ハイパフォーマンス
コンピューティング 55－1
（1995．3．9）

数値シミュレーション言語NSLにおける並列処理手法

川合 隆光　市川 周一　島田 俊夫

名古屋大学工学部 電子情報学科

概要

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．これを克服する一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．本稿では，領域形状記述法に特徴を持つ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，領域分割に主眼を置い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その並列処理方式，評価結果について報告する．

Parallel Processing Techniques in NSL (Numerical Simulation Language)

Takamitsu Kawai　Shuichi Ichikawa　Toshio Shimada

Department of Information Electronics, Faculty of Engineering, Nagoya University

Abstract

Communication, synchronization, and data distribution are fundamental difficulties of multiprocessor programming. In such large applications as numerical simulations, these difficulties become more serious and dominant. One of approaches to relieve these difficulties is automatic generation of programs from high level description of problems. In this paper, NSL, a new numerical simulation language for PDE problems, is proposed, which features extensive domain shape description. Its translator and parallel processing techniques are also described.

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が積極的に使用されるようになってきた．しかしそのプログラム開発に用いられる言語の主流は依然としてFORTRANやCであり，大規模なソフトウェアを作成する場合，膨大な工数を伴う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，領域分割に注意してプログラミングを行う必要があるため，プログラミング上の誤りが混入しやすくデバッグも行いにくい．従ってプログラマへの負担が大きくなり，計算機の有効活用も難しくなる．

この問題を解決するため，高水準の問題記述から，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考えられている．主なものに，DEQSOL [1] [2] (日立製作所)，DISTRAN [4] (慶應義塾大学)，//ELLPACK [5] (Purdue大学)，などがある．これらは偏微分方程式で表される分布系の物理問題を対象とし，高水準言語による問題記述から数値解を求めるFORTRANプログラムを出力する．また，複雑な領域形状を入力するためのプリプロセッサや，結果表示のためのポストプロセッサを持つものもある．

このようなシステムの利点は，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ことにある．また問題レベルの記述言語を用いているため，並列計算機で実行するときプログラマがFORTRAN等を直接用いて作成したプログラムを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．

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がある．差分法では通常，解を直交格子上で求めるので，矩形の物理領域の場合は適用が容易である．しかし実用的な任意形状の境界を持つ物理領域の場合，曲線境界を階段状の直交格子で離散化すると境界条件を設定する際に精度の低下を招く．

このような問題を解決する有力な手段として，境界適合法 [6] が用いられる (図1)．境界適合法は，複雑な曲線境界をも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．領域の写像により偏微分方程式の変換が必要となり複雑な形になるが，これは数値的に解決できる．また，格子点が物理境界上に確実に置かれるため，境界条件を精度良く課すことができる．

境界適合法において複雑な形状の物理領域を計算領域に写像するとき，物理領域を複数の部分領域に分割し，それぞれに対応する矩形の計算領域を相互に接続して全体の系を構成する場合がある (図2)．DEQSOLは複雑領域に対応するための機能の一つとしてBFM機能を持ち，境界適合格子を扱うことができる．この機能では複数の部分領域を接続した計算領域を構成できるが，格子生成はそれぞれの部分領域を単位として独立に行われ，全ての部分領域をまとめて一つの領域とみなした格子生成は行われない．すなわち隣接する部分領域間で格子を滑らかに変化させることが難しく，計算誤差の増大につながると考えられる．また，ある部分領域の境界の一部が別の部分領域(もしくは自分自身)の境界の一部に接続されるような複雑なトポロジの格子はサポートされていない．

これらのシステムに対し，本稿で報告するNSLでは境界適合法における複雑領域の記述をより柔軟に行うための構文が用意されており，多様な格子を生成できる．さらに，並列計算機においてこのような問題を解くとき領域分割が必要になる．複雑なトポロジの格子を分割すると，一つの矩形を分割する場合に比べて，発生する通信がより複雑になり，分割形状およびデータ分散の方法によって実行時間が大幅に変わる可能性がある．NSLトランスレータはこのような場合の通信が最適となるように領域分割やデータ分散を行うことを目指している．

本稿ではNSLの概要，現在開発中の分散メモリ並列計算機向け計算機トランスレータとその並列処理方式，評価結果について述べる．

2 NSLの概要

2.1 対象とする問題と解法

NSL(Numerical Simulation Language)は偏微分方程式用数値シミュレーション言語である．現時点では，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

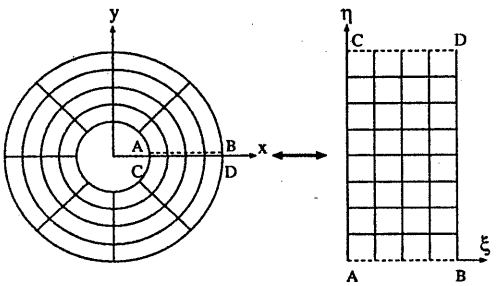

図 1: 境界適合法

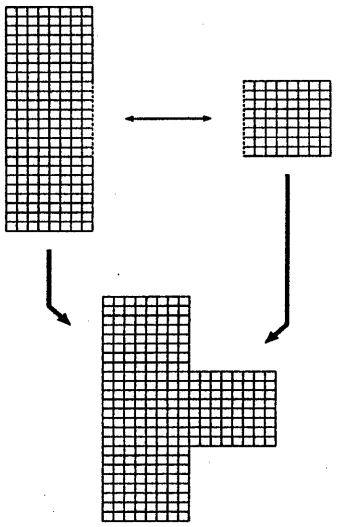

図 2: 複数ブロックの接続

求める。陽解法は数値安定性の面で問題があるが，これは時間および空間の離散化幅を小さくすることによりある程度解決できる．また，演算の並列性が高いため，並列計算機に適している．

2.2 言語仕様

2.2.1 領域の記述

NSL では解析領域全体を部分領域 (以下ブロックと呼ぶ) に分割して記述できる．ブロックは計算領域においては矩形領域に，物理領域においては 4 つの曲線境界で囲まれた閉領域に対応する．格子生成と数値計算のために，物理境界上の格子点の座標値とブロック間の接続状態の指定が必要となる．物理境界上の格子点の座標値全てを指定すると記述量が増大するので，いくつかの基本図形を用いて境界形状を記述する．

- `line` 文：線分を表現する．
- `arc` 文：円弧を表現する．
- `spline` 文：スプライン曲線を表現する．
- `topology` 文：ブロック間の接続状態を指定する．

これらの文により，複雑な物理領域形状を記述できる．NSL においては以上の事項を `domain` 文中で記述する．

2.2.2 変数・方程式等の記述

変数・方程式等の記述のために以下の文を使用できる．

- `time` 文：時間領域の範囲，タイムステップを指定する．
- `const` 文：定数を定義する．
- `variable` 文：未知変数を宣言する．複数個の宣言が可能である．
- `icond` 文：初期条件とそれが成立する領域を指定する．
- `bcond` 文：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` の微分を表せる．
- `equation` 文：偏微分方程式とそれが成立する領域を指定する．`equation` 文において使用できる微分演算子は現時点では時間に関する 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 $\frac{\partial u}{\partial x}$ を，`dxx[u]` は $\frac{\partial^2 u}{\partial x^2}$ を表す．
- `output` 文：計算結果を出力したい未知変数を指定する．

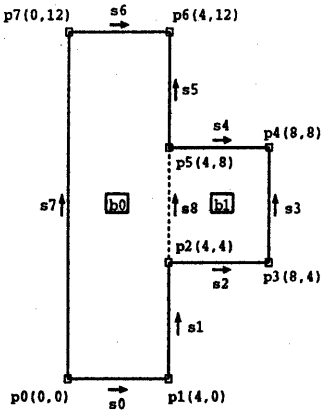

図 3: 領域記述

```
domain   p0 = point[0, 0];
         p1 = point[4, 0];
         p2 = point[4, 4];
         p3 = point[8, 4];
         p4 = point[8, 8];
         p5 = point[4, 8];
         p6 = point[4, 12];
         p7 = point[0, 12];

         s0 = line[p0, p1, 10];
         s1 = line[p1, p2, 10];
         s2 = line[p2, p3, 10];
         s3 = line[p3, p4, 10];
         s4 = line[p5, p4, 10];
         s5 = line[p5, p6, 10];
         s6 = line[p7, p6, 10];
         s7 = line[p0, p7, 30];
         s8 = line[p2, p5, 10];

         b0 = block[s7,{s1,s8,s5},s0,s6];
         b1 = block[s8, s3, s2, s4];

         topology[b0,s8, b1,s8, +];

time     0.0, 1.0, 1.0e-6;
variable u;
icond    u = 1.0, s6;
bcond    u = 1.0, s6;
         dn[u] = 1.0, s3;
equation dt[u] = dxx[u] + dyy[u], b0, b1;
output   u;
```

2.3 記述例

以下に NSL の記述例を示す．簡単のため，2 つのブロックが接続され，境界が直線で構成された領域で問題を解く場合の領域記述例を示す (図 3)．

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線分を定義し，名前と分割数を与える (line 文)．この分割数はそのまま計算領域の分割数となる．

3. 2. で定義した線分を用いてブロックの 4 つの辺を構成し，各ブロックの名前を与える (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．

4. 3. で定義したブロックの境界の接続関係を記述する (topology 文)．記述例では，ブロック b0 の線分 s8 がブロック b1 の線分 s8 に接続されていることを表す．+ 記号は接続部分の計算領域における添字が各ブロックで同方向に変化することを表す．topology 文で指定された境界は，格子生成および未知変数の計算の際，差分演算子が適用される境界となる．

2.4 NSL トランスレータの構成

以下に NSL トランスレータの構成を述べる．

1. 構文解析部：ソースファイルを構文解析し，処理系の内部表現に変換する．写像された微分演算子に対する差分演算子も求められる．

2. 領域分割部：1. で得られた情報を基に，計算領域の接続関係を解析し，通信コストを評価

関数として，この値が最小になるように領域分割を行う．

3. コード生成部：内部表現を基にC言語によるオブジェクトコードを生成する．

2.5 NSLトランスレータの入出力

NSLトランスレータの入力は，解くべき問題の領域形状，初期／境界条件，方程式を記述したNSLソースファイルであり，出力はターゲット並列計算機上で利用可能なC言語によるオブジェクトコードである．このオブジェクトコードはさらにCコンパイラによってコンパイルされ，並列ライブラリがリンクされて実行形式となる．計算結果はファイルにセーブされ，別途ポストプロセッサにて確認する．

2.6 オブジェクトコードの処理の流れ

NSLトランスレータは次のようなオブジェクトコードを出力する．

1. 各プロセッサの担当ブロックにおける格子を生成する．
2. 各プロセッサの担当ブロックの未知変数を初期条件に従って初期化する．
3. ブロックの接続部分の未知変数をプロセッサ間ないしはプロセッサ内通信 (すなわちプロセッサ内データ転送) により交換する．
4. バリア同期をかける．
5. 各プロセッサの担当領域の未知変数を並列に計算する．
6. 再びバリア同期をかける．
7. 終了条件を満たしていなければ3. に戻る．

オブジェクトコードは大きく分けて1. の格子生成部と2. 以降の数値計算部から成る．ここで，5. については，構文解析部において得られた偏微分方程式に対応する差分演算子が各プロセッサの担当ブロック内のすべてのデータに対して適用されるようループのコードが生成され，3. については，領域解析部で得られたブロックの接続関係の情報を基に，通信のコードが生成される．

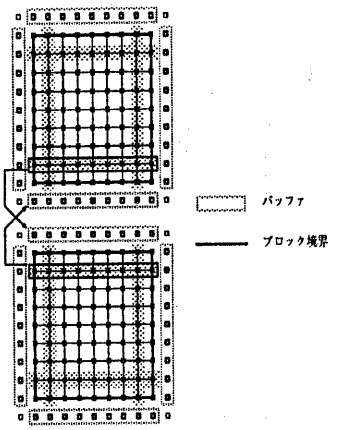

図4: ブロックのバッファ領域

3 並列処理方式

3.1 ブロックのデータ構造

NSL内部では，各々のブロックを分割しプロセッサに割り当てて並列処理を行う．

ブロックに付随するデータとして主に次のデータを保持する．

1. 格子点の物理領域における座標値
2. 未知変数値
3. 隣接ブロックとの接続情報

1. と2. についてはブロック境界の値を交換する必要があるので，隣接ブロック境界の内側に沿った値のコピーを格納するバッファ領域を持つ (図4)．領域の幾何学的な分割を容易にするため，境界上の格子点の値は隣接ブロックで重複して持つ．バッファへのデータ転送は，3. の接続情報に応じて通信することにより行われる．

3.2 領域分割

領域分割は様々なパターンが考えられるが，今回は各ブロックを縦方向または横方向にプロセッサ台数で等分した．通信コストがブロック境界の長さに比例する ($O(N)$) のに対し，演算コストはブロック面積に比例する ($O(N^2)$) ため，プロセッサ間の演算コストの不均衡の低減が重要となるからである．

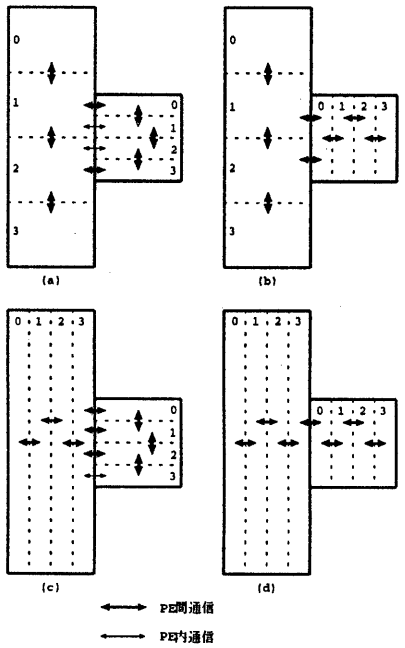

図 5: ブロック分割と発生する通信

ブロックが 2 個でプロセッサが 4 個の場合，分割パターンの候補は図 5のようになる．分割境界ではプロセッサ内通信またはプロセッサ間通信のいずれかが発生する．図中で太い矢印はプロセッサ間通信を，細い矢印はプロセッサ内通信を表す．0 ～ 3 はプロセッサ番号を表す．分割のパターンにより，発生する通信のパターンと量が異なり，通信コストも異なる値となる．

全てのブロックの分割を縦方向か横方向に定めた後，通信コストを計算する．プロセッサ間およびプロセッサ内通信のオーバヘッドをそれぞれ，

$$T_{inter_i} = s_{inter} + c_{inter} n_i$$

$$T_{intra_i} = s_{intra} + c_{intra} n_i$$

とモデル化する．ここで，s は通信のセットアップ時間，c は定数，n は転送データ数，i は境界の番号を表す．次に全通信コストを

$$T = \sum_i (T_{inter_i} + T_{intra_i})$$

で計算する．

以上のモデル化に従い，2^M(Mはブロック数) 通りの分割を試みて通信コスト T が最小になる分割を定める．

3.3 格子生成部の並列化

格子が時間的に変化しない場合，格子生成部は未知変数の計算に入る前に一度だけ実行すれば良く，大規模な計算では全体の計算時間に比べて無視できる程度の処理時間になる．しかし今後，移動境界問題，解適合格子などを採り入れることを考慮すると，全体の計算ループの中に入ってくる．そこで格子生成部も並列化を試みた．

滑らかに変化する格子を生成するには様々な方法がある．ここでは楕円型偏微分方程式を解く方法を採用し，並列化した．まず，格子座標計算の発散を防ぐため，ブロック境界上の格子点の物理領域における座標値を用いて代数的格子生成法によりブロック内部の初期格子を生成する．この処理は偏微分方程式を解く必要がないため，ブロック毎に独立に行う．次に初期格子を基に，楕円型偏微分方程式を解き，滑らかな格子に収束させる．このとき，全てのブロックに渡って計算を行うので数値計算部と同様のパターンの通信が発生する．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変える必要があるが，現時点では同等の分割を使用している．

3.4 数値計算部の並列化

格子生成部で生成された格子上で，数値計算本体が行われる．数値計算部ではそれぞれのブロックにおける未知変数を，時間変数で積分して求める．まず境界を除くブロックの内部領域のデータに対して差分演算子を適用し，新たなタイムステップの値を計算する．次に境界条件が課された境界部分に対して新たなタイムステップの値を計算する．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通信を行う．時間変数が `time` 文で定められた値になるまでイタレーションが行われる．偏微分方程式を解く処理の流れは終了条件を除けば格子生成部と同様となる．

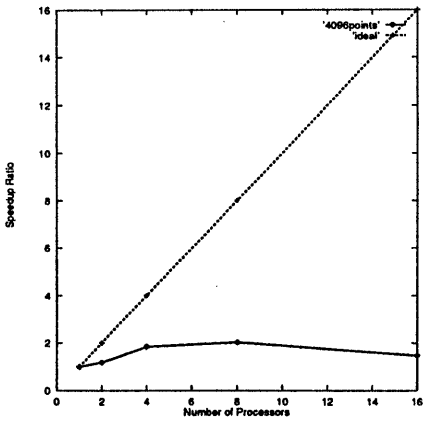

図 6: 速度向上率 (格子点数 4096)

格子点数	4096				
プロセッサ数	1	2	4	8	16
実行時間 [sec]	51	43	27	25	34
並列化効率 [%]	100	59	47	26	9

表 1: 実行時間，並列化効率 (格子点数 4096)

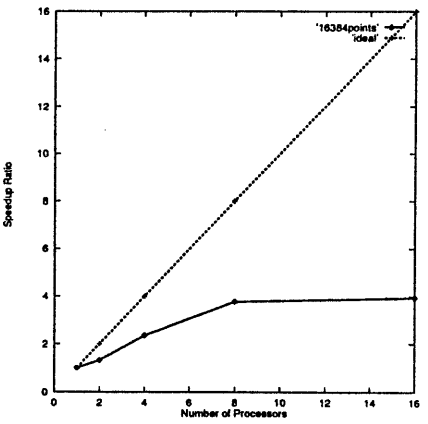

図 7: 速度向上率 (格子点数 16384)

格子点数	16384				
プロセッサ数	1	2	4	8	16
実行時間 [sec]	201	152	84	53	51
並列化効率 [%]	100	66	60	47	25

表 2: 実行時間，並列化効率 (格子点数 16384)

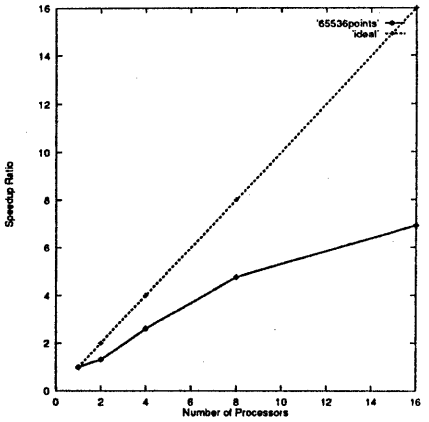

図 8: 速度向上率 (格子点数 65536)

格子点数	65536				
プロセッサ数	1	2	4	8	16
実行時間 [sec]	783	591	302	164	113
並列化効率 [%]	100	66	65	60	43

表 3: 実行時間，並列化効率 (格子点数 65536)

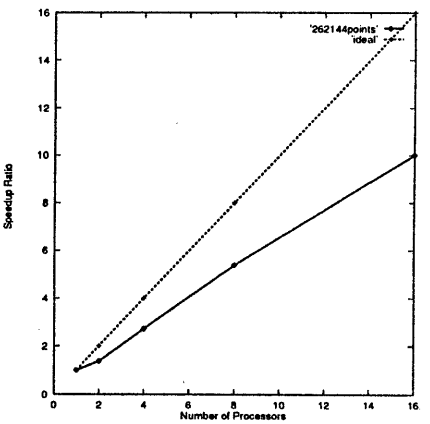

図 9: 速度向上率 (格子点数 262144)

格子点数	262144				
プロセッサ数	1	2	4	8	16
実行時間 [sec]	3722	2727	1360	692	372
並列化効率 [%]	100	68	68	67	62

表 4: 実行時間，並列化効率 (格子点数 262144)

4 評価

4.1 並列計算機 Cenju-3

今回の実装に用いた並列計算機 Cenju-3 の諸元を表5に示す．Cenju-3 は，多段接続網を持つ分散メモリ型並列計算機である．C プログラムからは，ライブラリ関数の呼び出しにより，リモートメモリリード／ライト，バリア同期，分散共有メモリ，リモートプロシジャコール等の機能が使用できる．NSL では，バリア同期とリモートメモリリード／ライト，リモートプロシジャコールの機能を使用している．Cenju-3 付属のライブラリではストライド転送はサポートされていないため，付属の通信関数の上に独自の通信関数を構成した．

PE 数	16
CPU	R4400SC
CPU 外部クロック	75MHz
命令実行ピーク性能 (1PE)	150MIPS
浮動小数点演算ピーク性能 (1PE)	50MFLOPS
ローカルメモリ容量 (1PE)	32MB
一次キャッシュ容量 (1PE)	32KB
二次キャッシュ容量 (1PE)	1MB
PE 間接続ネットワーク	多段接続網 (40MB/sec)

表 5: Cenju-3 の諸元

4.2 実行時間，加速率，並列化効率

評価のための問題として熱伝導方程式を用いた．格子点数とプロセッサ台数を変えたときの実行時間，加速率，並列化効率を表 1～表 4，図 6～図 9に示す．格子点数が 262144 のとき，並列化効率は最大 68%となった．

5 おわりに

本稿では，領域形状記述法に特徴を持つ偏微分方程式用数値シミュレーション言語 NSL を提案し，領域分割に主眼を置い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その並列処理方式，評価結果について述べた．今後の課題としては，

- 領域分割アルゴリズムの改良
- 演算と通信のオーバラップの導入
- ネットワークレイテンシが不均一な並列計算機上での評価
- 3 次元への拡張
- 解法の多様化

等がある．

参考文献

[1] 佐川暢俊, 金野千里, 梅谷征雄: 数値シミュレーション用プログラミング言語 DEQSOL, 情報処理学会論文誌, Vol.30, No.1, pp.36–45(1989).

[2] 大河内俊夫, 金野千里, 猪貝光祥: 数値シミュレーション向き高水準言語 DEQSOL の分散メモリ型並列計算機向けトランスレータに関する一考察, 並列処理シンポジウム JSPP'93, pp.39–46(1993).

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

[4] 鈴木清弘, 山崎信行, 他: 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).

[5] Weerawarana, S. et al.: Integrated Symbolic-Numeric Computing in //ELLPACK: Experiences and Plans, Technical Report CSD-TR-92-092, Department of Computer Sciences, Purdue University(1992).

[6] Thompson, J.F. et al.: Numerical Grid Generation: Foundations and Applications, North Holland, 1985.