スーパーコンピューター SX4 による 分子動力学シミュレーションの並列計算実行の試み (超高速衝突破壊シミュレーションへの適用)

大阪府立大学工学部航空宇宙工学科　中谷敬子
nakatani@aero.osakafu-u.ac.jp

1 はじめに

ここでは，報告者が，ワークステーション上で行なっている分子動力学シミュレーションに関するプログラムを SX4 上で実行させる場合に，並列計算を行なうことにより，どの程度効率化がなされるかについて，検討した結果について報告する．

問題としては，スペースデブリの超高速衝突破壊現象を取り上げ，18,185 個から成る系の分子動力学シミュレーションについて検討する．

以下では，まず，シミュレーションのモチベーションと解析対象の概要について簡単にふれた後，SX4 上での計算について報告する．具体的には，ワークステーション上で使用している，そのままのプログラムを単独の CPU で実行したもの，プログラムはそのままで自動並列化したもの，および，自動並列化指示行を加えたものの検討結果について示す．

2 解析対象の概要

2.1 宇宙環境問題の現状

宇宙開発の進展とともに，地球周辺の宇宙空間には数多くのスペースデブリ (使用済み衛星やロケット，運用上の廃棄物，それらの破片や塗料片) が残されることになった．このようなスペースデブリは地上から観測されているもので約 5000 個，観測されない小さなものも含めると 300 万個，総質量は 3000 トンに達する [1]．それらは猛スピードで宇宙を飛び回り，運用上の衛星と衝突する時の速度は，平均で 10km/s になり非常に大きな破壊力を持っている．宇宙ステーションの設計では，観測可能な 10cm 以上のデブリとの衝突は，軌道制御によって回避し，1cm 以下のデブリはシールド板で防御する．ここでは，微小デブリが防御壁に衝突する時の，その破壊のメカニズムを分子動力学シミュレーションによって解析することが狙いである．

2.2 解析モデルと結果の概要

図 1 に示すように，前述の問題を球形のプロジェクタイルと平板のターゲットによりモデル化し，これらはともに，アルミニウムの結晶構造である面心立方構造とし，プロジェクタイルは，1721 個の原子から成り，ターゲットは，16465 個の原子を最密面 ((111) 面) が表面となるよう配置する．

解析条件を表 1 に示す．原子間の相互作用は，式 (1) のモース・ポテンシャルに基づくとし，カットオフ距離での相互作用がなめらかに零となるように修正した，Shifted-Forse ポテンシャルを用いる．

$$\phi_{\mathrm{SF}} = D\left(e^{-2A(r-r_0)} - 2e^{-A(r-r_0)}\right) \tag{1}$$

ここで，$D = 4.330717408 \times 10^{-20}$ J，$A = 1.1646 \times 10^{10}$，$r_0 = 3.253 \times 10^{-10}$m であり，$r_c = 2r = 0$ とした．

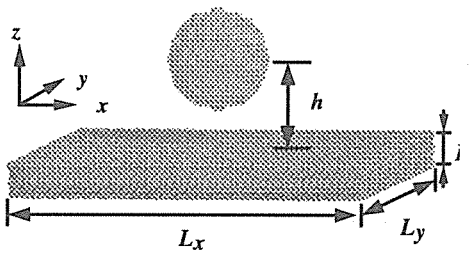


図 1: 超高速衝突破壊の解析モデル

表 1: シミュレーション条件

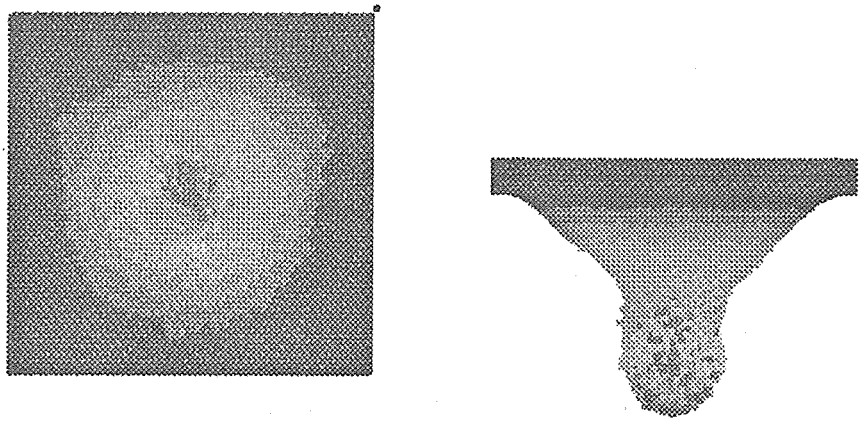
Number of atoms	18186 ($N_{\text{plate}} = 16465, N_{\text{projectile}} = 1721$)
Initial temperature	0K
Potential	Shifted-Force potential (Morce potential)
Time increment	2.0×10^{-15} s
External stress	0 Pa

衝突のシミュレーションでは，ターゲットの周囲の 2848 個の原子は境界条件として完全に固定し初期温度を絶対零度に設定して，プロジェクタイルに一定の初速度を与えて，分子動力学シミュレーションを行なう．

Fig. 2 に 最終ステップの原子配置のスナップショットを示す．ここで，原子の色の濃淡は，初期位置からの変位量による等高線である．ターゲットの裏面に大きな盛り上がりが短時間でてき，ターゲットは破壊され，プロジェクタイルはほぼ貫通していることが分かる．

このシミュレーションでは，3900 ステップまで計算を実行した結果を示しているが，以下では，この問題に対して，初期の 10 ステップ (10 回の繰り返し) の計算を問題とする．(10 回の繰り返し計算では，リストベクトルの構築部分の全計算量に占めるウエイトが大きく，長時間ステップの計算へ適用した場合とではの対応は，比例関係にないが，定性的な計算量を比較する場合には十分であると考えられる．)

以下，SX4 上でのシミュレーションを追って述べる．



(a) xy-plane,t=7.8ps　　(b) xz-plane,t=7.8ps

図 2: 衝突破壊時のターゲットの変形

3 プログラムのコンパイルと実行

3.1 コンパイル

SX4 には，f77sx，f77，f90 の 3 つの FORTRAN コンパイラーが用意されている．ここでは，f77 を使用し，以下のオプションをつけてコンパイルした．

```
% f77 -C vopt -pi -Wo'-l mono.lst' -o allmono all.f
```

ここで，用いているオプションの意味は次の通りである [4].

-C vopt	最適化，および，ベクトル化は通常 (MIDDLE) レベル (default) 他の指定は， サブオプション　最適化レベル　ベクトル化レベル debug　NO　NO ssafe　LOW　NO vsafe　LOW　LOW sopt　MIDDLE　NO hopt　HIGH　HIGH LOW:副作用なしレベル，MIDDLE:通常，HIGH:最大限の高速化
-pi	手続きのインライン展開機能を利用することを指定
-Wo'-l mono.lst'	コンパイル時の出力リストを mono.lst として指定

出力リスト “mono.lst” には，プログラムのどのループをベクトル化または並列化したかといった情報が得られ，プログラムの並列化向上に役立つ [5].

標準エラー出力には，ループのベクトル化に関する情報が以下の例のように表示される．

```
 MDMAIN Loops: 13 examined, 6 optimized
    subroutine TITLE
                              (中略)
 BOOKKEEP Loops: 2 examined, 0 optimized
    subroutine FORCE
 FORCE Loops: 9 examined, 6 optimized
    subroutine KENERG
v.all.f:
   ERROR TOTAL :  **** : 0    *** : 0    ** : 4    * : 0
   PROGRAM : mdmain$1
       363        DO I3 = 1, NATOM

       VEC   1 : ＤＯループ全体をベクトル化する．ＤＯ変数は i3

   ERROR TOTAL :  **** : 0    *** : 0    ** : 0    * : 0
   PROGRAM : mdmain$2
       389        DO I3 = 1, NATOM

       VEC   1 : ＤＯループ全体をベクトル化する．ＤＯ変数は i3
   NO ERROR
   PROGRAM : mdmain$3
       407        DO I4 = 1, NATOM

       VEC   1 : ＤＯループ全体をベクトル化する．ＤＯ変数は i4
   NO ERROR
   PROGRAM : mdmain$4
       427        DO I6 = 1, NATOM

       VEC   1 : ＤＯループ全体をベクトル化する．ＤＯ変数は i6

   ERROR TOTAL :  **** : 0    *** : 0    ** : 0    * : 0
                              (後略)
```

【参考】 Makefile を用いた分割コンパイル ワークステーション上で用いていた Makefile がそのまま使える．ただし，SX4 では，デフォルトで使用する FORTRAN コンパイラーが f77sx に指定されているようである．ここでは，コンパイラーを f77 に指定したいので，Makefile 内で，

```
F77             = f77
```

あるいは，

```
FC              = f77
```

と，明示的に指定する必要がある．

3.2 シミュレーションプログラムの実行

3.2.1 NQS ジョブによる実行

a. シェルスクリプトの作成 以下のようなスクリプトファイル (run.nqs) を作成した．

```
#!/usr/bin/csh
#@$-q p4
#@$-lt 0:10:00
#@$-o stdout.dat      ··· 標準出力ファイル名の指定
#@$-e stderr.dat      ··· 標準エラー出力ファイル名の指定
#@$
date                  ··· ここから実行したい内容を書く
setenv F_FILEINF YES
setenv F_PROGINF YES
../../src/mdemo3d < stdin.dat
date                  ··· ここまで実行したい内容を書く
```

作成したスクリプトファイル “run.nqs” を使って，ジョブを投入するには，以下のように qsub コマンドを用いてサブミットする．

b. qsub コマンド

```
ccsx40 /usr1/x60440/Debris/case1f<585>%qsub run.nqs
Request 1239.ccsx40 submitted to queue: p4.
```

これで，ジョブクラス p4 にジョブが投入されたことが分かる．qstatq コマンド (実行中のジョブとクラスのジョブ投入状況を表示) と qstatr コマンド (qstatq と同じく，実行中のジョブの状況が示されるが，ジョブを投入したユーザーの番号など，より詳細に表示される) により，ジョブが正常に走っているかどうかを確認した．

```
ccsx40 /usr1/x60440/Debris/case1f<422>% qstatq
=============================================================
NQS (R08.10) BATCH QUEUE SUMMARY   HOST: ccsx40
=============================================================
QUEUE NAME       ENA STS PRI/BPR/ TMS /MPR RLM  TOT QUE RUN WAI HLD SUS ARR EXT
---------------- ---------------------------- ---------------------------------
P32              ENA STP  31/ 80/ 2000/  0   1    0   0   0   0   0   0   0   0
P64              DIS STP  31/ 80/ 2000/  0   1    0   0   0   0   0   0   0   0
PA4              ENA RUN  31/ 80/ 2000/  0   2    1   0   1   0   0   0   0   0
PA8              ENA RUN  31/ 80/ 2000/  0   1    1   0   1   0   0   0   0   0
PB4              ENA RUN  31/ 80/ 2000/  0   2    1   0   1   0   0   0   0   0
---------------- ---------------------------- ---------------------------------
   <TOTAL>                                  50   3   0   3   0   0   0   0   0
---------------- ----------------------------- --------------------------------
                              (後略)
```

```
ccsx40 /usr1/x60440/Debris/case1f<423>%qstatr -t 2
=============================================
NQS (R08.10) BATCH REQUEST  HOST: ccsx40
=============================================
REQUEST ID       NAME     OWNER    QUEUE    PRI NICE MEMORY  TIME  STT  JID   R
---------------- -------- -------- -------- ---- ---- ------- ----- --- ------ -
1275.ccsx40      run.nqs  x60440   PA4        31    0  15065    825 RUN   1097 -
---------------------------------------------------------------------------------
                              (後略)
```

上記の出力では，リクエスト番号 1275 のジョブとして，x60440 のユーザーによって，run.nqs をスクリプトファイルとしてジョブクラス PA4 に投入されたジョブが，現在走っている状態 (RUN) であることが分かる．

【参考】 qdel コマンド　上記ジョブを実行中，または，実行待ちの状態で，キャンセルする場合は，次のようにリクエスト ID(qstatr 出力の 1 カラム目) を指定してキルする．

3.2.2 実行結果

実行が終了後，NQS スクリプトファイルで指定した標準出力ファイル (stdout.dat) と，標準エラー出力ファイル (stderr.dat) が作成される．エラー出力ファイルに出力された，プログラム情報は以下の通りである．

NQS スクリプトファイルの中で，setenv F_FILEINF YES を指定しているので，標準エラー出力ファイルに，ファイル情報が記録される．また，setenv F_PROGINF YES とした場合は，詳細なプログラム情報が得られる (setenv F_PROGINF を指定していない時は，“MIPS 値” 以下の最後の 9 行の情報が省略される)．

```
                ******    ファイル  情報    ****** ··· F_FILEINF YES をしなければ省略.
外部装置識別子    :  9
ファイル名        : energy.p4
名前付きファイル  : YES
ディレクトリ名    : /home/ccns01/user5/x60440/Debris/case1f/jp4.org

入出力実行回数        :      READ         WRITE        OPEN       CLOSE    INQUIRE
                              0            25           1          0          0
                          REWIND     BACKSPACE     ENDFILE
                              0            0           0

FORM 指定子              :    FORMATTED
BLANK 指定子             :       NULL       ACCESS 指定子        : SEQUENTIAL
ファイルサイズ (Byte)    :       1963       ファイル記述子       :          6
ファイルシステムタイプ   :                  オープンモード       :  READWRITE
端末接続                 :         NO       ファイル縮小機能     :         NO
```

```
入出力バッファサイズ(KByte,F_SETBUF) :          4

                              全体(入出力)              入力              出力
総転送量    (Byte)       :          1963,              0,              1963
最大転送量 (Byte)        :                             0,               139
最小転送量 (Byte)        :                             0,                76
平均転送量 (Byte)        :            78,              0,                78

                ******    ファイル 情報    ******
                          (中 略)
      ******    プログラム 情報    ******
経過時間 (秒)                    :        30.052694
ユーザ時間 (秒)                  :        21.503723
システム時間 (秒)                :         5.704713
ベクトル命令実行時間 (秒)        :         3.357745
全命令実行数                     :      1031015249.
ベクトル命令実行数               :        27074601.
ベクトル命令実行要素数           :      6753762353.
浮動小数点データ実行要素数       :      3071995335.
MOPS 値                          :       360.760927
MFLOPS 値                        :       142.858767
平均ベクトル長                   :       249.450116
ベクトル演算率 (%)               :        87.058790
メモリ使用量 (MB)                :        93.031250
MIPS 値                          :        47.945895 …F_PROGINF YES をしなければ，以下の
行が省略.
命令キャッシュミス (秒)          :         2.385445
オペランドキャッシュミス (秒)    :         1.396742
バンクコンフリクト時間 (秒)      :         0.005179
書式なし入出力転送量 (合計) (MByte)
  全ファイルシステム             :         0.000000
書式なし入出力転送量 (平均) (MByte)
  全ファイルシステム             :         0.000000
```

4 プログラムのコンパイルと実行 (並列化の場合)

ここでは，まず，SX4 の自動並列化機能を利用した並列化計算により，CPU の個数と，その経過時間や，コストについて考える．並列化の効率を上げることを目標にする．

4.1 自動並列化

4.1.1 コンパイルと実行

以下のオプションを指定してコンパイルする [6]．これにより，a.out という実行ファイルができる．

```
% f77 -P auto -C vopt -pi -Wo'-l auto.lst' all.f
```

ここで，並列化しない場合以外で指定しているオプションは，以下のとおりです．

-P auto	自動並列化機能を利用する．f77sx では，-multi-fopp par と同等

これにより得られた実行ファイルを，NQS スクリプトファイルを作成し，qsub コマンドでサブミットすることにより実行する．

4.1.2 自動並列化指示行を指定した場合

プログラムに全く変更を加えなかった場合では，ほとんどのループが自動並列化の対象にならなかった．そこで，以下のような自動並列化指示行をプログラムの先頭に加えた場合について検討する．

```
*odirf concur,inner,cncall
```

4.1.3 CPU の個数の影響

次に，CPU の個数の違いによって，どのように経過時間が異なるのかを調べた．CPU の個数を変えるのは，具体的には，NQS スクリプトファイルの #@$-q の行で指定しているクラスを変えるとともに，環境変数 F_RSVTASK を設定する．

```
#!/usr/bin/csh
#@$-q p4     ··· p4 なら CPU4 個，p8 なら CPU8 個，p16 なら CPU16 個
#@$-lt 8:00:00
#@$-o stdout.p4
#@$-e stderr.p4
#@$
date
setenv F_FILEINF YES
setenv F_PROGINF DETAIL
setenv F_RSVTASK 4
../../src/a.out < stdin.p4
date
```

標準エラー出力ファイルのプログラム情報の部分から，CPU の個数を，4 個，8 個，16 個と変化させた場合の比較を表 2 に示す．いずれのシミュレーションにおいても，並列化によって計算結果が異なることはなかった．

指示行なしの場合に比べて，指示行をつけた場合には，並列化が促進されていることが分かる．並列化を行なった場合，8 台の CPU を使った計算で，1 台以上の CPU で実行した時間が最も短くなった．ただし，ここでの自動並列化では，単 CPU での計算に勝るような効果は得られなかった．

自動並列化はなされるものの，1 台以上の CPU で実行した時間が 2 台以上のそれにほとんどに 1 個の CPU に 70%以上の負荷が集中しており，残り負荷が残りの CPU にほぼ均等に割り振られているのがわかる．本来なら，平均的に負荷が割り振られ，多少計算時間が増加しても，大阪大学大型計算機センターの場合，ジョブクラスごとに換算係数と称する課金の重み付け制度があるため，並列化により，安価に短時間で計算結果を得ることができるはずだが，今回用いたプログラムでは，プログラムはそのままで自動並列化機能を利用するだけでは均等に負荷が割り振られず，この制度を有効に活用することができない．今後，プログラムの並列化のための工夫が必要となる．

5 おわりに

予想以上に，通常のプログラミングに時間がかかり，プログラムの並列化にはほとんど手をつけられなかった．しかしながら，超並列計算研究会 (主査:同志社大学工学部知識工学科 三木光範教授) [7] に参加す

るなどして，並列化の現状や，最新の研究成果についての勉強をこの1年を通じて行なって来たので，今後，超高速衝突のシミュレーションに対して並列計算機SX4の性能を最大限に活用し，従来扱えなかった大規模かつ，長時間のシミュレーションを行なえるようなコーディングを目標としたい．

従来，分子動力学法は，現象の本質的な一面を的確に評価できる一方，取り扱う系のスケールから実際の現象の定量的な予測は必ずしも満足のいくものではないとされてきた．しかしながら，並列計算機を活用すれば，より現実に近い大きな系で長時間の解析を実施することが可能となり，本研究手法の並列計算機プログラムの獲得は，ここで扱っている問題に制限されず，計算機能力の制約から従来扱うのが困難であった大規模計算が可能となり，これにより，放射線による熱制御材などの劣化の原因および対策など，宇宙環境で問題となっている多くの問題に対しても重要な視点を与えるといった幅広い応用分野が期待される．解析手法の高度化により，必要な強度を有する材料組成の迅速な提案が可能となり，材料開発 (材料設計) 分野の活性化にもつながるものと考えられる．

最後になりましたが，並列計算の経験が全く無かった報告者に，SX4のモニターの機会をお与え頂きました，大阪大学大型計算機センターの皆様，豊富な情報を提供頂きましたSX4ユーザーのメーリングリストの皆様に感謝致します．プログラムが簡単に並列化出来なかったため，高い処理能力を持つSX4の性能を十分に活かし切れなかったことが残念ですが，来年度もSX4を用いて並列計算を実施し，今回残された課題に取り組んで行きたいと思います．

参考文献

[1] 八坂哲雄，“宇宙のゴミ問題”，(1997)，裳華房．

[2] 大阪大学大型計算機センターニュース，**27**-1, (1997), 4-6.

[3] 大阪大学大型計算機センターニュース，**27**-1, (1997), 72.

[4] 大阪大学大型計算機センター講習会資料．SX-4 FORTRAN のご紹介 (並列処理を中心として)NEC(1997)．

[5] 大阪大学大型計算機センター速報，スーパーコンピュータSX-4の並列化率の向上について，No.264, (1997), 9-11.

[6] 大阪大学大型計算機センターニュース，**26**-4, (1997), 12-14.

[7] 超並列計算研究会 `http://mikilab.doshisha.ac.jp/SMPP/`.

表 2: プログラム情報

CPU の個数	1 台	指示行なし 4 台	指示行あり 4 台	8 台	16 台
経過時間 (秒)	30.052694	37.194836	35.109383	36.082033	39.162933
ユーザ時間 (秒)	21.503723	34.289883	35.977190	47.503381	83.040032
システム時間 (秒)	5.704713	5.883778	5.897756	6.785421	7.748653
ベクトル命令実行時間 (秒)	3.352488	3.357745	3.354296	3.366672	3.368899
全命令実行数	1031015249.	1204643501.	1232420586.	1368104189.	1641270384.
ベクトル命令実行数	27074601.	27075129.	27075504.	27096460.	27118420.
ベクトル命令実行要素数	6753762353.	6753893759.	6753940863.	6754178263.	6754653063.
浮動小数点データ実行要素数	3071995335.	3071995388.	3072035557.	3072088693.	3072197123.
MOPS 値	360.760927	231.306188	221.231451	170.412838	100.780369
MFLOPS 値	142.858767	89.588973	85.388424	64.670948	36.996579
MOPS 値 (実行時間換算)		280.561630	300.082920	306.774974	286.414467
MFLOPS 値 (実行時間換算)		108.666475	115.822626	116.419799	105.143052
平均ベクトル長	249.450116	249.450105	249.448389	249.264231	249.079890
ベクトル演算率 (%)	87.058790	85.153199	84.856115	83.434504	80.712277
メモリ使用量 (MB)	93.031250	98.000000	98.000000	102.000000	113.000000
最大同時実行可能プロセッサ数	1.	4.	4.	8.	16.
1 台以上で実行した時間 (秒)	21.503723	28.269946	26.523622	26.388026	29.219212
2 台以上で実行した時間 (秒)		2.019954	3.203904	3.052577	4.589662
3 台以上で実行した時間 (秒)		2.011287	3.185310	3.041397	4.122864
4 台以上で実行した時間 (秒)		1.988801	3.064317	3.032829	4.041492
5 台以上で実行した時間 (秒)				3.023430	3.957186
6 台以上で実行した時間 (秒)				3.014218	3.891569
7 台以上で実行した時間 (秒)				3.008389	3.831080
8 台以上で実行した時間 (秒)				2.942695	3.788472
9 台以上で実行した時間 (秒)					3.758644
10 台以上で実行した時間 (秒)					3.728852
11 台以上で実行した時間 (秒)					3.695822
12 台以上で実行した時間 (秒)					3.658531
13 台以上で実行した時間 (秒)					3.623406
14 台以上で実行した時間 (秒)					3.582830
15 台以上で実行した時間 (秒)					3.437054
16 台以上で実行した時間 (秒)					0.115373
イベントビジー回数		0	0.	0.	0.
イベント待ち時間 (秒)		0.000000	0.000000	0.000000	0.000000
ロックビジー回数		0.	0.	0.	0.
ロック待ち時間 (秒)		0.000000	0.000000	0.000000	0.000000
バリアビジー回数		0.	0.	0.	0.
バリア待ち時間 (秒)		0.000000	0.000000	0.000000	0.000000
MIPS 値	47.945895	35.131164	34.255610	28.800143	19.764809
MIPS 値 (実行時間換算)		42.612161	46.465018	51.845644	56.170932
命令キャッシュミス (秒)	2.385445	4.494942	3.972116	3.980217	4.044716
オペランドキャッシュミス (秒)	1.396742	1.744726	1.601862	1.798368	2.124382
バンクコンフリクト時間 (秒)	0.005179	0.005180	0.005192	0.005199	0.005191
書式なし入出力転送量 (合計) (MByte) 全ファイルシステム	0.000000	0.000000	0.000000	0.000000	0.000000
書式なし入出力転送量 (平均) (MByte) 全ファイルシステム	0.000000	0.000000	0.000000	0.000000	0.000000

モニターとして計算したプログラムに関する情報

****** プログラム 情報 ******	
経過時間（秒）	5244.146403
ユーザ時間（秒）	12551.411303
システム時間（秒）	20.605429
ベクトル命令実行時間（秒）	12065.460115
全命令実行数	270732895286.
ベクトル命令実行数	159973001332.
ベクトル命令実行要素数	38596019332266.
浮動小数点データ実行要素数	1759010509051 7.
MOPS 値	3083.858722
MFLOPS 値	1401.444401
MOPS 値　（実行時間換算）	8439.766311
MFLOPS 値（実行時間換算）	3835.410213
平均ベクトル長	241.265832
ベクトル演算率（%）	99.713849
メモリ使用量（MB）	666.000000
最大同時実行可能プロセッサ数	4.
1台以上で実行した時間（秒）	4586.238268
2台以上で実行した時間（秒）	2661.883867
3台以上で実行した時間（秒）	2661.430830
4台以上で実行した時間（秒）	2643.007797
イベントビジー回数	0.
イベント待ち時間（秒）	0.000000
ロックビジー回数	0.
ロック待ち時間（秒）	0.000000
バリアビジー回数	0.
バリア待ち時間（秒）	0.000000
MIPS 値	21.569917
MIPS 値（実行時間換算）	59.031581
命令キャッシュミス（秒）	12.655809
オペランドキャッシュミス（秒）	9.315687
バンクコンフリクト時間（秒）	21.687243

金属吸着Si(001)表面の局所状態密度の計算

理論計算で求めた局所状態密度とSTM/STSで得られた局所状態密度を比較することにより吸着金属の同定を行う。このために、Si(001)表面、及びその表面に金属原子が吸着した時の局所状態密度をあらかじめ知っておく必要がある。そこで、局所密度近似、平面波基底、擬ポテンシャル、スーパーセル法などの計算手法を用いた量子力学的計算を行い、表面の局所状態密度のシミュレーションを行うことを試みた。

1．第一原理分子動力学による電子状態の計算方法

第一原理計算では、計算対象となる物質の原子の種類、属性のみを計算の入力とし、それ以外はなんら経験的なパラメータを含むことなく、設定された原子系における電子の量子力学的状態を求めることができる。しかし、結晶は膨大な数の原子の集まりであり、各原子が数個から数十個の電子を持つような系であるから、これを完全に扱うことはもともと不可能である。そこで以下に述べるようないくつかの手法、近似を導入する必要がある。