Parallelising Image Registration and the HPC Porting Journey

Tooley, P.[1,2], Benemerito, I.[1], Melis, A.[1], Narracott, A.[1], Marzo, A.[1], Viceconti, M.[1], 3

[1] Department of Mechanical Engineering and Insigneo Institute for in silico Medicine, University of Sheffield, Sheffield, S1 3JD, United Kingdom

[2] The Numerical Algorithms Group, Manchester, M1 3LD, United Kingdom

[3] Department of Industrial Engineering, Alma Mater Studiorum, University of Bologna, Bologna, 40126, Italy

Image registration is widely used in many areas of computational biomedicine, both as a research tool and a component in workflows providing clinical decision support. There exists a wide range of both open-source and commercial tools for performing image registration based on a variety of different methods.[1] However, these tools are designed to be run on a single machine, with the associated limitations of computational performance and available memory of the system, placing a limit on the maximum size of images which can be handled. A key application for image registration at the University of Sheffield is strain measurement of bone samples using digital volume correlation (DVC).[2] This makes use of tomographic imaging from synchrotron light sources, which can be many tens to hundreds of gigabytes in size — too large to be handled at full resolution by these existing codes. The solution is therefore to create a parallelised image registration code, capable of leveraging HPC infrastructure to register images of such sizes using the memory and computational capacity of multiple HPC nodes.

In this presentation we will give an overview of the process of developing a new scientific code, in particular with a view to deployment on HPC infrastructure, using our newly developed image registration code "pFIRE" as an example. pFIRE (The Parallel Framework for Image Registration) is based on an existing registration method developed at the University of Sheffield in the early 2000s.[3] This uses an optical flow based registration method, and was chosen as the underlying algorithm for pFIRE as it is well proven in applications, and allows use of pFIRE as a drop-in replacement for a previous serial code in existing workflows.

pFIRE is implemented in modern C++ (C++14 standard), using the PETSc[4] parallel toolkit to provide the necessary parallel linear algebra routines and MPI for computation between parallel processes. The primary challenge in parallelising algorithms using MPI is in choosing efficient data layout and computational strategies to minimize the time the code spends waiting for communication to occur. We will discuss potential strategies to achieve this using the pFIRE implementation as an example, showing how initial choices in data layout can have major consequences for subsequent code performance. Another key part of the development process is benchmarking and profiling the code to determine the

performance and scaling behaviour, as well as any areas of poor performance and potential for improvement. We will highlight available tools and methods to achieve this, both open source and commercial offerings.

Finally we will consider the need for testing to ensure correct behaviour. This can be particularly difficult for scientific software as typically the problems being solved have no analytic solution for direct comparison. We will discuss various approaches for effective testing, with particular emphasis on the benefits of automated testing as a part of the development cycle.

[1] Francisco Oliveira and Joao Tavares. "Medical image registration: A review". In: *Computer methods in biomechanics and biomedical engineering* 17 (Jan. 2014), pp. 73–93.

[2] Marco Palanca et al. "Local displacement and strain uncertainties in different bone types by digital volume correlation of synchrotron microtomograms". In: *Journal of Biomechanics* 58 (2017), pp. 27–36.

[3] D.C. Barber et al. "Efficient computational fluid dynamics mesh generation by image registration". In: *Medical Image Analysis* 11.6 (2007), pp. 648–662.

[4] Satish Balay et al. *PETSc Web page*. `http://www.mcs.anl.gov/petsc`. 2019.