Linux Japan 9月号への掲載原稿です.
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(第四回)」
-- 性能解析とOctaveの入門の入門

清水尚彦 nshimizu@et.u-tokai.ac.jp

1. はじめに

最近はどこの発表を見てもパワーポイントを使うものが増えてきています。私自身はパワーポイントを使うほどのプレゼンテーションはあまりないのですが、それでも新しい装置のプレゼンテーションなどにはお話とOHPよりも実際にデモを見せた方が分かりやすいですね。ということで、手持ちでサブノートに近い大きさのCOMPAQ AEROをプレゼンテーションマシンに仕立て上げることにしました。まずはVGAを出力する必要がありますが、AEROの背面にはVGAのコネクタはありません。ポートエクスパンダの入手は検索エンジンを使って通信販売で行いました。 fjの個人売買で入手した500MBのHDDをオリジナルの84MBから換装してDOSのパーティションとLinuxのパーティションを組み込み、magic pointを載せればプレゼンテーションマシンの完成となります。Alphaで簡単に持ち運べるマシンはないので、当面はこいつが主力プレゼンマシンとなります。AlphaのLinuxでもう一つ欠けているのがNetscapeをはじめとするブラウザです。が、Netscapeのソースが入手できるようになった今私は案外早く出現するだろうと楽観しています。(お前がやれって?)私の仕事場ではLinux/Alphaのマシンのブラウザはxonで Digital UNIXのマシンのものを使うことで対応しています。Digital UNIX用のバイナリは適切なシェアードライブラリを配置するとLinux/Alphaでも動作するそうですが、ライブラリのライセンスの問題がありますので一般的な方法ではありませんね。Digital UNIXをサイトライセンスで契約しているところではこの方法がよいと思います。Linux/AlphaでDigital UNIX互換シェアードライブラリが作れれば状況は随分変わるかもしれません。ブラウザに関して実現できそうな方法を上げてみます。

1. chimeraなどのフリーなブラウザを移植
2. Digital UNIXでスタティックリンクしたNetscape communicatorを構築
3. LesstifでNetscape communicatorを構築
4. Digital UNIX互換シェアードライブラリを作りDigital UNIX用のNetscapeを使用

時間とメモリとディスクがたっぷりあったら3にトライするのが面白いと思います。メイリングリストでも話題に上がりはじめたので解決するのも時間の問題だと思います。

新しいAlphaのプロセッサである21264は韓国の半導体メーカーであるSAMSUNGでも作っています。

http://www.samsungsemi.com/

5月にSAMSUNGが21264のサンプル出荷をはじめたので、この雑誌が出ることにはすでに新しいマシンを触ったことのある人もいるかもしれません。SAMSUNGのプレスリリースではクロック周波数については何も書いてなく、X2.0からX3.5までの数種類のチップが同時に発売されるとなっています。手元の情報を突き合わせるとおそらくX3.0は600MHzだと思うので、X3.5は700MHzのチップということになるのでしょうか? 現物をみるのが楽しみですね。前回ミスアドレスファイルがロード命令をアウトオブオーダーで実行する仕組みだと書きましたが、21264ではロード命令に限定せずにアウトオブオーダーで実行できる仕組みを導入したのでコンパイラが少々貧弱でも高い性能が見込めます。とはいっても、命令の並び替えのレベルで得られる並列性なんてたかが知れているというのが私の持論で、本当に性能を出そうと思えばレイテンシの長いメモリアクセスを隠すようにプログラムを工夫する必要があるし、アルゴリズムも非常に大切です。アウトオブオーダーが魔法の手法のように思っている方は21264が出ても性能的に大したことがないなぁと感じるかもしれませんね。ところで、数値計算のためのプロセッサとしてはどのくらい嬉しい機能があるのでしょうか。プロセッサの概要はすでに発表されているので近いうちに整理してみたいと思います。この原稿を書いている時点ではまだハードウェアリファレンスマニュアルが出ていないようなので、詳細機能についてはこれが発行されてから調べたいと思います。概要が知られているだけでもロード命令やストア命令の実行保留可能な数が増えていたりするので今までの最適化手法の流れを大きく変えるものではないと思います。

プレスリリースに出ているプロセッサの性能と第二四半期の価格は次の通りです。

製品名	概算性能(SPECint/fp95)	価格
KP21264-2.0X	28/42	$2,500
KP21264-2.5X	33/50	$3,500
KP21264-3.0X	38/58	$4,500
KP21264-3.5X	43/64	$6,500

性能は素晴らしいのですが、プロセッサ単体でもまだまだ値段が高いですね。一方、安価版として出発した21164PCはそれなりに値段もこなれてきたようです。21164自体の価格も下がっているのでなかなかお買い徳感がでるところまではいきませんが、21164PCのユーザーも段々と増えてきたようなのでこのプロセッサについてもそのうちに整理して紹介したいと思っています。

Vol.6の記事で後藤さんからUNOPはどのパイプラインでも流れるのではなくパイプラインの前で捨てられるというコメントをもらいました。訂正致します。

2. 21164の内部アーキテクチャ(その3)

さて、21164を使いこなすために知っておくと良いことはまだまだたくさんありますが新しいチップも次々と出て来るし、主要な特性の多くを書いたので内部アーキテクチャは今回で一段落とします。 21164の内部アーキテクチャの話は聞きあきたという人は読み飛ばして構いません。

今回はライトバッファに付いての補足とアドレス変換の話、外部キャッシュの話を書きます。

1) ライトバッファに付いての補足
前回簡単にライトバッファに付いて書きましたが、書き忘れていたことがあったので簡単に補足します。ライトバッファがメモリへの書き込みデータを保持すると書きましたが、ここに保持されたデータは64クロック毎にメモリに書き込みに行きます。この書き込み要求はたとえバッファに空きがあっても発生します。設計者が64クロックに設定していて変更できないのはちょっと残念ですがこのくらいのクロック待てば十分だとみなしているのでしょうね。これ以外にもロック付のロード命令が発行されても書き込み要求がでます。

2) アドレス変換
仮想記憶の元々の出処はメモリが高価な時代に安く大きな記憶領域を確保するところにあったと思いますが、現在のようにメモリが安くなってくると必要なだけメモリを積むことも可能です。しかし、いくらメモリが安くても他のプロセスのデータを簡単に書き換えることができたり、ユーザーがシステムの情報に勝手にアクセスしたりできる場合には安定した動作は望めません。そこで、Linuxをはじめとする普通のOSはプロセス毎の資源を保護するために仮想記憶を使います。プロセス毎別々のアドレス空間を与えることで不用意なプログラムが他のプログラムを不安定にする危険を減らせます。仮想記憶を実現するには仮想アドレスを実アドレスに変換するアドレス変換が必要になります。アドレス変換はメモリ空間をページと呼ぶ単位で分割して仮想アドレスのページ番号と実アドレスのページ番号の対応表であるページテーブルをメモリ上に用意しOSの管理によって変換することによって実現します。メモリ上の変換テーブルを読み込むには時間がかかるため通常変換バッファと呼ぶ小さなキャッシュをプロセッサ上に用意します。21164ではこのバッファは命令用として48個データ用として64個が用意されています。各バッファのエントリは8KBのページの場所を示すのでそのままだとデータには512KBの領域しか指定できないことになります。大規模な数値計算を行う時にこれでは不便なので各バッファエントリに最大512個の連続した8KBページが割り付けられるようになっています。こちらを使えば最大4MBのアドレス空間が一つのバッファエントリで示すことができるのですが、OSが異なる単位のメモリ管理をする必要が出て来るので実際にLinuxでは使えないようです。 512KBといってもあまりピンとこないと思いますが、正方行列に倍精度の浮動小数点を並べるとすると256次元の正方行列が取れます。この大きさを目安にすれば良いでしょう。
キャッシュミスではハードウェアが適切なメモリ領域から自動的にデータを取って来るのですが、アドレス変換のミスは一旦特殊な割り込みが発生してプログラムによって処理がなされます。そこで、常に変換ミスが発生するような状態になる大きさのデータを扱うとかなり性能が落ちてしまいます。一度に扱うデータは512KBです、また、データ量が少なくてもバラバラのアドレスを扱うと64個までしかバッファに入りません。おぼえておきましょう。

3) 外部キャッシュ
21164 では外部キャッシュのインターフェイスにパイプラインアクセスやウェーブパイプラインが使えるようになっています。これらの機能は対応するチップセットがあってはじめて有効なのでマザーボードの種類によってキャッシュメモリの性能に差が出てきます。また、ちょっと変わり種のキャッシュ構成として Victim キャッシュと呼ぶ方式が取れるようになっています。外部キャッシュは大きく取れるとはいってもダイレクトマップで構成されています。そこで、ソフトウェアがたまたまキャッシュの同一のエントリに格納されるべき複数のアドレスを頻繁に使うことが考えられます。その一番の候補は下位のビットが 0 となるような切りのよいアドレスです。このような場合にはソフトが使うたびにキャッシュの入れ換えが発生して大きく性能を落してしまうことになります。Victim キャッシュはこの状況を救うために数エントリの連想度の高いキャッシュ(Victim キャッシュ)を用意してダイレクトマップから追い出されたデータを Victim キャッシュに退避させます。退避したキャッシュは必要な時にすぐに使えるので小容量の Victim キャッシュに格納できる程度の領域の連想度の低さは解消できます。たいへん面白い機構ですが、これもオプショナルな機能なので一般に入手できるマザーボードには付いていない可能性が高いですね。

4) 性能カウンタ
21164 にはアーキテクチャで定義されているプロセッササイクルカウンタ以外に三つの性能カウンタがあってキャッシュミスや変換バッファミス、浮動小数点命令数や分岐予測の予測ミスなどさまざまな性能に関連したパラメータを取得できるようになっています。これらのカウンタはオーバーフロー時に割り込みが発生するようになっているので割り込みハンドラを用意して使えるように環境を整えなくてはなりませんが、たいへん有用な機能です。21164 のマシンを入手したら真っ先に利用環境を整えたい機能です。次に紹介する性能解析ツールのように汎用の解析方法では性能が低下している本当の現場を押えることが難しいのでこのようにハードウェアに組み込んだモニタ機能が有効になるのです。私は 21164 のマシンを持っていないのですが、後藤さんの dgemm ルーチンのチューニングで一つ性能的に納得できていない部分がありますのでマシンがあればすぐにでも dgemm ルーチンのコア部分でどんなことが起こっているのか確かめたいと思っています。

3. 性能解析ツール gprof

さて、今回は性能の詳細な解析をするつもりだったのですが、Octave の話が進んでいないので詳細な解析は後に譲って性能解析ツールの話を書きます。みなさんがプログラムを高速化したいと思った時に大変有用なツールとしてプロファイラというツールがあります。Linux にも gprof というプロファイル表示ツールが入っています。これはサブルーチン毎に実行時間を集計して画面に表示することを可能にするものです。なんでこれが有用なツールになるかというと計算機の世界で有名なアムダールの法則というものがあって全体に占める割合の高い部分を優先して性能の改良をしないと全体性能は上がらないということが分かっているからです。ちなみにアムダールは IBM の System/360 シリーズの設計者で後に独立してアムダールコーポレーションを設立、ソフトウェア互換のプロセッサがビジネスになることを示した人です。さて、それではさっそく前回の LINPACK の例題を元に gprof の使い方を見てみましょう。

3.1 プログラムのコンパイル

コンパイルと言っても gprof をインストールするためのものではありません。これは Stataboware にははじめから入っています。実は性能解析のために対象のプログムが解析のヒントとなる情報を出力するように小さなコードを追加します。そこで、解析前に対象のプログラムを次のように -p オプションをつけてコンパイルする必要があります。

```
et7beta:{11} g77 -O2 -funroll-loops -p lin100.f sec.s -o lin100p
```

3.2 基礎データの収集

次に基礎データを収集するためにこのプログラムを実行します。

```
et7beta:{12} ./lin100p
 PLEASE SEND THE RESULTS OF THIS RUN TO:

 JACK J. DONGARRA
 MATHEMATICS AND COMPUTER SCIENCE DIVISION
```

```
 ARGONNE NATIONAL LABORATORY
 ARGONNE, ILLINOIS 60439

 TELEPHONE: 312-972-7246

 ARPANET: DONGARRA@ANL-MCS

    NORM. RESID      RESID           MACHEP         X(1)         X(N)
 1.67117300E+00  7.41628980E-14  2.22044605E-16  1.00000000E+00  1.00000000E+00


    TIMES ARE REPORTED FOR MATRICES OF ORDER   100
      DGEFA      DGESL      TOTAL     MFLOPS       UNIT      RATIO
 TIMES FOR ARRAY WITH LEADING DIMENSION OF 201
  2.539E-02  9.770E-04  2.637E-02  2.604E+01  7.680E-02  4.709E-01
  2.539E-02  0.000E+00  2.539E-02  2.704E+01  7.395E-02  4.534E-01
  2.539E-02  9.770E-04  2.637E-02  2.604E+01  7.680E-02  4.708E-01
  2.480E-02  8.789E-04  2.568E-02  2.674E+01  7.481E-02  4.586E-01

 TIMES FOR ARRAY WITH LEADING DIMENSION OF 200
  2.344E-02  9.760E-04  2.441E-02  2.813E+01  7.111E-02  4.360E-01
  2.344E-02  9.770E-04  2.441E-02  2.812E+01  7.111E-02  4.360E-01
  2.344E-02  0.000E+00  2.344E-02  2.930E+01  6.826E-02  4.185E-01
  2.314E-02  7.812E-04  2.393E-02  2.870E+01  6.969E-02  4.272E-01
```

前回報告した値よりMFLOPS値が下がっていますね。これは追加したコードの分のオーバヘッドがかかるからです。ですから、タイミングの要求が厳しいプログラムではプロファイルをとることによって挙動が変わってしまうことがあります。観測によって状態が変化して観測が不可能になるなんてまるで量子力学のようですね。

3.3 プロファイルの表示

最後にようやくgprofの出番です。gprofによって収集した基礎データから性能のプロファイルを編集して表示します。

```
Flat profile:

Each sample counts as 0.000976562 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  us/call  us/call  name
 75.99      0.43     0.43   133874     3.21     3.21  daxpy_
 12.26      0.50     0.07       26  2666.77 18742.29  dgefa_
 10.71      0.56     0.06       27  2242.48  2242.48  matgen_
  0.52      0.56     0.00     2574     1.14     1.14  dscal_
  0.35      0.56     0.00     2574     0.76     0.76  idamax_
  0.17      0.57     0.00       26    37.56   676.28  dgesl_
  0.00      0.57     0.00        1     0.00     0.00  dmxpy_
  0.00      0.57     0.00        1     0.00     0.00  epslon_
```

%timeは実行時間に占めるこのサブルーチンの時間の割合です。cumulative secondsは時間の累計を示します。self secondsはサブルーチンの実行時間、callsは呼出回数、self us/callは一回当たりのサブルーチンの実行時間をマイクロ秒で示し、total us/callは下位のサブルーチン呼出を含む実行時間を同じくマイクロ秒単位で示したものです。

gprofはもう一つの形式の画面であるcall graphも出力します。後半の部分を省略していますが、call graphの画面は次のようになります。

```
                     Call graph (explanation follows)


granularity: each sample hit covers 4 byte(s) for 0.17% of 0.57 seconds

index % time    self  children    called     name
                                                 <spontaneous>
```

```
[1]    100.0    0.00    0.57                  MAIN__ [1]
                0.07    0.42      26/26           dgefa_ [2]
                0.06    0.00      27/27           matgen_ [4]
                0.00    0.02      26/26           dgesl_ [5]
                0.00    0.00       1/1            dmxpy_ [8]
                0.00    0.00       1/1            epslon_ [9]
-----------------------------------------------
                0.07    0.42      26/26           MAIN__ [1]
[2]     86.2    0.07    0.42      26            dgefa_ [2]
                0.41    0.00  128700/133874       daxpy_ [3]
                0.00    0.00    2574/2574         dscal_ [6]
                0.00    0.00    2574/2574         idamax_ [7]
-----------------------------------------------
                0.02    0.00    5174/133874       dgesl_ [5]
                0.41    0.00  128700/133874       dgefa_ [2]
[3]     76.0    0.43    0.00  133874            daxpy_ [3]
-----------------------------------------------
```

このcall graphは名前からも分かるようにどのルーチンが何を呼び出しているかもしくはどこのルーチンから呼ばれているかというクロスリファレンスの役目を持っています。単にクロスリファレンスというだけでなく呼び出し回数が表示されるのでサブルーチンの性能の改善の検討においてどのような呼び出し方をされた時の性能を上げるべきかという優先順位の検討にも使えます。

3.4 性能改善の検討

プロファイルの結果を見て性能改善の方法を検討します。まず真っ先に検討するべきものはdaxpy_のサブルーチンです。これは実行時間の76%を占めていますのでこの部分を10%早くすると全体性能は7.6%上がります。ただし、self us/callの項を見ると実行時間の改善がどれだけできるか心配になります。このサブルーチンは大変実行時間が短いのでチューニングの余地が少ないことがありえます。逆に実行時間が短い分だけサブルーチン呼出のオーバヘッドが実行時間に占める割合が大きなことが予測できるので、インライン展開の効果が期待できます。さて、二番目に実行時間の割合が多いdgefa_についてはどうでしょうか？これは実行時間の12%ほどを占めるので、このサブルーチンを50%性能改善すると全体では6%ほど性能が上がることになります。daxpy_に比べて頑張って性能改善しても全体性能に対する貢献は少ないことが分かりますね。それでもdaxpy_でやるべきことがないほどになったら次にはこのサブルーチンを性能改善の候補としなくてはいけないでしょう。

4. Octave入門の入門

いろいろとテーマが広がりすぎて隔月の雑誌なのに毎回の進捗があまりないのですが、２号空いてしまうとすっかり忘れてしまうのでOctaveのおはなしの続きを書きます。続きといっても前の話は使い方のサンプルしか出していなかったのでここらできちんと入門編を書きます。入門編の初回は表現形式と簡単な演算子についてです。

4.1 Octaveとは

GNUのGPLに従って配布される数値計算パッケージです。ほとんどのLinuxのディストリビューションに含まれているし、FreeBSDなどにもパッケージとして入っています。サポートされる主な機能は以下のようになっています。もちろんAlphaに限らず多くのプラットフォームをサポートしています。

- 線形代数
- 多項式演算
- 非線形方程式
- 常微分方程式
- 最適化制御
- 数値積分
- 制御理論
- 信号処理
- 集合
- 統計
- グラフプロット
- イメージ処理

・音声処理
・入出力
・文字列処理
・システム関数
・プログラミング

私はこのうちのほんのわずかな部分を利用しているだけですが、特に理工系の学生さんのデータ処理などには大変有用なツールになると思います。私自身未経験な機能についても私の勉強も兼ねて連載のなかで実際に試して行きたいと思います。

4.2 表現形式

Octaveでは複素数を含む数値と文字列、行列、構造体などが利用可能です。簡単な例題でそれぞれの表現形式を見てみましょう。また、Octaveでは変数も使えますが大域変数を除いて変数をあらかじめ宣言して使う必要はありません。

```
種別                          表現
-------------------------------------------------------
数値        1.05e-1, 3 + 2i, 0.2e1 + 4e-2j
文字列      "abcd", 'abcd'
行列        [1, 2; 3, 4], [ 1 2   , []
                            3 4]
構造体      x.a=1; x.b=[1 2 ; 3 4]; x.c="abc"
```

複素数を表す文字はｉとｊのどちらでも構いません。また、行列の行を示すためにセミコロン';'か改行を用います。

4.3 算術演算

Octaveでは行列も演算対象にするために算術演算は要素毎の演算と行列を対象とする演算に分かれているものもあります。要素毎の演算を指示する場合には演算子の前にピリオド(.)が付きます。

演算	説明	演算	説明	演算	説明
x + y	加算	x .+ y	要素毎の加算	x - y	減算
x .- y	要素毎の減算	x * y	行列積	x .* y	要素毎の乗算
x / y	右からの商	x ./ y	要素毎の右からの商	x \ y	左からの商
x .\ y	要素毎の左からの商	x ^ y	べき乗(**も使用可)	x .^ y	要素毎のべき乗
-x,	マイナス	+x	単項演算のプラス	x'	共役転置
x.'	転置				

4.4 その他の演算子

その他の演算子として次のようなものがサポートされています。

比較演算子: <, <=, ==, >=, >, !=, ~=, <>
論理演算子: &, |, !, ~, &&, ||
その他: ++, --

4.5 代入

一般のプログラム言語と違って関数の結果が一つではないこともあり代入はちょっと変わっています。例えばLU分解を考えて見ましょう。LU分解は行列を上三角と下三角の行列の積に分解するのですが、LU分解を行う関数は当然ながら最低限この二つの行列を返さなければなりません。さらに、場合によってはピボット選択の結果列の入れ換えが起きたりすると入れ換えの情報も返して貰わないと計算ができません。そこで、このような複数の結果を返す関数における代入もできるようになっています。次に例を見てみましょう。

```
octave:1> a=[1 2; 3 4]
a =

  1  2
  3  4
```

```
octave:2> [l u]=lu(a)
l =

  0.33333  1.00000
  1.00000  0.00000

u =

  3.00000  4.00000
  0.00000  0.66667

octave:3> [l u p]=lu(a)
l =

  1.00000  0.00000
  0.33333  1.00000

u =

  3.00000  4.00000
  0.00000  0.66667

p =

  0  1
  1  0

octave:4> lu(a)
ans =

  0.33333  1.00000
  1.00000  0.00000
```

お分かりですか？Octaveでは複数の結果を返す関数を右辺とする代入は左辺の変数が少ない時には左から代入されていきます。また、左辺の並びが関数の動作を変えることもあります。上記の二番目のlu(a)の呼び出しで置換行列pを左辺に指定したことで下三角行列lの並びは置換後のものに変更されているのが分かると思います。

4.6 配列添字

代入や参照において行列や配列の一部分のみを対象にしたいことが時々あります。Octaveではそのような方法も用意されています。添字の指定ではすべての行もしくは列を意味する特別なオペレータである':'を利用することができます。簡単な行列で調べてみましょう。

```
octave:14> a=[1 2
> 3 4]
a =

  1  2
  3  4

octave:17> a(1,:)
ans =

  1  2

octave:18> a(2,:)
ans =

  3  4

octave:19> a(:,1)
ans =

  1
  3
```

```
octave:20> a(:,2)
ans =

  2
  4
```

また、代入の時には空行列または空ベクトルを意味する[]も使えます。先程の
行列を用いて部分行列に空行列を代入してみましょう。

```
octave:22> a(:,2)=[]
a =

  1
  3
```

いかがですか？2列目の要素が空ベクトルに入れ替わって行列aが変化した様子
が分かると思います。

5. ちょっとだけMPI

並列計算の標準アプリケーションインターフェイスであるMPIも新たにMPI-2の
規格の日本語訳のプロジェクトが始まりました。MPI-2を実装しているシステ
ムはまだまだ少ないと思いますのでMPI-1も実用上は大事です。MPI-1の資料は
有志で翻訳してありますのでご利用ください。 MPI-1のドキュメントはPHASE
から入手できます。

http://phase.etl.go.jp/contrib/MPI-j/

Linux/AlphaではMPICHは簡単に動作するようです。LAMはCライブラリのバグで
まだ完全には動きません。alpha.gnu.orgから新しいライブラリを取ってきた
ら動いたと聞いています。

6. おわりに

私は自宅ではCOMPAQのAERO と IBM PC/XTのマザーボードを487-25MHzに換装した
マシンを使っていますが、ちょっとしたことで64ビットの環境がないと不便を
感じることが多くなってきたので安売りで出ていたAlphaStation 200 4/100を
発注しました。まだ到着していませんが、次号にはこのマシンとの格闘記も載
せられると思います :-0 私の持っている資料ではAS200はARCとSRMの二つのコ
ンソールBIOSが入っていることになっていますが、最近のニュースの書き込み
ではSRMしかないという情報もあり、来てみてのお楽しみです。
こんな無駄使いをしているから21164のマシンを買うお金が貯らないのですが
数値計算の話ばっかりが続くよりも目先が変わって面白いかもしれません :-)