立教大の場合は 10〜13%程度である．このように，OmniRPC を介在させることにより，単一クライアントのサービスにおいては若干オーバヘッドが加えられることが確認された．

### 6.4 マルチクライアントによる処理性能

5.2 節で述べたように，HMCS-G における server agent には，クライアントからのアクセス権の制御だけでなく，データをバッファリングし，特に高レイテンシ・低バンド幅の遠隔通信によって，サーバがデータの送受信のために長時間占有されることを防ぐ効果が期待される．ここでは，その性能について評価する．

我々のサイト内のローカルなクライアントと，HMCS-G プロトコルに基づき server agent を介してアクセスしてくるリモートなクライアントを同時に実行させ，ローカルクライアントの処理時間にどれくらいの遅れが生じるかを測定する．前節で取り上げたように，リモートクライアントからのアクセスには OmniRPC 経由のものと，ssh port-forwarding を利用した HMCS-L プロトコルによる直接アクセスの両者を比較する．また，参考のために，測定対象とするローカルクライアントと同じものをもう 1 つ，我々の

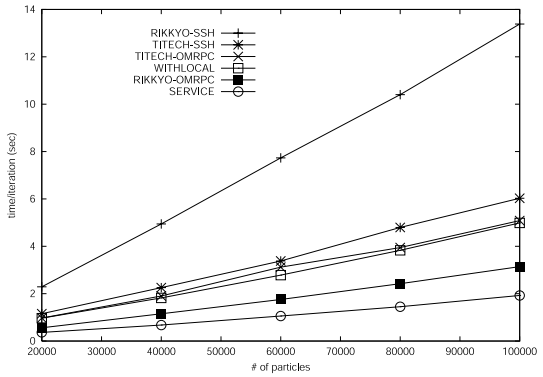



**図 6** マルチクライアント処理におけるローカルクライアントの処理性能低下
Fig. 6 Performance degradation of local client interfered by other clients.

サイト内の別のクライアントマシンから走らせ，その結果とも比較する．

**図 6** に結果を示す．図中の ‘RIKKYO’，‘TITECH’，‘OMRPC’，‘SSH’ の意味は図 5 と同じであるが，ここに示されているのはそれらのリモートクライアントの処理時間ではなく，それと並行して行われているローカルクライアントの処理時間（round-trip 時間）である．また，‘WITH LOCAL’ と示されているのは，もう 1 つのローカルクライアントと一緒に走らせた場合の結果である．

この図から，server agent によるバッファリング効果が非常に大きいことが分かる．特に，レイテンシが大きくバンド幅が狭い立教大学との通信においては，ssh port-forwarding による HMCS-L プロトコルでの通信によってローカルクライアントが被る「被害」が大きく，OmniRPC stub を用いた場合に比べ処理時間は 4 倍以上に増大してしまう．5.1 節で述べたように，HMCS-G のサーバ構築のため，従来の HMCS-L プロトコルを見直し，処理フェーズの細分化等も行っている．しかし，サーバが通信フェーズに入ってしまうと，通信が完了するまでは他のクライアントの処理が行えないため，結果的にリモートクライアントとの実通信時間が他のローカルクライアントへの影響を決定してしまう．

OmniRPC stub を介在させることにより，サーバへの通信時間は他のローカルクライアント並となり，この結果，複数のローカルクライアントがサーバを共有しているのと同じ状態になる．この影響は，東工大における OmniRPC 利用の場合（‘TITECH-OMRPC’）と，2 つのローカルクライアントを走らせた場合（‘WITH LOCAL’）の結果がきわめて近いということから見てとれる．興味深いのは，‘RIKKYO-OMRPC’ と ‘WITH-LOCAL’ の逆転現象である．ここでは，立教大からのリモートクライアントを OmniRPC 経由で処理している場合のほうが，ローカルクライアント同士を走らせている場合よりも性能が上がっているように見える．しかし，このグラフはそのような他のクライアントが存在する場合の，ローカルクライアントの処理性能を示しているため，矛盾は生じていない．すなわち，この場合はリモートクライアントの 1 反復の処理が終わると，次にリクエストが来るまでの時間が非常に長い（この間，リモートクライアントは server agent と通信している）ため，ローカルクライアントに割り当てられるサービス機会が増えることにより，結果的にローカルの処理性能が相対的に向上しているためである．

**表 3** ダークマターを server agent 上に保持する場合の処理時間（round-trip）
Table 3 Round-trip time when keeping dark matters on server agents.

| 粒子数 | 処理時間 (sec) | | 処理時間比 |
|---|---|---|---|
| | Tnormal | Tdarkmatter | |
| 20000 | 2.708 | 1.700 | 0.628 |
| 40000 | 5.386 | 3.314 | 0.615 |
| 60000 | 7.920 | 4.759 | 0.601 |
| 80000 | 10.826 | 6.395 | 0.591 |
| 100000 | 13.470 | 8.206 | 0.609 |

以上の実験の結果，HMCS-G における server agent のデータバッファリング処理が非常に効果的に機能し，グリッド環境におけるリモートクライアントの存在がローカルクライアントに与える影響を最小限に抑えることが可能であることが示された．

### 6.5 ダークマターの扱い

最後に，server agent 上でダークマターの粒子データを保存し，リモートクライアントにはそれ以外の有効粒子（たとえば RT-SPH 過程における SPH 粒子）のみを返すようにプログラムした場合の性能評価を行った．立教大学にクライアントを置いて処理した結果を**表 3** に示す．なお，この実験ではダークマターの粒子数を全粒子数の半分としている．これは我々の RT-SPH シミュレーションの典型的なモデルである．

表の中で，$T_{normal}$ はダークマターを保持しない通常の方式の処理時間，$T_{darkmatter}$ はダークマターを server agent 内に保持する方式の処理時間，処理時間比は $T_{normal}$ に対する $T_{darkmatter}$ の比である．見てわかるように，ダークマターの保持を行うことにより，データ通信量がほぼ半分に抑えられた結果，処理

時間は約 60%に短縮されている．処理時間が半分にならない理由は，ダークマターを保持したとしても，server agent から HMCS-G サーバまでの通信量は削減されないことと，加速度データをクライアントに返す前に，ダークマターの時間積分を行っているため server agent 内でもある程度の数値計算を行っているためである．

ダークマターの時間積分処理量はダークマターの粒子数に比例する．また，我々のモデルのように SPH 粒子数とダークマター粒子数の比率が一定であるならば，結局クライアントと server agent の通信量，server agent と HMCS-G サーバとの間の通信量，ダークマターの時間積分時間はすべてその比率が保たれることになる．よって，この速度比は粒子数がさらに増大した場合でもほぼ一定に保たれると予想される．したがって，OmniRPC stub の持つ永続性機能を利用し，ダークマターを server agent 内に保持する方式は，クライアントと server agent 間の通信量を削減する効果により，特に高遅延・低バンド幅の環境において非常に有効であることが確認された．

### 6.6 実アプリケーションの結果

**図 7** に，本システムを利用して実行したシミュレーション結果の一例を示す．これは，宇宙初期の異なる時期にガス雲の形成が始まった 2 種類の銀河の様子を示しており，(カラー表示で) 赤い点は熱いガス（〜$10^4$ K），緑の点は冷たいガス（〜$10^2$ K），そして青い点は星の要素を表している．これはシミュレーションの最終段階を画像化したものであり，ピーク性能 50 GFLOPS の Alpha CPU ベースのクラスタをクライアントとして用いた場合，1 つのシミュレーションについて 40,000 ステップの実行に約 50 時間を要している（シミュレーション時間はパラメータにより異なる）．平均すると 1 ステップあたり約 4.5 秒かかっており，通信遅延隠蔽処理により，1〜2 秒程度の HMCS-G サーバ round-trip 時間が十分実用になることを示している．

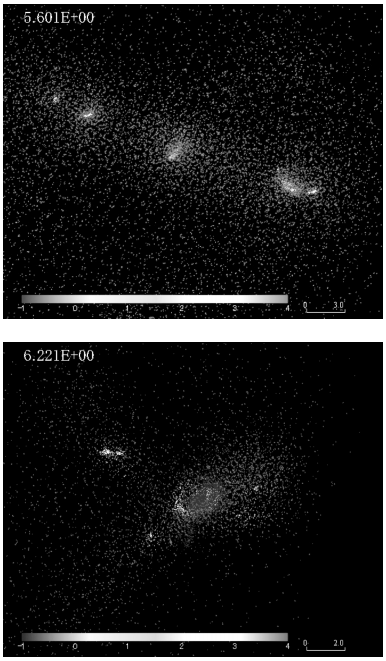



**図 7** HMCS-G によるシミュレーション結果例
Fig. 7 Example of simulation result with HMCS-G.

## 7. おわりに

我々は，グリッド環境上に分散した汎用並列計算機群と専用並列計算機を結合し，各種物理現象の複合シミュレーションを可能とする計算環境 HMCS-G を設計し，重力専用計算機 GRAPE-6 クラスタを中心とする計算宇宙物理学用のシステムを実装した．アプリケーションプログラミングの簡便さとシステム全体のスループット向上のため，グリッド RPC として OmniRPC を用い，HMCS-G の特性に合わせた効率的なシステムを実現した．各種ネットワーク環境における性能評価の結果，ある程度のネットワークバンド幅のもとで，HMCS-G は実アプリケーションの現実的な実行に耐える性能を提供できることが確認された．また，HMCS-G により，貴重な専用計算リソースである GRAPE-6 のようなシステムを世界規模で共有することが可能となる．

このような各種の物理現象を含む複合シミュレーションは，次世代の計算科学における必須要件の 1 つである．HMCS-G は汎用計算機群と専用計算機をローカルな環境に揃えることなしに，両者を効率的に利用する手段を提供する．このような複合システム構築の試みは，グリッド環境を実アプリケーションに活用するうえで非常に重要な意義を持つ．今後の予定としては，RT-SPH を中心とする各種アプリケーションを実際の分散環境で実行し，システムの有意性をさらに確認するとともに，クライアント自身の分散を自動化する工夫等を行っていく予定である．

**謝辞** 本研究を進めるにあたり，貴重な御意見をいただいた筑波大学および東京大学の HMCS プロジェクト関係者に感謝します．また，遠隔通信実験の環境を提供していただいた東京工業大学松岡聡教授ならびに松岡研グリッド研究チームの皆様に感謝します．本研究の一部は，科学研究費補助金特定領域研究（2）課

題番号 14019011「計算物理学分野の Grid アプリケーションと並列プログラミングシステムの研究」による．

## 参 考 文 献


1) Boku, T., et al.: Heterogeneous Multi-Computer System: A new platform for Multi-Paradigm Scientific Simulation, *Proc. ICS02*, pp.26–34 (2002).
2) 朴　泰祐ほか：HMCS における重力効果を含む宇宙輻射流体計算，情報処理学会論文誌：ハイパフォーマンスコンピューティングシステム，Vol.43, No.SIG6 (HPS5), pp.219–229 (2002).
3) Nakazawa, K., et al.: CP-PACS: A massively parallel processor at the University of Tsukuba, *Parallel Computing*, Vol.25, pp.1635–1661 (1999).
4) Umemura, M.: Three-dimensional hydrodynamical calculations on the fragmentation of pancakes and Galaxy formation, *The Astrophysical Journal*, Vol.406, pp.361–382 (1993).
5) Nakamoto, T.: A 3-D Radiative Transfer Solver using a Massively Parallel Computer, *Numerical Astrophysics 1998* (1998).
6) GRAPE-6 documents (on WWW), http://grape.astron.s.u-tokyo.ac.jp/pub/people/makino/softwares/GRAPE6/.
7) Makino, J., et al.: A 29.5 TFLOPS simulation of planetesimals in Uranus-Neptune region on GRAPE-6, Gordon Bell Prize nominee, SC2002 (CD-ROM) (2002).
8) Foster, I. and Kesselman, C.: Globus: A metacomputing infrastructure toolkit, *Int. Journal of Supercomputer Applications*, Vol.11, No.2, pp.115–128 (1997).
9) http://www.rwcp.or.jp/
10) Sato, M., et al.: OmniRPC: a Grid RPC facility for Cluster and Global Computing in OpenMP, *Proc. Workshop on OpenMP Applications and Tools 2001* (LNCS 2104), pp.130–135 (2001).
11) 佐藤三久ほか：OmniRPC: グリッド環境での並列プログラミングのための Grid RPC システム，先進的計算基盤システムシンポジウム SACSIS2003 論文集，pp.105–112 (2003).
12) 小沼賢治ほか：重力計算専用計算機 GRAPE-6 のリモートアクセス環境，情報処理学会研究会報告ハイパフォーマンスコンピューティング，2003-94-006, pp.31–36 (2003).
13) Sato, M., et al.: Ninf: A Network based Information Library for Global World-Wide Computing Infrastructure, *Proc. HPCN'97* (LNCS 1225), pp.491–502 (1997).
14) http://ninf.apgrid.org
15) "What is Super SINET", http://www.sinet.ad.jp/english/super_sinet.html.


(平成 15 年 2 月 3 日受付)
(平成 15 年 5 月 9 日採録)

**朴　泰祐**（正会員）

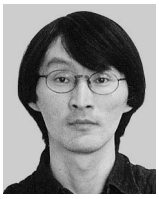
昭和 59 年慶應義塾大学工学部電気工学科卒業．平成 2 年同大学大学院理工学研究科電気工学専攻後期博士課程修了．工学博士．昭和 63 年慶應義塾大学理工学部物理学科助手．平成 4 年筑波大学電子・情報工学系講師，平成 7 年同助教授，現在に至る．超並列処理ネットワーク，超並列計算機アーキテクチャ，ハイパフォーマンスコンピューティング，並列処理システム性能評価の研究に従事．平成 14 年度情報処理学会論文賞受賞．電子情報通信学会，日本応用数理学会，IEEE 各会員．

**佐藤　三久**（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 61 年同大学大学院理学系研究科博士課程中退．同年新技術事業団後藤磁束量子情報プロジェクトに参加．平成 3 年通産省電子技術総合研究所入所．平成 8 年新情報処理開発機構並列分散システムパフォーマンス研究室室長．平成 13 年より，筑波大学電子情報工学系教授．同大学計算物理学研究センター勤務．理学博士．並列処理アーキテクチャ，言語およびコンパイラ，計算機性能評価技術，グリッドコンピューティング等の研究に従事．日本応用数理学会会員．

**小沼　賢治**

平成 15 年筑波大学第三学群情報学類卒業．現在，同大学大学院システム情報工学研究科に在学中．専攻はコンピュータサイエンス．ハイパフォーマンスコンピューティング向けミドルウェアに関する研究に従事．



**牧野淳一郎**（正会員）

昭和 60 年東京大学卒業．平成 2 年同大学大学院総合文化研究科博士課程修了，博士号取得．東京大学教養学部助手，同教養学部助教授を経て，現在東京大学大学院理学系研究科天文学専攻助教授．専門は理論天文学，特に恒星系力学．主に興味があるのは球状星団，銀河中心等の高密度恒星系の熱力学的進化．平成 7 年，8 年および平成 11〜13 年ゴードンベル賞受賞．平成 10 年日本天文学科林忠四郎賞受賞．

**須佐　元**

平成 6 年京都大学理学部卒業後，平成 9 年同大学大学院理学研究科物理学宇宙物理学専攻で理学博士取得．平成 12 年 4 月より筑波大学で助手，平成 14 年 4 月より立教大学理学部物理学科専任講師．専門は宇宙物理学の理論的研究．特に輻射流体の問題としての銀河および第一世代天体の形成を研究課題としている．最近は隕石中のコンドリュールという構造の形成に関しても研究を始めている．

**高橋　大介**（正会員）

昭和 45 年生．平成 3 年呉工業高等専門学校電気工学科卒業．平成 5 年豊橋技術科学大学工学部情報工学課程卒業．平成 7 年同大学大学院工学研究科情報工学専攻修士課程修了．平成 9 年東京大学大学院理学系研究科情報科学専攻博士課程中退．同年同大学大型計算機センター助手．平成 11 年同大学情報基盤センター助手．平成 12 年埼玉大学大学院理工学研究科助手．平成 13 年筑波大学電子・情報工学系講師．博士（理学）．並列数値計算アルゴリズムに関する研究に従事．平成 10 年度情報処理学会山下記念研究賞，平成 10 年度情報処理学会論文賞各受賞．日本応用数理学会，ACM，IEEE，SIAM 各会員．

**梅村　雅之**

昭和 32 年生．昭和 57 年北海道大学物理学科卒業．昭和 62 年同大学大学院理学研究科物理学専攻博士課程修了．理学博士．昭和 61，62 年学振特別研究員．昭和 63 年京都大学基礎物理学研究所非常勤講師．同年，国立天文台助手．平成 4 年米国プリンストン大学客員研究員．平成 5 年筑波大学物理学系（計算物理学研究センター）助教授．同年，国立天文台客員助教授．平成 14 年から筑波大学物理学系教授．専門は理論宇宙物理学．特に宇宙輻射流体力学による銀河形成，宇宙構造形成の研究に従事．平成元年〜2 年日本天文学会理事．日本天文学会，米国天文学会各会員．国立天文台客員教授，同理論計算機専門委員会委員，同総合計画専門委員会委員．