# GAMESS ユーザーマニュアル

# 目次

# 1　GAMESS について

GAMESS は、非経験的分子軌道法/密度汎関数理論計算プログラムです。開発の過程で複数の系統に枝分かれしており、著名なものとして以下が挙げられます。

1. Iowa 州立大の M.S.Gordon を中心とした研究グループが中心となって[1]開発及び配布が行われている系統（　通称、GAMESS(US)　）
2. イギリスの CFS によって開発および配布が行われている GAMESS-UK
3. Moscow 大学の Alex A. Granovsky らによって開発された Firefly（旧：PC GAMESS）

本書で記載する内容は 1.についてであり、これを以下単に GAMESS と記載します。

GAMESS は使用者、使用グループが公式ホームページに登録する事により、無償で入手できます。ライセンスの詳細については、GAMESS のホームページにあるライセンス規約（ https://www.msg.chem.iastate.edu/gamess/License_Agreement.html ）を参照ください。

本マニュアルでは GAMESS の実行方法をご案内します。より詳細な内容については以下の GAMESS の公式 HP をご確認下さい。

https://www.msg.chem.iastate.edu/gamess

[1] 日本では、大阪府立大の小関らが開発に携わっている

# 2　GAMESS 実行方法

## 2.1　GAMESS の並列計算手法

GAMESS は、バイナリ作成時に、DDI(Distributed Data Interface) 版あるいは MPI(Message Passing Interface)版という２つの並列手法からいずれかを選択します。

この選択は、計算原理そのものの選択ではなく、GAMESS の並列計算における、複数の CPU 間（プロセス間）の通信手法を選択するものです。

前者の DDI 版は、昔から GAMESS が標準としてきた通信手法です。ソースコード中にある、通信用のライブラリを使用する手法で、代々の GAMESS において使用されてきた実績があります。複数ノードをまたいだ計算にも対応していますが、基本的には Gigabit Ethernet による接続を想定しており、最近 HPC 用計算機で採用が多い InfiniBand 等の高速インターコネクトでは十分な性能が得られません。

後者の MPI 版は、比較的最近 GAMESS が公式にサポートした手法です。名前の通り、並列計算において現在主流である、MPI 通信を使用したバイナリになります。公式サポートしてからの期間は長くはありませんが、高速インターコネクトの性能を引き出す事ができます。

本書では DDI 版の実行方法について解説します。

また、本書においては、GPU を用いた計算については考慮していません。

## 2.2　rungms の編集方法

GAMESS は、rungms というスクリプトを実行環境に応じて編集し、実行することで動作します。rungms は GAMESS のインストールディレクトリ内に存在します。rungms は、GAMESS の Version によって内容が変化しますので、必ず使用する GAMESS のソースコード内にある rungms を使用してください。

なお、以下の記述は、GAMESS September 30, 2023 R2 Version のものであり、異なる Version の rungms では内容や行数が異なる可能性があります。

GAMESS を実行する前に、必要に応じて rungms の修正を行ってください[2]。

- インストールディレクトリ、出力ディレクトリの指定

  GAMESS のインストールディレクトリ、スクラッチファイルの出力ディレクトリを環境に合わせて修正する必要があります。修正箇所は rungms の 99 行目からの以下の部分です。

```
set SCR=~/gamess/restart
set USERSCR=~/gamess/restart
set GMSPATH=~/gamess
set DEL_OLD=yes
```

  設定している変数は以下の意味となります。

  SCR : 計算途中に作成されるスクラッチファイルが設置されるディレクトリ。
  このディレクトリに実行ユーザーがファイルを書き込めない場合、
  GAMESS は正常に実行されませんので、該当ディレクトリには
  実行ユーザーの書き込み権限が必要となります。

  USERSCR : 計算後に一部の出力ファイルが作成されるディレクトリ。
  SCR の指定と同じく、実行ユーザーの書き込み権限が必要となります。
  また、該当ディレクトリに、新たに作成される出力ファイルと同名の
  ファイルが存在している場合、GAMESS は正常に実行されません。
  USERSCR 以下のファイルの保存が必要ない場合、後述の DEL_OLD の項目
  を yes とする事でファイル消去の手間を省くことが可能です。

  GMSPATH : GAMESS がインストールされているディレクトリ。
  弊社が設定の補助をしている場合、設定済となります。

  DEL_OLD : 以前に同名のインプットを実行し、その際の出力ファイルの一部が
  USERSCR ディレクトリに存在している場合、前に実行した該当ファイルを
  消去して新規の計算を実行するかどうかの切り替え。

  上記の 4 つの変数を、実行環境に応じて編集します。
  複数ノードで GAMESS を実行する場合、上記 4 変数で指定するディレクトリは、計算を実行するすべてのノードから参照できる必要があります。

[2] rungms は csh のシェルスクリプトであるため、csh の文法に従って任意に修正できます。

## 2.3 GAMESS バイナリの実行方法

### 2.3.1 GAMESS 実行時の rungms の引数

前述の通り、GAMESS は実行スクリプト rungms を使用して実行します。実行の際には、rungms の後ろに計算条件を指定する引数（オプション）をつけて実行します。

1 番目の引数：
実行する GAMESS のインプットファイル名。

2 番目の引数：
使用する GAMESS バイナリのバイナリ指定名。この番号は、実行時に使用するバイナリのファイル名 gamess.XX.x の XX の番号を指定します。

3 番目の引数：
rungms を実行する計算機のみで計算する場合には並列計算数を指定します。
それ以外の場合、計算に使用するノード等の計算条件の指定ファイル名を指定します。
詳細については、2.3.2 節以降を参照してください。

以上の 3 つの引数については、GAMESS を実行する際、基本的に必要になります[3]。
GAMESS September 30, 2023 R2 Version では、追加で更に 4 つの引数が使用可能です。
4 番目以降の引数は、特定の条件・機能を使用する場合にのみ設定が必要となり、
必要でない場合には省略が可能です。4 番目以下の引数を省略した場合、
解説内でデフォルトと示された設定で計算が実行されます。

4 番目の引数：（MPI 版のみ有効な設定のため、省略時の影響なし）
MPI 版のみで有効。計算に使用する計算機に、均等に使用するコアを割り振る場合の
各計算機に割り振るコア数。

5 番目の引数： デフォルト設定　0 (論理ノードを使用しない)
DDI 版の場合、1 ノードでの計算実行時のみに使用可能で、使用する論理ノードに
割り振る計算コア数を指定します。論理ノードを使用しない場合、値に 0 を入力します。
MPI 版の場合、指定された値が環境変数 DDI_LOGICAL_NODE_SIZE に設定されます。

[3] 2 番目の引数を省略すると"00"、3 番目の引数を省略すると"1"の値が使用されますが、省略するとノード指定も並列計算もできないため、運用上大きな制限を受けます。

6 番目の引数：（MPI 版のみ有効な設定のため、省略時にも影響なし）
MPI 版のみで有効。OpenMP 並列を行う場合の並列数を指定します[4] 。

7 番目の引数：デフォルト設定　テンポラリファイルの保存を行わない
計算時に作成されるテンポラリファイルの一部を保存する場合の引数です。
この引数を有効にするためには、rungms 内の SCR で指定されるディレクトリ指定と USERSCR で指定されるディレクトリが異なる必要があります。

本書で解説している GAMESS は DDI 版（socket 版）を想定しているため、
4 つ目と 6 つ目の引数は効果がありません。しかし、rungms スクリプトは、
引数を順番で管理しているため、5 番目や 7 番目の引数を使用したい場合、
4 番目と 6 番目の引数として適当な値、例えば "0" を入力してください。

5 番目、7 番目の引数の詳細については、2.3.6 節で詳細に解説を行っておりますので、そちらを参照ください。

[4] GAMESS で OpenMP を有効にするには、MPI 版バイナリのビルドが必要となります

### 2.3.2 GAMESS の実行方法（Job 投入ノード内計算の場合）

まず GAMESS を実行するディレクトリに rungms とインプットファイルをコピーします。GAMESS 実行ディレクトリを ~/gamess-test、
GAMESS のインストールディレクトリが /usr/local/gamess、
インプットファイルが /usr/local/gamess/tests/standard/exam01.inp
の場合、具体的なコマンドは以下となります。

```
$ cd ~/gamess-test
$ cp /usr/local/gamess/rungms
$ cp /usr/local/gamess/tests/standard/exam01.inp .
```

プロンプトで作業しているノードにおいて、N 並列で計算を実行する場合、

```
$ ./rungms exam01 00 N > output.log
```

というコマンドで GAMESS が実行され、output.log という名前でアウトプットファイルが出力されます。（ 00 は数字のゼロ２つです。）

本節のタイトルにもありますが、このように、3 番目の引数に数字を指定した場合には、プロンプトから rungms を実行したノードで、指定した数字分 CPU（コア）を使用して GAMESS の計算が実行されます。

複数のノードを使用した計算を実行する場合、次節の方法で GAMESS を実行してください。

### 2.3.3　GAMESS の実行方法（複数ノードを使用する場合）

前節において、数字を指定した 3 番目の引数ですが、この引数にはファイル名を指定する事が可能です。ファイル名を指定する場合、計算に使用する計算機や、その計算機で使用するコア数等、細かい指定が可能となります。

3 番目の引数でファイル名を指定する場合、計算に使用するノードとそれに割り振るコア数の指定は以下の書式で行います。

```
hpcnode01  4
hpcnode02  8
hpcnode03  4
```

上記の例の場合、計算機 hpcnode01 と hpcnode03 でそれぞれ 4 コアを、
計算機 hpcsnode02 で 8 コアを使用する設定となっています。
上記の設定ファイルの名前を hostfile とした場合、計算を実行する場合には、
実行ディレクトリにインプットファイル（今回の例では exam01.inp）と
rungms、hostfile をコピーした後、以下のコマンドを入力します。

```
$ ./rungms exam01 00 hostfile > output.log
```

細かい注意点として、以下が挙げられます

- ファイル名で実行する場合、Job を投入するノードの指定が必ず必要

3 番目の引数でファイルを指定して実行する場合、rungms を実行するノードに少なくとも 1 コア以上、計算を割り当てる必要があります。
例えば、GAMESS がインストールされている計算機、hpcnode01、hpcsnode02 で

```
hpcnode01  4
```

というファイルを使用する場合、hpcnode01 で rungms コマンドを実行すると正常に計算が終了しますが、hpcnode02 で rungms コマンドを実行すると計算が異常終了します。

計算機がログインノードと計算用ノードで構成されているようなクラスタにおいて、ログインノードから計算ノードに計算を実行させるような場合には､次節以降で解説を行う LSF や PBS 等のジョブ管理システムの使用をご検討ください。

- ファイル名が数字だった場合の取り扱い

rungms の 3 番目の引数は、まずファイル名として取り扱われ、ファイルを探しに行きます。ファイルが見つからなかった場合、次に 3 番目の引数が数字かどうかを判別し、
数字だった場合、コマンドが実行されたノード内で、その数字分コアを使用して計算が実行されます。
そのため、計算条件を記載したファイル名を数字の名前で作成しても、3 番目の引数で指定すればきちんと読み込める事となります。
ただし、計算条件を記載したファイル名を数字にすることは、後で実行コマンドを再確認した場合等に間違いの元になる可能性がありますので、弊社としてはお勧めしません。

### 2.3.4 GAMESS の実行方法（LSF 使用の場合）

LSF や PBS 等、ジョブ管理システムを使用して GAMESS を実行する場合、rungms を修正する必要があります。具体的には、ジョブ管理システムが割り当てる計算機の情報を取得し、rungms 内で GAMESS 実行オプションに引き渡す作業が必要となります。

HPC SYSTEMS が GAMESS をセットアップした場合、LSF 用の GAMESS 実行スクリプト、rungmsLSF を用意しております。

rungmsLSF の使用方法は以下となります。LSF の設定が有効になっている状態で、実行ディレクトリにインプットファイル（今回の例では exam01.inp）と rungmsLSF をコピーした後、以下のコマンドを入力します。（X：並列数、Y：出力ファイル名）

```
$ bsub -n X -o Y ./rungmsLSF exam01 00 X
```

上記の例では、3 番目の引数までしか記載していませんが、7 番目の引数まで対応しています。ただし、2.3.1 節に記載した通り、5 番目の引数は（DDI 版では）1 ノード時限定で使用できるオプションですので、LSF が 2 ノード以上を割り振ってしまった場合には有効になりません[5]。

LSF は bsub コマンドでジョブを投入時に、`-m` オプションや `-R "span[ptile=XX]"` オプションで、投入するノードや、それぞれのノードに投入するコア数を制限することが可能です。5 番目の引数を使用する場合、それらのオプションを使用することも検討ください。

[5] 複数ノードが割り振られた場合、5 番目の引数は無効（デフォルト値の 0）の扱いになります。

### 2.3.5　GAMESS の実行方法（PBS 使用の場合）

PBS を使用する場合にも、rungms 内で PBS が割り当てる計算機の情報を GAMESS 実行オプションに引き渡す修正が必要となります。

HPC SYSTEMS が GAMESS をセットアップした場合、PBS 用の GAMESS 実行スクリプト、rungmsPBS を用意しております。
rungmsPBS は、ジョブスクリプトを使用して実行する想定となっております。

最初に、GAMESS ジョブ投入用スクリプトを作成します。例として以下のようになります。

```
#!/bin/bash

#PBS -j oe
#PBS -o pbs-gamess.log
#PBS -q workq
#PBS -l nodes=2:ppn=4

cd $PBS_O_WORKDIR
./rungmsPBS exam01.inp 00 8
```

使用ノードや並列数をどうするかについては、このジョブ投入用スクリプト内で指定します。指定方法については PBS のマニュアルを参照ください。

投入方法は、PBS の設定が有効になっている状態で、以下のように実行します。
作成したジョブ投入用スクリプトを pbs-gamess.sh とした場合、実行ディレクトリにインプットファイル（今回の例では exam01.inp）と rungmsPBS をコピーした後、
以下のコマンドを入力します。

```
$ qsub ./pbs-gamess.sh
```

前頁のジョブ投入スクリプトの場合、pbs-gamess.log という名前の出力ファイルが作成されます。

全ページのジョブ投入スクリプトの例では、3 番目の引数までしか記載していませんが、7 番目の引数まで対応しています。ただし、2.3.1 節に記載した通り、5 番目の引数は（DDI 版では）1 ノード時限定で使用できるオプションですので、PBS が 2 ノード以上を割り振ってしまった場合には有効になりません。5 番目の引数を使用する場合には、ジョブ投入スクリプト内で必要な設定を行ってください。

### 2.3.6　rungms 実行時の 5 番目、7 番目の引数の詳細解説

以下、rungms の 5 番目と 7 番目の引数についてより詳細に解説します。

最初に、5 番目の引数の解説です。前頁の解説にある"論理ノード"とは、

"計算する際に用いられるリソースのブロック"

の事です。

この論理ノードの使用は、rungms の 5 番目の引数の解説コメントに、
" logical node size (how many cores per logical node), as used by GDDI runs."
と、GDDI 計算を実行するような場合に使用されると書かれています。また、
"Simple guide: you may like to define NGROUP logical nodes, where NGROUP is a parameter in $GDDI." と、GDDI 計算用の設定パラメータ、NGROUP についても軽く触れられています。

具体例として、1 ノード環境（計算機名 hpcnode01）でインプット exam01.inp を 8 コアで並列計算する場合を考えてみます。

最初に、普通に（論理ノードを使用せずに）計算する場合です。

7 番目の引数で指定できる一部出力ファイルの保存を行わない場合、
実行コマンドは以下となります[6]。

```
$ ./rungms exam01 00 8
```

この場合、計算ログ内に以下のように出力されます。
(XXXXXXXXX は変数 SCR で指定したディレクトリ名)

[6] インプットファイル名の拡張子が ".inp" の場合、1 番目の引数の指定で拡張子を省略する事が可能です

```
/usr/local/gamess/ddikick.x  /usr/local/gamess/gamess.00.x  exam01
-ddi 1 8 hpcnode01.local:cpus=8 -scr XXXXXXXXX
```

“ -ddi ”の後の数字が、順に、

ノード数、トータル並列数、計算に使用する計算機名：左の計算機で使用するコア数

の意味となります。上記の例ですと、

ノード数：1、トータル並列数：8、計算機 hpcnode01 で 8 コアを使用

という意味になります。
今回は論理ノードを使用していないため、8 コア全ての計算が、計算機 hpcnode01 に１まとまりで投入されています。

次に、同じ環境で、論理ノードに割り振るコア数を 2 として、一部出力ファイルの保存をしない場合で exam01 を投入する場合を考えてみます。
実行コマンドは以下のようになります。

```
$ ./rungms exam01 00 8 0 2
```

この場合、計算ログ内に以下のように出力されます。
(XXXXXXXXX は変数 SCR で指定したディレクトリ名)

```
/usr/local/gamess/ddikick.x  /usr/local/gamess/gamess.00.x  exam01
-ddi    4    8    hpcnode01.local:cpus=2    hpcnode01.local:cpus=2
hpcnode01.local:cpus=2 hpcnode01.local:cpus=2 -scr XXXXXXXXX
```

-ddi の後の数字を見てみると、

ノード数：4、トータル並列数：8、“計算機 hpcnode01 で 2 コアを使用”が 4 組

という指定に変化しています。

今回の例では、5 番目の引数に値を指定しなかった場合と異なり、１ノード内で
計算しているにもかかわらず、ノードを 4 つ使用するような設定となっています。
ここで、ノード数が 4 となっているのは、5 番目の引数で指定したコア数の指定が
2 なので、トータルコア数の 8 を分割して 4 となっています。

上記から分かるように、論理ノード数によるトータル 5 番目の引数で指定するコア数は、
トータルコア数で割り切れる数である必要があります。

続いて、7 番目の出力ファイルの保存について解説を行います。

例えば、テストインプット exam01.inp を 4 並列計算する際に作成されるファイル
“ exam01.F08 と exam01.F10”を保存したい場合には、rungms 実行時に
以下のような引数をつけます。
この際、以前にも記載した通り、rungms 内の SCR 変数で指定するディレクトリと
USERSCR 変数で指定するディレクトリが異なる必要があります。

```
$ ./rungms exam01 00 4 0 0 0 F08,F10
```

この場合、USERSCR ディレクトリで指定したディレクトリに、保存したファイルが
コピーされます。例えば USERSCR に ~/gamess ディレクトリを指定した場合、
以下のようなファイルが作成されています。

```
$ ls ~/gamess
exam01.dat       exam01.F08       exam01.F08.001       exam01.F08.002
exam01.F08.003  exam01.F10
```

このうち、F08 ファイルについては、並列数分のファイルが作成されるため、
今回の例では 4 つファイルが作成されています。

# 3　既知の問題点

## 3.1　論理ノードを使用し、かつ NGROUP を使用した場合の動作

rungms の 5 番目の引数は、論理ノードの設定を行うために使用されますが、
この引数の解説コメントに、

" logical node size (how many cores per logical node), as used by GDDI runs."
" Simple guide: you may like to define NGROUP logical nodes, where NGROUP is a parameter in $GDDI."

と、GDDI 計算用の設定パラメータ、NGROUP についても軽く触れられています。

この NGROUP について調査を行った所、以下のような状況であることが判明しました。

- GAMESS は`/usr/local/gamess/tests` 以下に様々なテストインプットが存在しているが、その内で、NGROUP の値を 1 以外に指定された状態のインプットは存在しません。
- テストインプット `/usr/local/gamess/tests/fmo/PA.inp` において、以下の記述があります：

  "! If you want to run say on 10 cores, then uncomment:
  ! $gddi ngroup=10 $end"

  このインプットの該当部分の!をコメントアウトして 1 ノードにて動作確認をした所、インプットが異常終了しました。この異常終了は、5 番目の引数で論理ノードを指定した場合でも、3 番目の引数で計算を実行するノードをファイル形式で指定した場合でも発生しました。
- テストインプット `/usr/local/gamess/spectra/parallel/vscf-coord-ddi.inp` でも、

  ```
  $scf       dirscf=.true. $end
  --- $gddi     ngroup=2 $end
  $zmat     izmat(1)=1,2,1 1,3,1 2,3,1,2 $end
  ```

  のように、ngroup の指定が無効化されています。このインプットを修正し、ngroup の指定

を有効にして実行してみましたが、1 ノード環境において、正常に計算が終了しませんでした。

前述の２つのインプットにおいて ngroup を有効にした場合のエラーメッセージは共通しており、

forrtl: severe (43): file name specification error, unit 26, file "Unknown"

というものであり、計算途中にファイルの展開に失敗している可能性が考えられます。

以上のように、今回のセットアップした GAMESS では、rungms のコメントで触れられていた NGROUP の指定を有効にしたインプットにおいて 5 番目の引数を有効にした場合、
テストインプットの正常動作を確認できませんでした。

前述した通り、今回テストした GAMESS のテストインプットは、ソースに同梱されている、テストインプットを修正しない状態では NGROUP が有効になっているものではありません。
そのため、弊社で実行したテストにおいて、そもそも NGROUP のテストとして有効なインプットを使用していない可能性が存在します。

上記の事情のため、正常に動作しなかった原因については断言できませんが、エラーメッセージの内容から推測すると、プログラムのバグの可能性も疑われます。

また、GDDI を始めとした一部の計算については、複数ノードで検証を行っている形跡もあるため、1 ノード環境での NGROUP の使用は想定されていない[7]可能性も考えられます。

[7] 少し古い資料ですが、次においては、明らかに複数ノードを想定して記述されており、資料内では、"SMP の CPU を分割して別々に実行することはできない"と明記されています。
https://www.cms-initiative.jp/ja/events/cmsi-kobe-event/mshyid/handson_1/fmo_130207-3/at_download/dggg7s.pdf

# 付録A

## A.1 HPC システムズ お問い合わせ先


弊社ホームページ　http://www.hpc.co.jp/support_index.html
サポート案内やお問い合わせの多い内容など様々な情報を掲載しております。
是非ご活用ください。

**HPC システムズ株式会社**
〒108-0022　東京都港区海岸 3-9-15 LOOP-X 8 階

HPC 事業部


【営業】03-5446-5531　【サポート】03-5446-5532
お電話によるサポート受付は祝日、弊社指定休日を除く月曜日から金曜日の 9:30～17:30 とさせて頂きます。


【FAX】03-5446-5550


【電子メール】hpcs_support@hpc.co.jp