VACCELERATOR 運算加速器使用手冊

汪經堯　　劉家瑄

國立台灣大學海洋研究所
Institute of Oceanography
National Taiwan University

國科會海研一號貴重儀器中心

中華民國七十九年十二月

目　錄

一、前言

VAX VMS 電腦的使用者在執行繁複運算的程式時常會懊惱電腦緩慢的運算速度。海研一號貴儀中心多頻道震測資料處理系統目前建立在台大海洋研究所的一台 Micro VAX Ⅱ 電腦上。爲了提升這台電腦的運算能力以有效處理大量的震測資料，貴儀中心購買了兩片由美國 Avalon 電腦公司發展出來的運算加速器，Vaccelerator AP/30，裝置於 Micro VAX Ⅱ 電腦上。一個程式用 AP/30執行能比在 Micro VAX Ⅱ 電腦主機上執行快十五倍左右。即使與 DEC的高級機型 VAX 8800 電腦相比，也能快上兩倍多。

Vaccelerator是利用精簡指令（RISC）技術發展出來的運算加速器。它本身的CPU在作32位元浮點運算時的最高速率可達20 MFlops。對使用者而言，它最大的好處在於容易使用。一般加速運算能力的陣列處理機（array processor）在使用時需在程式中呼叫特殊的副程式（subroutine），而 Vaccelerator 卻只需要把在VAX 電腦上執行的程式重新編譯（Compile）即可。因此幾乎所有能在VAX電腦上跑的程式，都可以利用 Vaccelerator運算加速器。

台大海洋研究所內Micro VAX Ⅱ電腦中的Vaccelerator AP/30加速器雖係爲海研一號多頻道震測資料處理系統而設置，但非常歡迎其它的使用者加以利用。目前購置的 Vaccelerator 編譯器（compiler）爲Fortran-77 compiler，故僅能對Fortran程式使用者提供服務。未來如果對C 或 Pascal 語言編譯器有需求時，亦可考慮增購。

本技術手册之目的在介紹 Vaccelerator AP/30 使用時之注意事項，其與 VAX Fortran不同之處，以及我們使用 Vaccelerator 的一些經驗，以使有意利用 Vaccelerator 加速器的人士能很容易的開始使用。更進一步的硬體特性說明請參閱手册中的附錄。詳細的使用說明與 Fortran 程式的 Run Time Library支援可參閱台大海洋研究所電腦室中的 Vaccelerator Operations Manual。

二、使用方法及說明

台大海洋研究所之 Micro VAX Ⅱ 電腦上共裝了兩片 Vaccelerator AP/30處理機，分別稱爲 ACA0 與 ACB0 。ACA0 有 8 MB 的RAM 記憶體而 ACB0 僅有 4 MB 的 RAM記憶體。使用者可自行選定其一使用，或由系統選定。

Ⅰ、使用方法

Vaccelerator 的 Fortran 使用方式與 VAX Fortran 類似，須經過（AFortran）編譯，及（ALinker） 連結，最後使用（ RUN）來執行。以下是將一個名爲AAA.for的程式編譯、連結、執行的例子：

```
$AFOR   AAA  | RETURN|   ！經由AFortran來編譯，產生AAA.o88
$ALINK  AAA  | RETURN|   ！經由ALinker來連結，產生AAA.exe
$RUN    AAA  | RETURN|   ！使用VAX的RUN指令來執行。
```

上例中，以及在本手册中，|RETURN|代表ENTER 或RETURN鍵，而＂！＂後面的句子爲說明。

注意：1.加速器 Fortran 在編譯及在連結時，其時間會較VAX Fortran 來的久。

2.執行時是使用 RUN 指令而不是 ARUN。

3.Library 的建立與VAX一樣，例如

```
$ALIB  lib_ap  AAA, BBB,CCC
```

Ⅱ、一般說明

(一)機器性能說明：

Avalon Vaccelerator 乃是一種附屬於 Q-bus 或Unibus的直接記憶體存取(DMA)的週邊裝置。它擁有本身的中央處理器(海研所裝置的爲88100CPU)，二個快取記憶體管理元件（Cache Memory Management Unit,CMMU, 88200），及 4 M 的DRAM。可增加VAX電腦額外的一般處理及浮點數運算能力，以減輕VAX的工作負擔。

對一般的VAX 程式而言，此加速器可增快20倍左右的運算速度。然而對一些大量使用I/O 動作的程式，其效率將會大大降低。若程式太小，也顯不出此加速器的作用；有時使用加速器反而會比VAX 還緩慢，這是因爲程式執行時須將所有的程式指令 load 到加速器上。

在load的這段時間內，也許直接執行VAX 版本的小程式早已完成了動作。

(二)加速器系統的一般支援：

1. ACSTAT command：

使用者可利用此指令來觀察加速器的狀態：

```
$ACSTAT     |RETURN|
CPU     S  In  Out  RSize  VSize  Utime    STime  Account  Image  AMSz  MsgC
AP30_0  R  4     0    32     32    0:03.95  0:03     TOM      AAA   298    85
AP30_1  I  20    0    232    232   0:12.19  0:62   R206BOL    FEST  785    58
```

CPU：指所用的加速器 AP30_0是加速器ACA0，AP30_1是加速器ACB0。

S：指state，R爲 running，I 爲idle。

In,Out：指 pages 的 paged in 及 paged out 數目。

RSize,VSize：指 real 及virtual image 的大小 (in kbytes)。

Utime,STime：指 user 及 System 的 CPU 時間。

Account：指使用者的帳號。

Image：指使用者執行的程式。

AMSz：指平均的訊息大小，(in bytes)。

MsgcC：指所有現行程式的訊息總和。

2. Run-time Options：

以下是加速器所提供給使用者的 run-time options. 使用者只要在執行前先定義 acs_acip 的數值，執行時加速器便可依使用者的要求而修正其執行方式（各參數皆可合併使用）：

```
Example 1.
$ define/user  acs_acip  t   ！定義 acs_acip 之參數
$ run  AAA                  ！執行加速器的程式 AAA.exe
Example 2.
$ define/user acs_acip as2  ！定義 acs_acip 之參數
$ run  AAA                  ！執行加速器的程式 AAA.exe
```

(1) a 參數：

加速器程式執行時，若程式試圖去讀或寫一個排列錯誤 (misaligned) 的資料，將會提供一個 fatal error 的訊息。若沒有指定此參數，此排列錯誤的存取將由作業系統的 trap handler 來處理，這將導致 performance

的嚴重減低，而且直到程式終結後，此排列錯誤的次數才會顯示在 SYS$ERROR (即螢幕) 中。本參數對於找出程式在那個地方發生排列錯誤資料的存取很有用。

(2) d 參數：

此參數將使 Avalon 系統 reboot 時，發生的點訊號消失。在VAX 重新開機後，第一次使用到加速器時，加速器系統將重新 reboot ，並印出 reboot...的訊息。

(3) p 參數：

這參數之設定將可促使加速器產生一種名爲 NEWMON.OUT的profile 檔。這個檔對 performance的分析有很大的幫助，詳情請參閱 Operations Manual P.3-11。

(4) r 參數：

可設定加速器的 rounding mode：

```
rm   !  rounding toward minus infinity
rp   !  rounding toward positive infinity
rz   !  rounding toward zero
rn   !  rounding to the nearest even number (default值)
```

例如：

```
$define/user   acs_acip rm
```

將使 0.04768993 → 0.0476899

-0.05981647 → -0.0598165

(趨向負無限大)

(5) t 參數：

設定時可使所欲執行的程式一遇到加速器有其他程式在執行時立刻跳出，並將 status 設爲 SS$_DEVALLOC (2112)。若不設定此參數則程式會一直等到加速器上的另一程式執行完後再進入執行。如果想要使程式馬上執行而不管是在加速器或 VAX上做（當然以加速器爲最先考慮）先決條件爲必須要有兩個執行檔版本，一個是 VAX版，一個是加速器版分別以(AAA-VAX.exe 及AAA-AP.exe)來代表，依下例來執行：

例如： EXE.COM

```
$ define/user acs_acip t
$ RUN  AAA_AP ！執行加速器版
$ if $STATUS .NE. 2112 then goto done  ！2112爲SS$_DEVALLOC
$ RUN  AAA_VAX  ！ 執行 VAX 版
$ done:
$ exit $STATUS
```

(6) S 參數：

用此參數可選擇您所欲使用的加速器。台大海研所的VAX上裝有兩片加速器可供選擇，若不設定此參數則系統將自動把程式載入未被使用的加速器上。但若兩片都正在使用中，則程式將等，到二片中有一片程式已執行完畢後再將等待中的程式載入執行。譬如使用者選的是 ACA0 (有8MB RAM) 則 ：

```
$define  acs_acip  s1
```

若ACA0正被使用，則程式將處於等待狀態。

若選ACB0 (4MB RAM) 則：

```
$define  acs_acip  s2
```

(7) V 參數：

可設定此參數使加速器印出所選擇的介面訊息層次：

```
$ define  acs_acip v2   ！設定層次爲1
```

註：層次可由1 至9 ，層次越高訊息資料越多。

V1：如果加速器有其他程式在執行，則印 waiting 訊息。

V2：印出有關指定(assigning) ，存取 (accessing)，及打開(opening) 加速器的訊息。

V3：印出有關opening 加速器裝置的所有細節。

V4：印出每一個加速器與中央介面程式的交換訊息。

V5：印出 V4 的訊息及所占空間。

V6：印出每次加速器存或取的所有訊息。

V7：產生一個名爲 parent. 的檔，存放所有 VAX 與加速器間交換的訊息。

V8：產生 parent. 並於執行中在螢幕上顯示出。

3．輸出緩衝的取消：

通常在加速器上執行程式時，所有的輸出將會被緩衝（buffered）在加速器中，等到程式執行完畢再將緩衝區中的資料送至VAX 印出。但是如果程式執行到一半發生了嚴重錯誤而停止，則所有輸出將不會被印出。以下所列指令可將此緩衝功能取消：

```
$ define  acs_nobuffer  1    ！使緩衝消失
$ RUN  AAA                   ！執行加速器程式
$ deassign  acs_nobuffer     ！恢復緩衝能力
```

請注意如果取消此buffering 之功能，執行速度將變慢。

(三)浮點數值之考慮：

1．對一般的使用者而言加速器對浮點數的處理與VAX 電腦並無太大的不同。然而VAX 與加速器是二種不同的CPU 元件，所以仍有其差異性。

基本上VAX 所使用的浮點表示法是 DEC floating point 而加速器所使用的則是IEEE Standard floating point，所以程式對於處理 binary 形式的資料時，一定要考慮浮點數的差異性。然對於整數及邏輯數而言，兩者是全然相同的。

2．一般在讀取一筆資料時，若宣告其爲實數或依據 Fortran的內隱宣告原則爲非 I,J,K,L,M,N, 開頭的變數時，自動轉換將自行發生，所以不必擔心其Format上的差異。但若欲將整數及浮點數讀入同一個緩衝資料陣列時，差異則會發生，因爲整數將被視爲實數而加以轉換！若以整數讀入，浮點數會因加速器與VAX 表示法的不同而表現不同的數值。要消除此項差異，可以使自動轉換功能消失，而在程式中自行轉換成浮點式的 format。

要取消自動轉換功能，在連結時要加上 avalon ：fnative

```
$ AFOR  AAA   ！ 產生 AAA.o88
$ ALINK  AAA,avalon:fnative   ！ 產生 AAA.exe
$ RUN  AAA   ！ 執行程式
```

在程式中若需自行轉換格式，可呼叫加速器的副程式。例如：

```
acs_cvtv2af(A)  ！ A 爲所欲轉換的浮點數(single precision)
acs_cvtv2ad(A)  ！ A 爲所欲轉換的浮點數(double precision)
```

Operations Manual的4-1 頁中列有十數個可供利用的副程式。

3．浮點數範圍之差異：

	Length (bits)	Bits in Exponent	Bits in Mantissa	Smallest Number	Largest Number
Single Precision					
Avalon	32	8	23	$\pm 0.12 \times 10^{-37}$	$\pm 0.34 \times 10^{39}$
VAX	32	8	23	$\pm 0.29 \times 10^{-38}$	$\pm 0.17 \times 10^{39}$
Double Precision					
VAX D_floating	64	8	55	$\pm 0.29 \times 10^{-38}$	$\pm 0.17 \times 10^{39}$
Avalon	64	11	52	$\pm 0.22 \times 10^{-307}$	$\pm 0.18 \times 10^{309}$
VAX G_floating	64	11	52	$\pm 0.56 \times 10^{-308}$	$\pm 0.90 \times 10^{308}$

4．輸出數值之差異性：

一個程式使用VAX 版本執行與用加速器版本執行所得之輸出結果如果不同，可用以下的方式改進：

(1)如果差異很小，這是必然的，因爲此兩種不同的浮點數值表示法互相轉換的結果會產生不同的精準度缺失。

(2)如果差異頗大，可試著用不同的 rounding mode， 即定義 acs_acip 的 r 參數(共四種mode)。

例： $ define acs_acip rm

如果4 種 mode 做出來的結果皆有很大的差異，此表示在計算過程當中precision 有很大的誤差。可選擇與VAX 版最接近的 rounding mode 爲以後執行同樣程式的標準。但是如果此四種mode皆沒有很大的差異，則此誤差必然是出現於程式的其他地方，如：

1.在變數傳遞時（副程式呼叫，或數值指定如ax = bx ）若有將兩倍精準度的數值傳給一倍精準度變數的情形，反之亦然，其精準度必然會改變。

2.程式中可能含有反覆的計算某一double precision數值一直到0 的式子。因爲 VAX G_floating 與加速器的 double precision 數值比 VAX D_floating 的範圍大很多，所以他們不會如同 VAX D_floating

一樣很快就變成0 （注意加速器最小爲 $\pm 0.22 \times 10^{-307}$ 而 VAX D_floating 爲 $\pm 0.29 \times 10^{-38}$ ）。

3.最後可能的原因是 VAX D_floating 有55 bits的 mantissa 而 VAX G_floating 與加速器只有52 bits的mantissa。

因此在編繹加速器的程式時，可以先試著將所有的程式（包括 Library ）都以VAX G_floating 的參數來compile ，然後比較一下用加速器執行與用VAX 執行所得的結果來確定是否解決了浮點數值差異的問題。

```
例： $For/G_FLOATING  AAA   ！使用 G_Floating 的參數來
     $LINK            AAA   ！ compile。
     $RUN             AAA
```

三、加速器 FORTRAN 與 VAX FORTRAN 之不同

加速器 Fortran 與 VAX Fortran 兩者間的差異性並不大，而且大部份的 VAX Fortran system subroutine在加速器上皆有支援。如果使用到加速器沒支援的 system call，可請 system manager 將其加入。下列的是一些加速器 Fortran 並未支援的項目：

1．有關於 VAX Frotran 的可變型式輸出入，例如：

10 Format(1x,I<m>.<n>) 此項並沒支援，但可用加速器 Fortran 提供的副程式 afor$vfsetup 來做同樣的工作。

例如在 VAX 上爲： write (6,10) A

```
10     Format (1X,'RESULT=',F<12-I>.<I>)
```

可改爲：

```
Character*200    AFOR$VFSETUP,FMT
. . . . . .
FMT=AFOR$VFSERUP('1X,"RESULT=",F<12-I>.<I>',12-I,I)
write(6,FMT) A
```

此 AFOR$VFSETUP 也可以由VAX 程式來呼叫，但須將原始程式的 obj 檔與 AVALON:ACS.OLB 來連結（LINK）。

2．加速器 fortran 並不支援 structure 宣告內的初值設定，所以必須在宣告 structure 後再設定其初值。例如在 VAX 程式中：

```
STRUCTURE
        integer  set1,set2/2/
        real     data1/3.0/,data2/4.0/
END STRUCTURE
```

須改爲

```
STRUCTRUE  /struc/
        integer  set1,set2
        real     data1,data2
ENDSTRUCTURE
        struc.sef2=2
        struc.data1=3.0
        struc.data2=4.0
```

3．關於RECORD的宣告，加速器不能接受同一RECORD宣告內有兩個不同的record。

例如在 VAX 程式中：

```
RECORD  /STRUCT_1/R1,/STRUCT_2/R2
```

須改爲

```
RECORD  /STRUCT_1/R1
RECORD  /STRUCT_2/R2
```

4．有關於ISHFT的Integer*2之使用：

加速器 Fortran 在處理integer*2 的ISHFT 或IISHFT副程式（function）時，如果n爲負值(即向右移)，將會錯誤的表現出arithmetic shifts ，而不是logical shifts。 所以在使用加速器時，不要呼叫 ISHFT 來處理integer*2 的數值，也不要呼叫IISHFT來處理n值爲 負值的情形。

例：
```
integer*2  ISHFT,Temp
Temp=ISHFT(-2)  ! arithmetic  shift
```

例：
```
integer*2  IISHFT,Temp
Temp=IISHFT(-2) ! arithmetic  shift
```

例：
```
integer*4  ISHFT,Temp
Temp=ISHFT(-3)   !  logical shift
Temp=ISHFT(3)    !  logical shift
```

5．當程式中有如下的表示式時，加速器fortran 編譯器會自動將常數去掉以加速運算

```
IF (1.0+tiny .eq. 1.0) THEN
```

會變成 `IF (tiny     .eq. 0.0) THEN`

此修正對無限精確的數值（infintely precise number）來說是正確的，但是如果程式本來就是要檢查此tiny的浮點數表示之精準度時，錯誤將會發生。要解決此問題，可使用／NOOPTIMIZE的qualifer來編譯程式，例如：

```
$AFOR/NOOPTIMIZE  AAA
$ALINK  AAA
$RUN    AAA
```

6．當使用AFORTRAN COMPILER 來COMPILE 程式時，最好加上/XOPTIONS=479 這個qualifer（當然不加也可以），如：

```
$AFOR/XOPTIONS=479  AAA
$ALINK              AAA
$RUN                AAA
```

此選擇項可以增快您程式的速度，其原因可參閱 Vaccelerator Operations Manual 5-7頁。

7．在 ALINKER 的選擇項中，沒有類似 VAX LINKER 的 /LIB/ 與 INCLUDE/兩項，所以若程式需要與某一個由 ALIBRARY 建立出來的 library連結，可直接寫出程式庫全名，再加上其附加名。例如：

```
$ALINK  AAA,BBB.881b  ！將  AAA.o88與 BBB.881b(程式庫)連結
```

而不能以如下方式連結：

```
$ALINK  AAA.BBB/LIB  ！錯誤!!
```

8．關於附加名方面：

	Avalon加速器	VAX
objct file	XXX.o88	XXX.obj
library file	XXX.881b	XXX.OLB

9．關於加速器之library功能：

ALIB的執行速度很慢，AFOR及ALINK的速度也很慢，所以在執行時請要有耐心。

１０．其他如 AFOR，ALINK，ALIB，Avalon RX88 debuger，Avalon MX88 Assember 的quaifier，可參閱 Vaccelerator Operations Manual 的相關章節。

四、VAX FORTRAN 程式轉換成 VACCELERATOR FORTRAN 程式問題處理

I、重新編譯

這是將程式轉換至加速器上的第一步。爲防萬一，可先在重新編譯時加入兩個選擇項，以便程式出現錯誤時使用Avalon debuger

```
$ AFOR/DEBUG=TRACEBACK/CHECK=BOUNDS  AAA
```

雖然因此而產生的機器碼會在速度上略爲減慢，但是能提早發現出錯的地方。等到程式完全無誤後，再以如下方式重新compile過：

```
$AFOR/XOPTTION=479  AAA  ！ 產生快速之機器碼
```

(一)AFortran 未顯示的警告訊息：

(1) Gap left in common block due to alignment rules

通常此warning 是可以忽略。若要加速器 Fortran 程式產生與VAX Fortran程式一樣不規則排列的 common block，則可在 compile時加入/XOPTIONS=89的選擇項。

(2) Illegal GOTO in loop

此警訊千萬不能忽視，因爲程式中的 GOTO loop 已經違反了FORTRAN-77 standard了。以下是一個會產生如此訊息的例子，以及其改進方法：

```
            Do  10
            Goto  10     ！此Goto爲不合法
            Do  10
            . . . . . .
        10  continue
```

我們改變程式爲：

改進 1 ：

```
     Do  10
     Goto  10
     Do....
     ......
     Enddo
 10  continue
```

改進 2 ：

```
     Do  10
     Goto  10
     Do  20
   20 continue
   10 continue
```

(二)錯誤訊息，而.o88檔並不產生：

(1) Error in complex constant:

此項訊息在使用內隱的do loop 於output時變數被括號括起來的情況，例如： write (*.*) ((X(I) ,Y(I)),I=1,N)
只需將多餘的括號去掉，如下：

```
write (*.*),(X(I),Y(I),I=1,N)
```

(2) Internal compiler error (or Compiler Crash):

如發生了如上的訊息，表示Avalon Fortran Compiler有問題產生，請與Avalon公司連絡。

(3) Multiple initialization:

此表示在同一個程式、副程式或模組中，重覆地初值化了同一塊記憶區（通常爲common block）。請試著只用宣告區來Compile，並盡量減化宣告，直到找到重覆宣告的地方，以刪除重覆宣告的部份。

(4) Out of Memory

此錯誤乃在於加速器 compiler 無法再配置任何的虛擬記憶體了。請 system manager把 加速器 的 swap file 空間加大，便可解決此問題！

(5)其他的錯誤及警告訊息皆與VAX Frotran一樣。

Ⅱ、ALINK 與 ALIB

用 ALINK來產生可執行檔，或用ALIB來產生程式庫：

(一)ALIB的執行動作很緩慢，請要有耐心！

(二)如在link時產生 undefined symbols 的訊息，若爲 main，請檢查主程式名稱是否爲用英文字母開頭的。若不是，則請改爲以英文字母開頭：

```
如：  $ALINK    1234.088    ! 不合法
      $REN      1234.088  t1234.088
      $ALINK    t1234.088   ! 合法
```

若undefined 的symbol 是VMS 的system subroutine ，請與Avalon公司連絡。

(三) Multiple defined symbol

在VAX 上不同版本的object file 可以link在一起，但會產生

警告訊息。而VAX 的 object Library 用在加速器上時，則不一定會產生警告。對於一些不同動作版本而同一名稱的subroutine而言，情況會變得很混亂。所以在加速器 Fortran中同一名稱的 subroutine 只留一個版本，或rename它！

(四) Alinker沒支援的選擇項： /LIBRARY及 /INCLUDE=mode

在VAX: $LINK AAA,BBB/LIB ！將AAA.obj與BBB.OLB(程式庫)連結！

但在Avalon 則不可,須用:

$ALINK AAA,BBB.88LB ！將AAA.o88與BBB.88LB(程式庫)連結。

Ⅲ、偵測執行時之錯誤：

要偵測執行時之錯誤可用Avalon公司提供的 RX88 bebugger。在程式中插入由D開頭的測試行，並在 compile 時加入/LINES 的選擇項參數以執行這些測試行指令。如果在編譯時沒有下/D_LINES的選擇項，此以D 開頭的測試行將會被認爲是 COMMENT 而 不被 COMPILE (此參數在VAX Fortran 中也有)。

例：

```
        program    AAA
C       This is a comment line!
D       PRINT*, This will not be printed out unless you
D      *USE /D_LINES qualifier
        Do    I=1,n
        .........
        Enddo
        .........
```

以上程式若以下列方式來編譯、聯結及執行：

```
$AFOR  AAA              $FOR  AAA
$ALINK AAA    或        $LINK AAA
$RUN   AAA              $RUN  AAA
```

則以D 爲第一列的print訊息將不被印出，而被視爲是一個comment。

但用：

```
$ AFOR/D_LINES  AAA      $ FOR/D_LINES  AAA
$ ALINK         AAA  或  $ LINK         AAA
$ RUN           AAA      $RUN           AAA
```

則print 的訊息將被印出!!

若程式在VAX 與在加速器上皆出現同樣的錯誤，則最好利用VAX 的debugger來偵錯，改錯完了以後，再以加速器來執行！

以下是一些在執行加速器版程式時可能出現的錯誤訊息：

(一) Accelerator system error number XX (XX 爲某數字)

可能是與系統架構有關 (system configuration)，請檢查系統的 swap file permissions!

(二) %SYSTEM-F-ACCVIO,access violation,....

此種錯誤不太可能發生，若發生了請檢查錯誤發生前的最後一個系統界面程式呼叫 (interface call)。尤其要注意系統呼叫中傳遞的參數是否正確。若仍檢查不出來，請用一個小的測試程式來引發相同的錯誤，並與Avalon公司連絡。

(三) %ACIP-F-TERM,access fault at PC 001908

%ACIP-F-TERM,numeric result error at PC 001862

這兩種錯誤比較常見。 numeric result error 指一個整數或浮點數被零除，常是因爲 format 自動轉換發生錯誤，可用 RX88 debugger 或 D_Lines 參數來解決。而access fault 則是因程式試圖去存取一個不合語法的記憶體位置，可加入 /CHECK=BOUNDS 重新編譯程式來找出問題所在。

(四) syntax error in format

此乃因程式中使用了可變變數的 format 宣告，如：

```
10    Format (1x,F<m>.<n>)
```

請用 afor$vfsetup(...)這個 subroutine 來解決可變變數格式宣告的問題。

(五) Open failure

這是因爲 open file page quota 不夠之故。加速器需要使用比VAX 大兩倍的 open file page quota ，可請system manager將 page quota加倍，以解決此問題。

(六) Fortran run-time error :subscript out of range for" iarray "

Subscrip was -1 at line 2434 in subr1_

%SYSTEM-F-SUBRNG,arithmetric trap,subscrip out of range

這是在用/CHECK=BOUNDS/時可能出現之訊息，表示用了錯誤

的 array reference。請用D_Lines 參數來找此錯誤的地方。再不然就是引用了錯誤的Goto loop(如前所述者)。

(七)其他的錯誤訊息與VAX Fortran一樣。

Ⅳ、加速器程式最佳化

(一)可定義 acs_acip P 來產生一個 profile，並執行 RX88 來觀看此 profile 的內容。profile中將包括呼叫每一副程式所用去的時間，可依此來改進呼叫時間最久的程式，而增進程式的效率。

(二)編譯時可使用 /Xoptions=479 的參數來產生更快的執行碼！

(三)多利用 Avalon 公司所提供的系統呼叫(列於 Vaccelerator Operations Manual 4.1 中)

(四)若使用 sys$qio 來寫出大筆資料，可改爲呼叫 acs$qio 來達成。但此呼叫只能用於寫出而不能用於讀入。

(五)其他資料可參閱手册的 p3-11 及 p3-37。

五、結語

這本手冊簡單的介紹了Avalon 公司出產的 Vaccelerator AP/30 加速器的功能與使用方法，以及在執行程式時可能會遇到的一些問題。這些問題有一些是我們在操作時眞正遇到了，也有許多是還未曾遇上而 Vaccelerator Operations Manual 中有記載，故一併記錄下來。在使用加速器時若發生了無法解決的問題，而在本手冊與 Vaccelerator Operations Manual 中均沒有列出時，請與 system manager 連絡，或請問代理此加速器的典漢公司工程師。

VACCELERATOR™ AP/30

The VACCELERATOR AP/30–more than twice the power of a VAX 8800!

The Avalon ***VACCELERATOR*** **Model AP/30** combines **VMS** and **RISC** in a single slot application accelerator for VAX and MicroVAX computer systems. The AP/30 runs compute–bound application programs 15 times faster than a MicroVAX II (15 MVUPs). In most cases, programs written in Fortran, C, or Pascal can be recompiled and run on the AP/30 without modification. Most VMS and Ultrix system services and run–time libraries are supported. Application file I/O and other host services are processed transparently by the AP/30 through Avalon's proprietary Central Interface Program (ACIP) that runs on the VAX.

VACCELERATOR AP/30 BENEFITS & FEATURES

High Performance

- Powerful RISC CPU
- 15x MicroVAX II
- 20 Mflops (peak) 32–bit floating–point
- 10 Mflops (peak) 64–bit floating–point

Application Transparency

- Optimizing Fortran–77 compiler with support for most VMS extensions
- Optimizing C and Pascal compilers
- VMS and Ultrix run–time libraries and system services
- Read/write VAX VMS or Ultrix files

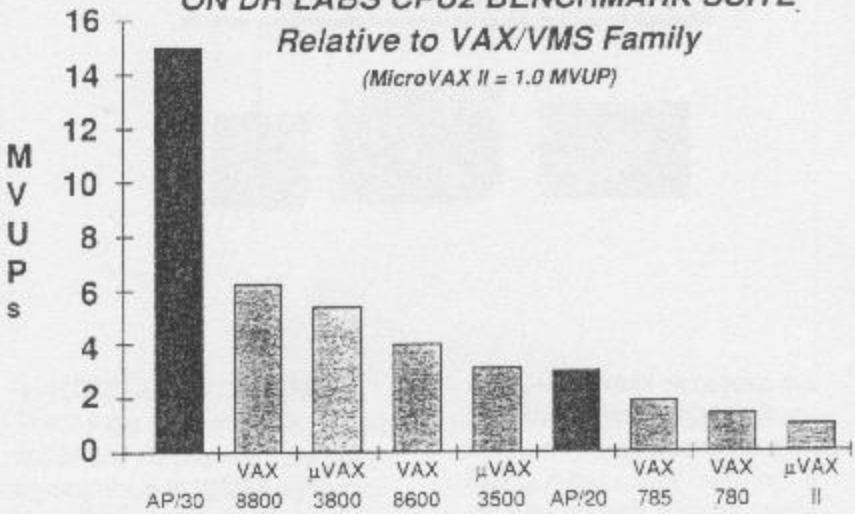
Greater Throughput

- Off–load host VAX system
- Run applications in parallel with multiple AP/30s
- Shared Memory
- Direct I/O to Q-bus/Unibus devices

VACCELERATOR PERFORMANCE ON DR LABS CPU2 BENCHMARK SUITE

Relative to VAX/VMS Family

(MicroVAX II = 1.0 MVUP)



The graph above depicts performance on the CPU2 suite of 34 benchmark programs, as reported by Digital Review, October, 1989.

VACCELERATOR AP/30 COMPARATIVE PERFORMANCE

Benchmark	µVAX II	AP/30	AP/30 vs. µVAX II
Dhrystones	1,405 Dhrystones	37,000 Dhrystones	26.3 times faster
SP Whetstones	94 MegaWhetstones	1,513 MegaWhetstones	16.2 times faster
DP Whetstones	69 MegaWhetstones	723 MegaWhetstones	10.5 times faster
SP Linpack	1.7 Megaflops	27 Megaflops	15.8 times faster
DP Linpack	1.2 Megaflops	14 Megaflops	11.9 times faster

The table above shows AP/30 performance relative to the MicroVAX II on a series of standard synthetic benchmarks.

VACCELERATOR AP/30 HARDWARE

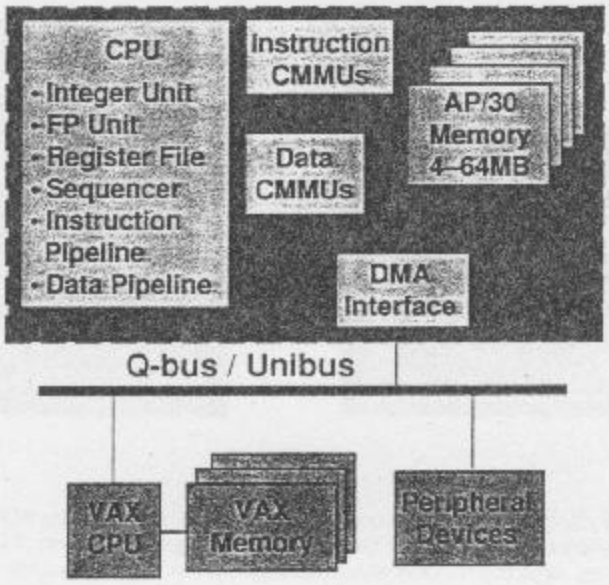
RISC

The *VACCELERATOR* AP/30 is based on state–of–the–art Reduced Instruction Set Computer (RISC) microprocessor technology. RISC is known to provide superior performance at low cost for applications written in high-level languages.

AP/30 Hardware Features

- RISC microprocessor CPU
- Integral floating–point unit
- 32KB 4–way set–associative instruction and data caches with streaming refill
- Up to 64MB main memory
- Q-bus/Unibus DMA interface for access to VAX memory and peripheral devices

AP/30 HARDWARE BLOCK DIAGRAM



VACCELERATOR CPU

The AP/30 is based on the Motorola MC88100 RISC microprocessor. Operating at 20MHz, the MC88100 executes most instructions in a single cycle. The integral floating–point processor contains pipelined adder and multiplier units that can operate in parallel yielding peak performance of 20 Mflops (32–bit floating–point) and 10 Mflops (64–bit floating–point), speeds previously attainable only through specialized array processors. With shared access to a large set of general purpose registers, floating–point operations are executed efficiently with no data movement or control latency.

VACCELERATOR CMMU

The AP/30 uses multiple Motorola MC88200 Cache Memory Management Units (CMMUs) to provide virtual memory access and high–speed cache for both instructions and data. Each CMMU is 16KB in size, and is 4–Way Set Associative, Write–Back in operation. The CMMU provides application programs with access to 1 Gbyte of virtual address space, with demand paging.

VACCELERATOR Memory

Main memory is composed of 4MB–64MB of 100ns DRAM. Memory is accessed only when instructions or data are not found in the cache. In such instances, memory words are loaded into cache 16 bytes at a time. This multi–word refill increases the likelihood that subsequent instruction/data references will be found in the cache. Memory is configured with surface mount DRAM devices for up to 20MB in a single slot.

Q–bus/Unibus Interface

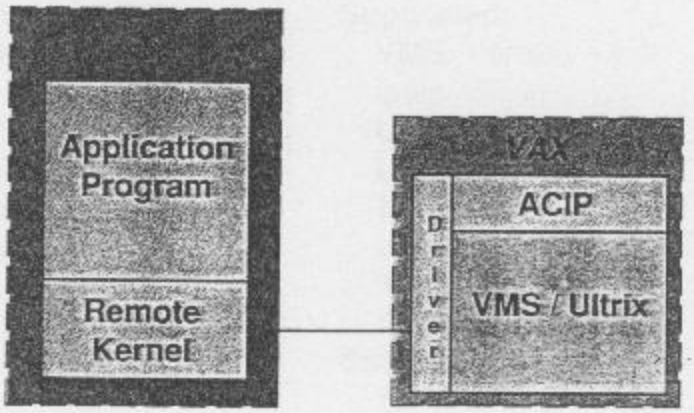
The AP/30 can access VAX memory and Q–bus/Unibus peripheral devices via high-speed DMA. Q–bus block mode is supported. VAX memory may be shared by AP/30 applications as a means of multi–processor communication.

VACCELERATOR AP/30 SOFTWARE

VACCELERATOR Software

The *VACCELERATOR* AP/30 is supported by a sophisticated set of development and operational software. Development software enables applications to be recompiled for the AP/30 with a minimum of effort, retaining the user's familiar VMS or Ultrix environment. Operational software provides run–time transparency to applications running on the *VACCELERATOR* that require access to VAX files, devices, or system services.

AP/30 SOFTWARE BLOCK DIAGRAM



VACCELERATOR software handles all communication between the *VACCELERATOR* and the host VAX. No special programming is required to implement data transfers, synchronization, program loading, interrupt response, peripheral I/O and other similar tasks.

Development Tools

The AP/30 Development Software suite consists of Fortran, C, and Pascal compilers, assembler/linker, debugger, and execution profiler. All Avalon compilers offer extensive state–of–the–art global optimization.

Compilers

The Avalon **Fortran** compiler is full ANSI–1977 compatible, and supports most VMS Fortran extensions. The Avalon **C** compiler is K&R compliant, and supports most VMS extensions. The Avalon **Pascal** compiler is ANSI/ISO compliant.

Assembler/Linker/Debugger

The Avalon Assembler enables user intervention at the assembly language level for special purpose algorithm optimization. The linker follows VMS and Ultrix usage conventions. A symbolic debugger is provided for AP/30 specific debugging.

Execution Profiler

The Avalon execution Profiler provides a subroutine-by-subroutine analysis of execution time as a percentage of overall time. Further, instruction level profiling is available for assembly language program optimization.

Operational Software

The Avalon Operational Software suite consists of an AP/30 Remote Kernel, Device Driver, and VAX resident Avalon Central Interface Program with support for system services.

Remote Kernel

The Remote Kernel supports application execution on the AP/30 by managing memory allocation, demand paging, interrupt handling, and other necessary functions. It interfaces with the ACIP on the VAX through message passing DMA.

ACIP

The Avalon Central Interface Program (ACIP) executes on the VAX and transparently performs I/O and system service requests on behalf of the *VACCELERATOR*. Through the ACIP, user applications running on the AP/30 appear to be running under the host operating system: VMS, Ultrix, or Unix.

System Services and Run–Time Libraries

An extensive set of VMS and Ultrix system services and run–time libraries is provided, enabling applications referencing these services and routines to run without change on the AP/30.

VACCELERATOR AP/30 SPECIFICATIONS

PERFORMANCE

Dhrystones (Rev 1.1): 37,000 Dhrystones
Whetstones (SP): 15,300 Kwips
Whetstones (DP): 7,300 Kwips
Linpack (SP): 2.7 Mflops
Linpack (DP): 1.4 Mflops
DR Labs CPU2: 15 MVUPs
Peak MIPS Rating: 26 (MicroVAX II =1)
Application MIPS Rating: 15 (MicroVAX II = 1)
Peak Mflops Rating: 20 (32–bit floating–point)
Peak Mflops Rating: 10 (64–bit floating–point)

HARDWARE

CPU: Motorola MC88100, with integral FPU, 20MHz clock rate
CMMU: Motorola MC88200, 16KB, 4–Way Set–Associative, Burst–Copyback
Memory: 4MB–64MB 100ns DRAM
DMA: 22–bit address, Q–bus/Unibus access Q–bus block mode supported
Size: Single Q-busQuad/ Unibus Hex slot
Power: 3.9 amps @5vdc
Bus Loads: 1

SOFTWARE

Operating Systems Supported:
VMS: Version 4.4 or later
Ultrix: All Versions
Unix: bsd 4.2, 4.3
Languages: Fortran–77 including VMS & Mil–Std 1753 extensions, C, Pascal
Software Tools: Assembler/Linker, Debugger, Program Execution Profiler, Remote Kernel, Avalon Central Interface Program (ACIP)

AVALON

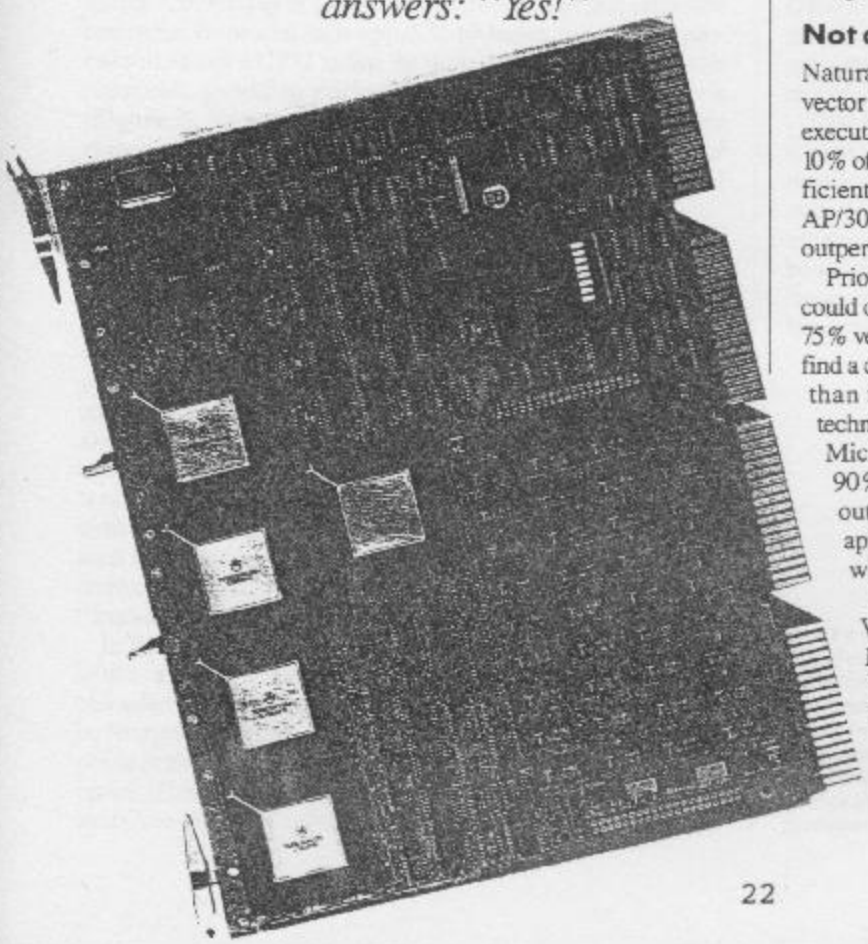
Avalon Computer Systems, Inc.
510 Castillo Street
Santa Barbara, CA 93101
Telephone: (805)965–9559
FAX: (805)965–9723

SYSTEM INTEGRATION

RISC Engine Revs VMS Applications

by John A. Carbone
and Tom Harvey,
Avalon Computer Systems
Glendale, CA

Can VAX integrators gain the mainframe-level performance of RISC while maintaining VMS compatibility? A board-level solution answers: "Yes!"



As RISC processors emerge on board-level subsystems, system integrators get closer to the goal of mainframe speed without mainframe price. Compute-intensive applications such as signal and image processing, geophysical analysis, graphics, and simulation will be the first to benefit. These all demand fast integer and floating-point processing speed, large memory, high-level language compilers, and a friendly user interface.

One of the first examples of a RISC-based attached processor board is the Vaccelerator AP/30 from Avalon. The Vaccelerator AP/30 operates in a VAX or MicroVAX computer system, and delivers real application performance of 15 MVUP (MicroVAX II Units of Performance)—more than twice that of Digital Equipment Corp.'s (Maynard, MA) high-end VAX 8800. Key to the Vaccelerator AP/30 design strategy is application-transparent operation. It generally can recompile and run existing Fortran, C, or Pascal code without modifications. Under VMS, the Vaccelerator AP/30 supports virtually all Fortran extensions, system services, and run-time libraries.

Clearly, software has become a prime consideration for those migrating to RISC. While RISC workstations are available, they may entail a sizable software retooling task. For its part, DEC has released a RISC-based workstation system, but it is not available with VMS. Here, an attached processor such as the Avalon design can represent a favorable alternative.

Not an Array Processor

Naturally, if an application is composed solely of microcoded vector operations, an array processor easily provides a dramatic execution boost. But if an application should contain as little as 10% of "other code," which the array processor cannot handle efficiently, the application will not perform nearly as well. The AP/30 is not an array processor, but a fast scalar processor that outperforms array processors on most scientific applications.

Prior to the availability of RISC processors, an array processor could outperform a scalar processor on any application with over 75% vector content. This is because it was virtually impossible to find a comparably priced scalar system that could perform better than four to five times a MicroVAX II. Now, with RISC technology delivering more than 10 times the performance of a MicroVAX II, an application would have to contain well over 90% vector content before a RISC scalar processor could be outperformed (**Table 1a**). Most scientific and engineering applications contain between 40% and 75% vector content, with very few over 90%.

Based on the Motorola (Tempe, AZ) 88000, the Avalon Vaccelerator AP/30 achieves 15 times the performance of a MicroVAX II on real-world compute-bound applications. This measurement represents total application performance, not just performance on certain parts of the program. MicroVAX II users can expect to see their entire application run 15 times faster. (**Table 1b** provides a cross section of relevant benchmark data.)

The Vaccelerator AP/30 combines the MC88100 RISC CPU and MC88200 Cache and Memory Management Unit (CMMU). The 88000 (88100 and

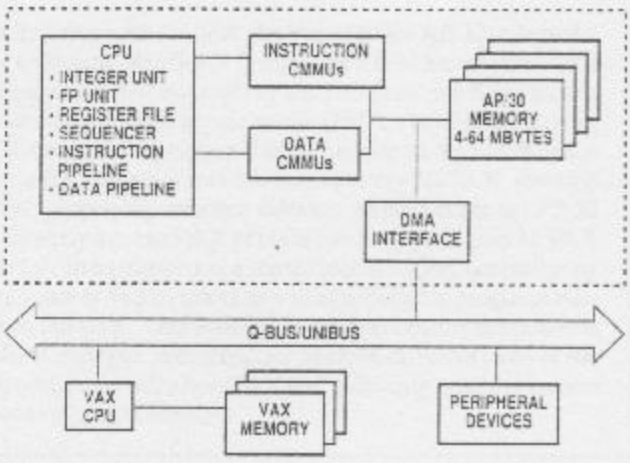


Figure 1: Avalon's Vaccelerator AP/30 combines the MC88100 RISC CPU and MC88200 Cache and Memory Management Unit (CMMU). Consisting of processor, memory, and bus interface connected via several high-speed 32-bit buses, the attached processor board's CPU is fed by instruction and data cache memories, providing two-cycle access over 90% of the time.

88200 collectively) architecture has several key advantages over other RISC implementations: cache coherency, multiprocessing support, a scalable architecture, built-in fault-tolerant capabilities, and the integration of floating-point and integer functions on the CPU chip. A three-chip set, MC88100 CPU, MC88200 Instruction CMMU, and MC88200 Data CMMU—plus memory—is all that's required for a complete implementation. Consisting of processor, memory, and bus interface connected via several high-speed 32-bit buses, the attached processor board's CPU is fed by instruction and data cache memories, providing single-cycle access over 90% of the time (**Figure 1**). Internal to the CPU are 32 32-bit general-purpose registers, from which all operands are accessed. In the design of the AP/30, a balance is achieved among the memory hierarchy of register, cache, and main memory.

As is typical in RISC systems, all operations are performed on data in one of the general-purpose registers. This data has to be loaded into the register prior to the operation. The compiler tries to maximize the use of a given data element once it is loaded into a register, to avoid having to store it and later reload it for another operation. However, since it is not always possible to avoid loads and stores, the performance characteristics of these operations directly influence system performance (**Table 1c**). Loads from cache take only two clock cycles to complete; thus the loaded data is not available to the very next instruction. This latency can be used by instructions that do not require the data just loaded, if such instructions are available. If not, a scoreboard hold occurs and a cycle is lost. Compiler efficiency determines how often this "load-delay-slot" can be filled, avoiding the no-op.

In RISC implementations, serious delays are encountered in the case of a cache miss. This is when the data required to be loaded into a register is not already in cache memory, and must be fetched from main memory. While data can be loaded from cache in two cycles, loading data from main memory takes six cycles. The AP/30 has a cache-burst refill feature, though, that loads into cache not only the data item that was referenced in the load instruction, but also the next three 32-bit words from memory—at only one clock cycle per word.

This is very effective since most memory references are sequential for at least four words. This means that if word N is needed by the program, chances are good that it will soon need words N+1, N+2, and N+3. Rather than incur a cache miss on all four—at a cost of six cycles each—the burst refill avoids cache misses on the references to N+1, N+2, and N+3, since they will be in the cache when referenced. The AP/30 uses this feature for both data and instructions, since instructions can cache-miss when needed as well.

Instruction and data caches each comprise 16-Kbyte, four-way associative memory. Up to 32 Kbytes each of instruction and/or data cache can be configured on the AP/30. Also, the AP/30 provides demand-paged virtual memory management and support for multiprocessor configurations.

The AP/30 can accommodate from 8 Mbytes of main memory in a single slot; up to 64 Mbytes in two slots. This memory is sufficient for most scientific and engineering applications. Demand-paged virtual memory is also supported, enabling the AP/30 to handle programs of any size up to 1 Gbyte. The amount of main memory configured affects performance, since demand paging from disk is substantially slower than main memory references. As with the VAX, a user must make the cost/performance trade-off of how much memory to configure on the system.

AP/30 memory is based on 1-Mbit DRAMs with 100-nsec cycle time and nibble-mode access. The AP/30 memory design can accommodate 4-Mbit DRAMs as well. Though they now incur a premium price, these 4-Mbit DRAMs will become standard around year's end as prices drop closer to 1-Mbit parts on a cost-per-bit basis, according to industry projections. With 4-Mbit DRAMs, the AP/30 can accommodate 64 Mbytes of local memory in a two-slot Q-bus board set. The AP/30 memory bus supports a peak data bandwidth of 80 Mbytes/sec, and a sustained rate of 35 Mbytes/sec.

The AP/30's large memory capacity enables an entire application process to load and run with no shuffling of data or commands back and forth from the host. The only communication required with the host is when the application requests I/O or some other operating system service to be performed. For compute-bound applications, this is extremely infrequent. Even though infrequent, this interaction is performed via block-mode DMA from the AP/30.

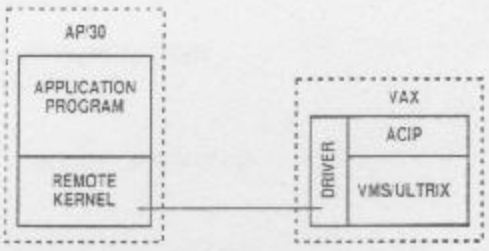


Figure 2: The AP/30 gives users the software they need to maximize the available processing power. Motorola's 88000 is supported by optimizing compilers. Avalon's Central Interface Program (ACIP) provides transparent OS service access to applications running on the AP/30. A local operating system, the Remote Kernel, handles all application system service requests that would normally (if the application were running on an unaided VAX) be handled by VMS.

SYSTEM INTEGRATION

To interface with the host, the Vaccelerator AP/30 utilizes the VAX Unibus or MicroVAX Q-bus. The AP/30 has a microcoded state machine that controls bus access, and can become bus master and perform block-mode DMA. As bus master, the AP/30 can access Q-bus or Unibus peripheral devices directly. Ordinarily, these devices are accessed via the VAX, through Avalon's proprietary interface software. As bus master, the AP/30 can directly access VAX private memory; a region of VAX memory can be mapped as a shared global region, accessible by any number of Vaccelerators as well as application programs running on the VAX. This is useful in multiprocessing applications in which multiple Vaccelerators work on different parts of the same problem, with a host VAX task providing synchronization and overall system control.

The AP/30 is fabricated almost exclusively with surface-mount technology, with low-profile memory expansion daughterboard packaging that meets Q-bus backplane spacing requirements in MicroVAX II systems. Surface-mount technology also enables efficient and accurate automatic machine assembly of AP/30 boards, eliminating assembly errors that frequently occur with DIPs.

Making the Software Connection

Sophisticated computer designers and users have come to realize that high performance without the software to drive it is of little use. Especially true of RISC systems, good compiler technology is necessary to unlock the full capabilities of a CPU. With RISC architectures, compilers must contend with allocation of large

Beating the MicroVAX at Its Own Game

```
      SUBROUTINE EDGETHIN

      IMPLICIT INTEGER*2(A-Y)
      INCLUDE 'TRACKERDATA.INC'
      INCLUDE 'IMAGEDATA.INC'

      integer*2 x1(8), y1(8)              !direction templates

      data x1 /0,1,1,1,0,-1,-1,-1/
      data y1 /1,1,0,-1,-1,-1,0,1/

         type*, 'thresh =', SOBEL__threshOLD

      DO 1200 LINE = 1, NLines
         DO 1100 SAMP = 1, NPixels

            CNTMAG = SOBEL__ARRAY (SAMP,LINE)
            CNTDIR = GradientDirImg__ARRAY (SAMP, LINE)

         OPPDIR = MOD (CNTDIR + 4,8) ! oppdir =
           180 degrees from CNTDIR
         IF (OPPDIR) .eq. 0) OPPDIR = 8

         IF ((CNTMAG .lt. SOBEL__ThreshOLD) .or.
     +     (REFLECT(SAMP+x1(CNTDIR),LINE+y1(CNTDIR))
              .gt.CNTMAG).or.
     +     (REFLECT(SAMP+x1(OppDIR),LINE+y1
              (OppDIR)).gt.CNTMAG)) THEN

            GradientDirImg__ARRAY(SAMP,LINE)=0
         ENDIF

1100     CONTINUE

1200     CONTINUE
```

```
         DO 110 LINE = 1, NLines
            DO 120 SAMP = 1, NPixels

               IF(GradientDirImg__ARRAY(SAMP,LINE).eq.0)THEN
                  Edge__ARRAY(SAMP,LINE) = 0
               ELSE
                  Edge__ARRAY(SAMP,LINE) =255
               ENDIF

120         CONTINUE
110      CONTINUE

         RETURN
         END

         INTEGER*2 FUNCTION REFLECT(X,Y)

         IMPLICIT INTEGER*2(A-Y)
         INCLUDE 'IMAGEDATA.INC'

         IF (X. EQ. 0) THEN
            X = 2
         ELSE IF (X. GT. NPixels) THEN
            X = NPixels - 1
         ENDIF

         IF (Y. EQ. 0) THEN
            Y = 2
         ELSE IF (Y. GT. NLINES) THEN
            Y = NLINES - 1
         ENDIF

         REFLECT = SOBEL__ARRAY(x,y)

         RETURN
         END
```

In an image processing application running on the Vaccelerator AP/30, a Sobel map and an image gradient are generated and applied to an image. The final step, the application of the filter to the image, is reproduced in Fortran, exactly as run on the MicroVAX II. VMS Fortran extensions are highlighted in boldface. These extensions, as well as the rest of the code, were compiled and run on the AP/30 without change; this program ran in .035 sec on the Vaccelerator AP/30, over 17 times faster than it ran on the MicroVAX II.

Figure 3: Fore! The Biomechanics golf swing analysis system, based on a MicroVAX with Vaccelerator AP/20, enables professional golf instructors to show proper swing mechanics, using the student golfer's own swing and physique as the model.

register sets, synthesis of complex operations from a limited set of instructions, and pipeline interlocks (in some RISC designs), and at the same time avoid unnecessary code expansion that can adversely impact cache utilization.

Motorola's 88000 is supported by optimizing compilers, including a suite from Green Hills Software (Glendale, CA). Together with Avalon's proprietary Central Interface Program (ACIP), which provides transparent operating system service access to applications running on the AP/30, these compilers handle VAX VMS applications "as is" (Figure 2).

Program development is best done on the VAX, using familiar editors, VAX compilers, and debuggers to get the program running. Or, users can start with an already operational application program. At this point, the user recompiles the application.

Recompiling often exposes incompatibilities between compilers. DEC VAX, IBM, CDC, and Cray machines all have their own idiosyncracies (called "extensions" by the vendor). Porting code from one to the other usually requires "sanitizing" the code to remove system-specific code and replace it either with new system-specific code or with system-independent (generic) code. AP/30 compilers, however, are designed to be compatible with the DEC VAX/VMS, Fortran, and C compilers in most respects. Virtually all of the VMS extensions offered by DEC are supported by the AP/30 compilers. Compiler switches are also compatible, thus their use is familiar to VAX/VMS users.

This compatibility means that users only need to maintain one version of source code, whether running their application on the VAX alone or on the AP/30. This is a significant advantage over other approaches that require program changes to run on new hardware platforms.

Once the program is recompiled, the AP/30 Linker links it with system libraries. The Linker is also compatible with the VAX VMS Linker, making existing link statements and option files compatible as well. The linking process produces an executable module that can be run with the standard VAX VMS "Run" command, just like a VAX application program is run. The program is paged into AP/30 memory and begins execution on the AP/30. At this point, the VAX is free to devote its full resources to other tasks not using the AP/30, resulting in faster response time for I/O applications.

To support the application's needs while running on the AP/30, a local operating system, called the Remote Kernel, handles all application system service requests that would normally (if the application were running on an unaided VAX) be handled by VMS. The Remote Kernel provides demand-paged virtual memory allocation, processor exception trap handling, local run-time library routine execution, and remote service request protocol. This last function is at the heart of the AP/30's ability to provide a true VMS environment for the application.

Handling Service Requests

Typical of service requests is the I/O operation, for example a READ. On most computer systems, READ operations are converted into subroutine calls by the compiler, and linked with the system's run-time library prior to execution. The same is true of

Table 1: Vaccelerator AP/30

a) Array Processor Versus Scalar Processor

Percent Vector Content	Array Processor Application Speedup vs. MicroVAX II	Scalar Processor Speedup To Match Vector	AP/30 Speedup
50	2x	2x	15x
75	4x	4x	15x
90	10x	10x	15x
95	20x	20x	15x
99	100x	100x	15x
99.9	1000x	1000x	15x

b) Benchmark Data

Benchmark	MicroVAX II	AP/30	AP/30 Speedup
Dhrystones	1308	37,000	28.29x
SP Whetstones	940,000	15,300,000	16.28x
DP Whetstones	624,000	7,300,000	11.70x
SP Linpack	0.169 MFLOPS	3.3 MFLOPS	19.50x
DP Linpack	0.09 MFLOPS	1.6 MFLOPS	17.78x
Digital Review CPU2 Suite	81.39 sec	5.39 sec	15.10x
Image Processing (User Application)	6.11 sec	0.35 sec	17.46x
X-Ray Powder Diffraction (User Application)	1 hour 10 min	4 min 40 sec	15.00x

c) Data Reference Latency by Type of Memory

Memory Type	Initial Latency	Subsequent Latency Per Word	When Needed
CPU Registers	0	0	All instructions
Cache	2 cycles (100 nsec)	2 cycles (100 nsec)	Not in register
Main Memory	6 cycles (300 nsec)	2 cycles (100 nsec)	Cache miss
Disk	10-50 msec	16 μsec	Not in main menu
Magnetic Tape	1-2 sec	32 msec	Archived

the AP/30. The AP/30 compiler creates a call to the appropriate READ utility subroutine. When the application makes this call, the AP/30 RTL subroutine, rather than interfacing directly with an I/O driver, constructs a command packet containing full information about the data to be read. This packet is written to the ACIP, running on the VAX, via high-speed DMA. The ACIP interprets the packet and constructs the appropriate VMS RTL call and executes it. Thus, the I/O is actually performed on the VAX, using the standard VMS file structures, RMS, and device drivers. This enables the application to remain unchanged, even down to file naming, READ statement options, and so on.

Once the I/O operation is complete, the ACIP indicates to the Remote Kernel that the AP/30 may now DMA the requested data and other information relevant to the READ back to the user's virtual address space in AP/30 memory. At this point, the application continues on the AP/30 until the next occasion for it to request a service of the VAX.

The AP/30 automatically handles over 300 VMS system services; more can be added as needed. Sometimes, users need to call subroutines that can only run on the VAX (e.g., library routines to which the user does not have source code to recompile, or device-dependent routines for which the VAX driver software must be used). In these instances, the remote service request facility provides for construction of a user-defined subroutine interface "wrapper" that enables run-time access to any VAX-resident subroutines from an application running on the AP/30. Naturally, these routines will only run at VAX speed, not AP/30 speed, and there will be some overhead. But if these routines do not represent the bulk of execution time, they can be accommodated easily by the AP/30.

Once the application is running on the AP/30, the AP/30 Profiler will show which subroutines are taking up the most CPU time. This can be used for application algorithm tuning, if desired, enabling the user to spend time only on those routines that represent significant opportunities for gain. For those who like to roll up their sleeves and really tune a program, the Profiler will provide an instruction count for each assembly language instruction executed. This exposes inner loops and enables alternate coding techniques to be used with measurable results.

Currently, Vaccelerators are being used for a variety of applications. For example, Biomechanics, Inc. (Marietta, GA) uses the Vaccelerator in a sophisticated 3D biomechanical analysis system (**Figure 3**). Applications include golf swing analysis and training, weightlifting training, robotics, motor skill training, and a range of motion simulation. **ESD:**

AVALON
Avalon Computer Systems
425 E. Colorado Street #710
Glendale, CA 91205
Phone: 818 247-2216
Fax: 818 246-7037