Volume Title
ASP Conference Series, Vol. **Volume Number**
Author

Scaling GDL for multi-cores to process Planck HFI beams Monte Carlo on HPC

A. Coulais[1,‡,⋆], M. Schellens[§], G. Duvert[2], J. Park[3], S. Arabas[4], S. Erard[5], G. Roudier[6,‡], E. Hivon[7,‡], S. Mottet[7,‡], B. Laurent[1], M. Pinter[1], N. Kasradze[1], and M. Ayad[1]

[1]*LERMA CNRS and Observatoire de Paris, Paris, France*

[2]*UJF-Grenoble 1/CNRS-INSU, Institut de Planétologie et d'Astrophysique de Grenoble (IPAG) UMR 5274, Grenoble, F-38041, France*

[3]*Department of Physics and Astronomy, Seoul National University, Seoul 151-747, Korea*

[4]*Faculty of Physics, University of Warsaw, Poland*

[5]*LESIA CNRS and Observatoire de Paris, Meudon, France*

[6]*JPL and California Institute of Technology, Pasadena, CA, USA*

[7]*Institut d'Astrophysique de Paris (IAP CNRS), Paris, France*

[‡]*co-authors for beams part* [§]*GDL leader* [⋆]*contact email: alain.coulais@obspm.fr*

Abstract.
After reviewing the majors progress done in GDL –now in 0.9.4– on performance and plotting capabilities since ADASS XXI paper (Coulais et al. 2012), we detail how a large code for Planck HFI beams Monte Carlo was successfully transposed from IDL to GDL on HPC.

1. GNU Data Language

GDL is a *free* clone of IDL, an interpreted language widely used in astronomy. GDL is under GNU GPL v2/v3 licence. Written in C++, GDL is easily extendable and "hackable". Most of IDL 7 syntax is available and a large subset of functionalities is implemented using state of the art free libraries (GNU Scientific Library (GSL), FFTw (Frigo & Johnson 2005), Eigen3, PLplot ...) for competitive results.

The current version is 0.9.4. Its main changes concern benchmarking, plotting outputs, extension of the syntax capabilities with the implementation of the compound data type known as "list", and better support of Complex data.

1.1. OpenMP and Eigen3

While preparing the move from IDL to GDL in the pipeline discussed in Sect. 2, it was discovered that the matrix multiplication in GDL was too slow compared to IDL for large matrices with sizes in the 1000/10000 range. After carefully studying the

situation and the literature, we decided to test Eigen3 capabilities (Guennebaud et al. 2010). Three points were clearly good for GDL: licence, simplicity of use, serious user basis (KDE, Google, ...). The result exceeds our expectations not only for matrix multiplication but also for some other operations (arithmetic operators, exp., log. ...), exploiting not only multi-cores very well, but also optimization tricks inside the CPU (SSE2/3/4 ...). The only trick we had to solve is to estimate good value for OpenMP thread numbers when multi-cores are partially loaded. Despite still having a low performance SMOOTH code, the whole TIME_TEST4 is faster in GDL 0.9.4 than in IDL 8 on all the testbeds we have, from single to 48 cores. Another interesting benchmark is the matrix one (we plan to extend our benchmarks, suggestions welcome!).

1.2. Plotting outputs

Large sections of the graphical outputs functionalities have been carefully rewritten to provide better quality outputs and are now in GDL 0.9.4. They request at least PLplot 5.9.7, which is compilable even on rather old Linux distributions (e.g. CentOS 5x or Suse 10). It is better to use PLplot 5.9.9 for the best results. No IDL 8 new graphical capabilities have been added up to now (contributions welcome!).

1.3. Important features

Due to licensing issues, we cannot package the CMSV library (`http://www.physics.wisc.edu/~craigm/idl/cmsave.html`) managing the SAVE & RESTORE facilities within GDL, but it is a matter of minutes for an user to retrieve this package and get the SAVE & RESTORE commands working.

The widely used MPFIT (Markwardt 2009) works fine in GDL. Due to the way numbers are managed internally, results between GDL and IDL can differ around the machine's precision. Up to now, the problematic cases between IDL and GDL encountered have always been in regions where MPFIT should not be used (small numbers or small relative differences) as documented in MPFIT, or related to inputs tainted by random numbers (command RANDOMN) for obvious reasons.

The computational part of HEALPix (Górski et al. 2005) works well. Some graphical capabilities are still limited (eg, transparent pixels in PNG outputs ...).

GDL is successfully used for large codes (PSM, iCosmos, Scanamorphos ...), dedicated tools (on-the-fly data conversion for VO server), pipelines on HPC (Planck HFI beams Monte Carlo, see Sect. 2), but having an existing IDL code running smoothly under GDL still needs some efforts, as described below.

1.4. Packaging vs compiling

Thanks to various contributors, new GDL versions are now quickly packaged for several distributions, including : Fedora, Debian, Ubuntu, Gentoo, FreeBSD, ArchLinux, OSX. Nevertheless, we have no schedule for version delivery, and major upgrades like Eigen3 usage can wait for months before being included into a packaged version. If you want to use a recent GDL version, the CVS version is for you. The CVS version is rather easy to compile on most "recent" Linux and OSX systems, as long as you have a recent CMake version ($\geq$ 2.8.11), a recent plplot version ($\geq$ 5.9.7), and a C++ compiler (`gcc, clang, icc, pathcc` tested). GDL and dependencies can be compiled and used in user space (no need to be *root*). If you use locally compiled dependencies you just must use the absolute path with `compile` or `cmake`. We try to avoid serious

regressions in the CVS, running very frequently a large (but perfectible, help wanted!) regression test suite.

1.5. MS Windows port

It was a long requested feature: thanks to J. Park, GDL should now compile correctly under MS–Win. We received little feedback on that, and since no core developer can produce MS–Win executable snapshots of the CVS, feedback and contributions are very welcome. We understand that the SPAWN procedure, widely used in the tests suite, and also few files related routines are currently not working on MS–Win.

2. Planck HFI Beam Monte Carlo

HFI on-board of the Planck satellite (Planck Collaboration et al. 2011) is an experiment dedicated to observations of the whole sky at sub-millimeters wavelengths, from 100 to 857 GHz. 52 bolometers are working simultaneously during satellite spinning.

In order to quantify the systematic errors on C_l (angular power spectrum) coming from uncertainties in beams reconstruction for each bolometer, a Monte Carlo pipeline was developed, written in IDL language, using rather intensive computations and using various external libraries in IDL, including MPFIT. The computations were done on Planck HPC Magique III (1128 cores `http://prof.planck.fr/article689.html`).

Since the bolometers are processed independently, natural parallelisation can be used. The amount of computation is about one hundred runs per bolometer, 52 bolometers, one 8–cores node per bolometer. Each elementary run takes about 10 to 20 minutes once the code is optimized for a total amount of 8–cores node of one day per bolometer.

Only a limited number of IDL licences are available on M III. Moreover when input data are ready for Monte Carlo, the time to run the pipeline is very scarce since Planck data have serious rendezvous. And any improvement in the pipeline code forces to run the whole pipeline again.

2.1. Preparing the code for GDL

Various constrains were mandatory to run the pipeline in GDL: (1) good arithmetic, (2) have a working MPFIT, (3) reading input data in various formats (text files, IDL SAVE files, via DLL third party PioLib) and (4) to be quite as fast as IDL.

Concerning (1), except one issue in the accuracy of the modulo operator on large values (fixed), no other arithmetic issue was found in GDL.

Concerning (2), we did have troubles related to MPFIT, at two levels. One is we were in cases (small relatives numbers) where MPFIT should not be used. It was easier to understand it because internally IDL and GDL do not deal with exactly the same representation in numbers (GDL is internally in Double). Another is IDL that before 8.2 does not compute with Double precision in INTERPOLATE (e.g. `b=DINDGEN(10)#REPLICATE(1.,10) & for ii=1, 12 do print, format='(e16.9)', INTERPOLATE(b, 2+10.D^(-ii), 4)-2.`).

Concerning (3), we still have issues with some ASCII files coming from OSX; it is still unfixed but a preliminary conversion was enough (`mac2unix`). The code is also using custom DLLs (AKA PioLib) to read Planck data and housekeeping. A careful comparison between values of various types made between IDL and GDL calls of PioLib did not show any discrepancy on this mandatory part of the code.

Concerning (4), a serious problem was to sort the time spend in various parts of the code. It was quickly found that the most consuming part was related to large (30000×600) rectangular matrix multiplications. On the IDL side, changing the way MATRIX_MULTIPLY was called was enough to gain a factor 10. On the GDL side, the solution was to use Eigen3. At the end, GDL was faster than IDL on this part.

No other issues were discovered, and the numerical outputs, using the same input, are equivalent at the *machine precision* level The final difference in time show that GDL (start to end) is about 25% slower than IDL. Two bottlenecks in GDL have been identified: large I/O using SAVE & RESTORE, slower in CMSVlib than in IDL, and large text I/O written in loop. Being able to run all the pipelines in parallel was a big gain in comparison !

We succeed to use GDL for this code, and the conclusions are: (1) It makes sense to use GDL on HPC, without licensing issues; (2) After correcting the slowest parts in GDL, both interpreters are comparable in speed; (3) A careful preparation with tests is mandatory to be sure code outputs make sense: at the beginning, it was not realized that, in one optimization, the code always stays in local minimum, without going to the expected global minimum. It is the difference between IDL and GDL outputs that helped to solve it; (4) Having two interpreters for the same syntax is better to catch numerical issues (e.g. in the MPFIT case).

3. Conclusion

A large fraction of IDL functionalities are today available in GDL, widely tested on various systems, with excellent computing time. Transferring a working code written IDL syntax from IDL interpreter to the GDL one needs some cautious but is is no more difficult than porting a C++ code for various C++ compilers !

Acknowledgments. The GDL team would like to warmly thanks all these various users and contributors who reports bugs, provide patches, are usually very patient and sometimes even spend time to write codes for GDL. Special thanks to O. Poplawski from Fedora, T. Enomoto for MacPort and J. W.. Big thanks also to all the packagers for GDL.

References

Coulais, A., Schellens, M., Arabas, S., Lenoir, M., Noreskal, L., & Erard, S. 2012, in Astronomical Data Analysis Software and Systems XXI, edited by P. Ballester, D. Egret, & N. P. F. Lorente, vol. 461 of Astronomical Society of the Pacific Conference Series, 615

Frigo, M., & Johnson, S. G. 2005, Proceedings of the IEEE, 93, 216. Special issue on "Program Generation, Optimization, and Platform Adaptation"

Górski, K. M., Hivon, E., Banday, A. J., Wandelt, B. D., Hansen, F. K., Reinecke, M., & Bartelmann, M. 2005, ApJ, 622, 759. arXiv:astro-ph/0409513

Guennebaud, G., Jacob, B., et al. 2010, Eigen v3, http://eigen.tuxfamily.org

Markwardt, C. B. 2009, in Astronomical Data Analysis Software and Systems XVIII, edited by D. A. Bohlender, D. Durand, & P. Dowler, vol. 411 of Astronomical Society of the Pacific Conference Series, 251. 0902.2850

Planck Collaboration, Ade, P. A. R., Aghanim, N., Arnaud, M., Ashdown, M., Aumont, J., Baccigalupi, C., Baker, M., Balbi, A., Banday, A. J., & et al. 2011, A&A, 536, A1. 1101.2022