What are 4k discs?

Background

We are used to a standard sector size of 512 bytes. Originally discs were simply a sheet of magnetic material that was spun at 3600 or 7200 rpm. The start of each sector on a track was marked by a special magnetic ‘marker’ and 256 or 512 (sometimes 1024) bytes were read or written a short time after this passed by the head. There was room for about 16 sectors on a track.

A single-sided disc had room for perhaps 35 tracks and the ‘address’ of some data was therefore composed of track number (0 to 34) and sector number (either 0 to 15 or 1 to 16). Double sided discs had also the head (0 or 1). Hard discs had a platter of discs with multiple heads and a disc address was thus now composed of a ‘CHS’ address, cylinder, head and sector.

FileCore (from RISC OS 3.80) used a 19-bit value (idlen=19) to identify a disc object, so there can be no more than 2^{19} objects, each requiring 20 bits in the map, to allow for the extra terminating bit. The granularity of a partition is the minimum separation between each disc object, given by the formula (idlen + 1)*LFAU. There can be no more than 2^{idlen} objects in the partition. This sets the minimum value of LFAU which is one parameter chosen when formatting a partition.

Partition size	Min. LFAU	Granularity (512)	Granularity (4096)	idlen
16GB = 2^{34}	2kB = 2^{11}	40k = 80s	-	19
128GB = 2^{37}	16kB = 2^{14} 4kB = 2^{12}	320k = 640s 88k = 176s	= 80s = 22s	19 21
256GB = 2^{38}	32kB = 2^{15} 8kB = 2^{13}	640k = 1280s 176k = 352s	= 160s = 44s	19 21
512GB = 2^{39}	64kB = 2^{16} 16kB = 2^{14}	-	1280k = 320s 352k = 88s	19 21
1TB = 2^{40}	128kB = 2^{17} 16kB = 2^{14}	-	2560k = 640s 704k = 176s	19 21
2TB = 2^{41}	256kB = 2^{18} 64kB = 2^{16}	-	5120k = 1280s 1408k = 352s	19 21

16GB: LFAU >= $2^{34} \div 2^{idlen} \div$ (idlen+1) i.e. 1.6k, so LFAU can be 2k/4k …

The granularity of a 256GB partition is 640k (idlen=19) or 176k (idlen=21).

The Large File Allocation Unit (LFAU) is the internal space allocation unit for large files. This can be increased above the default to give improved performance but at the expense of consuming more space per file (wasting on average about half the LFAU for each large file).

Detail you can ignore

Device drivers typically use a 29-bit addressing method to identify a particular sector within a partition (plus 3 bits to identify the disc drive). A partition using 4k sectors can be up to 2TB (=2^{29} x 4k).

RISC OS 5.22 extended the maximum sector size to 4k (maximum partition size now 2TB). RISC OS 5.24 extended idlen to 21 bits so the granularity of a 2TB partition became 1408kB and on a 256GB 4k disc became 176k.

An alternative to the ‘CHS’ address is the ‘LBA’ address - this is just a 29-bit value giving the sector number. Conversion between ‘CHS’ and ‘LBA’ (Logical Block Addressing) addresses is simple arithmetic:

LBA = ((C * N) + H) * SPT) + S - 1

where N is the number of heads (max. 255) and SPT is the number of sectors per track (max 63). Note that C is a 10-bit quantity (0 to 1023), H is 8-bit (0 to 255) and S is 6-bit (1 to 63).

Where a hard disc is partitioned the 512 bytes at CHS (0,0,1) = LBA 0 identify where each partition starts and how big it is. FileCore ignores this partition table and stores its 512 byte boot block &C00 bytes from the start which is CHS(0,0,7) = LBA 6 (with 512 byte sectors).

A disc object can store more than one file or directory.

Solid state discs

NVMe and SATA discs are pretty much all supplied with geometry that uses 512 byte sectors. Some actually arrange data in 4096 byte units and emulate a disc with 512 byte sectors. Some NVMe drives using 512B sectors can be switched to use 4096B sectors and reformatted.

SATA drives for RISC OS using 4k sectors are nothing new, you've been able to use such drives since 2017 (on Titanium at least, due to updated ADFS and SATADriver modules). Elesar has carried stock of preformatted 2TB SATA drives since April 2018 as a spare part.

There are two advantages in switching to 4k sectors for RISC OS: it increases the maximum partition size from 256GB to 2TB and, in the case of NVMe discs, offers very much improved performance for random reads and writes (two or three times the speed than on the same NVMe drive with 512B sectors - roughly equal speed to SATA drives).

The incentive to use 4k SATA drives on RISC OS has however been muted as a 256GB partition is enough for most users.

So now that we have a strong incentive to use 4k NVMe discs (much faster than with 512B sectors), how easy is it to switch them to 4k?

For this we need Linux, which provides a set of nvme utilities:

Switching to 4k

There are two ways to switch an NVMe drive from 512e to 4k - in RISC OS using some software called NVME4kFmt (still in beta development) or in Linux, as described below.

Switching a drive to or from 4k sector size destroys all the data on the disc. Once the switch has been done, the driver needs to be formatted again.

```
sudo apt install nvme-cli
sudo nvme id-ns -H /dev/nvme0n1
...
LBA format 0 : ... 512 bytes (in use)
LBA format 1 : ... 4096 bytes
sudo nvme format --lbaf=1 /dev/nvme0n1
```

this installs the 'nvme' command and uses it to show whether the drive supports 4k sectors and, if so, to re-format it to use 4k sectors vice 512B sectors. Changing between 512e and 4k will destroy all data on the disc.

The blank drive then needs to be formatted.

```
RISCOSmark 2.06 (29-Mar-2016) by Richard Spencer 2003
Filing system: NVMe:HardDisc4.$.SpeedTests sector size=512B LFAU=4k partition=110GB
HD Read - Block load 8MB file        165415    42%
HD Write - Block save 8MB file       270415    20%
FS Read - Byte stream file in          1050    21%
FS Write - Byte stream file out        1558    30%

Filing system: NVMe:HardDisc4.$.SpeedTests sector size=512B LFAU=32k partition=110GB
HD Read - Block load 8MB file        194661    49%
HD Write - Block save 8MB file       281098    21%
FS Read - Byte stream file in          2937    60%
FS Write - Byte stream file out        2967    58%

Filing system: NVMe:HardDisc4.$.SpeedTests sector size=4k LFAU=8k partition=240GB
HD Read - Block load 8MB file        192752    48%
HD Write - Block save 8MB file       270415    20%
FS Read - Byte stream file in          4216    86%
FS Write - Byte stream file out        4347    85%

percentages are proportion of RAMfs speed
```

Speed comparisons using the same NVMe drive with different formatting options, as shown.

Cunning plan

I had a cunning plan to use a multi-format NVMe drive to include a 'LinuxBoot' FAT partition for the Linux boot drive '/boot', a filecore partition for RISC OS and an 'ext4' partition for the Linux rootfs '/'. I thought the FAT partition could be used to share files between Linux and RISC OS.

A filecore partition has to start at the start of the disc (actually from address &C00 onwards where information about the disc can be read, including its size and thus where the map information is stored, always halfway through the partition onwards). This conveniently leaves space untouched at the start of the disc (&000 to &BFF) where other operating systems expect to read information about how the disc is partitioned.

FileCore stores its disc map at about half way through the partition space reserved for it and allocates space for files from there onwards.

What the partition table says about a partition of type 'AD' (filecore) is ignored by RISC OS. So long as it covers the space on disc up to where the filecore partition actually ends, other operating systems will not trample on the filecore data.

This looks difficult

PartMgr can place a DOS image file !Boot.Loader at a suitable position within the filecore partition, near the start but at a 1MB boundary. The area on the disc that precisely corresponds to the space occupied by this file can then be marked as the extent of a FAT partition.

Either Linux or RISC OS can write to this area and the result be viewed on either platform. On a disc with 4k sectors, however, the contents of the FAT partition can either be suitable for RISC OS or for Linux but not both as DOSFS is 'hard-coded' to use 512B sectors.

```
0000000000 ...
00000001BE 00 00 00 00 0E 00 00 00
00000001C6 00 01 00 00 00 2C 01 00
00000001CE 00 00 00 00 AD 00 00 00
00000001D6 00 2D 01 00 D4 82 70 0B
00000001DE 00 ...
000002C000 xx xx xx 4D 53 44 4F 53
000002C008 ...
0000100000 start of 300 MB FAT partition
0000100008 ...
0012D00000 end of 300 MB FAT partition
0012D00008 ...
0012E00000 end of 301MB 'Protect' object
0012E00000 'start' of 750GB partition
0012E00008 ...
B71B000000 end of 750000MB partition
B71B000000 start of next partition
Note:
012C00000 = 300MB ; 012D00000 = 301MB
00002C000 + 012D00000 = 012D2C000
2C 4k sectors = 00002C000
301MB FileCore object encompasses the
300MB FAT partition
```

The partition table entries at 01BE and 01CE protect the 750000MB FileCore partition and define the start and end of a 300MB FAT partition. Note: 750,000MB = &B71B000000

PartMgr will also create a suitable 'MBR' partition table at disc address &000-&1FF to protect the two partitions, see above. Other operating systems place restrictions on where partitions may start and end (e.g. on a 1MB boundary on Linux with 4k sectors). This depends on the drive design.

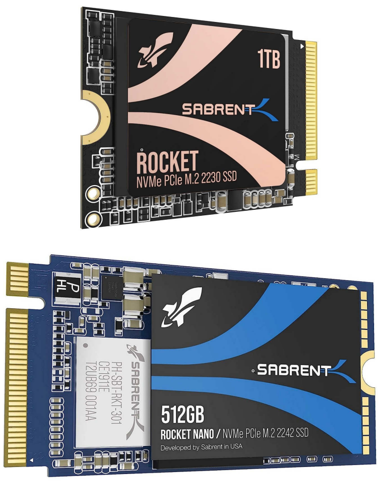
This is the method used for SD cards on the Raspberry Pi to present a FAT partition when starting up and a FileCore structure to RISC OS.

So, if RISC OS filecore is not the only show in town, it is a good idea to have a partition table defining the space used and how much of the disc space is still unallocated.

Other operating systems use the partition table at CHS(0,0,1) to identify which parts of the disc have been already used by partitions and whether or not it can recognise them and read or write to or from them.

```
DiscKnight 1.55 (12-Aug-2018) [32bit] Serial no. ########

Arguments: -v -l NVMe 4

Boot block - Boot Record
  Number of Tracks        : 59757
  Number of Heads         : 17
  Sectors per track       : 63
  Sector size             : 4096
  Density                 : hard disc
  ID Length               : 21
  Bytes per map bit (LFAU) : 8192
  Minimum object size     : 176K
  ...

Checking directory structure

  Directory $ SIN=&0B1A5001 length=&00000800=2048
    Chunk at &0000001E889A4000 +&0002C000 sharing sectors 0 to 0

  Directory $.!Boot SIN=&0B1A5101 length=&00000800=2048
    Chunk at &0000001E889D0000 +&0002C000 sharing sectors 0 to 0

  File $.!Boot.Loader SIN=&00000300 length=&12C00000=300M
    Chunk at &0000000000100000 +&0FB00000
    Chunk at &000000000FC00000 +&03100000

----------------------------------------
                Disc is good
----------------------------------------
```

DiscKnight shows that the file Boot:Loader is at disc address &100000


1TB
SABRENT
ROCKET
NVMe PCIe M.2 2230 SSD
SABRENT
512GB
ROCKET NANO / NVMe PCIe M.2 2242 SSD
Developed by Sabrent in USA

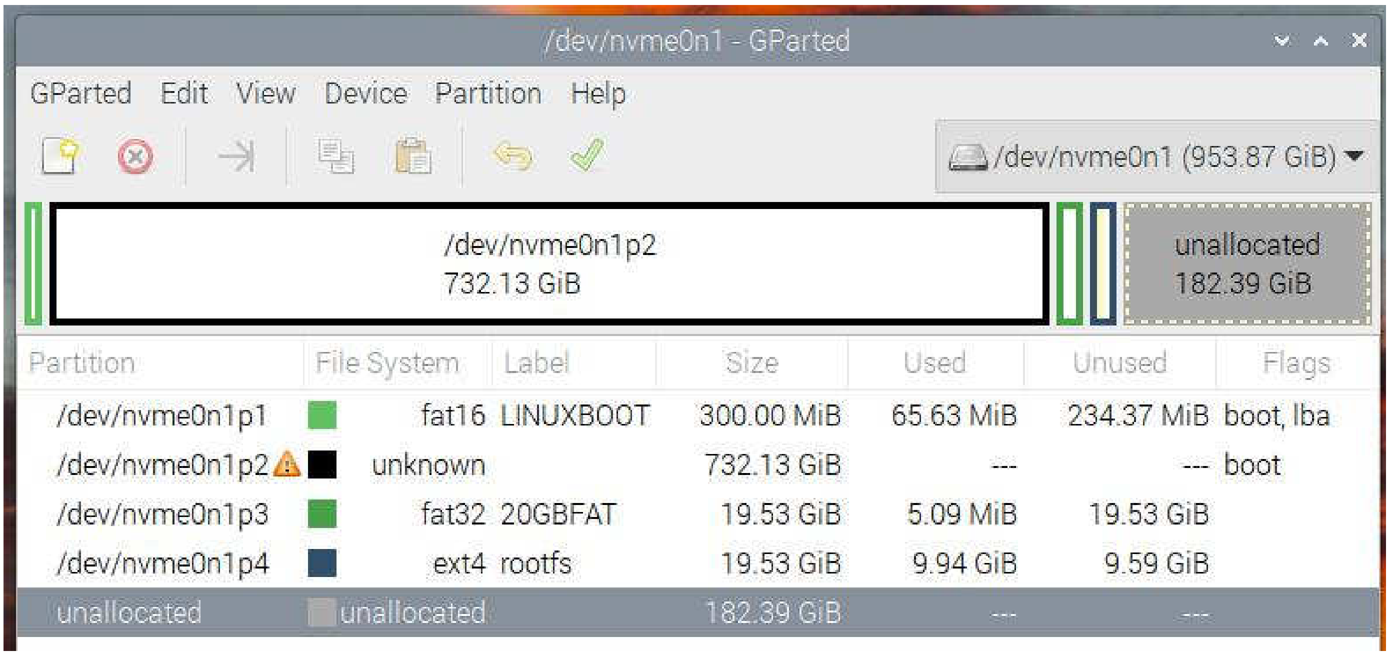
I have tried two drives so far - a 1TB 2230 Sabrent model SB-2130-1TB (£94.99) and a 512GB 2242 Sabrent model SB-1342-512 (£59.99).

Both are single-sided (so fit Waveshare IO board) and both can do 4k. Only the 2242 drive will fit DeskPi.

Which NVMe drives will do 4k?

A good question. They all do 512B sector emulation but only some do 4k. The specification for an NVMe drive does not usually mention whether it is capable of doing 4096B sectors so it is a matter of guesswork when purchasing.

A 128GB or 256GB NVMe drive is unlikely to benefit very much from using 4096B sectors.



Partition		File System	Label	Size	Used	Unused	Flags
/dev/nvme0n1p1	🟩	fat16	LINUXBOOT	300.00 MiB	65.63 MiB	234.37 MiB	boot, lba
/dev/nvme0n1p2 ⚠	■	unknown		732.13 GiB	---	---	boot
/dev/nvme0n1p3	🟩	fat32	20GBFAT	19.53 GiB	5.09 MiB	19.53 GiB	
/dev/nvme0n1p4	■	ext4	rootfs	19.53 GiB	9.94 GiB	9.59 GiB	
unallocated	▨	unallocated		182.39 GiB	---	---	

What the 1TB drive looks like in Linux:

Is there a purpose to this?

Yes. It means that we can share a drive between Linux and RISC OS without them interfering with each other. There is now no need for RISC OS to access the contents of the FAT partition.

Files can be shared between RISC OS and Linux by using a FAT-formatted USB drive.

Summary

PartMgr will initialise a Pi-style two-partition NVMe drive creating a 300MB FAT partition and co-located FileCore 'Boot:Loader' object within a 250000MB or 750000MB filecore partition.

We can use Linux to add a 20GB 'ext4' partiton in the unused 250GB on the drive and a standard Linux distro (some 9GB) can be copied to it using the Linux 'dd' command.

Rebooting Linux again and 'checking' the Linux partition will extend it to the full size of the partition that you created without losing any data.

The last step is to edit the file /etc/fstab in this new partition to include the UUID of the NVMe partitions rather than the UUID of the drive it was copied from.

Finally edit the file 'CMDLINE.TXT' to specify the NVMe drive's UUID and Linux will now boot from the NVMe drive rather than from a USB drive.

That's it for now, folks!

Chris Hall *chris@svrsig.org*