Linux/Alpha 活用講座

清水尚彦　nshimizu@keyaki.cc.u-tokai.ac.jp

HPCプログラミング入門

21世紀になりましたが、LSIの加工技術はまだまだ向上しそうな雰囲気で、計算機アーキテクトは忙しい日々を送っていることでしょう。IntelやIBMがプロセッサ技術に投資を行うと、他社も追従しなくてはならず、投資余力の少ない会社は厳しくなってきますね。

もっとも、ファブ専業メーカーも、最近では力を付けてきたので、ファブレスとの組み合わせで、どこまで戦えるのか興味が尽きないものがあります。

21世紀ということで、ますます環境問題も厳しくなりますが、かつて「グリーンPC」が流行ったことなど忘れたかのように、ひたすら消費電力を注ぎ込んで速度競争をしているこの業界も、モバイルだけでなく、デスクトップの消費電力についても見直される時期がくるのだと思います。特に、多くの命令を予測で実行して大量に実行結果を捨てているようなタイプのプロセッサ設計を改めないと、性能電力比はなかなか良くなっていきませんね。並列処理が当り前の世界では、性能電力比が悪いと、場合によっては単体性能だけ勝っていても受注競争に負けることもあり得ると思っています。

車の世界でも、アメリカのカリフォルニア州では販売台数の2%を排気ガスを出さない車にしなくてはならない(ZEV*1)ので、General Motorsからも電気自動車速度記録を持っている「EV1」の市販版が(カルフォルニア州でだけ)出ていたりします(記事末のRESOURCE[1]を参照)。

この車は(運転や状況によって幅がありますが)、航続距離が160kmほどで、電池残量20%から80%への充電は6.6キロワットで2時間で終了します。すると、日本で使っても、電気代は10km当たり30円ほどと、ガソリン車より安上がりになります(アメリカはガス代が安いからと思ったら電気代も安いので、やっぱりガソリン車より安上がりなんだそうです)。

6.6キロワットほどなら、スーパーカブなんかの90ccエンジンでも出せそうですから、遠出のときには充電しながら走る「手製のハイブリッドカー」なんて作るのもいいかもしれません。電気自動車というと、遅くて使い物にならないと思っている人もいるかもしれませんが、EV1は、0→96km/hが約9秒と、ガソリン車と比較しても、それほど遜色ないものとなっています。

293km/hの速度記録を出したプロトタイプと違って、市販車の最高速は約130km/hとずっと抑えられていますけれど、これだけの速度が出れば、日本でもアメリカでもあまり困ることはないでしょう。市場価格は350万円ぐらいとまだまだ高いため、なかなかユーザーレベルでの経済性は引き合わないのですが、自治体レベルでプールして貸し出すような、所有にこだわらない社会システムを考えれば面白い存在ですね。

電気自動車やハイブリッドカーは、定常走行ではほとんどパワーが要らないという自動車の特性を利用して、バッファとなる電池のパワーを使用して実用上問題ない走行性能を確保しつつ、ブレーキ時にパワーを回生することで、さらに加速で消費したパワーを取り戻しています。

コンピュータも、パーソナルコンピュータなどの使い方を見れば、定常時にはあまりCPUパワーも必要ないことが理解できると思います。ところが自動車と違って、状況によって最高速で延々と走らなければならない場合があって、それに耐えられるだけのシステムが普通のマシンに対しても要求されます(自動車でもサーキットなどでは同じ状況でしょうけれど、これは特殊ケースと考えることができます)。しかも運動エネルギーと違って、熱になってしまったものは減速しても

*1　カリフォルニアZEV法(ZERO EMISSION VEHICLE)

回生は難しいですね。もっとも、使われ方の状況を考えればそういった用途は極めてまれで、サーキットで走るレーシングカーと同じだと言えるのかもしれません。

だとすると、特殊なチューニングをしたプロセッサだけ速く動けばいい、ということになってくるでしょう。性能を上げることばかり考えていたIntelですが、Transmetaのような会社が登場して、ユーザーが「これで十分」と言い始めたら必然的に方向転換が必要になります。

もっとも、投資余力がたっぷりあるIntelのことですから、複数のプロセッサプロジェクトを立ち上げてフルラインを揃える可能性もあって、市場のすみ分けがどうなっていくのかについては、まだ予断を許しません。

会社などで、多くのパーソナルコンピュータを使っているケースでは、隣であまりプロセッサを使っていない人がいたら、ちょっと処理能力だけ借用するような仕組みが普通になれば、設置済みのコンピュータが電気自動車のバッテリのようにバッファの役割をして軽くて電力／性能比の良いプロセッサを使って、さらにグリーン化を推進できるでしょう。

Alpha Linuxのニュースから

●Mathematica

数式処理システムとして有名な「Mathematica」がAlpha Linux用にリリースされたようです。私は使わないのですが、「なくてはならない」という人もいるようなのでそういった人達にとっても良いニュースですね。

数式処理というのは「記号処理」の仲間で、Lisp系の言語の得意とするところです。Mathematicaも、任意多倍長の演算とか、数式とか、Lisp系の言語が扱ってきた問題の領域を得意としています。Lisp系の言語では、すべてのオブジェクトを「リスト」と呼ぶ形式で保持するのですが、リストの実現はほとんどの処理系でポインタを使っています。すると、プログラムのメインの部分では、ポインタを追いかけることにプロセッサの大部分の時間が費やされることになります。実は、ポインタを追いかける処理は、ポインタを読んでみないと、次の処理が決まらないというデータ依存の典型的な問題になってしまいます。そのため、データ依存の処理が苦手なプロセッサには手強い相手となる可能性があります。

また、処理の進行と共に、オブジェクトを割り当てているセルの局所性が低下して、ポインタを追いかけるたびにアドレスが飛び飛びの場所をアクセスしにいくことになって、TLBミスが多発する可能性もあります。Mathematicaを使う人は、ぜひこういったことを頭において性能を評価してみてください。

●JDK 1.2.2

JDK 1.2.2がコンパックからリリースされました。Javaを Alpha Linuxで使ってみたいと思っていた方はお試しください。執筆時点ではβ版で、私はまだ試していないので詳細なコメントは控えさせていただきます。

新企画「HPCプログラミング入門」

さて、前回から始まったCXMLの利用方法も引続き連載で取り上げていきますが、今回から新たにHPCプログラミングの入門編として、プログラム初心者向けのプログラミング解説を行っていきます(というわけでCXMLは今回はお休みします。期待していた方ごめんなさい)。

HPCプログラミング入門は、まず「octave」による初級プログラミングに始まって、FORTRANプログラムとライブラリ、性能計測、Cプログラムの順に説明をしていきます。

●プログラムの役割

現在は、いろいろなソフトウェアが市販されていて、パーソナルコンピュータを使うために、自分でプログラムを作らなくてはならない人はほとんどいない時代になっています。ここで「なっています」と言ったのは、パーソナルコンピュータのプログラムを、自分で作成することが当り前という時代があったからです。

初期のパーソナルコンピュータは、「BASIC」というプログラム言語が添付されていて、自分でBASICのプログラムを作成して遊んだものです。やりたいことがあっても、自分でプログラムが作れなければできないということで、「コンピュータは難しい」と思われていたし、実際に役に立つことをさせようとすれば大変難しかったものです。

70年代の後半にヒットした「Apple II」というパーソナルコンピュータが、フロッピーディスクを備え、さらに「Visicalc」という表計算ソフトウェアやワープロソフトが販売されるようになって、ようやくパーソナルコンピュータがビジネスに使えるようになったのです。

その後、ビジネスに使うためにソフトを自作する人は激減して、「プログラム(ソフトウェア)は買ってくればいい」という時代になりました。ソフトウェアを購入できるということは、作って売る人がいる訳です。受注生産でないのなら、そ

のプログラムが多くの人にとって役に立つものでなければビジネスとして成り立ちません。

そこで、購入するソフトウェアは、多くの人にとって共通した作業を軽減するものであることが求められます。ワープロしかり、表計算しかりです。例えば、メールやインターネットブラウザなどは、人によってやることはそれほど変わらないので、ソフトウェアビジネスとして成立します(もちろん、囲い込みなどの理由で無償で配布することもあるのは言うまでもありません)。

ところが、こんな時代でも、あえて自分でプログラムを作らないと困るような分野がまだまだ存在するのです。その代表分野が「High Performance Computing(HPC)」もしくは「High Performance Technical Computing(HPTC)」と呼ばれる分野です。この分野では、買って来たプログラムを直接使うことが困難な場合が少なくありません。

「HPCって何をやっているんだろう?」と疑問に思う人も多いですよね。身近で分かりやすい例を挙げると、天気予報、車の衝突時の変形解析や空力特性解析、建築物の強度計算などがあります。これらの計算処理に共通しているものは、物理的な法則を表す方程式をコンピュータを使って数値的に計算することで、実際に発生する現象を予測するという処理です。このことを、われわれは「シミュレーション」と呼びます。

例えば、車の開発でシミュレーションを行う場合には、実際に物を作って壊してみる前にコンピュータの中だけでいろいろな可能性を実験してみることができるため、開発期間や開発コストを大幅に削減することができます。

シミュレーションのための物理法則は、「微分方程式」で表されていることが多いのは、多少でも物理をかじったことのある読者ならお分かりですよね。微分方程式を手計算で解くのは結構難題で、ちょっと複雑な方程式になると解けなくなります。これは計算機でも同じで、Mathematicaなどの数式処理ソフトでは、かなりの式まで解けるようなことをうたい文句にしていたと思いますが、複雑な問題の完全な解を求めるのは困難です。

そこで、微分の定義に戻って考えると、微分というのは、「ある変数の差分を微小値で割ったものの極限をとっている」ので、極限をとるのを止めて、「ある程度の小さな値で割って結果を出す」とするだけでも、何らかの近似値が得られるのです。

これを利用して微分方程式を「差分方程式」と呼ぶ近似式に変換して数値計算するという方法がよく使われます。すると、精度を上げるためには、なるべく微分方程式に近いほうが良いので、なるべく微小値を小さくとることで、方程式が大規模になっていきます。

例えば、3次元の物体(簡単のために立方体とします)にかかる力を計算することを考えてみましょう。物体を表す微分方程式を差分方程式で表すときに、物体を代表する「点」を考えます。一辺に10点ほど代表点を考えると、立方体を表すのには1000点を使用します。これを少し精度を上げるため、一辺100点とすると、立方体の代表点は100万点と一気に増えてしまいます。さらに一辺を1000点とすると10億点となってきますが、各点を倍精度の浮動小数点数で表すとなると、約8Gbytesの記憶場所(メモリ)がこの物体を表すために必要になるのです。

例えば、天気予報で日本の上空を表すとき、山や川などの地形を考慮することを想定すれば、1kmごとの代表点では物足りないと思いませんか? そう考えると、南北で2000kmもある範囲を計算することはかなり大変だということが実感できるでしょう。

●プログラミングの準備

プログラミング入門としては、何らかのプログラム言語を使わずには説明できません。HPC/HPTC分野では、従来からFORTRANが使われることが多かったのですが、まずはもっと簡単な言語からスタートすることにしましょう。簡単というと、よく引き合いに出されるのが「BASIC」という言語ですが、これはFORTRANに比べて、特に簡単になっている所もなく必然性もないので使いません。では何を使うかというと、「octave」という言語です。

Linuxの多くのディストリビューションでは、octaveはすでにインストールされているか、もしくは後からインストールできるようになっています。コンソールから「octave」と打って動作するようであれば、すでにインストールされています。インストールされていないようであれば、パッケージの説明書を読んでインストールしてください。ソースから構築する人は、ぜひCXML(Compaq eXtended Math Library)/CPML(Compaq Portable Math Library)をリンクして構築してください(CXML/CPMLのリンク方法については本誌2000年6月号を参照してください)。

もし、パッケージにも入っていないときはどうしたらいいのでしょう? そういったディストリビューションは、HPC分野を全く対象にしていない可能性があるので、さっさと別のディストリビューションに乗り換えましょう。これらのライブラリを使うことで、ディストリビューションの標準添付のものよりも相当高速なoctaveにすることができます。

octaveは、「インタプリタ」と呼ばれる逐次実行型のプログ

ラム環境です。プログラムをファイルに書いて実行することもできますが、ターミナルから直接プログラムを打ち込んで実行することもできます。まず、コンソールやxtermなどのコマンドラインからoctaveを起動します。

```
% octave
```

すると次のようにプロンプトが現れます。

```
octave:1>
```

ここにさまざまなコマンドや計算式を打ち込んで計算を行います。

●変数とメモリの話

すでに何かしらの手続き型プログラム言語を使ったことがある人には当り前の話ですが、octaveの計算では、「変数」と呼ぶオブジェクトに値を設定することが計算の大事な過程となります。変数に記憶することで、計算結果を他の計算に再利用できるため、計算の簡単化と計算量の削減が図れます。

C言語やFORTRAN言語の場合、変数はコンパイル時にメモリ上の場所が割り振られます(C言語の局所変数は、「スタック」という領域の先頭からの変移として割り振られます)。そこで、プログラム作成時には、どれだけの大きさの変数を割り当てるかを計画しておく必要があります。octaveの場合には、インタプリタで変数へのメモリ割り付けは変数への値の代入ごとに割り当てが必要かどうかを判定して、必要量だけ割り当てを行うように動作します。

すでに割り当てられているメモリには入らないような新しい代入が起きた場合には、前のメモリ割り当ては捨てられます。このようにoctaveでは、「メモリ上の場所」という下位レベルの管理からプログラム作成者を解放し、多少上位のメモリ管理システムを用いていると言えます。私は、プログラムの高級度はこのメモリという物理構造をいかにプログラマから隠ぺいできているかによって決まると勝手に思っています。

機械語の時代からアセンブラになることで、数値で表されるアドレスを文字で代替でき、アセンブラから高級言語に移ることで、さらに抽象度を上げてきています。Lispなどの記号処理が容易な言語では、当初からガベージコレクションを導入して、メモリアドレスからユーザーのオブジェクトを解放することに成功しています。もっとも、その代償として性能の劣化が発生したので、計算機の性能が低い時代には、なかなか使うのが難しかったのです。

現在は計算機の性能が向上したことで、LispやJavaなどの言語も、実用的なスピードで動作するようになりました。この観点から見ると、Cなどより、昔のLispの方がメモリ管理の面では高級だと言えます。

さて、それではoctaveの変数／定数の話をします。octaveでは、変数はアルファベットから始まる文字で指定します。変数の型はそこに代入したオブジェクトによってきまります。オブジェクトには数値、配列、文字列、構造体などがあります。

数値としては浮動小数点数と複素数が使えます。

```
1
1+2i
1e-2
```

などと表します。iが「虚数単位」です。「e」の後ろの数値で10のべき乗を表します。つまり1e-2は0.01のことです。こういった数値のことを、ベクトルや行列と区別して「スカラ」と呼びます。

配列の宣言は「[]」で囲ってします。普通に宣言すると行ベクトルになりますから、セミコロン「;」で行の切れ目を入れます。

```
[ 1 2 ; 3 4 ]
```

と打つと、

```
 1 2
 3 4
```

という配列として扱えます。行ベクトルの作り方はほかにもあります。例えば、「1:10」とすると「[1 2 3 4 5 6 7 8 9 10]」と同じことになります。「10:-1:1」とすると「[10 9 8 7 6 5 4 3 2 1]」となります。これらの数値の指定は整数でなくても構わないので、「0:0.1:0.9」のように、いろいろな値で試してみてください。

文字や文字列は「'」や「"」で囲んで示します。「abc」という文字列を示すには「"abc"」または「'abc'」とします。

これらのオブジェクトは、変数に代入することができます。また、特別な変数として「ans」というものがあって、直前に評価した式を記憶していますから、変数に入れなくても計算結果を再利用することも比較的簡単にできます。

配列を変数に入れると、その一部だけを使いたいこともあります。「a = [1 2 ; 3 4]」という配列から「(1,1)」の要素を取り出すときには、「a(1,1)」とすると「1」と、値が返ります。面白いことに文字列にもこのやり方が通用します。例えばaという変数に「a = "abc"」と文字列を代入すると、aは文字型になります。ちなみに「a(3)」とすると答えは「c」となります。つまり文字列は文字配列として扱われていることが分か

ります。

C言語と同じようにデータ構造も使えます。

```
x.a = 1
x.b = "abc"
```

などとして「x」と打つと、

```
x =
{
  a = 1
  b = abc
}
```

のようにデータ構造となって格納されます。また、「y = x」のように、他の変数に構造ごと代入することもできます。加減乗除などの演算は数値と配列の双方に使えますが、構造体には使えません。

●octaveの演算

さて、変数やオブジェクトを理解したところでこれらに対する演算について示しましょう。octaveにはさまざまな演算子が定義されていますが、一度に大量に示しても、混乱するので必要なものから徐々に示していきます。これらの演算は若干の制約はあっても、行列とスカラの両方に使えます。

最初に説明するのは比較的馴染みのにあると思われる表1に示す演算です。これらの演算は数値もしくは行列に対して実行できます。行列には要素ごとの演算も別途定義されていますが、これらはまた後述します。さて、下の2つの演算は何だろうと思った人もいるのではないでしょうか？　これは実は2つとも除算を表しています。

```
(a/b)*b = a
a*(a\b) = b
```

となるような値をそれぞれとる演算になります。a/bはaの右からbの逆行列を掛けるという意味で、a\bのタイプの除算はaの逆行列にbを掛けるという意味になります。ということは、「Ax=b」という連立一次方程式を考えると、両辺に左からAの逆行列を掛け、「A\b=x」となって方程式の解xが計算できることになります。例えば、

```
a = [1 2 ; 3 4]
b = [1 ; 0]
```

という指定のあと、「a\b」と入力すると、

```
ans =

  -2.0000
   1.5000
```

となります。これは、

```
x0+2*x1 = 1
3*x0+4*x1 = 0
```

の解になっていることはx0に-2、x1に1.5を代入すれば確認できます。

さて、べき乗の計算も行列に対して行うことができます。「a = [1 2 ; 3 4]」という指定のあと、「a^2」と入力すると、

```
ans =

   7  10
  15  22
```

と答えが返ってきます。面白いのは「2^a」という計算さえできるところです。この場合に答えは次のようになります。

```
ans =

  10.483  14.152
  21.228  31.711
```

これは何の役に立つかのでしょうか。

eのべき乗を使うと、微分方程式の解が簡単に求まるのです。例えば単純な微分方程式を考えます。

「dx/dt = ax」の解は「Ke^(at)」という形をしていることが知られています。xがベクトルでaが行列でも同じです。すると「e^a」が計算できれば簡単に解が得られることになります。octaveでは、eは

```
e = 2.7183
```

と定義されていますから、先ほどのaを用いて、「e^a」と入力すると、

表1　octaveで試してみる演算

演算式	意味
a+b	加算
a-b	減算
a*b	乗算
a^bまたは a**b	べき乗（aまたはbの少なくとも一方はスカラとする）
a'	共役転置（スカラなら共役のみ）
a/b	右からbの逆数もしくは逆行列を掛ける
a\b	左からaの逆数もしくは逆行列を掛ける

```
ans =

   51.969   74.737
  112.105  164.074
```

となります。これで、後は境界条件を入れて係数を整えれば、微分方程式の解が得られることになります。実はoctaveには、このほかに微分方程式を解く関数が備わっていますが、その話は関数の話をした後にしたいと思います。

●微分方程式を解く

それでは、試しにこの機能を使って線形の微分方程式を解いてみます。例として、単純な一階の微分方程式を、抵抗とインダクタンスと直流電源の直列回路で考えてみましょう。

```
E = L(dI/dt) + RI
```

これを「E = 100[V]」、「R = 100[Ω]」、「L = 100[H]」、Iの初期値を「0[A]」とします。まず、式を少し整理すると

```
(dI/dt) = -(R/L)I + E/L = -I + 1
```

となります。定数項が付いたので、解の形は「I = Ke^(at) + C」のように、定数が付いているはずです。そこで、この式に「a = -1」を入れて「t = 0[s]」のときに「I = 0[A]」とすると「K = - C」となります。次に、無限時間経ったときに定常状態になることを考えると「I = E/R = C = 1」と計算できます。そこで、方程式は

```
I = -e.^(-t) + 1
```

と解けることになります。ここで「.^」という演算を新たに使用しましたが、これは要素ごとにべき乗を取る演算子です。係数行列のaは行列としてべき乗を取るのですが、時間tはスカラなので、要素ごとにかかるのです。ここでは係数行列も定数なのでまとめてスカラとして扱っています。それではこの解を計算してみましょう。回路の時定数は1なのでその5倍くらいのところまで時間をとれば特徴はつかめるので

```
t = 0:0.1:4.9 ;
```

とします。最後のセミコロン「;」は結果を画面に出さないためのおまじないですから、あまり気にしないでください。で、電流の計算ですが、

```
I = -e.^(-t) + 1 ;
```

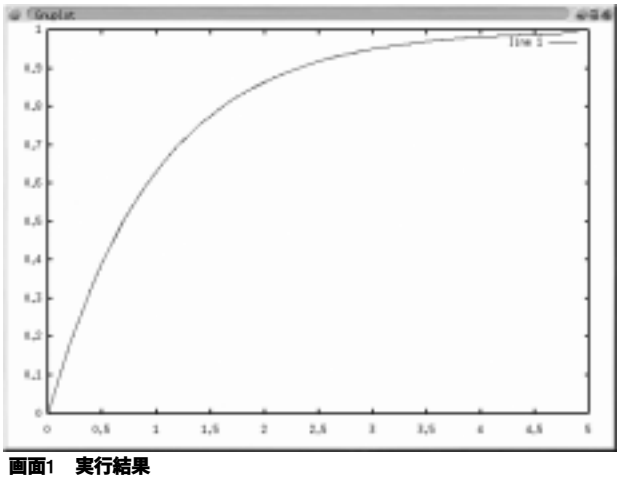
とします。こちらも先ほどの式の最後にセミコロンを付けただけになっています。tは行ベクトルになっているのでこのIはtの各要素ごとに計算した行ベクトルとして格納されることになります。計算はこれでできたのですが、結果を見ないと安心できませんよね。そこで、

```
plot(t,I)
```

と打つと見事グラフが表示されることになります(画面1)。簡単ですから皆さんやって見てください。

さて、HPCプログラミング入門の第1回はここまでにします。この範囲でも、実験レポートの計算や、線形代数の宿題などに結構役に立つと思います。octaveにCompaqのライブラリをリンクして構築すると、簡単なだけでなく、かなり高速な計算ができるようになります。

基本はインタープリタなので、あんまり速くなさそうな気がすると思いますが、行列の計算はライブラリを直接呼び出すため、ライブラリの処理が多ければ、octaveだけでも相当高速なプログラムが可能になります。octaveでアルゴリズムのデバッグを十分にやったあとFORTRANに移植するなど、プロトタイプ作成にも役に立つでしょう。



画面1　実行結果

R E S O U R C E

[1] General Motors Corporation. EV1 ELECTRIC
http://www.gmev.com/