卒業論文

第一原理計算ソフトVASPのための
高速実行環境の構築

関西学院大学　理工学部　情報科学科
４６１３　山本　洋佑

2008年　2月

指導教員　　西谷　滋人　教授

概 要

西谷研究室では第一原理計算ソフト VASP(Vienna Ab-initio Simulation Package) の入力ファイルを GUI を用いて容易に作成できるソフト MedeA により作成を行い，VASP で計算を実行して研究を進めている．しかし，100 原子程度の現実的な系を計算しようとすると，並列環境の整っていない計算環境では，数日という計算時間を要する．したがって，西谷研究室では VASP の高速化を目指し，マシン単位での並列化を可能とする SGE（Sun Grid Engine）と，CPU 単位での並列化を可能とする MPI（Message Passing Interface）の導入により計算の並列化を実現し，計算時間の短縮を計ってきた．

ところが計算資源となるマシンの変更，及び遠隔地への移設により，現在は異機種混在環境となっており，ファイルの共有等が困難である．そこで現在の環境下でも，並列計算を実現する環境の再構築，また並列計算が手軽に実行できるシステムの構築を行った．また計算結果の確認の容易化のため，gnuplot による E-V 曲線の視覚化を目指した．

結果，MPI の導入により計算の CPU 単位での並列化が可能となり，計算時間短縮を実現した．また SGE の導入によってマシンレベルでの並列化にも成功した．さらに SGE 導入のメリットとして，ジョブの実行過程に於いてもリソースの確保，ジョブに制限を持たせる等の細かい管理が可能となった．さらにシェルスクリプトによって，ジョブの作成，投入はもちろん，使用する CPU の設定までも容易に行えるようになり，結果の確認も手軽に行えるようになった．

目 次

第1章　序論

第一原理計算ソフト VASP(Vienna Ab-initio Simulation Package) は平面波基底ー擬ポテンシャル法電子構造を非常に高速かつ高精度に計算するプログラムであり，固体物性が再現できることから広く使われている．また西谷研究室ではVASPの入力ファイルの作成をMedeAの使用によって行っている．MedeAは，VASPの入力ファイルを図1.1のようにGUIを用いて容易に作成できるソフトであり．結晶構造のデータベース検索から，モデルの構築，計算，解析までを全て一つのプラットフォーム上で行うことができる．これらはすべてWindowsシステム上で稼働し，また他のマシンへジョブを投入することも可能である．デフォルトの設定ではジョブを直接複数のマシンへ投入して実行する手法をとるが，1つのジョブを1CPUでしか実行できない，自動でスケジューリングができない，また実行するための新しいマシンを追加する時にコストがかかる等の問題が生じる [1].

さらに100原子程度の現実的な系をVASPで計算しようとすると，並列環境の整っていない環境下では，数日という計算時間を要する．したがってVASPの高速化は重要な課題である．しかし，1CPU処理能力の向上には限界があり，最新のCPUを設置，または最新のCPUを搭載しているPCの導入は経済的に困難である．従って限られた計算資源の処理能力を前提として，実行時間の短縮を実現する環境の構築を考えなければならない．そこで，1つの処理を複数の計算資源へ分散することにより計算時間の短縮を計る並列処理は，高速化を計る有力な手段である [2][11].

並列処理は，複数のCPUがメモリを共有するという共有メモリ型と，メモリは各CPUにローカルに接続されているという分散メモリ型に分類される．本研究で対象としたLinuxクラスタのような分散メモリ型の並列処理では，各CPU上でそれぞれプロセスを実行させ，そのプロセスが互いにメッセージを送受信することで，協調的な並列処理を進める方法が一般的である．このようなメッセージの送受信を利用した並列処理を，メッセージパッシングと呼び，メッセージパッシングを実現するためのライブラリーの仕様としてはMPI(Message Passing Interface)が最も一般的である．MPIとは，C言語もしくはFortranで利用されるライブラリーの仕様の名称である．MPIを実際に利用するための具体的なパッケージとしては，米国アルゴンヌ国立研究所とミシシッピ州立大学で開発されているMPICHが標準である．

一方，図1.2のようにネットワーク上に存在するコンピュータ資源を結びつけ，一つの複合したコンピュータシステムとしてサービスを提供するシステムとして

グリッドコンピューティングがあり，そのツールキットとして，Sun Microsystems 社が公開しているグリッドコンピューティングシステム構築ソフトウェア，SGE（Sun Grid Engine）がある．SGE はジョブをバッチ処理するためのツールとして利用可能であり，複数の PC で構成された環境に大量のジョブが投入されたとき，ジョブを自動的にスケジューリングし，各 PC にジョブを割り振ることができる．また，大規模なジョブを実行する前，実行過程においてもリソースの確保，また確保されたリソースに小規模なジョブを割り当てることができる．

昨年度は同一機種において上記内容に基づく並列計算を実行する環境を造り上げた．しかし現在は新たなマシンの導入，また計算資源となるマシンの変更により図 1.3 のように異機種混在環境となっており，ファイルの共有等が困難である．そこで現在の環境下でも，並列計算が手軽に実行できるシステムの構築を目指し，計算発行を行うシェルスクリプトの作成，また並列計算実行環境の再構築を行った．また昨年度まで先述した MPICH による MPI を利用していたが，本研究ではインテル提供の MPI を利用し CPU 単位での並列化を行っている [5][13][14]．またマシンレベルでの並列化，計算実行環境の高管理性の実現を目指し，SGE も使用した．

さらに VASP による計算結果の確認において，より良く理解するため，2 次元もしくは 3 次元のグラフを作成するための高性能なコマンドアプリケーションであり，入力した数式, データ等から，図 1.4 のように画面もしくは画像ファイルへグラフの生成を実現する gnuplot を使用した．gnuplot の画像ファイルのフォーマットは JPEG など，多くの形式に対応しており，互換性に優れており，E-V 曲線の作成を容易に行う事が可能となったため，計算結果の確認の容易化も目的の 1 つとした [16].

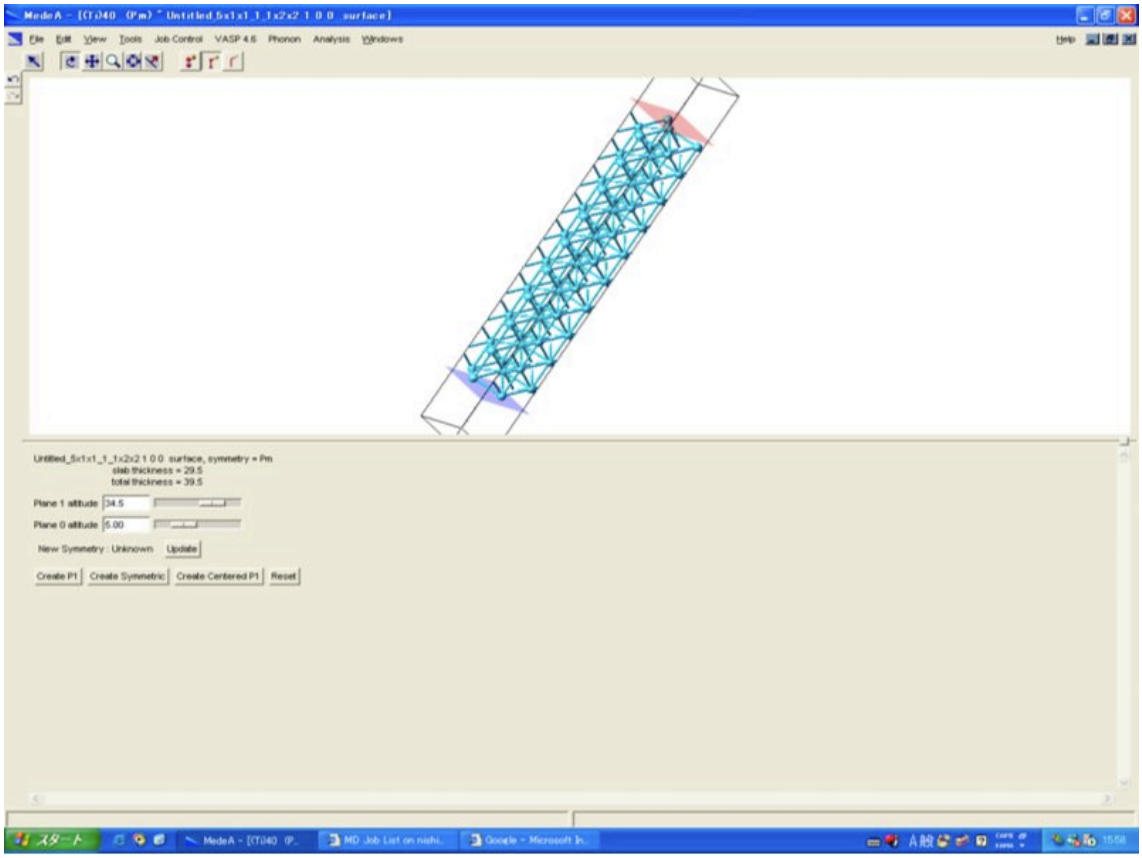

図 1.1: MedeA による格子モデルの作成

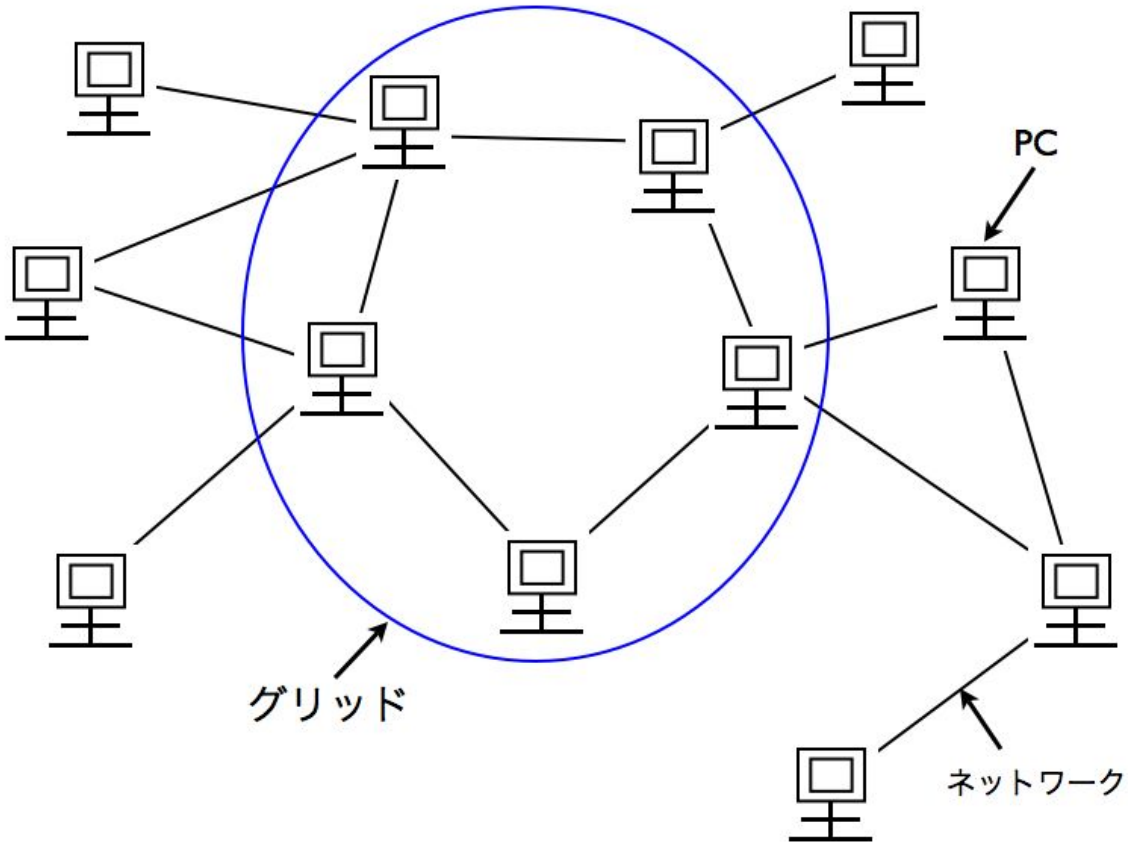

図 1.2: グリッド環境

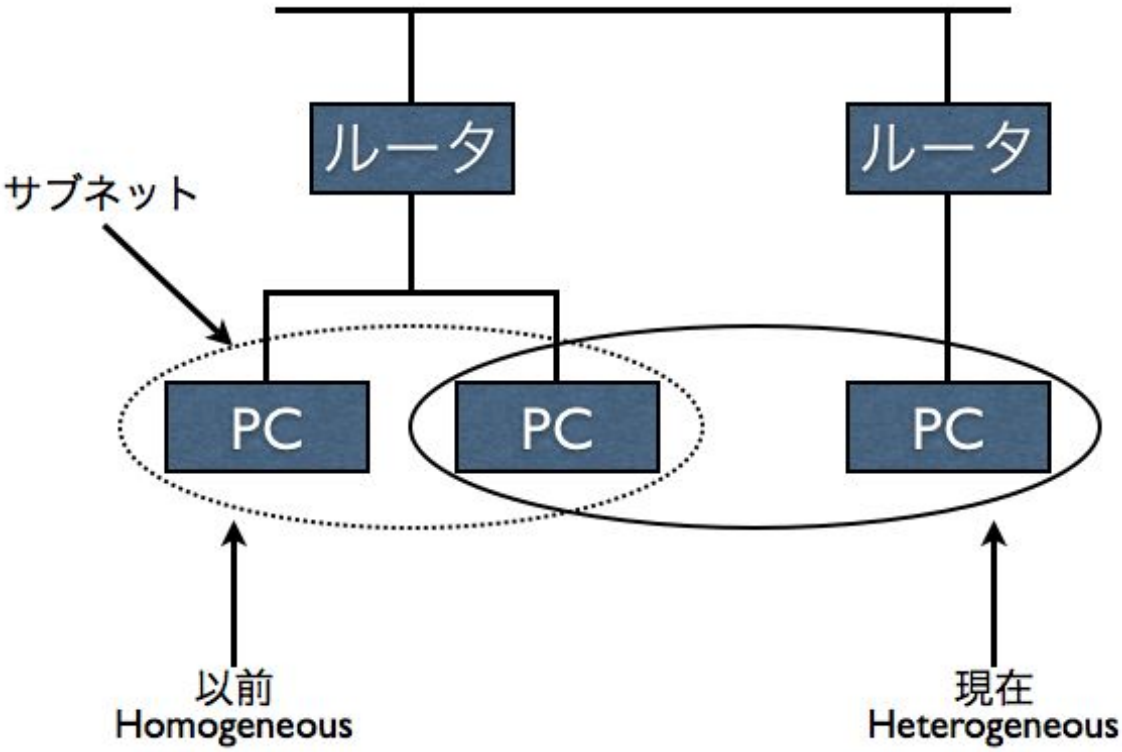

図 1.3: 西谷研究室の計算実行環境

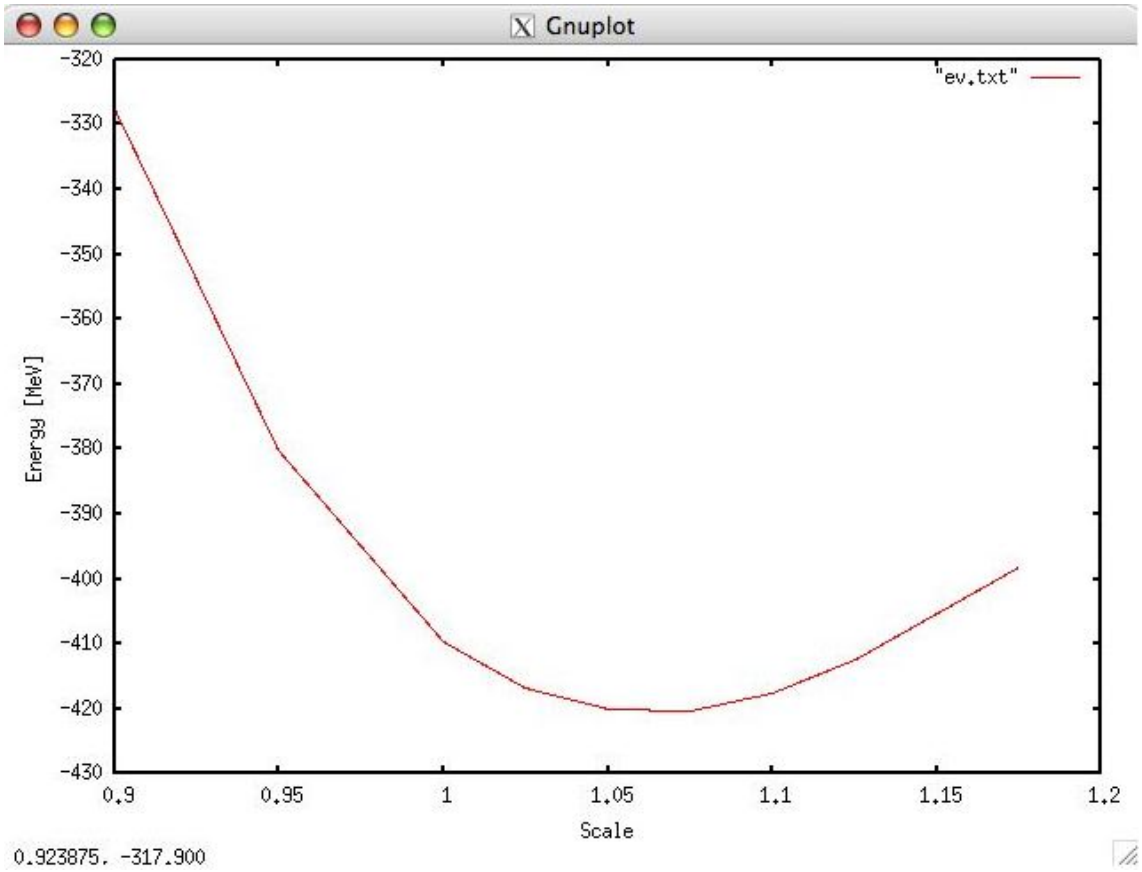

図 1.4: gnuplot による E-V 曲線の視覚化

第2章　手法

VASP の実行効率の向上を目的とし，以下の行程を行った．

- 大量の計算発行などの効率化，またリソースの管理の容易化を目指し，SGE を導入．
- VASP の実行速度向上を目指し，MPI を導入．
- 大量の計算発行，計算結果の確認，視覚化，再計算等の容易化を実現する Ruby によるシェルスクリプトの作成を行う．

なお，MPI の導入については 2.4，ならびに SGE の導入については 2.2 に示した．

2.1　SGE(Sun Grid Engine) 解説

Sun Grid Engine とは Sun Microsystems 社が公開しているグリッドコンピューティングシステム構築ソフトウェアである．グリッドコンピューティングとはネットワーク上の様々なリソース（コンピュータ，ストレージなど）を仮想的に 1 台のコンピュータとして利用可能にする技術である．グリッドコンピューティングの基本技術はバッチシステムであり，小規模なクラスタシステムではバッチジョブシステムとして利用可能である [6].

グリッドコンピューティングは，現在多くのエンジニアリング市場に最も有効であり，非常に多くのリソースを使用する技術計算分野において，積極的に採用されている．グリッドコンピューティングの採用による最も重要なメリットは，ユーザーは何も考慮する事なく，システムリソースを有効に活用できる事であり，トータル・ターン・アラウンドの削減に繋がる．また，オープンソースのソフトウェアであることも，大きなコスト削減効果を生み出している [7].

本研究では主にジョブをバッチ処理するためのツールとして使用しており，SGE によって複数の PC で構成された環境に大量のジョブが投入されたとき，ジョブを自動的にスケジューリングし，各 PC にジョブを割り振ることができる．これによって図 2.1 のように複数のプロセスのマシンレベルでの並列化を実現した．また，大規模なジョブを実行する前，実行過程においてもリソースの確保，また確保されたリソースに小規模なジョブを割り当てることができる．さらに SGE は 1 つのジョブを並列化することができないが，MPI と併用する事によって並列化することが可能となる．

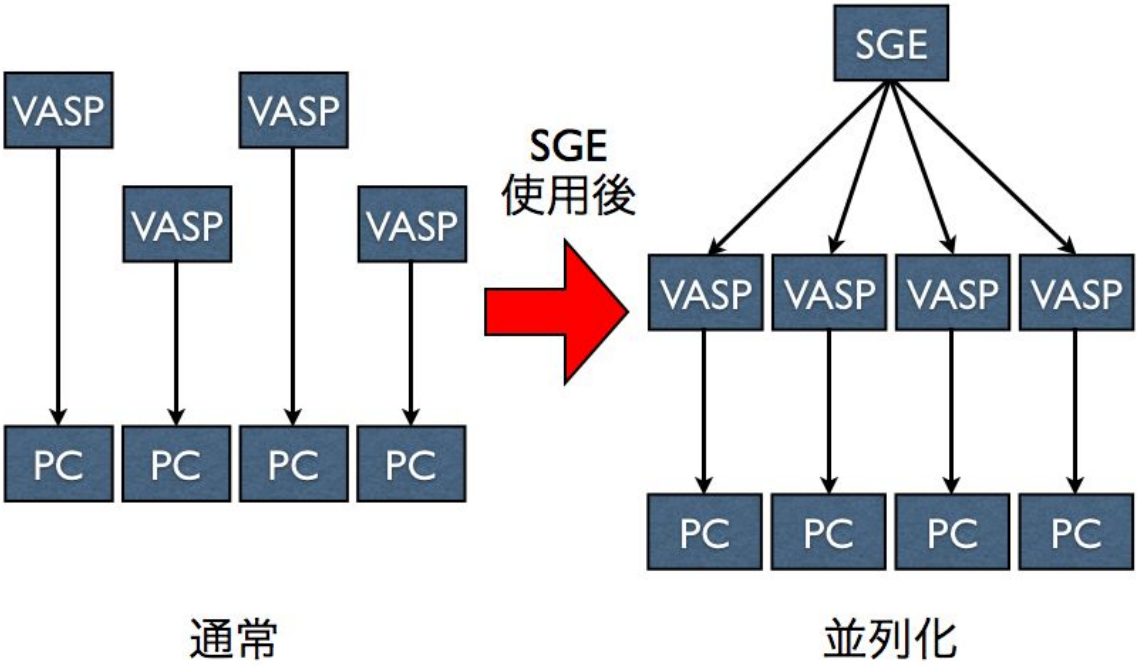

図 2.1: SGE によるマシン単位での並列化

2.1.1 特徴・利点

SGE でグリッドコンピューティングシステムの構築に伴い，ネットワーク上のリソースを有用に活用する事が可能となり，生産性を高める事が実現される．グリッドコンピューティングの主な特徴，機能，利点を表 2.1 に示した [9].

2.1.2 主な機能

SGE で利用できる主な機能を以下に示す [12].

インタラクティブジョブのサポート コンパイルやテストの実行などの対話実行が必要とされるインタラクティブジョブを効率よく実行するためにスケジューリングを行う.

パラレルジョブのサポート MPI 等で作成されたパラレルプログラムに必要なリソースを適切に管理し，他のシングルジョブと同様にスケジューリングを行う.

ジョブ実行前後でシステム指定のプロシジャを実行可能 キュー毎にジョブの実行前後に実行するプロシジャを定義する事ができる.

表 2.1: グリッドコンピューティングの主な特徴、機能、利点

特徴	機能	利点
クラスタグリッドの構築	グリッドコンピューティングの基本構成であり，クラスタ内のリソースを有効に利用できる．	リソースの使用率の向上，および最適な管理が行える．ユーザーがリソースの状況を気にする事なく，本来の設計・開発業務に専念できる．
動的なリソースのバランシング	スケジューリング機能により，投入されたジョブやユーザーの要求に対応して，利用可能なリソースを動的に提供する．	リソースに対する容易なアクセスを提供するのでユーザーはどのリソースが使用可能なのか考慮する必要がない．アイドル状態のリソースは必要とされているジョブに割り当てられるのでリソースの利用率の向上する．
アレイジョブ	パラメトリックジョブ（同じジョブのパラメータを変更して繰り返し実行する）の実行を支援する．約 200,000 個のジョブを一度にサブミットが可能．	多数のジョブが発生するパラメトリックジョブの作業効率を改善する．
パラレルエンパイロメント（PE）管理	既存の並列アプリケーション（MPI 等）の並列実行の管理が可能である．	並列処理アプリケーションを効率よく実行する事が可能である．
シャドウマスタホスト	マスタマシンに障害が発生した時，シャドウマスタとして定義されたホストが，グリッド管理を引き継ぐ．	高可用性を実現する．
自動ジョブの再実行	ジョブの実行中のホストに障害が発生した時，他のホストでジョブを再実行する．	高可用性を実現する．

ユーザ，グループによる権限の設定 ユーザー，グループによってキューの使用を制限する事が可能である．また特定のユーザーに管理者，オペレーターといった権限を与え，権限レベルによってバッチジョブシステムの管理を分担することができる．

GUI によるジョブ管理，および，システム管理 ジョブのサブミットや操作，キューの管理，システム構成の変更を GUI で実現する．

ジョブ情報のロギング 実行したジョブの情報をロギングし，様々なパラメータでの検索が可能．

カレンダーによるキュー制御 曜日，日時などでキューの動作を変更可能．

外部のスケジュールプログラムが利用可能 ジョブスケジューラとして広く使用されている Maui Scheduler 3.2 と組み合わせて使用できる．

2.1.3 ジョブの実行

SGE におけるジョブの実行イメージを図 2.2 に示した．SGE と従来のバッチジョブシステムではキューの概念が異なり，SGE ではキューを意識する事なくジョブをサブミット（投入）する事ができる．ユーザーがサブミットしたジョブにはスケジューリングにより自動的に利用可能なリソースが割り当てられ，発行されたジョブの必要とするリソースの空きがない場合，ペンディング（先延ばし）ジョブとしてキューに格納される．なお実行順序はジョブに付加されたプライオリティ（優先度），スケジューリング方式によってサブミットされる順番が異なる場合がある．なお本研究ではプライオリティに関して，ユーザー，またジョブの種類による優劣はつけず，そのジョブの要求する CPU 数が多ければ多い程，プライオリティを高く設定した．

2.1.4 動作条件

SGE の動作条件を表 2.2 に示した．ジョブを実行するエグゼキューションホストでは，ジョブの特性に適したプロセッサ，メモリ容量，スクラッチ用ディスク容量が必要である．

2.2 SGE のインストールと環境設定

2.2.1 NFS サーバーの設定

- ここでの作業は管理ホストで行う．

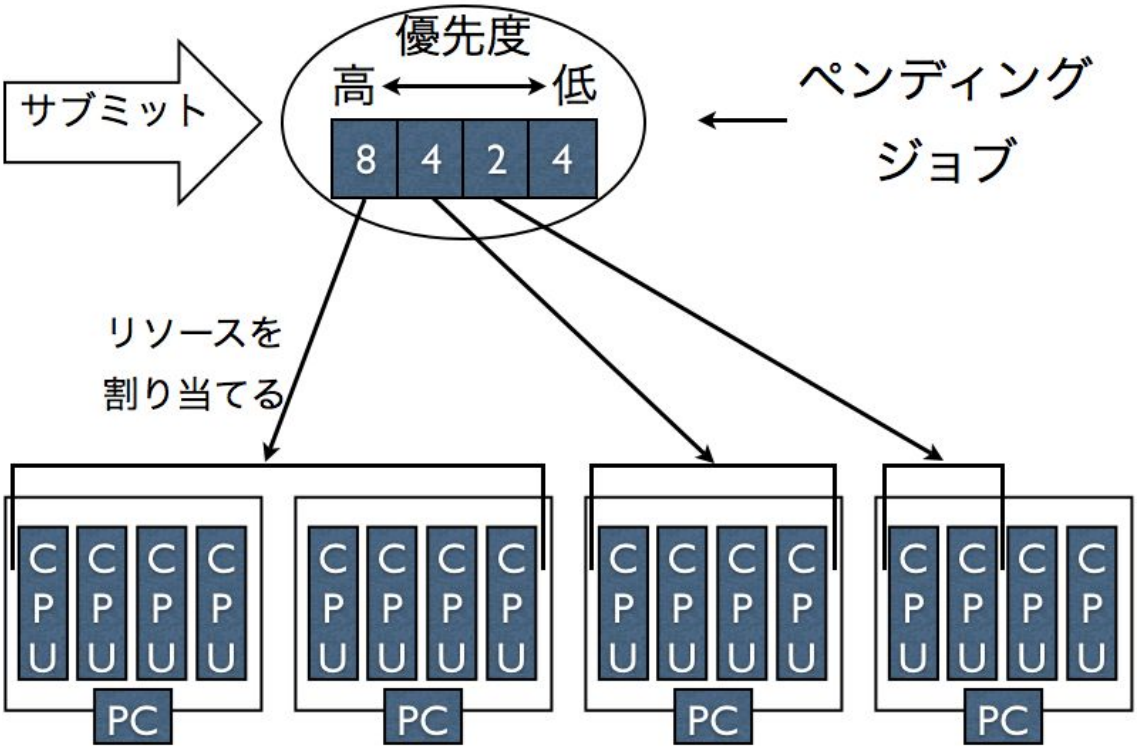

図 2.2: ジョブの実行状況

表 2.2: SGE の動作条件

項目	条件
CPU	クロック数 200MHz 以上の IA-32 プロセッサ
OS	Linux（2.2 以上のカーネルと glibc2.1.3 以上が必要）
メモリ	マスタホストでは 64MB 以上
共有ディスク容量	150MB 以上

- ここでの作業は root で行う．

NFS のアクセス制御を行う設定ファイルに公開するディレクトリとアクセスを許可するホスト名，権限などを記述する [3][4]**．**

```
asura0:~ # cat /etc/exports
/usr/intel      *(rw,no_root_squash,async)
/usr/local/vasp *(rw,no_root_squash,async)
/home           *(rw,no_root_squash,async)
```

「/etc/exports」の設定を有効にする．

```
asura0:~ # exportfs -a
```

登録の確認を行う．

```
asura0:~ # exportfs -v
```

NFS を起動する．

```
asura0:~ # /etc/init.d/nfs start
```

NFS の起動設定を行う．

```
asura0:~ # chkconfig nfs on
```

NFS の起動設定を確認する．

```
asura0:~ # chkconfig --list | grep nfs
```

2.2.2 SGE 管理ホストのインストール

- ここでの作業は管理ホストで行う．
- ここでのログインは root で行う．そのため SGE 管理ユーザーは root となる．なお SGE 管理ユーザーは root である必要はなく，SGE 管理専用のユーザーを作成しても良い．なお，本文では root で行っている．

この行程はグリッド環境の構築の中心となる作業であり，簡単な設定ミスが，トラブルの原因となるので注意が必要である．

インストールの個々の作業は，ほとんどが誤っても再度行う事で，上書きされ修正が可能となっている．しかし，SGE のグリッドエンジンのインストーラーは，状況を確認しながら進むので，以前行った作業が途中の段階である場合，先に進めなくなる事があるので要注意である．

SGE のパッケージの入手する．

http://gridengine.sunsource.net/

ファイルはコモンファイルとプラットフォームごとにあるバイナリファイルの両方を入手する．なるべく最新のパッケージを利用する．また具体的な手順は以下のようになる．

- 上記サイトに接続する．
- 左メニューの Get GE Binaries の GE 6.1 Binaries をクリック.
- メインページ下の Download Grid Engine の here をクリック.
- メインページ下の License Agreement の I Accept をクリック.
- 1. Download common files (necessary for all architectures) の Grid Engine common files をダウンロードする．

  sge-6.1-common.tar.gz を入手.
- 3. Download platform-specific binary and data files の Linux - x86, kernel 2.4, 2.6, glibc >= 2.3.2 をダウンロードする．

  sge-6.1-bin-lx24-x86.tar.gz を入手.

SGE のパッケージの展開を行う．

上記のダウンロードしたファイルを，/usr/sge に配置し，展開を行う．

```
asura0:~ # cd /usr/sge/
asura0:/usr/sge # tar zxvf sge-6.1-common.tar.gz
asura0:/usr/sge # tar zxvf sge-6.1-bin-lx24-x86.tar.gz
```

非常に多くのファイルが展開され，この中のファイルを用いてグリッドをインストールし，この中のコマンドによりグリッドを動作させる．また，グリッドの設定ファイルも保存されるため，不用意にファイルを削除や修正を行ってはならない．

「/etc/services」の編集による SGE 用のポートの設定を行う．

ここで用いるポート番号は，グリッドの中で統一されていれば，以下のように 536，537 に限定されるわけではない．ただし，他のポート番号と重ならないようにし，1024 以下になる必要がある．Sun Microsystems 社のインストールドキュメントを見ると，536，537 を利用しているため，ここではそれにあわせ，以下に記述した 2 行を追記する．

```
asura0:~ # cat /etc/services
     :              :
     :              :
     :              :
sge_qmaster     536/tcp
sge_execd       537/tcp
```

SGE 管理ホストのインストールプログラムを実行する．

ここでグリッドエンジンのサーバーの構築を行う．先に展開した/usr/sge/内の，インストールプログラム install_qmaster を利用する，

```
asura0:~ # cd /usr/sge/
asura0:~ # ./install_qmaster
```

インストールスクリプトのメッセージは英語であるが，簡単な質問に答えるだけでインストールは進み，殆どの入力はデフォルトでよい．また管理アカウントをもつユーザーを記述の際は，SGE 管理ユーザーを記入する．なお本書の場合は root と記入する．

SGE の動作確認を行う．

まずグリッドエンジンのデーモンが動作していることを確認する．

```
asura0:~ # ps ax |grep sge
10969 ?        Sl    28:12 /usr/sge/bin/lx24-amd64/sge_qmaster
10989 ?        Sl    37:57 /usr/sge/bin/lx24-amd64/sge_schedd
11041 ?        S    240:19 /usr/sge/bin/lx24-amd64/sge_execd
31684 pts/3    S+     0:00 grep sge
```

上記のように以下の 2 つが共に表示されていれば正常に動作している．

/usr/sge/bin/lx24-amd64/sge_qmaster

/usr/sge/bin/lx24-amd64/sge_schedd

SGE の依頼ホストの登録を行う．

```
qconf -as SV_HOST.DOMAIN
```

SGE の登録された依頼ホストの確認を行う．

```
qconf -ss
```

SGE の実行ホストの登録を行う．

ここでは依頼ホストを実行ホストとして登録する事も可能であり，また複数の実行ホストの登録も可能である．

```
qconf -ah CL_HOST.DOMAIN
```

SGE の登録された実行ホストの確認を行う．

```
qconf -sh
```

以上の設定では，管理ホストのインストールにおいて，管理対象のホストとして登録されていると，依頼ホストおよび実行ホストとしても登録されていることになるが，明示的に登録している．

また以上の依頼ホスト，実行ホストの登録，削除は共に 1 人のユーザーが変更する事によって，グリッド環境を用いる全てのユーザーに影響があるため注意が必要である．

2.2.3 SGE 実行ホストのインストール

- ここでの作業は実行ホストで行う．
- ここでの作業は root で行う．

この作業は，実行ホストにそれぞれログインし，それぞれ作業を行う．なお管理ホストも計算の実行を行うため，最後に行う実行ホストのインストールプログラムのみを実行する．

「/etc/services」の編集による SGE 用のポートの設定

ここでは前章と同様に「/etc/services」の編集によって実現する.

```
asura1:~ # cat /etc/services
     :              :
     :              :
     :              :
sge_qmaster     536/tcp
sge_execd       537/tcp
```

SGE 実行ホストのインストールプログラムを実行する.

以下のインストールは実行ホストのみならず，管理ホストも行う.

```
asura0:~ # cd /usr/sge/
asura0:~ # ./install_execd
```

SGE 管理ホストのインストールの際と同様に，インストールスクリプトのメッセージは英語であるが，簡単な質問に答えるだけでインストールは進み，全ての入力はデフォルトでよい.

SGE の動作確認を行う.

```
asura0:~ # ps ax |grep sge
10969 ?        Sl    28:12 /usr/sge/bin/lx24-amd64/sge_qmaster
10989 ?        Sl    37:57 /usr/sge/bin/lx24-amd64/sge_schedd
11041 ?        S    240:19 /usr/sge/bin/lx24-amd64/sge_execd
31684 pts/3    S+     0:00 grep sge
```

上記のように/usr/sge/bin/lx24-amd64/sge_execd が表示されていれば正常に動作している.

2.2.4 トラブルシューティング

インストールが正常に行うことができない場合，以下の条件を確認する [10].

- 「/etc/hosts」で，自分のマシンのホスト名の IP アドレスが「127.0.0.1」になっていないか.
- 「/etc/services」で，SGE 用のポートを設定しているか.
- 管理ホストで実行ホストの登録を行っているか.（qconf -ah で登録）

2.3 MPI(Message Passing Interface)解説

並列処理は，複数の CPU がメモリを共有するという共有メモリ型と，メモリは各 CPU にローカルに接続されているという分散メモリ型に分類される．本研究で対象となるクラスタのような分散メモリ型の並列マシンでは，各プロセッサ上で並列に動作するプロセスがそれぞれ独立したアドレス空間を持っているため，その並列化インタフェースとしてメッセージパッシング方式を用いる．そのメッセージパッシングライブラリの使用として MPI が最も一般的である．またメリットとしてライブラリレベルでの並列化を実現するため，細かい調整が可能である．[5][13][14]

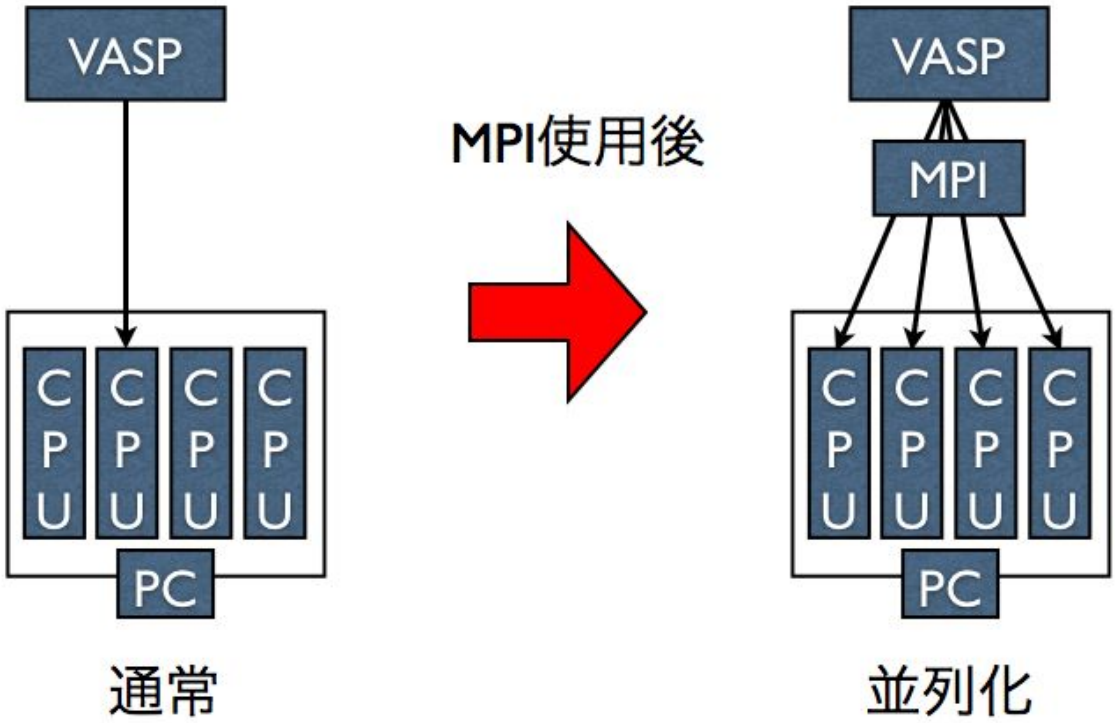

図 2.3: MPI による CPU 単位での並列化

2.4 MPIのインストール

西谷研究室の計算ではインテルの MPI を用いているが，ここでは一般的に入手できるフリーソフト mpich のインストールについて述べる．また SGE 管理ユーザーを MPI 管理ユーザーとし，それらの管理を行うユーザー，sgeadmin のアカウントを作成済みである事を前提とする．なお西谷研究室では SGE 管理，MPI 管理を共に root で行っている [1][10].

2.4.1 mpich のソースの入手

https://www-unix.mcs.anl.gov/mpi/mpich1/

- 上記のサイトにアクセスし，左メニューの Download MPICH をクリック.
- メインページ上部の表中の，Unix 欄の File mpich.tar.gz の http を選択し，ダウンロードを開始する.

2.4.2 ソースの展開

- ここでの作業は SGE 管理ユーザー（asura では root）で行う.
- SGE 管理ユーザーのホームディレクトリで作業を行う.

```
$ tar -zxvf mpich.tar.gz
```

2.4.3 mpich の構築とインストール

- ここでは，各実行ホストにも mpich をインストールしなくてもいいように，SGE_ROOT ディレクトリ内にインストールする.
- なお，ここで NFS により共有されているディレクトリは/home/sgeadmin/sge 以下とする.

はじめに mpich をインストールするディレクトリを作成する.

```
$ mkdir /home/sgeadmin/sge/share
```

次に configure を実行する．またこの行程では多少の時間を要する.

```
asura0:~ # ./configure --prefix=/etc/sge/mpi/mpich
```

次に make によりコンパイルを行い，make install によりインストールを行う.

```
$ make
# make install（この作業は root で行う.）
```

2.4.4 PATH の設定

インストールを行った mpich が使用可能とするため，ここでは以下の内容を記述した/etc/profile.d/mpich.sh ファイルを新規作成し，パスの設定を行う．

```
# cat /etc/profile.d/mpich.sh

#!/bin/sh
export PATH=$PATH:/usr/sge/mpi/
```

上記ファイルを実行可能にする．

```
# chmod 755 /etc/profile.d/mpich.sh
```

2.5 使用するマシン情報

表 2.3 にコンカレントシステムズ社提供であり，本研究で使用したマシンのスペックを示した．CPU はクロック周波数 3.16GHz（現在の最大が 3.20GHz）を誇る Intel Xeon-X5460 Harpertown を 2 つ搭載し，8 コアとなっている．Harpertown は Clovertown の後継の Penryn 世代の CPU で 45nm プロセスで製造され，キャッシュ12MB，トランジスタ数は 8 億 2,000 万，ダイサイズは 107mm2 × 2．45nm プロセステクノロジでは High-k（高誘電率）ゲート絶縁膜とメタルゲートを採用している．また，Half Clock Divider によって 0.5 刻みの倍率でプロセッサを動作させることにより大幅なクロックアップを実現している．

またメモリでは FB-DIMM（Fully Buffered DIMM）を採用している．FB-DIMM はモジュール上に AMB（Advanced Memory Buffer）と呼ばれるチップを持ち，コマンド，アドレス，データの全てがここを経由する．したがって現行のパラレルメモリに置ける配線遅延の問題を解消している．なお，実際には AMB チップは単なるバッファではなく，実質的なメモリ・コントローラであり，プロトコル変換を行うキー・コンポーネントである [15]．

なお OS は OpenSUSE Linux 10.2 を採用している．SUSE Linux を特徴づけるのが　GUI ベースの管理ツールである「YaST」の存在であり，YaST を使用することにより，ネットワークの環境設定，ユーザー管理，さらにはソフトウェアのインストール・アップデートを集中管理することができる．

表 2.3: 使用するマシン

構成	内容
CPU	Intel Xeon-X5460 (3.16GHz) Harpertown × 2
マザーボード	Intel S5400SF Server Board
キャッシュ容量	L2 : 12MB × 2
メモリ	64GB (SMART DDR2-667 ECC FB-DIMM 4GB × 16)
DVD ドライブ	日立 LG GSA-H58N　(DVD-MULTI)
ハードディスク	Seagate ST3160815AS (SATA Ⅱ　１６０ GB)
フロッピディスク	TEAC FD-235HF(2 モード)
ビデオカード	ATI ES1000 Video controller 32MB(on board)
ネットワークカード	Intel 82563EB Gigabit Ethernet Controllers × 2(on board)
OS	OpenSUSE Linux 10.2 インストール
筐体	CA2U(Rev.4)
電源ユニット	Zippy P2P-5700p(700W)
その他	フロント USB ポート　× 1

第3章　結果

3.1　計算実行環境

VASP を実行する並列環境の構築にあたり，MPI を使用した．これにより CPU レベルでの並列化を実現した．また複数のマシンを SGE の利用による並列化によってマシンレベルでの並列化も実現した．さらに計算発行のシェルスクリプトの作成によって一度のプロセスにより，複数のジョブの発行に成功した．この計算実行環境を図 3.1 に示した．

図 3.1 では通常 16 回ものプロセス，つまり手作業で計算を 16 個走らせるのに対し，並列環境の整った環境下では 1 回の手作業で 4 個の計算を走らせる事が可能である．また SGE の導入によって 16CPU を搭載する 1 つのコンピュータとして管理ができるため，計算の発行の際，各マシンへのログイン，ログアウトを繰り返す作業の削減が実現した．さらにジョブの管理，リソースの管理も容易に行え，たとえ 1 つのマシンがダウンしたとしても，そのマシンに投入されていたジョブは再びキューに戻り，再スケジューリングされ，他のマシンに投入される．

また MPI の導入によって各々の計算に対し，4CPU 使用しているため 1 つ 1 つの計算速度の向上が見込める．

3.2　計算の流れ

ここで西谷研究室の計算の流れを図 3.2 に示した．MedeA による格子モデルの作成から VASP の入力ファイルの作成を手作業で行う．この段階ではまだ格子モデルの体積の値はデフォルトとなっているため，さらに異なる体積での計算を行うため，計算を行うべき体積の値を記述した scale.txt の作成を行い，以上の行程を計算の仕込みとする．次に計算の実行を先述したシェルスクリプト allqsub.rb にて行う．また計算の結果確認をプログラム out.rb，また視覚化による確認を graph.gpl で行い，最後に必要ならばシェルスクリプト reqsub.rb により再計算を行う．なお計算の流れの詳細について付録 B に記述した．またシェルスクリプトのフローチャートについては 3.3 に図示し，詳細については付録 A.2 に記述した．

次に先述したシェルスクリプトを用いた具体的な計算手順について図 3.3 に，また計算発行の様子を図 3.4 に示す．

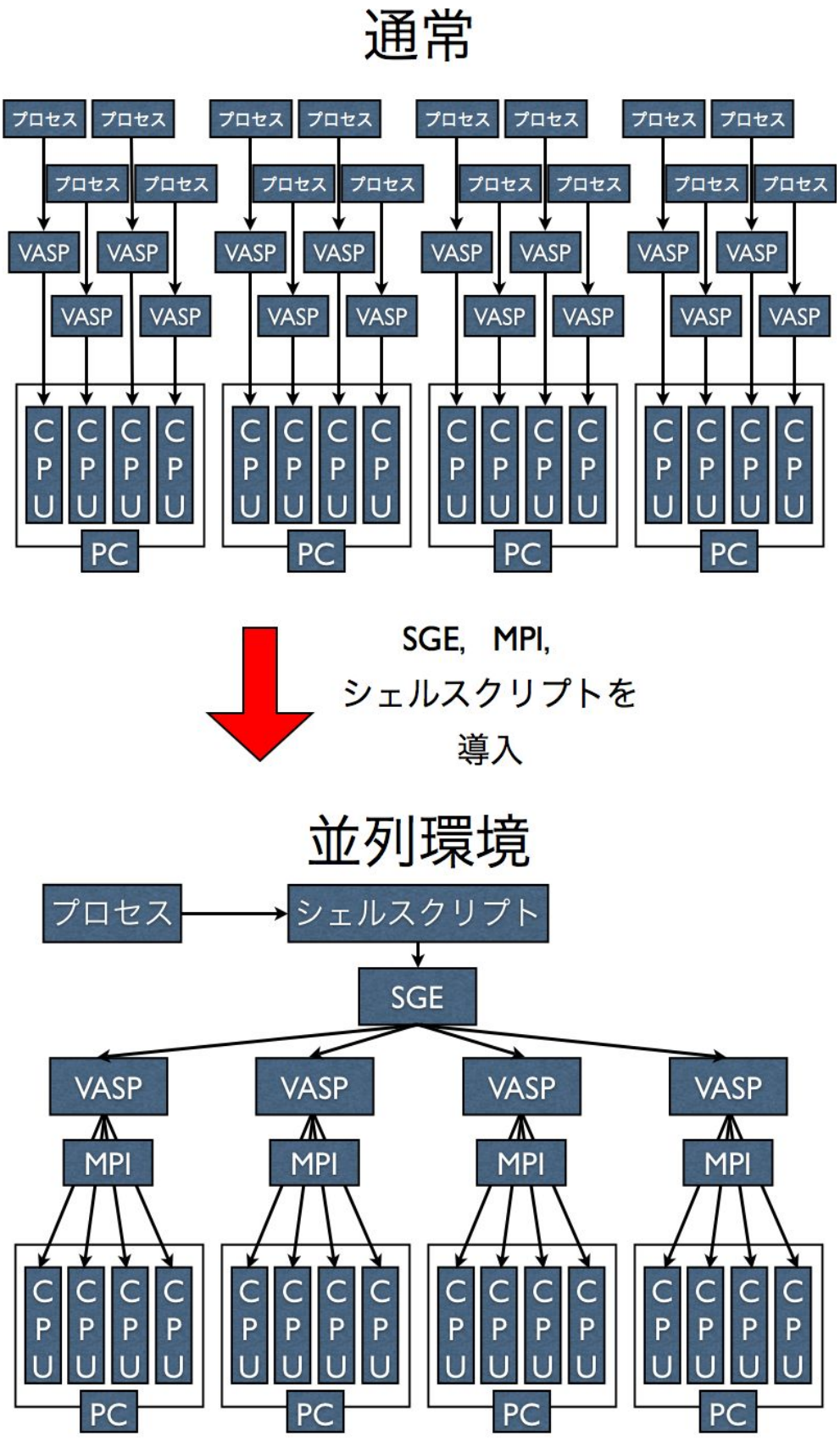
通常
プロセス
VASP
CPU
PC
SGE, MPI,
シェルスクリプトを
導入
並列環境
プロセス
シェルスクリプト
SGE
VASP
MPI
CPU
PC

図 3.1: 計算実行環境

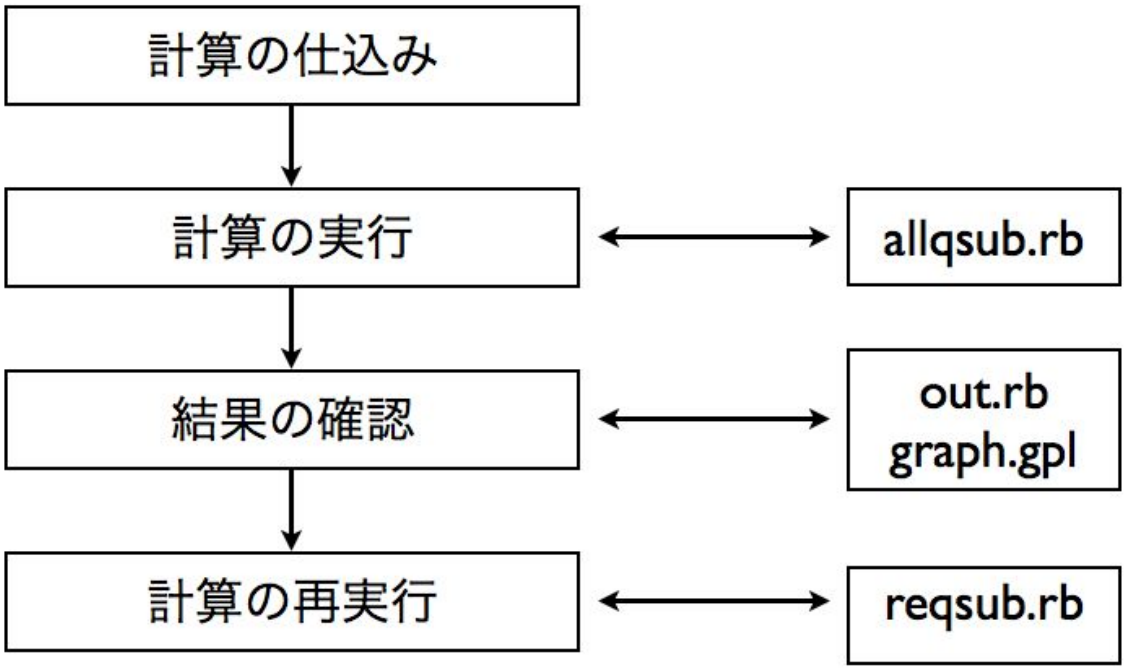

図 3.2: 計算の流れ

3.3 シェルスクリプト，プログラムのフローチャート

ここで各シェルスクリプト，プログラムのフローチャートを図示した．なおgraph.gplのフローチャートは省略する．

3.3.1 allqsub.rb

allqsub.rb は計算を発行するシェルスクリプトであり，多少の仕込み（scale.txtの作成）は必要であるが 1 度の実行により複数のジョブを発行する事ができる．なお，このシェルスクリプトによって発行される各々のジョブは，scale.txt に記述されている各体積の値の計算に相当する．フローチャートは図 3.5 に示した．

3.3.2 out.rb

scale.txt の内容を読み込み，各要素に対応する計算結果を読み込み，１つのファイルにまとめる．なお図 3.6 にフローチャートを示した．

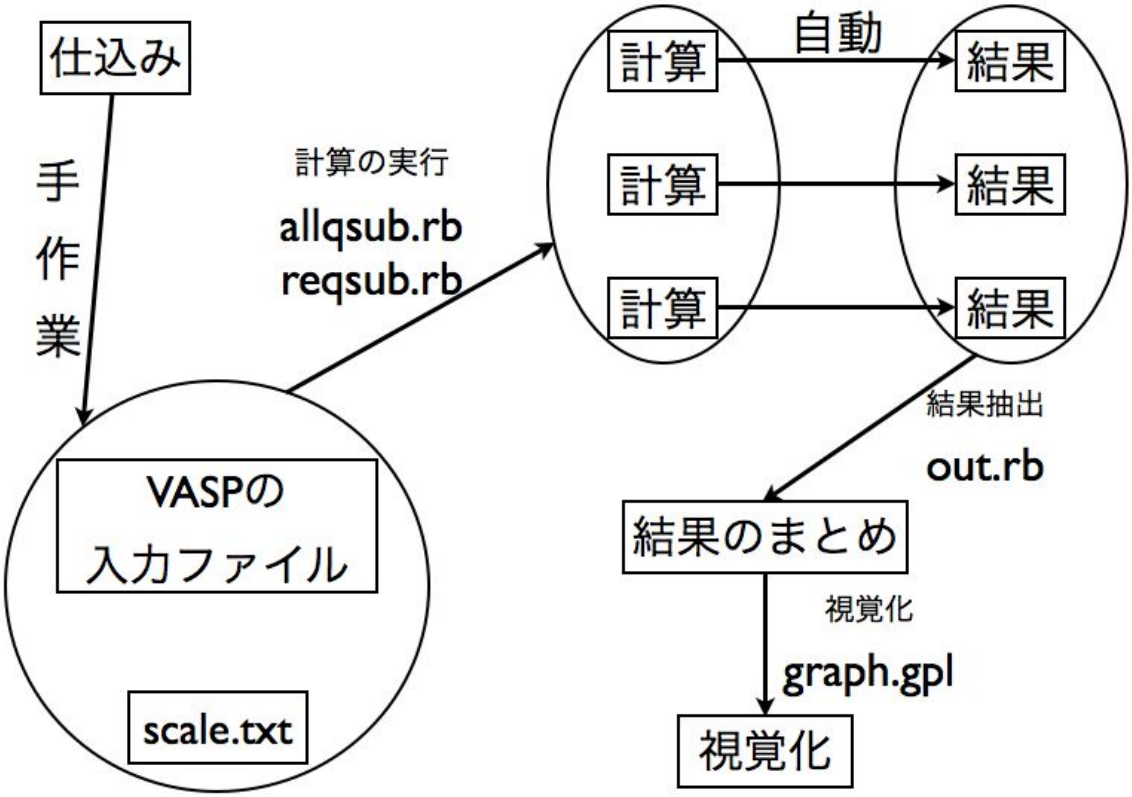
仕込み
手作業
計算の実行
allqsub.rb
reqsub.rb
計算
計算
計算
自動
結果
結果
結果
結果抽出
out.rb
VASPの
入力ファイル
scale.txt
結果のまとめ
視覚化
graph.gpl
視覚化

図 3.3: 計算手順

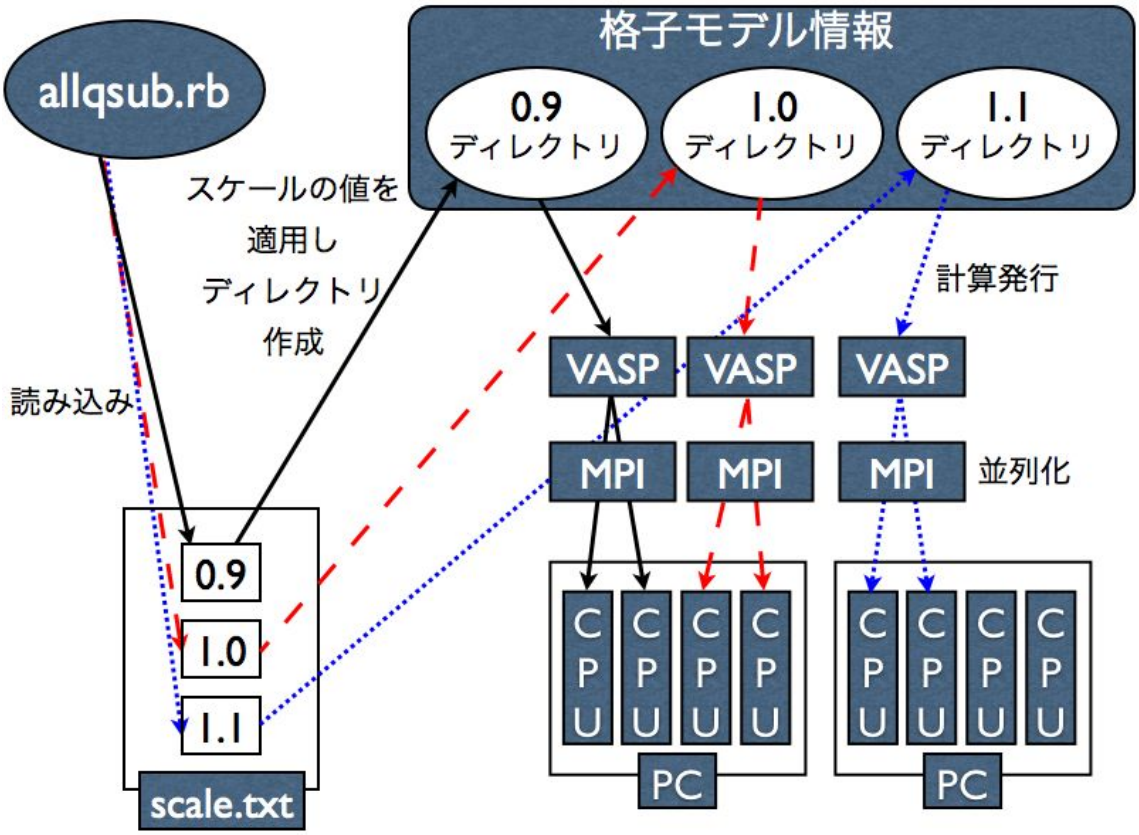
格子モデル情報
allqsub.rb
0.9
ディレクトリ
1.0
ディレクトリ
1.1
ディレクトリ
スケールの値を
適用し
ディレクトリ
作成
読み込み
計算発行
VASP
VASP
VASP
MPI
MPI
MPI
並列化
0.9
1.0
1.1
scale.txt
CPU
CPU
CPU
CPU
PC
CPU
CPU
CPU
CPU
PC

図 3.4: 計算発行の様子

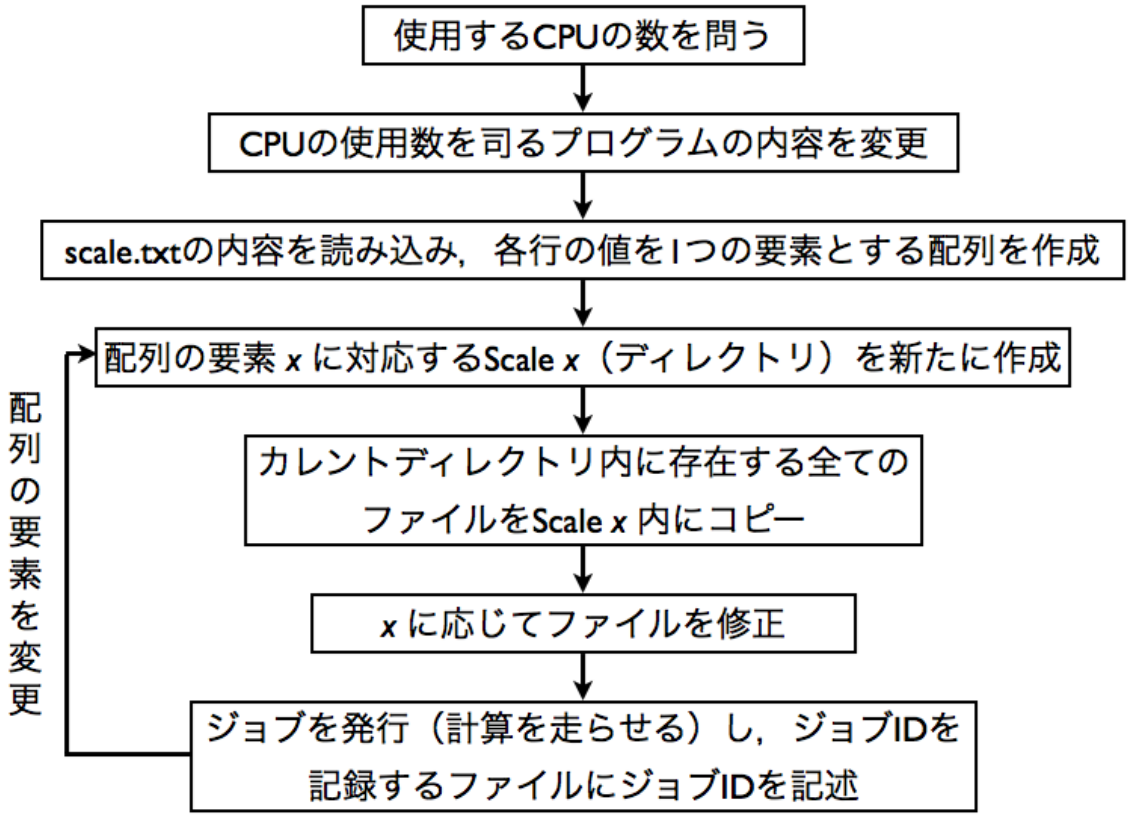

図 3.5: allqsub.rb のフローチャート

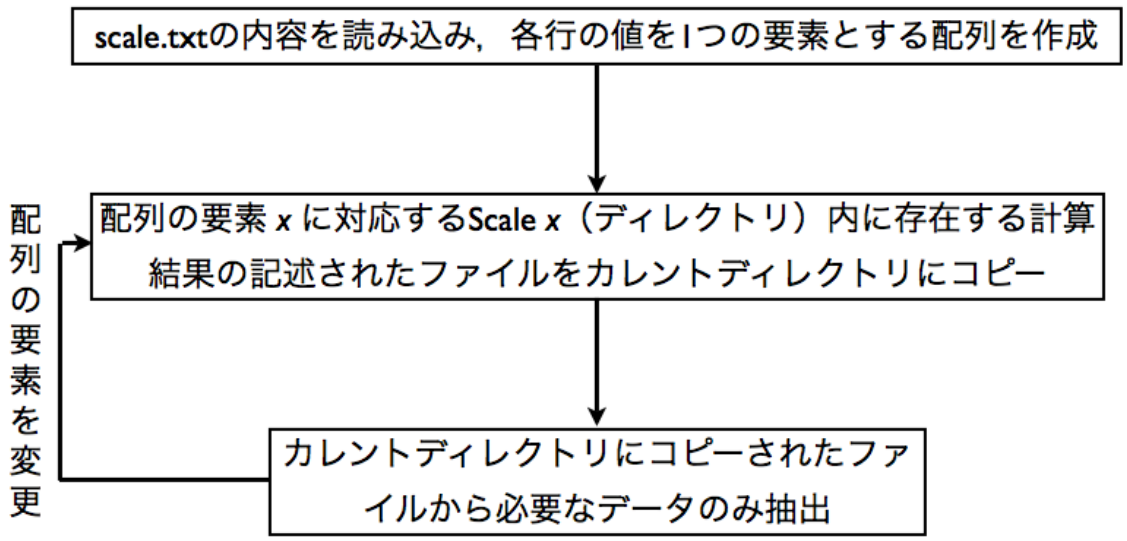

図 3.6: out.rb のフローチャート

3.3.3 reqsub.rb

計算結果確認の後，再計算を行いたい場合このシェルスクリプトを使用する．また，シェルスクリプト実行の後，新たな scale の値を入力する事も可能となっている．reqsub.rb のフローチャートを図 3.7 に示す．

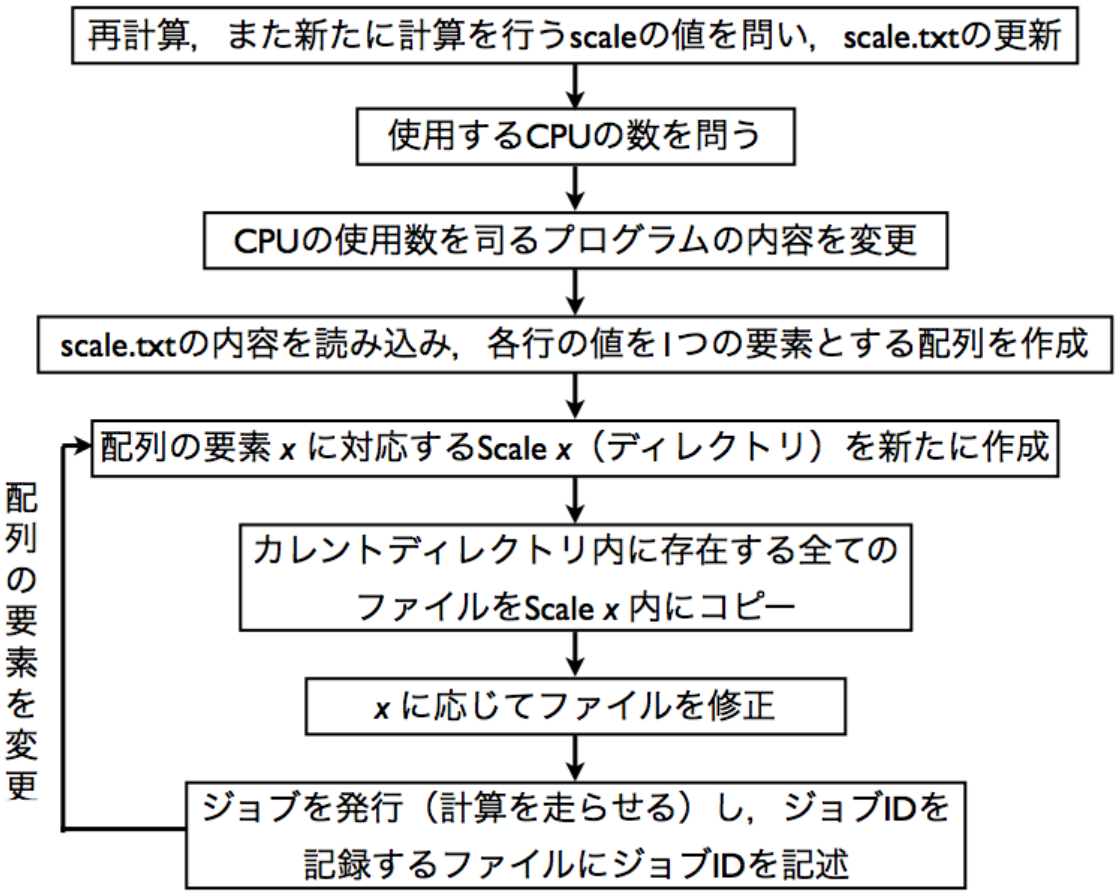

図 3.7: reqsub.rb のフローチャート

3.4 計算時間の計測

VASP のベンチマークとして頻繁に用いられる Hg の電子構造計算をターゲットとし，グリッド環境にジョブが発行されていない事を確認し，計算時間計測を行った．計算機として使用するマシンの持つ CPU がそれぞれ 8 つずつであることから，使用する CPU 数は 1，2，4，8，16 と定めた．また 1CPU 使用時の計算時間を目安とし，2CPU 使用時では 1CPU 使用時の計算時間の 2 分の 1，4CPU では 4 分の 1，8CPU では 8 分の 1，16CPU では 16 分の 1 と目標を設定した．さらに実際にかかった計算時間/目標時間を達成率とした．

結果は表 3.1，図 3.8 に示した．1CPU 使用時と 2CPU 使用時，4CPU 使用時では達成率が 50%を超え，大きく計算時間に変化が現れ，この場合並列化は計算時間の短縮に大いに役立ったといえる．しかし 8CPU 使用時，16CPU 使用時では達成率 50%を下回り，CPU を有効活用しているとは言い難い．特に 16CPU 使用時では 20%と大きく達成率を下げ，8CPU 使用時からさらなる計算時間の短縮を実現したとはいえない．これは CPU のみならず 2 台のマシンに 1 つのジョブを振り分けたため，メッセージの送受信に時間がかかったためである．

今回の時間測定時の状況のように，ジョブが少量しか発行されていない場合，出来る限りリソースを使用する事が，計算時間短縮に繋がる．したがって 1 つ 1 つのジョブの処理時間に焦点を合わせた場合，1 つのジョブに対し，多数にわたる CPU の使用が早期完了につながる．

一方，多量のジョブが発行され，リソースに空きのない場合，全ての計算に目を向け，全てのジョブの処理の完了をいち早く行いたい場合には，1 つのジョブに対し CPU 数を 1 つ，2 つ，あるいは 4 つ程度にしぼる事が英断といえる．

しかし現実に 1CPU，2CPU 程度の使用では数ヶ月にもわたる計算時間を要するジョブが存在し，さらにそのジョブを大量に行わなければならない事もある．その場合，マシンのシャットダウン等のアクシデント（停電など）を考慮すると，安全性を考え 1 つ 1 つのジョブの早期完了が求められるため，多数の CPU 使用が良いと考えられる．

表 3.1: Hg による計算時間計測

使用した CPU 数	理想時間 (秒)	実際の計算時間 (秒)	達成率 (%)
1	70.3	70.300	100
2	35.15	43.747	80
4	17.575	25.910	68
8	8.7875	23.037	38
16	4.394	21.941	20

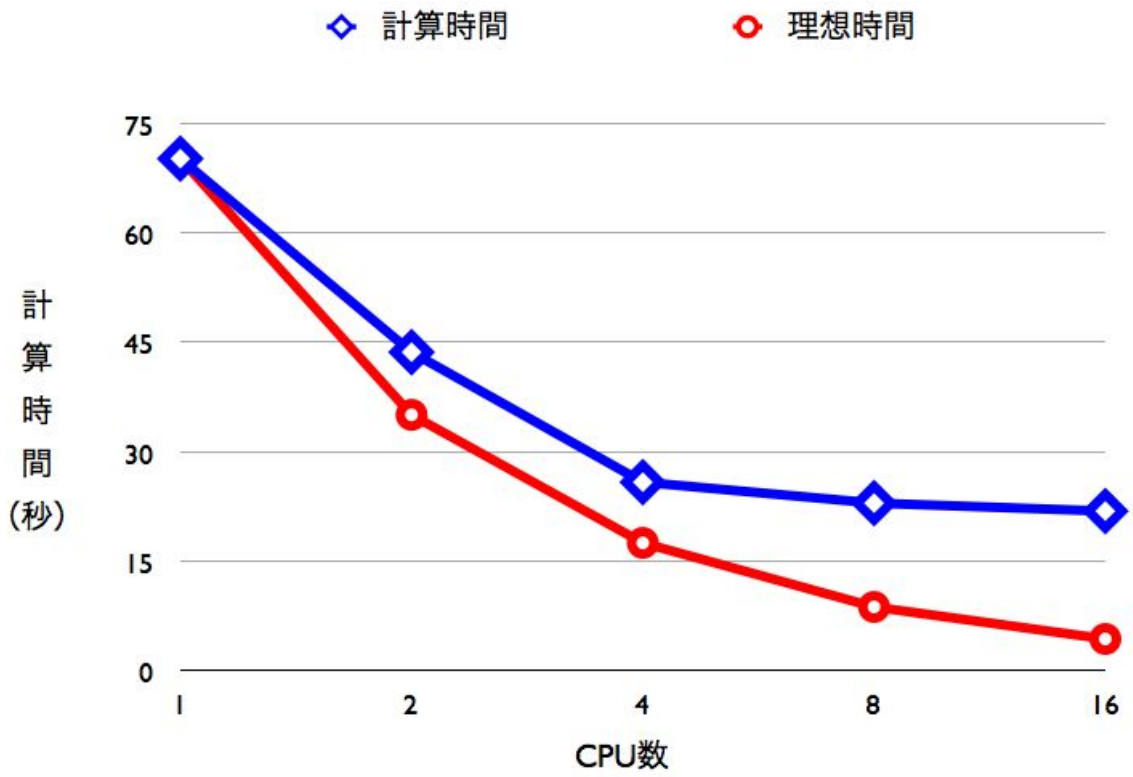

図 3.8: Hg による計算時間計測

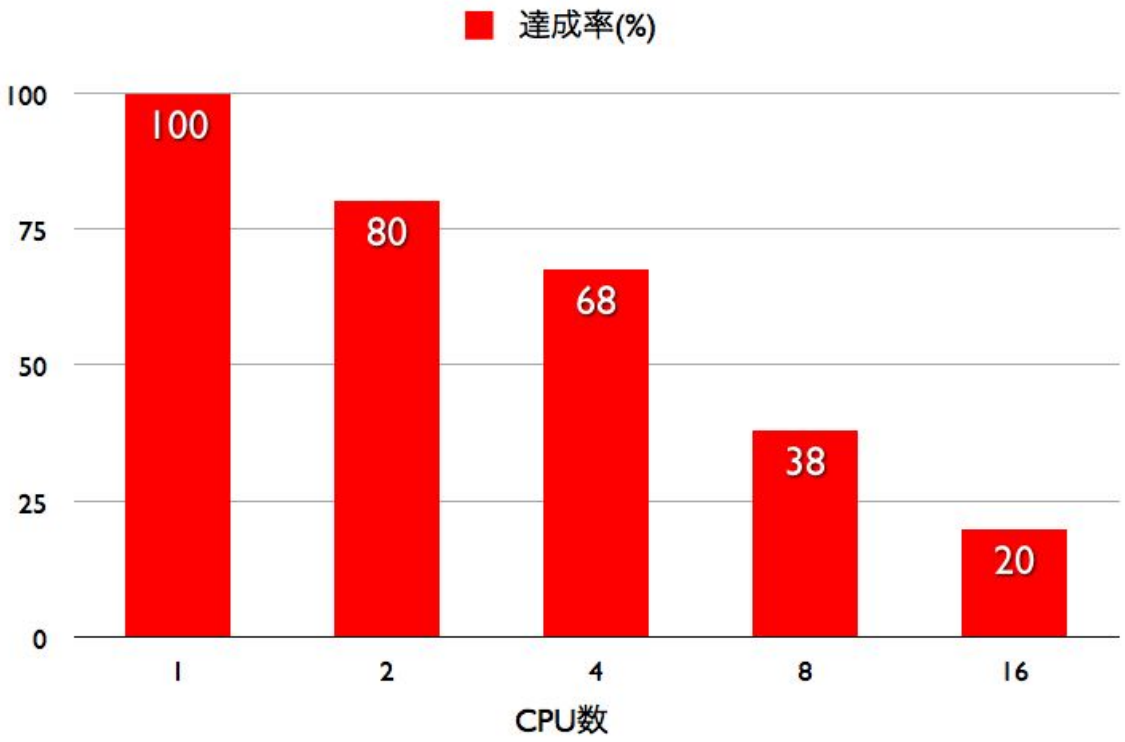

図 3.9: 計算時間の目標達成率

第4章　総括

MPI，SGE の導入により計算速度の短縮につながった．計算時間に関して，使用する CPU 数に比例するとまではいかなくとも，使用する CPU の増加に応じて計算時間の短縮を実現している．なお，1 つのジョブに対し複数のマシンを介する場合，マシン間のメッセージの送受信に多少の時間を要する．したがって 1 台のマシンに設置されている CPU の使用で，計算時間が許容範囲に留まると予測可能な場合，その CPU 数以下の CPU の使用で計算を行うのが良い．

また多量の計算の実行を望む場合も同様に，CPU 数をある程度少数に絞り込むのが，全ての計算の完了への近道となる．しかし，その各々の計算が 1CPU，2CPU 程度の使用では膨大な時間を要する場合，シャットダウン等のアクシデントを考慮に入れると，4CPU，8CPU，またそれ以上の CPU を 1 つの計算につぎ込むのが良いと考えられる．

またシェルスクリプトの作成によって計算発行の手間を省く事が可能となった．多少の仕込みが必要になるとはいえ，計算発行のミス等を犯しにくく，また計算結果の確認も視覚化によって容易になったといえる．

さらにシェルスクリプトによる計算の発行では SGE を介し，ジョブの作成，発行を行うため，SGE の特徴であるジョブの管理性能も損なう事なく，管理しやすい．1 つのマシンがダウンしたとしても，SGE のスケジューリング機能により，そのマシンで実行中であったジョブは再スケジューリングされ，他のマシンに投入され引き続き計算を実行する．また各々ユーザー，ジョブの種類等にプライオリティをつける事も可能で，ジョブの実行する順番などの操作も行える．

グリッド環境の構築によって，本研究では 4 台のマシンをネットワークを介し，1 台のスーパーコンピュータとして研究を進めている．ここで更なるマシンの追加は計算資源の追加と見なす事が可能である．したがってグリッド環境の拡大に比例し，計算機としての性能をアップさせる事ができる．またマシンの追加を行ったとしても SGE による管理性は損なわれず，新マシンの導入以前と同程度の管理量で運営が出来るシステムが構築できた．

4.1 結論

MPI，SGE の導入．またシェルスクリプト作成により以下のメリットを生み出した．

- 計算速度の向上．
- 計算発行の容易化．
- 計算結果確認の容易化．
- ジョブの管理の容易化．
- グリッドの拡大，つまりマシンの追加に比例して計算機としての機能性の上昇．
- マシンの追加，削除に関わらず，高レベルでの管理性を保てる．

4.2 考察

- グリッドの拡大，つまりマシンの追加に比例して計算機としての機能性の上昇．
- マシンの追加，削除に関わらず，高レベルでの管理性を保てる．

付 録A　シェルスクリプト，設定ファイルの概形と配置

A.1　プログラムの配置，エイリアスの設定

A.1.1　計算プログラム，シェルスクリプト，マシンファイルの配置

/usr/local/bin/vasp_program/内に計算を行うプログラムを配置する．

計算を行う全てのユーザがvasp_programを使用するため，/usr/local/，/usr/local/bin/等の計算を行う全てのユーザが共有しやすいように，全てのユーザが共有するディレクトリ内に配置するのが望ましい．

```
yosuke@asura0:/usr/local/bin/vasp_program> ls
allqsub.rb*  hostfile*       mkdir_qsub_asura1.rb*  reqsub.rb*  run.sh*
graph.gpl*   mkdir_qsub.rb*  out.rb*                run.orig*
```

A.1.2　.bashrcの記述，変更．

ホームディレクトリに存在する.bashrcに以下の内容を追記，また.bashrcが存在しない場合，新たに作成し記述する．追記，作成の完了後，1度ログアウトし，再びログインし直し，bashの変更を読み直す．なお以下のようにエイリアスを追記せず，シェルスクリプトを起動させても問題はないが，この作業の完了により，冗長なコマンド入力を省略する事が可能となる [8].

```
alias allqsub='ruby /usr/local/bin/vasp_program/allqsub.rb'
alias mkdir_qsub='ruby /usr/local/bin/vasp_program/mkdir_qsub.rb'
alias mkdir_qsub_asura1='ruby /usr/local/bin/vasp_program/mkdir_qsub_asura1.rb'
alias reqsub='ruby /usr/local/bin/vasp_program/reqsub.rb'
alias qsub='qsub /usr/local/bin/vasp_program/run.sh'
alias out='ruby /usr/local/bin/vasp_program/out.rb'
alias graph='gnuplot /usr/local/bin/vasp_program/graph.gpl'
```

A.2 計算プログラム，シェルスクリプト，マシンファイルの概形

A.2.1 SGEを介し，ジョブを発行するシェルスクリプト

allqsub.rb

```
puts 'How many CPU to use for 1 scale?'
x = gets.chomp

if x.to_i>0 then

  if x.to_i<17 then

    Dir.chdir("/usr/local/bin/vasp_program/"){
      file = open('run.orig')
      text = file.read
      File.open 'run.sh', 'w' do |f|
        f.write text.gsub(/CPU/,x)
      end
    }

    require "fileutils"
    require "find"

    now_dir = Dir.pwd

    File.open("job_number.txt","w")
    File.open("submit_job.txt","w")

    Values = []
    File.read('scale.txt').each{|line|
      Values << line.chomp
    }

    index = 0
    while (index< Values.length)

      Dir.mkdir("Scale"+Values[index])
```

```
  Dir.glob("**/*"){|path|
    next unless File.file?(path)
    FileUtils.cp(path,"Scale"+Values[index])
  }

  Dir.chdir("Scale"+Values[index]){

    file = open('POSCAR.orig')
    text = file.read
    File.open 'POSCAR', 'w' do |f|
      f.write text.gsub(/#SCALE/,Values[index])
    end

  }
  index += 1
end

index = 0
while (index< Values.length)

  Dir.chdir("Scale"+Values[index]){

    File.open("job_number.txt","w")
    dfoutput =`qsub /usr/local/bin/vasp_program/run.sh`
    File.open 'submit_job.txt' , 'a' do |f|
      f.write dfoutput
    end

    File.open("submit_job.txt"){|file|

      while line = file.gets

        Dir.chdir(now_dir){

          File.open 'job_number.txt' , 'a' do |f|
            f.write line.sub(/Your job /,'').sub(/ has been submitted/,'')
.sub(/("vasp-job")/,'').sub('()','').sub(/ /,'')
          end
```

```
            }
          end
        }


      }
      index += 1
    end


  else

    puts 'Sorry, number of CPU is 1~16'

  end

else

  puts 'Sorry, number of CPU is 1~16'

end
```

mkdir_qsub_asura0.rb

```
require "fileutils"
require "find"

now_dir = Dir.pwd

File.open("job_number.txt","w")
File.open("submit_job.txt","w")

Values = []
File.read('scale.txt').each{|line|
  Values << line.chomp
}

index = 0
while (index< Values.length)

  Dir.mkdir("Scale"+Values[index])

  Dir.glob("**/*"){|path|
    next unless File.file?(path)
    FileUtils.cp(path,"Scale"+Values[index])
  }

  Dir.chdir("Scale"+Values[index]){

    file = open('POSCAR.orig')
    text = file.read
    File.open 'POSCAR', 'w' do |f|
      f.write text.gsub(/#SCALE/,Values[index])
    end

  }
  index += 1
end

index = 0
while (index< Values.length)
```

```
  Dir.chdir("Scale"+Values[index]){

    File.open("job_number.txt","w")
    dfoutput =‘qsub -q all.q@asura0 /usr/local/bin/vasp_program/run.sh‘
    File.open ’submit_job.txt’ , ’a’ do |f|
      f.write dfoutput
    end

    File.open("submit_job.txt"){|file|

      while line = file.gets

        Dir.chdir(now_dir){

          File.open ’job_number.txt’ , ’a’ do |f|
            f.write line.sub(/Your job /,’’).sub(/ has been submitted/,’’)
.sub(/("vasp-job")/,’’).sub(’()’,’’).sub(/ /,’’)
          end
        }
      end
    }


  }
  index += 1
end
```

reqsub.rb

```
require "fileutils"
require "find"

now_dir = Dir.pwd

File.open("submit_job.txt","w")

Dir.glob("scale.txt"){|path|
  if path=nil then
  else
    Values = []
    File.read('scale.txt').each{|line|
      Values << line.chomp
    }
  end
}

puts
puts 'Please input the value of a scale newly'
puts

y=gets.chomp
puts

scale =[]
scale = y.split(/ /)

puts 'How many CPU to use for 1 scale?'
x = gets.chomp

if x.to_i>0 then

  if x.to_i<17 then

    #CPUnumber
    Dir.chdir("/usr/local/bin/vasp_program/"){
          file = open('run.orig')
```

```
      text = file.read
      File.open 'run.sh', 'w' do |f|
        f.write text.gsub(/CPU/,x)
      end
}
#CPUnumber

index = 0
while (index< scale.length)

  #rmdir
  Dir.glob("Scale"+scale[index]+"/"){|path|
    if path=nil then
    else

      Dir.chdir("Scale"+scale[index]){
        FileUtils.rm_r(Dir.glob("**/*"))
      }
      Dir.rmdir("Scale"+scale[index])

    end
  }


  #mkdir
  Dir.mkdir("Scale"+scale[index])

  Dir.glob("**"){|path|
    next unless File.file?(path)
    FileUtils.cp(path,"Scale"+scale[index])
  }

  Dir.chdir("Scale"+scale[index]){

    file = open('POSCAR.orig')
    text = file.read
    File.open 'POSCAR', 'w' do |f|
      f.write text.gsub(/#SCALE/,scale[index])
    end
```

```
  }

  index += 1
end

#keisan
index = 0
while (index< scale.length)

  Dir.chdir("Scale"+scale[index]){

    File.open("job_number.txt","w")
    dfoutput =‘qsub /usr/local/bin/vasp_program/run.sh‘
    File.open ’submit_job.txt’ , ’a’ do |f|
      f.write dfoutput
    end

    File.open("submit_job.txt"){|file|

      while line = file.gets

        Dir.chdir(now_dir){

          File.open ’job_number.txt’ , ’a’ do |f|
            f.write scale[index].to_s + ’   ’ + line.sub(/Your job /,’’)
.sub(/ has been submitted/,’’).sub(/("vasp-job")/,’’).sub(’()’,’’).sub(/ /,’’)
          end
        }
      end
    }


  }
  index += 1
end


#add_scale
```

```
  Dir.glob("scale.txt"){|path|
    if path=nil then
      File.open("scale.txt","w")
      lastscale=[]
      lastscale = scale.sort{|a,b| a.to_f <=> b.to_f}

      index = 0
      while (index< lastscale.length)

        File.open 'scale.txt', 'a' do |f|
          f.write lastscale[index] + "n"
        end

        index += 1
      end

    else

      lastscale=[]
      lastscale = Values + scale
      lastscale2 = lastscale.uniq
      lastscale3 = lastscale2.sort{|a,b| a.to_f <=> b.to_f}

      File.open("scale.txt","w")

      index = 0
      while (index< lastscale3.length)

        File.open 'scale.txt', 'a' do |f|
          f.write lastscale3[index] + "n"
        end

        index += 1
      end

    end
  }

else
```

```
    puts 'Sorry, number of CPU is 1~16'
  end
else
  puts 'Sorry, number of CPU is 1~16'
end
```

A.2.2 MPIを用いてジョブの並列化を行うプログラム

run.sh

```
#$ -S /bin/bash
#$ -pe vasp 2
#$ -cwd
#$ -N vasp-job

source /opt/intel/mpi/3.0/bin64/mpivars.sh
source /opt/intel/cmkl/9.1.023/tools/environment/mklvarsem64t.sh

export VASP=/usr/local/vasp/vasp.4.6.28/vasp
mpiexec -machinefile $USER.$PE.$JOB_ID -env I_MPI_PIN_MODE 0 -np $NSLOTS $VASP
```

run.orig

```
#$ -S /bin/bash
#$ -pe vasp CPU
#$ -cwd
#$ -N vasp-job

source /opt/intel/mpi/3.0/bin64/mpivars.sh
source /opt/intel/cmkl/9.1.023/tools/environment/mklvarsem64t.sh

export VASP=/usr/local/vasp/vasp.4.6.28/vasp
mpiexec -machinefile $USER.$PE.$JOB_ID -env I_MPI_PIN_MODE 0 -np $NSLOTS $VASP
```

A.2.3 MPIによるジョブの並列化の際，有効となるマシンの記述ファイル

hostfile

```
asura0:8
asura1:8
asura2:8
asura3:8
```

A.2.4 並列ジョブによる各々の計算結果をまとめるプログラム

out.rb

```
File.open("last.txt","w")
File.open("ev.txt","w")

now_dir = Dir.pwd

require "fileutils"
require "find"

Values = []

File.read('scale.txt').each{|line|
  Values << line.chomp
}

index2 = 0
while (index2< Values.length)

  Dir.chdir(now_dir+"/"+"Scale"+Values[index2]){

    m = ""
    n = ""

    File.read('submit_job.txt').each{|line|
      m =  line.chomp
    }
    n = m.sub(/Your job /,'').sub(/ has been submitted/,'')
.sub(/("vasp-job")/,'').sub('()','').sub(/n/,'').sub(/ /,'')


    FileUtils.cp("vasp-job.o"+n.to_s,Values[index2]+".txt")

    FileUtils.cp(Values[index2]+".txt",now_dir)

    Dir.chdir(now_dir){
      file1 = open(Values[index2]+".txt")
      text1 = file1.read
```

```
      File.open 'last.txt', 'a' do |f|
        f.write  Values[index2] + "n" + text1.grep(/F=/).to_s
      end

      #ev.txt sakusei
      x=[]
      x=text1.grep(/F=/).last.to_s.scan(/F= ................/)

      File.open 'ev.txt', 'a' do |f|
        f.write  Values[index2] + " " + x.to_s.sub(/F= /,"").to_s + "n"
      end

    }

  }

  index2 += 1
end
```

A.2.5 gnuplot を介し，計算結果の視覚化を行うプログラム

graph.gpl

```
set xlabel "Scale"
set ylabel "Energy [MeV]"

#set xrange [0.01:20]
#set yrange [0:2]

set xtics 0.1
#set mxtics 4
#set grid

#set ytics 1
#set mytics 10

plot "ev.txt" w l
pause -1
```

付 録B　シェルスクリプトマニュアル

B.1　使用方法

B.1.1　仕込み

MedeA 等からはかれたデータ（POSCAR,INCAR 等）を要素とする計算を行うディレクトリに移動し，scale.txt，POSCAR.orig を作成する．

```
yosuke@asura0:~/Ti00/Ti00> ls
CHG*      DOSCAR_old*  KPOINTS*  PI15430*       POSCAR.orig*      POSCAR~*
scale.txt*
CHGCAR*   EIGENVAL*    OSZICAR*  POSCAR*        POSCAR.orig~*     POTCAR*
scale.txt~*
CONTCAR*  IBZKPT*      OUTCAR*   POSCAR.fore*   POSCAR.test*      WAVECAR*
vasprun.xml*
DOSCAR*   INCAR*       PCDAT*    POSCAR.fore~*  POSCAR.testfile*  XDATCAR*

yosuke@asura0:~/Ti00/Ti00> cat POSCAR
Tihcp
1.00
         16.225     0.00000000 0.00000000
         0.00000000 5.11025000 0.00000000
         0.00000000 0.00000000 4.68330000
  20
Selective dynamics
Direct
   0.05000    0.08333    0.25000 T F F
   0.05000    0.41666    0.75000 T F F
   0.15000    0.58333    0.25000 T F F
   0.15000    0.91666    0.75000 T F F
   0.25000    0.08333    0.25000 T F F
   0.25000    0.41666    0.75000 T F F
```

```
   0.35000    0.58333    0.25000 T F F
   0.35000    0.91666    0.75000 T F F
   0.45000    0.08333    0.25000 T F F
   0.45000    0.41666    0.75000 T F F
   0.55000    0.58333    0.25000 T F F
   0.55000    0.91666    0.75000 T F F
   0.65000    0.08333    0.25000 T F F
   0.65000    0.41666    0.75000 T F F
   0.75000    0.58333    0.25000 T F F
   0.75000    0.91666    0.75000 T F F
   0.85000    0.08333    0.25000 T F F
   0.85000    0.41666    0.75000 T F F
   0.95000    0.58333    0.25000 T F F
   0.95000    0.91666    0.75000 T F F
```

```
yosuke@asura0:~/Ti00/Ti00> cat POSCAR.orig
Tihcp
1.00
         #SCALE     0.00000000 0.00000000
         0.00000000 5.11025000 0.00000000
         0.00000000 0.00000000 4.68330000
  20
Selective dynamics
Direct
   0.05000    0.08333    0.25000 T F F
   0.05000    0.41666    0.75000 T F F
   0.15000    0.58333    0.25000 T F F
   0.15000    0.91666    0.75000 T F F
   0.25000    0.08333    0.25000 T F F
   0.25000    0.41666    0.75000 T F F
   0.35000    0.58333    0.25000 T F F
   0.35000    0.91666    0.75000 T F F
   0.45000    0.08333    0.25000 T F F
   0.45000    0.41666    0.75000 T F F
   0.55000    0.58333    0.25000 T F F
   0.55000    0.91666    0.75000 T F F
   0.65000    0.08333    0.25000 T F F
   0.65000    0.41666    0.75000 T F F
   0.75000    0.58333    0.25000 T F F
   0.75000    0.91666    0.75000 T F F
```

```
   0.85000    0.08333    0.25000 T F F
   0.85000    0.41666    0.75000 T F F
   0.95000    0.58333    0.25000 T F F
   0.95000    0.91666    0.75000 T F F

yosuke@asura0:~/Ti00/Ti00> cat scale.txt
16.200
16.225
16.250
16.275
16.300
```

B.1.2 マシンと CPU の稼働状況の確認

qstat -f コマンドにより現在のジョブの投入状況を確認する.

```
yosuke@asura0:~/Ti00/Ti00> qstat -f
queuename                      qtype used/tot. load_avg arch          states
---------------------------------------------------------------------------------
all.q@asura0                   BIP   4/8       4.10     lx24-amd64
   4722 0.55500 vasp-job   machiko      r     01/14/2008 15:49:12     2
   4726 0.55500 vasp-job   machiko      r     01/14/2008 15:49:12     2
---------------------------------------------------------------------------------
all.q@asura1                   BIP   6/8       3.89     lx24-amd64
   4728 0.55500 vasp-job   yosuke       r     01/14/2008 22:38:49     2
   4729 0.55500 vasp-job   yosuke       r     01/14/2008 22:38:49     2
   4731 0.55500 vasp-job   yosuke       r     01/14/2008 22:38:49     2
---------------------------------------------------------------------------------
all.q@asura2                   BIP   2/8       4.34     lx24-amd64
   4727 0.55500 vasp-job   machiko      r     01/14/2008 15:49:12     2
---------------------------------------------------------------------------------
all.q@asura3                   BIP   4/8       4.01     lx24-amd64
   4723 0.55500 vasp-job   machiko      r     01/14/2008 15:49:12     2
   4725 0.55500 vasp-job   machiko      r     01/14/2008 15:49:12     2

################################################################################
 - PENDING JOBS - PENDING JOBS - PENDING JOBS - PENDING JOBS - PENDING JOBS
################################################################################
   4728 0.00000 vasp-job   yosuke       qw    01/14/2008 22:38:44     2
   4729 0.00000 vasp-job   yosuke       qw    01/14/2008 22:38:44     2
   4730 0.00000 vasp-job   yosuke       qw    01/14/2008 22:38:44     2
   4731 0.00000 vasp-job   yosuke       qw    01/14/2008 22:38:45     2
   4732 0.00000 vasp-job   yosuke       qw    01/14/2008 22:38:45     2
```

B.1.3 計算の実行

allqsub（コマンド）または mkdir_qsub_asura0（コマンド）等で計算を走らせる．なおバックグラウンドで行う必要はない．コマンドを入力後，1 スケールごとの CPU 使用数を問われるので入力する．また後述するジョブの投入状況の確認により，マシン，CPU の稼働状況を見極め，CPU 数を設定する事が望ましい．

```
yosuke@asura0:~/Ti00/Ti00> allqsub
How many CPU to use for 1 scale?
2
```

上記の作業の後，各スケールに対するディレクトリが作成され，また各々のスケールに対する計算のジョブ ID が記述されたファイル job_number.txt が作成される．

```
yosuke@asura0:~/Ti00/Ti00> ls
CHG*          IBZKPT*   PI15430*       POSCAR.test*      Scale16.250/
scale.txt*
CHGCAR*       INCAR*    POSCAR*        POSCAR.testfile*  Scale16.275/
scale.txt~*
CONTCAR*      KPOINTS*  POSCAR.fore*   POSCAR~*          Scale16.300/
submit_job.txt
DOSCAR*       OSZICAR*  POSCAR.fore~*  POTCAR*           WAVECAR*
vasprun.xml*
DOSCAR_old*   OUTCAR*   POSCAR.orig*   Scale16.200/      XDATCAR*
EIGENVAL*     PCDAT*    POSCAR.orig~*  Scale16.225/      job_number.txt


yosuke@asura0:~/Ti00/Ti00> cat job_number.txt
16.200   4716
16.225   4717
16.250   4718
16.275   4719
16.300   4720
```

各スケールのディレクトリ内にもそのスケールの値に対するジョブ ID が記述されたファイルが作成される．

```
yosuke@asura0:~/Ti00/Ti00> ls Scale16.200
CHG*          EIGENVAL*  OUTCAR*       POSCAR.fore~*     POSCAR~*
scale.txt*
CHGCAR*       IBZKPT*    PCDAT*        POSCAR.orig*      POTCAR*
scale.txt~*
```

```
CONTCAR*     INCAR*     PI15430*      POSCAR.orig~*     WAVECAR*
submit_job.txt
DOSCAR*      KPOINTS*   POSCAR*       POSCAR.test*      XDATCAR*
vasprun.xml*
DOSCAR_old*  OSZICAR*   POSCAR.fore*  POSCAR.testfile*  job_number.txt

yosuke@asura0:~/Ti00/Ti00> cat Scale16.200/submit_job.txt
Your job 4716 ("vasp-job") has been submitted
```

B.1.4 計算結果の確認

結果抽出

計算終了後，out（コマンド）で計算結果をまとめる．

```
yosuke@asura0:~/Ti00/Ti00> out
yosuke@asura0:~/Ti00/Ti00> ls
16.200.txt  CHGCAR*      INCAR*    POSCAR*        POSCAR.testfile*  Scale16.275/
16.225.txt  CONTCAR*     KPOINTS*  POSCAR.fore*   POSCAR~*          Scale16.300/
16.250.txt  DOSCAR*      OSZICAR*  POSCAR.fore~*  POTCAR*           WAVECAR*
16.275.txt  DOSCAR_old*  OUTCAR*   POSCAR.orig*   Scale16.200/      XDATCAR*
16.300.txt  EIGENVAL*    PCDAT*    POSCAR.orig~*  Scale16.225/      ev.txt
CHG*        IBZKPT*      PI15430*  POSCAR.test*   Scale16.250/      job_number.txt

yosuke@asura0:~/Ti00/Ti00> ls Scale16.200/
16.200.txt   EIGENVAL*  PCDAT*         POSCAR.orig~*     XDATCAR*
vasp-job.o4716
CHG*         IBZKPT*    PI15430*       POSCAR.test*      job_number.txt
vasp-job.pe4716
CHGCAR*      INCAR*     POSCAR*        POSCAR.testfile*  scale.txt*
vasp-job.po4716
CONTCAR*     KPOINTS*   POSCAR.fore*   POSCAR~*          scale.txt~*
vasprun.xml*
DOSCAR*      OSZICAR*   POSCAR.fore~*  POTCAR*           submit_job.txt
yosuke.vasp.4716
DOSCAR_old*  OUTCAR*    POSCAR.orig*   WAVECAR*          vasp-job.e4716
```

各スケールに対する計算結果はX.txt（Xはスケールの値）に記述される．また各々のスケールに対する計算結果はそのスケールに対するディレクトリ内に存在するvasp-job.oY（Yはジョブ ID）に記述され，X.txtと等価である．しかしX.txtはoutによって新規作成，更新されるのに対し，vasp-job.oYは計算の進行と共に随時更新される．

```
yosuke@asura0:~/Ti00/Ti00> diff 16.200.txt Scale16.200/vasp-job.o4716
```

```
yosuke@asura0:~/Ti00/Ti00> cat 16.200.txt
 running on    2 nodes
 distr:  one band on    1 nodes,    2 groups
 vasp.4.6.28 25Jul05 complex
 POSCAR found :  1 types and   20 ions
```

```
 -----------------------------------------------------------------------------
|                                                                             |
|  ADVICE TO THIS USER RUNNING 'VASP/VAMP'   (HEAR YOUR MASTER'S VOICE ...):  |
|                                                                             |
|      You have a (more or less) 'large supercell' and for larger cells       |
|      it might be more efficient to use real space projection operators      |
|      So try LREAL= Auto  in the INCAR   file.                               |
|      Mind: If you want to do an extremely accurate calculations keep the    |
|      reciprocal projection scheme          (i.e. LREAL=.FALSE.)             |
|                                                                             |
 -----------------------------------------------------------------------------

 LDA part: xc-table for Pade appr. of Perdew
 POSCAR, INCAR and KPOINTS ok, starting setup
 WARNING: wrap around errors must be expected
 FFT: planning ...          10
 reading WAVECAR
 WARNING: random wavefunctions but no delay for mixing, default for NELMDL
 entering main loop
       N       E                     dE             d eps       ncg     rms
RMM:   1    -0.101093819295E+04   -0.10109E+04   -0.56149E+04  1600   0.282E+02
RMM:   2    -0.163074764769E+03    0.84786E+03   -0.23261E+03  1600   0.517E+01
RMM:   3    -0.185425645845E+03   -0.22351E+02   -0.10818E+02  1600   0.150E+01
RMM:   4    -0.187024522867E+03   -0.15989E+01   -0.90139E+00  1600   0.442E+00
RMM:   5    -0.187176641708E+03   -0.15212E+00   -0.99373E-01  1600   0.150E+00
RMM:   6    -0.187196527585E+03   -0.19886E-01   -0.13756E-01  1600   0.564E-01
RMM:   7    -0.187199350622E+03   -0.28230E-02   -0.19332E-02  1600   0.211E-01
RMM:   8    -0.187199757191E+03   -0.40657E-03   -0.28932E-03  1600   0.810E-02
RMM:   9    -0.187199827225E+03   -0.70034E-04   -0.52500E-04  3106   0.312E-02
RMM:  10    -0.187199830784E+03   -0.35584E-05   -0.25713E-05  2580   0.719E-03
RMM:  11    -0.187199830958E+03   -0.17491E-06   -0.11968E-06  1684   0.168E-03
RMM:  12    -0.187199830988E+03   -0.29264E-07   -0.17352E-07  1486   0.559E-04
0.273E+01
RMM:  13    -0.163579820766E+03    0.23620E+02   -0.11935E+01  3200   0.472E+00
0.563E+00
RMM:  14    -0.162624694203E+03    0.95513E+00   -0.43204E-01  3200   0.941E-01
0.256E+00
RMM:  15    -0.162507377291E+03    0.11732E+00   -0.47198E-02  3200   0.263E-01
RMM:  16    -0.162503526355E+03    0.38509E-02   -0.25565E-03  3200   0.649E-02
```

```
RMM:  17    -0.162503335991E+03    0.19036E-03   -0.89236E-05  3200   0.103E-02
RMM:  18    -0.162503320451E+03    0.15541E-04   -0.47888E-06  2646   0.309E-03
RMM:  19    -0.162503307599E+03    0.12852E-04   -0.35102E-07  2020   0.818E-04
RMM:  20    -0.162503309302E+03   -0.17027E-05   -0.25803E-08  1336   0.166E-04
   1 F= -.16250331E+03 E0= -.16244072E+03  d E =-.162503E+03
```

すべてのスケールに対する計算結果の全てのエネルギーは last.txt に記述される.

```
yosuke@asura0:~/Ti00/Ti00> cat last.txt
16.200
   1 F= -.16250331E+03 E0= -.16244072E+03  d E =-.162503E+03
16.225
   1 F= -.16247362E+03 E0= -.16241143E+03  d E =-.162474E+03
16.250
   1 F= -.16243967E+03 E0= -.16237782E+03  d E =-.162440E+03
16.275
   1 F= -.16240791E+03 E0= -.16234644E+03  d E =-.162408E+03
16.300
   1 F= -.16237489E+03 E0= -.16231374E+03  d E =-.162375E+03
```

全てのスケールに対する計算結果の最終的なエネルギーの値は ev.txt に記述される.

```
yosuke@asura0:~/Ti00/Ti00> cat ev.txt
16.200 -.16250331E+03
16.225 -.16247362E+03
16.250 -.16243967E+03
16.275 -.16240791E+03
16.300 -.16237489E+03
```

計算結果の視覚化

out によって計算結果を，last.txt，ev.txt にまとめた後，graph（コマンド）によって ev 曲線を gnuplot による視覚化を行う．なおこのコマンドは X ウィンドウシステムを起動時のみ有効．また常に最新の結果抽出を行った後，つまり out を実行した後に行うのが望ましい．以下に記述した例では，西谷研究室で MacBook を使用し，X11 ターミナルにてログインし，作業している．

```
Yamamotos:~ yosuke$ asura
xhost +
access control disabled, clients can connect from any host
Password:
```

```
Last login: Tue Feb  5 15:36:30 2008 from sstg41.ksc.kwansei.ac.jp
Have a lot of fun...
yosuke@asura0:~> cd Ti00/Ti00
yosuke@asura0:~/Ti00/Ti00> graph
```

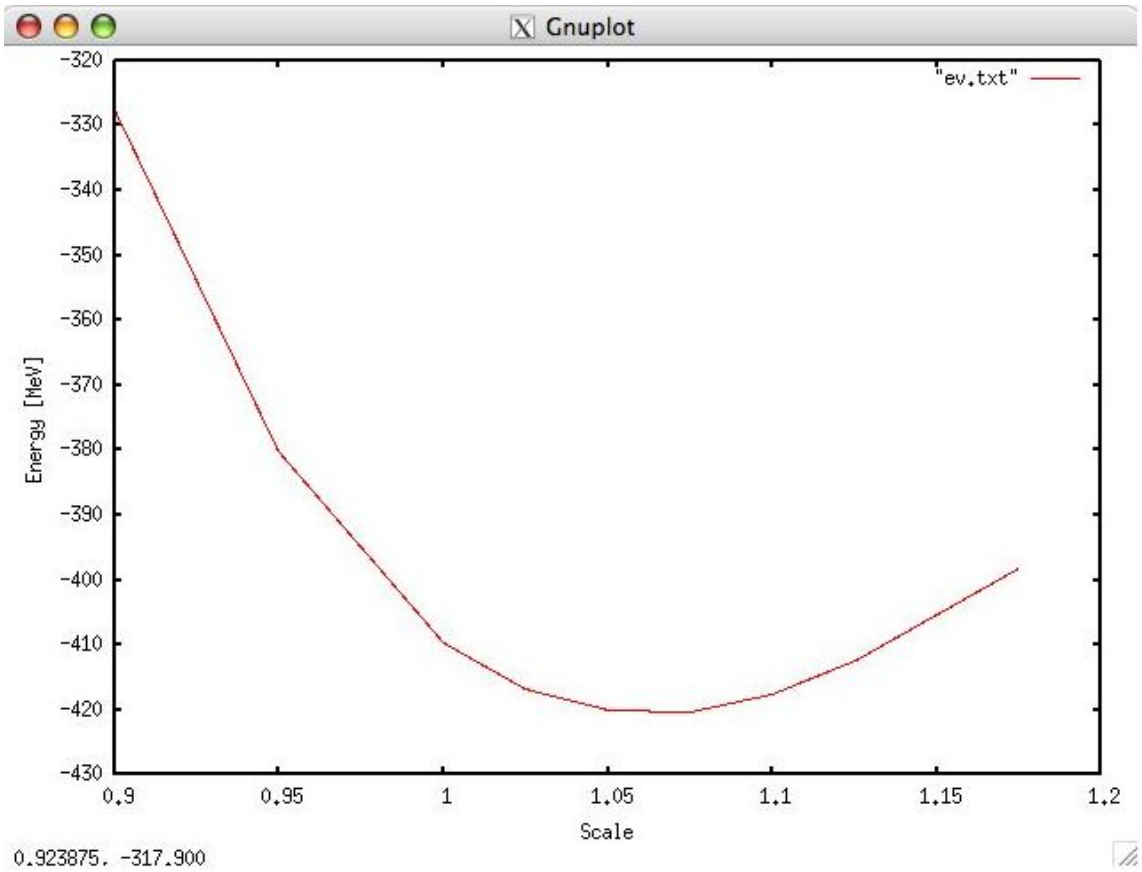

図 B.1: gnuplot による E-V 曲線の視覚化

B.1.5 再計算の実行

計算結果の確認後，reqsub（コマンド）によって再計算を行う．

```
yosuke@asura0:~/Ti00/Ti00> reqsub

Please input the value of a scale newly

16.225 16.35 16.40 16.300
```

```
How many CPU to use for 1 scale?
2
```

上記の作業の後，allqsub と同様に各スケールに対するディレクトリが作成される．また allqsub により既に作成されていたディレクトリは削除され，新たにディレクトリが作成され，計算を行う．また各々のスケールに対する計算のジョブ ID がファイル job_number.txt に追記される．

```
yosuke@asura0:~/Ti00/Ti00> ls
16.200.txt  DOSCAR*      PCDAT*         POSCAR.testfile*  Scale16.275/
scale.txt*
16.225.txt  DOSCAR_old*  PI15430*       POSCAR~*          Scale16.300/
scale.txt~*
16.250.txt  EIGENVAL*    POSCAR*        POTCAR*           WAVECAR*
submit_job.txt
16.275.txt  IBZKPT*      POSCAR.fore*   Scale16.35/       XDATCAR*
vasprun.xml*
16.300.txt  INCAR*       POSCAR.fore~*  Scale16.40/       ev.txt
CHG*        KPOINTS*     POSCAR.orig*   Scale16.200/      job_number.txt
CHGCAR*     OSZICAR*     POSCAR.orig~*  Scale16.225/      job_number.txt~
CONTCAR*    OUTCAR*      POSCAR.test*   Scale16.250/      last.txt

yosuke@asura0:~/Ti00/Ti00> cat job_number.txt
16.200   4716
16.225   4717
16.250   4718
16.275   4719
16.300   4720
16.225   4915
16.35   4916
16.40   4917
16.300   4918
```

また allqsub と同様に各スケールのディレクトリ内にもそのスケールの値に対するジョブ ID が記述されたファイルが作成される．

```
yosuke@asura0:~/Ti00/Ti00> cat Scale16.225/submit_job.txt
Your job 4915 ("vasp-job") has been submitted
```

B.2　コマンド説明

計算を行うにつれ非常に使用頻度の高いコマンドの紹介を行う．なお，ここで紹介するコマンドは付録 A.1.2 のエイリアスの設定に従事する．

B.2.1　計算発行

allqsub

コマンド入力後，1 スケールの対して使用する CPU の数を聞かれ，入力する．CPU 数を入力後，自動でスケールごとにディレクトリを作成し，各々のスケールの値に対し，計算を行うジョブを発行する．

reqsub

コマンド入力後，再計算を行うスケールの値，また使用する CPU 数を問われるので，各々入力する．スケールの値の入力の際，複数のスケールの入力が可能であり，その場合，各々のスケールとスケールの間にスペースを入力する．また新たなスケールの値を入力する事も可能である．CPU 数，スケールの値を入力後 allqsub と同様に自動でスケールごとにディレクトリを作成し，各々のスケールの値に対し，計算を行うジョブを発行する．

mkdir_qsub

allqsub，reqsub と同様にスケールごとにディレクトリを作成し，計算を走らせる．なお allqsub との違いは CPU の数を聞かれる事がないため，冗長な手間を省く事が可能．しかし，使用する CPU の数は計算を走らせる際の run.sh で設定されている CPU の数に従うので注意が必要である．なお run.sh で設定されている CPU の数は，最後に allqsub，または後述する reqsub によって入力された CPU の数と等価である．したがってまず allqsub によって CPU 数を 2 と入力し，次に mkdir_qsub と入力し，計算を走らせると，CPU の数は 2 となる．

mkdir_qsub_asura0

mkdir_qsub と機能は同じであるが，異なる点はジョブをすべて asura0 に発行するという点である．なお asura1，asura2 等の１つのマシンのみで計算を行う場合も同様にプログラムを作成する事が可能である．

qsub

主に計算ミスの際に，使用するコマンドである．たとえばスケールが 5 つの計算を実行し，1 つのスケールのみ以上な値．また計算ミスが検出された場合，そのスケールのディレクトリに入り，この qsub コマンドをする．以上の過程によりこのスケールの計算行うことができる．しかし，このコマンドにより計算を発行すると前回の計算結果を削除し，submit_job.txt の内容を再計算を行ったジョブ ID に手動で変更しなければ，後述する out コマンドにより新しい計算結果を反影する事ができない．従って以上の問題から再計算の際には reqsub を用いるのが望ましい．

B.2.2 結果確認

out

計算結果出力のコマンドであり，out を実行する事によって，last.txt，ev.txt に計算結果がまとめられる．また各々のスケールの計算結果も 0.9.txt（例，スケールが 0.9 の場合）等に出力される．なおこのコマンドは計算の途中でも実行が可能であり，計算の進み具合も確認できる．

graph

ev 曲線を gnuplot による視覚化を行う．なおこのコマンドはローカルエリアネットワーク内のみで実現される．Mac では X11 ターミナルでのみ，また Xeon ではリモートログインの際に ssh -X yosuke@asura0 とし，X ウィンドウシステムの起動から graph とする事で視覚化が可能である．Mac からターミナルでログインし，X ウィンドウシステムが起動されないまま graph と行った場合，また Xeon から ssh yosuke@asura0 とすると視覚化されないので注意．また計算終了後 out コマンドによって計算結果をまとめてから graph 行う必要がある．

B.2.3 SGE 関連

qstat -f

現在，発行されているジョブ等の確認を行う．

qacct -j 123

実行時間等の確認を行う．（ジョブ ID が 123 の場合）

qdel 123

一つのジョブ（ジョブ ID が 123 の場合）を削除.

qdel -u yosuke

ユーザ（発行者が yosuke の場合）のジョブすべてを削除.

qmon

図 B.2 のように SGE によるジョブの設定，制御等行う GUI 画面を開く．各アイコンは GUI ボタンで，クリックするとさまざまな作業を開始する事ができる．また各ボタンの名前はそのボタンの説明にもなっており，ボタンの上にマウスポインタを置くと，名前が表示される [9].

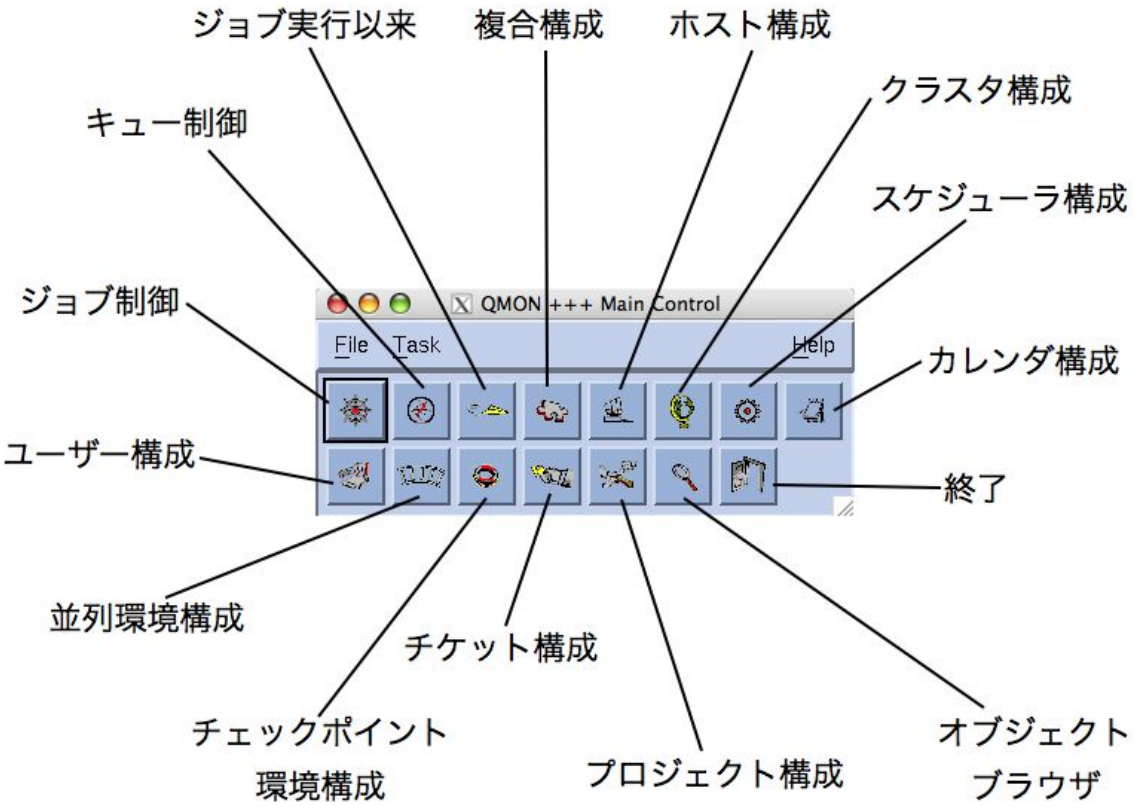

図 B.2: qmon による GUI

参考文献

[1] 大久保 宏樹　著　「VASP，GRID を高効率で運用する手法の開発」（関西学院大学　卒業研究　2006）

[2] 竹田 諒平　著　「第一原理計算ソフト VASP の高速実行環境の構築」（関西学院大学　卒業研究　2005）

[3] 大津 真，まえだ ひさこ　著　「SUSE Linux 10.0 ビギナーズバイブル」（株式会社毎日コミュニケーションズ　2006）

[4] 長原 宏治，倉沢 良一　著　「公式 SUSE LINUX 管理ガイド」（株式会社アスキー　2005）

[5] P. パチェコ　著　秋葉 博　訳　「MPI 並列プログラミング」（培風館　2001）

[6] 日本アイ・ビー・エム・システムズ・エンジニアリング株式会社　著　「グリッドコンピューティングとは何か」（ソフトバンク パブリッシング株式会社　2004）

[7] グリッド協議会　著　「グリッドトレーニングシリーズ　第１回：SGE と Ganglia のインストールと利用」（国立情報学研究所　2007）

[8] Cameron Newham，Bill Rosenblatt　著　QUIPU LLC / 遠藤 美代子　訳　「bash 入門」（株式会社アスキー 1997）

[9] PC クラスタ管理システム　Sun Grid Engine の概要
http://mikilab.doshisha.ac.jp/dia/research/report/2004/0817/001/report20040817001.html

[10] グリッド協議会　http://www.jpgrid.org/index.html

[11] VASP マニュアル　http://cms.mpi.univie.ac.at/vasp/

[12] Sun Grid Engine 機能詳細　http://www.softek.co.jp/Cluster/grid2.html

[13] MPI の基礎
http://mikilab.doshisha.ac.jp/dia/research/report/2005/0820/002/report20050820002.html

[14] インテル MPI ライブラリー 3.0 Linux 版
http://www.intel.com/cd/software/products/ijkk/jpn/329262.htm

[15] FB-DIMM がサーバのメモリを変える？
http://www.atmarkit.co.jp/fsys/kaisetsu/073fb-dimm/fb-dimm_01.html

[16] GNUPLOT http://t16web.lanl.gov/Kawano/gnuplot/