卒業論文

VASP, GRIDを高効率で運用する手法の開発

関西学院大学　理工学部　情報科学科

3687 大久保　宏樹

２００７年　3月

指導教員　西谷　滋人　教授

概要

VASP(Vienna Ab-intio Simulation Package)　は高速,高精度に固体物性が再現できることから広く使われている．しかし,１００原子程度の現実的な系を計算しようとすると,現在の西谷研究室の計算環境では2,3日程度の計算時間が必要となる．従って西谷研究室ではVASPの高速化は重要な問題である．現在の計算環境では，各ＣＰＵに対して最適な数値計算ライブラリー（LAPACK,BLAS,FFT）を使用している．よって，１CPUでVASPを実行した場合の実行速度を上げることは困難である．そのため，複数の計算資源へ処理を分散することにより実行時間を短くする並列処理は有用な高速化の手段である．また，ネットワーク上のコンピュータを接続し，複合したコンピュータを構築するグリッドコンピュータを並列環境と組み合わせて活用すると実行効率が向上する．そこで，本研究ではグリッド環境を構築すること，VASPを並列処理したときの実行速度とグリッド環境をもちいたVASPの実行速度とを比較しどのくらい性能が上がるかを測定することを目的とした．

グリッド環境の構築おいて，MPI(Message Passing Interface)，SGE(Sun Grid Engine)を用いる．MPIは，並列処理を実行するためのツールである．VASPはMPIを用いると並列計算することができるようになる．SGEはグリッド環境を構築するためのツールである．SGEによって複数のCPUで構成された環境に大量のジョブが投入されたとき，それらを自動的にスケジューリングし，各CPUにジョブを割り振って効率よく実行する．また，SGEは１つのプログラムを並列実行できないが，MPIと組み合わせて利用すると並列実行が可能になる．プログラムを実行するシェルスクリプトをSGEへ投入し，SGEが各CPUに割り振って実行するように環境を構築した．

VASPの実行速度の計測には，現在の西谷研究室で最も基本的な操作であるEV曲線を求めるプログラムを実行するシェルスクリプトを使用した．そして，グリッド環境の構築によってどのくらい実行速度が上がるかを計算量を変化させながら測定した．

結果，グリッド環境でVASPを実行したとき，MPIを用いて並列実行したときときより約1.9倍実行速度が速くなった．SGEの環境を用いてVASPを実行すると，MPIで並列実行するより効率よくCPUを使用することがわかる．そのため，今後グリッド環境を拡大化するとさらに実行速度は向上する．

目次

第1章　緒言

第一原理電子構造のプログラムの一つであるVASP(Vienna Ab-initio Simulation Package)　は高速，高精度に固体物性が再現できることから広く使われている．しかし,１００原子程度の現実的な系を計算しようとすると,現在の西谷研究室の計算環境では2,3日程度の計算時間が必要となる.従って西谷研究室ではVASPの高速化は重要な問題である.しかし，１CPUの処理能力の向上には限界があり，常に最新のCPUをPCに設置することは経済的に困難である．限られた計算資源の処理能力を前提として，実行時間を短くする方法を考えなければならない．そこで，複数の計算資源へ処理を分散することにより実行時間を短くする並列処理は有用な高速化の手段である．

並列処理を行う実現するソフトウェアとして第一にあげられるのはMPICHである．共有メモリを持たないLinuxクラスタ環境のような分散メモリ型では，メッセージの送受信により複数のCPUが協調して処理を行うというメッセージパッシングが一般的である．そして，メッセージパッシングの仕様としては，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が事実上の標準パッケージとして広く利用されている．本研究では，並列化するにあたりこのMPICHを使用する．また近年話題となっているグリッドコンピューティングとともに使用し，並列処理による実行時間の短縮を図る．

並列処理をより効率的に実行するためにグリッドコンピュータと呼ばれるものがある．グリッドコンピュータは，ネットワーク上にあ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である．提供されるサービスは主に計算処理やデータの保存，利用である．そのグリッドコンピュータのツールキットとして，サン・マイクロシステムズが開発したSGE(Sun Grid Engine)がある．　並列処理の効率向上には，スケジューリングやキューなどの機能が重要であり，SGEにはその機能が備わっている．

本研究では，SGE，MPICHの環境を用いて図1.1のように全てのプロセス，プログラムがSGEを経由して各マシンでVASPを実行するように並列環境を構築する.さらに，MPICHのみを用いて並列化するときとSGEを組み合わせて並列化するときで，どのくらい性能が上がるか求めることを目的とする.

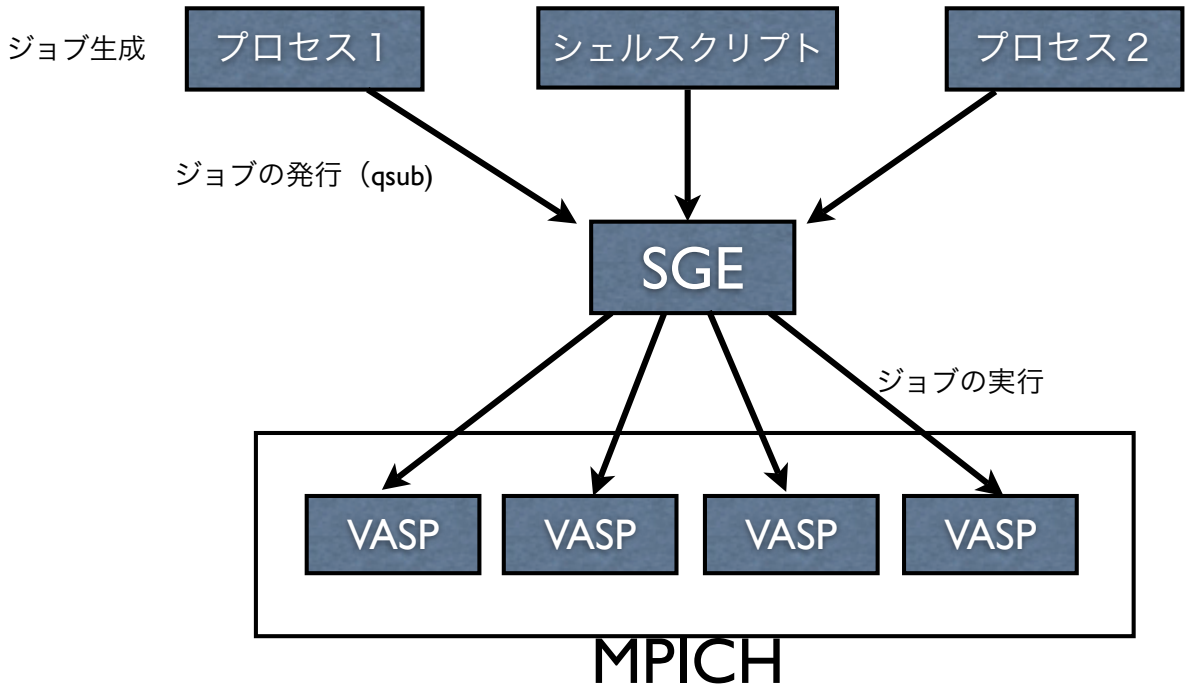


図1.1 ジョブ(プロセス)の生成から実行までの流れ

第2章　手法

VASPの実行効率の向上とその計測のために以下の手順をとった.

・多数の実行プログラムをより効率的に実行するために，SGEを用いた並列環境を構築する.

・EV曲線を求めるプログラムを実行するシェルスクリプトautocalcにおいて，MPICHとSGEの環境での実行速度を計測する．データをとるために，最も性能の良い1種類のCPUを並列化して実行する.

・さらにautocalcの計算量の変化によってどれくらい速度が変化するかを測定し，SGEとMPICHの並列環境で性能を比較する.

2.1　使用するCPU

VASPの計測をするため以下のCPUを使用した.

1．Pentium4 Northwood
OS：SuSE Linux9.3
キャッシュ容量：512KB
動作周波数：3.2GHz
フロントサイドバス：800MHz

2．Pentium4 Prescott
OS：SuSE Linux9.3
キャッシュ容量：2MB
動作周波数：3.4GHz
フロントサイドバス：800MHz

3．Xeon Woodcrest
OS：SuSE Linux9.3
キャッシュ容量：４MB
動作周波数：3.0GHz
フロントサイドバス：1333MHz

2.2　並列環境

VASPを実行する並列環境を構築するにあたり，MPICHを使用する並列環境とSGEを使用する並列環境の２つの方法を用いた．

2.2.1　MPICHを用いた並列環境

並列化するにあたり，まずMPICHを使用する．MPICH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する．図2.1に，シェルスクリプトからプロセスを送信し，MPICHを用いて各CPUで実行する様子を表す．プロセスを送信し実行する際には，MPICHのコマンドのmpirunを用いて実行する．

MPICHの環境ではシェルスクリプト内のコマンドmpirunで直接マシンに実行させる手法をとる．

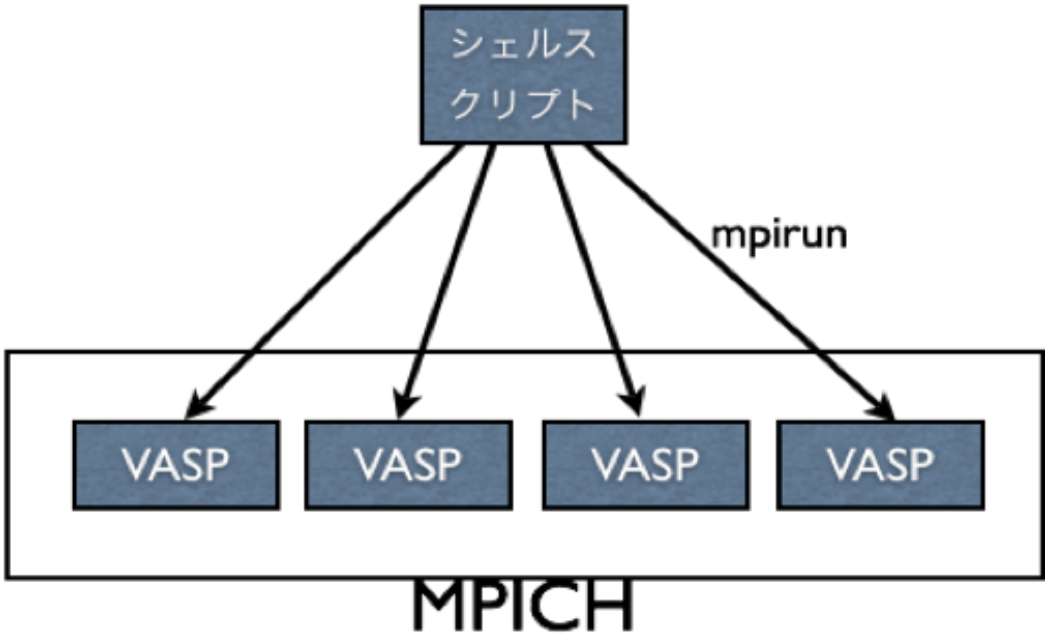


図 2.1　MPICHの並列環境

2.2.2　SGE(Sun Grid Engine)を用いた並列環境

次に，SGEを用いた環境を構築する．SGEは，ネットワークを介して複数のコンピュータを接続し，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．主にサービスとは，計算処理やデータの利用，保存などである．図2.2に，ジョブの生成から実行までを示す．図2.2でジョブとは，プログラムを実行するシェルスクリプトである．ジョブを実際に実行する際には，SGEのコマンドのqsubを用いてジョブをSGEへ発行し，SGEが自動的にジョブを各CPUに割り振って実行する．

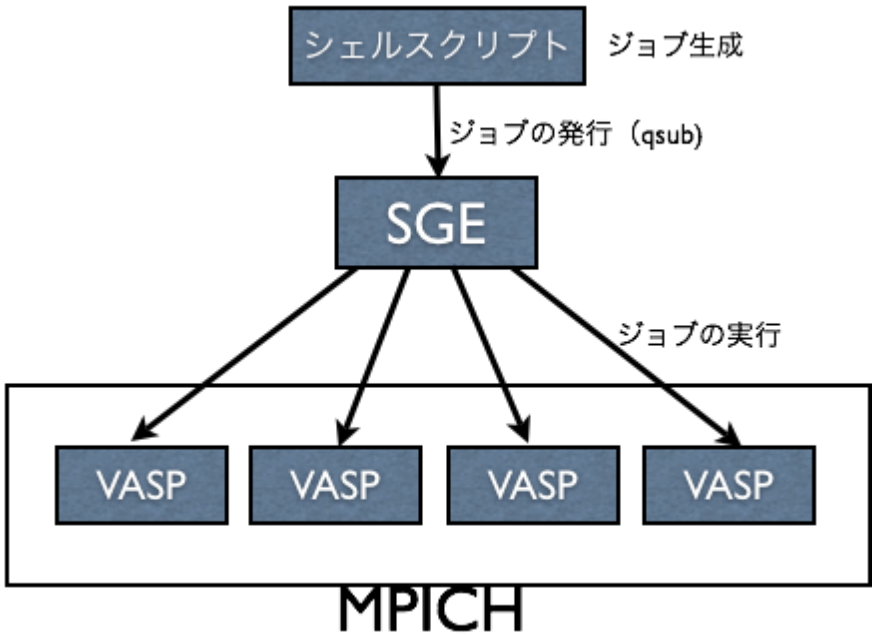


図 2.2　SGEの並列環境

SGEの主な機能を以下に記述する．

・スケジューリング

スケジューリングとは，仕事を効率的に行うためのシステムである．あるプロセスやジョブがSGEへ投入され，キューに載せられたとき，自動的にCPUにジョブを割り振って一番速く実行できるCPUへジョブを投入する．また，大量のジョブが投入されたときでも，キューへジョブを入れて待機させ，FIFO(first in first out)方式で順次CPUへ割り振って実行する． 例えば，スーパーマーケットなどのレジで処理の速

い人のレジと遅い人のレジがある．はたしてどのレジに行けば一番速く購入できるかという問題がある．その問題を瞬時に解決し，購入者をそのレジへ導くことを自動的に処理するのがスケジューリングである

・CPUの管理
　定間隔でCPUを管理し，CPU使用率やCPUの障害など瞬時に判断する．そのため使用率の低いCPUから順にジョブを投入することができる．

・並列ジョブの処理
　MPICHと併用して活用すれば 1 つのジョブを並列実行できる．

・自動ジョブ再実行
　何らかの障害が出たCPUを発見したとき，途中まで動いていたジョブを再スケジューリングして他のCPUで再実行する．SGEは決められた時間間隔でマシンの状況をチェックしているので，その時間間隔内でジョブの再実行が行われる．

第3章　SGEの環境構築

SGEの環境を構築するために以下の設定を順次行う．

1．各マシンのIPアドレス設定
2．NIS，NFSサーバクライアントの設定
3．ネットワークサービスの設定
4．SGEのPATHを通す
5．各ホストのSGEのインストール
6．キューの設定
7．並列ジョブの環境設定

3.1　IPアドレスの設定

個々のマシンにIPアドレスを割り当てるとき，DHCPで割り当てず確実に空いているIPアドレスに手動で設定する．SuSEでは「YaST」の「ネットワークデバイス」の「ネットワークカード」で設定する．

takeda3.nishitani.com	192.168.1.29
takeda4	192.168.1.28

3.2　NIS,NFSサーバの設定

NIS (Network Information Service)は，ネットワーク上のすべての計算機で必要な情報を共有するサービスのことである。NISサーバーには，マスターサーバーとスレーブサーバーがある．マスターサーバーはNIS上のデータ，情報を管理し分配する．スレーブサーバーはデータ，情報を分配される．ひとたびマスターサーバーが情報を分配すればNIS上のスレーブサーバーが同じ情報を持つことができる．そこで，並列環境に必要なNIS，NFSサーバの設定を行う．

3.2.1　マスターサーバーの設定

・マスターサーバーの作成
　「YaST」の「ネットワークサービス」の「NISサーバ」を開いてマスターサーバーを作成する．「ＮＩＳ　Ｄｏｍａｉｎ　Ｎａｍｅ」にｔａｋｅ-daを入力し，他はデフォルトのまま設定する．

・サーバー上のマシンのIPアドレス割りあて
/etc/hostsに
takeda3.nishitani.com　192.168.1.29
takeda4　　192.168.1.28　　を追加する．

3.2.2　スレーブサーバーの設定

・スレーブサーバーの作成
「YaST」の「ネットワークサービス」の「NISクライアント」を開いてスレーブサーバーを作成する．「NIS Domain」にtakedaを入力する．

・サーバー上のマシンのIPアドレス割り当て
/etc/hostsに
takeda3.nishitani.com　192.168.1.29
takeda4　　192.168.1.28　　を追加する．

3.2.3 NFSサーバの設定

　NFS(Network File System)サーバはファイルを共有するサーバである．並列処理を行う場合，全てのマシンで同じ場所に同じ実行ファイルがなければならない．ファイルを変更する場合にマスターサーバーだけ変更すれば他のマシンも変更されるので便利である．

・ファイルの共有
　マスターサーバーの /usr/home を全てのマシンの /home　に追加する．
　ユーザーは，/usr/home でファイルを保存したり実行する．

3.3　ネットワークサービスの設定

グリッドエンジンシステムのサービスにはsge_execdとsge_qmasterがあります.
これをTCPに登録するには開いているポート番号を選択する.

/etc/services に sge_qmaster　　536/tcp
　　sge_execd　　537/tcp　　を追加する.

3.4 SGEのPATHを通す

以下のコマンドでPATHを通す．PATHとは，その場所への道しるべのことである．全てのコマンドにはPATHがあり，SGEで使用するコマンドは/usr/sgeにあるということを設定する.

> SGE_ROOT=/usr/sge;
> export SGE_ROOT

3.5　各ホストのSGEインストール

SGEには，マスターホストと実行ホストがあります．マスターホストは，クラスタの中心となるホストであり，マスターデーモン（sge_qmaster）とスケジューラデーモン（sge_schedd）を実行する．この二つのデーモンはともに，キューやジョブなどのSGEコンポーネントのすべてを制御し，コンポーネントの状態やユーザーのアクセス権限などを管理する機能を持つ．実行ホストは，SGEのジョブを実行する権限を持つノードである．このため，実行ホストはSGEのキューのホストの役割を果たし，SGE実行デーモン（sge_execd）を実行する.

・マスターホストのインストール
マスターサーバーでマスターホストをインストールする．（スレーブサーバーでは不要）
1．http://gridengine.sunsource.net/project/gridengine/23o701rb60.htmlから圧縮ファイルをダウンロードし，/usr/sgeへ解凍する.
2．/usr/sgeで　./install_qmaster　でインストールする．（付録B参照）

・実行ホストのインストール
スレーブサーバーで実行ホストをインストールする.

1. http://gridengine.sunsource.net/project/gridengine/23o701rb60.htmlから圧縮ファイルをダウンロードし，/usr/sgeへ解凍する．
2. /usr/sgeで　./install_execd　でインストールする．（付録B参照）

3.6　キューの設定

キューに載せられたジョブを実行するホストを設定する．

1. SGEの各設定を行うアプリケーションを開くコマンドであるqmonで設定画面を開く．
2. 「Queue control」でdefaultを追加し，「Hostlist」に@all.qを追加する．「for Host/Host group」に@/を追加する．
3. 「User Configuration」の「Manager」「Userset」「Users」にroot,takedaを追加する．
4. 「Host Configraiton」の「ExecutionHost」の各ホストの「UserAccess」にUsersを追加する．さらに，UsersをAccess Alowに追加する．

3.7　並列ジョブの環境設定

MPICHを用いて 1 つのジョブを並列実行するための設定を行う．

1. qmonで設定画面を開く．
2. 「Parallel Enviroment」で「PE」にMPICHを追加する．
3. 「Modify」で，slots=2，「UserList」にUsers，「start proc」に/usr/sge/mpi/startmpi.sh $pe_hostfile，「stop proc」に/usr/sge/mpi/stopmpi.shを追加する．
4. 「Queue Control」の「Modify」「Parallel Enviroment」にmpichを追加する．

第4章　CPUの性能

VASPを実行計測するにあたり，Pentium4 Northwood, Pentium4 Prescott, Xeon Woodcrest の３つのCPUを使用した． MPICHとSGEの環境で実行測定するにあたり，使用するCPUを決めるの比較検討を行った．また，いくつCPUを用いるかを決める．

4.1　使用する数値計算ライブラリー

VASPを実行計測するにあたり，Pentium4 Northwood, Pentium4 Prescott, Xeon Woodcrest の３つのCPUを使用した．VASPの基本となる数値計算は，連立方程式，固有値，FFTから構成される．そして，実行速度はこれらを解くライブラリーに大きく依存する．今回，その数値計算ライブラリーをそれぞれのCPUで最適なものを使用した．

VASPに使用される数値計算ライブラリーには，BLAS(Basis Linear Algebraic Subprograms), LAPACK(Linear Algebra PACKage)がある．LAPACKは，線形代数の問題を解くためのライブラリーである．連立方程式，固有値問題，線形の最小二乗問題，特異値問題について解くことができ，関連して行列の分解などの多くの計算を行うことができる．LAPACKは下部ルーチンをBLASに任せることにより，高い移植性を保っており幅広いコンピュータで使うことができるようになっている．

BLASは行列およびベクトルの基本的な演算を行うライブラリーで，LAPACKへ線形演算レベルでの標準関数を提供する．したがって，LAPACKで用いられるBLASの関数仕様は標準化されているが，通常はアセンブラで書かれているため移植性は高くないが，高速に線形代数演算が行われるようになっている．

4.1.1　各CPUに対する数値計算ライブラリー

1．Pentium4 Northwood
BLAS：libgoto
LAPACK：lapack_double

2．Pentium4 Prescott
BLAS：lmkl
LAPACK：lmkl_lapack64

3. Xeon Woodcrest
BLAS：MKL8.1.1
LAPACK：MKL8.1.1

4.2 並列時のVASPの性能

CPUがPentium4 Northwood搭載のコンピュータを使用してノードを増やしてVASPを実行する．実行速度の計測には，VASPに付属する標準的なベンチマークbench.Hgを使用する．

実行結果は表４.1のようになった．CPU数2でCPU数1のときより約1.4倍，CPU数4のときで約２．５倍実行速度が向上した．理想としては，CPU数を増やすと実行時間は１／CPU数になるが，VASPは複雑なコードのため実行時間はそのようにはならない．そのうえ，CPU数を8に増やしたとき，CPU数が4のときより性能が落ちる．

次に，CPUがXeon Woodcrest搭載のコンピュータを使用してノードを増やしてVASPを実行する．CPU数2でCPU数1のときより約1.7倍あがり，CPU数４のときで約２．2倍実行速度が向上する．さらに，NorthwoodがCPU数4のときとXeonのCPU数1のときで同じ実行速度だった．また，XeonでもNorthwoodと同様に，CPU数を8に増やしたとき，CPU数が4のときより性能がほとんど上がらなかった．

これはMPICHがプロセスのメッセージを送受信するために時間を費やしているためであることが予想できる．並列処理では，実行するCPU数が増えれば増えるほどプロセスの送信に時間がかかり，それぞれのCPUで実行するのが遅れてしまう．そのため，送受信の時間に対して実行時間が短くなり，ノードを増やすと性能が落ちてしまう．したがって，MPICHはメッセージ送受信の時間よりプロセスの実行時間の方がはるかに長いときに性能を発揮できると考えることができる．実際に，bench.Hgより複雑なプログラム(autocalc)をXeonでCPU数が1,４,８のときで実行した．(表 ４.2)

その結果，4CPUで2.25倍，8CPUのとき3.8倍に実行速度が向上した．実際の西谷研究室でのVASPの計算はさらに時間がかかるものが多いので，実行速度の倍率はさらに向上すると考える．次の章から最も性能が良かったXeon WoodcrestのCPU8個を用いてautocalcの実行測定をする．

CPU\ノード	1	2	3	4	8
	time(s)				
Northwood	210	154	107	84	102
Prescott	201	137	-	-	-
Xeon Woodcrest	84	50	44	38.7	38.5

表4.1　bench.Hgの実行時間

CPU数	1	4	8
time(s)	10020	4440	2640

表4.2 Xeonでのautocalcの実行時間

第5章　MPICHによる並列時のVASPの性能

5.1　MPICHで用いるautocalc

EV曲線（E-energy V-volume)の極小値を求めるプログラムを実行するシェルスクリプト（autocalc）を用いて実行速度を計測する．EV曲線を求めるのは，現在の西谷研究室で最も基本的な操作である．autocalcの性能を上げることが西谷研究室において非常に重要である．まず，昨年まで使用していたMPICHの環境に対するautocalcを用いて検証する．図5.1に簡潔なautocalcを示す.

```
#!/bin/tcsh -f
i=1
while   i<n
      j=1
      while  j<m
            mpirun  -np 8 ./vasp  #8CPUでVASP実行
            j++
      end
      i++
end
```

図5.1　MPICHの環境に対するautocalc

MPICHの環境に対して使用するautocalcは，入力ファイルの変数を変えながら繰り返し実行する．この処理を，逐次処理，パイプライン処理，パラレル処理に分類するならば，逐次処理に属する．なぜなら，ループ内でMPICHのコマンドであるmpirunを用いてVASPを実行し，VASPの実行がすべて終了してから逐次的に次のループに入り処理を始めるからである．VASPの実行は8CPUで行う．

5.2　MPICHによるautocalcの実行の様子

MPICHによる並列実行は，4章でも述べたようにCPU数が増えれば増えるほどメッセージの送受信に時間を費やしてしまう．図5.1のループ１回分の実行の様子を図5.2に示す．

図5.2はマシンが４つの場合で，左の全仕事がループ１回分の計算量を表している．四角は各マシンで実行中であることを示す．この全仕事の大半は４つのマシンで実行しているが，１つのマシンで実行する時間もある．そのとき，他の３つのマシンが動いていないため効率が悪くなる．さらに，メッセージの送受信やループとループの合間のときなど，どのマシンも動いていない時間も生じる．１番効率が良いのは全てのマシンが常に実行していることであるが現実的には困難である．

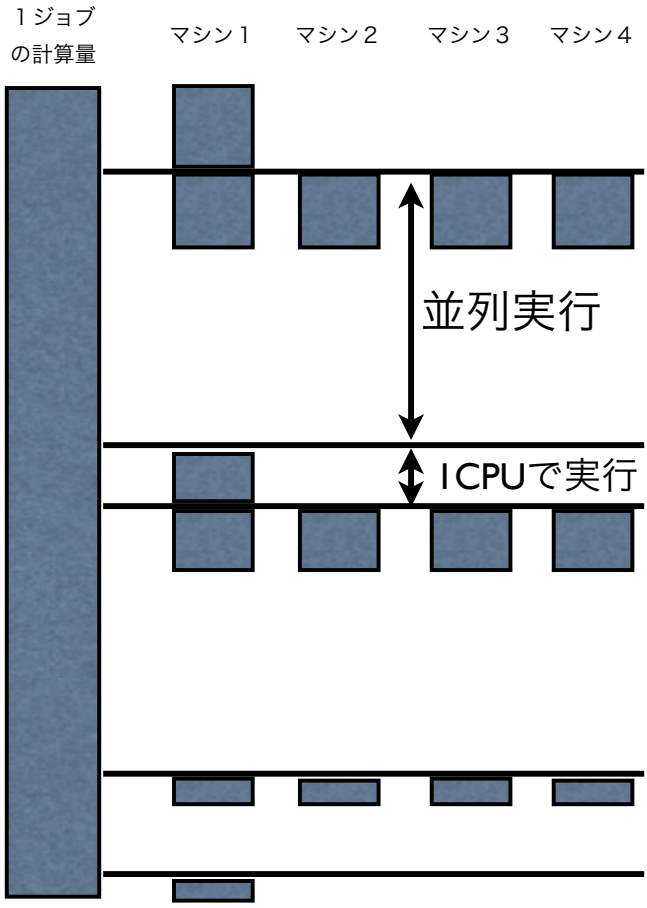


図5.2　MPICHの実行の様子

5.3　autocalcの実行速度

autocalcを実行するにあたり，Xeon WoodcrestのCPUを8個並列化して実行した．また，autocalc内のループ数を変化させて実行時間の測定を行った．ループしないときの実行時間が120秒なので，（ループ数）×120秒が実行時間になるのが普通だが，autocalcは入力ファイルの変数を複雑に変化させて実行するのでそうはならない．結果を図5.3に示す.

	ループ数	time(second)
MPICH	1	120
	2	840
	4	2160
	8	2640
	12	5220
	16	9120
	18	12660

表5.1　MPICHによるautocalcの実行時間

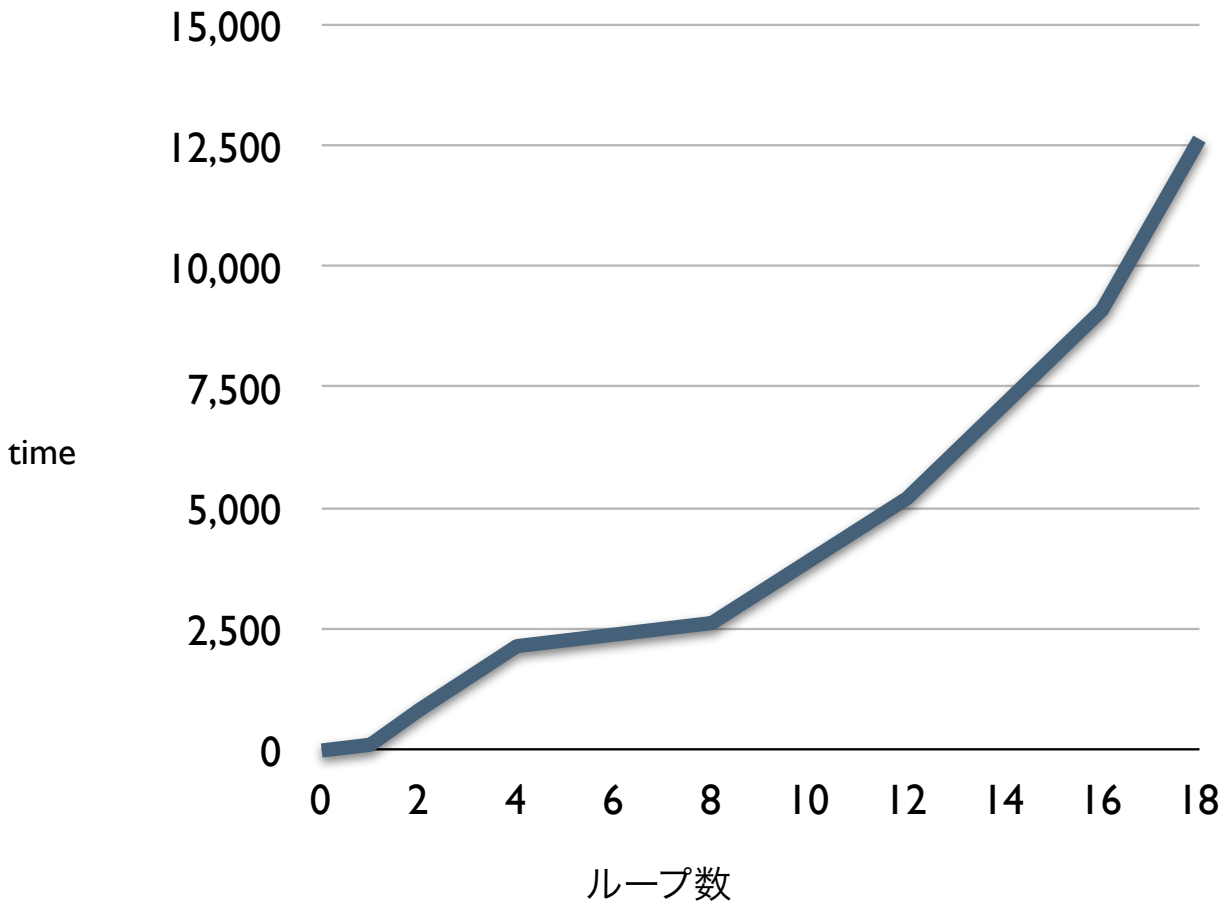


図5.3　autocalc実行時のループ数と時間の関係

5.4　まとめ

1回目のループは計算量が少ないためループ数が2回以上のときから実行時間が長くなった．また，何らかの障害であるCPUが動作しなくなれば，全てのCPUでの実行を中断し，autocalcも中断してしまう．その場合，計算を再会させるにはループする入力ファイルの変数の初期値を中断した値に変え，出力ファイルも新しく作る必要があり，非常に手間がかかる．

第6章　SGEによる並列時のVASPの性能

6.1　SGEで用いるautocalc

SGEを用いた並列環境に対するautocalcは，VASPの実行スクリプトをSGEへ投入するシェルスクリプトである．ループの中でSGEのコマンドであるqsubを用いて，VASPを実行するシェルスクリプトをSGEへジョブとして投入する．このシェルスクリプトはこれを繰り返し処理するだけであり，VASPの実行はSGEが各CPUに割り振って実行する．図6.1に簡潔なautocalcを示す.

```
#!/bin/tcsh -f
i=1
while   i<n
        j=1
        while  j<m
                qsub  {vasp実行のシェルスクリプト}  #ジョブの投入
        j++
        end
        i++
end
```

図6.1　SGEを用いた並列環境に対するautocalc

6.2　SGEによるautocalcの実行の様子

autocalcによってSGEへ多数のジョブが投入された後，そのジョブを全てSGEのキューに貯める．SGEは貯めたキューをFIFO(first in first out)方式で空いているCPUに割り振って実行する．これらは全てSGEが自動的に行います．その様子を図6.2に示す.

図6.2のキューの中で区切られている四角はそれぞれ図6.1のループ１回分の計算量である．また，ループ１回分のジョブを１CPUで実行します．全てのCPUは，実行中のジョブが終了すると，キューに貯められていたジョブが送られてきてすぐに実行を開始するので稼働していない時間はほとんどない．よって，非常に効率的である．

他の方法として，一つのジョブをMPICHと組み合わせて複数のCPUで実行することも可能である．この場合，ループ回数がCPU数を超えると効率が悪くなることが予想できる．

図6.2においてループが４回のときを考える．MPICHを用いて一つのジョブを２CPUで実行する場合，２つのジョブが２CPUずつ使って実行する．残り２つのジョブは待機し，実行中のジョブが終了しCPUが空いてから実行される．MPICHを用いることで図5.2のようにジョブの実行中にマシンが稼働しない時間が増えてしまう．それに対して１ジョブを１CPUで実行する場合，キューに４つのジョブが入り各CPUに割り振られる．その後，全てのCPUが常に稼働している状態が続き，どれか１つのジョブが終了するまで全て稼働しているので効率が良い．ただし，それは４つのジョブが同じ程度の計算量の場合である．一つのジョブだけ計算量が大きいと１CPUだけが稼働する時間が長くなり効率が悪くなる．かといって最後のジョブだけ４CPUで実行するというのは不可能である．したがって，MPICHを利用した方が効率的である場合もあると予想できる．実際に，autocalcで一つのジョブだけ計算量を大きくして実行した．計算量の大きさは約1.3倍のときと約1.5倍のときで実行した．結果，表6.1のように１つだけ約1.5倍のジョブを実行したときSGEの方が時間がかかった．

しかし，autocalcの計算は比較的複雑ではあるが，計算量は1.5倍も違わないのがほとんどである．また，１CPUで実行されるのは最後のジョブだけなのでループ回数が増えると1CPUだけが稼働している時間の割合は短くなる．よって，１つのジョブを１つのマシンで実行する方法を用いるのが懸命である．

n＼環境	MPICH	SGE
約1.3倍のジョブ	512	354
約1.5倍のジョブ	557	606

表6.1　１つのジョブだけ約n倍の計算量のときのautocalcの実行時間

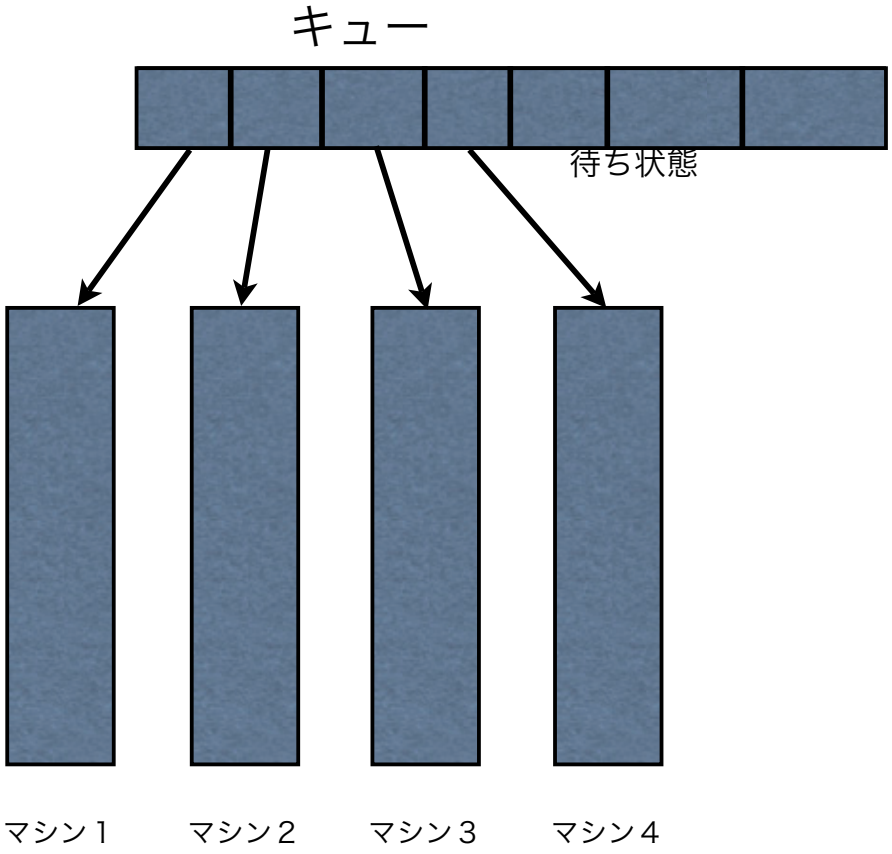


図6.2 SGEによるVASPの実行の様子

6.3　autocalcの実行時間

１つジョブに対してCPU数１で実行した．ただし，８個のCPUを用いているため，ループ数が４以下のものはCPUが余るためMPICHと組み合わせて実行した．ループ数１，２，４の場合でそれぞれ１ジョブをCPU数８，４，２で並列実行した．結果は表6.2の通りである．また，MPICHの実行結果と比較したグラフを図6.3に示す．

	ループ数	time(second)
SGE	1	120
	2	600
	4	960
	8	1500
	12	1860
	16	5220
	18	7200

表6.2　SGEの環境に対するautocalcの実行速度

time(s)		ループ回数						
		1	2	4	8	12	16	18
並列環境	MPICH	120	840	2160	2640	5220	9120	12660
	SGE	120	600	960	1500	1860	5220	7200

表6.3　MPICHとSGEのautocalcの実行時間

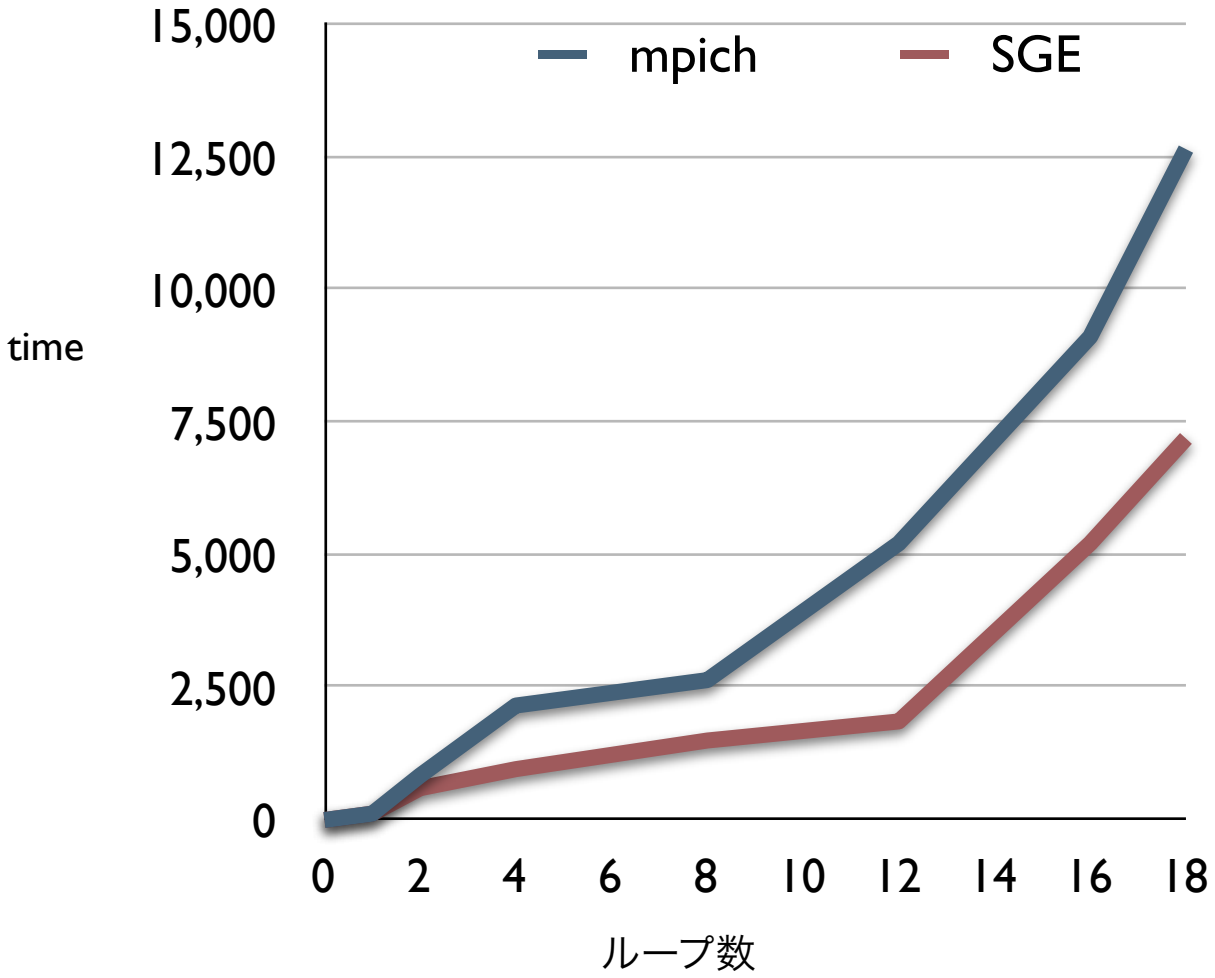


図6.3　autocalc実行時のループ数と時間の関係

6.4　まとめ

図6.3のように，一般的に用いるautocalcを実行したときSGEを用いた環境の方がMPICHを用いた並列環境より実行時間は短くなる．また，表6.1の結果ように一つのジョブだけが非常に計算量が大きい場合，MPICHの方が実行時間が短かった．一般的に使用するautocalcでは一つのジョブだけが極端に大きいということはほとんどないので，これはごく稀なケースである．

さらに，実行中あるCPUに何らかの障害が起きて稼働しなくなった場合，そのCPUで実行中のジョブだけが中断する．その中断したジョブはSGEが自動的に再実行するので問題はない．

第7章　まとめ

7.1　MPICHとSGEによる環境の性能

計算量をautocalcのループ回数で変えながらそれぞれの環境で実行時間を測定した．その結果，全てのループ回数でSGEによる環境での実行時間がMPICHを用いた実行速度より速くなる．

・実行速度向上率
- ループ2回　1.4倍
- ループ4回　2.1倍
- ループ8回　1.8倍
- ループ12回　2.8倍
- ループ16回　　1.7倍
- ループ18回　1.8倍

平均すると1.9倍速くなる．西谷研究室では，autocalcのループ回数が605回のものなどが多くあるので，実行時間が1週間かかっていたものが3,4日で実行できるようになる．

7.2　SGEの環境構築

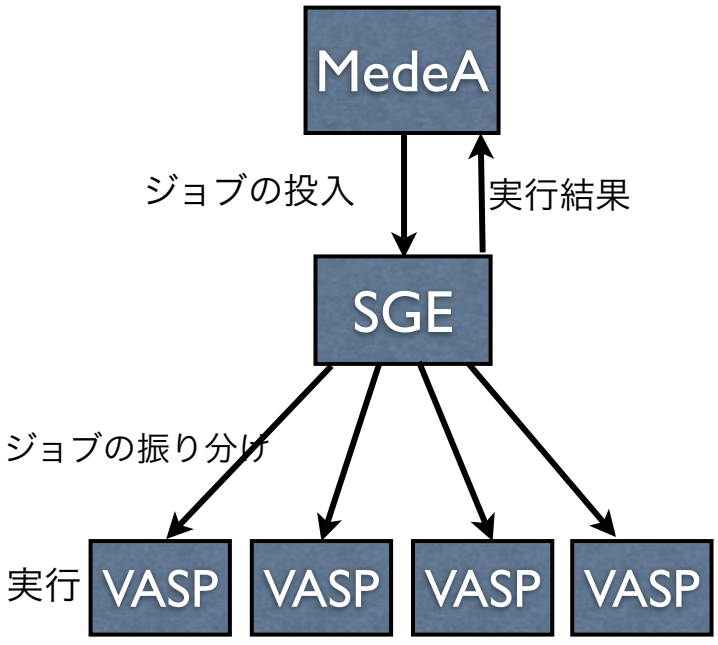
本研究結果より，SGEを用いた並列環境はMPICHを用いた並列環境より実行効率が向上する．さらに，グリッドを拡大すれば図6.3の結果より実行速度が向上することが予想できる．ただし，ジョブを発行する数が少数で計算量が大幅に違う場合，一つのジョブを複数のCPUで実行した方が実行速度は向上する．

今後，SGEの環境で様々な処理を行うためにautocalcだけでなく他の実行プロセスをSGEで実行できるように設定すれば，SGEにジョブを投入するだけで結果が出るようになり，VASPの実行効率も向上する．

付録A　MedeAに対するSunGridEngineの環境設定

・MedeA

MedeAは，VASPの入力ファイルをGUIを用いて容易に作成できるソフトである．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システム上で稼働する．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．しかし，一つのジョブを 1 CPU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，などの問題が生じる．そこで，SGEを用いてMedeAからSGEへ実行プログラムを投入し，各マシンに振り分けて実行する．その計算結果をMedeAへ返す環境を設定する．その環境でのジョブの様子を図A.1に示す.



図A.1　MedeAからマシン

・MedeAに対するSGEの環境設定

現在，MedeAは各マシンへ直接ジョブを送信する手法をとっている．そのため，SGEのマスターホストへ実行ファイルを転送することは可能である．その実行ファイルを手動でSGEへジョブとして投入することも可能である．そこで，図A.1のように環境を設定するにはマスターホストへのファイル転送からジョブの投入までを自動化すればよい．その設定を以下のように順次行った．

1．SGEの環境構築
2．PBS.tclの編集
3．SGEのファイルアクセス権

1．SGEの環境構築

第３章の設定を順次行う．環境の設定ができれば，MedeAのTaskcontrolのtakeda3の画面で「Queue type」をPBSに編集する．その後，MedeAで適当な結晶モデルの計算を実行するとエラーが出てtakeda3にファイルが転送されたままになる．takeda3でそのファイルを手動でSGEへ投入して実行する．実行できれば，MedeAで作ったファイルがSGEで実行できるということである．実行できなければSGEの環境を再度設定する．

2．PBS.tclの編集

PBS.tclは，MedeAがマスターホストでSGEを用いてジョブを投入するときにSGEの環境を読み取るフィイルである．MedeAはPBS.cslを読み込んでシェルスクリプトを作成する．PBS.tclを以下のように編集する．

/usr/home/takeda/MD/2.0/TaskServer/Tools/vasp4.6.25/PBS.tcl の140行目の
set qsub /opt/gridengine/bin/glinux/qsub ;# Rock cluster　を
set qsub /usr/sge/bin/lx24-amd64/qsub ;# Rocks cluster　に編集する．

編集後，MedeAで適当な結晶モデルの計算を実行すると，「error: Please set the environment variable SGE_ROOT.」と表示される．

3. SGEのファイルアクセス権

MedeAから転送されたファイルやPBS.tclの所有権はtakedaユーザーであるので，SGEでジョブを投入できるユーザーを追加する．以下の手順を行う．

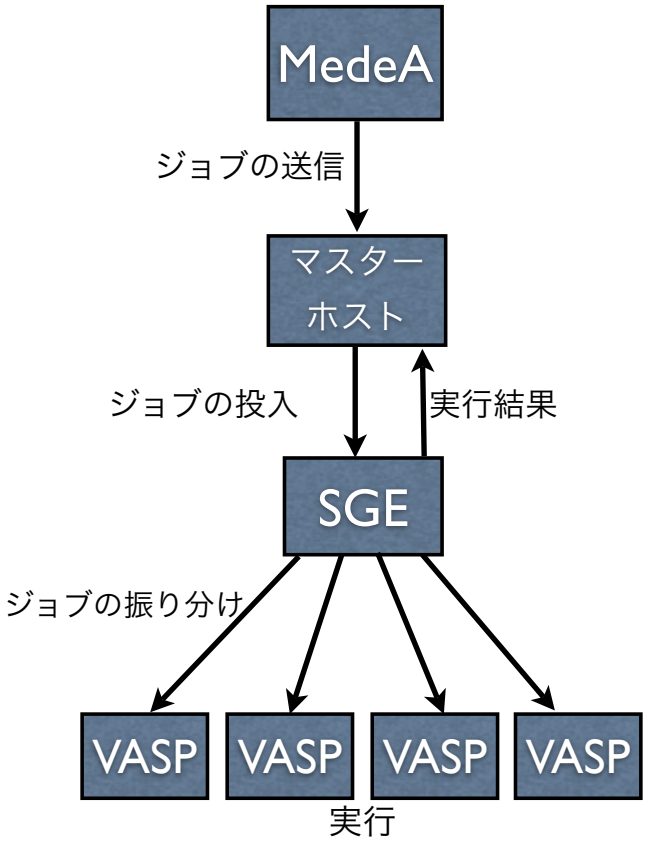
1．qmonコマンドで設定画面を開く．

2．「User Configuration」の「Manager」「Operator」にtakedaを追加する．

編集後，MedeAで実行したが表示内容は変わらなかった．

4．結果

以上の設定を行った結果，図A.2のようにMedeAからSGEのマスターホストへジョブを転送し，マスターホストで手動でそのジョブをSGEへ投入する環境となった．



図A.2　MedeAからのジョブの送信から実行結果までの流れ

付録B　SUN GRID ENIGINEマニュアル

・マスターホストのインストール

./install_qmaster　でインストールする．インストールガイドはhttp://192.18.109.11/817-7678/817-7678.pdfに掲載している．その設定内容を以下に示す.

・管理アカウントをもつユーザーを選択する画面でrootを記入する.
・実行ホストとしてインストールするホストにtakeda3,takeda4を記入する.
・その他の設定はデフォルトのまま「enter」を入力する.

・実行ホストのインストール

./install_execd　でインストールする．全てデフォルトの設定のまま「enter」を入力して完了する.

・デーモンの起動

各ホストでインストール後，デーモンを起動するために以下のコマンドを入力する.

source /usr/sge/default//common/setting.sh

・管理ホストの登録

実行ホストになるために管理ホストに登録する必要がある.

qconf -as ホスト名　で登録する.

・インストールの確認

ps -ax |grep sge

のコマンドを入力して、sge_qmaster, sge_execd, sge_schedd　があればインストールの完了である．（実行ホストではsge_scheddは不要）

・ジョブ発行の確認

/usr/sge/example/jobs/ にあるサンプルの実行ファイルを選択し，以下のコマンドを入力する．

```
# qsub sample.sh
```

これをCPU数以上入力して大量に発行し，実行状況を見るために以下のコマンドを入力する．

```
# qstat -f
```

全てのホストで実行するかどうかを見る．全てのホストで実行していればインストール作業は終了である．

付録C autocalc

SGEの環境へジョブを発行するautocalc (シェルスクリプト)

・autocalc

```
#!/bin/tcsh -f
set Values=(\
"48.90600000"\
"54.05400000")           //Valuesの配列の数によってループの数を変化させる
rm -r res*
set i_final=$argv[2]
@ i_final += 1
set j=$argv[3]
set j_final=$argv[4]
@ j_final += 1
while (${j}<${j_final})
   sed "s/ytrans/${j}/g" test.maple > orig/test.maple2
   set i=$argv[1]
   while (${i}<${i_final})
         sed "s/xtrans/${i}/g" orig/test.maple2 > orig/test.maple3
         /usr/maple9.5/bin/maple < orig/test.maple3
         cat POSCAR.fore > orig/POSCAR.orig
         cat result >> orig/POSCAR.orig
         cat result >> orig/auto.POSCAR
         mkdir res${i}_${j}          //x,y座標軸へずらす度にディレクトリを作成する
         cp -r orig/* res${i}_${j}   //作成したディレクトリに必要な入力ファイルをコピー
         cd res${i}_${j}             //ディレクトリに移動する
               @ index=1
         while (${index} < ${#Values} + 1)
               mkdir res${index}           //各点の実行を行うディレクトリを作成する
               cp -r ../orig/* res${index} //必要な入力ファイルをコピーする
               cd res${index}              //ディレクトリへ移動する
               echo "START calc" > res.all
               echo "POSITION" > auto.POSCAR
               echo "#TRIAL" ${i} ${j} >> res.all
               echo "#TRIAL" ${i} ${j} >> auto.POSCAR
                     sed "s/#SCALE/$Values[$index]/g" POSCAR.orig > POSCAR
               qsub vasprun.sh       //SGEへVASP実行のジョブを発行する.
               echo "#BOB SCALE =" $Values[$index] >> res.all
               cd ..
               @ index++
               end
         cd ..
         @ i++
   end
    @ j++
end
```

・ディレクトリーorig内のファイル（ディレクトリーorigに必要な入力ファイルや実行ファイルを入れる）

INCAR
POSCAR
POSCAR.orig
auto.POSCAR
POSCAR.fore
KPOINTS
POTCAR
result
test.maple
test.maple2
test.maple3
vasprun.sh

・vasprun.sh（VASPを実行して結果をファイルへ出力するシェルスクリプト）

```
#!/bin/tcsh -f
#$ -cwd

/usr/local/mpich/bin/mpirun -np 1 /usr/local/vasp/vasp.4.6/vasp >> res.all
echo "#END " >> res.all
grep -B 7 "#END" res.all > res.all2
grep "F=" res.all2 > res.all3
awk '{print $3}' res.all3 > res.all4
tail -10 res.all4 >> res.energy
```

・energyall.sh（各フォルダのres.energyの結果を収集するシェルスクリプト）

```
#!/bin/tcsh -f
rm res.energy
@ i=1
while (${i} < 3)
  @ j=1
  while (${j} < 3)
    cd res${i}${j}
    rm res.energy
    echo "#TRIAL" ${i} ${j} >> res.energy
    @ index=1
    while (${index} < 3)
      tail -5 res${index}/res.energy >> res.energy
      @ index++
    end
    tail -10 res.energy >> ../res.energy
    @ j++
    cd ..
  end
  @ i++
end
```

・コマンド

./autocalc 1 2 1 2　（1 2 1 2の値を変えてループの数を変化させる）

全ての実行終了後

./energyall.sh

尚，第5,6章でのループ数の変化はValuesとコマンドの値を以下の通りにした.

ループ1回	Values= 48.90600000	./autocalc 1 1 1 1
ループ2回	Values= 48.90600000	./autocalc 1 2 1 1
ループ4回	Values= 48.90600000	./autocalc 1 2 1 2
ループ8回	Values= 48.90600000, 54.05400000	./autocalc 1 2 1 2
ループ12回	Values= 48.90600000, 54.05400000	./autocalc 1 2 1 3
ループ16回	Values= 48.90600000, 54.05400000	./autocalc 1 2 1 4
ループ18回	Values= 48.90600000, 54.05400000	./autocalc 1 3 1 3

参考文献

[1] Rovert W. Lucke　「Building Clustered Linux Systems」

[2] Hal Stern 著　倉骨 彰 訳　「NFS and NIS」（株式会社アスキー　1992）

[3] 馬場 敬信 著　「コンピュータアーキテクチャ」（株式会社オーム社　2005）

[4] 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）

謝辞

本研究を遂行するにあたり，終始，多大なるご指導及び御教示を賜りました関西学院大学理工学部情報科学科教授西谷滋人先生に深く感謝の意を示すと共に，厚く御礼申し上げます.

また，研究を進めていくうえで直接のご指導を頂いた，関西学院大学院大学理工学研究科情報科学専攻修士一回の竹田諒平氏，西川篤史氏，坂本憲氏に心から感謝します.

最後に，それぞれ別の研究内容ではありましたが，関西学院大学理工学部情報科学科西谷研究室の皆様に心からの謝意を表します.