

2. マルチコア CPU による並列計算

最近のパソコンは、CPU 速度の改善（速度増加）に足踏みが見られるが、その代わり、1つのCPUの中に複数の演算処理機能を持たせた、複数コア・複数スレッドを持つCPUが出現し、OpenMP（参考文献 22）を使った並列計算に対応するコンパイラも一般的になってきた。DSMC 計算を最も効率よく並列化するためには、流れ場をスレッド数と同じ数の領域に分割して各領域を一つのスレッドに受け持たせることであるが、領域を越えて移動する分子を領域間で移し替える必要がある。これまでに報告されている DSMC 計算の並列化はこの方法によるものがほとんどで、単純な流れ場に限定されるが、計算速度は CPU の数（あるいはコア数）にほぼ比例するという報告がある。一方、別の並列計算の方法として、DSMC 法の各計算ルーチンごとに並列化を実行することが考えられる。この方法では、複数の計算を完全に独立させて並列化することは難しく、並列化の効果を十分期待できない面もあるが、その計算法を一度確立しておけば、プログラムの簡単な書き換えで、どんな DSMC 計算の高速化にも適用できる。本報告は、後者の立場に立ち、Intel Fortran コンパイラ V12 を用いて、DSMC 計算プログラムの各ルーチンを並列化する方法を考案し、それによって得られた計算の高速化の程度と解析結果を示す。DSMC 法の主たる計算ルーチンは、分子移動、分子所属セル調査、分子並べ替え処理、分子間衝突、流れ場サンプリングの 5 つで、これら 5 つの計算がこの順序で繰り返し実行される。以下では、これらルーチンすべてについて並列化の手順を説明する。

2.1 並列計算プログラミングにおける一般的事項 いま、N 回の繰り返しループから成る並列実行ループを、ある一つのループ（自己ループと呼ぶ）とそれ以外の N-1 個のループ（他者ループと呼ぶ）に分けて考えてみる。今回の並列処理では、異なるスレッド各々の計算自体は独立に（並列に）行えても、データの記憶場所は共通で使うことになるため、自己ループで呼び出した変数を計算処理して元の記憶場所に戻す際、戻し終える前に他者ループが同じ変数を呼び出したりすると、本来と異なる結果が生じることになって都合が悪い（これを「変数参照交差」と呼ぶことにする）。本研究は、これを防ぎながら、いかに計算速度の速い DSMC 並列プログラムを作るかというのが目的となる。OpenMP では、変数参照交差を防ぐ基本手順として、まずは DO ループ直前に次のような宣言をする。

```
!$OMP PARALLEL DO PRIVATE(..., ..., ...)
!$OMP REDUCTION (+: ..., ..., ...)
```

すなわち、PRIVATE の後のカッコ内で、並列計算で独立して用いる変数を宣言し、REDUCTION の後のカッコ内では、並列計算を越えて合計を求める等を用いる変数を宣言する。

2.2 一様乱数の作成 本報告の DSMC 計算の並列プログラミングでは、16 個のスレッドで並列計算する際、最低でも、独立した 16 系列の一様乱数を使う。今回は、周期 2^{46} の独立した 2 つの乱数列を利用することとし、一周を 8 あるいは 9 個に分割した位置に初期値を定め、それらから出発する 17 系列（1 つは逐次実行用）の一様乱数列を作成して用いた。

2.3 分子あるいはセルをスレッドに割当て方法 DSMC 計算では、個々の分子あるいは個々のセルによって独立している計算を並列実行するが、16 個のスレッドに可能な限り均等に割当てて必要がある。ループの中で乱数を使わないならば、単純に、分子数あるいはセル数の DO ループを並列実行すれば良いが、乱数を使う場合は、OMP_GET_THREAD_NUM()+1 によってそのループのスレッド番号（1~16 の整数）を求め、この整数値から対応する乱数列を得るのがよい。また、スレッドごとの計算負荷を均等にするための工夫として、分子移動では、個々の分子の移動計算の負荷に大きな差異はないので Schedule (Guided) を用い、分子間衝突では、セル内分子数が大きく異なり、セルごとの負荷に差異の生じる可能性があるので Schedule (Dynamic) を使用するとよい。

2.4 分子移動(Molecular movement)の並列化 この計算ルーチンでは、基本的に個々の分子は独立して空間移動するので、全分子数を 16 のグループに分割して並列化すれば良い。しかし、計算領域外に流出した分子を捨て去るとき、分子情報を記憶している配列を整理する処理が必要で、そこは単純には並列化できない。今回は、捨てる分子数がそれほど多くないので、空間移動の並列計算内では捨てる分子にタグ（付け札）を付けるにとどめ、実際の分子捨去り処理は、並列計算の外で逐次処理により行った。なお、境界から流入する分子の移動は、通常は、元から流れ場に存在する分子の移動が終了した後に行うが、今回は、先に境界上に分子を発生させておき、それを含めたすべての分子を一つの並列計算内で空間移動させた。

2.5 分子所属セル調査(Cell identification)の並列化 この計算は、分子ごとに独立であり、簡単に並列化して高速化が実現できる。今回は、処理の更なる効率化を図るために、分子の空間移動の並列計算の中で、所属セル調査も合わせて行った。

2.6 分子並べ替え処理 (Molecular sorting) の並列化 分子参照配列 Cross Referencing Array: LCR(K) に、分子をセルごとに並べ直すこのルーチンは、分子番号とセル番号が相互に依存した計算であるため、変数参照交差が生じて簡単には並列化できない。しかし、プログラムの主要部分に並列化できない箇所があると、そこがボトルネックになって大幅な高速化が達成できない。そこで、特殊な方法の並列計算によって高速化を試みた。その方法を説明するために、まず、分子並べ替え処理について理解する必要がある。第 1 図に示すように、この処理は、全分子をそれが所属するセルの順に並べて「一次元の参照用配列」を作ることにある。すなわち、セルごとに連続した配列位置に分子が並ぶので、同一セル内の衝突ペアとなる二分子は乱数により容易に選択できる。

	Cell 1							Cell 2					Cell 3						
K	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	
LCR(K)																			

K: Address number

LCR (K): Molecular identity number is stored.

Cell 1: Address 1~7

Number of molecules 7

Starting Address - 1 = 0

Cell 2: Address 8~12

Number of molecules 5

Starting Address - 1 = 7

Cell 3: Address 13~18

Number of molecules 6

Starting Address - 1 = 12

第 1 図 分子参照配列の例

この参照用配列を作るために、全分子数をスレッド数で割り、一つのスレッドが担当する分子を決めて、最初、各スレッドが担当した分子に限定した参照配列を並列計算により作成する。各スレッドで作成した部分的参照配列をセルごとにつなぎ合わせれば、求めるべき「分子参照配列」が出来上がる。ただし、この「つなぎ合わせ処理」についても、できる限り並列計算を利用できるような工夫が必要である。以上の方法は、セル数×スレッド数といった大きな配列を必要とするが、どのような DSMC 解析にも同じプログラムを使えるという利点がある。

2.7 分子間衝突 (Intermolecular collision) の並列化 このルーチンは、セルごとに計算が独立しているので、並列化が可能で、しかも密度が高くなるほど全計算時間に占める分子間衝突の割合が増加するので、高速化に最も寄与できる。

なお、先に述べたように、特に今回のような軸対称解析では、セル内分子数がセルによって大きく異なり、必ずしも個々のセルで計算負荷が一定しないことが起こるので、**Schedule (Dynamic)** を指定して、空いているスレッドに優先して仕事が割当てられるようにするのがよい。

2.8 流れ場サンプリング (Sampling of flow properties) の並列化 このルーチンは、計算規模が大きくなると全体の計算時間に占める割合が無視できなくなるので高速化したい部分である。今回は、先の分子並べ替え処理で作成した分子参照用配列 **LCR(K)** を用いてセルごとに並列化することとし、また、分子間衝突計算のループの中に組み入れることで更なる効率化を図った。

2.9 分子情報記憶配列の整列化 全分子の位置座標や速度成分を記憶する配列変数に初期値を与える際は、セルの番号順に分子に番号を与えるので整然と並んでいるが、その後、分子の空間移動が繰返されると、それら配列の分子情報は、セルの順とは無関係な並びとなる（参照配列 **LCR(K)** が必要なのはそのためである）。しかし、分子情報を記憶している配列変数が、セルの順に整然としていた方が、明らかに、計算速度は速くなることが判明したので（ただし厳密にセル順である必要はない）、全分子が空間移動を 10 回繰返したら、その配列変数の整列化を行うという操作を追加した。なお、この操作は並列計算に限らず、むしろ逐次計算で効果が大きく、今回の圧力比 50 の解析では、並列計算で約 1.45 倍の速度増加があったのに対し、逐次計算では約 1.7 倍の速度増加となった。

3. 並列計算による高速化

軸対称流れ場における Ar 単一気体の不足膨張噴流に関して並列計算の効果を調べた。第 1 表と第 2 表は、**U-system** を用いて計算した圧力比 50 の結果で、各計算ルーチンを 1 つのコアで逐次計算した場合と、8 個のコア（16 個のスレッド）で並列計算した場合に要した計算時間の違いを示す。なお、並列計算では、いくつかの計算ルーチンが結合されているので、その場合は合計を示している。第 1 表はオリフィスより上流の流れ場を持たない簡易プログラムの結果（分子数約 5800 万個）で、第 2 表は上流の流れ場を持ち、混合気体にも対応したプログラムの結果（分子数約 1 億 4 千万個）である。表の数値は、並列計算の全ルーチンの合計時間を 1.0 として無次元表示している。なお、逐次計算の場合は、逐次計算用に作成したプログラムコードの結果と並列計算用に作成したコードを逐次計算したときの結果を共に示す。逐次実行でプログラムコードが異なっ

ていても計算時間にほとんど差がないのは意外な結果であるが、逐次計算であっても、実際の数値計算以外の部分で、他のコアが計算機を補助している可能性があるためこのようになったものと思われる。今回の計算について言えば、全体時間に占める分子間衝突計算の割合が大きく（全体の 9 割以上）、かつ、衝突計算は並列化によって 7~8 倍の速度増加となっているので、計算全体の速度増加で見ても 7~8 倍となり、8 個のコアの CPU としてはまずまずの効果が得られたものとする。

第 2 図~第 5 図は、第 1 表の①に相当する計算について、スレッド数が計算速度に及ぼす影響を調べたものである。計算機の BIOS 設定は、使用できるコア数を制限でき、また、Hyper Thread Technology (HTT) を使用するかどうかの指定も行える。その場合、8 コアで HTT オフにすると 8 スレッドとなるが、4 コアで HTT オンとしても 8 スレッドとなるので、計算は 16 種類あるが、スレッド数の変化は 12 種類となる（図の横軸参照）。縦軸は、スレッド数 1 の場合の逐次計算用コードの計算速度を 1 とし、任意のスレッド数の並列計算用コードの速度がその何倍になるかを speed-up と示している。なお、第 2 表（分子数約 2.4 倍、セル数約 1.05 倍）に対応する同様の計算も試みたが、ほぼ同じ傾向となったのでここでは省略する。

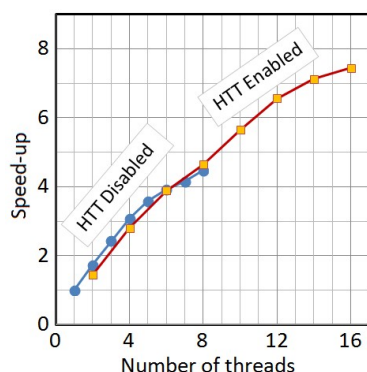
第 1 表 並列計算の全ルーチンを 1 とし無次元表示した「逐次計算」と「並列計算」の DSMC 各ルーチンの計算時間の比較（上流計算領域なしのプログラム）

	Sequential ① (Sequential Code)	Sequential ② (Parallel Code)	Parallel
Movement	0.166	0.34	0.057
Cell ident.	0.188		
Sorting	0.06	0.08	0.029
Collision	6.6	6.6	0.91
Sampling	0.14		
Whole	7.2	7.0	1.0

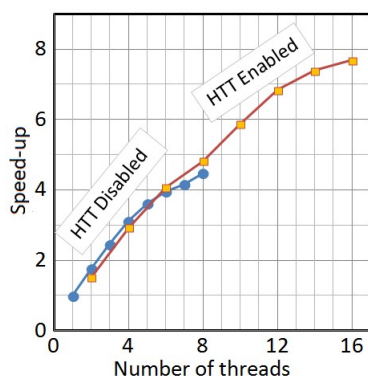
第 2 表 並列計算の全ルーチンを 1 とし無次元表示した「逐次計算」と「並列計算」の DSMC 各ルーチンの計算時間の比較（上流計算領域あり、混合気体に対応）

	Sequential ① (Sequential Code)	Sequential ② (Parallel Code)	Parallel
Movement	0.074	0.185	0.022
Cell ident.	0.098		
Sorting	0.02	0.11	0.012
Collision	7.5	7.9	0.97
Sampling	0.13		
Whole	7.8	8.2	1.0

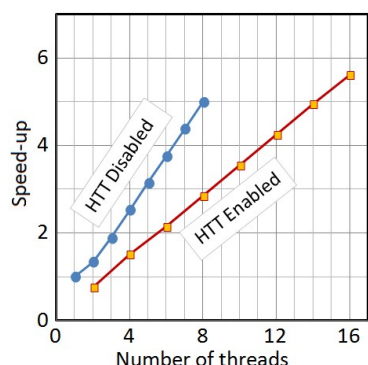
第2図は、全計算ルーチンの合計時間の比較で、青色がHTTを使用しない場合、赤色がHTTを使用してコア1つあたりスレッドを2としたものである。この場合、HTT使用の如何にかかわらず、スレッド数の増加につれて計算速度は増加している。しかし、スレッド数が最大値16に近づくと速度増加の程度は鈍くなり、16スレッドで約7.4倍となっている。第1表の①に示した値が7.2倍であるのに対し、第2図で約7.4倍となった理由は、第1表では、DSMC計算以外で計算機が行うべき諸々の仕事（Windows OS等の仕事）は余裕のある他のスレッドが担っていたが、第2図での比較においては、DSMC計算を含めたすべての仕事を、許可されたスレッドだけで行わなければならないために、基準となる1つだけのスレッドの計算時間が増加してこのようになったものと思われる。第3図は、分子間衝突と流れ場サンプリングの合計時間について求めたものである。本計算は、密度が比較的高く、分子間衝突の計算割合が大きい流れ場であるために、この第3図の結果が全計算ルーチンの結果に大きく影響して、第2図と第3図は類似した傾向となっている。この計算ルーチンは、衝突セルごとに並列化しているが、各セルに所属する分子の数が一定ではないため、各スレッドの計算負荷に違いが生じ易い。スレッド数が最大値に近づくと速度増加の程度が鈍くなったのは、このことが原因の一つと考えられる。



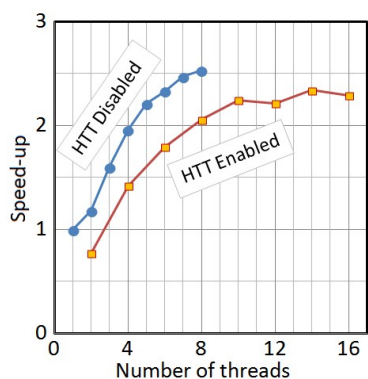
第2図 スレッド数と計算速度（全計算ルーチン合計）



第3図 スレッド数と計算速度（分子間衝突と流れ場サンプリング）



第 4 図 スレッド数と計算速度（分子移動と分子所属セル調査）



第 5 図 スレッド数と計算速度（分子並べ替え処理）

第 4 図は、分子移動と分子所属セル調査の合計時間の比較である。この計算ルーチンでは分子ごとに単調な計算を並列化しており、分子一個あたりの計算量がほぼ等しく、また分子数が多いことから、コア数が増加するとそれに比例して計算速度は増加している。しかし、同じコア数において、HTT をオンとオフで比較した場合、HTT オンすなわちスレッド数を 2 倍にした効果はわずかし

か出ていない。

第 5 図は、分子並べ替え処理に当たる部分で、従来のプログラムでは並列化できないものを、特殊な操作で並列化したために速度増加の程度は低い。しかし、今回の計算例では全計算量に占める割合が 3%以下であるため、全体の高速度化にはそれほど影響を与えていない。スレッド数が 10 を超えると、速度増加は頭打ちで、そのため HTT オンの方が HTT オフより低速となっている。この部分の並列化にはまだ改善の余地あると思われる。第 4 図と第 5 図のコア数 1 の場合、HTT オンの方が HTT オフよりも速度低下しているのは、基準となるスレッド数 1（コア数 1 で HTT オフ）は逐次計算コードであるのに対し、それ以外は並列計算コードのため、速度をかえって遅くしてしまったのであろう。