On the Usability of High-Level Parallel IO in Unstructured Grid Simulations

Dries Kimpe^{1,2}, Stefan Vandewalle¹, and Stefaan Poedts²

¹ Technisch-Wetenschappelijk Rekenen, K.U.Leuven,
Celestijnenlaan 200A, 3001 Leuven, België

{Dries.Kimpe, Stefan.Vandewalle}@cs.kuleuven.be

² Centrum voor Plasma-Astrofysica, K.U.Leuven,
Celestijnenlaan 200B, 3001 Leuven, België

Stefaan.Poedts@wis.kuleuven.be

Abstract. For this poster, the usability of the two most common IO libraries for parallel IO was evaluated, and compared against a pure MPI-IO implementation. Instead of solely focusing on the raw transfer bandwidth achieved, API issues such as data preparation and call overhead were also taken into consideration. The access pattern resulting from parallel IO in unstructured grid applications, which is also one of the hardest patterns to optimize, was examined.

Keywords: MPI, parallel IO, HDF5, parallel netcdf.

Parallel IO is of vital importance in large scale parallel applications. MPI offers excellent support for parallel IO since version 2, particularly because of its fundamental and complete support for user defined data types. However, as demonstrated by the popularity of netcdf[5] and HDF5[3], applications are in need of a higher level API that enables them to deal with data more naturally. Parallel netcdf[4] and the implementation of MPI-IO support in HDF5 fulfill this need for MPI applications. As the software stack for this kind of storage can be quite complex, performance is easily lost if the coupling between the layers is not done carefully.

The motivation for this work originates in the investigation of a performance problem in a parallel unstructured grid code relying on HDF5 for file storage. In this code, after partitioning, all CPUs need to read the coordinates and values of all grid points that were assigned to them by the mesh partitioner. This results in an almost random access pattern consisting of collective read operations. While eventually the total dataset is read, a non-contiguous subset is accessed during every read operation.

The authors did everything possible to assure efficient IO, for example by utilizing collective data transfers with complete HDF5 type descriptions and a continuous storage layout. Still, the code performed poorly when scaling to larger CPU counts. Examination of the HDF5 source code revealed that no parallel IO was supported for point selections in a dataset.¹ This resulted in

¹ This holds for both the latest stable release (1.6.5) and the current alpha release (1.8.0).

every CPU executing an independent read request for every accessed element of the dataset, leading to seriously degraded IO performance.

HDF5 has extensive support for partial dataset selection and another method to access the same subset was found. While this method did support parallel IO, it suffers from another kind of problem. Although the final read operation itself takes advantage of parallel IO and custom data types, the API needed to setup this selection requires repeatedly calling a function with time complexity $O(\text{number_of_currently_selected_elements})$. For a random selection of n points, this leads to $n!$ operations. Searching for alternatives, parallel netcdf was tested as well, but was also shown to have issues preventing efficient data access (for the described access pattern).

For this poster, an effort was made to describe best practices to achieve high performance with the discussed storage libraries, evaluated in the context of unstructured grid applications. Problems affecting performance, in both the API and internal implementation, are highlighted. Actual performance measurements, demonstrating achievable bandwidth, were made in combination with true parallel filesystems such as lustre[1] and PVFS2[2] and more traditional ones such as NFS. All tests were performed on an opteron based cluster situated at K.U.Leuven.

As a preliminary conclusion, application writers in need of directly available performance are better off directly using MPI-IO whenever possible. This is particularly true for the class of irregular access patterns considered in our study. Relying on a storage library that fails to utilize the flexibility and power that MPI-IO offers results in a significant loss of performance. In principle, nothing prevents high level IO libraries from achieving the same performance as raw MPI-IO. However, at this moment their implementations need to mature somewhat more before this becomes true.

References

1. Lustre: A Scalable, High-Performance File System, white paper, November 2002, <http://www.lustre.org/docs/whitepaper.pdf>.
2. Rob Latham, Neil Miller, Robert Ross and Phil Carns: A Next-Generation Parallel File System for Linux Clusters, LinuxWorld, Vol. 2, January 2004.
3. HDF5: <http://hdf.ncsa.uiuc.edu/HDF5/>.
4. Li, J., Liao, W., Choudhary, A., Ross, R., Thakur, R., Gropp, W., Latham, R., Siegel, A., Gallagher, B., and Zingale, M. 2003. Parallel netCDF: A High-Performance Scientific I/O Interface. In Proceedings of the 2003 ACM/IEEE Conference on Supercomputing (November 15 - 21, 2003). Conference on High Performance Networking and Computing. IEEE Computer Society, Washington, DC, 39. Jianwei Li, Wei-keng.
5. Rew, R., Davis, G., and Emmerson, S., "NetCDF User's Guide, An Interface for Data Access Version 2.3," available at <ftp.unidata.ucar.edu>, April 1993.