JAERI-M
90-192

直接シミュレーション・モンテカルロ法による希薄流れ解析コードの高速化

１９９０年11月

渡辺　健二*・横川三津夫・山本　浩康・蕪木　英雄

日本原子力研究所
Japan Atomic Energy Research Institute

JAERI-Mレポートは，日本原子力研究所が不定期に公刊している研究報告書です。
入手の問合わせは，日本原子力研究所技術情報部情報資料課（〒319-11茨城県那珂郡東海村）あて，お申しこしください。なお，このほかに財団法人原子力弘済会資料センター（〒319-11茨城県那珂郡東海村日本原子力研究所内）で複写による実費頒布をおこなっております。

JAERI-M reports are issued irregularly.
Inquiries about availability of the reports should be addressed to Information Division Department of Technical Information, Japan Atomic Energy Research Institute, Tokai-mura, Naka-gun, Ibaraki-ken 319-11, Japan.

編集兼発行　日本原子力研究所
印　　刷　いばらき印刷㈱

JAERI－M 90－192

直接シミュレーション・モンテカルロ法による希薄流れ解析コードの高速化

日本原子力研究所東海研究所情報システムセンター
渡辺　健二*・横川三津夫・山本　浩康+・蕪木　英雄+

（1990年10月9日受理）

直接シミュレーション・モンテカルロ法（DSMC法）による希薄流れ解析コードの高速化を行った。DSMC法は，希薄気体から連続流体に近い流れまで幅広い領域の流れのシミュレーションに有効な手法である。一般に，DSMC法を用いたコードはベクトル化が困難であると言われていた。ここでは，DSMC法をベクトル計算機で高速に実行するために，ベクトル化のための処理手順を検討しコードの最適化を図った。この結果，FACOM VP－100でのフリージェット流の解析において，ベクトル化コードは，オリジナルコードのスカラ処理に対し，スカラ処理で約7倍，ベクトル処理で約25倍に高速化された。本報告書では，このコードにおける最適化・ベクトル化の手法とその性能評価について述べる。また，DSMC法のベクトル性能を更に向上させるための方法についても検討する。

東海研究所：〒319－11　茨城県那珂郡東海村白方字白根2－4
＋　燃料・材料工学部
＊　外来研究員，富士通株式会社

JAERI-M 90-192

Vectorization of the Rarefied Gas Flow Analysis Code using the Direct Simulation Monte Carlo Method

Kenji WATANABE*, Mitsuo YOKOKAWA, Hiroyasu YAMAMOTO+
and Hideo KABURAKI+

Computing and Information Systems Center
Tokai Research Establishment
Japan Atomic Energy Research Institute
Tokai-mura, Naka-gun, Ibaraki-ken

(Received October 9, 1990)

A rarefied gas flow analysis code using the Direct Simulation Monte Carlo (DSMC) method has been vectorized. The DSMC method is a powerful technique for simulating a wide range of flows from rarefied to near continuum region. It has been known that the DSMC code is difficult to implement on vector processors. We have developed and optimized the DSMC code to use the vector processors effectively. Consequently, for the analysis of the free-jet flow on FACOM VP-100, the vectorized code runs 7 and 25 times faster in scalar and vecor processing respectively, in comparison with the scalar processing of the original code. In this report, we present the optimization and vectorization techniques for the code and the results of their bench mark test. The algorithm has also been studied to make the DSMC code much more effective for vector calculation.

Keywords: The DSMC Method, Rarefied Gas Flow, Vector Processors, Vectorization, Optimization, Simultion, FACOM VP-100

+ Department of Fuels and Materials Research

* On leave from Fujitsu Limited

目　次

Contents

1. はじめに

直接シミュレーション・モンテカルロ法（ＤＳＭＣ法）は，実在気体の流れを計算機上の模擬分子によりモデル化する数値シミュレーション手法であり，希薄気体流れの解析に有効である．近年の計算機処理能力の向上に伴い，ＤＳＭＣ法は連続体に近い流れに対しても有力な解析手法となってきた．そのため，連続流に近い流れから希薄気体流れへの幅広い領域を含む原子ビーム形成過程の解析のため，ＤＳＭＣ法による解析コードが開発された．しかしながらＤＳＭＣ法では，一般にコードのベクトル化は困難であると言われていた．そのため，従来から計算はスカラ処理で行われており，連続体に近い流れの解析には，非常に長い計算時間を必要とした．ここでは，ＤＳＭＣ法のベクトル計算機上での高速処理を目指して，ベクトル化のための処理手順を検討し，コードの最適化を図った．

本報告書の構成は，第２章で，コードに関する物理的，数値解析的な側面について述べる．次に，第３章で，コードの構造，動的挙動の分析を行い，問題点の抽出を行う．第４章では，問題点について，その高速化の手法をコーディングと併せて示す．第５章では，高速化の手法の性能評価，および高速化の結果を示し，それらについての考察を行う．最後に，本報告書の結論を第６章に要約する．

2. 直接シミュレーション・モンテカルロ法（DSMC法）

2.1 希薄気体流れの解析

解析対象とする流体体積中に十分な分子がなく，分子間衝突があまり起らない流れは希薄な流れと呼ばれ，その流れの解析には分子運動論によるアプローチが必要である．流れを希薄気体として取り扱うかを判定するパラメタとしては，式 (2.1)で定義するクヌーセン数Kn を用いる．

$$Kn = \lambda / L \qquad (2.1)$$

λ ： 分子の平均自由行程

L ： 流れの代表長さ

このクヌーセン数が$Kn > 10^{-2}$の希薄気体領域にある場合は，連続流体の仮定に基づくナビエ・ストークス方程式による解析ができず，ボルツマン方程式を支配方程式にした解析を行わなければならない．分子運動論に基づくボルツマン方程式は，ごく簡単な場合の厳密解しか得られておらず，近似的な方程式に基づく数値解析手法が数多く開発されている [1]．

一方，近年の飛躍的な計算機能力の向上に伴い，乱数を用いて確率論的に問題をとくモンテカルロ法が有力なシミュレーション手段となっている．希薄流れの解析においても，乱数によってボルツマン方程式を確率的に解く直接シミュレーション・モンテカルロ法（ Direct Simulation Monte Carlo method：以下ＤＳＭＣ法)が考案され，$10^{-2} < Kn < 10$の中間流領域を解析するために用いられている．

この方法では，気体を構成する各分子の運動を模擬した分子を追跡することによって流れ場を決定する．すなわち，アボガドロ数個のオーダーの分子の運動を，数千から数万個の模擬分子で代表させ，流れ場の統計的な分布を求めることになる．ここでは，ある微小体積にある模擬分子の速度の平均が，その位置における流体の速度を与えることになる．

希薄流れの解析は，スペースシャトル等の宇宙飛行体の設計や，ＣＶＤ（ Chemical Vapor Deposition ） による半導体製造過程の解析，原子法レーザー同位体分離法における原子蒸気流の挙動解析などに重要である．

2．2　ＤＳＭＣ法

ＤＳＭＣ法[2,3]は，“分離の原理（principle of uncoupling)”に基礎を置いている．この原理に基づけば，シミュレーションの時間ステップΔｔを平均自由時間（分子が平均自由行程を走るのに要する時間）より十分小さくとれば，分子の移動と衝突を分けて扱うことができる．したがって，速度分布関数ｆをもつ集団からランダムに抽出された模擬分子に対して，分子間衝突を確率に基づいて行い，その後でΔｔ毎に模擬分子を移動させる．ＤＳＭＣ法による定常状態のシミュレーションのアルゴリズムは Fig. 2.1 のようになる．

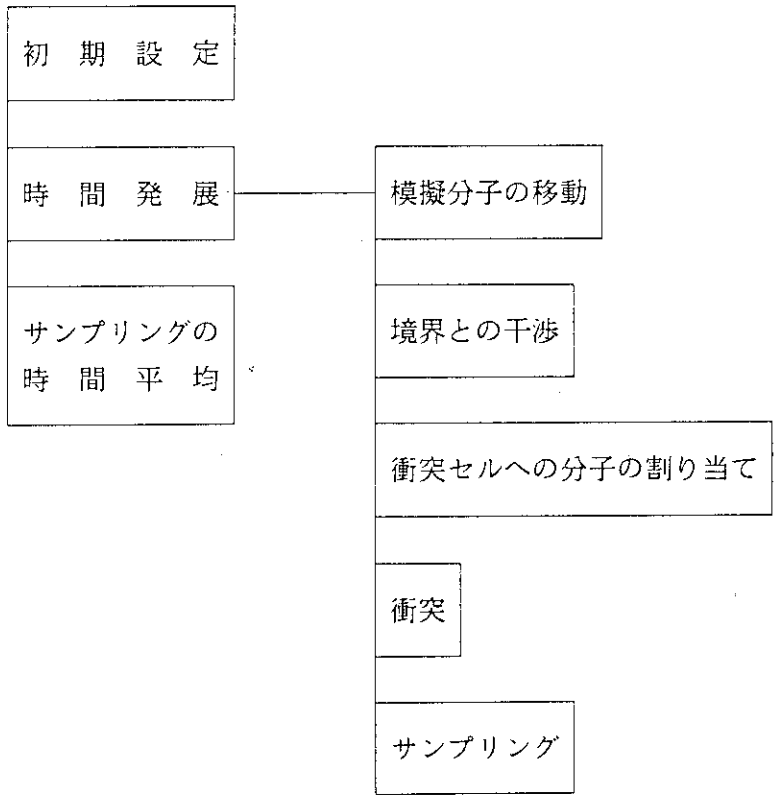


Fig. 2.1　The basic algorithm for the DSMC method

初期設定では，模擬分子の初期位置，初期速度，境界条件，衝突を行わせる衝突セル，統計量を取るためのサンプリングセルを設定する．衝突セルの大きさは，平均自由行程程度またはそれ以下に取ることが必要である．

模擬分子の移動は，各分子の持っている速度でΔｔだけ移動させる．境界条件は拡散反射条件か鏡面反射条件が設定されるが，境界と干渉した模擬分子に対しては，それぞれの条件に従って速度が更新され，Δｔ内の残り時間分移動させる．分子の座標が定まったら，その座標に基づいて分子を衝突セルに割り当てる．衝突は，衝突セルの中の模擬分子に対して確率的に行う．

定常状態の解析においては，流れ場の状態が十分定常に近づいた後に，適当な時間間隔でサンプリングを行い，場の速度や数密度などを求める．サンプリングの時間間隔は，前後のサンプリング値に相関がなくなる程度に大きく取ることが理想的である．得られたサンプリング値の時間平均をとることによって定常解が得られる．

衝突アルゴリズムにはBird法，南部法，修正南部法などがある．Bird法と修正南部法の計算量は模擬分子数Nのオーダーに比例するが，南部法はN^2 のオーダーに比例することが知られている．以下に，今回用いた修正南部法について説明する．

１つの衝突セル内のすべての模擬分子に対して，分子ｉが分子ｊと衝突する確率は，次の式で表わされる．

$$P_{ij} = (n/N) \cdot \sigma_T \cdot |g_{ij}| \cdot \Delta t \qquad (2.\ 2)$$

ｎ　：　分子数密度
Ｎ　：　衝突セル内の模擬分子数
σ_T　：　全衝突断面積
g_{ij}：　衝突分子対ｉ，ｊの相対速度
Δｔ：　時間ステップ

したがって，分子ｉが他の分子と衝突する確率は

$$P_i = \sum_{j=1}^{N} P_{ij} \qquad (2.\ 3)$$

となる．いま，区間［0，1］をN等分し確率P_{ij}を Fig. 2.2 のように割り当てる．一様乱数を１つ発生させ，この乱数が区間［(j-1)/N，j/N］にあれば分子ｊと衝突する可能性がある．ここでは乱数がその区間の右側，すなわち確率P_{ij}の部分にあれば，分子ｉは分子ｊと衝突し，そうでなければ，分子ｉは衝突しないと決める．衝突する場合には，運動量保存則とエネルギー保存則を満たすように分子ｉの速度を更新する．この手続きを衝突セル内のすべての模擬分子ｉに対して行う．これらを，すべての衝突セルに対して行うが，各衝突セル内の衝突の計算は独立に行うことができる．すべての衝突計算が終了した後，すべての模擬分子の速度が一斉に更新される．

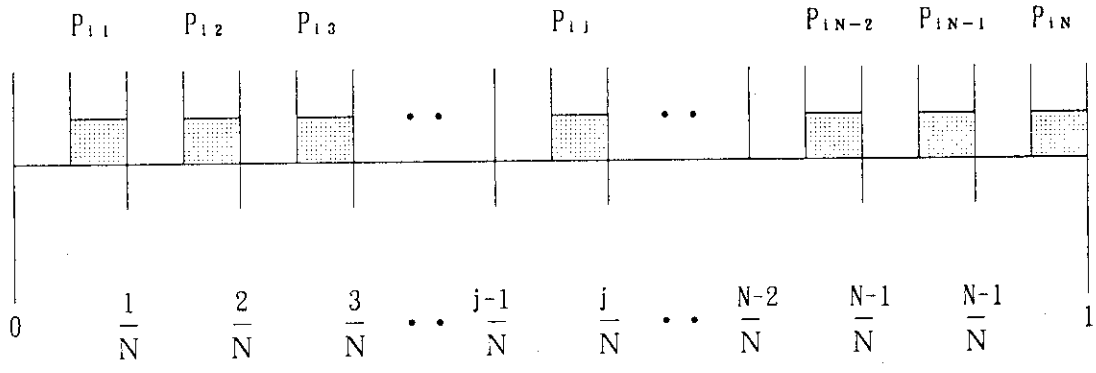


Fig. 2.2 Distribution of the probabilities P_{ij} for the modified Nambu method

2．3　衝突サブセル

衝突セルの大きさは，その場における平均自由行程λ程度またはそれ以下にとることが望ましいが，このλを事前に正確に知ることはできない．そのため，従来のＤＳＭＣコードでは，適当なλを推定して，その値に応じて衝突セルの大きさを決めていた．また，統計量をとるためのサンプリングセルも衝突セルと同じ大きさにしていた．しかし，原子蒸気噴流（またはジェット）のように真空中にガスが拡散する場合には，解析領域において流れが連続流から希薄流れに変化し，λが非常に大きく変化するため，従来の手法では正しくセル分割をすることは困難である．

今回ベクトル化の対象としたコードでは，ＤＳＭＣ法の各時間ステップで，λに応じて自動的に衝突セルを衝突サブセルに再分割する方法を用いている．また，衝突セルとサンプリングセルは同じ大きさにとらず，各々を独立に取っている[4]．

衝突セルを自動的に衝突サブセルに再分割する手順は以下の通りである．

(1) 計算領域を推定されるλの２～５倍程度の大きさの衝突セルに分割する．衝突セルは同じ大きさでなくても良い．最初に設定された衝突セルは，シミュレーションを通して固定されている．

(2) 各衝突セルに含まれるサンプリングセルの統計量から，その衝突セルの平均自由行程λを計算する．

(3) (2) で求めたλ程度に衝突セルを衝突サブセルに分割する．ただし，衝突サブセルの大きさがサンプリングセルよりも大きい場合には，衝突サブセルをサンプリングセルの大きさにとる．

(4) 各衝突サブセルについて衝突アルゴリズムを適用する．

(5) 各時間ステップで，(2) ～(4) を繰り返す．

この方法により，事前に衝突セルを正しく取ることができる．ただし，サンプリングセルには統計量を取るための最小限の模擬分子が含まれている必要がある．

３．　コードの解析

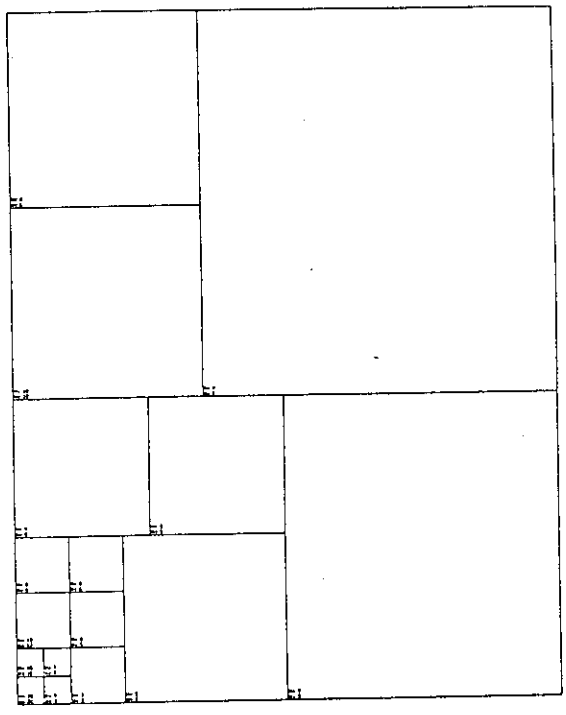
３．１　コードの静的解析

３．１．１　解析対象

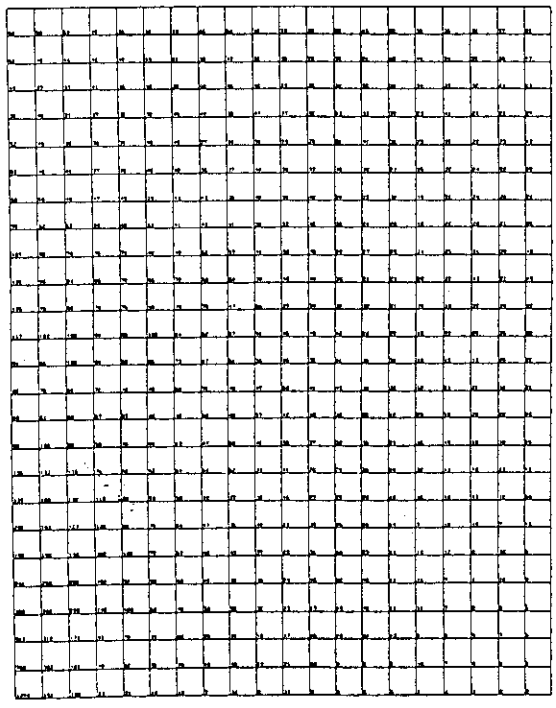
ベクトル化するにあたって用いたデータは，幅 10mm のスリットから真空中に蒸発するガドリニウム原子蒸気の流れを解析するためのものである．計算は，問題の軸対称性を考慮して，２次元の領域（250mm*200mm)で計算を行う．境界条件は，対称軸では鏡面反射，その他の境界では真空を設定している．（噴出源の温度および圧力は，それぞれ 2500 [K]，628 [Pa] であり，一定であると仮定している．また，噴出源のクヌーセン数は 0.037である．）

この問題では，解析領域において，平均自由行程が急激に変化すると考えられるが，2.3 節で説明したセルの自動的再分割により衝突セルが正しく設定される．

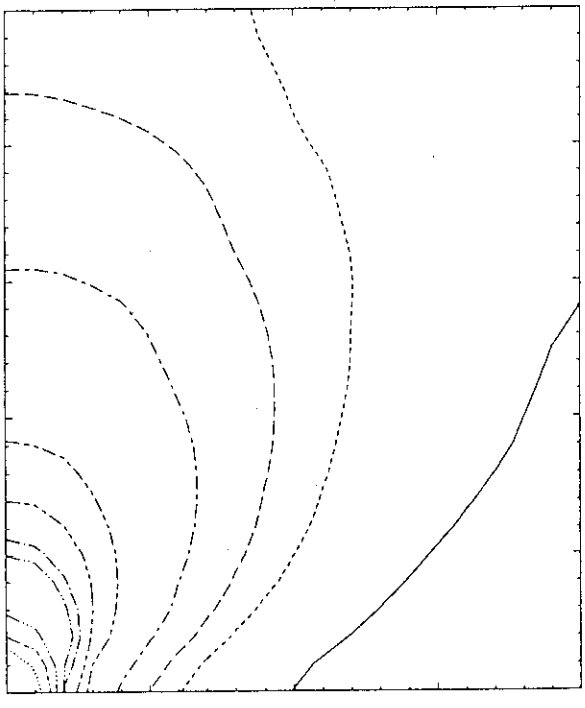
サンプリングセルと衝突セルは等しくとられていない．Fig. 3.1 (a),(b),(c),(d)にそれぞれ衝突セル，サンプリングセル，等密度線，速度ベクトルの例を示す．



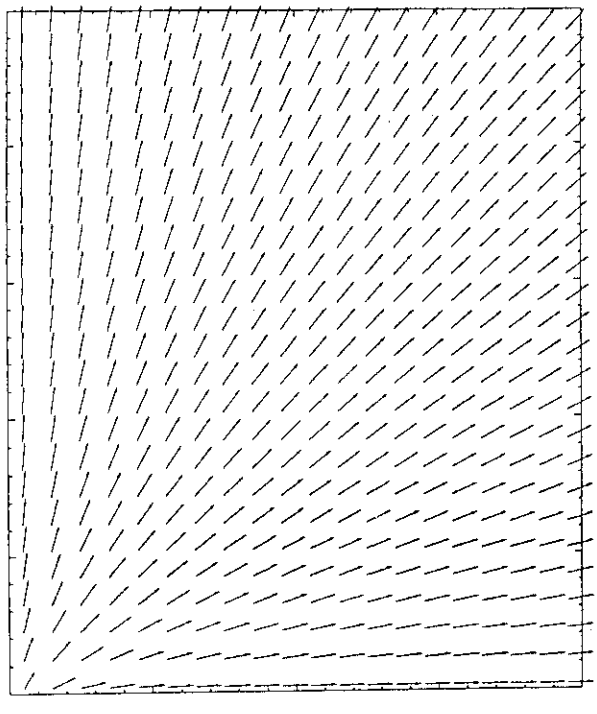
(a) Collision cells



(b) Sampling cells



(c) Density contours



(d) Velocity vectors

Fig 3.1 Expansion of gas into vacuum space

3．1．2　コードの構造

コードの tree 構造を Fig. 3.2 に示す．時間発展の計算は，SIMULA以下のサブルーチンで行われている．

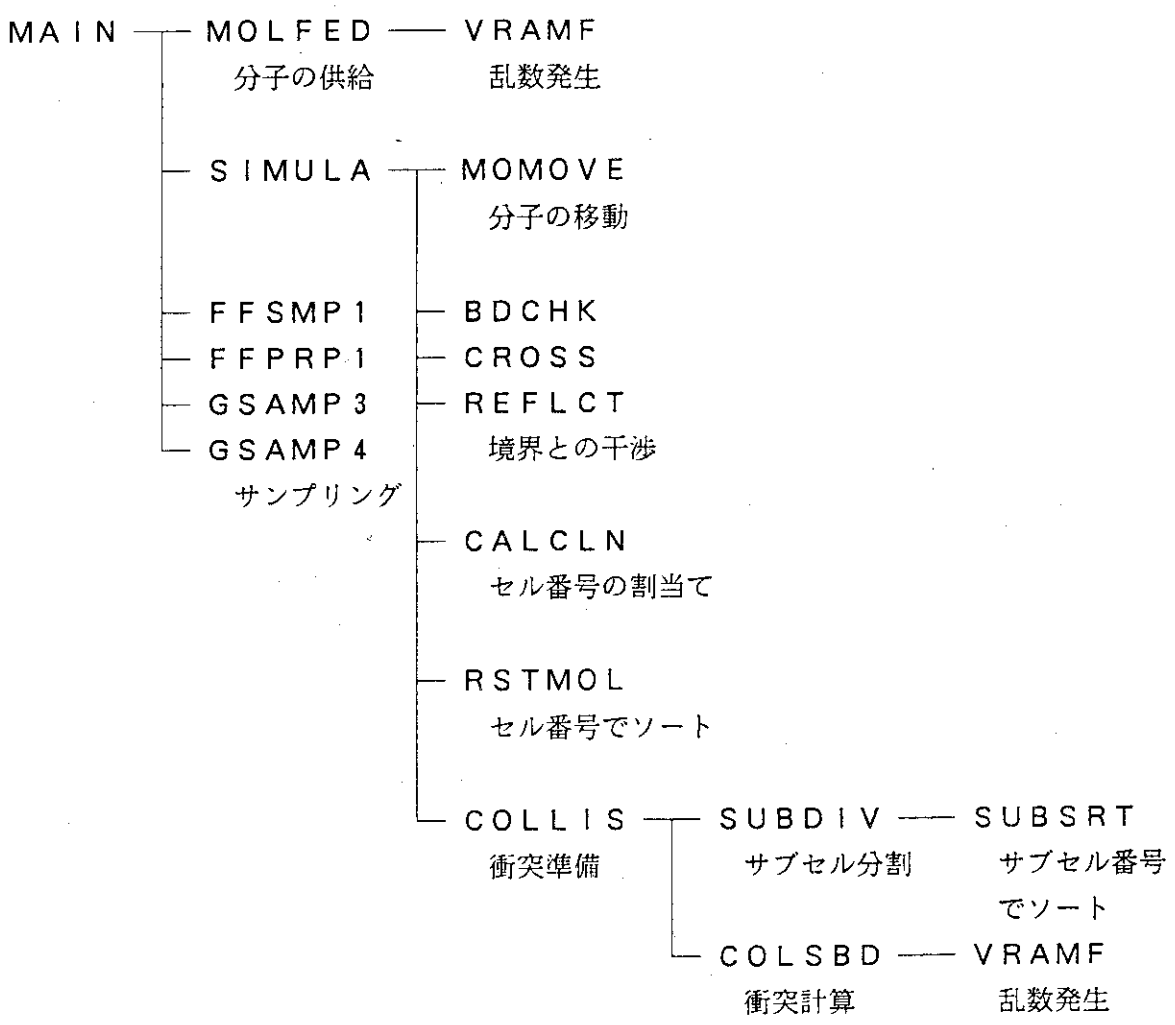


Fig. 3.2　The main tree structure

3．2　コードの動的解析

3．2．1　ルーチン単位の分析

コードの動的挙動を分析するために，主要なルーチンについてＣＰＵ時間の測定を行った．測定はまずスカラ実行で行い，次に現状のコードのベクトル処理効率をみるために，コードをそのままベクトル実行した場合のＣＰＵ時間も併せて測定した．

測定結果を，Table 3.1 に示す．

Table 3.1　Dynamic behavior for main routines

	スカラ実行 CPU時間 t_s （比率）	ベクトル実行 CPU時間 t_v （比率）	スカラ/ベクトル比 t_s/t_v
MAIN	2451.90 sec	666.28 sec	3.68
COLLIS	1725.30 (70.37 %)	199.48 (27.28 %)	8.65
CALCLN	433.60 (17.68 %)	290.43 (39.71 %)	1.49
SUBDIV	136.45 (5.57 %)	90.55 (12.38 %)	1.51
COLSBD	37.61 (1.53 %)	17.59 (2.41 %)	2.14
SUBSRT	35.34 (1.44 %)	13.99 (1.92 %)	2.53
RSTMOL	30.68 (1.25 %)	20.79 (2.58 %)	1.48
VRAMF	11.74 (0.48 %)	0.40 (0.06 %)	29.35
GSAMP3	7.62 (0.31 %)	1.67 (0.23 %)	4.56
BDCHK	6.04 (0.25 %)	0.60 (0.09 %)	10.06
GSAMP4	4.86 (0.20 %)	1.13 (0.15 %)	4.30
MOMOVE	4.32 (0.18 %)	0.20 (0.03 %)	21.60
REFLCT	3.41 (0.14 %)	0.38 (0.05 %)	8.97
CROSS	2.18 (0.09 %)	9.79 (1.34 %)	0.22

これより，COLLIS，CALCLN，SUBDIVの上位３ルーチンで全体の計算コストの大部分を占めていることがわかる．すなわち，スカラ実行で 93.63 %，ベクトル実行で 87.12 %である．これより以下ではルーチン COLLIS, CALCLN, SUBDIV を重点的にベクトル化を行うことにする．

3．2．2　ループ単位の分析

前節で計算コストの高い3つのルーチンがわかったが，それらについて更にＤＯループ単位の分析を行った．その結果，どのルーチンも処理の中心部である2重ＤＯループがそれぞれのルーチンの計算コストの殆どを占めていることがわかった．分析結果を Table 3.2　に示す．

Table 3.2　Analysis for main loops

	所　属 ルーチン	ループ 番　号	ルーチン内 計算コスト	物　理　的 処　理　内　容
ループ1	COLLIS	70, 80	96.0 %	衝突計算の準備
ループ2	CALCLN	100,200	99.8 %	サンプリングセルの割り当て
ループ3	SUBDIV	151,155	98.7 %	衝突サブセルの割り当て

この結果より，前記の3つのルーチンの高速化は，これら3つのループの高速化と等価であることがわかる．これら3つのループについての動的挙動の分析結果を Table 3.3　に示す．

Table 3.3　Dynamic behavior for main loops

	スカラ実行 CPU時間 t_s　（比率）	ベクトル実行 CPU時間 t_v　（比率）	ｽｶﾗ/ﾍﾞｸﾄﾙ比 t_s/t_v
ループ1	1673.25　(68.24 %)	190.89　(28.65 %)	8.77
ループ2	432.60　(17.64 %)	290.30　(43.57 %)	1.49
ループ3	132.09　(5.39 %)	86.15　(12.93 %)	1.53

これより，スカラ実行ではループ1の比率が圧倒的に大きいが，ベクトル実行ではループ2，3の比率が大きくなることがわかる．その原因は，ループ2，3の ｽｶﾗ/ﾍﾞｸﾄﾙ 速度比が 1.5倍程度の低い値しか得られていないからである．これより，コードの高速化における課題の1つとして，ループ2，3のベクトル性能の向上があげられる．また，ループ1はスカラ実行において 70 % 近くの計算コストを占めている．これより，もう1つの課題として，ループ1の処理手順の最適化があげられる．

4. コードの高速化

4. 1 衝突準備処理

ルーチン COLLIS にあるループ1 (DO 70 & DO 80) は，コード全体の実行時間の 68.2 % を占める．このループは，衝突計算を行うために，注目する衝突セルに属するサンプリングセルを選び出し，セル内にある分子を作業配列に格納する処理をしている．

ループ1のコーディングを Fig. 4. 1 に示す．

```
      DO 1000 NCC = 1,NCCX
        ..........
        MCL = 0

        DO 80 I=1,IXMAX*IYMAX
          IF( TC6(I) .EQ. REAL(NCC) ) THEN

            DO 70 J=1,MLAST
              IF( TM6(J) .EQ. REAL(I) ) THEN

                MCL = MCL + 1
                CM1(MCL) = TM1(J)
                CM2(MCL) = TM2(J)
                CM3(MCL) = TM3(J)
                CM4(MCL) = TM4(J)
                CM5(MCL) = TM5(J)
                CM6(MCL) = TM6(J)
                CM7(MCL) = 0.
                CM8(MCL) = REAL(J)

              END IF
   70       CONTINUE

          END IF
   80   CONTINUE
        ..........
 1000 CONTINUE
```

各記号の意味

NCCX : 衝突セルの総数
MCL : カウンタ
IXMAX : x方向のサンプリングセル数
IYMAX : y方向のサンプリングセル数
MLAST : 全分子数

TC6(I) : i番目のサンプリングセルが所属する衝突セルの番号
TM6(J) : j番目の分子が所属するサンプリングセルの番号

TM1(J)~TM5(J) : 分子の座標と速度
CM1(J)~CM8(J) : 作業配列

Fig. 4. 1 The original coding of the LOOP-1

この処理手順は以下の通りである．

ｉ）NCC 番目の衝突セルについて計算する．（ DO 1000 ）
ii）全てのサンプリングセルについて，衝突セル NCC に属するか判定する．（ DO 80 ）
iii）所属するサンプリングセルがあれば，全ての分子について，サンプリングセルｉに属するか判定する．（ DO 70 ）
iv）所属する分子があれば，ローカルインデックス MCL をインクリメントし，グローバル配列 TM1 ~ TM6 をローカル配列 CM1 ~ CM6 に格納する．
CM7，CM8 はサブセル分割のための作業配列である．

オリジナルのコーディングには２つのＩＦ文が存在するが，その実行回数を計算すると以下のようになる．

（１番目のＩＦ文）＋（２番目のＩＦ文）
＝（衝突ｾﾙ数）＊（ｻﾝﾌﾟﾘﾝｸﾞｾﾙ数）＋（ｻﾝﾌﾟﾘﾝｸﾞｾﾙ数）＊（全分子数）
＝ NCCX * IXMAX * IYMAX ＋ IXMAX * IYMAX * MLAST
＝ 16 * 20 * 25 ＋ 20 * 25 * 25600
＝ 128,080,000

しかし，この処理の物理的な内容は，ある衝突セルに属する分子を，作業配列に移すことである．分子と衝突セルの対応は一意に定まることを考えると，この実行回数はもっと減らすことができる．このループ１の改善案として３つの方法を試みた．以下にそれらの考え方とコーディング方法を示す．

（1）方法1

この方法は，TM6,TC6 というリストを2重に間接参照することにより，分子と衝突セルとの対応を示すリスト IBCELL を作成する方法である．この方法では，ＩＦ文の実行回数は，（衝突セル数）＊（全分子数）に削減される．

方法1のコーディングを Fig. 4. 2 に示す．

```
      DO 1000 NCC = 1,NCCX
        ..............
        MCL = 0

        DO 70 J=1,MLAST

          IBCELL = INT( TC6( INT( TM6(J) ) ) )
          IF( IBCELL .EQ. NCC ) THEN

            MCL = MCL + 1
            CM1(MCL) = TM1(J)
            CM2(MCL) = TM2(J)
            CM3(MCL) = TM3(J)
            CM4(MCL) = TM4(J)
            CM5(MCL) = TM5(J)
            CM6(MCL) = TM6(J)
            CM7(MCL) = 0.
            CM8(MCL) = REAL(J)

          END IF
   70   CONTINUE
        ..............
 1000 CONTINUE
```

Fig. 4. 2 The 1st modified coding of the LOOP-1

（2）方法2

この方法は，方法1における分子と衝突セルの対応を示すリストを，IBWORK　という配列にする方法である．この方法では，ＩＦ文の実行回数が（衝突セル数）＊（全分子数）に削減されるのに加え，リストの作成回数が方法1では（衝突セル数）＊（全分子数）であったのに対し，方法2では（全分子数）に削減される．

方法2のコーディングを Fig. 4. 3　に示す．

```
      DO 10 J=1,MLAST
         IBWORK(J) = INT( TC6( INT( TM6(J) ) ) )
   10 CONTINUE

      DO 1000 NCC = 1,NCCX
         ..............
         MCL = 0

         DO 70 J=1,MLAST
            IF( IBWORK(J) .EQ. NCC ) THEN

               MCL = MCL + 1
               CM1(MCL) = TM1(J)
               CM2(MCL) = TM2(J)
               CM3(MCL) = TM3(J)
               CM4(MCL) = TM4(J)
               CM5(MCL) = TM5(J)
               CM6(MCL) = TM6(J)
               CM7(MCL) = 0.
               CM8(MCL) = REAL(J)

            END IF
   70    CONTINUE
         ..............
  1000 CONTINUE
```

Fig. 4. 3　The 2nd modified coding of the LOOP-1

（3）方法3

分子がサンプリングセルの順序でソートされていること，及び，サンプリングセルと衝突セルの関係が不変であること，の2つを利用すると，演算量を（全分子数）に減らすことができる．これを方法3と呼び，そのコーディングを Fig. 4. 4 に示す．

方法3は以下の3つのステップからなる．

(Step 1)

まず時間発展のループに入る前に，衝突セルとサンプリングセルの関係をリストに格納する．すなわち，NCC 番目の衝突セルに属するサンプリングセルの個数は，LASTC(NCC) であり，ICNT 番目のサンプリングセルの番号は，LSMPC(ICNT,NCC) である．

Step 1 のコーディングを Fig. 4. 4 (a) に示す．

```
      DO 20 NCC=1,NCCX
         ICNT = 0
         DO 10 N=1,IXMAX*IYMAX
            IF( TC6(N) .EQ. REAL(NCC) ) THEN
               ICNT = ICNT + 1
               LSMPC(ICNT,NCC) = N
            END IF
   10    CONTINUE
         LASTC(NCC) = ICNT
   20 CONTINUE
```

各記号の意味

ICNT	:	カウンタ
TC6(n)	:	サンプリングセルの所属する衝突セルの番号
LSMPC(i,n)	:	サンプリングセルの番号
LASTC(n)	:	サンプリングセルの個数

Fig. 4. 4 (a) The 3rd modified coding of the LOOP-1 (Step 1)

(Step 2)

次に，ソーティングのルーチン RSTMOL において，各サンプリングセルにおける分子番号の基底値を MBASEにセットする．基底値とは，（各サンプリングセルで最小の分子番号 — 1）を意味する．この基底値を用いると，ローカルなインデックスからグローバルなインデックスを参照することができる．

Step 2 のコーディングを Fig. 4. 4 (b) に示す．

```
   MBASE(1) = 0

   DO 30 I=1,IXMAX*IYMAX
      MBASE(I) = ITA(I-1)
30 CONTINUE
```

各記号の意味

ITA(i)	:	分子数の累計値
MBASE(i)	:	分子番号の基底値

Fig. 4. 4 (b) The 3rd modified coding of the LOOP-1 (Step 2)

(Step 3)

最後にルーチン COLLIS で作業配列への移動を行う．各衝突セルについて，
（衝突セル内のサンプリングセル数）＊（サンプリングセルにある分子数）
の演算を行う．
Step 3 のコーディングを Fig. 4. 4 (c) に示す．

```
   DO 1000 NCC=1,NCCX
      ..........
      MCL = 0
      DO 80 L=1,LASTC(NCC)
         LL = LSMPC(L,NCC)

         DO 70 M=1,INT( TC5(LL) )
            MCL = MCL + 1
            MM  = M + MBASE(LL)

            CM1(MCL) = TM1(MM)
            CM2(MCL) = TM2(MM)
            CM3(MCL) = TM3(MM)
            CM4(MCL) = TM4(MM)
            CM5(MCL) = TM5(MM)
            CM6(MCL) = TM6(MM)
            CM7(MCL) = 0.
            CM8(MCL) = REAL(MM)
70       CONTINUE

80    CONTINUE
      ..........
1000 CONTINUE
```

各記号の意味

MCL	:	カウンタ
LL	:	サンプリングセル番号
M	:	ローカルな分子番号
MM	:	グローバルな分子番号
LASTC(n)	:	サンプリングセルの個数
LSMPC(i,n)	:	サンプリングセルの番号
TC5(LL)	:	サンプリングセルの分子数
MBASE(LL)	:	サンプリングセルの基底値

Fig. 4. 4 (c) The 3rd modified coding of the LOOP-1 (Step 3)

4．2　サンプリングセルの割り当て処理

ループ２は，ルーチン CALCLN にある２重ＤＯループ（ DO 100 & DO 200 ） であり，コード全体の計算時間の 17.6 % を占めている．このループは，全分子について，その座標に応じて分子の存在するサンプリングセル番号を決める処理をしている．

ループ２のコーディングを Fig. 4. 5 に示す．

```
      DO 200  L = 1,MLAST
      DO 100  K = 1,IXMAX*IYMAX

         IF ( (TM1(L).GE.TC1(K)) .AND.
     .        (TM1(L).LE.TC3(K)) .AND.
     .        (TM2(L).GE.TC2(K)) .AND.
     .        (TM2(L).LE.TC4(K)) ) THEN
            TM6(L) = REAL(K)
            GO TO 200
         ENDIF

  100 CONTINUE
  200 CONTINUE
```

各記号の意味

MLAST　：全分子数
IXMAX　：ｘ方向のサンプリングセル数
IYMAX　：ｙ方向のサンプリングセル数
TM1(L)：Ｌ番目の分子のｘ座標
TM2(L)：Ｌ番目の分子のｙ座標

TC1(K)：ｋ番目のサンプリングセルの左側境界の座標
TC2(K)：同下側境界の座標
TC3(K)：同右側境界の座標
TC4(K)：同上側境界の座標

TM6(L)：Ｌ番目の分子が所属するサンプリングセルの番号

Fig. 4. 5　The original coding of the LOOP-2

処理手順は以下の通りである．

ｉ）Ｌ番目の分子について注目する．（ DO 200 ）
ii）注目している分子がどのサンプリングセルに所属しているかを分子の座標とセル境界の座標の大小を比較することにより判定する．（ DO 100 ）
iii）所属するセルがみつかったら TM6 にセル番号を格納し，外側のループに飛び，次の分子の計算に移る．（ GO TO 200 ）

このコーディングの特徴は，内側から外側に飛び出しのある２重ＤＯループになっていることである．ベクトル処理可能ではあるが，得られる性能は非常に低い．

また，オリジナルのコーディングによるループ２の実行回数は，以下のようになる．

（全分子数）　＊　（サンプリングセル数）　／　２
＝　MLAST　＊　IXMAX　＊　IYMAX　／　２
＝　２５６００　＊　２０　＊　２５　／　２
＝　６，４００，０００

物理的に考えると，ある１つの分子は必ずあるセルに所属するから，分子とセルの一意的な対応関係をつくってやれば，演算量は，全分子数　MLAST ですむはずである．
このループ２の改善案を以下に示す．

今，サンプリングセルは，ｘ方向が幅ＤＬ，ｙ方向が高さＤＨで等間隔に区切られている．これより，ＤＬ，ＤＨを用いて分子の座標（ TM1, TM2 ）から　ｘ，ｙ方向のインデックスｉ，ｊを計算することができ，さらにｉ，ｊからセル番号ｋを計算できる．
この対応関係を用いた改善案のコーディングを Fig. 4. 6　に示す．

```
      DO 200 L = 1,MLAST

         I = INT ( TM1(L) / DL + 1.0 )
         J = INT ( TM2(L) / DH + 1.0 )

         K = (I-1)*IYMAX + J

         TM6(L) = REAL(K)

  200 CONTINUE
```

j

IYMAX = 4	4	8	12	16	20
3	3	7	11	15	19
2	2	6	10	14	18
1	1	5	9	13	17
	1	2	3	4	5　i

Fig. 4. 6　The modified coding of the LOOP-2

この方法における問題点として，分子が計算領域外にある場合には，正しいセル番号が計算されないことがあげられる．この問題を解決するためには，境界条件処理の部分で，領域外にある分子をあらかじめ除去してやればよい．

4．3　衝突サブセルの割り当て処理（ループ３）

ループ３は，ルーチン SUBDIV にある２重ＤＯループ（ DO 151 & DO 155 ） であり，コード全体の計算時間の 5.4% を占めている．このループでは，各衝突セルに内にある分子について，その座標に応じて分子の存在する衝突サブセル番号を決める処理をしている．

ループ３のコーディングを Fig. 4. 7 に示す．

```
      DO 155  I = 1,INT(CTC5(NCC))
      DO 151  J = 1,NSBW(NCC)*NSBH(NCC)

         IF ( (CM1(I).GE.SC1(J)) .AND.
     .        (CM1(I).LE.SC3(J)) .AND.
     .        (CM2(I).GE.SC2(J)) .AND.
     .        (CM2(I).LE.SC4(J)) ) THEN
            CM7(I) = REAL(J)
            GO TO 155
         ENDIF

  151 CONTINUE
  155 CONTINUE
```

各記号の意味

CTC5(NCC)	: NCC 番目の衝突セルの分子数
NSBW(NCC)	: 同ｘ方向の衝突サブセル数
NSBH(NCC)	: 同ｙ方向の衝突サブセル数
CM1(I)	: ｉ番目の分子のｘ座標
CM2(I)	: ｉ番目の分子のｙ座標
SC1(J)	: ｊ番目の衝突サブセルの左側境界の座標
SC2(J)	: 同下側境界の座標
SC3(J)	: 同右側境界の座標
SC4(J)	: 同上側境界の座標
CM7(I)	: ｉ番目の分子が所属する衝突サブセルの番号

Fig. 4. 7 The original coding of the LOOP-3

このループ３は，ループ２と同じアルゴリズムであるから，同様の改善をすればよいことがわかる．ただ１つ注意しなければならないのは，インデックスｉ，ｊを計算するときに，分子の座標（ CM1, CM2 ）から衝突セルの原点の座標（ CTC1, CTC2 ）を引いた相対座標を用いなければならない点である．

また，改善案では，衝突サブセルのｘ方向の幅 CDDL とｙ方向の幅 CDDH を用いてインデックスを計算する．オリジナルのコーディングでは，SC1 ~ SC4 という各衝突サブセルの境界の座標を毎回計算していたが，改善案ではこれらの計算は不要になり，計算時間が一層短縮される．

この改善案のコーディングを Fig. 4. 8 に示す．

```
      DO 155 L = 1, INT(CTC5(NCC))

         I = INT ( (CM1(L) - CTC1(NCC)) / CDDL + 1.0 )
         J = INT ( (CM2(L) - CTC2(NCC)) / CDDH + 1.0 )

         K = (I-1)*NSBH(NCC) + J

         CM7(L) = REAL(K)

  155 CONTINUE
```

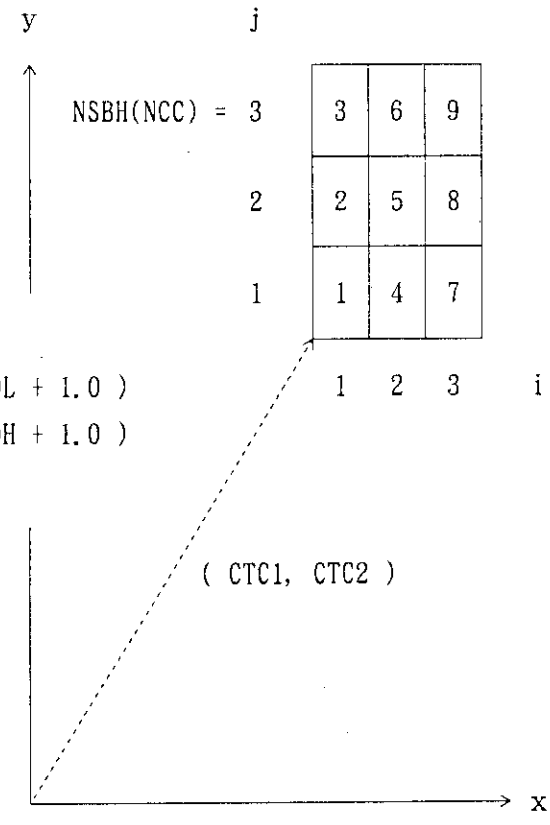


Fig. 4. 8 The modified coding of the LOOP-3

4．4　その他の処理

4．4．1　真空境界条件処理　（ルーチン RSTMOL）

ルーチン RSTMOL は，分子をその所属するサンプリングセル番号に従ってソートするルーチンである．更に，このルーチンは，真空境界条件が設定されている場合には，真空境界における分子の除去も同時に行うように設計されている．すなわち，真空境界にある分子に対しては，サンプリングセル番号として“0”が代入されていて．ソーティングの際にサンプリングセルの番号が“0”かどうかを判定し，成立すればその分子を除去するという処理を行っている．

しかし，4.2 節で述べたように，ループ2の改善の際に，分子にサンプリングセル番号を与える前に，計算領域外にある分子をあらかじめ除去しておく必要がある．すなわち，ループ2の前処理の時点で分子の除去を行っておけば，ソーティングの時点では境界条件処理を行う必要はなくなることがわかる．

ソーティングの処理に，真空境界条件を含めた場合と分離させた場合についての性能評価については　5.1.2 節で述べる．

4．4．2　反射境界条件処理　（ルーチン CROOS）

ルーチン　CROSS は，反射境界条件が設定されている場合に，境界における分子の反射位置を計算するルーチンである．

このルーチンの核となるＤＯループでは，真率が非常に低いＩＦ文が2重のネストを形成している．このため，実行ベクトル長が小さくなり，またマスクデータ作成のためのオーバーヘッドが大きくなるため，ベクトル処理を行った場合，スカラ処理よりも遅くなったと考えられる．

このＤＯループについて，いくつかのコーディングの改良を試みたが，スカラ実行より速くすることは出来なかった．このため，解決策としては，最適化制御行でベクトル処理を抑止することにした．

5. 高速化の結果

5. 1 ベンチマークテスト

5. 1. 1 ループ1における性能評価

この節では，4．1節で示したループ1に対する3つの改善案についての速度性能の評価を行った結果を示す．テストデータとしては，シミュレーション時間をオリジナルデータの 1/10 とした．

ループ1のＣＰＵ時間の測定の結果を Table 5.1に，改善案のオリジナル版に対する速度向上率を Table 5.2にそれぞれ示す．

Table 5. 1 The comparison of the CPU time among original and 3 modification methods for LOOP-1

	スカラ実行 ＣＰＵ時間 T_s	ベクトル実行 ＣＰＵ時間 T_v	ｽｶﾗ/ﾍﾞｸﾄﾙ 比 T_s/T_v
オリジナル	T_{s0} = 145.92 sec	T_{v0} = 17.39 sec	8.39
方法1	T_{s1} = 12.24	T_{v1} = 2.08	5.90
方法2	T_{s2} = 5.30	T_{v2} = 0.81	6.59
方法3	T_{s3} = 1.40	T_{v3} = 0.13	10.81

Table 5. 2 The speedup ratio among 3 modification methods for LOOP-1

	スカラ実行における 速度向上率 T_{s0}/T_{si}	ベクトル実行における 速度向上率 T_{v0}/T_{vi}	ループ全体としての 速度向上率 T_{s0}/T_{vi}
方法1	r_{s1} = 11.92	r_{v1} = 8.38	r_{t1} = 70.29
方法2	r_{s2} = 27.52	r_{v2} = 21.60	r_{t2} = 181.27
方法3	r_{s3} = 104.60	r_{v3} = 134.81	r_{t3} = 1131.16

まず，スカラ実行における速度向上率 r_{si} を比較すると，方法3が最も優れており，オリジナルと比べて 100倍以上の速度向上が得られている．方法2が約27倍，方法1が約12倍である．これは，4.1 節で述べたＩＦ文その他の演算量の比較から考えて妥当な結果であると言える．これより，物理的な処理内容は同じでも，コーディングの方法によって処理時間は大きく異なることがわかる．

次に，ベクトル実行における速度向上率 r_{vi} を比較すると，方法1および2では r_{vi}が r_{si} と比べて低下しているのに対し，方法3では逆に向上している．これは，ｽｶﾗ/ﾍﾞｸﾄﾙ比を比較することにより説明できる．すなわち，オリジナルでは，ｽｶﾗ/ﾍﾞｸﾄﾙ比が 8.39 倍であるのに対し，方法1および2では，それぞれ 5.90 倍， 6.59 倍と下回っている．ところが方法3では，10.81 倍とオリジナルを上回る速度向上率が得られている．これより，方法3は，ベクトル実行においても高い速度性能を持つことがわかる．

方法3は，4.1 節で示したように，３つのステップにより構成される．各ステップのＣＰＵ時間測定により，方法3における前処理 step 1, step 2 は，ほとんどＣＰＵ時間を消費しないことがわかった．

ループ1の高速化についてまとめると，３つの改善案のうちでは，方法3が最も高い速度性能の向上を示し，更にベクトル処理にも適していることがわかった．

参考のために，３つの改善案の特徴について， Table 5. 3 にまとめて示す．

Table 5. 3 The comparison of the feature among 3 modification methods for LOOP-1

	速度向上率	メモリ増加量	高速化の手法
方法1	７０ 倍	なし	普通
方法2	１８０ 倍	２００ KB	普通
方法3	１１３０ 倍	１５ KB	やや複雑

5．1．2　ループ２における性能評価

ループ２では，4.2 節で示したコーディングの改良に伴い，模擬分子が計算領域の外にある場合，分子の除去という前処理を行う必要がある．前節と同じテストデータを用いて，前処理を含めたループ２のＣＰＵ時間を測定した．

オリジナル版と修正版の測定結果を，それぞれ Table 5.4，5.5 に示す．

Table 5. 4　The CPU time of original coding for LOOP-2

	スカラ実行 ＣＰＵ時間　T_{so}	ベクトル実行 ＣＰＵ時間　T_{vo}	ｽｶﾗ/ﾍﾞｸﾄﾙ 比 T_{so}/T_{vo}
前処理	0.0　msec	0.0　msec	———
ループ２	43,394.42	26,390.20	1.90
合計	43,394.42	26,390.20	1.90

Table 5. 5　The CPU time of modified coding for LOOP-2

	スカラ実行 ＣＰＵ時間　T_{sm}	ベクトル実行 ＣＰＵ時間　T_{vm}	ｽｶﾗ/ﾍﾞｸﾄﾙ 比 T_{sm}/T_{vm}
前処理	1,272.49　msec	39.40　msec	32.29
ループ２	2,733.01	331.69	8.24
合計	4,005.49	371.09	10.79

これより，オリジナル版と修正版の合計のＣＰＵ時間を比較すると，以下のようになる．

スカラ　実行　：　T_{so}/T_{sm} ＝ 12.53
ベクトル実行　：　T_{vo}/T_{vm} ＝ 71.12

すなわち，修正版は，スカラ実行，ベクトル実行でともにオリジナル版より高い速度性能を持つことがわかる．また，ベクトル実行において特に高い性能を示すこともわかる．

また，ループ２の前処理として，計算領域外にある分子を除去することにより，現在ソーティングのルーチン RSTMOL で行っている真空境界条件の判定を除くことができる．

ソーティングのルーチン RSTMOL について，真空境界条件を含む場合（オリジナル版）と含まない場合（修正版）について，前節とおなじテストデータを用いてＣＰＵ時間を測定した．

測定の結果を，Table 5.6 に示す．

Table 5. 6　The comparison of the CPU time for RSTMOL concerned with the vacuum boundary condition

	スカラ実行 ＣＰＵ時間　T_s	ベクトル実行 ＣＰＵ時間　T_v	ｽｶﾗ/ﾍﾞｸﾄﾙ 比 T_s/T_v
オリジナル版	3,089.47　msec	1,889.43　msec	1.64
修正版	2,865.75	1,351.00	2.12

これより，修正版は，スカラ実行，ベクトル実行でともにオリジナル版より高い速度性能を持つことがわかる．また，ｽｶﾗ/ﾍﾞｸﾄﾙ比を比較すると，修正版の方がより効率的なベクトル処理を行っていることがわかる．

5．1．3　ループ3における性能評価

ループ3では，4.3 節で示したコーディングの改良に伴い，各衝突サブセルの境界の座標を毎回計算するという前処理を行う必要がなくなった．前節と同じテストデータを用いて，前処理を含めたループ3のＣＰＵ時間を測定した．

オリジナル版と修正版の測定結果を，それぞれ Table 5.7，5.8 に示す．

Table 5. 7　The CPU time of original coding for LOOP-3

	スカラ実行 ＣＰＵ時間　T_{so}	ベクトル実行 ＣＰＵ時間　T_{vo}	ｽｶﾗ/ﾍﾞｸﾄﾙ 比 T_{so}/T_{vo}
前処理	85.83　msec	65.23　msec	1.32
ループ3	13,745.53	8,010.97	1.72
合計	13,833.36	8,076.20	1.71

Table 5. 8　The CPU time of modified coding for LOOP-3

	スカラ実行 ＣＰＵ時間　T_{sm}	ベクトル実行 ＣＰＵ時間　T_{vm}	ｽｶﾗ/ﾍﾞｸﾄﾙ 比 T_{sm}/T_{vm}
前処理	0.0　msec	0.0　msec	———
ループ3	1,346.76	61.90	21.76
合計	1,346.76	61.90	21.76

これより，オリジナル版と修正版の合計のＣＰＵ時間を比較すると，以下のようになる．

スカラ　実行　：　T_{so}/T_{sm} ＝ 10.27

ベクトル実行　：　T_{vo}/T_{vm} ＝ 130.48

すなわち，修正版は，スカラ実行，ベクトル実行でともにオリジナル版より高い速度性能を持つことがわかる．また，ベクトル実行において特に高い性能を示すこともわかる．

5．2　高速化の結果

高速化の効果をみるために，オリジナル版コード・修正版コードの両方について，スカラ実行時・ベクトル実行時のＣＰＵ時間の測定を行った．前節における速度性能の評価結果をふまえ，ループ１の高速化については，方法３を使用した．

測定の結果を Table 5.9 に示す．

Table 5.9 The comparison of the CPU time between the original code and the modified code

		オリジナル版	修正版
VP100 スカラ実行	CPU 時間*1	37 min 13.28 sec	5 min 01.37 sec
VP100 ベクトル実行	CPU 時間*2	10 min 04.48 sec	1 min 27.84 sec
	VU 時間	8 min 42.22 sec	43.46 sec
VP2600 スカラ実行	CPU 時間*3	11 min 46.34 sec	1 min 14.51 sec
VP2600 ベクトル実行	CPU 時間*4	1 min 56.78 sec	17.36 sec
	VU 時間	1 min 37.46 sec	6.61 sec

*1 ： コンパイルオプション OPT(3) でコンパイル，VP100 で実行
*2 ： コンパイルオプション VP(100) でコンパイル，VP100 で実行
*3 ： コンパイルオプション OPT(3) でコンパイル，VP2600で実行
*4 ： コンパイルオプション VP(400),VPEでコンパイル，VP2600で実行

これより，高速化の効果を計算した結果を Table 5.10 に示す．

Table 5.10 Speed up ratio

	VP 100	VP 2600
ｵﾘｼﾞﾅﾙ版 ｽｶﾗ実行時間 / 修正版 ｽｶﾗ実行時間	7.41 倍	9.48 倍
ｵﾘｼﾞﾅﾙ版 ｽｶﾗ実行時間 / 修正版ﾍﾞｸﾄﾙ実行時間	25.42 倍	40.69 倍

これより，スカラ処理の最適化による高速化率は７～９倍，ベクトル処理の最適化も含めた高速化率は25～40倍であることがわかる．

しかし，修正版コード自身のベクトル性能（ｽｶﾗ/ﾍﾞｸﾄﾙ比）についてみると，VP100 で3.43倍，VP2600で4.29倍とそれほど高くないことがわかる．これは，修正版コードのベクトル化率が 85%程度とそれほど高くないためである．修正版コードのベクトル性能をさらに向上させるための方法については，5.4 節で検討する．

５．３　計算結果の検証

ベクトル化によるコードの高速化を行う場合，コーディング・アルゴリズム等の変更により，計算結果がかわらないことが要求される．ただし，演算順序の違いによる丸め誤差等により，結果が完全には一致しない場合もある．また，FACOM VPシリーズでは，ベクトル演算は必ず倍精度で行われるため，単精度のコードの場合には，自動ベクトル化の際にスカラ計算とベクトル計算で結果が微妙に異なってくる場合もある．

今回示したベクトル化の手法については，修正ルーチン単体でテストしたところ，結果は完全に一致することを確認した．ただし，コード全体でみると結果は微妙に異なってくる．しかし，これはベクトル化のためではなくオリジナルコードの問題である．すなわち，オリジナルコードは単精度であり，かつ精度に非常に敏感なコードであるため，自動ベクトル化によってスカラ計算とベクトル計算で結果が微妙に異なってくる．ただし，コンパイラのオプションで倍精度を指定した場合には，結果は完全に一致する．

5．4　ベクトル性能向上のための考察

一般に，ベクトル計算機において高い性能を得るためには，データに並列性があり，かつその並列度が高いことが重要なポイントとなる．こうした観点から，今回ベクトル化を行ったＤＳＭＣコードの主要な処理について整理したのが Table 5.11 である．ここで，計算コストとは，スカラ実行時のコード全体に対するＣＰＵ時間比，倍率とは，スカラ実行時に対するベクトル実行時のＣＰＵ時間の短縮率を意味する．また計算コストと倍率の値は，修正版の実行結果より計算したものを示した．

Table 5.11　The parallelism of main processes in the DSMC code

処理	並列度　（ベクトル長）	計算ｺｽﾄ	倍率
分子の移動	全分子数	1.5 %	27.4 倍
境界との干渉	全分子数	4.5	3.6
セル番号の割当て①	全分子数	4.7	30.0
ソーティング①	全分子数　＋　サンプリングセル数	10.3	2.2
衝突準備	衝突セル内分子数	4.7	11.1
衝突サブセル分割	衝突セル数	5.8	4.0
セル番号の割当て②	衝突セル内分子数	14.4	11.9
ソーティング②	衝突セル内分子数　＋　衝突サブセル数	12.8	2.9
衝突計算	衝突サブセル内分子数	7.9	1.5
乱数発生	全分子数＊3	4.1	33.9

全分子数　＝　44800
衝突セル内分子数　＝　1600
衝突サブセル内分子数　＝　20
サンプリングセル数　＝　500
衝突セル数　＝　16
衝突サブセル数　＝　100

並列化という観点からみると，4章で示したループ2，ループ3の修正は，隠れていた処理の並列性を引き出す修正である．すなわち，全分子に関して，（分子）→（セル番号）という対応付けにより，処理が並列にできるということを意味する．

Table. 5.11 より，計算コストが比較的大きい処理で，ベクトル処理における性能が得られていないものは，ｉ）ソーティング処理，ii）衝突計算処理　の2つであることがわかる．これらについての分析を以下に示す．

（ｉ）ソーティング処理

ソーティング処理において，ベクトル処理における性能向上率が小さいのは，ソーティングのアルゴリズムに起因する．このコードでは，ソーティングの手法として，頻度法（distribution counting sort）を採用している．この手法は，処理の一部にベクトル処理の出来ない回帰演算を含んでいるため，スカラ処理における性能は高いが，ベクトル処理における性能向上は，1.5 ~ 3 倍程度にしかならない [5]．頻度法におけるベクトル処理向けの改良については，宇佐美らによる報告があるが，決定的な解決策とはなっていない [6]．

筆者らは，頻度法に代わるソーティングの手法として，基底法（radix sort）について性能評価を行ったが，満足のいく結果は得られなかった．ＤＳＭＣ法におけるソーティングは，要素数（分子数）に比べて Key長（セル数）が小さく，かつ，データが常にある程度ソートされた形で並んでいるという特徴を持つ．こうした特徴を活用したＤＳＭＣ法向けのソーティング手法の検討が課題である．

（ii）衝突計算処理

ＤＳＭＣ法における衝突計算では，衝突セル内にある模擬分子同士が衝突を行うという制約がある．衝突サブセル分割法を導入した場合には，衝突サブセル内にある分子同士が衝突を行うことになり，制約は更に厳しくなる．今回のテストデータでは，衝突サブセル内にある分子数は平均 20 であり，これが衝突計算のループにおけるベクトル長になっている．衝突計算処理が 1.5 倍という低い性能向上率しか得られていない原因はベクトル長が短いためである．

しかし，修正南部法においては，衝突の計算自体は，各分子独立に行うことが可能であり，その意味では全分子数の並列性を持つ．Ploss はリストベクトルをうまく用いた修正南部法によるベクトル処理向けの衝突計算アルゴリズムを示した [7]．

筆者らは，この Plossの手法を改良し，衝突サブセル分割法を含む今回のＤＳＭＣコードに応用する試みを行った．結果として，衝突計算部分の速度向上率を5倍程度まで向上させることが可能であることがわかった．

6．おわりに

従来から難しいとされていた，直接モンテカルロ・シミュレーション法（ＤＳＭＣ法）を用いた希薄流れ解析コードの高速化を行った．コードの分析を行い，高速化を阻害している原因を抽出した．その中で大きな比重を占める３つの処理（衝突準備処理，サンプリングセルの割り当て処理，衝突サブセルの割り当て処理）については，根本的なコーディングの改良を行った．特に衝突準備処理については，３つの改善案を示し，それらについて速度性能の評価を行った．その結果，物理的に同じ処理を行う場合でも，コーディングの最適化と効率的なベクトル処理を行うことにより，処理時間が大きく異なることを示した．

コーディングの修正に伴い，コード全体として 25.42倍の高速化が達成された．高速化のポイントとしては，ｉ）無駄な処理をなくして効率化を図る（スカラ・チューニング），ii）問題の持つ並列性を引き出す（ベクトル・チューニング）の両方を検討することが重要であることを示した．

また，今後より一層の高速化を図るには，ソーティング処理，および衝突計算処理の高速化が必要となることを示した．

謝　　辞

本研究の機会を与えて頂きました，情報システムセンター主任研究員　石黒美佐子氏に感謝致します．富士通（株）の南多善氏には，作業全般に渡り貴重な助言を頂きました．深く感謝致します．情報システムセンター室長　浅井清氏には，本報告書を書く機会を与えて頂きましたことを感謝致します．

6．おわりに

従来から難しいとされていた，直接モンテカルロ・シミュレーション法（ＤＳＭＣ法）を用いた希薄流れ解析コードの高速化を行った．コードの分析を行い，高速化を阻害している原因を抽出した．その中で大きな比重を占める３つの処理（衝突準備処理，サンプリングセルの割り当て処理，衝突サブセルの割り当て処理）については，根本的なコーディングの改良を行った．特に衝突準備処理については，３つの改善案を示し，それらについて速度性能の評価を行った．その結果，物理的に同じ処理を行う場合でも，コーディングの最適化と効率的なベクトル処理を行うことにより，処理時間が大きく異なることを示した．

コーディングの修正に伴い，コード全体として 25.42倍の高速化が達成された．高速化のポイントとしては，ｉ）無駄な処理をなくして効率化を図る（スカラ・チューニング），ii）問題の持つ並列性を引き出す（ベクトル・チューニング）の両方を検討することが重要であることを示した．

また，今後より一層の高速化を図るには，ソーティング処理，および衝突計算処理の高速化が必要となることを示した．

謝　　辞

本研究の機会を与えて頂きました，情報システムセンター主任研究員　石黒美佐子氏に感謝致します．富士通（株）の南多善氏には，作業全般に渡り貴重な助言を頂きました．深く感謝致します．情報システムセンター室長　浅井清氏には，本報告書を書く機会を与えて頂きましたことを感謝致します．

参考文献

1）S.M.Yen; "Numerical Solution of the Nonlinear Boltzman Equation for Nonequilibrium Gas Flow Problems", Ann. Rev. Fluid. Mech., Vol.16, pp.67-97 (1984)

2）K.Nanbu; "Theoretical Basis of the Direct Simulation Monte Carlo Method", Rarefied Gas Dynamics, Proc. of the 15th International Symposium, Vol-1, pp.369-383 (1986)

3）村田他; 工学における数値シミュレーション, 丸善 (1988)

4）H.Kaburaki, H.Yamamoto, M.Yokokawa, and T.Arisawa; "A New Collision System for the Direct Simulation Monte Carlo Method", Proc. of 1st International Conference on Supercomputing in Nuclear Applications, pp.51-56, March 12-16 (1990)

5）津田; 数値処理プログラミング, 岩波書店 (1988)

6）宇佐美, 加藤; 直接シミュレーション・モンテカルロ法の強制ベクトル計算, 情報処理学会論文誌 Vol.28, No.12, pp.1217-1226 (1987)

7）H.Ploss; "On Simulation Methods for Solving Boltsman Equation", Computing 38, pp.101-115 (1987)