総合解説 スーパーコンピュータとプラズマ・核融合シミュレーションの発展とその将来

3．次世代コンピュータに向けた技術課題と展望

岡部寿男，妹尾義樹[1)]，岩下英俊[2)]
（京都大学学術情報メディアセンター，[1)]NECインターネットシステム研究所，
[2)]富士通㈱ソフトウェア事業本部）

Technological Issues and Prospect of the Next-Generation Supercomputer Systems

OKABE Yasuo, SEO Yoshiki[1)] and IWASHITA Hidetoshi[2)]
Academic Center for Computing and Media Studies, Kyoto University, Kyoto 606-8501, Japan
[1)]Internet Systems Research Laboratories, NEC Corporation, Kawasaki 211-8666, Japan
[2)]Software Group, Fujitsu Limited, Numazu 410-0396, Japan

(Received 6 April 2004)

Tremendous progress has been made in the enhancement of the peak performance of high-end supercomputers like the Earth Simulator. Tera-Flops machines are being very common, and the key to their success is large-scale parallel processing. From the programmers' point of view, the difficulty of parallel programming has remained the same in comparison with the progress made in hardware systems. In this paper, we outline recently developed parallel programming software technologies, and discuss the direction of their future development.

Keywords:
supercomputing system, parallel programming software, data-parallel programming language, High Performance Fortran, large-scale numerical simulation

3.1 はじめに

分散メモリ並列システム向けの並列プログラミング言語HPF（High Performance Fortran）[1]について，1993年春にその最初の仕様が定められてから，10年以上が経過した．その間の分散メモリ型並列コンピュータの性能向上は著しく，当時は夢と言われていた1 TFlopsを軽く凌駕し，TOP500（http://www.top500.org）における2003年11月のランキングにおいて，実に131台ものシステムが，Linpackベンチマークにおける実効値で1 TFlops以上の性能を達成している．

なかでも我が国が誇る地球シミュレータ（http://www.es.jamstec.go.jp）の性能は図抜けており，2002年11月に米国で行われたSuperComputing 2002国際会議において，この分野で最も権威があるGordon Bell賞を，地球シミュレータを用いた研究成果3件が実効性能部門・最高性能賞実効性能部門・言語賞，特別賞をそれぞれを受賞するなど，輝かしい評価を得ている．

言語賞を受賞したのは，データパラレル言語HPFを用いたプラズマシミュレーションの並列化の成果である[2]．陽解法のTVD差分スキームを用いたプログラムで，並列化が比較的容易なコードではあるが，手続きにまたがったSHIFT通信やリダクション演算を含む実用コードであり，MPI[3]で記述すればプログラムの全体構造の変換や，通信のための大量のMPI呼び出しが必要になる．このコードに対してわずか30数行程度（includeファイルに記述した共通の宣言文を複数手続きで引用しているので，実質的には10行程度）のデータマッピング指示を挿入するだけで，ピーク性能比45%にあたる14.9 TFlopsを達成したことが認められた．1996年頃から，ユーザである国内の大学，国立研究機関等のアプリケーション研究者・開発者と，国産スーパーコンピュータベンダの計算機言語・コンパイラ開発者が参加したインフォーマルな「HPF合同検討会」（JAHPF: Japan Association for HPF）の活動を続けてきた筆者らにとっては，ようやくここまで来たか，と感慨深いものがある．しかし，HPFの本当の実用化という意味では，まだまだ課題が残っているのも事実である．

本稿では，主としてコンパイラ技術者の視点から，スーパーコンピュータシステムの動向，並列処理のためのコンパイラ技術の動向を述べる．特に最近の並列プログラミングインタフェースの紹介とMPIやOpenMP[4]を含めた種々のプログラミングインタフェースの比較を行い，次世代コンピュータに向けての技術的な課題を整理するとともに，実用アプリケーションのHPF化促進や実環境でのHPFの評価などHPFの普及促進に関わっている立場から，今後の展望についての私見を述べる．

3.2 スーパーコンピュータシステムの動向

3.2.1 ベクトルマシンから並列コンピュータへ

Cray-1に始まる演算パイプライン方式のスーパーコンピュータ，いわゆるベクトルマシンは，1985年頃から1990

author's e-mail: yasuo@i.kyoto-u.ac.jp

年頃にかけて，商用システムとして全盛期を迎えた．CRAY，NEC，富士通，日立のハイエンドスーパーコンピュータだけでなく，Alliant や Convex などのミニスーパーといわれるシステムも広く使われ，ごく最近まで，ベクトルマシンはスーパーコンピュータの代名詞といえるほどであったが，この10年ほどに限って言えば，ベクトル型プロセッサ単体の演算性能はほとんど向上していない．

たとえば NEC の場合，SX-3（1990年）の倍精度浮動小数点演算性能が 6.4 GFlops であるのに対し，SX-6（2001年）は 8 GFlops である．富士通の場合，VP2600（1988年）の 5 GFlops に対し VPP5000（1999年）の 9.6 GFlops である．一方この間のスカラ型プロセッサの性能向上は著しく，富士通 PRIMEPOWER HPC2500に採用されている SPARC64V（1.56 GHz）は 6.24 GFlops を達成している．

すなわちこの10年間の性能向上は，プロセッサ単体の性能を維持しつつ小型化，低消費電力化，そして低コスト化をすすめ，それを多数台並列動作させることを可能にしたことによって達成されたものである．

3. 2. 2　並列コンピュータアーキテクチャの分類

並列計算機のアーキテクチャは大別して，共有メモリ型と分散メモリ型に分けることができる．

共有メモリモデルは，複数のプロセッサ間で主記憶のデータが共有されるシステムである．このため，プロセッサ間でのデータの明示的な受渡しは不要である．そのためプログラミングは容易になるが，接続できるプロセッサ台数に限界があるなどの欠点がある．共有メモリ型のシステムでも，すべてのプロセッサが対等に主記憶にアクセスできるものは SMP（Symmetric Multi Processors）と呼ばれる．

分散メモリモデルは，プロセッサと主記憶から構成されるシステムが複数個互いに接続されたシステムである．共有メモリモデルに比較して，大規模なシステムを構築可能であるという特徴がある．しかし，他のプロセッサのデータを利用するためには，明示的にデータの授受を行う必要があるなど，プログラミングは共有メモリモデルに比較すると難しい．

また，この 2 つの中間に位置づけられる分散共有メモリ（Distributed Shared Memory; DSM）型アーキテクチャ[5-7]もある．DSM システムとは，物理的に分散配置されたメモリを共有メモリとして構成したシステムであり，論理的にはすべてのアドレス空間を通常のロード，ストア命令でアクセスできる．ただし，物理的には分散配置されているので，自プロセッサに直接接続されたメモリへのアクセスは高速だが，他のプロセッサに接続されたメモリへのアクセスは低速となる．CC-NUMA（Cache-Coherent Non-Uniform Memory Access）と呼ばれる，キャッシュの一貫性制御に関するサポートを持つ DSM はその例である．DSM には，分散共有のためのハードウェアのサポートを持つものと，ハードウェア的には分散メモリシステムで，この上に純粋にソフトウェアだけで DSM を実現するソフトウェア DSM とがある．

3. 2. 3　最近のスーパーコンピュータシステムの実例

1980年代後半から1990年代前半にかけて，Thinking Machines（CM5）や nCUBE, FPS, Cenju, AP1000など，分散メモリを採用したスカラプロセッサベースの高並列システム（Massively Parallel Processors; MPP）が多く開発・商用化された．また，富士通 VPP シリーズや日立の SR シリーズのような，ベクトル型のプロセッサを高速ネットワークで結合したスーパーコンピュータも現れた．

しかし最近のハイパフォーマンスコンピューティングシステムのトレンドは，共有メモリ型である SMP を多数台ネットワークで結合し分散型構成とした階層型の並列システムであるSMPクラスタである．ハイエンドのスーパーコンピュータでは超高速のネットワークが用いられる一方，SMP を Gigabit Ethernet などの汎用の LAN（Local Area Network）技術で結合した構成もコストパフォーマンスを重視したシステムで多く採用されている．以下に，SMP クラスタ型のスーパーコンピュータシステムの実例として，地球シミュレータおよび京都大学学術情報メディアセンターのスーパーコンピュータシステムをあげる．

地球シミュレータ

2002年 3 月より海洋科学技術センター地球シミュレータセンターで運用されている地球シミュレータ（Earth Simulator）は，640台の計算ノードを，単段クロスバの結合ネットワーク（双方向 12.3 GB／秒）で結合した分散メモリ型並列計算機システムである．各計算ノードは，ピーク演算性能8 GFlopsのベクトル型プロセッサ 8 台が 16 GB の主記憶を共有する SMP（Symmetric Multi Processor）構成を取る．総ピーク演算性能が 40 TFlops，総主記憶容量は 10 TB である．Linpack ベンチマークによる実効演算性能値で 35.61 TFlops を記録している．

京都大学学術情報メディアセンター　スーパーコンピュータ

2004年 3 月に京都大学学術情報メディアセンターに導入された富士通 PRIMEPOWER HPC2500からなるシステムは，スカラ型プロセッサ128台からなる SMP として構成される計算ノード11台と，同64台からなるSMPで外部ディスクへの I/O 処理を行うための I/O ノード 1 台とを，単段クロスバの高速光インターコネクト装置（双方向16 GB／秒）を用いて結合した構成である．各プロセッサの演算性能は 6.24 GFlops，計算ノード単体のピーク演算性能は 798 GFlops 主記憶容量は 512 GB である．計算ノード群全体では，総ピーク演算性能が 8.785 TFlops，総主記憶容量は 5.632 TB である．Linpack ベンチマークによる実効演算性能値として 4.552 TFlops（参考値）を得ている．

このほかに，本総合解説**2. 1**および**2. 2**で紹介されている核融合研の大型シミュレーション解析装置および原研におけるスーパーコンピュータ環境のいずれも，同様に超高速ネットワークで結合された SMP クラスタ構成のスーパーコンピュータシステムである．

一方，インターネットの普及と高速化に伴い，地理的に分散して配置されたスーパーコンピュータシステムを，WAN（Wide Area Network）経由で接続し，大規模な疎結

合システムとしてもちいることも考えられている．その典型が，グリッドコンピューティング（Grid Computing）と呼ばれる技術である．Grid の語は電力網に由来する．すなわち電力の消費者はその電力がどこで発電されたものかを全く意識せずに使うことができるのと同様，ネットワーク上のコンピュータのリソースをインターネットを用いて統合し，計算需要を持つ者が，あたかも手近に大きなコンピュータがあるかのごとく必要な計算を行うことができる環境を実現しようとするものである．グリッドコンピューティングの基盤となるのは，地理的にも離れ異なる組織に属してそれぞれのポリシーで運用されているコンピュータを，安全に相互開放し，組織間の壁を意識することなく利用できるようにするためのソフトウェア技術であり，古典的な分散処理技術，分散資源管理技術と VPN（Virtual Private Network）や PKI（Public Key Infrastracture）などのセキュリティ技術を統合したものである．グリッドコンピューティングに関する標準化は GGF（Global Grid Forum; http://www.gridforum.org）を中心に行われており，今後の動向が注目されている．

以上に述べた並列コンピュータシステムのトレンドを，後述の並列処理ソフトウェアとの関係と合わせて Fig. 1 に示す．

3.3 並列処理ソフトウェアの動向

3.3.1 共有メモリ向け並列化技術

自動ベクトル化と自動並列化は，データ依存関係解析という意味ではほとんど同様であり，リダクションなどのプログラム中の特定構造（イディオム）の認識，ループ融合・分割・入れ替えなどのループリストラクチャリング技術，ループ長などの実行時パラメータに基づく最適化切り替え，種々の指示行の導入によるユーザによる最適化促進技術など，必要な技術は基礎理論としては1980年代後半までにイリノイ大学などの研究活動によって完成している．

しかし，ベクトル化が局所的なループネストだけを対象に最適化できるのに対し，並列化はできるだけ並列処理単位を大きくして並列処理や同期などのオーバヘッドを減らす必要があり，手続きをまたがる最適化が必須となるのが困難な点である．手続き間解析も，技術的には完成の域に達し，Convex などいくつかの商用コンパイラで実装されたが，手続き間の解析時間が実用の大規模なプログラムになると膨大になることや，手続きごとに別ファイルして分割コンパイルする環境で，オブジェクトの管理がややこしくなるなどの理由で日の目を見なかった．一方で，手続きをインライン展開する手法は，展開すべき手続きをユーザが指定するなどの方法で，簡便に用いることができ，また効果も大きいことから，現在でも広く用いられている．

自動並列化のもうひとつの問題は，ベクトル化に比べて最適化の選択肢が多いことである．多重ループの並列化に関してだけでも，ループ入れ替えやアンローリング，並列化すべきループの選択などが複雑に絡み合い，選択によって性能が大幅に変動するケースが多く，選択を誤ると性能が極端に劣化する問題がある．このため，並列化で最初にユーザに普及したのは，CRAY-XMP や NEC の SX-3などでサポートされた，マイクロタスキングおよびマクロタスキングとよばれるユーザが明示的に並列化の方法を指定するためのインタフェースである[8]．

マクロタスキングとはスレッド並列処理の枠組みであり，手続きを非同期に呼び出し，呼び出された手続きを別々のプロセッサで実行することで並列処理を実現する．呼び出された手続き間の同期処理や排他制御は，イベントやロックという特別な変数を用いたライブラリ呼び出しで実現されている．UNIX におけるスレッド並列処理記述インターフェースである pthread ライブラリも，これと同じ考え方である．

マイクロタスキングとは，プログラム中の主にループに対して並列化指示文を挿入するインタフェースであり，ワークシェアリングといって，並列化対象以外の部分は，全プロセッサで重複して実行されるというプログラミングモデルを採用している．自動並列化（autotasking）との相違点は，コンパイラが勝手に並列化を判断することが一切ないという点と，ワークシェアリングモデルのため，指示文を無視して逐次実行したものとプログラムの意味が異なり得る点である．OpenMP は，マイクロタスキングの考え方をもとに共有メモリ用並列化指示行インタフェースの標準として設計されたもので，ループの各繰り返しや計算処理のまとまりを単位とし，それらを parallel region として

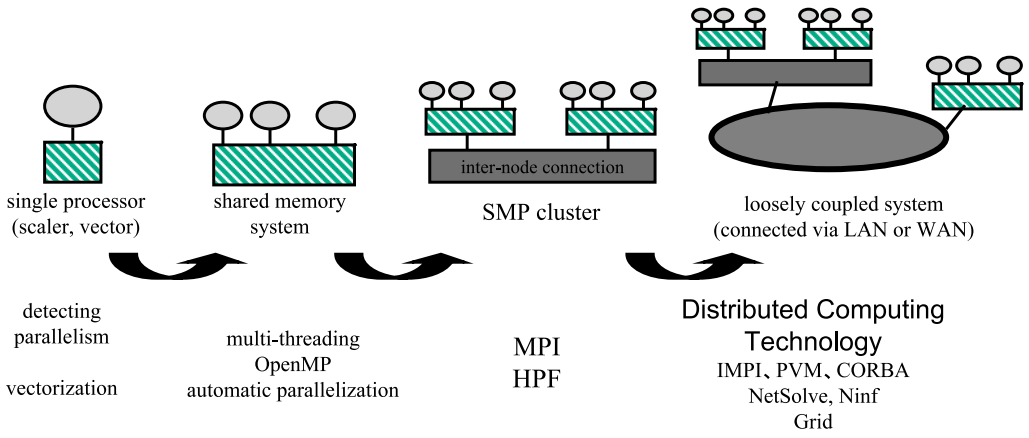

Fig. 1 Trend of parallel computing systems.

別々のプロセッサで実行させることをユーザに明示的に指示させる，fork-joinモデルを採用している．たとえば，Fortranでの DO ループを並列化する場合，ループを!$OMP OMP DO 〜 !$OMP END DO で囲むことにより，各繰り返しを並列実行させることができる．

3.3.2　分散メモリ向け並列化技術

分散メモリシステム上の並列化における最も重要な2つのポイントは，データ分割（処理対象データの分散メモリ上への配置）と，計算マッピング（計算処理の各要素プロセッサへの分配）である．並列化の良し悪しは，並列性の抽出のみならず，いかにデータアクセスの局所性を高め，プロセッサ間通信を減らすかに大きく依存する．プログラマの立場からすると，データ分割と計算マッピングの両方を指定するというのは大きな負担になる．これが分散メモリマシン上での並列化の難しさの本質なのであるが，データ分割を主に考えてプログラミングする場合には，決めたデータ分割に合わせるように計算マッピングを指定する必要があるし，逆に，計算マッピングを主に考えて並列化を決めた場合は，これに合わせたデータ分割およびデータ転送処理の記述が必要となる．

分散メモリシステムでの並列プログラムの実行は，各プロセッサのための独立したプログラムが通信を行いながら同時に動作することで成り立つ．プログラムにおいては，各プロセッサの実行時のデータ配置と計算を記述し，プロセッサ間でデータの授受に矛盾が生じないように同期を取る制御が必要である．このようなプログラミングの負担を軽減するための技術として，分散メモリ向けの計算機言語では，大域的名前空間（global name space）と単一ストリーム（single stream）の考え方を多かれ少なかれ導入し実装している．大域的名前空間とは，変数Aと言えばどのプロセッサでも論理的に同じ実体を指すことを意味し，利用者は「どのプロセッサ上のAなのか」を識別し指定する必要はない．単一ストリームとは，すべてのプロセッサにおいてプログラムが論理的には同期して実行されることを意味し，利用者は，プロセッサ間でのデータの定義・参照の順序のずれや，重複して配置されているデータの値の一致の問題に煩わされることなく，逐次プログラムと同じ感覚で実行文の列を記述することができる．

このデータ分割と計算マッピングをユーザがプログラム上でどのように記述するか，また，大域的名前空間と単一ストリームの考え方をどこまで実現できているかによって，並列プログラミングインタフェースを類型化することができる．

MPIなどのメッセージパッシングプログラムはSPMD（Single Program Multiple Data Stream）と呼ばれ，各プロセッサで実行されているプログラムは共通であるが，並列実行に参加するすべてのプロセッサがそれぞれ独自の制御を持つ．SPMDでは，IF文などによるプログラムの実行経路がプロセッサごとに異なることをユーザが意識してプログラムする必要がある．また，メッセージパッシングプログラムでは，プログラム上で宣言されたデータは，すべてのプロセッサが独自のコピーを持つことが前提であり，プロセッサ1上の配列Aとプロセッサ2上の配列Aは別々のデータとして扱われる．

このようにMPI（Message Passing Interface）は，データ分割と計算マッピングをすべて利用者が指定するインタフェースであり，大域的名前空間や単一ストリームの考えは入っていない．一方，商用コンパイラとしては実在しないが，逐次プログラムをそのまま並列プログラムとして処理する理想的な完全自動並列化コンパイラというものがあれば，これはデータ分割と計算マッピングについて何も指定しないというインタフェースであり，大域的名前空間や単一ストリームの考えが完全に実現されている．現実の並列プログラミング言語では，データ分割と計算マッピングのどちらか片方を指定するようにしたり，大域的名前空間や単一ストリームを部分的・限定的に実現していたりすることで，利用者の負担を軽減しようとしている．たとえばHPFなどのデータパラレル言語は，データ分割をユーザが明示的に指示し，それ以外の仕事を処理系（コンパイラ）に任せようというものである．逆に，計算マッピングだけを指示するのが，共有メモリ用システム用のインタフェースとして発展したOpenMPなどの言語である．

次の**3.4**節では，分散メモリ向け並列プログラミングインターフェースについて実例をあげ，以上のような観点から分類を試みる．

3.4　分散メモリ向け並列プログラミングインターフェース

3.4.1　スレッド並列化

アーキテクチャとして分散メモリのシステムに，ハードウェアまたはソフトウェアによる分散共有メモリ（DSM）を実装し，この上で共有メモリ用の並列化インタフェースであるOpenMPを用いることにより，分散メモリシステムのプログラミングインタフェースを実現することができる．

OpenMPは共有メモリが対象であるため，データの分散メモリ上への配置については，ユーザは指定できない．ある計算処理とそれに用いられるデータが同じプロセッサとメモリにあれば高速処理ができるが，これらを全く考慮しなければ，大半の計算処理でのメモリアクセスがネットワーク経由となり効率低下を招く[9]．このための制御（Affinity制御と呼ばれる）をコンパイラ，ユーザでどのように分担するか，またその際の言語インタフェースはどうすべきかが重要である．OpenMPの発展として，配列のDSMへのデータマッピングを指定する拡張や，First Touchと呼ばれる，最初にアクセスしたプロセッサメモリにデータを配置するような制御をコンパイラで行う方法が提案され，一部のコンパイラには実装されている[10, 11]．

3.4.2　データパラレル

データパラレル言語とは，大規模シミュレーション計算に典型的な，巨大配列の各要素に同じ処理を行う計算の並列性に着目して考案された言語インタフェースである．商用システムとしては，コネクションマシンに実装されたCM-Fortranが最初である．

データマッピングについては配列の分散指定を利用者が明示的に指定する一方，計算マッピングについては，更新されるデータを保持するプロセッサが処理するいわゆるOwner Computes Ruleにより，ループ並列化や必要となるデータ転送の生成をコンパイラが自動的に行う．

データパラレル言語の代表は，HPFである．HPF（High Performance Fortran）は，その名のとおり高性能コンピューティング用に開発されたプログラミング言語である．Fortran言語に最小限の指示文を付加することにより，分散メモリ並列システムで簡単に高い性能を得ることを目指している．

HPFの本質的な考え方は，データ分割をユーザが明示的に指示し，それ以外の仕事を処理系（コンパイラ）に任せようというものである．これにより，ユーザは煩わしいプログラムのSPMD化や通信管理から解放され，従来のFortranプログラミングと非常に親和性の高い方法で分散メモリシステムを利用することが可能となる．HPFでは，プログラムは従来の逐次言語と同様に単一ストリームのセマンティックスで記述され，データはすべて大域名前空間で扱われる．

HPFの仕様は，HPFF（High Performance Fortran Forum）によって検討され，1993年HPF1.0が定められ，コンパイラの開発が進められた．さらに仕様が大きく，コンパイラの実現が困難であるという反省のもとに，1997年にHPF2.0で基本および公認拡張という形で仕様が整理された．国内では，1997年にJAHPFが発足し，1999年にはHPF/JA1.0という拡張仕様がまとめられた[12]．HPF/JA仕様は，並列記述性の向上および適用範囲の拡大とコンパイラの解析能力の限界を補うことを目的として，次項で述べる明示的データパラレルに近い，利用者によるきめ細かい並列化および最適化記述を可能にしたものである（Fig. 2）．

3.4.3 明示的データパラレル

スレッド並列の考え方にデータ並列の考え方を加え，データ転送を明示に指定するようにする「スレッド並列＋データ並列」の例として，VPP Fortran[13]やCo-Array Fortran（CAF）[14]があげられる．

VPP Fortranは，もともとNWT（Numerical Wind Tunnel）をターゲットとして開発された言語であり，日本におけるデータ並列言語としてはもっとも成功したものである．指示行形式を中心にFortranに最小限の拡張を加え，データマッピングと計算マッピングの両方を利用者に指定させる．Co-Array Fortranは，CRAYが開発したデータパラレル言語で，Fortran95にデータマッピングを記述するための簡単な言語拡張が施されている．

いずれもSPMDモデルに基づき，計算実行の並列化指定はOpenMPなどとほぼ同じである．データマッピングについては，プロセッサごとに重複して領域が用意され高速なアクセスが可能なローカルデータと，分散して配置されプロセッサをまたがってのアクセスが可能だがアクセスが低速なグローバルデータの2種類が用意される．CAFでは

```
REAL, DIMENSION(N)[*] :: X,Y
X(1) = Y(1)[Q]
```

のように配列参照に[ ]指定を追加する言語拡張がなされており，これを通じて他のプロセッサのデータを参照するグローバルアクセスと，自プロセッサのデータを参照するローカルアクセスを区別することができる．VPP Fortranでは，グローバル変数とローカル変数を別々に宣言し，この両者をequivalence文で結合することにより，変数名でグローバルアクセスとローカルアクセスを書き分けることができる．

言語拡張のそれぞれの要素を見ると，前述のHPFと非常によく似た構成になっているが，言語の設計コンセプトはかなり異なる．もっとも大きな相違は，HPFがデータマッピングのみを指定させ，計算マッピングをコンパイラの自動処理に期待しているのに対して，VPP FortranやCAFでは，コンパイラの自動処理に期待するものは非常に少ない．データマッピングも計算マッピングも，性能に影響を与えるものはすべて利用者に明示的に指定させ，コンパイラにできるだけ負担をかけずに，高い性能が得られるようにすることをめざして設計されている．

3.4.4 共有アレイ

GA（Global Arrays）[15]は，ライブラリ呼び出しの形で共有メモリビューを提供するインタフェースであり，FortranやC，C++などの言語と一緒に用いることができる．MPI処理系などのメッセージパッシングインタフェー

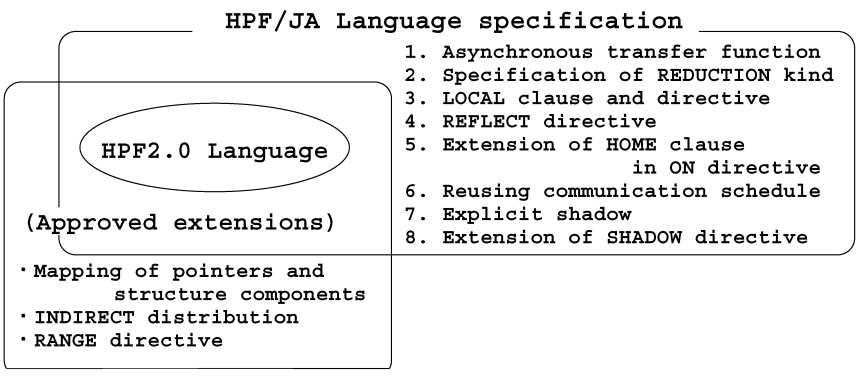

Fig. 2 Specifications of HPF.

スの上にも実装されており，この場合，メッセージパッシングインタフェースと混在して用いることができる．

基本的な機能は，グローバル配列（GA）の宣言，およびGAへの一方向通信によるアクセス（putとget），データ一致制御のための同期，総和などの上位機能，GAの格納アドレスのなどの情報取得機能である．機能的には，先述のCo-Array Fortranと非常に似ているが，これらのGA操作がすべてライブラリ呼び出しの形式で実装されている点が異なる．メッセージパッシング関数が，プロセッサ識別子（MPIではrank）とアドレスの組によりデータ転送の対象を指定するのに対し，GAでは，配列要素番号によるアクセスが可能であり，明示的にデータ格納先のプロセッサを意識する必要はない．具体的には，

```
nga_create(type, ndim, dims, array_name,
chunk, g_a)
```

で，n次元のGAを宣言でき，これに対して

```
subroutine NGA_Put(g_a, lo, hi, buf, ld)
subroutine NGA_Get(g_a, lo, hi, buf, ld)
```

でput/get操作が可能である．

同期用のルーチンも用意されており，データ一致制御の必要に応じて挿入する必要がある．また，総和などのcollective通信を行うルーチンも用意されている．

3.4.5　メッセージパッシング

メッセージパッシングは，プロセッサ間のデータ転送を，ユーザがライブラリを用いて明示的に記述する枠組である．SPMD形式に基づいて，複数の制御の流れを意識したプログラミングが必要になる．

MPI（Message Passing Interface）は，分散メモリ型のプログラミングインタフェースとして，現在，最も標準的に用いられているものである．仕様はMPIF（MPI Forum）と呼ばれるプライベートなフォーラムで検討され，1994年6月にMPI1.0がまとめられた．その後，並列I/Oやプロセスの動的生成・消滅，1方向通信（One Sided Communication）などの機能をさらに追加したMPI2.0の言語仕様が1997年7月に定められた．

MPIは，現在，大半の商用並列マシン上でサポートされている．これらのMPIにはアルゴンヌ国立研究所とミシシッピ州立大学との共同で開発されたMPICH（http://www.mcs.anl.gov/mpi/mpich/参照）をもとにしたものが多い．MPICHは仮想デバイスの概念を導入することにより，システム依存部がコンパクトに切り離された構造になっており，移植性が高い．ベンダは，システム依存部を対象ハードウェアに適するように開発することで，効率の良いMPIを実装することができる．

3.5　並列化プログラミングインタフェースの比較

前節で説明した種々の並列化インタフェースをプログラミングの容易さと性能の関係で比較すると，Fig. 3のようになる．大きな相違点としては，下記のものがあげられる．

(1)　並列化インタフェースの形式

並列化インターフェースの形式として，言語拡張によるもの，指示行追加によるもの，ライブラリ呼び出しがある．最も自由度が高いのが，並列プログラミングをサポートするよう言語そのものに拡張を加えたり，新しい言語を設計する方法である．しかし言語処理系の開発コストは最も高い．既存の言語ではコメントとして解釈される指示行により拡張を加える方式では，コンパイラに手を入れる必要がある点では言語拡張と同じだが，既存言語と指示行言語の処理（たとえば構文解析処理など）を比較的切り分けられるため，処理系の開発コストが下がる．反面，行単位の指示行での記述性には限界がある．ライブラリ呼び出しは，コンパイラに手をいれずに済むため，もっとも手軽な実装が可能だが，ライブラリ呼び出しのオーバヘッドが必要になったり，ライブラリ化することで，コンパイラの最適化が阻害されたりする場合がある．

(2)　並列化指定

計算マッピングとデータ分散指定をどのように切り分けるかも重要なポイントである．分散メモリシステムの分類

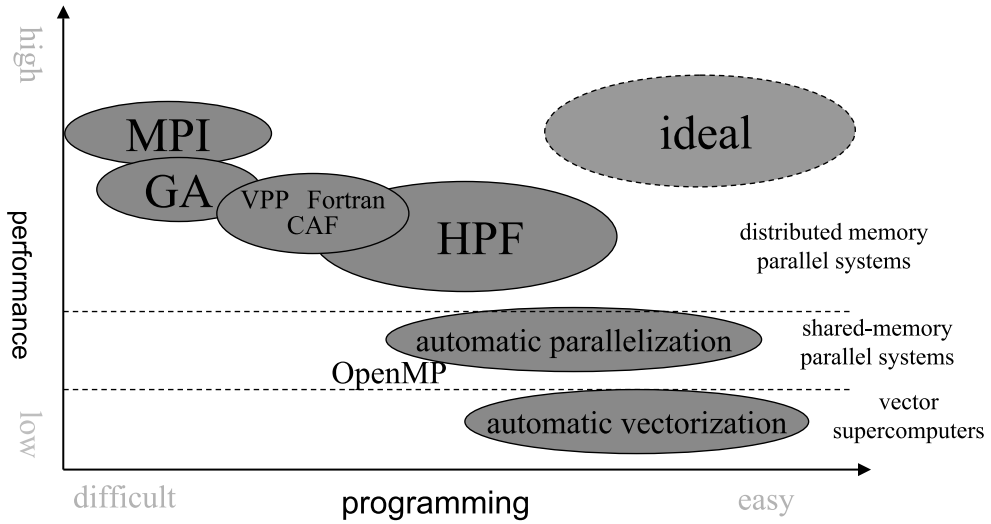

Fig. 3　Comparison of parallel programming interfaces.

においては，言語機能そのものという意味で最も大きな違いが出る部分である．HPF の採用した，利用者にデータ分散指定をさせ計算マッピングは処理系が責任を持つという考え方は，シンプルでわかりやすい切り分けである．

課題は，手続き間のデータマッピングの扱いにある．データマッピング指定は，コンパイラなどの処理系が配列の論理イメージと物理的アドレス上のイメージのマッピングを管理する必要があるため，Fortran90のインターフェースブロック仕様を用いてきれいに書かれたコードでない限り，手続き間の扱いに限界がある．特に，Fortran の参照呼び（call by reference）を用いて最適化のために手続き間で異なる形状の配列の受け渡しがされているようなプログラムでは，コンパイラによる解析は不可能である．この点は，GA や Co-Array Fortran，VPP Fortran についても共通である．データ並列言語を普及させるには，手続き間の配列をどのように扱うべきかというプログラミング方法論を確立し，ユーザが物理的メモリイメージを考慮してコーディングするケースを減らしていくしかないかもしれない．

⑶ コンパイラの自動化への期待度

コンパイラの自動化にどこまでを期待するかも分類のための尺度である．理想的な完全自動並列化はそれらすべてを求めるものである．特殊な例を除けば，ライブラリ呼び出し形式の MPI などは，コンパイラに何も期待しないし，VPP Fortran や GA や Co-Array Fortran も，コンパイラにほとんど期待しない形式になっている．最近，HPF と MPI の中庸を求める動きの中で，これらのインタフェースが再度注目を浴びている．

⑷ インタフェースの美しさ

インタフェースの完備性，仕様の美しさもインタフェースのよしあしを決める上で大きな要素となる．シンプルな美を持つ仕様がもっとも望ましいが，特定ケースの対応ができず，実用上問題がでる場合が多い．かといって，実用性を追求していくと，インタフェースの肥大化や複雑化を招く．特定機能を追加することで，思いもかけなかった副作用が他の機能に発生することも，よくある．HPF では指示行追加の方式を取りつつ構文としては Fortran90言語の拡張として扱えるように設計されている．そのため言語仕様は美しいが，コンパイラの開発には多大なコストを要した．ある意味で，HPF は言語の完全性，美しさを追求しすぎたきらいがある．その点，VPP Fortran は，完備製や美しさをある程度犠牲にして，実用性を追求した言語であると言える．

3.6 展望

かつて，いずれは Fortran の後継とも目されていた HPF は，真に実用に耐え得るコンパイラの登場が当初の予想より 3 年から 5 年遅れ，利用者に最も大きな期待を寄せられた1990年代後半に実用レベルのコンパイラを提供することができなかったことが最大の原因で，残念ながら，今や欧米では忘れ去られようとしているとも言われている．機を逸したことで，地球シミュレータ用の HPF/ES を始め高い完成度のコンパイラが実用で使えるようになったとはいえ[16]，現状の HPF が，これから爆発的に利用されるようになるということは考えにくい．

HPF をさらに進化させて，データの分散配置もコンパイラが自動的に決定できるようになれば，完全自動並列化が実現される．しかし，実用に耐える性能をもつそのようなコンパイラが登場することは，少なくとも近い将来のうちには期待できそうにない．分散メモリのコンピュータにおいて，利用者にデータの分散配置を指定させ，計算マッピングやデータ転送など残りをコンパイラが自動的に処理するというデータパラレルの基本的考え方は正しい方向であると考えている．

並列化インタフェースとしては，当面は MPI が中心となることは間違いないであろう．だが，MPI 同様すべてを利用者が指定するという立場を取り，また SPMD のプログラミングモデルでありながら，データパラレルや大域名前空間の考え方を部分的に導入して記述量の削減を狙う，VPP Fortran や Co-Array Fortran，GA などは，インタフェースがシンプルで利用者の習得が容易なことから，今後の普及が期待できる．また，OpenMP にデータマッピング機能を付加するアプローチも同じ方向性である．いずれも，MPI に比べて，とりあえず動く並列コードを作成するところまでが非常に容易である点を評価したい．これらを通じてデータパラレルの良さが理解されてくるようになれば，HPF でコード開発・並列化を加速できるユーザを地道に増やしていくことができるのではないかと考えている．

HPF コンパイラのさらなる性能・機能向上のためには，さまざまな実用アプリケーションの HPF 化促進と実環境での HPF の評価が必須である．そのために，JAHPF を発展的に改組して2001年に設立された HPF 推進協議会（HPFPC，http:///www.hpfpc.org）において，利用者アプリケーションの HPF 化支援やフリーの PC クラスタ用処理系の開発[17]，ベンチマーク用プログラムの開発[18]，ドキュメント整備や講習会などの活動を続けている．

現世代の商用スーパーコンピュータのアーキテクチャは SMP クラスタ型に収束してきているが，次世代においてさらにプロセッサ数が増大することになれば，プロセッサ間のネットワークが階層化されたものとなろう．そのような階層的な分散メモリに対する直接的なサポートは HPF を含め現在のデータパラレル言語では考えられていない．グリッドコンピューティングなどで典型的なヘテロジニアスな並列環境への対応も，発展課題である．

HPF の今後についての最大の懸案は，言語やコンパイラの技術的課題よりも，Fortran で並列プログラム開発を行うというスタイルが今後どれだけ広がるかにあると思われる．新規のコード開発において Fortran の利用割合が減っていること，並列コード開発が，スクラッチからの開発ではなく，既存の MPI プログラムを改造していくケースが多いことなどがマイナス要因である．今後の HPF，あるいはその後継となるデータパラレル言語では，こういった dusty deck と呼ばれる古い Fortran プログラムや MPI プログラムとの結合性（interoperability）を維持する努力も必要

であろう．並行して，並列化により利用者の研究自体が大いに加速されるような逐次コードをできるだけ多く発掘して，HPF を用いて並列化することでその有効性を地道に利用者にアピールしていく活動も継続していきたい．

参考文献

［1］High Performance Fortran Forum. 1997. High Performance Fortran Language Specification. Version 2.0. Technical Report TR, Rice University, January 1997.

［2］H. Sakagami, H. Murai, Y. Seo and M. Yokokawa, 14.9 TFLOPS Three-dimensional Fluid Simulation for Fusion Science with HPF on the Earth Simulator, SC2002, http://sc-2002.org/abstracts_papers/ab_paper17.html.

［3］Message Passing Interface Forum. MPI: A Message-Passing Interface Standard, Int. J. Supercomput. Appl. High Perform. Comput. **8**, No. 3/4, 165 (1994).

［4］OpenMP Architecture Review Board, OpenMP Fortran Application Program Interface, Version 2.0, November 2000 (http://www.openmp.org).

［5］D. Lenoski *et. al.*, The Stanford Dash Multiprocessor, IEEE Computer, **25**(3), p.63 (March 1992).

［6］J. Laudon and D. Lenoski, The SGI Origin ccNUMA Highly Scalable Server, SGI White Paper (March 1997).

［7］細見岳生他；情処論文誌 **41**, No.05-17（2000 年 4 月）.

［8］http://www.nas.nasa.gov/Groups/HSP/UserGuide/

［9］K. Kusano, S. Satoh and M. Sato, Performance Evaluation of the Omni OpenMP Compiler, WOMPEI 2000, Oct. 2000, Tokyo. http://pdplab.trc.rwcp.or.jp/Omni/home.ja.html.

［10］B. Chapman, HPF Features for Locality Control on ccNUMA Architectures, HUG2000 Invited Presentation, Tokyo, Oct. 2000 (http://rist03.tokyo.rist.or.jp/jahpf/hug2000/presen/Chapman.pdf).

［11］廣岡孝志，太田　寛，菊池純男：情処論文誌 **41**, No.05-020（2000 年 4 月）.

［12］Japan Association of High Performance Fortran, 1999, HPF/JA Language Specification Version 1.0, RIST (English version is available at http://www.tokyo.rist.or.jp/jahpf/index-e.html).

［13］岩下英俊，HPF からみた VPP Fortran，情報処理学会誌 **38**, 2, p.114（1997年 2 月）.

［14］http://www.co-array.org/welcome.htm

［15］http://www.emsl.pnl.gov:2080/docs/global/ga.html

［16］H. Murai *et al.*, Implementation and Evaluation of an HPF Compiler for Vector Parallel Machines, HPF/SX V2, 4th HPF User Group Meeting, Oct. 2000, Tokyo.

［17］岩下英俊：HPF から MPI へのトランスレータ fhpf，第75回京都大学学術情報メディアセンター研究セミナー報告集（2004年 3 月）.

［18］村井　均，岡部寿男：地球シミュレータ上の HPF による NAS Parallel Benchmarks の実装と評価，SACSIS 2004－先進的計算基盤システムシンポジウム（2004年 5 月）.