Linux Japan 1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(第六回)」
-- 高性能クラスタとAlphaのアセンブリ言語のおはなし
清水尚彦 nshimizu@et.u-tokai.ac.jp

自宅で使っていたPC/XTケース入りの486マシンは500MBのハードディスクが付いているのですが最近のドライブの低価格化を見てそろそろドライブの追加をしようと考えました。安いドライブは例外なくIDEのものなのですが古いマシンでもIDEは動作はするはずと思い4GBのドライブを買ってきました。ところが、このドライブがDOSでもLinuxでも全く認識されません。家の中の三台のマシンでいろいろと試してみるとどうやらVLバス以前のISAオンリーのボードでは新しいIDEは認識できないようです。Linuxでもダメなところを見るとBIOSの問題ではなさそうです。どこかでタイミング関係の問題があるのかもしれません。コストをあきらめてSCSIにするか思い切ってマザーボードを替えてしまうか悩ましいところですが仕事に使う環境はなかなかドラスティックな変更は困難です。
というわけで、環境のマイグレーションは難しいと痛感しています。2000年問題で情報システムの更新をしている部門も多いと思いますが移行前のシステムが古くなればなるほどマイグレーションは困難になることでしょうね。全面的に買い替えればよいかというとソフトのマイグレーションはもっと大変になってしまいます。
ドライブ容量の不足に対してはホームディレクトリをAS200にしてNFSマウントしてしまえばいいのですが、AS200とPC/XTと両方常に電源を入れる必要があるのも自宅のシステムとしてはどうも納得が行かない気がします。
もう少しいろいろ検討してみようと思います。

IntelとNetscapeがRedHatに投資をすると取りざたされていますが、日本でも日経コンピュータ」9.14号の特集にLinuxの記事が出たことはとうとうLinuxがビジネスシーンで本格的に認知されはじめたことを意味しており嬉しい驚きでした。おそらくデータベースベンダー各社がLinuxサポートを打ち出したことで取り上げざるを得なくなってきたのだと思いますが、「毎日リブートすればそこそこに安定して使える」なんてOSを疑問を感じずに使う人達に取ってこの記事はかなりインパクトを持っているのではないでしょうか。クライアントとしては(Linux Japan 11月号のVJEの開発者の話にも出ましたが)いまだにライブラリにlocaleがきちんと実装されていないことやアプリケーションも不足しているなど問題も多いのですがサーバーは管理者が対応すれば良いだけなので進出は早いと思います。もっともlocaleやアプリケーションは
(必要を感じているなら開発してコントリビューションしろっていう)
使う側の問題でもありあまり強く主張できないところです。

21264の話題が多くなってきましたが、一世代前のプロセッサである21164Aもまだまだ元気です。Samsungでは着実な改良によって700MHzまでのプロセッサを販売しています。
ニュースグループcomp.sys.decに21164Aのプロセッサの98年9月の価格一覧が出ていました。

```
  KP21164-533CN (533MHz 21164A) $400.00
  KP21164-566CN (566MHz 21164A) $900.00
  KP21164-600CN (600MHz 21164A) $1050.00
  KP21164-633CN (633MHz 21164A) $1460.00
  KP21164-667CN (667MHz 21164A) $2475.00
  KP21164-700CN (700MHz 21164A) $5300.00
```

この値段には正直びっくりさせられます。一般にクロック周波数が多少上がったとしても性能にはリニアには影響しないのですが販売価格はこれほどまで違うとするとやはり533MHzのマシンを複数使って性能を出すのが正しい方法だと思います。もともと性能の高いマシンにはプレミアが付きやすいのですがせめて性能に比例した価格にしてくれれば納得しやすいのにと思ってしまいます。
インテルのXeonの出荷によって性能競争はますます拍車がかかりこれが価格の低下につながってくれればユーザーとしては嬉しい限りです。(もっとも私のところで購入できるのはいつのことやら分かりませんが)

Alphaチップを使った高性能並列クラスタシステムを構築するロスアラモス国立研究所のAvalonプロジェクトは執筆時点で70台のプロセッサのクラスタを動かしていますが、彼らはまさに533MHzのプロセッサを使っています。
ちなみに彼らのマシンはいくつかのベンチマークにおいてSGIのOrigin2000と同等の性能を1/10の価格で実現しています。中でも並列LINPACKのベンチマ

ークにはLinux-Alpha-JPメイリングリストで高速化の議論をして後藤さんが
公開したDGEMMルーチンが使われており70台のマシンで19.7GFLOPSの性能をた
たき出しています。もちろんLINPACKのベンチマークを公開しているDongarra
Reportにも出ています。
Avalonで使っている技術は21164Aのプロセッサと164LXと
呼ぶ一般的なマザーボードにIDEのハードディスク、g77、100Base-Tのネット
ワークカードにスイッチングハブとなっていてそれこそ誰にでも使える普通の
技術だけで構成されています。ネットワークの通信ライブラリはMPIを使って
います。
特に注目すべきは100M Ethernetのような安価なネットワークでこれだけの性
能が出せるということです。今後の展開にも目が離せません。

Samsungの21264は98年10月から量産開始なのですがすぐ11月には21264Aが量産
されます。こちらは0.25μmのプロセスで製造されクロック周波数は750MHzか
ら800MHzとアナウンスされています。21264Aのチップ面積は21164Aと同等なの
で歩留まりもかなり良いのではと期待されます。
量産時期が一カ月しか違わないのでクローンマシンの販売店からは21264Aを
使った機械しか販売されないと思いますがコンパックなどではインテルから
調達した21264のマシンも販売すると思われますので最初は価格や品揃えに混
乱があるかもしれません。
21264の世代になるとOut-Of-Order実行(OOO)が実装されるので(今でもロード命令で
はMAEを使って実現できていますが)コンパイラの命令並び替えが多少下手でも
性能がでやすいのでフリーのOSを使う必要がある我々にとって(インテルのPII
のように)楽に性能が出せるようになる可能性が高く期待が持てます。
もちろん本格的に性能を引き出すための最適化は21264になっても必要になる
のでチューニングが趣味だという人にも面白いチップです。OOOだけでなく
メモリ回りの性能が大変向上しているのでその意味でも性能を引き出しやすい
プロセッサになっています。
Samsungの広告には264DPという2プロセッサ用のマザーボードも出ています。
写真によるとDIMMソケットが16個もついているようでメモリ容量の不足に困っ
ていた方もこれでずいぶん助かるのではないでしょうか?

1. 高性能を目指したロスアラモス国立研究所のクラスタ Avalon

さてさっそくAvalonの性能を見てみましょう。
WEBにでている各種ベンチマーク結果を表にまとめてみました。
対抗馬として大規模なシステムを構築
する例が多いSGIのOrigin2000(の64プロセッサモデル)と比較してみます。

ベンチマーク	性能(GFLOPS)	Origin2000(195MHz 64Proc)
parallel Linpack	19.7	20.1
molecular dynamics(SPaSM)	12.8	左と同等
gravitational treecode	10.0	左と同等
NAS ClassB 2.3 BT	2.2	4.1
NAS ClassB 2.3 SP	1.0	3.7
NAS ClassB 2.3 LU	3.5	6.2
NAS ClassB 2.3 MG	2.1	4.0

上位3つのベンチマークの結果は195MHzのSGI Origin2000の64プロセッサモデ
ルとほぼ同等の性能とされています。Origin2000を購入する場合に
はAvalonにくらべて10倍ものお金を払う必要があるのですからそのコスト
パフォーマンスの高さは素晴らしいものがあります。
NASの並列ベンチマークではOrigin2000に大きく負けていますが、
ネットワーク性能が桁違いに違うのでソースコードの変更が許されていないこ
れらのベンチマークでは性能がでにくいのでしょう。
ちなみにAvalonのNASベンチマーク結果の多くは国産超並列コンピュータの代表格である
日立のSR2201より優れた値を出しています。

このシステムは市販の部品を使って組み上げた70台のプロセッサからなるクラ
スタです。ここで使っているマザーボードはDECがWindowsNT用に販売している
PC用のボードです。その他の部品は普通にPCを作る時に利用するものばかりで
す。これらはコモディティハードウェアと呼ばれますが、自由競争のなかで価
格が非常に安く抑えられています。(もっともプロセッサはともかくマザーボードは
もう少し安くてもよいかなと思いますね)

部品	仕様/容量/周波数	説明

CPU	Alpha 21164A 533Mhz	言わずと知れたアルファプロセッサ
M.B.	AlphaPC164LX	DEC 製の ATX マザーボード
メモリ	SDRAM ECC DIMM 128MB	64MB の DIMM を二枚使っています
HDD	Quantum ST3.2A EIDE 3GB	EIDE の PC 用ハードディスクです。
NETWORK	100Mbps FAST ETHERNET CARD	100Mbps の普通のイーサネットカード
ケース	ATX ケース	PC 用の ATX ケース

このノードを接続するネットワークは 3COM のファストイーサネットスイッチを
4 台使ってスイッチ間はギガビットリンクで接続しています。
接続コストはノード当たり$300 であるとしています。
ノード本体は原稿執筆時点で購入したとすると$2000 を切っています。
日本でも 30 万円前後で購入できるスペックです。

次に示すのが利用したソフトウェアです。

OS	RedHat Linux 5.0
コンパイラ	gcc, g77, egcs-1.0.2
数値ライブラリ	Goto's DGEMM
ネットワークライブラリ	MPICH, ロスアラモス開発の独自 MPI

これら Avalon で利用したソフトはすべて GNU ライセン
スに従うフリーソフトとなっています。(ロスアラモス開発の MPI も準備が整い
次第公開するとしています) OS には RedHat を使っていますが、コンパイラや
ライブラリは異なるものを使っています。

Avalon についてもっと知りたい方はつぎの URL を参照ください。

http://swift.lanl.gov/Internal/Computing/Avalon/FAQ.html

2. Alpha のアセンブリ言語入門(その二)

今回は実際にアセンブラの小さなサブルーチンを作ってアセンブラと C との
インターフェイスについて見てみましょう。アセンブラでコーディングするに
は前回のレジスタの使い方の他にもアセンブラディレクティブと呼ばれる疑
似命令の理解が必要です。アセンブラディレクティブには多くの種類があって
覚えるのは大変そうですが必要なものはそれほど多くないので例題を眺めてい
るうちに覚えてしまうと思いますが、良く使うものを簡単に表 1 にまとめて見ます。

表 1 Alpha のアセンブラにおけるディレクティブ

分類	ディレクティブ	説明
領域制御	.align N	二の N 乗のアドレス境界まで NOP で埋める。
	.data	データ領域の開始。
	.text	命令領域の開始。
	.space N	N バイトの 0 を挿入。
記号宣言	.extern label	外部参照名の宣言。
	.globl label	大域名の宣言。
エントリ宣言	.ent label	エントリポイントの宣言。
データ	.ascii "string"	文字列の宣言。複数指定はカンマで区切る
	.asciiz "string"	文字列の宣言、\0 で終端。複数指定はカンマで区切る
	.byte value	8bit 整数の宣言。複数指定はカンマで区切る
	.word value	16bit 整数の宣言。複数指定はカンマで区切る
	.long value	32bit 整数の宣言。複数指定はカンマで区切る
	.quad value	64bit 整数の宣言。複数指定はカンマで区切る
	.float value	単精度浮動小数点の宣言。複数指定はカンマで区切る
	.double value	倍精度浮動小数点の宣言。複数指定はカンマで区切る
	.x_floating value	4 倍精度浮動小数点の宣言。複数指定はカンマで区切る
手続属性	.prologue flag	プロローグ部の終りを示す。gp を利用する場合 flag に 1 を設定。
	.end	手続の終りを示す。
	.frame reg,sz,ret	フレームポインタとフレームの大きさを指定。
	.mask mask,off	汎用レジスタの退避マップと退避エリアのオフセット

| .fmask mask,off | FP レジスタの退避マップと退避エリアのオフセット

word,long,quad の部分はちょっと違和感がありますね。
ディレクティブ以外の一般の命令においても同じですが、もともと PDP-11 の後継機として作られた VAX11 のアセンブラから派生した部分でワードが 16 ビットとして定義されているのでこのようになっています。C では long は 64 ビット整数になるので間違えないようにしましょう。
大体のディレクティブは読めばすぐ分かると思いますが、若干の説明が必要だと思われるのが手続属性の部分です。このディレクティブはデバッガへの情報を与えますが、実際にアセンブラから機械語へ変換する時には利用されません。そこで、デバッガなんて使わないっていう人はこれらのディレクティブを使わなくてもかまいません。
.frame はスタック上に確保する手続のローカル領域であるフレームを宣言します。宣言はアセンブラに対してするだけなので実際に対応するフレームを確保する命令を記述する必要があります。
フレームポインタは通常$30 に設定しますが、これを変更する必要はないでしょうから reg の指定は$30 としておきましょう。フレームのサイズは局所変数とリターンアドレスを加えた数に大体なるはずです。手続で新たにサブルーチンを呼ぶ必要がなければ局所変数の為に必要な大きさを sz に設定します。ret は手続の戻りアドレスを格納するレジスタを指定します。が、これも特に理由がなければ$26 から変える必要はないでしょう。
.prologue はフレームにレジスタを退避するまでのプロローグ部が終ることを宣言します。また、gp をプログラムが必要とするかどうかも併せて宣言しています。gp が必要かどうかの判断は 11 月号に書いた基準通りです。
.mask と .fmask はどのレジスタがフレーム上のどこに格納されるのかを示すものです。マスク情報は 32 本の整数もしくは浮動小数点レジスタに対してそれぞれ一ビットずつ割り当てた整数を指定します。このビットは対応するレジスタをフレームにセーブした場合に 1 とし、$0 もしくは$f0 が LSB(2^0 のビット)となる 32 ビット数になります。オフセットはフレームポインタから格納領域の先頭までの距離とします。

Alpha は RISC とは言っても結構命令数もありますが数値計算で使いそうな命令に限定して紹介することにします。

まず前提として Alpha のアセンブラでは

```
inst op1,op2,op3
```

となっていた時に op1 と op2 が入力で op3 が出力になります。また、データの型が異なる複数の命令がある時には 32 ビットの整数を扱う命令は命令の終りに t の文字が入り、64 ビット整数では q が入ります。また、単精度の浮動小数点では s、倍精度では t という文字でデータの型を表しています。
倍精度や quad の数値はメモリ上の 8 バイト境界(つまり下位 3 ビットが 0 であるアドレス)に置かなくてはならず、単精度や long の数値は 4 バイト境界に置く必要があります。
また、Alpha には条件コードがありませんので、分岐命令はレジスタの値を見て分岐条件を判定します。比較命令は比較結果をレジスタに格納することになります。
小さな整数をプログラムの中で使うことが良くありますが、整数演算命令は 8 ビットまでの符号なし整数を第二オペランドの代わりに指定することができます。Alpha には整数の割算命令はありません。どうしても割算がしたい時にはサブルーチンで実現するか比較的小さな数であれば浮動小数点の割算で代用することができます。

以下の表に数値計算等で良く使う命令を示します。

分類	命令
整数演算	addl subl s4addl s4subl s8addl s8subl addq subq s4addq s4subq s8addq s8subq cmpeq cmplt cmple cmpult cmpule mull mulq
論理演算	and andnot or ornot xor xornot sll sra srl

	cmovxx
分岐命令	beq bge bgt ble blt bne br bsr jmp jsr ret
ロードストア	lda ldah ldl ldq stl stq lds ldt sts stt
浮動小数点演算	adds addt subs subt muls mult divs divt cmptxx cvtqs cvtqt cvtst cvttq cvtts cvtlq cvtql cpys cpyse cpysn fcmovxx
浮動小数点分岐	fbeq fbge fbgt fble fblt fbne

大体見れば想像がつくと思いますが、特徴的な命令について解説します。
s4add 等の命令群はスケール付き加減算命令になっています。これは第一オペランドの値を 4 倍ないしは 8 倍して加減算を行います。配列の要素のアドレス計算などに使います。cmovxx と fcmovxx は条件付きレジスタコピー命令です。条件部分の xx には eq,ge,gt,le,lt,ne などの条件が入ります。パイプラインの長いプロセッサでは分岐によるペナルティが大きいので分岐を発生させないこのような命令は便利です。cvtxx 命令は型変換の命令です。
浮動小数点のレジスタと整数レジスタの間のコピーは直接はできないので型変換をしたい時など一旦メモリにデータを格納してから再度読み出す必要があります。
cpys は符号をコピーする命令ですが、符号部分と値の部分のコピー先を変えられるので絶対値を取る場合など一命令で済ませられます。GCC などでコンパイルした結果を見るとよく bis という命令が出てきますが、これは or 命令のことです。

リスト 1 に非常に単純なアセンブラ手続の例題を示します。この手続は与えられた引数の二倍の数値を返す関数になっています。
この手続を呼び出す例をリスト 2 に示します。
Alpha の API は MIPS と同様手続呼び出しで保護されないレジスタがいくつか定義されているので簡単な手続ならこのようにフレームを使わずにコーディングできますので、いろいろと試してみてください。

リスト 1　アセンブラ手続の例題

```
.data                       # データ領域の開始
        .align 3            # 8 バイト境界への調整
$two:                       # ラベルの宣言
        .double 2.0e0       # 倍精度の数値
.text                       # テキスト領域の開始
        .align 3            # 8 バイト境界への調整
        .globl foo          # 大域名の定義
        .ent foo            # 手続の開始宣言
foo:                        # ラベルの宣言
        ldgp $29,0($27)     # gp の設定
$foo..ng:                   # gp 設定なしの場合の手続入口
        .frame $30,0,$26    # フレーム設定
        .prologue 1         # プロローグ部の終り
        lda $1,$two         # ラベル$two のアドレスを$1 にロード
        ldt $f0,0($1)       # 倍精度数を$f0 にロード(=2.0)
        mult $f0,$f16,$f0   # 浮動小数点引数に$f0 を掛ける
        ret $31,($26),1     # 手続リターン
        .end foo            # 手続の終り
```

リスト 2　例題を呼び出すメインルーチン

```
#include <stdio.h>
double foo(double);
void main()
{
printf("%f\n", foo(2));
}
```

なお、リスト1を times2.s リスト2をmain.cとするとコンパイル方法は

```
cc -o main main.c times2.s
```

となります。
これを実行しても4.0と表示されるだけで実用的なことは何一つないのですが、アセンブラと言えどもそんなに大したことではないなと思って頂ければ十分です。

つぎに引数を取る場合のコーディングを見てみます。どうせ引数を取るならいっそのこと配列の引数を与えて数値計算に使えるようにしてみましょう。
まず、C言語で作成する関数のアウトラインを示します。

リスト3 配列引数を使った手続(C言語によるプロトタイプ)

```
double inpr(int n, double a[], double b[])
{
        int i;
        double sum;
        sum=0.0;
        for(i=0;i<n;i++)
        {
          sum += a[i]*b[i];
        }
        return sum;
}
```

これをアセンブラに変換します。今回はアセンブラ入門なのでアンロールはしません。性能が必要な方は7月号を参考にアンロールしてみてください。ただし、使うレジスタが増えてくれば当然ながら保護すべきレジスタはフレームに退避回復する必要が生じます。アセンブラに変換した関数をリスト4に示します。この例では実際にはフレームを使わなくても保護されないレジスタだけで処理は済むのですが、フレームの使い方も併せて示したいので敢えて保護されるレジスタに一時変数を確保します。
局所変数として i を$9に、sumを$f2に割り当てます。

リスト4 配列変数を使ったアセンブラ手続

```
.text
            .align 3
            .globl inpr
            .ent inpr
inpr:
inptr..ng:
            lda $30,-16($30)    # SP を変更し16バイトのフレームを確保。
            stq  $9,0($30)      # $9 をフレームに退避
            stt  $f2,8($30)     # $f2 をフレームに退避
            lda  $9,0($31)      # i($9)を0にする。
            cpys $f31,$f31,$f2  # sum($f2)を0にする。
$loop:      s8addq $9,$17,$4    # 配列a[i]のアドレス計算
            s8addq $9,$18,$5    # 配列b[i]のアドレス計算
            ldt  $f10,0($4)     # 配列a[i]をロード
            ldt  $f11,0($5)     # 配列b[i]をロード
            mult $f10,$f11,$f13 # a[i]*b[i]の計算
            addt $f13,$f2,$f2   # sum の計算
            addq  $9,1,$9       # i=i+1
            cmplt $9,$16,$3     # i<n ?
            bne   $3,$loop      # つぎの要素の計算へ
            cpys $f2,$f2,$f0    # 関数戻り値の設定
```

```
ldt  $f2,8($30)     # フレームから$f2を回復
ldq  $9,0($30)      # フレームから$9を回復
lda  $30,16($30)    # スタックポインタの回復
ret $31,($26),1     # 関数から戻る
```

関数からさらに関数を呼び出す場合には関数の戻りアドレスを格納している$26の退避回復も必要となります。また、11月号に書いたようにいくつかのケースではgpの設定が必要となることに注意してください。

個々の命令の説明は誌面の都合でできないのですが、11月号と今回を合わせるとかなりの関数を自分で作れるようになると思います。数値計算では配列を使う場合が多いのですが、配列の要素の並びはCとFORTRANで異なりますので今回は割愛させていただきました。またの機会に改めて配列の使い方やFORTRANとのインターフェイスをまとめてみたいと思います。

3. おわりに

Avalonのシステムは誰でも簡単に使えるものだけを使って優れた性能を出しています。21264の時代にはEthernetも1000BaseTあたりが普及してますます高性能が簡単に安く手に入る時代になるのではないでしょうか? でもHigh Performanceは儲からないっていうのがこのコストパフォーマンスを見れば納得です。

主要データベースベンダーがLinuxに参入を表明していますが、まだx86のプラットホームだけのサポートのようです。21264の高性能はおそらくデータベース部門でも十分に役に立つと思いますので、フリーのデータベースのポーティングも試してみたいと思っています。もっともフリーのデータベースではデータの安全性がどこまで確保されるのか心配ですのでプラットホームとしてのAlphaがもっと世界的に認知されてサードパーティからのサポートを増やしたいですね。