OpenVMS テクニカル・ジャーナル V9

アラインメント・フォルト – 性能への影響と対処方法

Guy Peleg，BRUDEN-OSSG 社 欧州・中東・アフリカ地域担当取締役

概要

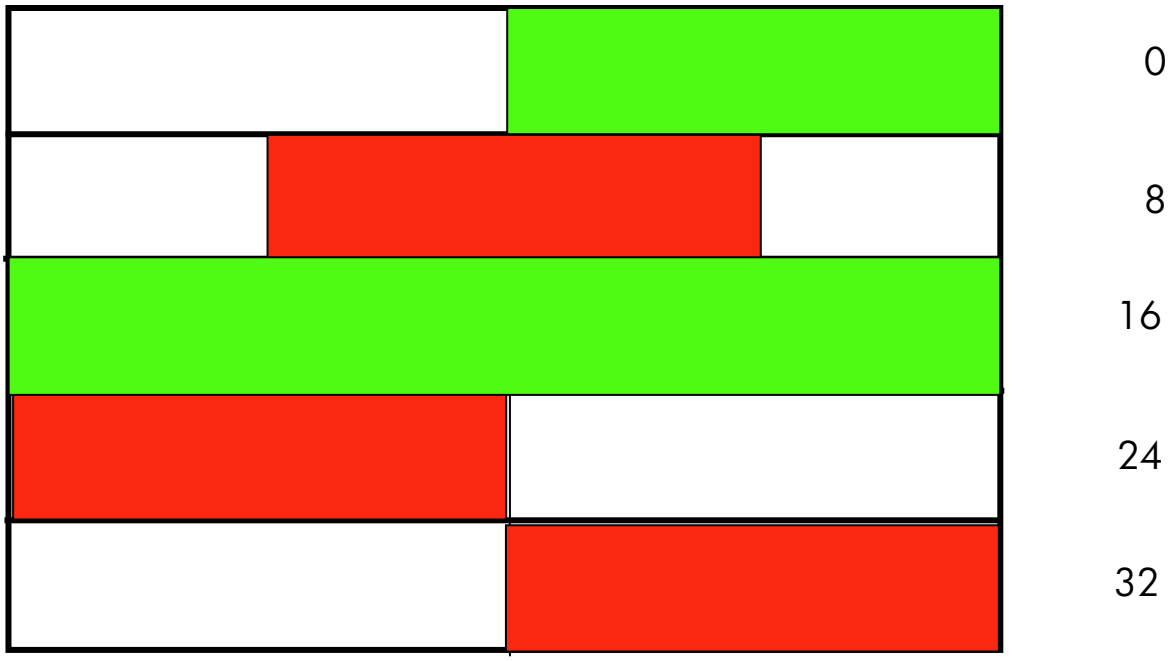
この記事では，アラインメント・フォルトとは何か，アラインメント・フォルトがアプリケーションの性能に及ぼす影響，動作中のシステムでアラインメント・フォルトを検知する方法，アラインメント・フォルトを修正するためのいくつかのアイデアについて説明します。

アラインメント・フォルトとは

AlphaServerおよびIntel® Itanium® 2 プロセッサでは，自然にアラインされたデータに高速にアクセスすることができます。自然にアラインされたデータとは，ワード境界上にあるワード・データ，ロングワード境界上にあるロングワード・データ，クォドワード境界上にあるクォドワード・データを指します。

自然にアラインされていないアドレスに置かれているクォドワード，ロングワード，ワードを，メモリとの間でロード/ストアする際には，アラインされていないアクセスを行うための一連の命令を実行するために，プロセッサは特別なルーチン（AlphaServerシステムではPALコード，Intel® Itanium® 2 システムではオペレーティング・システムルーチン）に制御を渡します。アラインされていないデータにアクセスするための特別なルーチンを実行する手続きを，アラインメント・フォルトと呼びます。

次の図は，アラインされたメモリ・アクセスとアラインされていないメモリ・アクセスの違いを示しています。



最初の段では，自然にアラインされたアドレス 0 で始まるロングワードにアクセスするため，問題はありません。2 番目の段では，アドレス 10 で始まるロングワードにアクセスします。このアドレスは自然にアラインされていない（10 を 4 で割った余りは 0 でない）ため，アラインメント・フォルトが発生します。3 番目の段では，自然にアラインされたアドレス 16 （16 を 8 で割ると余りが 0 となるため）から始まるクォドワードを読み込むため，問題は起こりません。4 番目の段では，アドレス 28 から始まるクォドワードにアクセスします。アドレス 28 はクォドワードにアラインされていないため，アラインメント・フォルトが発生します。

アラインメント･フォルトに注意すべき理由

アラインされていないデータをコンパイラが検出できる場合，AlphaServer システムで通常は 3 つの命令で実行できる処理が，15 命令必要となります。これらの命令すべてがメモリにアクセスするわけではありませんが，命令ストリームの性能はトータルで 3 倍遅いという結果になります。コンパイラが問題を修正できない場合は，実行時にアラインメント・フォルトが発生します。アラインメント・フォルト・ハンドラは，自然にアラインされたデータにアクセスする場合よりも，10～20 倍低速です。

Intel® Itanium® 2 システムの動作もAlphaServerと同様ですが，アラインメント・フォルトが発生すると，自然にアラインされたデータにアクセスする場合よりも数百～数千倍低速になります。これは，アラインメント・フォルトが，PALコード（ファームウェア）ではなくオペレーティング・システムで処理されるためです。また，アラインメント・フォルトを処理するためにシステム全体に影響が出ます。これは，スピンロック（MMG）とそれに付随するMP同期時間が必要になるためです。

簡単な例を見てみましょう。次のプログラムは，1 GB の仮想メモリを P2 空間から割り当て，50,000,000 個のクォドワードをランダムに増分します。

```
$ ty aligned.c
#include <far_pointers>
#include <gen64def>
#include <ints>
#include <starlet>
#include <stdio>
#include <stdlib>
```

```
#include <lib$routines.h>
#include <unistd.h>
#include <stsdef>

#define random_key(upper_bound) (abs (random () % upper_bound))

void main()
{
int             NumberOfBytes    =      1000000000;     // 1GB (SI 接頭辞表記)
int             status;
VOID_PQ         MappedVA;
INT64_PQ        RandomVA;


        lib$init_timer();               // タイマを初期化

        //
        // P2 空間から 1GB を割り当て
        //
        status = lib$get_vm_64 (&NumberOfBytes, &MappedVA);

        if (!$VMS_STATUS_SUCCESS(status))
        {
                lib$signal (status);
                return;
        }

        RandomVA = MappedVA;

        for (int i=0; i<50000000; i++)
        {

                // ランダムなクォドワードを増分
                RandomVA [random_key((100000000/8) -1)] ++ ;

        }

        //
        // VM を解放
        //
        status = lib$free_vm_64 (&NumberOfBytes, &MappedVA);

        if (!$VMS_STATUS_SUCCESS(status))
        {
                lib$signal (status);
                return;
        }

        lib$show_timer();
}

$! プログラムを実行 – rx2600 1.3 GHZ
```

```
$ cc/pointer=long aligned
$ link aligned
$ r aligned
 ELAPSED:    0 00:00:18.97  CPU: 0:00:18.97  BUFIO: 0  DIRIO: 0  FAULTS: 713808
$
```

1.3 GHz の Integrity rx2600 サーバ上で 50,000,000 個のランダムなクォドワードを増分するのに，18.97 秒かかりました。

では，上記のプログラムを，アラインされていないポインタを使用して 50,000,000 個のクォドワードを増分するように変更してみます。

```
$ ty not_aligned.c
#include <far_pointers>
#include <gen64def>
#include <ints>
#include <starlet>
#include <stdio>
#include <stdlib>
#include <lib$routines.h>
#include <unistd.h>
#include <stsdef>

#define random_key(upper_bound) (abs (random () % upper_bound))

void main()
{
int             NumberOfBytes    =      1000000000;     // 1GB (SI 接頭辞表記)
int             status;
VOID_PQ         MappedVA;
INT64_PQ        RandomVA;


        lib$init_timer();               // タイマを初期化

        //
        // P2 空間から 1GB を割り当て
        //
        status = lib$get_vm_64 (&NumberOfBytes, &MappedVA);

        if (!$VMS_STATUS_SUCCESS(status))
        {
                lib$signal (status);
                return;
        }

        //
        // アラインされていない状態にポインタを変更
        //
        RandomVA = (INT64_PQ)((char *) MappedVA + 1);
```

```
        for (int i=0; i<50000000; i++)
        {

                // ランダムなクォドワードを増分
                RandomVA [random_key((100000000/8) -1)] ++ ;

        }

        //
        // VM を解放
        //
        status = lib$free_vm_64 (&NumberOfBytes, &MappedVA);

        if (!$VMS_STATUS_SUCCESS(status))
        {
                lib$signal (status);
                return;
        }

        lib$show_timer();
}

$ cc/pointer=long not_aligned.c
$ link not_aligned
$ r not_aligned
 ELAPSED:    0 00:03:45.62  CPU: 0:03:45.53  BUFIO: 0  DIRIO: 0  FAULTS: 200027
$
```

同じ 1.3 GHz の Integrity rx2600 サーバで 50,000,000 個のアラインされていないクォドワードを増分するのに，3 分 45 秒かかりました。

この簡単なテスト・プログラムでは，アラインされていないデータにアクセスすると，性能が12倍以上低下するという結果が出ました。

アラインメント･フォルトの検出

アラインメント・フォルトがいかに性能に悪影響を与えるか分かったと思いますので，次に，アラインメント・フォルトを検出するために OpenVMS に用意されている以下のツールについて説明します。

- MONITOR ALIGN （V8.3）
- SDA の FLT 拡張
- シンボリック・デバッガ

MONITOR ALIGN

OpenVMS V8.3 では，monitor ユーティリティに新しいクラスが追加されています。align クラスは，システム全体で現在発生しているアラインメント・フォルトを監視し，モードごとに出力します。

NOT_ALIGNED プログラムを実行した場合の出力例を以下に示します。

```
$ monitor align/int=1

                          OpenVMS Monitor Utility
                        ALIGNMENT FAULT STATISTICS
                              on node IT13
                          21-NOV-2006 01:50:13.26

                                      CUR        AVE        MIN        MAX


    Kernel Fault Rate                0.00       0.44       0.00       4.00
    Exec   Fault Rate                0.00       0.00       0.00       0.00
    Super  Fault Rate                0.00       0.00       0.00       0.00
    User   Fault Rate           445492.00  220809.67       0.00  445492.00

    Total  Fault Rate           445492.00  220810.12       0.00  445492.00
```

このテスト・プログラムでは，1 秒あたり 445,000 回以上のアラインメント・フォルトが，すべてユーザ・モードで発生しています。

MONITOR ALIGNを使用すると，現在システムで発生しているアラインメント・フォルトの概略を知ることができます。このツールは，アラインメント・フォルトを検出し，アラインメント・フォルトによってシステムの性能が低下していることを知るのに役立ちます。しかし，MONITOR ALIGNでは，アラインメント・フォルトが起きているプロセスやプログラムに関する情報を得ることはできません。MONITOR ALIGNの目的は，アラインメント・フォルトによって性能が低下しているかどうかを確認することです。アラインメント・フォルトが起きているプロセスやプログラムを調査するためには，別のツールを使用する必要があります。なお，MONITOR ALIGNは，現在Intel® Itanium® 2 システムでのみ利用できます。

SDA の FLT 拡張

システムでアラインメント・フォルトの問題が発生していることがわかったら，次のステップは，アラインメント・フォルトが発生している場所を特定することです。SDAのFLT拡張は，アラインメント・フォルトを検出しログに記録するための非常に強力なツールです。ログの取得機能を有効にしている間、発生した各アラインメント・フォルトに関して，発生した時刻，発生したCPU，アラインされていない仮想アドレス，アクセス・モード，アクセスIDが記録されます。開発者は，この情報を使用して，アプリケーション内でアラインメント・フォルトが発生した正確な場所を特定することができます。FLT拡張は，AlphaServerシステムとIntel® Itanium® 2 システムの両方で利用できます。

FLT の使用例を次に示します。

```
$ ana/sys

OpenVMS system analyzer

FLT拡張をロード

SDA> flt load
FLT$DEBUG load status = 00000001
```

トレース開始...

```
SDA> flt start trace
Tracing started...
```

要約を表示

```
SDA> flt show trace/sum

Fault Trace Information: (at 21-NOV-2006 02:07:21.87, trace time 00:00:00.190015)
-------------------------------------------------------------------------------
Exception PC              Count   Exception PC                          Module
Offset
-----------------   ------------   -----------------------------------  ---------------------------------
-
00000000.000103D1          39384   SYS$K_VERSION_16+00391
00000000.000103E1          39383   SYS$K_VERSION_16+003A1
```

2 つのプログラム・カウンタが表示され，**PC 103D1** と **103E1** を指しており，各 **PC** で **39834** 回のフォルトが発生しています。次に，詳細を表示して犯人を探します。
トレース・バッファ内の個々のエントリについて詳細を表示することができます。

```
SDA> flt show trace

Unaligned Data Fault Trace Information:
---------------------------------------
Timestamp             CPU  Exception PC                                  Unaligned VA       Access
EPID    Trace Buffer
---------------------- ---  --------------------------------------------  -----------------  ------  ----
----  -----------------
21-NOV 02:08:22.002794  00  00000000.000103E1 SYS$K_VERSION_16+003A1       00000000.840BECF9  User
2160057F  FFFFFFFF.7E4E86C0
21-NOV 02:08:22.002791  00  00000000.000103D1 SYS$K_VERSION_16+00391       00000000.840BECF9  User
2160057F  FFFFFFFF.7E4E8658
21-NOV 02:08:22.002789  00  00000000.000103E1 SYS$K_VERSION_16+003A1       00000000.84617049  User
2160057F  FFFFFFFF.7E4E85F0
21-NOV 02:08:22.002786  00  00000000.000103D1 SYS$K_VERSION_16+00391       00000000.84617049  User
2160057F  FFFFFFFF.7E4E8588
21-NOV 02:08:22.002784  00  00000000.000103E1 SYS$K_VERSION_16+003A1       00000000.8252A0E1  User
2160057F  FFFFFFFF.7E4E8520
21-NOV 02:08:22.002781  00  00000000.000103D1 SYS$K_VERSION_16+00391       00000000.8252A0E1  User
2160057F  FFFFFFFF.7E4E84B8
21-NOV 02:08:22.002779  00  00000000.000103E1 SYS$K_VERSION_16+003A1       00000000.850E3241  User
2160057F  FFFFFFFF.7E4E8450
21-NOV 02:08:22.002776  00  00000000.000103D1 SYS$K_VERSION_16+00391       00000000.850E3241  User
2160057F  FFFFFFFF.7E4E83E8
21-NOV 02:08:22.002774  00  00000000.000103E1 SYS$K_VERSION_16+003A1       00000000.84CD53D1  User
2160057F  FFFFFFFF.7E4E8380
21-NOV 02:08:22.002771  00  00000000.000103D1 SYS$K_VERSION_16+00391       00000000.84CD53D1  User
2160057F  FFFFFFFF.7E4E8318


……
```

すべてのエントリが，**ID** が **2160057F** のプロセスを指しています。
そのプロセスが実行しているイメージを確認します。

```
SDA> set proc/id=2160057F
SDA> show proc/image
```

```
Process index: 017F   Name: Faulty           Extended PID: 2160057F
------------------------------------------------------------------

                          Process activated images
                          ------------------------

             Image Name                    Type       IMCB          GP
------------------------------------ ------------ -------- -----------------
NOT_ALIGNED                          MAIN         7FE89290 00000000.00240000
DCL                                  MRGD     SHR 7FE88BD0 00000000.7B0D8000
LIBRTL                               GLBL     SHR 7FE8BC10 00000000.7B546000
LIBOTS                               GLBL     SHR 7FE8A690 00000000.7B560000
CMA$TIS_SHR                          GLBL     SHR 7FE88010 00000000.7B73C000
DPML$SHR                             GLBL     SHR 7FE88270 00000000.7B904000
DECC$SHR                             GLBL     SHR 7FE883A0 00000000.7BB10000
SYS$PUBLIC_VECTORS                   GLBL         7FE886C0 FFFFFFFF.8CA00400
SYS$BASE_IMAGE                       GLBL         7FE88920 FFFFFFFF.8CA24E00

Total images = 9                     Pages allocated = 322
SDA> map 0103E1
Image                                      Base              End          Image Offset
NOT_ALIGNED
    Code                             00000000.00010000 00000000.0001059F 00000000.000103E1
SDA>
```

すべてのアラインメント・フォルトが，**NOT_ALIGNED** イメージを実行している，
"Faulty"プロセスで発生しています。次の手順は，リストを参照し，
オフセット **103E1** にある問題のコードを調べることです。

リストを参照する前に，イメージにトレースバック情報が格納されており，そのイメージがシステム空間にあれば，**FLT** 拡張は，問題のある **PC** を解釈することができます。まず，**NOT_ALIGNED.EXE** を常駐イメージとしてインストールします。これにより，イメージはシステム空間にコピーされます。

```
SDA> flt stop trace
SDA> spawn instal add/resi SYS$SYSDEVICE:[PELEG]NOT_ALIGNED
SDA> flt start trace
Tracing started...
SDA> flt show trace/summ

Fault Trace Information: (at 21-NOV-2006 02:13:23.77, trace time 00:00:00.190637)
------------------------------------------------------------------------------
Exception PC             Count   Exception PC                        Module
Offset
----------------   ------------   ---------------------------------  ---------------------------------
-
FFFFF802.11EFE3D1          39384   NOT_ALIGNED+103D1                  NOT_ALIGNED
000103D1
                                   NOT_ALIGNED + 000003D1 / main + 000002D1
FFFFF802.11EFE3E1          39383   NOT_ALIGNED+103E1                  NOT_ALIGNED
000103E1
                                   NOT_ALIGNED + 000003E1 / main + 000002E1
```

```
SDA>
```

再度トレースを開始します。要約表示に，イメージ内のフォルトが発生した正確な場所が表示されます。この例では，**NOT_ALIGNED.EXE** 内の，ルーチン **main+2D1** と **main +2E1** です。

NOT_ALIGNED.LIS のリスト内の該当する部分を見てみます。

```
001000000046     0240         (pr6)  break.m 1048577
00C7080121C0     0241                setf.sig f7 = r9
018402242200     0242                cmp4.lt pr8, pr0 = i, r34 ;;             // pr8, pr0 = r33, r34
// 023707
                         }
                         { .mfi
00C708006180     0250                setf.sig f6 = r3                      // 023711
000008000000     0251                nop.f   0
000008000000     0252                nop.i   0 ;;
                         }
                         { .mfi
000008000000     0260                nop.m   0
0000E000E240     0261                fcvt.xf f9 = f7
000008000000     0262                nop.i   0
                         }
                         { .mfi
000008000000     0270                nop.m   0
0000E000C200     0271                fcvt.xf f8 = f6
000008000000     0272                nop.i   0 ;;
                         }
                         { .mfi
000008000000     0280                nop.m   0
000630910280     0281                frcpa.s1 f10, pr6 = f8, f9
000008000000     0282                nop.i   0 ;;
                         }
                         { .mfi
000008000000     0290                nop.m   0
018448A021C6     0291            (pr6)  fnma.s1 f7 = f10, f9, f1
000008000000     0292                nop.i   0 ;;
                         }
                         { .mfi
000008000000     02A0                nop.m   0
010438A142C6     02A1            (pr6)  fma.s1  f11 = f10, f7, f10
000008000000     02A2                nop.i   0
                         }
                         { .mfi
000008000000     02B0                nop.m   0
010438700186     02B1            (pr6)  fma.s1  f6 = f7, f7, f0
000008000000     02B2                nop.i   0 ;;
                         }
                         { .mfi
000008000000     02C0                nop.m   0
010450800106     02C1            (pr6)  fma.s1  f7 = f8, f10, f0
000008000000     02C2                nop.i   0 ;;
                         }
                         { .mfi
000008000000     02D0                nop.m   0
010430B16286     02D1            (pr6)  fma.s1  f10 = f11, f6, f11
000008000000     02D2                nop.i   0 ;;
                         }
```

```
                        { .mfi
000008000000    02E0              nop.m   0
0184389102C6    02E1        (pr6)  fnma.s1 f11 = f9, f7, f8
000008000000    02E2              nop.i   0
                        }
                        { .mfi
000008000000    02F0              nop.m   0
018448A02186    02F1        (pr6)  fnma.s1 f6 = f10, f9, f1
000008000000    02F2              nop.i   0 ;;
                        }
                        { .mfi
```

main + 2D1 と main + 2E1 は，ソース内の行番号 23711 を指しています。

```
     1   23707        for (int i=0; i<50000000; i++)
     2   23708        {
     2   23709
     2   23710              // Increment a random Quadword
     2   23711              RandomVA [random_key((100000000/8) -1)] ++ ;
     2   23712
     1   23713        }
```

次の手順は，アラインされていないメモリ・アクセスをしないようにプログラムを修正することです。

シンボリック･デバッガ

シンボリック・デバッガを使用することで，アラインメント・フォルトを検出することができます。SET BREAK/UNALIGN コマンドを実行すると，アラインメント・フォルトが発生するたびにデバッガが停止します。その際，アラインメント・フォルトが発生した仮想アドレス，現在の PC，フォルトが発生したソース行が表示されます。

```
$ run/debug not_aligned

         OpenVMS I64 Debug64 Version V8.3-009

%DEBUG-I-INITIAL, Language: C, Module: NOT_ALIGNED
%DEBUG-I-NOTATMAIN, Type GO to reach MAIN program

DBG> set break/unaligned
DBG>
* SRC: module NOT_ALIGNED -scroll-
source**********************************************************************************
****
 23703:         // アラインされていない状態にポインタを変更
 23704:         //
 23705:         RandomVA = MappedVA + 1;
 23706:
 23707:         for (int i=0; i<50000000; i++)
 23708:         {
 23709:
 23710:                 // ランダムなクォドワードを増分
->3711:                 RandomVA [random_key((100000000/8) -1)] ++ ;
 23712:
```

```
 23713:         }
 23714:
 23715:         //
 23716:         // Free VM
 23717:         //
 23718:         status = lib$free_vm_64 (&NumberOfBytes, &MappedVA);
 23719:
* OUT -
output*****************************************************************************************
**
Unaligned data access: virtual address = 0000000081E0E7E1, PC = 00000000000103E2
break on unaligned data trap preceding NOT_ALIGNED\main\%LINE 23711+402
 23711:                 RandomVA [random_key((100000000/8) -1)] ++ ;


DBG>
```

注：FLT ユーティリティの使用中は，SET BREAK/UNALIGNED は使用できません。FLT が動作しているときに，デバッガを使ってアラインメント・フォルトをレポートさせようとすると失敗し，次のエラーが出力されます。

```
DBG> set break/unalign
%SYSTEM-E-AFR_ENABLED, alignment fault reporting already enabled
-FOR-W-NOMSG, Message number 00189E80
DBG>
```

アラインメント･フォルトを修正するためのガイドライン

アプリケーションが完璧であれば，完全にアラインメント・フォルトを避けることができますが，物事は常に完璧であるとは限りません。アラインされていないデータを使っているモジュールが，アラインされたデータを想定している別のモジュールのルーチンを呼び出す場合にも，アラインメント・フォルトが発生する可能性があります。アラインメント・フォルトは，AlphaServerシステムでも有害ですが，Intel® Itanium® 2 システムでは非常に有害であることを忘れないでください。

アラインメント・フォルトの中には，修正が簡単なものもあれば，修正が非常に難しいものや，不可能に近いものもあります。アラインメント・フォルトを修正するための最も一般的な方法は以下のとおりです。

- データをアラインする。
- アクセスしようとしているデータがアラインされていない（またはその可能性がある）というヒントをコンパイラに与える。
- アラインされているバッファにデータをコピーする

データをアラインする

データをアラインするのは，アラインメント・フォルトを避けるための最善の方法です。

現在のコンパイラは高機能であり，ほとんどの場合アラインメント・フォルトの問題を検出し，複数のロード，シフト，マスク操作でデータにアクセスするためのコードを追加します。
データをアラインすることが不可能だったり，現実的でないこともあります。そのような例としては，システム間でデータを転送する場合や，ファイル内の固定レイアウトのレコードを読み書きする場合が挙げられます。

データ構造内のフィールドが自然にアラインされていることを確認してください。C や C++などのコンパイラは，デフォルトでフィールドを自然にアラインします。MACRO では，.align [quad|long]を使用します。SDL では，basealign [quad|long]を使用します。

コンパイラにヒントを与える

プログラミング言語によっては，アラインメントされていないことを前提としたコードを生成する宣言修飾子をサポートしていることがあります。このようなコードでは，アラインされていないデータをテストし，アラインメント・フォルトが発生しないような方法でデータを操作します。

このような宣言修飾子には，以下のものがあります。

- __unaligned (C)
- .set_registers unaligned=<Rx> (Macro)
- align(x) (Bliss32/Bliss64)
- aligned(x) (Pascal)

このオプションを使用することで，アラインメント・フォルトが無くなりますが，アラインされたデータにアクセスするコードは，通常よりも遅くなります。

コンパイラに対して，データがアラインされていない可能性があるというヒントを与えた場合に生成される追加のコードは，*アラインメント・フォルトが発生した場合よりもはるかに効率よく動作します。*

では，NOT_ALIGNED プログラムを変更して，ランダムなデータへのポインタがアラインされていないことを宣言します。

```
$ ty not_aligned.c
#include <far_pointers>
#include <gen64def>
#include <ints>
#include <starlet>
#include <stdio>
#include <stdlib>
#include <lib$routines.h>
#include <unistd.h>
#include <stsdef>

#define random_key(upper_bound) (abs (random () % upper_bound))

void main()
{
int             NumberOfBytes    =      1000000000;     // 1GB (SI 接頭辞表記)
int             status;
VOID_PQ         MappedVA;
INT64_PQ        RandomVA;


        lib$init_timer();               // タイマを初期化
```

```
        //
        // P2 空間から 1GB を割り当て
        //
        status = lib$get_vm_64 (&NumberOfBytes, &MappedVA);

        if (!$VMS_STATUS_SUCCESS(status))
        {
                lib$signal (status);
                return;
        }

        //
        // アラインされていない状態にポインタを変更
        //
        RandomVA = (INT64_PQ)((char *) MappedVA + 1);

        for (int i=0; i<50000000; i++)
        {

                // ランダムなクォドワードを増分 – ポインタを unaligned として宣言
                __int64 __unaligned *MyData = &RandomVA [random_key((100000000/8) -1)];
                *MyData = *MyData + 1;


        }

        //
        // VM を解放
        //
        status = lib$free_vm_64 (&NumberOfBytes, &MappedVA);

        if (!$VMS_STATUS_SUCCESS(status))
        {
                lib$signal (status);
                return;
        }

        lib$show_timer();
}
$ cc/pointer=long not_aligned.c
$ link not_aligned
$ r not_aligned
 ELAPSED:    0 00:00:20.74  CPU: 0:00:20.67  BUFIO: 0  DIRIO: 0  FAULTS: 703741
$
```

変更した結果，プログラムは **20.74** 秒で完了しました。これは，データがアラインされていないことをコンパイラが想定していなかった場合の **3** 分 **45** 秒に比べると大きな改善です。

データをコピーする

アラインメント・フォルトを修正するための最後の方法は，アラインされているバッファにデータをコピーすることです。この方法は，データ自体はアラインされているものの，データが格納されているバッファがアラインされていない場合に有効です。

移動が必要なデータの量が少なく，何度も参照される場合は，データをコピーする方法は有効です。しかし，移動するデータの量が多く，参照回数が少ない場合は，アラインメント・フォルトが少し発生したとしても，データをそのままにするほうが効率的です。

まとめ

性能の面からは，アラインメント・フォルトの影響はAlphaServerシステムでは大きく，Intel® Itanium® 2 システムでは非常に大きくなります。Intel® Itanium® 2 システムで優れた性能を得るためには，アラインメント・フォルトを解決する必要があります。OpenVMSでは，MONITOR ALIGN コマンド，SDAのFLT拡張，デバッガを使用してアラインメント・フォルトを監視することができます。

アラインメント・フォルトを避けるには，データを自然にアラインするか，アラインされていないものとしてポインタを宣言するか，有効な場合にはアラインされているバッファにデータをコピーします。

詳細情報

OpenVMS テクニカル・ジャーナルの最新号を入手するには，次の URL にアクセスしてください。
http://h41379.www4.hpe.com/openvms/journal/

著者について

Guy Peleg 氏は，昨年の 9 月に BRUDEN-OSSG 社に入社しました。彼は，技術スタッフの上級メンバであり，EMEA（欧州・中東・アフリカ地域）の取締役でもあります。BRUDEN-OSSG に入社する前は，OpenVMS のエンジニアリング・グループで，ソフトウェア・エンジニアとしてさまざまなユーティリティを開発していました。Peleg 氏は OpenVMS を Integrity サーバ・プラットフォーム(IPF)に移植したチームのメンバであり，LMF の IA64 への移植，EDCL プロジェクト，IPF 上の各種の仮想化プロジェクトのリーダーを務めました。エンジニアリング・グループのメンバになる前は，Compaq/DEC の現場部門で顧客サポートとコンサルティングを担当しており，OpenVMS ユーザへの貢献で世界的に知られています。Peleg 氏はさまざまな技術プレゼンテーションを行った経験を持ち，OpenVMS システムのテクニカル・ジャーナルに寄稿してきました。彼のプレゼンテーションは面白く，非常にためになります。