卒業論文

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　理工学部　情報科学科
2703 竹田　諒平

2006年　3月

指導教員　　西谷　滋人　教授

概 要

VASP は比較的高速，高精度に固体物性が再現できる事から広く使われている．しかし，100 個程度の現実的な系を計算しようとすると，最高性能 PC を用いても 3,4 日程度計算時間が必要となる．従って VASP の高速化は西谷研究室において不可避の課題である．基本となる数値計算は行列・行列演算，行列・ベクトル演算，高速フーリエ変換から構成されている．VASP のような複雑なコードの実行速度は単純に見積もることができない．そのため，まず一般的な数値計算の問題を使って数値計算ライブラリー自身の性能を比較する．その上，VASP は数値計算の塊であり連立方程式，固有値，FFT を計算する性能に大きく依存する. このため数値計算ライブラリ (LAPACK,BLAS,FFT), コンパイラー及びコンパイルオプションを変えて，実行ファイルを作り実行速度を計測する必要がある．本研究では，様々のライブラリー，CPU で VASP を実行し最適な組み合わせを求めること，並列化することによってどれくらい性能が上がるかを求めることを目的とした.

連立 1 次方程式，固有値を求める C 言語のプログラムにおいて，数値計算ライブラリー LAPACK,BLAS を変えて計測した．数値計算ライブラリーを使用し VASP をコンパイルし，実行し一般的な場合と VASP で実行する場合とで数値計算ライブラリーの性能を比較した．VASP の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 を使用した．その．コンパイルオプションを変えて VASP を実行し性能を比較した．並列化によってどれぐらい速度が上がるかを測定し，さらに CPU 数による速度変化を測定した.

結果，数値計算ライブラリーは CPU によって性能が変わってくるため，一般的な場合で数値計算ライブ ラリーの性能を確かめる必要があると考えられる．CPU が northwood の場合 LAPACK が lmkl_lapack64, BLAS が lmkl_p4 とすることで LAPACK が liblapack, BLAS が libblas に比べて 51% 良い性能がでた．CPU が prescott の場合 LAPACK が lapack_double, BLAS が libgoto とすることで LAPACK が lapack_double, BLAS が lmkl_em64t に比べて 40%良い性能がでた．コンパイルオプションを-O0 から-O1 に変えることで 60%実行速度が 速くなり，数値計算ライブラリーとコンパイルオプションを最適にすると 2 倍実行 速度が速くなった．並列時の VASP の性能は CPU が northwood の場合に LAPACK が lmkl_lapack64, BLAS が lmkl とすると CPU2 個で 36%，CPU3 個で 95%，CPU4 個で 249%実行速度が速くなった.

目 次

第1章　イントロダクション

VASP は比較的高速，高精度に固体物性が再現できる事から広く使われている．しかし，100 個程度の現実的な系を計算しようとすると，最高性能 PC を用いても 3,4 日程度計算時間が必要となる．従って VASP の高速化は西谷研究室において不可避の課題である．基本となる数値計算は行列・行列演算，行列・ベクトル演算，高速フーリエ変換から構成されている．VASP のような複雑なコードの実行速度は単純に見積もることができない．そのため，まず一般的な数値計算の問題を使って数値計算ライブラリー自身の性能を比較する．その上，VASP は数値計算の塊であり連立方程式，固有値，FFT を計算する性能に大きく依存する. このため数値計算ライブラリ (LAPACK,BLAS,FFT), コンパイラー及びコンパイルオプションを変えて，実行ファイルを作り実行速度を計測する必要がある．本研究では，様々のライブラリー，CPU で VASP を実行し最適な組み合わせを求めること，並列化することによってどれくらい性能が上がるかを求めることを目的とする.

第2章　手法

1. 連立1次方程式，固有値を求めるC言語のプログラムにおいて，数値計算ライブラリーLAPACK,BLASを変えて計測する．
2. 数値計算ライブラリーを使用しVASPをコンパイルし，実行し一般的な場合とVASPで実行する場合とで数値計算ライブラリーの性能を比較する．コンパイルオプションを変えてVASPを実行し性能を比較する．VASPの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．
3. 並列化によってどれぐらい速度が上がるかを測定し，さらにCPU数による速度変化を測定する．

2.1　使用するCPU

1. CPU: Intel Pentium4 Northwood (3.2GHz FSB=800MHz)

   OS: SuSE Linux9.3

   マザーボード: ASUS P4C800

   キャッシュ容量: 2次:512KB

   メモリ: 1GB (Trancend PC3200 512MB ECC DIMM × 2)

   ハードティスク: Seagate ST380817AS(SerialATA 80GB) × 1

2. CPU: Intel Pentium4 Prescott 650(3.4GHz FSB=800MHz)

   OS: SuSE Linux9.3

   マザーボード: SuperMicro PDSGE

   キャッシュ容量: 2次:2MB

   メモリ: 2GB (PC2-4200 ECC 1GB DIMM × 2)

   ハードティスク: Seagate ST3800817AS(SerialATA 80GB) × 1

第3章　BLAS,LAPACKの性能

3.1　連立方程式

連立1次方程式の解を求めるC言語のプログラム (A参照) において数値計算ライブラリーLAPACK,BLASを変えて計測する．LAPACK(Linear Algebra PACKage)とは，数学ソフトウェア，スーパーコンピュータの性能評価データなどの情報を提供するnetlibで公開されている数値計算ライブラリである．各ルーチンはFORTRAN 77で記述されている．CLAPACKはそのC版である．LAPACKはCPUによる違いをBLAS(Basic Linear Algebraic Subprograms) によって吸収してある。すなわち、 線形代数の基本演算をBLASの関数にし、これを標準化している。そして、LAPACKの関数をBLASを用いて書くことによってポータブル化を実現している．自分の使用するCPUに対して最適化されているBLAS(アセンブラ) を用いることによって、C言語で書かれたBLASを 用いた場合の十倍程高速化が達成されている．線形方程式の直接解法に関しては、最適化BLASとLAPACKの組み合わせは、最も優れた組み合わせの一つとなる．連立方程式を解くことによって一般的な場合のLAPACK,BLASの性能が分かる．性能の評価はLAPACK利用の手引 [1] からn × nの行列のときLAPACKドライバに必要な”標準”浮動小数点演算数から引用し連立1次方程式の演算数は$0.67 \times N^3$になりこれを時間で割ると性能がでる．

3.1.1　Northwood

CPUがIntel Pentium4 northwood 3.2GHz搭載のコンピュータで連立1次方程式の解を求めるC言語のプログラムにおいて数値計算ライブラリーLAPACK,BLASを変えて計測する．

表3.1でLAPACKがliblapack,BLASがlibblasのときで1000 × 1000の行列を解くのに0.82秒の時間を要し817Mflopsの性能がでている．Mflopsとはコンピュータの処理速度をあらわす単位の一つで、1秒間に100万回の浮動小数点数演算 (実数計算) を実行できることを意味する．flopsは「Floating point number Operations Per Second」の略で1秒間に1回の計算ができる処理速度をあらわし、M(メガ) は100万 (10の6乗) を意味する接頭辞．同じライブラリーで2000 × 2000の行列を解く場合は8.24秒の時間を要し325Mflopsの性能がでている．ここで分かることは行列のサイズを増やせば性能も落ちていること，計算量2倍になれば計算時間

表 3.1: $N \times N$ の行列のときの連立方程式の解を求める性能.

LAPACK	BLAS	N=1000		N=2000	
		time	Mflops	time	Mflops
liblapack	libblas	0.82	817	8.24	325
	lmkl_p4	0.23	2913	1.53	1752
	libgoto	0.42	1595	3.72	720
lmkl-lapack64	libblas	0.22	3045	1.33	2015
	lmkl_p4	0.20	3350	1.32	2030
	libgoto	0.20	3350	1.31	2045

は単純に 2 倍にならずそれ以上の時間を要することである．VASP はこれより膨大な計算量であるので，計算時間はさらに増えると考えられる．

LAPACK が liblapack,BLAS が libgoto のときで 1000 × 1000 の行列を解くのに 0.42 秒の時間を要し 1596Mflops の性能がでている．2000 × 2000 の行列を解く場合は 3.72 秒の時間を要し 720Mflops の性能がでている．先ほどと同じように計算量を増えると性能が落ちることが分かる．あと，LAPACK は先ほどと同じなので libgoto は libblas より liblapack に対する相性が良いと考えられる．

LAPACK が liblapack,BLAS が lmkl_p4 のときで 1000 × 1000 の行列を解くのに 0.23 秒の時間を要し 2913Mflops の性能がでている．2000 × 2000 の行列を解く場合は 1.53 秒の時間を要し 1752Mflops の性能がでている．同じように計算量を増えると性能が落ちることが分かる．LAPACK は同じなので lmkl_p4 は libgoto と libblas より liblapack に対する相性が良いと考えられる．

LAPACK が lmkl_lapack64,BLAS が libbas, lmkl_p4, libgoto のときで 1000 × 1000 の行列を解くのに 0.22 秒, 0.2 秒, 0.2 秒の時間を要し 3045, 3350, 3350Mflops の性能がでている．2000 × 2000 の行列を解く場合は 1.33 秒, 1.32 秒, 1.31 秒の時間を要し 2015, 2030, 2045Mflops の性能がでている．これも同じように計算量を増えると性能が落ちることが分かる．LAPACK は lmkl_lapack64 の方が liblapack より性能が良いと考えられる．lmkl_lapack64 のときは BLAS による相性の違いがそれほどないと考えられる．

・ 表 3.1 で使用している LAPACK,BLAS の説明

・liblapack

SuSE Linux に付属されている LAPACK

・lmkl_lapack64

lmkl は lib Math Kernal LIbrary の略

lapack64 は倍精度データ型の LAPACK のこと

・**libblas**

SuSE Linux に付属されている BLAS

・**lmkl_p4**

Intel Math Kernal Library のライブラリー

p4 は Pentium4 プロセッサ用に最適化されたカーネル (BLAS,FFT)

・**libgoto**

テキサス大学オースチン校の後藤和茂氏による

Intel Pentium4 northwood 用の BLAS

http://www.tacc.utexas.edu/resources/software/ からダウンロード

3.1.2 Prescott

CPU が Intel Pentium4 prescott 3.4GHz 搭載のコンピュータで連立 1 次方程式の解を求める C 言語のプログラムにおいて数値計算ライブラリー LAPACK,BLAS を変えて計測する.

表 3.2: $N \times N$ の行列のときの連立方程式の解を求める性能.

LAPACK	BLAS	N=1000		N=2000	
		time	Mflops	time	Mflops
liblapack	libblas	0.86	779	6.60	406
lmkl_lapack64	lmkl_em64t	0.45	1489	4.27	628
lmkl_lapack64	libgoto	0.16	4188	1.07	2505

表 3.2 で LAPACK が liblapack,BLAS が libblas のときで 1000 × 1000 の行列を解くのに 0.86 秒の時間を要し 779Mflops の性能がでている．2000 × 2000 の行列を解く場合は 6.60 秒の時間を要し 406Mflops の性能がでている．ここで分かることは行列を増やせば性能も落ちていること，VASP はこれより膨大な計算量であるので，計算時間はさらに増えると考えられる．

LAPACK が lmkl_lapack64,BLAS が lmkl_em64t のときで 1000 × 1000 の行列を解くのに 0.45 秒の時間を要し 1489Mflops の性能がでている．2000 × 2000 の行列を解く場合は 4.27 秒の時間を要し 628Mflops の性能がでている．

LAPACK が lmkl_lapack64, BLAS が libgoto の場合は 1000 × 1000 の行列を解くのに 0.16 秒の時間を要し 4188Mflops の性能がでている．2000 × 2000 の行列を解く場合は 1.07 秒の時間を要し 2505Mflops の性能がでている．CPU が Prescott のときは Intel Math Kernal Library の BLAS より libgoto の方が Intel Math Kernal Library の LAPACK に対する相性がよいと考えられる．

・ 表 3.2 で使用している LAPACK,BLAS の説明

・**liblapack**

CLAPACK に付属されている LAPACK

CLAPACK とは本来 Fortran で記述されている LAPACK を C に

変換したもの

・**lmkl_lapack64**

Intel Math Kernal Library のインテル EM64T 対応プロセッサ用の

倍精度データ型の LAPACK

・**libblas**

CLAPACK に付属されている BLAS

・**lmkl_em64t**

Intel Math Kernal Library のインテル EM64T 対応プロセッサ用に

最適化されたカーネル

・**libgoto**

libgoto の Intel Pentium4 prescott 用の BLAS

3.2 固有値

CPU が Intel Pentium4 northwood 3.2GHz 搭載のコンピュータで固有値を求める C 言語のプログラム (B 参照) において数値計算ライブラリー LAPACK,BLAS を変えて計測する．これによって一般的な LAPACK,BLAS の性能が分かる．性能の評価は LAPACK 利用の手引 [1] から n × n の行列のとき LAPACK ドライバに必要な”標準”浮動小数点演算数から引用し固有値の演算数は $1.33 \times N^3$ になりこれを時間で割ると性能がでる．

表 3.3: N × N の行列のときの固有値を求める性能.

LAPACK	*BLAS*	N=1000		N=2000	
		time	Mflops	time	Mflops
liblapack	libblas	10.41	128	79.39	67
liblapack	lmkl_p4	9.90	134	74.87	71
lmkl_lapack64	lmkl_p4	9.62	138	72.47	73

表 3.3 で LAPACK が liblapack, BLAS が libblas のときで 1000 × 1000 の行列を解くのに 10.41 秒の時間を要し 128Mflops の性能がでている．2000 × 2000 の行列を解く場合は 79.39 秒の時間を要し 67Mflops の性能がでている．BLAS を lmkl_p4 に変えると 1000 × 1000 の行列ときは 9.9 秒で 134Mflops 出ていて，2000 × 2000 のときは 74.87 秒で 71Mflops の性能が出ている．

LAPACK が lmkl_lapack64, BLAS が lmkl_p4 のときで 1000 × 1000 の行列を解くのに 9.62 秒の時間を要し 138Mflops の性能がでている．2000 × 2000 の行列を解く場合は 72.47 秒の時間を要し 73Mflops の性能がでている．

3.3 まとめ

固有値で求めた結果は連立方程式で求めたときほど数値計算ライブラリーの性能に差はでなかったがライブラリーの性能の順番は同等であった．

しかし，同じ種類の数値計算ライブラリーを使用しても CPU によって性能が変わる場合が確認される．例えば libgoto の BLAS など Northwood では Intel Math Kernal Library の BLAS より性能は劣るが，Prescott になると libgoto の方が数段よい性能がでていることがわかる．数値計算ライブラリーは CPU によって性能が変わってくるため一般的な場合で数値計算ライブラリーの性能を確かめてみる必要がある．

第4章　シングルCPU時のVASPの性能

4.1　数値計算ライブラリーの依存性

4.1.1　Northwood

CPU が Intel Pentium4 northwood 3.2GHz 搭載のコンピュータで数値計算ライブラリーを変えて VASP をコンパイル (C 参照) する．VASP の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． コンパイルオプションは Intel コンパイラーで同じオプションを指定する．表 4.1 では数値計算ライブラリーが同じものを使用するとき一般的な場合と VASP を実行した場合では同じくらいの性能がでることが分かる．

表 4.1: ライブラリーを変えた時の VASP の性能.

BLAS	*LAPACK*	time
lmkl_p4	lapack_double	203.5
	lmkl_lapack64	201.2
	liblapack	202.9
libgoto	lapack_double	294.4
libblas	liblapack	306.5

4.1.2 Prescott

CPU が Intel Pentium4 prescott 3.4GHz 搭載のコンピュータで数値計算ライブラリーを変えて VASP をコンパイルする．VASP の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． コンパイルオプションは Intel コンパイラーで同じオプションを指定する．表 4.2 では数値計算ライブラリーが同じものを使用するとき一般的な場合と VASP を実行した場合では同じくらいの性能がでることが分かる．

表 4.2: ライブラリーを変えた時の VASP の性能.

BLAS	*LAPACK*	time
lmkl_em64t	lapack_double	192.7
	lmkl_lapack64	192.1
libgoto	lapack_double	137.4

数値計算ライブラリーを変えたときの性能は一般的な場合 (連立方程式, 固有値) と VASP で測定した場合で同等な性能がでることが分かる．

4.2 コンパイルオプション

コンパイルオプションは Intel コンパイラーでは基本的に O0,O1,O2,O3 とある．

・O0 は最適化は行われない

・O1 はオブジェクトのサイズを増やす傾向がある最適化を省略する．多くの場合，最小限のサイズで最適化されたコードが作成される．コードのサイズが大きいために，メモリ・ページングが問題になっている巨大なサーバー/データベース・アプリケーションにおいて，このオプションは効果的であることが証明されている．

・O2 は IA-32 プロセッサ搭載の Linux システム環境では O1 と O2 の違いはない．

・O3 は O1 に相当するが，ループ変換およびメモリ使用効率を向上させるためのデータ・プリフェッチ機能も含まれている．このオプションは，多様なアプリケーションに効果的であることがわかっている．ループが多いカーネル・ベースのコードに大して非常に効果的である．また，Pentium4 プロセッサおよびそれ以降の IA-32 プロセッサ環境で O3 を最大限に活用するには，ベクトル化のオプション指定し，コンパイルする. オプションは x{K|W|N|B|P} になる.「x」に続く英文字は Intel 製プロセッサの開発コード名の先頭 1 文字

である.「K」は Pentium Ⅲ プロセッサの「Katmai」,「W」が初代 Pentium 4 プロセッサの「Willamete」,「N」が「Northwood」,「B」が Pentium M プロセッサの「Banias」,「P」が「Prescott」になる.

例えば，-xP オプションを指定したとする．これは，Intel コンパイラーがサポートするベクトル化機能を有効にするだけでなく，SSE3 命令を利用することも表している．SSE2 までの命令セットを使用する場合は，-xN か-xW オプションを指定する．もし northwood のように SSE3 の命令セットがサポートされていないコンピュータで実行-xP オプションを実行した場合「This program was not built to run on the processor in your system.」というエラーメッセージが表示される．また不特定のプロセッサで実行される可能性のあるアプリケーションをビルドする場合は ax{K|W|N|B|P} オプションを使用する．

4.2.1 Northwood

CPU が Intel Pentium4 northwood 3.2GHz 搭載のコンピュータでコンパイルオプションを変えて VASP をコンパイルする．VASP の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．

表 4.3: コンパイルオプション変えたときの VASP 性能.

OPTION	*BLAS*	*LAPACK*	time
-O0	lmkl_p4	lapack_double	323.8
-O1			204.6
-O3 -xW -tpp7			203.5
-O3 -axN -xN-tpp7 -ip -mp1			203.0
-O0	lmkl_p4	lmkl_lapack64	323.9
-O1			206.8
-O3 -xW -tpp7			201.2
-O3 -axN -xN-tpp7 -ip -mp1			200.2
-O0	libgoto	lapack_double	435.5
-O1			316.7
-O3 -xW -tpp7			309.0
-O3 -axN -xN-tpp7 -ip -mp1			308.7

4.2.2 Prescott

CPU が Intel Pentium4 prescott 3.4GHz 搭載のコンピュータでコンパイルオプションを変えて VASP をコンパイルする．VASP の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．

表 4.4: コンパイルオプション変えたときの VASP 性能.

OPTION	$BLAS$	$LAPACK$	time
-O0	libgoto	lapack_double	254.0
-O1			140.8
-O3 -xW -tpp7			136.4
-O3 -axP -xP-tpp7 -ip -mp1			133.8
-O0	lmkl_em64t	llapack_double	309.0
-O1			195.3
-O3 -xW -tpp7			191.0
-O3 -axP -xP-tpp7 -ip -mp1			188.7
-O0	lmkl_em64t	lmkl_lapack64	308.1
-O1			194.7
-O3 -xW -tpp7			190.2
-O3 -axP -xP-tpp7 -ip -mp1			187.9

表 4.3, 表 4.4 はどの組み合わせの数値計算ライブラリーを使用してもコンパイルオプション-O0 は-O1 に比べて大幅に性能が落ちる．-03 関連のオプションでは-O0,-O1 に比べてコンパイルするのに数分程度長い時間を必要とする．しかも，性能は数パーセントしか性能が向上しない．しかし，VASP を実行するには膨大な時間を必要となる場合があるため数パーセントでも大幅な時間の短縮となるため，コンパイルに少々時間を要してもなるべく優れているコンパイルオプションを指定する必要があると考えられる．

第5章　並列時のVASPの性能

CPU が Intel Pentium4 prescott 3.4GHz 搭載のコンピュータを 2 台を並列化して VASP の実行速度計測する．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．分散型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，協調性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ぶ．そして，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．そして，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．

並列化するにあたりこの MPICH(C 参照) を使用する．VASP をコンパイルする際にも MPI 用にする必要がある．実行する際には MPICH のコマンドの mpirun を使って実行する．

表 5.2 の CPU 数 1 で BLAS が lmkl_em64t,LAPACK が lmkl_lapack64 の VASP の実行時間は 192.7 秒かかり，CPU 数 2 にすると 121.9 秒かかる．CPU 数を 2 に増やすと VASP の実行速度は 58%向上する．

CPU 数 1 で BLAS が lmkl_em64t,LAPACK が lapack_double の VASP の実行時間は 195.3 秒かかり，CPU 数 2 にすると 123.3 秒かかる．CPU 数を 2 に増やすと VASP の実行速度は 58%向上する．

CPU 数 1 で BLAS が libgoto,LAPACK が lapack_double の VASP の実行時間は 137.4 秒かかり，CPU 数 2 にすると 93.5 秒かかる．CPU 数を 2 に増やすと VASP の実行速度は 47%向上する．

CPU 数を 2 にすると VASP の実行速度の上昇率は数値計算ライブラリーによって違うことが分かる．

表 5.1: CPU 数を変えたときの VASP 性能.

NODE	*BLAS*	*LAPACK*	time
1	lmkl_em64t	lmkl_lapack64	192.7
2	lmkl_em64t	lmkl_lapack64	121.9
1	lmkl_em64t	lapack_double	195.3
2	lmkl_em64t	lapack_double	123.3
1	libgoto	lapack_double	137.4
2	libgoto	lapack_double	93.5

5.1 ノードを増やしたとき VASP の性能

CPU が Intel Pentium4 northwood 3.2GHz 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．ノードが 1 のときはシングル CPU 時の VASP の設定になっており，ノードが 2 以上のときは並列時の VASP の設定になっている．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ならない．その上，納入ままのシステムで導入された MPICH では図 5.1 の点線のように CPU 数を 2 に増やすとシングル CPU 時より性能が落ちる傾向になっていた．数値計算ライブラリーの BLAS を lmkl から libgoto に，LAPACK を-lmkl_lapack64 から lapack_double に変え，コンパイルオプションを-O0 から-O3 -mp1 -tpp7 に変えることによって図 5.1 の実線のように CPU 数を 2 に増やした場合にも性能が落ちることは解消された．図 5.1 の実線は CPU 数の 2 と 4 でみると 1/CPU 数が成り立っていて，CPU 数を 8 にすると性能が落ちてくるので CPU 数を 4 までにすることがよいと考えられる．

表 5.2: CPU 数を変えたときの VASP 性能.

BLAS	*LAPACK*	OPTION	NODE	time
lmkl	lmkl_lapack64	-O0 -mp1	1	341.7
			2	220.9
			3	155.3
			4	116.6
			8	119.2
lmkl	lmkl_lapack64	-O1 -mp1	1	213.0
			2	158.2
			3	110.1
			4	85.8
			8	102.8
lmkl	lmkl_lapack64	-O3 -mp1 -tpp7	1	210.3
			2	154.7
			3	107.6
			4	84.5
			8	102.1
libgoto	lapack_double	-O3 -mp1 -tpp7	1	244.0
			2	175.9
			3	123.0
			4	95.6
			8	108.3

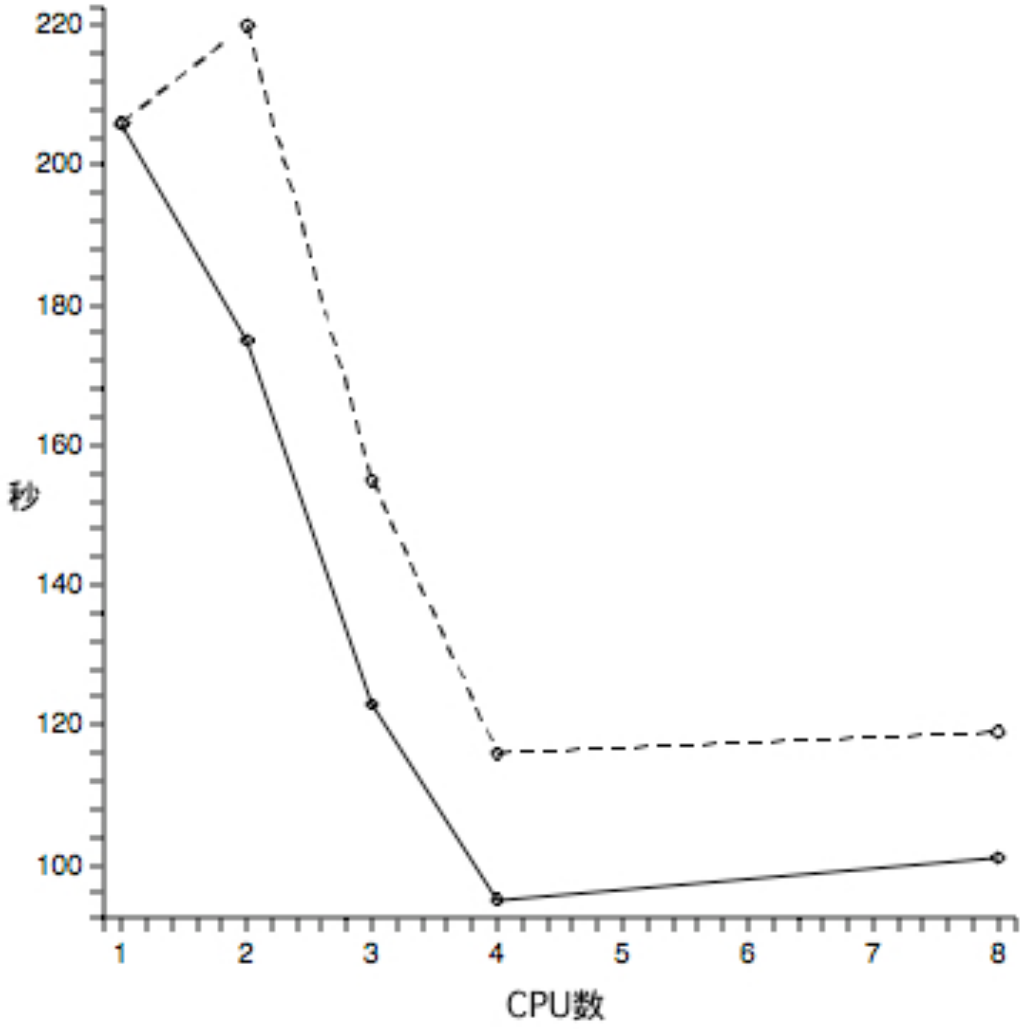

図 5.1: VASP 実行時の CPU 数と時間の関係

第6章　まとめ

1. BLAS,LAPACK の性能

・数値計算ライブラリーは CPU によって性能が変わってくるため，一般的な場合で数値計算ライブ ラリーの性能を確かめる必要がある．

2. シングル CPU 時の VASP の性能

・CPU が northwood の場合 LAPACK が lmkl_lapack64, BLAS が lmkl_p4 とすることで LAPACK が liblapack, BLAS が libblas に比べて 51% 良い性能がでる．

・CPU が prescott の場合 LAPACK が lapack_double, BLAS が libgoto とすることで LAPACK が lapack_double, BLAS が lmkl_em64t に比べて 40%良い性能がでる．

・コンパイルオプションを-O0 から-O1 に変えることで 60%実行速度が速くなる．

・数値計算ライブラリーとコンパイルオプションを最適にすると 2 倍実行速度が速くなる．

3. 並列時の VASP の性能

・CPU が northwood の場合に LAPACK が lmkl_lapack64, BLAS が lmkl

・CPU2 個　36%実行速度が速くなる．

・CPU3 個　95%実行速度が速くなる．

・CPU4 個　249%実行速度が速くなる．

付 録A　連立方程式を求めるプログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
//#include <vecLib/vecLib.h>
#include "/usr/local/include/f2c.h"
#include "/usr/local/include/clapack.h"
void printMatrix(double *a, double *b, int n);

int  main(void){
  long n, nrhs=1, lda, ldb, info;
  // double A[LDA*LDA], B[LDA*NRHS];
  clock_t start, end;
  int i,j;
  double *a, *b;
  long *ipiv;

  scanf("%ld",&n);
  printf("%dn",n);

  lda=ldb=n;
  a=(double *)malloc(n*n*sizeof(double));
  b=(double *)malloc(n*sizeof(double));
  ipiv=(long *)malloc(n*sizeof(long));

  for(i=0;i<n;i++){
    for(j=0;j<n;j++){
      a[i*n+j]=2*(double)random() / RAND_MAX -1.0;
    }
  }
```

```
  for(i=0;i<n;i++){
    b[i]=2*(double)random() / RAND_MAX -1.0;
  }
  // printMatrix(a,b,n);
  start=clock();
  dgesv_(&n,&nrhs, a, &lda, ipiv, b, &ldb, &info);
  // MatrixInverse(a,b,n);
  // printMatrix(a,b,n);
  end=clock();
  printf("%10.4fn",(double)(end-start)/CLOCKS_PER_SEC);

  free(a);
  free(b);

  return 0;
}

void printMatrix(double *a, double *b, int n){
 int i,j;

 for(i=0;i<n;i++){
   for(j=0;j<n;j++){
  printf("%10.5f",a[i*n+j]);
  }
  printf("%10.5f",b[i]);
  printf("n");
  }
  printf("n");
  return;
}
```

付 録B　固有値を求めるプログラム

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
//#include <vecLib/vecLib.h>
#include "/usr/local/include/f2c.h"
#include "/usr/local/include/clapack.h"
void printMatrix(double *a, double *b, int n);

int  main(void){
long n, lda, lwork, info;
// double A[LDA*LDA], B[LDA*NRHS];
char jobs='V', uplo='U';
clock_t start, end;
int i,j;
double *a, *w, *work;
long *ipiv;

scanf("%ld",&n);
printf("%dn",n);

lda=n;
lwork=n*3;
a=(double *)malloc(n*n*sizeof(double));
w=(double *)malloc(n*sizeof(double));
work=(double *)malloc(n*sizeof(double));

for(i=0;i<n;i++){
  for(j=0;j<n;j++){
    a[i*n+j]=2*(double)random() / RAND_MAX -1.0;
    }
  }
```

```
 //  printMatrix(a,w,n);
  start=clock();
  //dgesv_(&n,&nrhs, a, &lda, ipiv, b, &ldb, &info);
  dsyev_( &jobs, &uplo, &n, a, &lda, w, work, &lwork, &info);
  // MatrixInverse(a,b,n);
  // printMatrix(a,b,n);
  end=clock();
  printf("%10.4fn",(double)(end-start)/CLOCKS_PER_SEC);

  free(a);
  free(w);

return 0;
}

void printMatrix(double *a, double *b, int n){
int i,j;

for(i=0;i<n;i++){
  for(j=0;j<n;j++){
    printf("%10.5f",a[i*n+j]);
      }
    printf("%10.5f",b[i]);
    printf("n");
}
printf("n");
return;

}
```

付 録C　VASPマニュアル

1. Intel Fortran Compiler Version 9 のインストール

・ライセンスファイル (*.lic) を /opt/intel/liceses/にコピーする.

例　ライセンスファイルがデスクットップにある場合.

mv /Desktop/commercial_for_l_*.lic /opt/intel/licenses/

・$ cd /media/l_fc_p_9_0

で Intel Fortran Compiler Version 9 に入る.

・$ ./install.sh

でインストールが開始される.

・インストール実行

Please type a selection: 1

Please type a selection: 2

・ライセンスを入力するとき

/opt/intel/liceses/commercial_for_l_*.lic とかく.

使用許諾に同意に accept を入力する.

インストールが成功していれば　x.Exit で出る.

2. Intel C++ Compiler Version 9 のインストール

・Fortran と同様にライセンスファイルを /opt/intel/liceses/にコピーする.

・$ cd /media/l_cc_p_9_0

・$ ./install.sh

・インストール実行

Fortarn と同様

しかし　Linux Application Debugger は Fortran でインストールしたので入れない.

3. Fortran, C++の環境設定

・.cshrc ファイルを設定する

・$ emacs ./cshrc

set path= /opt/intel/fc/9.0/bin /opt/intel/cc/9.0/bin を付け加える

setenv LD_LIBRARY_PATH /opt/intel/mkl72/lib32:

/opt/intel/fc/9.0/lib を付け加える

4. YaST2 でパッケージのインストール

・gcc 関連，glibc インストール

・fftw3 関連 (fftw fftw3 fftw3-debuginfo fftw3-devel fftw3-threads) インストール

・lapack,blas インストール

5. vasp.4.6.tar vasp.4.lib.tar の解凍

・$ tar -xvf vasp.4.6.tar

・$ tar -xvf vasp.4.lib.tar

これで，vasp.4.6/ vasp.4.lib/が作成される.

6. vasp.lib の編集

・$ cd vasp.lib/

・$ cp makefile.linux_ifc_P4 makefile

vasp.lib/のフォルダーに Linux で Intel fortran compiler(ifc),P4 で書かれている

makefile の例を makefile にコピーする

・$ emacs makefile

FC=ifc を FC=ifort に書き換える

Intel fortran compiler を ifc から ifort に変える.

・$ make でコンパイルされる.

7. vasp.4.6 の編集

・$ cd vasp.4.6

・$ cp makefile.linux_ifc_P4 makefile

・$ emacs makefile

7.1. FC=ifc を FC=ifort に書き換える

7.2. BLAS

makefile の中で

BLAS /opt/libs/lbgoto_p4_512-r0.6.so になっているところを

・northwood の場合

BLAS=-L/opt/intel/mkl72/lib/32 -lmkl_p4 -lsvml に変更する

・prescott の場合

BLAS=-L/opt/intel/mkl72/lib/em64t -lmkl_em64t -lpthread -lsvml

に変更する

・libgoto の BLAS を使う場合

$ cd /opt/libs/

$ mkdir libgoto

libgoto のフォルダーを作りこの中 libgoto の BLAS を入れておく

・makefile では CPU が Intel の northwood(presccot) の場合

・BLAS=-L/opt/libs/libgoto/libgoto_northwood(prescott)32p-r1.00.so

-lpthread -lsvml

・LINK = -lirc -lguide -lsvml -lcprts -lunwind -lcxa -lifport

Wl,-rpath=/opt/libs/libgoto

の２行を書き換える

7.3. LAPACK

LAPACK= ../vasp4.lib/lapack_double.o とそのままにする.

・Intel Math Kernel Library の LAPACK を使う場合

・northwood

LAPACK=-L/opt/intel/mkl72/lib/32 -Imkl_lapack64

・presccot

LAPACK= -L/opt/intel/mkl72/lib/em64t -lmkl_lapack64 -lguide

7.4. FFT3D

・northwood

FFT3D = fftw3d.o fft3dlib.o /usr/lib/libfftw3.a

・presccot

FFT3D= fft3dfurth.o fft3dlib.o /usr/lib64/libffw3.a

7.5. MPI

D の MPICH マニュアルを参照

FC=ifort -I/usr/lib/mpich-1.2.5.2/

FCL=/usr/lib/mpich-1.2.5.2/bin/mpif90 と書き換える

7.6. コンパイル

$ make でコンパイル

cannot open shared object file とエラーが出る場合は PATH が

通っていない

コンパイルが通れば vasp.4.6 の中に./vasp の実行ファイルができる

8. 実行

VASP を実行するために Hg.tar をダウンロードする

$ tar -xvf Hg.tar で解凍する

$ cd Hg

$ directory_where_VASP_resides/vasp で実行

付 録D　MPICHマニュアル

1. 準備

・/etc/hosts に IP アドレスとドメイン名を入力する

例

192.168.3.4 bob1

192.168.3.5 bob2

・/etc/hosts.equiv にドメイン名を入力する

これをすることによって実行する際 ./hosts がいらない

例

bob1

bob2

1. MPICH をインストールする

http://www-unix.mcs.anl.gov/mpi/mpich/から

mpich-1.2.5.2.tar.gz をダウンロードする

$ tar -xvf mpich-1.2.5.2.tar.gz で解凍する

$ cd mpich-1.2.5.2

$ ./configure –prefix=/usr/lib/mpich-1.2.5.2 –prefix=の後は自分の好きなディレクトリを選べる

例 $ ./configure –with-arch=LINUX –with-device=ch_p4

-fc=ifort -f90=ifort -prefix=/usr/local/bin

mpich-1.2.5.2 の中で

$ make

$ make install でインストールは終わり

2. 動作を確認

$ cd /usr/lib/mpich-1.2.5.2/examples

$ make cpi

$ ./mpirun -np 1 cpi が動くか確かめる.

例

Process 0 on takeda1

pi is approximately 3.141600989231254, Error is 0.000000833333333323

wall clock time =0.000000 となれば動いている

3. 並列時の動作確認

3.1. マシンファイルの設定

$ cd /usr/local/mpich-1.2.5.2/share/

その中にある machines.LINUX を環境に合わせて修正する

ここでは takeda1，takeda2 を追加する

例# コメント行省略

takeda1

takeda2

3.2. パスの設定

最後に，パスを通す

PATH に/usr/local/mpich-1.2.5.2/bin を追加する

・bsh の場合

$ export PATH=$PATH:/usr/local/mpich-1.2.5.2/bin

・csh の場合

.cshrc ファイルに

set path=/usr/local/mpich-1.2.5.2/bin を記述する

3.3. 実行

$ ./mpirun -np 2 cpi が動くか確かめる.

例

Process 0 on takeda1

Process 1 on takeda2

pi is approximately 3.141600989231254, Error is 0.000000833333333323

wall clock time =0.000000 となれば動いている

3.4. VASP での実行

$ directory_where_VASP_resides/vasp で実行の所に

$ ./mpirun -np 2 を付け加えて

$ ./mpirun -np 2 directory_where_VASP_resides/vasp で実行

謝辞

本研究を遂行するにあたり，終始多大な御指導及び有益な御指示を賜りました西谷滋人教授に深く感謝の意を示すとともに厚くお礼申し上げます.

参考文献

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorense 小国 力 訳, 「行列計算パッケージ LAPACK 利用の手引」(丸善株式会社 1995)

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

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