Analyzing large data with Mappable Vector Library

Vladimir Dergachev[1,2]

[1] *Max Planck Institute for Gravitational Physics (Albert Einstein Institute), Callinstrasse 38, 30167 Hannover, Germany* vladimir.dergachev@aei.mpg.de
[2] *Leibniz Universität Hannover, D-30167 Hannover, Germany*

Abstract.
A common problem when working with large data sets is that one is limited at looking at snippets of the whole data, because retrieving the entire data set from a database is inefficient. The solution is to bring the data closer to the user, allowing interactive exploration at the speed of the underlying storage medium. Mappable Vector Library (MVL) accomplishes this by memory mapping the entire data set, allowing direct access from R using RMVL package (C library is also available). We introduce MVL and illustrate MVL capabilities using 1TB of Gaia data.

1. Introduction

Mappable Vector Library (MVL) is a file format optimized for memory mapping. It is designed for ease of use by both low-level C code and high-level scripting languages (such as R).

You can use MVL files to:

- Analyze as much data as fits on your solid state drives
- Exchange bulk binary-level data between C and R programs
- Share large data between multiple processes running on the same computer
- Distribute large structured data to users

There are two libraries for C (libMVL [1]) and R (RMVL [2]) providing seamless array-style access as well as database functionality of sorting the data and creating hash-based value and spatial indices. The library functions have been optimized to reduce the number of writes to solid state drives.

The development of MVL file format was prompted by the observation that modern solid state drives had transfer speeds similar to the speed of main memory of older computers. However, the direct use of solid state drives (SSD) as swap space to emulate physical memory runs has difficulties, stemming from mismatch between block size of

[1] https://github.com/volodya31415/libMVL

[2] https://cran.r-project.org/package=RMVL

SSDs and main memory, requirement to minimize writes to SSD and an initial cost of loading terabytes of data into a program during startup.

A simple solution to these issues to is to memory map input data. This allows quick "load" by a memory map call, with the program effectively starting in swapped-out state, bringing data in as needed. The ordering of input data avoids inefficiencies of a random swap-out process. As a bonus such scheme allows sharing of input data between multiple processes.

The alignment of data to a specific memory boundary is a key requirement for vector operations and to improve efficiency of CPU caches. A survey of existing data formats such as hdf5, MyISAM, sqlite, FITS (Taylor 2019; Taylor & Page 2008), parquet and others revealed none that were capable of providing aligned binary data that is directly suitable for computation. Portability is also an issue, for example, it is difficult to incorporate hdf5 library into Android applications.

A new file format was therefore required that would guarantee alignment of binary data. To make programming convenient one needs a simple way to obtain location of data after memory mapping - an equivalent of dynamic linking. In addition, it would be nice to have "global" functions that operate on large data as a whole and provide functionality traditionally available in relational databases.

2. MVL architecture

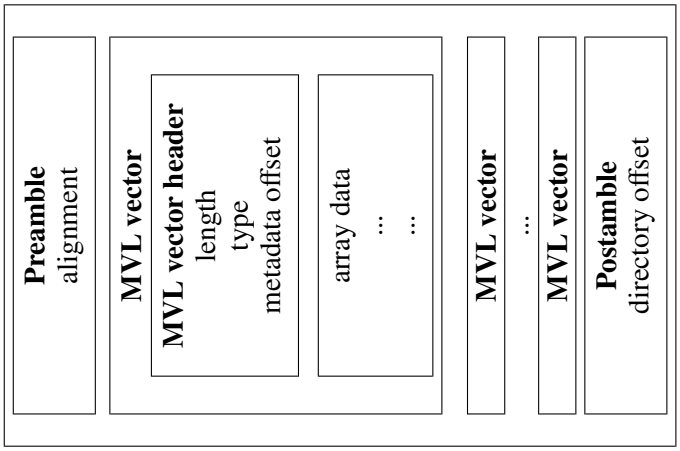

Figure 1. MVL file layout. Each MVL vector is positioned so that the offset of the data is divisible by the value of *alignment* field. The default is 64. The postamble contains a "directory offset" that points to MVL vector storing a named list of offsets to other vectors. In filesystem terminology, this is the root directory of MVL file.

Unlike application-level databases the MVL files are designed for direct access to data. The basic element of MVL file (Figure 2) is an MVL vector composed of a header, describing type and length of the data, followed by a linear array of data elements. The array is aligned to a configurable boundary, 64 bytes by default. This allows to use the data as is with vector arithmetic operations.

Each MVL vector is uniquely identified by the 64-bit offset from the beginning of the MVL file. This makes it very easy to access the data in C - just memory map the file and use the offset to obtain a pointer to data.

The MVL vectors support character, integer and floating point types, as well as, a special *offset* type used to store a list of offsets to other vectors. This allows to organize the data into a tree with arbitrary complexity and maps nicely to the internal representation used by scripting languages such as R or Tcl.

Each MVL vector header can have an optional offset to metadata, such as a vector of character names associated to elements of MVL vector array. This allows to create named lists, which can be used to represent sets, dictionaries and structures. The MVL library functions supporting named lists use 64 bit hashes for fast lookups.

The MVL file ends with a postamble containing an offset to the top-level vector of offsets called a *directory*.

MVL files are designed to be written sequentially, with the directory and postamble written out during the closing of the file.

3. Database functionality

Memory mapping of data makes it easy to access values at a known address. However, often one needs to solve an inverse problem - find the address of data by its value. In SQL databases this is used to perform searches and merge tables by column values.

A common way to make this computationally efficient is to use data indices. For example, one can create an array of indices that sorts input data in ascending order. The values can then be looked up by performing a binary search.

This approach is difficult to use for MVL files because the size of the data can be so large that a binary search performs too many random lookups. On modern computers a memory access at an unpredictable location takes a very long time to complete, and having many of them for each lookup is unacceptable.

The solution used in libMVL, at the time of publication of this paper, is to utilize hash based indices. The hash function used produces 64-bit hash values that have enough entropy to have few (or none) collisions for very large data. The hash function has been optimized for speed and is able to saturate read bandwidth of modern SSDs on a single CPU thread.

An associative hash map has fixed number of random address dereferences allowing the algorithm to scale to arbitrary large data sizes. A similar hash-based technique is used to construct spatial indices which lookup values based on proximity to a given data point. The hashes are also used to perform computations that process data in groups. The hash function values have good statistical properties and can be used to obtain randomized samples in a deterministic fashion.

4. Use case: a galaxy in a file

One of the most exciting developments in astrophysics is the emergence of gravitational wave observatories (Aasi 2015) as an observational tool. A particularly exciting area of gravitational wave research is the quest to find persistent gravitational wave sources such as rapidly spinning neutron stars with equatorial deformation (Dergachev & Papa 2022; Riles 2022).

A recently released gravitational wave atlas of the sky (Dergachev & Papa 2022) in MVL format provides frequency-resolved all-sky data describing potential strength of such sources. As the signals are very weak, the visibility of any viable source is limited to a few hundred parsecs. The size of the data set and correspondingly large trials factor makes identification of weak sources challenging.

A way to solve this problem is to use an astrophysically motivated prior. The Gaia data (Brown 2022; de Bruijne 2016) provides a comprehensive database of stars in the Milky Way and is an excellent resource for cross-correlation studies.

However, the main Gaia data is over 1 TB in size and is provided as 3386 compressed CSV files. While one can use online databases to search for a given object, discovery of unknown sources requires exploration of the Gaia data as a whole. To make this possible Gaia data was converted to MVL format.

The resulting main Gaia data in MVL format (1.3 TB) is easily memory mapped into R. With commonly available solid state drive speeds of 3.5 GB/s, it is possible to scan the entire data in under 7 minutes. Of course, in practice, one does not need all the different variables. For more sophisticated access there are indices allowing to search and scan in order of distance from Earth, or by sky position.

Gaia data in MVL format is now available to the wider community [3] for download using HTTP and Torrent.

5. Summary

Mappable Vector Library has been designed for memory mapped access to large data sets. It supports structured data and database functions. MVL files can be used on a cluster to reduce memory footprint of single threaded programs. MVL files can also be provided directly to other scientists, which can use them directly, without need for database setup.

Acknowledgments. This work has made use of data from the European Space Agency (ESA) mission *Gaia*, processed by the *Gaia* Data Processing and Analysis Consortium.

References

Aasi, J. e. a. 2015, Classical and Quantum Gravity, 32, 074001. 1411.4547

Brown, A. G. A. e. a. 2022, arXiv e-prints, arXiv:2208.00211. 2208.00211

de Bruijne, J. H. J. e. a. 2016, A&A, 595, A1. 1609.04153

Dergachev, V., & Papa, M. A. 2022, A frequency resolved atlas of the sky in continuous gravitational waves. URL https://arxiv.org/abs/2202.10598

Riles, K. 2022, Searches for continuous-wave gravitational radiation. URL https://arxiv.org/abs/2206.06447

Taylor, M. B. 2019, in Astronomical Data Analysis Software and Systems XXVII, edited by P. J. Teuben, M. W. Pound, B. A. Thomas, & E. M. Warner, vol. 523 of Astronomical Society of the Pacific Conference Series, 43. 1811.09480

Taylor, M. B., & Page, C. G. 2008, in Astronomical Data Analysis Software and Systems XVII, edited by R. W. Argyle, P. S. Bunclark, & J. R. Lewis, vol. 394 of Astronomical Society of the Pacific Conference Series, 422

[3] https://www.atlas.aei.uni-hannover.de/work/volodya/Gaia_dr3/