並列数値シミュレーション用高水準言語 NSL

川合隆光[†,☆] 市川周一[†,☆☆] 島田俊夫[†]

並列計算機のプログラミングは通信や同期，データ分散を意識して行う必要があるため一般に難しい．数値シミュレーションの分野において大規模な並列プログラムを開発する際，工数が膨大になりプログラミング上の誤りの発生も多くなる．これを克服する1つのアプローチとして高水準の問題記述からのプログラムの自動生成がある．本論文では，柔軟な問題記述機能と自動並列処理を特徴とする偏微分方程式用数値シミュレーション言語 NSL（Numerical Simulation Language）を提案し，試作した分散メモリ型並列計算機向けトランスレータの概要と並列処理方式について述べる．システムの性能評価を行った結果，格子点数 40000 点，PE 数 16 のとき格子生成部で効率 87.6%，求解部で 90.2%となり，本システムの有効性が確認できた．

NSL: High Level Language for Parallel Numerical Simulation

TAKAMITSU KAWAI,[†,☆] SHUICHI ICHIKAWA[†,☆☆] and TOSHIO SHIMADA[†]

Communication, synchronization, and data distribution are fundamental difficulties in multiprocessor programming. In such large applications as numerical simulations, these difficulties become more serious and dominant. One of approaches to overcome these difficulties is automatic program generation from high level problem description. In this paper, NSL, a new numerical simulation language for PDE problems, is proposed. NSL provides advanced features: flexible problem description function and automatic parallel processing. Its translator and parallel processing techniques are also described. The efficiency of the system is 87.6% for the grid generation part and 90.2% for the solver part in case that the number of grid points is 40000 and the number of PEs is 16.

1. はじめに

近年，数値シミュレーションの分野において並列計算機の有効性が認識されている．しかしプログラム開発はFORTRANやCなどの言語から並列ライブラリを呼び出す方式が主流であり複雑なプログラミングが要求される．とりわけ分散メモリ型並列計算機では通信や同期，データ分散に注意してプログラミングを行う必要があるため効率の良いプログラムを誤りなく作成するのは容易ではない．またこの種の並列計算機はデバッグが一般に困難である．これは作成するプログラムが大規模になるにつれ顕著になる．このためプログラマへの負担は大きくなり計算機の有効活用が難しくなる．

この問題を克服する1つの方向として，物理モデルレベルの問題記述言語から数値シミュレーションを行うプログラムを自動生成するシステムが提案されている．特に偏微分方程式で表される物理問題を対象とした数値シミュレーションでは，対象となる物理モデル中に本質的な並列性が内在しているはずである．このためモデルそのものを記述する言語を用いれば，逐次処理言語で書かれたプログラムを改めて並列化する場合に比べて高い並列性を抽出できる可能性がある[1)]．また解くべき問題を簡潔に記述できることや方程式および計算スキームの変更に容易に対応できるという利点がある．

本論文では，従来のシステムにない特徴である境界適合法[2)]とマルチ・ブロック法[3)]のための柔軟な領域形状記述機能および自動並列処理機能を持つ偏微分方程式用数値シミュレーション言語NSLを提案する．試作した分散メモリ型並列計算機向けトランスレータとその並列処理方式および評価結果について報告する．

† 名古屋大学工学部電子情報学科
Department of Information Electronics, School of Engineering, Nagoya University
☆ 現在，名古屋大学大学院工学研究科
Presently with Graduate School of Engineering, Nagoya University
☆☆ 現在，豊橋技術科学大学知識情報工学系
Presently with Department of Knowledge-based Information Engineering, Toyohashi University of Technology

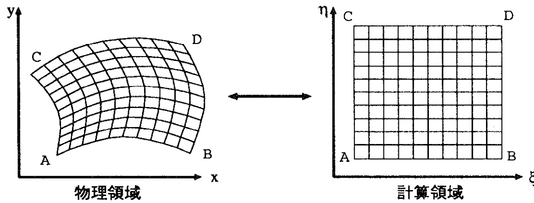

図1 境界適合法
Fig. 1 Boundary-fitted coordinate system.

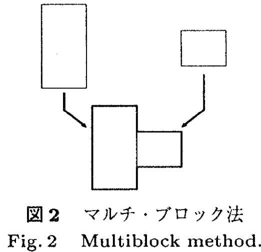

図2 マルチ・ブロック法
Fig. 2 Multiblock method.

2. 境界適合法とマルチ・ブロック法

熱流体力学，電磁気学等において現れる偏微分方程式を解く代表的な手法に差分法[4]がある．差分法を用いて矩形の物理領域で成立する方程式を解く場合，直交格子が用いられる．しかし実用的には複雑な曲線境界を持つ物理領域で方程式を解く必要がある．このとき曲線境界を直交格子で離散化すると一般に境界条件の精度が低下する．

この問題を解決する有力な手段として境界適合法が用いられる（**図1**）．境界適合法は，複雑な曲線境界を持つ物理領域で成立する偏微分方程式を矩形の計算領域に写像して解く方法である．格子点を物理境界に一致させることができるため境界条件を精度良く課すことができる．領域の写像により偏微分方程式の変換が必要となり元の方程式より複雑な形になるが，計算は矩形の計算領域で行われるためプログラム中では簡単なループで実行できる．

物理領域が非常に複雑になると単一の矩形の計算領域だけでは写像が困難になる場合がある．そこで複数の矩形の部分領域（ブロック）を接続して，物理領域とトポロジカルに同一な計算領域を構成することにより写像を容易にする方法が用いられる（**図2**）．これは一般にマルチ・ブロック法と呼ばれる．

NSLは，境界適合法とマルチ・ブロック法に基づいて並列計算機上で偏微分方程式を容易に解くことを目的とした言語システムである．

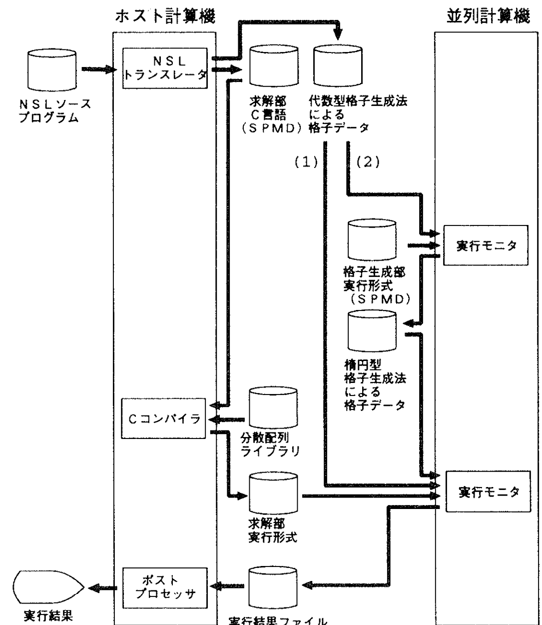

図3 NSL システムの構成
Fig. 3 NSL system.

3. NSL システムの概要

3.1 システムの構成

図3 に NSL システムの構成を示す．問題の領域形状，初期/境界条件，偏微分方程式を記述したNSLソースプログラムをNSLトランスレータに入力すると，求解部の並列プログラムが生成される．このプログラムはターゲット並列計算機上で利用可能なC言語によるSPMD（Single Program Multiple Data）形式のプログラムである．このプログラムはさらにCコンパイラによってコンパイルされ，分散配列ライブラリがリンクされて実行形式となる．分散配列ライブラリとはPE（Processing Element）に分散した配列に対するアクセスやデータ転送を管理するランタイムライブラリである．並列計算機にあらかじめ提供されている通信ライブラリの上位に独自に構築した．

NSLでは境界適合法を用いるので求解部の実行の際，領域の格子データが必要となる．このため格子生成を行う必要がある．NSLでは格子生成法は（1）代数型格子生成法（2）楕円型格子生成法の2つを選択できる．

（1）の場合には図3の（1）の経路で代数型格子生成法による格子データが求解の際にそのまま用いられる．この格子データは求解部の並列プログラムが生成されると同時に生成される．

（2）の場合には図3の（2）の経路で生成された格子データが求解の際に用いられる．格子生成部はあらかじめ用意したSPMD形式の並列プログラムである．代数型格子生成法による格子データおよび格子制御パラメータを基に楕円型格子生成を行う．

求解部の実行結果はファイルに保存される．実行結果の確認はポストプロセッサで行う．

3.2 対象とする問題と解法

NSLは時間に関して1階，空間に関して2階までの2次元偏微分方程式の初期値・境界値問題を対象としている．数値解は陽解法に基づく差分法により求める．陽解法は演算の並列性が高いため並列計算機に適している[5]．なお陽解法は数値安定性の面で問題があるが，解が発散する場合は空間および時間の離散化幅を適宜調節することにより解決するものとする．

4. 他のシステムとの比較

数値シミュレーション言語システムの例として，DEQSOL[6),7)]（日立製作所），DISTRAN[8)]（慶應義塾大学），//ELLPACK[9)]（Purdue大学），FEEL[10)]（NEC）などがある．NSLと他のシステムとの主な相違点は以下のとおりである．

DEQSOLはユーザによる計算アルゴリズムの柔軟な記述を主眼におく．DEQSOLでは複雑領域に対応するため境界適合格子機能を持つ[11)]が，格子の疎密制御は領域境界上の点の分布を指定することにより行うのみである．領域内部の特定の格子線や格子点へ格子を集中させる等の格子密度制御は考慮されていない．このため領域内部の格子を所望の形状に変化させることが難しく，計算誤差の増大を招くと考えられる．これに対しNSLでは格子内部の格子密度を制御する機能を持つため誤差を低減できる．また格子生成を並列計算機上で行うことにより処理時間の短縮を図っている．DISTRANは数値計算について専門的な知識を持たないユーザでも信頼性の高い計算ができることを目指している．//ELLPACKは標準的な解法ライブラリ群を並列化したものを呼び出す方式をとっている．FEELは有限要素法を採用し，領域分割をサポートしている．DISTRANや//ELLPACKでは直交格子を用いているため，2章で述べた境界における誤差の問題が発生すると考えられる．NSLでは境界適合法を用いることにより誤差を低減できる．

並列計算機においてこの種の問題を解くとき領域分割が必要になる．マルチ・ブロックからなる領域を分割すると，単一の矩形領域を分割する場合に比べて，発生する通信がより複雑になる．NSLでは領域分割にともなって発生する複雑な通信，データ分散をシステムが管理し，効率の良い並列処理を行うプログラムを自動生成することを目指している．またNSLは変数宣言および定数定義，座標，方程式，各種制御パラメータ等の指定を行うことができ，スキーム定義において一般的な高水準言語が備える制御構造をサポートしており，生成プログラムにその構造が反映される．したがってNSLは定型的なアプリケーションとは異なるシステムであるといえる．

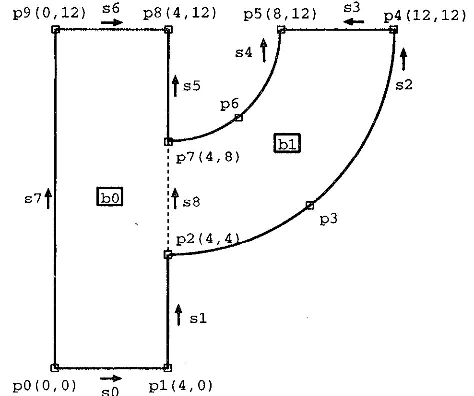

図4　領域の例
Fig. 4　Example of domain.

5. 言語仕様

問題記述例を示し，NSLの言語仕様の概要を述べる．NSLでは（1）領域記述，(2）方程式および初期/境界条件の記述を行うことにより問題記述を行う．

5.1 領域記述

領域を構成する基本単位を4つの曲線（または直線）境界で囲まれた閉領域とする．この閉領域をブロックと呼ぶことにする．ブロックを任意個接続して領域全体を構成する．境界形状は，線分，円弧，Bézier曲線などのセグメントの集合で表現する．領域記述は`domain`文中で行う．

例として直線境界を持つブロックと曲線境界を持つブロックが接続された領域（**図4**）において拡散方程式を解く場合のNSLソースプログラムを**図5**に示す．

(1) 物理形状を表現するための基本となる点の座標と名前を定義する（`point`文）．
(2) (1)で定義した点を用いてセグメントを定義し，名前と分割数を定義する（直線は`line`文，円弧は`arc`文で記述する）．ここで指定した分割数が計算領域の分割数となる．
(3) (2)で定義したセグメントを用いてブロックの

```
const double d1=2.0/sqrt[2], d2=4.0/sqrt[2];
domain {
    p0 = point[ 0,  0];
    p1 = point[ 4,  0];
    p2 = point[ 4,  4];
    p3 = point[ 4+4/d1, 12-4/d2];
    p4 = point[12, 12];
    p5 = point[ 8, 12];
    p6 = point[ 4+2/d1, 12-2/d2];
    p7 = point[ 4,  8];
    p8 = point[ 4, 12];
    p9 = point[ 0, 12];
    s0 = line[p0, p1, 10];
    s1 = line[p1, p2, 10];
    s2 = arc [p2, p3, p4, 10];
    s3 = line[p4, p5, 10];
    s4 = arc [p7, p6, p5, 10];
    s5 = line[p7, p8, 10];
    s6 = line[p9, p8, 10];
    s7 = line[p0, p9, 30];
    s8 = line[p2, p7, 10];
    b0 = block[s7, {s1,s8,s5}, s0, s6];
    b1 = block[s8, s3, s2, s4];
    gridcontrol[b0, point_xi, 3, 3, 10, 1.5];
}
variable u;
icond    u = 1.0, s0;
bcond    u = 1.0, s0;
         dn[u] = 1.0, s6;
scheme {
    int i;
    for (i = 0; i < 1000; i++) {
        dt[u] = dxx[u] + dyy[u];
    }
    output[u];
}
```

図5 記述例
Fig. 5 Example of description.

4つの辺を構成し，各ブロックの名前を定義する（block 文）．記述例では，ブロック b0 の4つの辺がそれぞれ s7, {s1, s8, s5}, s0, s6 で構成されることを表している．ブロック間で共通なセグメントにおいては，求解の際ブロック間でデータ交換が行われる．

差分法を用いて偏微分方程式を解く場合，解くべき方程式の解が領域のある部分で急激に変化していることが予想される場合には，その部分に格子を集中させることが望ましい．このため分割数の指定と同時にセグメント上の格子点分布の制御パラメータを指定できる．等分割，等比級数による分割に加え，Vinokur の補間関数[12]による分割の指定が可能である．Vinokur の方法では，始点，終点での格子間隔のみを陽に指定する．中間の格子点は tan および tanh に基づく補間関数で決定される．これにより境界に近い部分のみに格子を集中させる等の制御を容易に行うことができる（図6）．

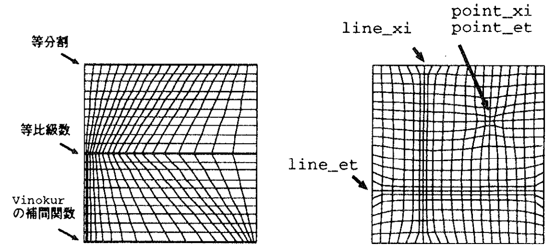

図6 格子密度の制御
Fig. 6 Grid control.

- 等分割，等比級数による分割：(例) 点 p0, p1 を結ぶ線分を等分割または等比級数で n 分割する．最初の格子幅を等分割のときに対し d 倍とする．
  s1 = line[p0, p1, n]; （等分割）
  s1 = line[p0, p1, {n, d}]; （等比分割）
- Vinokur の補間関数による分割：(例) 線分を n 分割する．始点，終点での格子幅は等分割時に対し dl1, dl2 倍とする．たとえば dl1 = 0.5, dl2 = 0.5 と指定することにより，両端は密，中間は疎な分割ができる．
  s1 = line[p0, p1, {n, dl1, dl2}];

NSL では Thompson の格子密度制御関数[2]を用いて格子をブロック内の特定の格子点または格子線に集中させることができる．この機能により，領域境界のみならず領域内部における格子密度制御が可能である（図6）．

- 格子線への集中：(例) ブロック b0 の格子線 $\xi =$ xi に格子を集中させる．a, b はそれぞれ，集中の強度，影響範囲を指定する．
  gridcontrol[b0, line_xi,xi,a,b];
- 格子点への集中：(例) ブロック b0 の格子点 (xi, et) に格子を集中させる．point_xi の場合，格子線 $\xi =$ xi の両側から，point_et の場合，格子線 $\eta =$ et の両側から集中させる．この2つを併用することにより ξ, η の両方向から格子を集中させることが可能である．
  gridcontrol[b0, point_xi,xi,et,a,b];
  gridcontrol[b0, point_et,xi,et,a,b];

5.2 変数・方程式等の記述

変数・方程式等の記述は以下の文により行う．

- const：定数を定義する．
- variable：未知変数を宣言する．複数個の宣言が可能である．
- icond：初期条件とそれが成立する領域を指定

する.

- `bcond`：境界条件とそれが成立する領域を指定する．Dirichlet 条件，Neumann 条件の指定が可能である．`dn[u]` を用いて境界の法線方向の u の微分を表す.
- `scheme`：計算スキームを記述する．偏微分方程式の記述に使用できる微分演算子は時間に関する1階の微分と空間に関する2階までの微分である．`dx[u]` は $\partial u/\partial x$ を，`dxx[u]` は $\partial^2 u/\partial x^2$ を表す．また，`if`，`for`，`do` ~ `while` といった制御構文を使用できる．`dt[u] =` の形の式が出現する場所では，新たなタイムステップの計算が行われる．`output` により，計算結果を出力したい変数を指定する.

6. 変換方式

3.1 節で述べたように，求解部の並列プログラムが NSL システムによって生成される.

求解部は分割されたブロックに対して並列に求解処理を行うように生成される．ブロックへの低レベルなアクセスは後述の分散配列ライブラリを呼び出すことにより行う．NSL ソースプログラム中に書いた `for` 文などの制御構文は生成された並列プログラム中でも同様の制御構文で生成される.

境界適合法の処理に必要な偏微分方程式の変換は微分演算子に対する変換式を用いて行う．たとえばラプラス演算子に対しては，変換式[4)]

$$\begin{aligned}\Delta f = &(\alpha f_{\xi\xi} - 2\beta f_{\xi\eta} + \gamma f_{\eta\eta})/J^2\\ &+[(\alpha x_{\xi\xi} - 2\beta x_{\xi\eta} + \gamma \xi_{\eta\eta})(y_\xi f_\eta - y_\eta f_\xi)\\ &+(\alpha y_{\xi\xi} - 2\beta y_{\xi\eta} + \gamma y_{\eta\eta})(x_\eta f_\xi - x_\xi f_\eta)]\\ &/J^3\end{aligned}$$

ただし

$$\alpha = x_\eta^2 + y_\eta^2,\ \beta = x_\xi x_\eta + y_\xi y_\eta,\ \gamma = x_\xi^2 + y_\xi^2,$$
$$J = x_\xi y_\eta - x_\eta y_\xi$$

を差分方程式に適用して変換する．ここで f は未知変数，x，y は物理領域の座標変数，ξ，η は計算領域の座標変数である.

7. 並列処理方式

NSL システムでは境界適合法を用いるため，まず対象領域において格子生成を行った後，その格子上で求解を行う必要がある．NSL では，格子生成部，求解部ともに並列処理を行い処理時間の短縮を目指す.

領域分割はそれぞれのブロックを2次元的に等分割することにより行う．なお 2 PE および 8 PE の場合のように PE 数の平方根が整数にならない場合は，分割された格子のアスペクト比が1に近くなるよう 1×2，2×4 のように分割する.

7.1 格子生成部の並列化

NSL システムでは，格子生成法として以下のどちらの方式も用いることが可能である.

(1) なめらかさは多少落ちるが計算量が少ない代数型格子生成法

(2) 計算量は多いが比較的なめらかな格子を生成できる楕円型格子生成法

(1)の方法を用いる場合，計算量が少ないため格子生成は並列計算機を用いずホスト計算機上のみで行う．このときブロック内部の格子座標はブロックの境界形状の座標値を用いて決定される.

(2)の方法を用いる場合，計算量が多いため格子生成部を並列計算機上で動作させることにより格子生成を行う．まず前処理として格子座標計算の発散を防ぐため(1)の方法によりホスト計算機上でブロック内部の初期格子を生成する．次にこの初期格子を基に並列計算機上で楕円型偏微分方程式を解き，滑らかな格子に収束させる．同時に，必要ならば格子内部の密度制御も行う.

このとき，誤差の2乗の総和が指定された一定値以内に収束するまで計算を続ける．ブロック間の接合部も考慮に入れて計算を行うので求解部と同様のパターンのブロック間データ転送が発生する．本来，数値計算部とは計算負荷が異なるため領域分割のパターンを別途考慮する必要があるが，現時点では同等の分割を使用している.

7.2 求解部の並列化

求解部では格子生成部で生成された格子上で求解を行う．求解部は NSL システムによって並列プログラムとして生成される．それぞれのブロックにおける未知変数を時間変数で数値的に積分することにより解を求める．変数値を更新した後，接続境界において他のブロックとのデータ転送を行う.

陽解法を行う場合，バッファはブロック1個に対し2個のタイムステップ分（step0，step1）必要となる．たとえばあるタイムステップの解が step0 に格納されているとすると，次のタイムステップの解は step1 に格納される．次のタイムステップでは逆になる.

まず，この処理を逐次計算機を用いて行った場合を考えると，ブロック間で外部セル（他のブロックが接続している部分のデータを保持する領域）におけるデータ交換のみが必要となる．たとえばブロックが2個（block0，block1）の場合，処理の流れは以下のようになる.

(1) block0，block1 における step0 の未知変数を初期条件に従って初期化し（初回のみ），block0，block1 における step0 の外部セルのデータ交換を行う．
(2) step0 のデータを基に新たなタイムステップの計算を行い，結果を step1 に格納する．
(3) step1 において (1) と同じ処理を行う．
(4) step1 において (2) と同じ処理を行う．結果は step0 へ格納する．
(5) 終了条件が満たされていなければ (1) へ戻る．

これを並列処理する場合，単純にブロック単位で PE に割り当てる方法が考えられる．しかし PE 数に対してブロック数が少ない場合やブロックのサイズが大きく異なる場合，負荷バランスが悪化する．そこでそれぞれのブロックをさらに 2 次元的に細分化して PE に割り当てることにより，負荷の均等化を図る．このとき，ブロックには外部セルに加えてブロック内部の分割部分におけるデータを交換する内部セルが必要となる．内部セルには，ブロック内データ転送によりデータ転送を行う．

たとえば，ブロックが 2 個の場合，処理の流れは図 **7** のようになる．

(1) 各 PE の担当ブロックの step0 の未知変数を初期条件に従って初期化し（初回のみ），それぞれのブロックで内部セルを交換する．
(2) ブロック間で外部セルを交換する．
(3) step0 のデータを基に新たなタイムステップの計算結果を step1 に格納する．
(4) step1 において (1) と同じ処理を行う．
(5) step1 において (2) と同じ処理を行う．
(6) step1 において (3) と同じ処理を行う．結果は step0 へ格納する．
(7) 終了条件が満たされていなければ (1) へ戻る．

NSL トランスレータは求解部として上記のような処理を行うプログラムを出力する．

7.3 分散配列ライブラリ

前節で述べたマルチ・ブロック法の並列実行の際のデータ転送をサポートする専用ライブラリを作成した．ここではこれを分散配列ライブラリと呼ぶ．以下にこのライブラリについて述べる．

7.3.1 機 能

本ライブラリでは以下の機能をサポートしている．

(1) ブロックの生成/消去
(2) ブロックの分割数の指定
(3) 分割したブロックをどの PE へ割り当てるかの指定

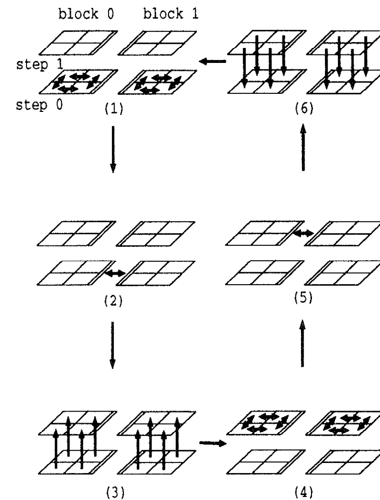

図 7 データ転送
Fig. 7 Data transfer.

(4) 各 PE におけるブロックの処理範囲の取得
(5) ブロックへのアクセス機能
(6) 同一ブロック内および異なるブロック間のデータ転送

本ライブラリでは 1 つのブロックを PE 間に分散する論理的な 1 つの配列と考える．配列の各次元には PE 間でグローバルな添字空間を与えることができる．ブロックのある領域を処理しようとするとき，PE は各自の処理範囲を取得する必要がある．このため，グローバルな添字空間の矩形の処理範囲を，各 PE が持つローカルな配列の処理範囲に変換する機能を持つ．ブロック間のデータ転送は転送元ブロックの矩形領域と転送先ブロックの矩形領域を指定することにより行う．このときグローバルな添字空間で領域を指定できる．内部セルのデータ交換は，対象となるブロックを指定して関数を発行することにより一度に行うことができる．

以上のように，マルチ・ブロック法の実行に必要な関数群が提供されている．本ライブラリは格子生成部および求解部にリンクされる．

7.3.2 データ構造

NSL の分散配列ライブラリでは (1) `Array`，(2) `Block` の 2 つのデータ構造の階層を持つ（図 **8**）．

`Array` は最もプリミティブなデータ構造であり，配列データ本体を保持する．一般には任意の次元の配列を扱えるのが望ましいが，現在の実装では 2 次元の配列としてアクセスできるようになっている．PE への割り当ては `Array` を単位として行われる．`Array` は各次元の上端，下端に袖領域を持つ．

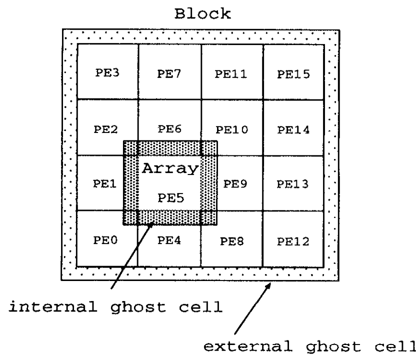

図 8 ブロックのデータ構造
Fig. 8 Data structure of block.

Block は Array を 2 次元に配列したデータ構造である．すなわち Block は Array に分割される形になる．ただし隣接する Array の袖領域（内部セル：internal ghost cell）は重ね合わされ，ブロック全体のグローバルな添字空間からは見えない．

Block も各次元の上端，下端に袖領域（外部セル：external ghost cell）を持つが，この実体は Block の最も外側の Array の袖領域である．

以上のように，ブロックの分散はライブラリのレベルで論理的に管理される．このことは生成されたプログラムを理解しやすくするのにも役立つ．

8. 評　価

NSL システムは現在ターゲット並列計算機として分散メモリ型並列計算機 Cenju-3[13]（CPU：V_R4400 75 MHz，ローカルメモリ：32 MB，PE 数：16）を用いている．分散配列ライブラリは Cenju-3 に提供されている mini-MPI（MPI[14]のサブセット）を用いて構築した．移植性を考慮してメッセージ・パッシングの標準規格である MPI を採用した．

格子生成部・求解部における並列処理性能について評価する．性能を表す指標として，台数効果，並列化効率および通信時間の比率を測定する．

対象とする偏微分方程式には代表的な問題として拡散方程式を用いる．**図 9** に示す曲線境界を持つ 2 種類の領域形状について問題を記述した場合を評価する．形状 1 は単一のブロックからなる形状であり，形状 2 は 5 つのブロックからなる形状である．扇形の部分は論理的には正方格子に写像されている．それぞれの扇形の内側の円弧に接する格子の半径方向の格子間隔が，外側の円弧に接する格子間隔に対して約 1/10 になるよう格子線を集中させた．

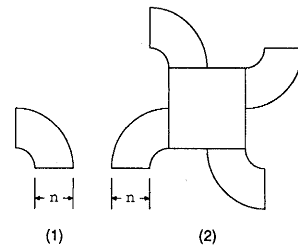

図 9 評価領域形状
Fig. 9 Domain shape for evaluation.

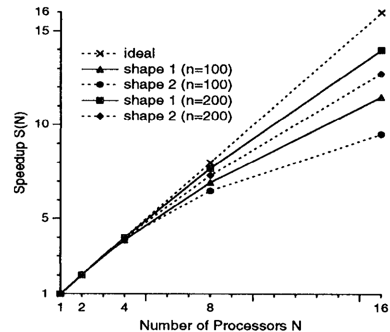

図 10 台数効果（格子生成部）
Fig. 10 Speedup (grid generation part).

8.1 台数効果および並列化効率

図 9 の形状に対して PE 数および格子点数を変化させたときの台数効果を測定した．格子点数は図 9 において $n = 100$（形状（1）で格子点数 10000 点）の場合と $n = 200$（形状（1）で格子点数 40000 点）の場合を評価した．ここでは PE 数 N 台の時の台数効果を $S(N) = T(N)/T(1)$，並列化効率を $E(N)[\%] = 100 \times S(N)/N$（ここで $T(N)$ は 1 イテレーションの実行時間）と定義する．なお 1 台の場合のプログラムも並列プログラムである．

格子生成部の台数効果を**図 10** に示す．n が大きい方が効率が高い．形状（1）の場合，$n = 200$，PE 数 16 のとき効率は 87.6% となった．形状（2）の場合 79.7%となった．

求解部の台数効果を**図 11** に示す．同様に n が大きい方が効率が高い．形状（1）の場合，$n = 200$，PE 数 16 のとき効率は 90.2%となった．形状（2）の場合 85.6%となった．以上の結果から，複数ブロックを接続した場合でも大きな効率の低下は発生していないことが分かる．

8.2 通信時間の占める比率

形状（1），形状（2）において 1 イテレーションの実行時間のうち通信時間の占める比率を測定した．格子生成部，求解部における比率をそれぞれ**図 12**，**図 13**

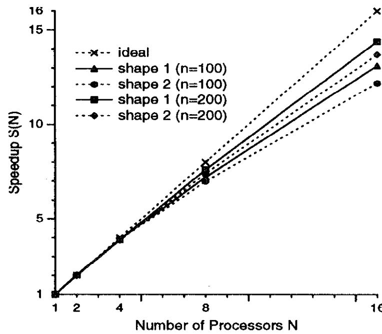

図11 台数効果（求解部）
Fig. 11 Speedup (solver part).

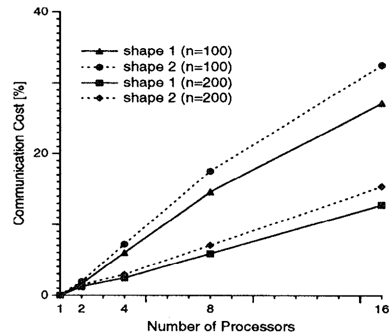

図12 通信量の割合（格子生成部）
Fig. 12 Communication cost (grid generation part).

に示す．全般的に形状（2）の方が，また格子生成部の方が通信コストが大きいことが分かる．

形状（1）では単一ブロックのためブロック間の通信は存在せず，分割されたブロック内の小領域間の通信のみであるのに対し，形状（2）では複数ブロックが接続されているため，ブロック間の通信も必要となる．このため複雑な形状の場合ほど通信コストが大きくなる．

8.3 考察

求解部での台数効果が格子生成部より全般的に良好であるのは以下の理由による．格子生成部では x，y の格子座標データを格納するため1ブロックあたり合計2個のバッファが確保され格子生成のイテレーションが行われる．このとき2個のバッファそれぞれに対して通信が必要である．これに対し求解部では格子座標データのためのバッファが2個と，変数1個に対し2つのタイムステップ分のバッファの合計4個が確保されるが，このとき格子座標データは格子生成部で得られた値を初期設定した後は変更の必要がなく，必要な通信は変数1タイムステップ分のバッファに対する通信のみであるためである．図10，図11において形状（2）が形状（1）に対して若干性能低下しているのは，現在の実装ではブロック内の小領域間の通信とブロック間の通信が別のメッセージとして送信されていることにより，通信時間が増加するためと思われる．これらをあらかじめベクトル化されたメッセージに変換しておくことにより性能は改善されると考えられる．

9. おわりに

マルチ・ブロック法と境界適合法をベースとし，柔軟な領域形状記述と自動並列処理を特徴とする偏微分方程式用数値シミュレーション言語NSLを提案した．試作した分散メモリ型並列計算機向けトランスレータを含むシステムを評価した結果，格子点数40000点，PE

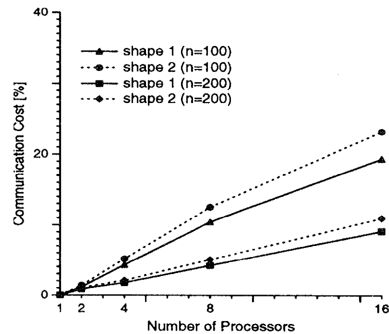

図13 通信量の割合（求解部）
Fig. 13 Communication cost (solver part).

数16のとき格子生成部で87.6%，求解部で90.2%となり，本システムの有効性を確認できた．通信および演算量の最適化，領域分割法の改良および解法の多様化（3次元問題，実行時に空間的な負荷分布が変わる問題等）が今後の課題である．

謝辞 本研究の一部は新情報処理開発機構の受託研究費，文部省科学研究費補助金試験研究（B）課題番号07458057およびNEC C&C研究所並列処理センタの支援を受けました．ここに謝意を表します．

参考文献

1) 小柳義夫：なぜユーザはベクトル計算機から離れられないのか，並列処理シンポジウム JSPP '92, pp.31–32 (1992).
2) Thompson, J.F., Warsi, Z.U.A. and Mastin, C.W.: *Numerical Grid Generation: Foundations and Applications*, North-Holland (1985).
3) Rubbert, P.E. and Lee, K.D.: Patched Coordinate Systems, *Numerical Grid Generation*, Thompson, J.F. (Ed.), North-Holland (1982).
4) 高見穎郎，河村哲也：偏微分方程式の差分解法，東京大学出版会 (1994).
5) 星野 力：PAXコンピュータ，オーム社 (1985).
6) 佐川暢俊，金野千里，梅谷征雄：数値シミュレ

```
domain-definition:
 domain { domain-definition-list }
domain-definition-list:
 domain-definition-list domain-definition
domain-definition:
 id = point [ expression, expression ];
 id = line [ point-list, decomposition-spec ];
 id = bezier[ point-list, decomposition-spec ];
 id = arc [ point-list, decomposition-spec ];
 id = block [ edge-specifier-list ];
 gridcontrol [ domain, control-type,
  control-parameters ];
const-definition:
 const type-specifier const-definition-list;
const-definition-list:
 const-definition-list, id = expression

scalar-variable-declaration:
 type-specifier scalar-variable-list;
variable-declaration:
 variable variable-list ;
icond-definition:
 icond id = expression , domain;
bcond-definition:
 bcond id = expression , domain;
 bcond dn [ id ] = expression , domain;
scheme-definition:
 scheme { scheme-definition-list}
scheme-definition-list:
 statement-list
statement-list:
 statement
 statement-list statement

statement:
 const-definition
 scalar-variable-declaration
 expression ;
 if ( expression ) statement
 if ( expression ) statement else statement
 for ( expression; expression; expression )
  statement
 while ( expression ) statement ;
 do expression while ( expression ) ;
 compound-statement
 output [ variable-list ];
compound-statement: { statement-list }
```

注：非終端記号はイタリック体で示した（一部の非終端記号の詳細は省略した）．dx, dxx, dy, dyy などは一部の expresssion 中で使用可能である．

図 14 NSL の構文規則（抜粋）
Fig. 14 Syntax of NSL (excerpt).

ーション言語 DEQSOL，情報処理学会論文誌，Vol.30, No.1, pp.36–45 (1989).

7) 大河内俊夫，金野千里，猪貝光祥：高水準数値シミュレーション言語 DEQSOL の並列計算機向けトランスレータ，情報処理学会論文誌，Vol.35, No.6, pp.977–985 (1994).

8) 鈴木清弘ほか：DISTRAN システムの並列計算機上への実装，並列処理シンポジウム JSPP'91, pp.301–308 (1991).

9) Houstis, E.N. and Rice, J.R.: Parallel ELLPACK: A Development and Problem Solving Environment for High Performance Computing Machines, *Programming Environments for High-Level Scientific Problem Solving*, Gaffney, P.W. and Houstis, E.N. (Eds.), Elsevier Science Publishers, North-Holland (1992).

10) Shun, D. et al.: FEEL: A Simulation Language for Finite Element Analysis, *Proc. IFIP/WG2.5 International Workshop on Current Directions in Numerical Software and High Performance Computing* (1995).

11) Konno, C. et al.: *The BF (Boundary-Fitted) Coordinate Transformation Technique of DEQSOL*, SIAM, pp.322–326 (1988). (ISBN 0-89871-228-9).

12) Vinokur, M.: On One-dimensional Stretching Functions for Finite-difference Calculations, *J. Comp. Phys.*, Vol.50, No.2, pp.215–234 (1983).

13) 広瀬哲也ほか：並列コンピュータ Cenju-3 のアーキテクチャ，情報処理学会研究報告 94-ARC-107-16, Vol.1, No.1, pp.121–128 (1994).

14) Message-Passing Interface Forum: *MPI: A Message-Passing Interface* (1994).

付　　録

NSL の構文規則（抜粋）を図 **14** に示す．

(平成 8 年 6 月 5 日受付)
(平成 9 年 3 月 7 日採録)

川合　隆光（学生会員）

平成 4 年岐阜大学工学部電子情報工学科中退．平成 6 年名古屋大学大学院人間情報学研究科物質・生命情報学専攻博士前期課程修了．現在名古屋大学大学院工学研究科電気工学・電気工学第 2 および電子工学専攻博士後期課程在学中．並列処理ソフトウェアの研究に従事．ACM 会員．

市川　周一（正会員）

昭和 60 年東京大学理学部情報科学科卒業．昭和 62 年東京大学大学院理学系研究科情報科学専門課程修士課程修了．新技術開発事業団，(株) 三菱電機，名古屋大学工学部助手を経て，現在豊橋技術科学大学知識情報工学系講師．理学博士．計算機アーキテクチャの研究に従事．IEEE，ACM 各会員．

島田　俊夫（正会員）

昭和 43 年東京大学工学部計数工学科卒業．昭和 45 年東京大学大学院工学系研究科計数工学専攻修士課程修了．同年通産省工業技術院電子技術総合研究所入所．昭和 63 年同所情報アーキテクチャ部計算機方式研究室長．平成 5 年より名古屋大学工学部電子情報学科教授．工学博士．人工知能向き言語，LISP マシン，データフロー計算機の研究に従事．最近はマイクロプロセッサのアーキテクチャやチップ内並列処理の研究を行っている．昭和 63 年度市村賞，平成 7 年度注目発明賞受賞．IEEE，ACM，電子情報通信学会，応用数理学会各会員．