FreeBSDのgpartコマンドでGPTパーティションを定義する

2025/08/23
現在の状況の追加を勧められたのでちょこちょこと補足追加実施。

2013/02/28
おまけにLBAの話やら何やらを追記。

2013/02/27
GPTでディスクを設定するときのメモ。すぐ忘れそうになる…

USB外付けディスク接続

dmesgでディスクの情報を確認する。

```
ugen5.2: <vendor 0x04b4> at usbus5
uhub6: <vendor 0x04b4 product 0x6560, class 9/0, rev 2.00/0.07, addr 2> on
usbus5
uhub6: 4 ports with 4 removable, self powered
usbd_set_config_index: could not read device status: USB_ERR_SHORT_XFER
ugen5.3: <I-O DATA DEVICE, INC.> at usbus5
umass0: <MSC Bulk-Only Transfer> on usbus5
umass0:  SCSI over Bulk-Only; quirks = 0x0100
umass0:6:0:-1: Attached to scbus6
da0 at umass-sim0 bus 0 scbus6 target 0 lun 0
da0: <I-O DATA HDCA-U 133B> Fixed Direct Access SCSI-2 device
da0: 40.000MB/s transfers
da0: 1907729MB (3907029168 512 byte sectors: 255H 63S/T 243201C)
```

デバイス da0 で認識されてる。

GPTスキーマでディスクを登録する

FreeBSDにはGEOMというフレームワークがあり、これを通すとディスクの種類種別を問わず共通のI/Fで操作が可能になる RAIDの構成もできたりする。

GEOMさんにこのディスクをGPTで使用する旨を教える GPTスキーマを作成する、というらしい。

```
# gpart create -s gpt da0
gpart: geom 'da0': File exists

# gpart show da0
=>        63  3907029105  da0  MBR  (1.8T)
          63  3907024002    1  ntfs  (1.8T)
```

```
  3907024065        5103       - free -  (2.5M)

# gpart delete -i 1 da0
da0s1 deleted
# gpart create -s gpt da0
gpart: geom 'da0': File exists
# gpart show da0
=>        63  3907029105  da0  MBR  (1.8T)
          63  3907029105       - free -  (1.8T)

#
```

NTFSでフォーマットされておりMBRスキーマが設定されていたのでGPTスキーマでの設定ができなかった模様。

一度 da0 の全てを消してやり直す。

```
# gpart destroy da0
da0 destroyed
# gpart create -s gpt da0
da0 created
# gpart show da0
=>        34  3907029101  da0  GPT  (1.8T)
          34  3907029101       - free -  (1.8T)

#
```

GEOMさんがGPTでda0を認識してくれた模様。

次は確保したスキーマにFreeBSDのパーティションを追加する。

パーティション追加

タイプを freebsd-ufs にしてパーティションを追加する。

```
# gpart add -t freebsd-ufs -b 40 -s 1907729mb da0
da0p1 added
# gpart show da0
=>        34  3907029101  da0  GPT  (1.8T)
          34           6       - free -  (3.0k)
          40  3907028992    1  freebsd-ufs  (1.8T)
  3907029032         103       - free -  (51k)

=>        34  3907029101  ufsid/512bd541ae9bcdf8  GPT  (1.8T)
          34           6                          - free -  (3.0k)
          40  3907028992                       1  freebsd-ufs  (1.8T)
  3907029032         103                          - free -  (51k)

# newfs -S 4096 /dev/da0p1
/dev/da0p1: 1907729.0MB (3907028992 sectors) block size 32768, fragment size
```

```
4096
        using 3048 cylinder groups of 626.09MB, 20035 blks, 80256 inodes.
super-block backups (for fsck_ffs -b #) at:
 192, 1282432, 2564672, 3846912, 5129152, 6411392, 7693632, 8975872,
10258112, 11540352, 12822592, 14104832, 15387072, 16669312, 17951552,
19233792,
 20516032, 21798272, 23080512, 24362752, 25644992, 26927232, 28209472,
29491712, 30773952, 32056192, 33338432, 34620672, 35902912, 37185152,
38467392,
 39749632, 41031872, 42314112, 43596352, 44878592, 46160832, 47443072,
48725312, 50007552, 51289792, 52572032, 53854272, 55136512, 56418752,
57700992,

-- 中略 --

 3862107072, 3863389312, 3864671552, 3865953792, 3867236032, 3868518272,
3869800512, 3871082752, 3872364992, 3873647232, 3874929472, 3876211712,
3877493952,
 3878776192, 3880058432, 3881340672, 3882622912, 3883905152, 3885187392,
3886469632, 3887751872, 3889034112, 3890316352, 3891598592, 3892880832,
3894163072,
 3895445312, 3896727552, 3898009792, 3899292032, 3900574272, 3901856512,
3903138752, 3904420992, 3905703232, 3906985472
#
```

フォーマットも済んだので、あとは /dev/da0p1 をマウントして利用するだけ。

その他

※以降は正しくない記述が含まれております

gpart add の -b で指定する開始ブロック番号(セクタ番号)については諸説あり。

AFTのHDDだと1セクタあたり旧来の512バイトから8倍の4096バイトになっている□HDDのファームウエアで論理的512バイトセクタ⇒物理4096バイトセクタへの変換を行っているとのこと。たとえば、論理ブロック番号0～7番が物理ブロック番号0番に格納、論理ブロック番号8～15番が物理ブロック番号1番に格納、という具合□ AFTを知らないOSだとアクセス効率が極端に悪くなる。 ※物理4096バイトセクタの途中に配置されるような論理512バイトセクタをアクセスすることが多くなる。

事前にAFTのHDDだと判明している場合は、OSが使用する論理セクタの開始位置(番号)を物理セクタの先頭位置(最初の512バイト部分)になるよう調整することでアクセス効率改善が可能な場合がある。

MBRだとブロック番号63からパーティションが確保されている。 0～62番がブートや管理のための領域だとすると、これらはAFTの物理ブロック番号0～7に格納されていることになる。

物理	論理	格納論理ブロック数
0	0□ 7	8 (0, 1, 2, 3, 4, 5, 6, 7)
1	8□15	8 (8, 9,10,11,12,13,14,15)
2	16□23	8 (16,17,18,19,20,21,22,23)
3	24□31	8 (24,25,26,27,28,29,30,31)

物理	論理	格納論理ブロック数
4	32□39	8 (32,33,34,35,36,37,38,39)
5	40□47	8 (40,41,42,43,44,45,46,47)
6	48□55	8 (48,49,50,51,52,53,54,55)
7	56□63	8 (56,57,58,59,60,61,62,63)
8	64□71	8 (64,65,66,67,68,69,70,71)

論理ブロック番号63は、物理ブロック番号7の最後の512バイトに対応することになる。 論理ブロック番号63を参照するたびに、物理ブロック7番を読み込み、最後の512バイトを取り出さなくてはいけない。

もしアクセスする論理ブロック単位が8個(4096バイト)だったら、物理ブロック番号7と物理ブロック番号8を読み込み、7番の最後の512バイトと、8番の先頭3584バイトを取り出すことになる。非効率的。開始論理ブロック番号を物理ブロック番号の先頭512バイトにあわせれば、1物理ブロック読み出しで8論理ブロックを取り出せて効率的。

ツールで設定可能な8の倍数の論理ブロック番号からパーティションを定義しよう案

上記表でいえば、物理ブロック番号8最初の512バイト部分に開始論理ブロック番号として論理ブロック番号64を割り当てられればいい。論理ブロック番号63は空きブロックとなるけど512バイトくらいなら目をつぶれる。 さすがに論理ブロック番号64はツールで設定ができないけど、ツールで設定可能な範囲で最小の8の倍数の論理ブロック番号に設定すればいけるんじゃないだろうか、という案。

トラック先頭にあたる論理ブロック番号からパーティションを定義しよう案

どうせなら開始物理ブロックがトラックの先頭から始まるようにすればさらに効率上がるじゃない。

1トラック63セクタなら、1セクタが8論理ブロックを含むので、 63×8＝504、だから論理ブロック番号504がトラック先頭の物理ブロック内で1番目の論理ブロックとなるはず、という案。

つまり、0〜503番が最初のトラックに割り当てられた論理ブロック番号だから、次のトラックの先頭論理ブロック番号は504のはず、だというお話。

シリンダ先頭トラックにあたる論理ブロック番号からパーティションを定義しよう案

さらに、開始ブロックがシリンダの先頭トラックからにすれば根拠はないけどすっきりしてるし(以下略

255ヘッドで1トラック63セクタなら、255×63×8＝128,520、だから論理ブロック番号128,520がシリンダの先頭トラック内最初の物理ブロック内で1番目の論理ブロックになるはず、という案。

つまり、0〜128,519番が最初のシリンダに割り当てられた論理ブロック番号だから、次のシリンダの先頭トラックの先頭論理ブロック番号は128,520のはず、だというお話。

ただ□HDDによって本当にこの考え方でいいのかは変わってくる

ターゲットにした論理ブロック番号がHDDの物理的構造のどこに配置されるかなんて分らない。 そんなものはファームウエアしかあずかり知らないことだ。

扱おうとしている製品の仕様を把握しているならともかく、実際に希望の物理的配置をできたか否かについて実際にアクセス速度等の計測をしてみないとなんともいえないと思う。

補足: AFTと4KiBセクタの最適化

AFT（512バイト論理/4KiB物理セクタ）は2010年代のHDDで一般的だったけど2025年では4KiBネイティブセクタのHDD/SSDが増えています。
SSDでは物理セクタの概念がなくなり効率問題はほぼ解消、FreeBSDでZFSを使う場合はプール作成時に`ashift=12`（4KiB）を指定するとパフォーマンスが最適化されます:

```
zpool create -o ashift=12 mypool /dev/da0p1
```

また、`newfs -S 4096`で4KiBセクタに合わせたフォーマットが推奨されます。

補足: FreeBSD 14.xでのgpartの進化

FreeBSD 14.xの`gpart`は、自動アライメント（例: `-a 4k`で4KiBセクタ最適化）やラベル設定（`-l`）をサポートしてたよ！

```
gpart add -t freebsd-ufs -a 4k -l mydata -s 1907601m da0
```

おまけ

補足: SSDとNVMeの時代におけるLBA

2013年当時はHDDが主流で、CHSは互換性のための仮想値だったけど、2025年現在はSSDやNVMe接続のストレージが一般的です。 SSDでは物理的な円盤やヘッドが存在しないのでCHSは廃止されLBAが標準的なアドレス指定方式です。
NVMeでは名前空間（Namespace）により1台のドライブを論理的に分割でき、セクタサイズも4KiBや16KiBが選択可能で、パフォーマンス最適化が図られています。
FreeBSD 14.x以降ではNVMeデバイス（例: /dev/nvme0ns1）を直接操作でき、gpartコマンドもNVMeに対応しています。

CHS管理

HDDは昔、CHSでその記録位置を指定していた。

- :シリンダ
- :ヘッド
- :セクタ
- :トラック

概念的な説明をすると、

- 最小の記録単位はセクタ、当時はセクタは512バイト。
- セクタはトラックに配置され、トラックには複数のセクタが記録される。
- トラックは、HDD内の磁性体が塗布された円盤に記録される。
- 円盤には中心から同心円を描くように複数のトラックが記録されている。中心に近いトラックも、円周に近いトラックも同じセクタ数。

- 円盤は複数枚入っており、各円盤の同じ位置にあるトラックのまとまりをシリンダと呼ぶ。
- 各円盤には磁性面から記録を読み出すヘッドが設置されており、ヘッドの数だけ円盤があると。

HDDのラベルに貼られているCHSの各値は、そのHDDのシリンダ・ヘッド・セクタの最大値を示している。

シリンダ□個、ヘッド□個、セクタ□個(□個/トラック)のHDDがあったとすると、

- 1シリンダには□個のトラックが含まれる。
- 1トラックには□個のセクタが含まれる。
- このHDDには□□×□×□個のセクタが記録できる。

ということ。そしてこの指定範囲内で「シリンダ3番、ヘッド0番、セクタ7番、に書き込むぞー」とかやっていたわけだ。

時は流れHDDが大容量化してくるとCHSを使った記録位置指定が困難になってきた HDD(というかHDDのインタフェース)の規格上CHSの範囲はしっかり決まっていたが、そのHDDを利用する機器が利用しているレイヤーによって、その範囲が変わってしまい、さらに混乱を引き起こすこととなった。

ある資料では、最大値は規格的にCHS = 65,536/16/256だとある。

```
         16 ×   256 □       4,096
     65,536 × 4,096 □ 268,435,456

268,435,456 × 512 ÷ 1,024 ÷ 1,024 ≒ 131,072MB
268,435,456 × 512 ÷ 1,000 ÷ 1,000 ≒ 137,438.95MB
```

この137GBという容量は記憶にある人がいるかもしれないCHSの最大値から必然的にこの容量が最大値となる。…この容量を超えるHDDが作れなくなってしまった。作れてもアクセスできないセクタが発生するし。
※なお、トータルセクタ数から容量を求めるとき、カタログの数値から減っているように見えるのは、単位変換時のせこいトリックを使われていたせい。 カタログでは単位換算に1,024ではなく、1,000をつかって計算している。

しかもHDDアクセスまでに通過するいくつかのレイヤー(ソフトウエアかもしれないし、ハードウエアかもしれない)が認識するCHS各値の最大値制限が異なるため、記録可能なセクタ数が制限されてしまう。あるレイヤーでは□が4,096までしか扱えなかったり、あるレイヤーでは□が255で□が63だったり。 結果、ハードウエアとソフトウエアの間にCHS範囲取り扱い上のギャップを生じさせてしまい、規格上利用できるはずの容量を充分に扱えない原因となってしまった。

LBA管理

一方LBAによるアクセスの規格もあった。

28bitの数値でセクタにブロック番号を振り、アクセスはそのブロック番号を指定する。0～268,435,455までを指定できるので、

```
268,435,456 × 512 ÷ 1,024 ÷ 1,024 □ 131,072MB
268,435,456 × 512 ÷ 1,000 ÷ 1,000 ≒ 137,438.95MB
```

とCHSの規格の最大値と一緒。

後に48bitの規格が出てきて、0〜281,474,976,710,655までを指定できる。範囲最大まで扱えるとすると、

```
281,474,976,710,656 × 512 ÷ 1,024 ÷ 1,024 ≒ 137,438,953,472MB ≒ 137.439PB
281,474,976,710,656 × 512 ÷ 1,000 ÷ 1,000 ≒ 144,115,188,075MB ≒ 144.115PB
```

137PB...137ペタバイト。
HDD内部ではそのHDD専用CHSへの変換をやっているだろうが、利用する側はCHSを意識したりしないでもいい。というか意識してやってらんない。この文書作成時現在はこちらが主流。

CHSの縛りがないので事実上の無制限状態となる。

利用・管理する側の問題(パーティション)

ただし、上記まではあくまでHDDの規格上の話である。

今度はHDDの問題ではなくHDDを管理する側の問題が発生した。 実際にHDDにデータを書き込んで管理を行うOSが48bitの規格を活用できず容量全てを使うことができない事態になる。

HDDを利用する際、パーティションという形でブロックを管理するためのエリアを設ける。 旧来から扱われているMBR形式でパーティションを作成する場合、ブロック番号を管理する値の範囲が0〜4,294,967,295(32bitの値)となるので、

```
4,294,967,296 × 512 ÷ 1,024 ÷ 1,024 ≒ 2,097,152MB
4,294,967,296 × 512 ÷ 1,000 ÷ 1,000 ≒ 2,199,023.26MB
```

と、管理できる容量2TBが限界になるHDDが48bitのLBAを扱えたとしても管理上は32bitの範囲のためHDDであってもMBR形式のパーティションを使う限り2TBまでしか利用できずTBは死蔵される。もったいないお化け出まくりである。

そんな容量のでかいHDDを扱うためGPT形式のパーティションが出てきた※2TBを越えるHDDが出回りだしたので普及しだした、が正しいかもしれない。

もともとMBR形式を置き換えるためのものなので、当然ブロック番号を管理する値の範囲も大きく0〜18,446,744,073,709,551,615(64bitの値)となり、

```
18,446,744,073,709,551,616 × 512 ÷ 1,024 ÷ 1,024 ≒
9,007,189,718,007,364.78MB ≒ 9,007.19EB
18,446,744,073,709,551,616 × 512 ÷ 1,000 ÷ 1,000 ≒
9,444,732,965,749,290.43MB ≒ 9,444.73EB
```

電卓が桁溢れしちまうんで手計算9,007.19EB...9,007.19エクサバイト。…計算合ってるかな….

FreeBSDのdc,bcコマンドで検算。

```
# dc
18446744073709551616 1024 / 1024 / 512 * p
9223372036854775808
18446744073709551616 1000 / 1000 / 512 * p
9444732965739290112
^C
```

```
#
# bc
18446744073709551616 / 1024 / 1024 * 512
9223372036854775808
18446744073709551616 / 1000 / 1000 * 512
9444732965739290112
^C
#
```

あー、なんか微妙に…ま、大体合ってるか。

とりあえず当面は足りるであろう容量を取り扱える。

補足: GPTとUEFI/ZFSの統合

GPTはUEFIブートに必須で、FreeBSD 14.xではEFIシステムパーティション（タイプ: efi、通常128MiB）を最初に作成するのが一般的です。

```
gpart add -t efi -s 128m da0
gpart add -t freebsd-zfs -s 1907601m da0
newfs_msdos /dev/da0p1
zpool create mypool /dev/da0p2
```

EFIパーティションは/boot/efiにマウントされ、ブートローダを格納します。
ZFSを使用する場合はGPTパーティション上にZFSプールを作成し、RAID-Zやスナップショット機能を活用できます。詳細は`man zpool`や`man gpart`を参照してね。

補足: GPTの最大容量と容量表記の注意

容量計算は間違ってるぞとGrokさんに指摘されたのでイカをご確認ください。
GPTの最大容量は2^64セクタ（512バイト/セクタ）で、計算すると約9.44ZB（ゼタバイト、10^21バイト）または8ZiB（ジビバイト、2^63バイト）です。元の「おまけ」章の「9,007.19EB」は単位の混在による誤り（正: 9.44ZB）
HDD/SSDメーカーは10^3（1,000）ベースで容量を表記（例: 1TB=1,000,000,000,000バイト）するが、OSは2^10（1,024）ベースを使うため、表示容量が異なる場合があります。FreeBSDでは以下で確認可能:

```
geom disk list
Geom name: da0
Mediasize: 2000398934016 (1.82TB)
Sectorsize: 512
```

おまけ２

さて。

先の2TBHDD、dmesgで以下のように出力されていた。

```
da0: <I-O DATA HDCA-U 133B> Fixed Direct Access SCSI-2 device
da0: 40.000MB/s transfers
da0: 1907729MB (3907029168 512 byte sectors: 255H 63S/T 243201C)
```

トータル3,907,029,168セクタ、セクタサイズが512バイト、255ヘッド、63セクタ/トラック、243,201シリンダ、とある。※LBAで扱える総セクタ数が3,907,029,168

```
  63×  255=    16,065 … 1シリンダあたりのセクタ数
16,065×243,201=3,907,024,065 … セクタの総数
```

実際にCHSで計算した値と、トータルのセクタ数から計算した値を比べる。

```
3,907,029,168 × 512 ÷ 1,000 ÷ 1,000 ≒ 2,000,398.93MB … この外付けHDDのカタログ値※dmesgの表示にあるセクタ数で計算
3,907,029,168 × 512 ÷ 1,024 ÷ 1,024 ≒ 1,907,729.08MB … dmesgの表示にあるセクタ数で計算
3,907,024,065 × 512 ÷ 1,024 ÷ 1,024 ≒ 1,907,726.59MB … dmesgのCHSから導いたセクタ数で計算
```

CHSで計算すると5,103セクタ、約2.49MBほど不足が出ている。これはCHS各値がつじつまを合わせるための仮の数値であり、実際のハードウエアの物理構成を示しているわけではないから。 なのでいまどきのHDDが使うLBAによるセクタ総数とは必ずしも一致しない。…さすがに255枚の円盤は入っていないと思う。

なおHDD製品によってはLBAの数値しか正しくないものがあるので注意が要る。

たとえば HITACHI Deskstar HDP725050GLA360
500GBの製品で、本体に貼られたシールにはLBAが976,773,168とある。しかし CHSが 16,383/16/63 となっていて計算してもLBAの個数に近づきすりゃしない。
CHSで計算するとこの製品は8.4GBのHDDということになる。さらに書くと、このDeskstarシリーズは別の容量の製品でもCHS表記が 16,383/16/63 になっている。

おまけ3: 2025年時点のストレージとFreeBSDのアップデート

Grokさんの提供。

SSDとNVMeの時代

2013年当時はHDD中心だったが、2025年ではSSD（特にNVMe接続）が主流。CHSはSSDでは意味を持たず、LBAが全て。NVMeは名前空間で論理分割をサポートし、セクタサイズは4KiBや16KiBも選択可能。FreeBSD 14.xでは`gpart`でNVMeデバイス（例: /dev/nvme0ns1）を直接操作可能。例:

```
gpart show nvme0
=>       34  3907029101  nvme0  GPT  (1.8T)
34          6         - free -  (3.0k)
40  3907028992     1  freebsd-zfs  (1.8T)
```

GPTとUEFI/ZFSの統合

GPTはUEFIブートに必須で、EFIパーティション（タイプ: efi、128MiB）を最初に作成するのが一般的:

```
gpart add -t efi -s 128m da0
gpart add -t freebsd-zfs -a 4k -l mypool -s 1907601m da0
newfs_msdos /dev/da0p1
zpool create -o ashift=12 mypool /dev/da0p2
```

ZFSプールを作成し、スナップショットやRAID-Zを活用可能。詳細は`man zfs`参照。

容量計算の訂正

GPTの最大容量は2^64セクタ（512バイト）で、約9.44ZB（10^21バイト）または8ZiB（2^63バイト）。HDD/SSDのカタログ表記（1TB=10^12バイト）とOS表記（1TiB=2^40バイト）の違いに注意。確認方法:

```
geom disk list
Geom name: da0
Mediasize: 2000398934016 (1.82TB)
Sectorsize: 512
```

AFTと4KiBセクタ

AFT（512バイト論理/4KiB物理）はHDDで一般的だったが、SSDでは4KiBネイティブが主流。ZFSでは`ashift=12`を指定:

```
zpool create -o ashift=12 mypool /dev/da0p1
```

UFSでは`newfs -S 4096`で4KiBセクタに最適化。

gpartの最新機能

FreeBSD 14.xの`gpart`は自動アライメント（`-a 4k`）、ラベル（`-l`）、バックアップ/リストアをサポート:

```
gpart add -t freebsd-ufs -a 4k -l mydata -s 1907601m da0
gpart backup da0 > da0.gpt
gpart restore da0 < da0.gpt
```

技術資料, FreeBSD, GPT, gpart