解 説 HPF言語の動向

6. HPFからみたVPP Fortran

A View of VPP Fortran in Contrast with HPF by Hidetoshi IWASHITA (Fujitsu Limited).

岩下英俊[1]

1 富士通(株)HPC本部第二開発統括部ソフトウェア開発部

1. はじめに

並列言語は，実用的であってほしい．

実用的であるためには，性能と使いやすさの両面が必要である．並列計算機のハードウェア性能を十分に引き出せなければ，実用的な言語とはいえない．しかし性能を出すだけならメッセージパッシングライブラリを使うこともできる．使いやすくなければ，並列言語を選ぶ理由がない．

VPP Fortranは，実用的であることを目指して1991年に作られた言語仕様である．設計のよりどころとしたのは実プログラムであった．実際のアプリケーションプログラムの研究によって，処理系として実際に必要な機能に仕様を絞り込んだ．現在，VPP Fortranは大規模な科学技術計算の最前線で使用されている．

一方で現在，HPFの仕様が固まりつつある．HPFは標準化を狙った言語であるため，研究者，ベンダ，ユーザ，それぞれの立場から意見を持ち寄って討論を重ねて，広く受入れられる言語仕様を目指している．

この2つの言語は，どちらもFortran言語の並列化拡張である．また，データを分割してプロセッサに配置し，それに合わせてループ計算を並列化することなど，基本的な部分で非常に似た考え方をもっている．しかし深く考察してみると，さまざまな相違点があることが分かってきた．みえてきたのは表現方法の違いだけではなく，設計時に想定するユーザやコンパイラ技術の違いが生み出した考え方の違いであった．興味深い結果を得たので，ここに紹介したい．

HPFの特徴と概要については，本特集の1編と2編を参照していただきたい．本編では，2章でVPP Fortranの概要を紹介し，3章でHPFとの比較を行い，考え方の違いを考察する．4章では，VPP Fortranの記述力を評価する．HPFの考え方を深く理解する上で，VPP Fortranとの比較を役立てていただければ幸いである．

2. VPP Fortranの概要

VPP Fortranは，1989年から1991年にかけて設計された言語である．当時の処理系の技術では並列に関する最適化はほとんど未成熟であり，分散メモリ型の計算機を対象にする言語は標準といえるものがなかったので，独自の開発となった[3]．

2.1 VPP Fortranが目指したもの

VPP Fortranは，標準のFORTRAN 77をベースとして，分散メモリ型並列計算機上での並列実行のための機能を付加した言語である．設計にあたって最も重視したことは以下の2点であった．

1) プログラミングが容易であること．とくに逐次プログラムからの移植を容易にすること．
2) ハードウェア性能を十分に引き出すこと．

2.1.1 プログラミングの容易さ

1)を目的として，VPP Fortranの言語仕様では次の3つを実現している．

1a) 自動データ転送機能の実現により仮想的なグローバルメモリ空間を提供する．
1b) 並列化のための書式はコンパイラディレクティブとして設計する．
1c) 標準のFortranで書かれたサブプログラムは，そのままVPP Fortranで書かれた並列プログラムと結合できるようにする．

1a)は，分散メモリや通信を意識しない，楽なプログラミングを可能にするための機能である．1b)は，並列化のための書式をコメント行とみな

してコンパイルしたときの逐次実行と同じ結果を保証している☆．これはプログラムのデバッグに役立つだけでなく，プログラムの理解も助けている．1 b)，1 c)にみられるように，VPP Fortranでは逐次プログラムとの親和性を重要視している．

これらの考え方は，基本的にHPFと共通している．1 a)について，HPFではより徹底した考え方である．すべての変数はすべてのプロセッサで共有されていて，一意性は完全に処理系の責任で保証している．1 c)については，HPFではEXTRINSIC宣言の1機能としてサポートしている．

2.1.2 性能が出せること

2)を目的として，次の2つを提供することにより，ユーザの手によるプログラムの性能改善を可能にしている．

2a) データ転送を直接記述するための書式

2b) グローバル/ローカル二階層のデータ配置属性

プロセッサ間で共有されるデータのアクセスは，通常コンパイラによって自動的にプロセッサ間のデータ転送に変換されるが，一括転送や非同期転送を目的として2 a)によりユーザが明示することもできる．また2 b)により，プロセッサ内に閉じたアクセスの行われる変数はローカルであると宣言できる．ローカルに配置された変数は，高速にアクセスされることが保証される．

これらは，HPFにはみられない低水準の機能であるが，実用性を重視するVPP Fortranでは重要な機能である．このような記述力により，VPP Fortranでは，通信ライブラリを直接使用した記述に匹敵する性能を達成することができる．

2.1.3 段階的なプログラミング

1)と2)は相反する要件であるが，VPP Fortranではこれらを両立するため，記述の程度をユーザが任意に選べる設計としている．並列プログラミングの最初の段階では，基本的な並列化ディレクティブだけを追加すれば並列実行を確認することができる．それだけでは十分な性能が得られない場合には，プログラム中の性能的にクリティカルな部分を探し出し，データ転送などの指示を加えて，徐々に性能改善を行うことができる．実用的なシステムとして重要なのは，いざ必要となった場合に，どんな手段でもよいから性能が出せることであろう．VPP Fortranでは，必要ならユーザが書いて性能を出す手段を残している．

HPFでは，低水準すぎる仕様を除外し，コンパイラの自動並列化・最適化機能をある程度前提にして設計されている．VPP Fortranは，現在実際に使用されている言語であり，HPFの1世代前の言語と位置づけられるだろう．

2.2 言語仕様の特徴

ここでは，VPP Fortran仕様の大きな特徴を紹介する．

VPP Fortranでは，プログラムの実行状態とハードウェア構成をモデル化してユーザに提供している．また，配列の分割配置とループ手続きの分割配置を統一的に表現する手段として，インデックスパーティションと呼ぶ独自の概念をもっている．

2.2.1 スプレッド-バリア方式

数値計算アルゴリズムの並列化は，ループ手続きの分割実行が中心となる．また，大規模な並列処理では，プロセッサ群をサブグループに分けてそれぞれで実行を行う，いわゆるMIMD (Multiple Instruction Multiple Data) モデルによる並列化を併用することも効果が期待できる．VPP Fortranではこれらを併せもつ実行モデルを**スプレッド-バリア実行方式**と呼んでユーザに提供している．

図-1を用いてこの実行モデルを説明する．並列実行の範囲はPARALLEL REGION構文で指定された区間である．SPREAD構文は，プロセッサをいくつかのグループに分割し，タスクをグループごとのタスクに分割する．SPREAD構文には，MIMD並列を記述するSPREAD REGION構文（図-1のCとD）と，SPMD (Single Program Multiple Data) 並列を記述するSPREAD DO構文（$F_1, \cdots, F_N$）がある．

VPP Fortranでは，プロセッサの実行状態を，**並列実行**（C-D間やF_i-$F_{j\neq i}$間），**冗長実行**（B内の8プロセッサやC内の2プロセッサ），**逐次実行**（AとH）の3つに分類する．並列実行の関係にあるプロセッサ間では，post/waitなどの

☆ post/wait同期などの逐次実行が保証できない機能は，ディレクティブでなくサービス手続きとして提供している．

明示的な通信を記述しない限り独立に実行が進む．冗長実行の関係にあるプロセッサ間では，処理系が必要な通信や同期を挿入して，あたかも1台で実行しているような見え方をユーザに提供する．

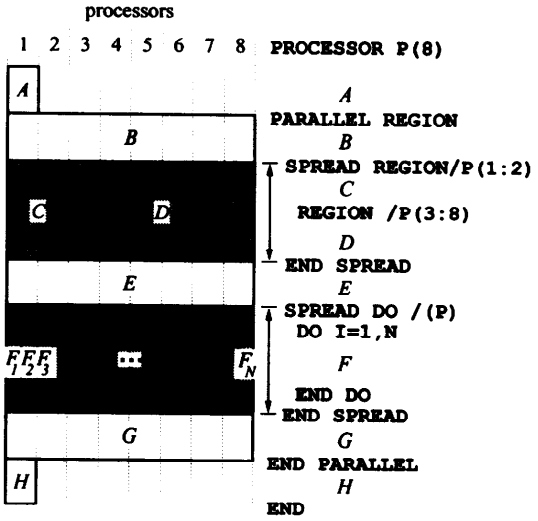

図-1 スプレッド-バリア実行方式

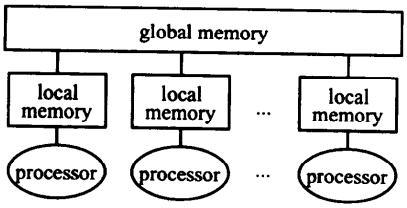

図-2 VPP Fortran のメモリ空間

2.2.2 2階層メモリモデル

VPP Fortran では，プログラミングの容易さと高性能の達成の両立のため，グローバルとローカルの2階層のメモリ空間モデルをユーザに提供する（図-2）．

グローバルメモリは論理的にプロセッサ間で共有されている．グローバルデータのアクセスに必要な通信と同期は処理系が自動生成するため，ユーザはグローバル変数を従来の変数と同様に使用することができる．

ローカルメモリは各プロセッサに固有であり，高速なアクセスが保証される．プロセッサを跨いでローカルメモリにデータを分割配置することもできるが，データ転送を明示しない限り他プロセッサのローカルデータをアクセスすることはできない．

同一のデータをグローバルとローカルの両方の性質でアクセスしたい場合，ユーザはグローバル配列とローカル配列を宣言し，EQUIVALENCE 文で結合することができる．そのようなデータは実際に1つの実体として確保される．

2.2.3 配列とループの分割

分散メモリ型計算機の並列実行では，データ転送をできるだけ少なく抑えるように，データの分割と計算の分割を整合させることが重要である．原理的には，データをもつプロセッサがそのデータに関する計算を行うようにすればよいのであるが（owner computes rule），自動化だけでは必

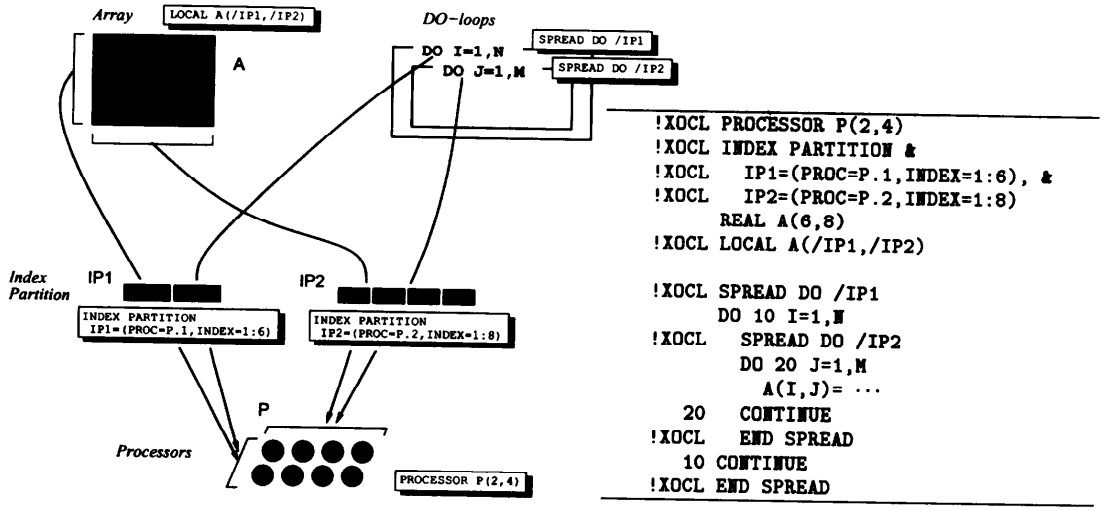

図-3 配列分割とループ分割の例

ずしも高い性能は期待できない．

そこで，VPP Fortranでは，データ分割と計算分割を統一的に分かりやすく表現できるように配慮した．VPP Fortranでは，配列の添字式とループのイテレーションを合わせて**インデックス**という用語で抽象化し，インデックスとプロセッサのマッピングを**インデックスパーティション**(以降，IPと略す）と呼んでいる．ユーザはIPを宣言し，これを配列とループの両方に適用することによって，両者の整合をはかることができる．

IPは，4つの属性をもつ．すなわち，マッピングの定義域であるインデックス範囲（INDEX指定子)，値域であるプロセッサグループのセグメント（PROC指定子）と，マッピング方法(PART指定子：BAND，CYCLIC，または不均等分割）の3つで配列とプロセッサのマッピングを表現し，さらに補助的な属性としてHPFのshadowに相当する袖（OVERLAP指定子）をもつ．

配列やループの多次元分割は，IPを分割次元数だけ組み合わせて表現する．**図-3**は，IPを使って配列Aと二重DOループI，Jをプロセッサグループ Pにマッピングする例である．Pは2×4の2次元形状に仮想的に構成されている．配列Aの1次元目とループIは，Pの1次元目にインデックスパーティションIP1によって分散されており，同様にAの2次元目とループJはIP2によってPの2次元目に分散されている．

3. HPFとの比較

HPF2.0の言語仕様には，approved extensionと呼ばれる拡張部分がある[2]．並列処理の3つの要素であるデータ分割，計算分割，データ転送について，それぞれの言語の表現力を**表-1**にまとめた．

この章では，VPP FortranとHPFが共通にもつ機能について比較を行う．対比してみると，表現だけでなく考え方の違いがあることが分かる．

3.1 分割の表現－Index Partition vs. ALIGN, DISTRIBUTE, ON－

分割する配列の次元数をm，分割先のプロセッサの次元数をnとする．HPFのDISTRIBUTEディレクティブがm次元対n次元のマッピングを表現するのに対して，IPは図-3に示したように1次元対1次元のマッピングを表現するため，m次元対n次元を表現するためにはIPを複数組み合わせて使用する．また，配列間の位置合わせ（alignment）の機能について，HPFではALIGNディレクティブで表現するが，VPP FortranではIPの表現する内容に含まれている．つまり，VPP Fortranでは配列－プロセッサ間のマッピングはIPの組で完全に表現されるのに対し，HPFではalignmentとdistributionの2段階のマッピングで表現される☆☆．

表-1 表現力の比較

		VPP Fortran	HPF 2.0	HPF 2.0+ approved extensions
データ分割	分割配置	○	○*1	○
	動的再分散	—	—	○
計算の分割	ループ分割	○	—*2	○
	タスク分割	○	—	○
データ転送	非同期転送	○	—	—
	一括転送	○	○*3	○*3

○ ユーザ記述により表現可能．
— 処理系に依存．ユーザは記述できない．
*1 袖つきの分割(3.4節)はサポートしていない．
*2 並列化可能であることは記述できるが分割方法は記述できない．
*3 書式でなくライブラリとして提供される(3.5節)．

VPP Fortranでは，ループの分割もまたIPによって表現できる（図-3)．HPF 2.0ではループとプロセッサを対応づける記述はないが，approved extensionにはON節による表現がある．これは，ループと配列，またはループとプロセッサのマッピングを記述するためのものである．ON節によるマッピングは，書式としても意味的にも，alignmentとdistributionによる配列のマッピングとは毛色の異なるものである．

これらの表現の違いは，それぞれの言語が何を重視して設計されたかに起因していると考えられる．HPFでは，配列間のマッピングを重視し，配列とプロセッサの間のマッピングは抽象化する傾向が言語仕様から読み取れる．たとえばap-

☆☆ 多段のalignmentは線形な結合によって1段のalignmentに解くことができるが，distributionの表現だけでalignmentを吸収することはできない．

proved extension にある REDISTRIBUTE ディレクティブの機能は，変数間の alignment を維持しながら複数の変数が同時に分散の変更を行うというものである．また，HPF ではループの分割は自動化をベースに考えているため，ON 節はコンパイラ最適化のアルゴリズムを補助する手段にすぎない．一方，VPP Fortran では，ユーザの手によってコードの性能チューニングを行えなければならないという要請から，配列の分割状態をプリミティブに表現できることが必要であった．たとえば，Spread-DO ループの中で分割ローカル変数 A をアクセスできるかどうかは，Spread-DO に指定した IP と変数 A に指定した IP を見比べるだけで容易に判断できる．

3.2 ループの並列化—SPREAD DO vs. INDEPENDENT—

VPP Fortran の SPREAD DO 構文は，DO ループの並列化の指示であるので，コンパイラは必ず並列実行を行うコードを生成する．それに対し，HPF の INDEPENDENT ディレクティブは，DO ループが並列化可能であるという情報をコンパイラに伝えるためのものであり，実際に並列化するかどうかはコンパイラの判断に任されている．

VPP Fortran と HPF の設計方針の違いがここにも現れている．VPP Fortran では，DO ループの並列化はユーザが指示するのが原則であり，IP を使った書式として提供されている．ここでは必ずユーザの記述したとおりの並列実行が行われる．一方 HPF では，DO ループの並列化は基本的にコンパイラの仕事であるため，コンパイラがユーザに期待するのは指示ではなく情報である．

3.3 配置属性—LOCAL vs. NEW—

2.2.2 項で説明したように，VPP Fortran では変数にはグローバルとローカルの 2 通りがある．HPF で出現する変数は，通常すべて VPP Fortran の用語でいうグローバルであるが，ローカルと考えられる変数も存在する．Independent DO ループ内の NEW 節または REDUCTION 節で指定された変数である☆3．

☆3 これらの変数は論理的にはループのイテレーションごとに固有であるが，インプリメント上はプロセッサごとに固有とするのが自然である．

ローカル変数の見え方の大きな違いは，その有効範囲である．VPP Fortran のグローバル/ローカルは変数の属性としてユーザが宣言するものであり，プログラム実行中に変化することはない．ユーザはローカル変数を積極的に使用することによって，プログラムを効率化することができる．HPF の new 変数はループ内だけで有効であり，ループを終了すると値が不定になると規約されている．HPF では並列実行部分の外側でプロセッサごとに変数の値が異なる状態を許さないためである．

3.4 袖—OVERLAP vs. SHADOW—

次のループで，配列 A，B は 4 プロセッサに block 分割されていて，DO ループもまた block 分割で並列化されているとする．

```
DO  i=2, n-1
  B(i)=0.25*(A(i-1)+2*A(i)+A(i+1))
ENDDO
```

通常の block 分割では，参照 A($i-1$) と A($i+1$) について，ループ内でプロセッサ間通信が必要となり，効率が悪い．そこで，VPP Fortran と HPF では，**図-4** に示すような，block 分割の分割範囲を近接するプロセッサと重なり合うように拡張した分割方法をサポートしている．VPP Fortran では，この重なりを**袖** (overlap area) と呼んでいる．A を袖つきの分割で割り付け，ループ内では必要なら袖部分から参照を行うことにより，ループ内から通信を排除できる．

VPP Fortran では，袖の値の一意性はユーザが管理する．上のプログラム例では，ループ実行の前で，袖転送を記述する OVERLAPFIX 文を使用する方法が最も簡単である．一方 HPF では，袖の値の一意性は処理系が保証する．処理系は OVERLAPFIX 相当の通信を自動で適切な場所に生成しなければならない．

3.5 データ転送—SPREAD MOVE, OVERLAPFIX vs. ライブラリ群—

VPP Fortran と HPF の仕様の規模を比較す

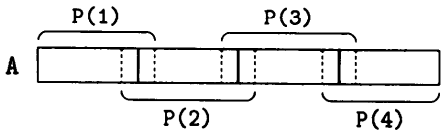

図-4 袖つき分割の例

る目安として，それぞれのディレクティブ行の種類の数とライブラリの数を表-2に示す．VPP Fortran と HPF では，ディレクティブの数では大差はないが，ライブラリの数が大きく違うことが分かる．HPF のもつライブラリの大部分はさまざまなパターンのデータ転送を実現するためのものである．一方，VPP Fortran では，データ転送機能として，配列から配列への非同期一括転送を記述する SPREAD MOVE 構文と，前述の OVERLAPFIX 文だけを提供している．これらは，逐次実行との親和性を重視して，ディレクティブ行として設計されている．

表-2 仕様規模の比較

	VPP Fortran	HPF 2.0	HPF 2.0+ approved extensions
宣言系 directive	6	6	7
実行系 directive	11	1	7
組込み手続き	0	3	6
ライブラリ手続き	6	50	65

4. 性能評価

VPP Fortran で記述されたアプリケーションの性能評価を通して，VPP Fortran の記述力を評価する．

図-5は，さまざまなマシンの LINPACK Highly Parallel Computing の性能[1]を元に，それぞれのハードウェアピーク性能に対する実現性能の比率を計算して示したものである☆4．このベンチマークは，問題サイズと記述手段を自由に選べるというルールで測定されている．つまり，熟練者がそれぞれのコンパイラとライブラリを使って，ハードウェアの潜在的な性能をどれだけ引き出せるか，という観点で比較することができる．

VPP 500 と VPP 700 の測定では記述手段として VPP Fortran だけを用いているが，非常によい並列化効率が得られている．これは，VPP Fortran 言語が，言語処理系を介してハードウェア性能を引き出すのに十分な記述力をもっていることを示しているといえる．

☆4 文献 1)の Table. 3 で上位に位置するマシンのうち商用の汎用機を対象とした．ただし，使用したアルゴリズムの違いにより flop/s 値を比較できない Cray C 90 などは除外している．

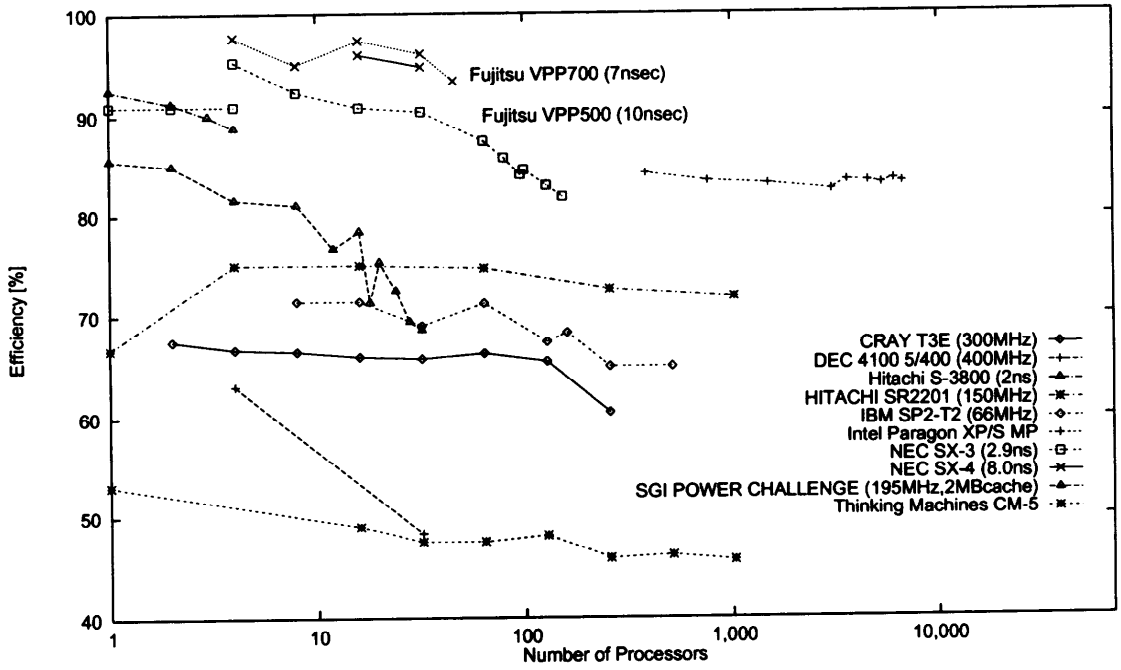

図-5 並列実行効率（LINPACK Highly Parallel ; Nov. 15, 1996）

5. おわりに

VPP Fortranは，実アプリケーションの検討をベースに，ユーザ記述によってハードウェアのピークに近い性能が引き出せることを目指して設計された言語である．将来，処理系の自動並列化技術が実用レベルに達してきても，「並列プログラムのアセンブラ」として，今後も使われてゆくことだろう．

一方，HPFは，処理系の最適化技術をある程度仮定した上で意味をもつ言語である．approved extensionによって，これまで処理系だけで自動的に行うと考えられていた多くの機能に関してもユーザが記述できるようになったが，最適化の主体は処理系で，ユーザ記述はそれをサポートするためであるという姿勢に変わりはない．

現在実用化されているVPP Fortranと，次世代の実用化が期待されるHPFの比較を行った．コンパイラは，徐々に高度な自動並列化・最適化の機能をもつようになるだろう．結果としてユーザは，コンパイラが行った最適化の内容や，最適化がうまく機能しない理由が把握しにくくなり，プログラムのどこをどう修正すればもっと性能を改善できるのか分からなくなることもあるだろう．そのような事態を避けるために，今後は言語仕様をプログラミング開発環境と結びつけて考えてゆく必要があると感じる．まず，コンパイラの内部情報をユーザにフィードバックするため，言語仕様に対応した実用的な開発支援ツールが不可欠である．そして，ツールの解析結果を生かしてプログラムに表現するためには，言語仕様にはやはりきめ細かな記述力が必要であると考える．

参考文献

1) Dongarra, J. J.: Performance of Various Computers using Standard Linear Equations Software (Nov. 1996). (最新版はhttp：//www.netlib.org/benchmark/performance. ps).
2) High Performance Fortran Forum, High Performance Fortran Language Specification Version 2.0. δ (Oct. 1996).
3) 岩下英俊, 進藤達也, 岡田 信: VPP Fortran: 分散メモリ型並列計算機向けプログラミング言語, 情報処理学会論文誌, Vol. 36, No. 7 (July 1995).

(平成8年9月27日受付)

岩下 英俊 (正会員)

1986年愛媛大学工学部電子工学科卒業．1988年同大学院工学研究科修士課程修了．同年(株)富士通研究所に入社，1994年より富士通(株)．並列Fortran言語の設計，並列化コンパイラの研究開発に従事．電子情報通信学会会員．