Linux Japan Vol.7への掲載原稿です.
26char/46line
#10203040506070809101112131415161718192021222324252627282930

「Linux/Alphaによる数値計算ならびにRISCアーキテクチャのおはなし(第二回)」

-- 21164Aの高性能内部アーキテクチャ

清水尚彦 nshimizu@et.u-tokai.ac.jp

月日のたつのは早いもので，隔月とはいえあっという間に〆切が近付いてきます．第一回はいかがでしたか? 全然内容がなかったじゃないかという声も聞こえてきそうですが，とりあえず第二回を書かせていただきます．前回インテルへの製造工場の売却という話をしましたが，Alphaの製造は韓国のSAMSUNGという会社もやっているのです．インテルの工場が軌道に載るのが遅れてもSAMSUNGの工場はかなり進んでいたので大丈夫と思っていたら韓国の国自体が大変なことになってしまいました．おかげでメモリのスポット品は大幅に安くなってメモリを増設した人も多いのではないでしょうか? 対岸の火事なんて思っていると明日は我が身ということにもなりかねません．日本は外貨準備が多いので投機売りで暴落することはないでしょうが，円が安くなると輸入品が高くなって嬉しくありません．たまたま韓国のウォンが円より大きく下がったために韓国製の物は高くはならないことになりましたが，日本はプロセッサなどの付加価値の高い半導体はほとんど輸入に頼っているので円安になるとインパクトが大きいですね．なんてことを書いていると，コンパックがDECを買収するという話が飛び込んで来ました．コンパックはタンデムを買収してトランザクション処理の分野における販売チャネルを確保したのでそれで満足していると思っていたらDEC を買収するという方法でシステムインテグレーションとトランザクション分野の市場に本格参入するつもりのようです．報道ではAlphaに対する投資を続けると言っているようですが，詳しい情報はまだ入手していないのですがAlphaのプロセッサにとって吉と出るか凶と出るかはまだ分かりません．コンパックにとってはPCは過当競争で利益率が低くなりがちなのでサーバー部門で技術と販売網を持つDECを傘下に収めるのはたいへん有利な取り引きだと思います．Digital UNIXを使った大型のサーバーからNTのPCサーバーまでコンパックの品揃えはIBM並にフルラインアップとなりますね．

さて，前回はAlphaの命令セットアーキテクチャ上の工夫を書きましたが，プロセッサは命令セットが決まれば工夫の余地はないのでしょうか? これが大違いなのですね．インテルのプロセッサの歴史を知っている方ならご存知だと思いますが，３８６，４８６，ペンティアムとプロセッサの世代はどんどん変わっても命令は若干の追加があっただけで基本は全く変わっていません．また，ＩＢＭの大型コンピュータのアーキテクチャであるＥＳＡ／３９０も６０年代に作られたシステム３６０のころからほとんど変わっていません．しかしこれらのプロセッサは世代毎に大きく性能を向上させてきた上に内部の構造もどんどん変わってきました．Alphaプロセッサも最初の世代である21064と第二世代のプロセッサ21164では全く内部の構造が変わっていますし，次の世代の21264も発表になっていますが，またまた内部構造を完全に変えてきています．ところで，みなさんが今Alphaチップの載ったコンピュータを購入する場合にはほとんどの人が21164Aというプロセッサが載ったマシンを入手されることと思います．最近，新しく廉価版の21164PCというプロセッサも出回り始めましたがまだ数も少なくこの原稿を書いている時点では廉価版というわりには値段も下がっていません．そこでおそらく普及にはしばらく時間がかかるものと思われます．今回は21164Aについてその内部構造と性能の関係を分かりやすく説明したいと思います．また，前回ちょっとだけ触りを紹介したOctaveの使い方をもう少し紹介したいと思います．

1. 計算機の性能

この雑誌の読者の間ではいろいろな機会に計算機の性能の話が出ることも多いと思います．性能の話をするときにはあの計算機は性能が良いとかこっちは悪いとかさまざまな見方から議論されます．では一体計算機の性能って何でしょうか? 計算機と名の付く教科書を書店で片っ端から見てみれば性能についていろいろな議論がされているのが分かると思います．また，簡単に思い付くだけでもいろいろな性能に関連した数値があります．簡単に上げて見ましょう．

1) クロック周波数

最近の多くのプロセッサは1クロック当たり複数命令を実行できるスーパースケーラと呼ばれる構造をしています．それでもクロック周波数が性能に与える影響は大変大きなものである事には変わりありません．しかしクロック周波数が性能そのものを表していると思っている人はいませんか? 例えばAlphaを使った計算機は625MHzもの高速クロックのマシンが出荷されています．これは200MHzのペンティアムに比べてどのくらい早いのでしょうか? また，200MHzのペンティアムプロとペンティアムの性能は同じでしょうか? 詳しく書いているとコンピュータの教科書のようになってしまいますのでこのあたりにしておきますが，興味がある方は参考文献に上げた書籍を見てください．

2) MIPS値

MIPSという値は簡単な性能指標として良く使われて来ました．これは一秒間に何百命令を実行できるかという数値として定義されます．ところが，アーキテクチャが違うと命令数を平等に比較できないのでMIPSの値そのものを一般的な性能指標として使うことはあまりないようです．大型汎用計算機などの数十年もアーキテクチャがあまり変わっていないものでは適当な命令の割合を決めてMIPSを算出した命令ミックスという数字を元に性能を比較することもされています．

3) MFLOPS値

MFLOPSは数値計算の分野では比較的良く使われます．これは一秒間に浮動小数点演算を何百万命令実行したかを数値として表したものです．MIPSと同じように浮動小数点演算もアーキテクチャによって若干命令の品揃えが違っていたり命令毎の性能も大きく違ったりするので違うプログラム同士を比較する時にこの値を使うのは危険です．4) SPECmark値

SPECという団体がベンチマークプログラムを指定してその実行時間を元に相対的な性能を値として出すものです．SPECint95とSPECfp95が代表的な値としてワークステーションのカタログにはほぼ例外なく記載されていると思います．

お話だけならどの数字を使おうと構わないのですが，自分でお金を出して買う場合にはコストパフォーマンスの良い物が欲しくなります．そこで，これらの数字の大小が気になりますね．自分が使う分野がはっきりしている人はその分野に近いプログラムを使ったベンチマークを探すことから始めるといいでしょう．また，大事なことですが，データの規模によって性能は大きく変動する可能性があります．分野が同じプログラムがあったとしても自分が使うデータの規模とベンチマークがあっていないと性能の指標にはなりません．ですから，一番良いのは自分が使おうとしているプログラムをそのままベンチマークとして使ってしまうことです．その時気にしなくてはならないものは実行時間です．MIPS値がいくら高くても実行時間が長ければ意味がありません．自分のプログラムの実行時間がどれだけ早い

かがその計算機が自分にとって役に立つかどうかを決めます．

実行時間が大事というのは感覚的にも納得できると思いますが，実行時間を短くするためにはコンピュータはどんな工夫をしたらよいのでしょう? 先程のクロックの話を振り返ってみましょう．100MHz のペンティアムを 200MHz に替えたら性能が 2 倍になると期待するなという方が無理がありますよね．でも残念ながらそうはなりません．どうしてクロック周波数と性能は比例しないのでしょう? プログラムによっていろいろな違いはありますが，数値計算を主とする場合には一番大きな問題は主記憶(すなわち DRAM)の性能がプロセッサの性能に全然追い付いていないところにあります．例えば 500MHz の Alpha プロセッサを動かすことを考えてみましょう．一命令当たり 2nS という時間で動作します．nS というのは 10 のマイナス 9 乗秒のことです．これに対して DRAM は LSI のピンの所で測定してベストケースで 60nS くらいの時間が立たないとデータを出してきません．そこで，ボードに組み込んで安全率を考えて設計するとプロセッサからはおそらく 200nS 程度の時間を見込まないとデータが到着しないことになると思います．この 100 倍のスピードの差が大きな問題になるのです．

通常スピードの差を埋めるためにキャッシュメモリが使われます．キャッシュは全部のデータを持っているわけではないので命令実行中に何%かの割合でキャッシュにデータがないことが起こります．これをキャッシュミスと呼びます．例えば命令の 1%がキャッシュミスを起こすとすると命令の実行時間は平均的に 2nS だけ増加することになりますね．これはプロセッサのクロック周波数とは関係なくかかるのでクロックの早いプロセッサには相対的に大きなオーバーヘッドになってしまいます．ですから，クロックが早くなればなるほど命令の実行時間を短くするのは難しくなります．

2．21164 内部アーキテクチャにおける工夫(その 1)

第一世代の Alpha はクロック周波数において当時の水準をはるかに超えてデビューしたことを前回書きましたが，出る杭は打たれるではありませんが，風当たりはものすごく強くいろいろな点で批判を受けていました．その多くはクロック周波数程の性能が出ていないというところに集中したような感じがします．これはある程度仕方がないところがあります．同じ技術を使ってクロック周波数を倍にしているのですから何かしら工夫をせざるを得ないのでそれをさか手に取れば性能のでない例題なんていくらでも作れるのです．DEC は第二世代のプロセッサでは初代で批判された部分を修正しながらさらにクロック周波数を上げる努力をしてきました．クロック周波数に拘る DEC の設計チームは同じ世代の他社のチップとは少し異なるスタンスで設計を進めた結果，21164 はユニークな特徴を持ったプロセッサになりました．命令セットアーキテクチャとしての特徴は前回簡単にまとめたので今回からプロセッサ内部の工夫について概説してみたいと思います．細かな条件を並び立てても読者の役には立たないと思いますので，プログラムを作る時に知っていると高速化できる特徴に限ります．これらの特徴のうち幾つかはアセンブラでプログラムしないと役に立たないものもありますが，一部は FORTRAN や C でプログラムをする時にも知っていればずいぶん性能向上の役に立つはずです．その 1 はパイプライン構造と 2 次キャッシュについてです．

・パイプライン

パイプラインの細かな制御はコンパイラではできません．少しでも性能を出すためにアセンブラも視野に入れて考える人はパイプラインを良く理解してください．

21164 では命令を 16 バイト毎メモリから取って来てそこに含まれる 4 命令の中で同時に実行できるものを選んで実行します．そのため，命令をどのように並べるかによってパイプラインの使い方が変わってしまうことになります．コンパイラはこのパイプラインを良く理解して命令を整列させる必要があるのですが，gcc ではまだそこまではやっていません．最近，gcc の命令スケジュールをもっと高率的にする為に egcs というプロジェクトが活動を活発にしています．egcs を使うことによってメモリの境界の制御はともかく命令の並び方はかなり洗練されるようです．

21164 では整数パイプラインが 2 本と加算と乗算の浮動小数点パイプラインがあり最大 4 個の命令が 1 クロックサイクルで実行できます．整数パイプラインは E0 と E1 と呼ばれています．浮動小数点パイプラインは加算が FA，乗算が FM と呼ばれます．命令はその種類に応じてどのパイプラインを使うかが決まっています．

ここで特徴的なものとして UNOP と浮動小数点コピー命令があります．UNOP は何もしない命令なのですが，前に書いたように命令の並べ方がまずいとパイプラインの関係で性能が悪くなる可能性がある場合に命令の並べ方を直す必要があるのです．その時に演算は何もしないけどどのパイプラインでも流すことのできる UNOP は大変有用です．また，浮動小数点のコピー命令は本来は演算の役には立たない訳です．ですが，レジスタの使い方を考えるとどうしてもコピーをしたくなるような時があります．そのときにコピーによって性能が落ちるくらいなら並べ方をもう一度再検討するべきです．しかし Alpha では乗算か加算のパイプラインのどちらでもコピー命令が動作するようになっているのでどちらか一方でも空いているのであればコピー命令を使うことによる性能の低下は避けられます．

以下各パイプライン毎に流せる命令の種類をリストアップしてみます．

整数パイプ E0 に限定されているもの:

 整数乗算命令，ストア命令，シフト命令，性能測定用の RPCC などの特殊命令．

整数パイプ E1 に限定されているもの:

 整数系の条件分岐やサブルーチン呼出．

整数パイプ E0，E1 どちらでも動作するもの:

 整数加減算，論理演算，条件転送，比較，ロード命令．ただし，E1 に入ったロード命令はキャッシュミスがある場合，1 クロック余分に実行時間がかかります．

浮動小数点パイプ FA に限定されているもの:

 浮動小数点条件分岐，浮動小数点加算，浮動小数点除算．

浮動小数点パイプ FM に限定されているもの:

 浮動小数点乗算．

浮動小数点パイプ FA,FM どちらでも動作するもの:

 浮動小数点コピー命令．

どのパイプラインでも動作するもの:

 UNOP 命令

パイプラインに流れる命令の結果を使う時には数クロック

サイクル待たなければいけない場合があります．浮動小数点演算では加減算や乗算では4サイクル後の命令が演算結果を使うことができますが，それ以前に使おうとすると命令の実行が止められます．使えるようになるまでのサイクル数を命令レイテンシと呼び，実行の停止をストールと呼びます．ストールがなるべくないように命令を並べることが高性能への道になります．(もちろん，アルゴリズムを工夫して本質的に命令数を低減する方がもっと性能が出るようになることも多いので，最初はパイプラインをそれほど意識する必要はないと思います.）命令によってはレイテンシが変動するものもあります．その代表例がメモリの読み出しを行うロード命令です．浮動小数点の除算や整数かけ算などもレイテンシの変動はあるのですが，ロード命令の変動に比べたら全然問題になりません．そこで，プログラムでストールを少なくするように命令を並べる際，ロード命令の扱いにもっとも神経を使う必要があるのです．

・2次キャッシュ

DECの21164発表時トランジスタの数があまりに多かったので大変驚いたのですが，DECは96KBものメモリを2次キャッシュに与えてしかもプロセッサ上に載せてしまいました．この2次キャッシュには賛否両論の意見が飛び交っていましたが，批判グループの意見は96KBでは2次キャッシュとして小さすぎて役に立たないというものでした．

しかし，良く見ると21164の2次キャッシュはうまい設計がなされています．従来のプロセッサの2次キャッシュとは異なる使い方によって数値計算では大きな性能向上の可能性を持つ優れたシステムだと思います．その大きな特徴として21164の2次キャッシュはパイプラインアクセスができるようになっています．これは一つの1次キャッシュミスによって起こされた転送要求が終了する前に次のキャッシュミスの転送要求を受け付けるものです．ですから，うまくプログラムを作ると96KB全部を高速なキャッシュとして命令の実行停止をさせずに動作させることができます．前回紹介した後藤さんが開発されメイリングリストで公開されているdgemm互換サブルーチンはこの特徴をうまく使っているため1000x1000の大きな行列を対象として500MFLOPSを超える実効性能を計測しています．

普通1次キャッシュがミスすると2次キャッシュからのデータを待つ間キャッシュミスを起こした後続の命令(もしくは次のキャッシュミスを起こした命令)は停止させられます．一方，最近のアウトオブオーダー実行ができるプロセッサでは実行できる命令を先に先行させてミスを起こした命令の停止の影響を最小限にしようとします．ところが21164ではアウトオブオーダーは実装されていないとされています．だとするとミスが起きると待つしかないのでしょうか?実は21164はロード命令に限ってアウトオブオーダー実行ができるようにミスアドレスファイルに実行が保留されたロード命令を記憶しておくことができるのです．

ミスアドレスファイルとライトバッファをたっぷり用意したことによってロードとストアの命令実行順序を実行時に動的に変更できるので，2次キャッシュのパイプライン構造の特徴がさらに活かされます．

ミスアドレスファイルとライトバッファの詳しい話は次回にします．

3．Alphaアーキテクチャと数値計算

一般にRISCプロセッサで高速な演算を行う場合にはキャッシュメモリを考慮して演算を分割するブロック化と呼ぶ手法を使います．が，今回は紙面の都合上ブロック化の手法の解説はしません．今回は小さな分かりやすい例題でAlpha

アーキテクチャとプログラムの関係を示したいと思います.

内積演算の例題

ベクトルの内積は数値計算では良く使われるものです. 例えば行列とベクトルの積などもよくよく見ると内積の固まりです. その演算は次のようなものです.

ベクトルX={x1,x2,x3,...,xn}, ベクトルY={y1,y2,y3,...,yn}の内積zは

z=x1*y1+x2*y2+...+xn*yn

となります. これをFORTRANの関数(名前をdiprdとします)で書くと

```
      function diprd(n, dx, dy)
      integer i,n
      real*8 diprd, dx(n), dy(n), z
      z=0.0
      do i=1,n
         z=z+dx(i)*dy(i)
      enddo
      diprd = z
      return
      end
```

となります. これを素直に解釈すると中間的な変数zはレジスタに置くとして, ベクトルxとyは毎回メモリから読み出して計算します. コンパイル後のコードは下記のようになります. このコンパイルにはg77というFORTRANコンパイラとegcsという命令スケジューラを使ってオプションとして "-O2"を指定しています. これを少しずつ読んで見ましょう. ピリオドから始まっている行は疑似命令ですからこれを除いて考えます. 整数の汎用レジスタは$0,$1,...,$31まであり, 浮動小数点のレジスタは$f0,$f1,...,$f31までになります. $31と$f31は常に0が入っています.

```
diprd_:
diprd_..ng:
        .frame $30,0,$26,0
        .prologue 0
```

関数の始まりの疑似命令と関数名のラベルです.

```
        ldl $1,0($16)
```

まず, Alphaの関数呼び出しでは引数は16番目の汎用レジスタ($16)から順番に格納される事になっています. FORTRANの引数は名前呼び出しなのでレジスタには引数の格納されるアドレスが書かれる事になります. そこで, 最初のロード命令 ldl は引数nを1番の汎用レジスタに読み出します.

```
        cpys $f31,$f31,$f0
```

cpys命令はパイプラインの所で説明したレジスタコピーの命令です. Alphaでいつも0が入っている$f31をコピーすることで $f0のレジスタを0にセットします. subl $1,1,$1
```
        blt $1,$35
```

$1のレジスタから1を引いて負になったら終了のラベルに分岐します.

```
        .align 5
$37:
        ldt $f1,0($17)
        ldt $f10,0($18)
```

2番目と3番目の引数のアドレスから浮動小数点の数値を一つずつ取り出します.

```
        addq $18,8,$18
        addq $17,8,$17
```

アドレスを格納しているレジスタに8(浮動小数点数の大きさ)をそれぞれ加えます.

```
        mult $f1,$f10,$f1
```

取り出したそれぞれの要素同士を掛け合わせて, 一旦$f1のレジスタに入れます.

```
        subl $1,1,$1
```

要素数を記憶している$1のレジスタから1を引きます.

```
        addt $f0,$f1,$f0
```

掛けた結果を変数z($f0)に格納します.

```
        bge $1,$37
```

要素がまだ残っているならループの最初に戻ります.

```
$35:
        ret $31,($26),1
```

関数を終了して戻ります. 戻り値は$f0に書く決まりになっているので自動的にzが返されます.

```
        .end diprd_
```

関数の終りを示す疑似命令です.

当り前ですがこのようにFORTRANプログラムとコンパイル結果は良く対応しています. このプログラムを動作させてどのくらいの性能が期待できるでしょうか? DOループの中がもっとも実行回数が多くなるので, この部分だけを取り出して考えてみましょう. ここで, ロード命令はレイテンシが変動するのですが, 今はレイテンシの変動なしの2サイクルで終了するとします. 21164は4命令ずつメモリから取り出して実行すると前に書きました. そこで, ループの中の先頭の4命令を見てみます.

まず, ロード命令が2つと整数加算命令が2つならんでいます. これらの命令は全てE0かE1の整数パイプラインで動作する命令なので2本しかないパイプラインでは2つの命令しか動作しません. そこで, 最初の2命令であるロード命令が実行されます. 次のクロックサイクルで残りの2命令を実行することになります. 次の4命令は浮動小数点の乗算と整数減算, 浮動小数点加算, 分岐命令になっています.

さて, ロード命令はレイテンシ2サイクルだったわけですが, ここで都合良く(?)加算命令が1サイクル間に入ってくれたおかげで次の浮動小数点乗算は遅れなく実行できます. また, 同時に読み出していた整数減算も減算を妨げるものはないので一緒に実行されます.

ところが, その次の浮動小数点加算は乗算の結果を使っています. そこで, この実行は乗算の実行が終了するまで4サイクル待たされることになってしまいます. 分岐命令は整数加算の結果を使うのですが, 浮動小数点加算の結果を待っているうちに整数加算の結果は出て来るので, 待ちは発生しません. そこで, 浮動小数点加算と同時に実行されることになります.

この話で重要なのはレイテンシを待っている間に何も命令が実行できない結果，本来のハードウェアの性能が出ないという点です．この為の対策として待っている間にやるべき命令を作って上げる方法がループアンローリングと呼ばれています．これは前出のDOループを展開して複数のループの命令を並べることによってレイテンシ待ちの空き時間に別のループの命令を実行するものです．

g77では前出のオプションに加えて"-funroll-loops"というオプションを付けることでループアンローリングができます．ループのところだけ抜き出してみると次のようになっているのが分かると思います．実はFORTRANソースコードでこれと同じことをやってもいいのですが，コンパイラでやった方がソースコードの保守が簡単になります．

```
$37:
        ldt $f14,0($17)
        ldt $f1,0($18)
        ldt $f13,8($17)
        ldt $f10,8($18)
        mult $f14,$f1,$f14
        ldt $f11,16($17)
        ldt $f12,16($18)
        mult $f13,$f10,$f13
        ldt $f1,24($17)
        subl $2,4,$2
        mult $f11,$f12,$f11
        ldt $f10,24($18)
        addq $17,32,$17
        addq $18,32,$18
        addt $f0,$f14,$f0
        mult $f1,$f10,$f1
        addt $f0,$f13,$f0
        addt $f0,$f11,$f0
        addt $f0,$f1,$f0
        bge $2,$37
```

先程のコードと比較すると今回は4回のループを展開して一つのループとしていることが分かると思います．
先程問題にした乗算と加算の間のレイテンシですが，
今回は間に他の命令がたくさん入っていて待ち時間が減少しています．しかし，最後の浮動小数点加算の部分を見るとこの部分は直前の命令の結果を使う命令が集中していてレイテンシが問題になりそうな感じが分かると思います．
実はこのような時にも別の手立て（ソフトウェアパイプラインや変数の変換)を使うことでレイテンシによる性能低下を防ぐ手立てがあります．これらの話は次回したいと思います．

4．Octaveの話

実は今回詳しい話をしようと思っていたらAlphaの話を書いているうちに紙面がつきてしまいました．ということで，今回も簡単に例題の紹介にとどめさせて頂きます．
前回，行列計算が簡単にできることを紹介しましたが，Octaveにはまだまだ優れた機能がたくさんあります．
今回紹介するのは常微分方程式の解を求めるものです．

一般に常微分方程式を変形すると次のような標準系に変換できます．

```
(dx/dt) = f(x,t)
```

ここで，xはベクトルであっても構いません．
例えば調和振動子の運動方程式

```
d(dx/dt)/dt + p*p*x = 0
```

を考えて見ましょう．

このままだと標準系とちょっとずれているので簡単な変換をします.

x1 = x

とおいて

x2 = d(x1)/dt

とすると運動方程式は

d(x2)/dt + p*p*x1 = 0

となります. これを並べると

d(x1)/dt = x2
d(x2)/dt = - p*p*x1

と標準系になったことが分かります.

それではOctaveでこの方程式を解いてみましょう.
常微分方程式の解法にはlsodeと呼ばれる関数を用いますが,
始めに標準系の方程式を関数として定義する必要があります.
例題なのでpは適当に 1.0 としましょう.

```
function xdot = f(x, t)
 p = 1.0;
 xdot(1) = x(2);
 xdot(2) = -x(1);
 endfunction
```

ここで, セミコロンは画面表示を抑えるための指示で本質ではありませんが,
画面に延々と評価値が表示されても見にくいので忘れずに入れておきましょ
う. さて, これを解析するわけですが

```
 x = lsode("f", [1; 2], (t = linspace(0, 50, 200)'));
```

と打ち込んで見てください. この関数は3つの引数を取ります.
"f"は関数名を表しています. 次のベクトルはxの初期値を意味します.
3つめは時間のベクトルを与えますが, ここでは0から50までを等間隔で200個
の値を与えるためにlinspaceという関数を使っています.
linspaceという関数は最初の2つの数値の間を3つめの数字の数だけ
補完したベクトルを返す関数です. これは横方向のベクトルになりますので
lsodeで要求される方向に転置してtに代入し渡します.
何も表示せずに終了したでしょう? これだけでは結果が正しいかどうか
分かりませんね. では, プロットコマンドで表示して見ましょう.

```
plot(t,x)
```

どうですか? 見事に調和振動子が振動している様子が表示されたでしょう.
このように簡単に微分方程式も解けてしまうのです.

今回のOctaveの話はここまでです. まだまだ便利な機能がたくさんあります
ので段々紹介して行きます.

5. おわりに

前回私はFORTRANを使うユーザーはStataboware を勧めますと
書きましたが, StataboweareはFTPか有志のボランティアによる
CDROMでしか配布されていなかったのでなかなか入手は難しいのが
難点でした. 今回間に合えばこの号のCDROMに入れていただく
ように編集部にお願いしておきました.(入ったのでしょうか> 編集部殿)

最近のいくつかの雑誌の広告によるとDECの代理店が
Stataboware を搭載したマシンを販売しているようです.
インストールの苦労をしたくない人にはこういったものも
便利に使えると思います.

どうも数値計算はビジネスとしてのうまみに欠けるらしくクレイを買収した
SGIにしてもあまり業績がぱっとしません.
ですからCOMPAQの買収によって財政的な基盤が強化されてAlphaの
開発にはずみがつくといいと思っています.
新しいプロセッサ21264のデビューもそう遠くないので数値計算に
興味がある人達にとってはまだまだAlphaは楽しめるアーキテクチャであり続
けることと思います.

6. 参考文献

1) パターソン&ヘネシー「コンピュータの構成と設計(上)」日経BP社

2) John W. Eaton「Octave」(パッケージに含まれるマニュアル)

3) Digital「Alpha 21164 Microprocessor Hardware Reference Manual」

4) Digital「Alpha AXP Architecture Reference Manual」

5) http://www.cygnus.com/egcs/