

# A Tool for Combined Geodynamical and Thermodynamical Analysis

Student Name: [REDACTED]

Supervisor Names: [REDACTED]

Submitted as part of the degree of M.Sc. MISCADA to the  
Board of Examiners in the Department of Computer Sciences, Durham University

**Abstract** — The processes of melt generation and migration are important in the study of geodynamics in areas near the Earth’s surface and are one of the main reasons why the planet has such a diverse geology. In fact, without melting, it is likely that the Earth would just have a single, uniform rock type instead of the continental and oceanic lithospheres that we have today. Typically geodynamical codes modelling the melting process assume parametrised melting models that depend on the pressure and temperature, and do not take into account the fact that the melting process is highly dependent on the chemical composition of the solid rock. In this work we introduce a new software tool combining the geodynamical code ASPECT with the thermodynamical code Perple\_X thereby allowing for a composition-aware analysis of the melting process. We evaluate the functionality and performance of this code using several model setups and provide some recommendations for future improvements to the software. The code is open-source and freely available to view and edit.

**Keywords** — phase equilibria; geodynamics; magma migration; numerical modelling

## I INTRODUCTION

Within the mantle and lithosphere, partial melting can occur whenever there is a significant change in either the pressure ( $p$ ), temperature ( $T$ ), or chemical composition ( $X$ ), that is, the amount of various compounds, usually oxides, that the mantle material contains. Some examples of this in action would be a rising mantle plume, bringing hot material to the surface, or a subduction zone or mid-ocean ridge, where volatile compounds such as  $H_2O$  and  $CO_2$  are added to the composition, reducing the solidus temperature. Correct modelling of this melt is important because it plays a significant role in determining the dynamics of such systems.

The modelling of the convection of mantle material is done by *geodynamical codes*. These calculate the velocity, pressure and temperature fields by solving the compressible Stokes equation as well as an additional temperature equation using the finite-element method.

This work will be using the modern geodynamical code ASPECT (Kronbichler et al. 2012). It is built upon the powerful finite-element library deal.II (Bangerth et al. 2007) and improves upon previous codes (e.g. ConMan: King et al. 1990, and CITCOM: Moresi et al. 1996) by leveraging modern numerical methods, the most important to this work being adaptive mesh refinement and parallel performance that scales to thousands of processors.

Calculating melt volume within geodynamical codes is typically done by assuming a parametrised melting function (e.g. Dannberg and Heister 2016; Katz et al. 2003). However, significant simplifications are involved and no account is taken of the chemical composition. In order to get a more realistic calculation of the melt volume, so-called *thermodynamical codes* must be used. These are codes that accept  $p$ - $T$ - $X$  input conditions and calculate the stable phases of the system (e.g. Olivine, Quartz, melt) via a minimisation of the Gibbs energy. The Gibbs energy is one of several potentials widely used in thermodynamics and it is used here because it is minimised when the system is stable. Inside the code, it is represented by a series of nonlinear equations that are defined in thermodynamic databases (e.g. Holland, Green, et al. 2018). These codes also output a variety of other thermodynamical information, such as the entropy and density, that would be useful to more general geodynamic simulations not involving partial melting. See Connolly (2017) for a more detailed description of this process.

At present there exist many different thermodynamical codes, all specialising in some particular process or region of the mantle and lithosphere. Examples include: MELTS (Ghiorso and Sack 1995), THERMOCALC (Powell et al. 1998), HeFESTo (Stixrude and Lithgow-Bertelloni 2011), BurnMan (Cottaar et al. 2014), and RCrust (Mayne et al. 2016). In this work we will be using Perple\_X (Connolly 2005; Connolly 2009). Perple\_X is different to many of the other solvers because it converts the nonlinear problem into a linear one by discretising the nonlinear phase functions into a series of *pseudocompounds*. This is described in detail in Connolly (2005). Unlike the nonlinear solvers, this linear formulation of the problem may be solved efficiently using the Simplex algorithm and has guaranteed convergence, at the expense of introducing artefacts into the final phase diagram.

Previous efforts to integrate Perple\_X into geodynamical codes have taken one of two approaches. They have either precomputed the thermodynamical properties of interest and stored them in a lookup table (e.g. Magni et al. 2014; Bouilhol et al. 2015; Freeburn et al. 2017) or, more recently, they have made direct calls to Perple\_X, performing the minimisations during the simulation (e.g. Kaislaniemi et al. 2018). The former method is fast to run during the simulation but preparation of the lookup table can take a long time and it is not completely exact because interpolation is required at each reading. It also will only work for a fixed composition so is unsuitable for modelling melt transport where the composition will necessarily change. By contrast, the latter approach produces very accurate results and it will work with a varying composition, but until recently this has been prohibitively expensive to calculate.

Although the examples above used the older code CITCOM instead of ASPECT, ASPECT already includes some integration with Perple\_X (Myhill 2018). However, the integration code focuses on calculating material properties such as the density and compressibilities, rather than our specific investigation of partial melting. It is additionally both slow to run and difficult to read, both areas that this project aims to improve upon.

So far we have only discussed measuring the amount and composition of the partially molten rock, disregarding the actual melt migration dynamics essential to a realistic model. The principal difficulty in modelling melt migration is in handling the length and time scales that differ massively between the solid and liquid materials. Melt transport occurs very quickly in comparison to the slow motion of the solid mantle material and this poses challenges for the finite element software, in particular if adaptive mesh refinement is not supported.

The simplest method to model partial melting inside of a geodynamical model is called *batch melting* and involves disregarding the separate melt dynamics altogether. The melt components advect alongside the solid residue so the overall bulk composition does not change. This is the most straightforward to implement but also the least realistic.

Another possible approach is *fractional melting*. Here, the melt advects with the residue until a melt percolation threshold is reached, at which point the melt is removed from the bulk composition of the simulation. This is designed to model the situation where the small pockets of melt become connected and it can flow to the surface. Since the process happens much more quickly than the advection of the rest of the mantle, it is modelled as being simultaneous. Fractional melting allows for a more realistic simulation of the remaining depleted residue, but the melt dynamics are still ignored. The method has the additional disadvantage that the overall amount of material will decrease over time, making long-running simulations unfeasible. See Kaislaniemi et al. (2018) for an example comparing batch and fractional melting with Perple\_X and CITCOM.

Finally, the most accurate solution possible is to consider the melt as a distinct material that advects separately to the solid residue, unlike the previous methods mentioned that only track the residue. Solving this requires complex changes to the underlying equations of the geodynamical codes. So-called *two-phase flow* is naturally more computationally intensive than the previous approximations but it has already been implemented in ASPECT (Dannberg and Heister 2016).

In this work we present new software integrating Perple\_X into ASPECT, allowing for analysis of melt amounts and composition. Implementations of all three types of melt model are provided. We begin with a detailed discussion of the implementation and design choices (Section II) before applying the code to some model setups (Section III) and finally discussing issues and potential improvements (Section IV).

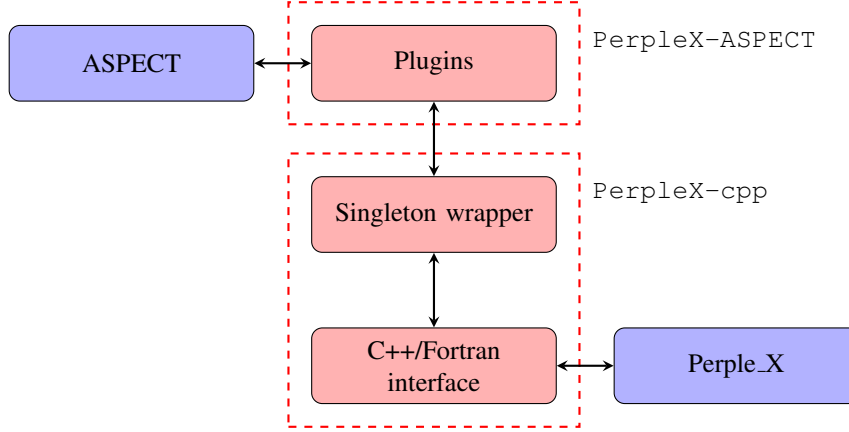


Figure 1: Diagram displaying the flow of data through the program. The red blocks represent the new code and the blue blocks represent the reused code. The libraries are denoted with the red dotted lines.

## II SOLUTION

The submitted code is split into two libraries, each kept in a separate repository. The first library, called `PerpleX-cpp` (<http://github.com/cward97/perplex-cpp>), contains the C++/Fortran interface and singleton wrapper that interface with `Perple_X`, and consists of both Fortran and C++ code. The second, called `PerpleX-ASPECT` (<http://github.com/cward97/perplex-aspect>), contains the plugin code that implements the wrapper inside of `ASPECT` and is written entirely in C++. The code for both libraries is compiled using CMake. A diagram displaying the relationship between the repositories as well as `Perple_X` and `ASPECT` is shown in Figure 1. Alongside the source code, a wiki is provided giving useful information about topics such as setting up a development environment on the Hamilton supercomputer, viewing the results in Paraview, and using Docker.

The justification for splitting the code into two repositories is twofold. Firstly, it ensures that the `Perple_X` wrapper code is entirely general and the library can be used outside of `ASPECT`. Secondly, it enforces a separation between `ASPECT` and `Perple_X`. This is potentially valuable if the plugin code were to be integrated upstream into the main `ASPECT` repository.

Regarding the code licensing, both `Perple_X` and `ASPECT` are licensed under the GNU Public Licence (GPL) version 2. In order for the submitted code to be compliant with these licences it has been licensed using the more modern GPL version 3.

### A *PerpleX-cpp*

`Perple_X` consists of a suite of several executables written in Fortran 77. The executable of interest to this project is `MEEMUM` which computes the stable phases, and their respective compositions, at a given point in  $p$ - $T$ - $X$  space.

Running `MEEMUM` is generally straightforward although collating a reasonable set of inputs can sometimes be challenging. Different versions of `Perple_X` sometimes refuse to run with certain input files because of minor changes to the way the files are parsed. The inputs to the program are: (a) a problem definition file containing the bulk composition made by another `Perple_X` binary `BUILD`; (b) a solution model file; (c) a `Perple_X` option file; and (d) a thermodynamic database file (e.g. Holland and Powell 1998). When the program is run it prompts the user for  $p$ ,  $T$  and  $X$  values and prints the results to the screen. The aim of the wrapper is to automate this process providing the  $p$ ,  $T$  and  $X$  conditions as inputs to a function returning the phase information that would otherwise be printed. The various input files are untouched and are read when the wrapper is first initialised.

#### A.1 Original `Perple_X` source code

Interacting with the `Perple_X` code is a complex task and one of the more difficult of this project. It is written in Fortran 77 and thus has certain features not encountered in a modern programming language,

the most significant for this work being the extensive use it makes of COMMON blocks. In addition, the tools in Perple\_X are designed exclusively for interactive use. There is no clear separation between the (command line) user interface and the program logic so there are, for example, random places in the code where the user is polled for input. This makes it challenging to automate the code without making radical changes to the codebase.

The decision was made to directly include the source code inside the repository rather than either encouraging the user to install Perple\_X themselves or downloading and compiling the latest version<sup>1</sup> during the build. The principal reason for this is reliability. Perple\_X receives frequent breaking updates and there is no easy way to access old versions, so the source code is stored inside the repository in order to ensure that the code does not break unexpectedly. The version used in the repository at the time of writing is 6.9.0.

When Perple\_X is downloaded it comes with many files that compile to different executables other than MEEMUM as well as lots of thermodynamic database files and various READMEs. To avoid confusion, only the necessary source files are included inside the repository.

## A.2 C++/Fortran interface

The basic concept behind integrating Fortran code into C++ is very simple. All that is required is a C++ header file declaring the required Fortran functions and variables using the correct symbols (e.g. the function `f00()` is normally represented with the symbol `f00_`) and then the codes can be linked during compilation. However, actually implementing this is far from straightforward: many of the data types are incompatible, array indexing starts from zero (C++) or one (Fortran), and arrays themselves are alternately row-major order (C++) or column-major order (Fortran).

This problem is compounded in Perple\_X by the sheer number of global variables, both constant PARAMETERS and COMMON blocks that are used by the program. The source code contains a 400 line file called `perplex_parameters.h` that contains the program parameters as well as many frequently used COMMON blocks. If one were to write a C++ wrapper then this file would need to be converted into a C++ header too.

This was done previously by Kaislaniemi (2015); the parameter file was rewritten by hand into C++. Myhill (2018) improved upon this by writing a Python script that ‘compiled’ the first part of the Fortran parameter file into a C++ header although the latter half still needed manually writing out.

This approach has a number of drawbacks: (a) whenever Perple\_X is upgraded this script must be re-run; (b) the process is complex and hard to debug in case errors occur; and (c) the resulting header file is very hard to read.

In this work, to solve these issues, an additional Fortran file (`f2c.f`) was created. Rather than being converted to C++, the parameter file is simply included once in this file, permitting a much simpler C++ header file (`f2c.h`) to be written. The new interface provides a small number of getters (e.g. `get_endmember_density()`) and setters (e.g. `solver_set_pressure()`) as well as a largely verbatim copy of the MEEMUM subroutine, only altered to eliminate any calls for user input and split into initialise (called once) and compute (called many times) functions. Arguments are passed by value (C-like) and only single values are passed (excluding strings), avoiding any complexities involved with arrays.

An additional advantage of this approach is readability. Having been written in Fortran 77, Perple\_X has a six character limit on its variable names, which can make it very hard to follow what the variables and functions do. The new interface code was not so restricted, allowing for much more verbose and understandable function names.

A final point about the interface is that, being effectively a duplicate of the original code, the ‘warm start’ functionality of Perple\_X remains active. This is an obscure feature found in the Perple\_X source code<sup>2</sup> and has unknown performance benefits. This feature will be ignored for the rest of the document but is recorded here for future reference.

<sup>1</sup>The latest version of Perple\_X may be downloaded from <https://petrol.natur.cuni.cz/~ondro/perplex>.

<sup>2</sup>The hot start features are implemented in the main `lpoprt0` optimisation routine.

### A.3 Singleton wrapper

Although the basic interface described above is a significant improvement on previous approaches, it is still fairly limited. Only basic data types are accepted and its use requires interacting with global variables, something that is considered poor software development practice.

Our solution to both these problems is the creation of a `Wrapper` class. The class presents initialise and compute methods to the user and hides the details of interfacing with the Fortran completely. Additionally, it permits the use of more complex data types, such as `std::vector` and `std::string`, that are in common use in ASPECT. An object-oriented solution was chosen because `Perple_X` consists of both functions and data, making it a suitable candidate for encapsulation into a class.

The class also implements the *singleton pattern*; that is, only a single instance of the class may ever exist. This approach was taken in order to prevent concurrent accesses to the global `Perple_X` data structures that would otherwise occur if multiple objects existed all accessing the same state.

Having been designed as an interactive program, `Perple_X` prints a lot of information to the screen whenever MEEMUM is executed. Whilst appropriate for use as the main program, it becomes problematic inside the wrapper because it obscures other useful output from the rest of the program. This is a particular issue if it is called frequently and on multiple processors as the output quickly becomes unreadable. To avoid this problem, the standard output stream is disabled during the calculations and re-enabled when they are completed. This fixes the stated problem but introduces another issue. Because `Perple_X` does not write its error messages to the standard error output stream but instead to standard output, which is disabled, any error messages from `Perple_X` will be hidden and the program may crash without warning. In order to permit the re-enabling of console output in these cases, an `ALLOW_PERPLEX_OUTPUT` compiler flag is introduced, although it is disabled by default.

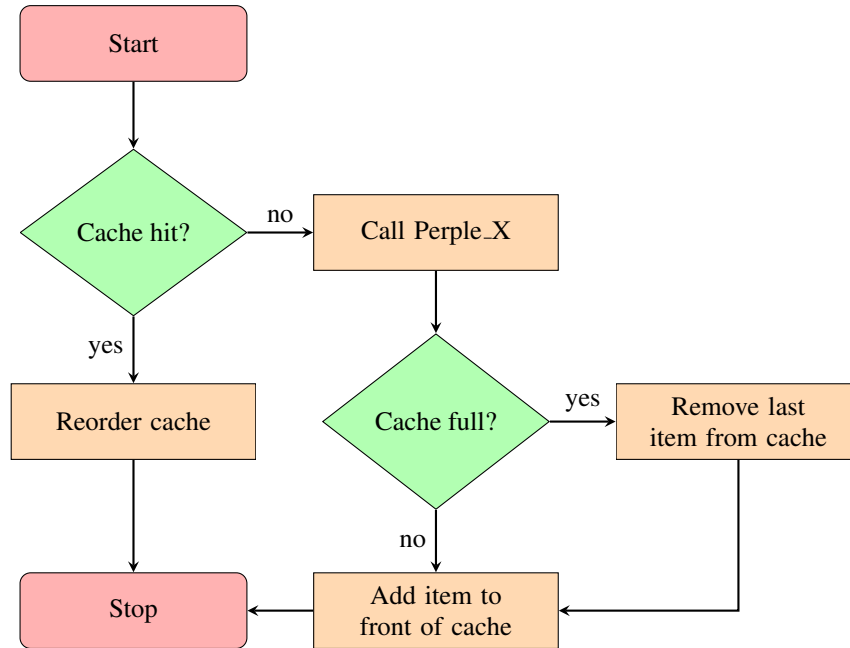


Figure 2: Flowchart showing the decision process for the least recently used (LRU) cache used by the `Wrapper` class.

During the project, it was noticed that many computations were being done with identical or almost identical inputs. It therefore made sense to store the most frequently used results and use them if subsequent inputs are sufficiently similar. A cache was implemented using a least recently used (LRU) eviction policy. In other words, it only stores the most recently used results, and results that have not been needed for a while are removed to save memory and must be recomputed if needed again. A flowchart detailing the decision process for an LRU cache is shown in Figure 2. Cache accesses are defined as being *hits*, where a matching result is found and returned, and *misses*, where no match is found. The *hit rate* is simply the number of cache hits divided by the total number of cache requests. In this work,

a cache hit is considered to have occurred when each of the submitted  $p$ - $T$ - $X$  conditions varies by less than a specified relative tolerance, normally on the order of  $1 \times 10^{-3}$ .

#### A.4 Testing

The library is tested using the Google C++ unit testing framework Google Test. For the C++/Fortran interface and singleton wrapper class, the tests consist largely of a direct comparison between the results of the function calls and previously collected Perple\_X output for a quick-to-run sample data set. More traditional unit tests testing the behaviour of individual functions have also been written for the cache implementation.

#### A.5 Parallelism

Perple\_X is not designed for parallel execution and as such the library is not thread-safe. However, there are only a few methods that are exposed to the user so it would be straightforward to add locks to them, making them thread-safe. In contrast, the code is entirely suitable for parallelism in a distributed memory environment (i.e. MPI) because the global state is still local to each processor. MPI is the primary parallelisation method used within ASPECT and so thread-safety was not considered a necessary feature of the library.

### B PerpleX-ASPECT

The Perple\_X wrapper code was integrated into ASPECT using its powerful *plugin* system. Within ASPECT, plugins are pieces of code that fulfil a particular piece of functionality; for example, declaring the initial temperature profile.

When it comes to integrating the new plugins with ASPECT, there are two main options: forking and modifying the main ASPECT repository, or creating another repository and compiling it into a shared library so that it can be linked to ASPECT at runtime. The latter was chosen for the following reasons: (a) updating ASPECT does not require either rebasing or merging the git repositories; (b) not having the new code mixed in with the original code makes things clearer; and (c) the code is rather specialist and may not ultimately be added to the ASPECT codebase.

The plugins expose to the user a set of options that may be included in the parameter file that is submitted to ASPECT. Given the small number of source files that are written, no automatic method of generating the documentation has been written and we recommend reading the source code for parameter details. Some example parameter files are included in the `cookbooks` directory in the repository providing examples of how to use the plugins.

#### B.1 Particle property plugin

Within ASPECT there exist several plugins related to tracking particles, sometimes known as tracers, around the simulation. In this work, an additional *particle property* plugin was written that returns phase information from Perple\_X using the pressure, temperature, and (at least initially) the bulk composition given in the problem definition file.

The basic idea of particles in a geodynamical simulation is that they are point objects passively advected around the medium with the velocity field. They store a set of properties that are updated at specified time steps using the state of the different fields present at their current position.

It is easy to control the performance of particles inside of a simulation because both the increment between updates and the number of particles are options that may be specified in the parameter file. This is particularly useful for investigations with Perple\_X because each evaluation can take a considerable amount of time to complete. An additional benefit of using particles is that they do not, unless desired, interfere with the rest of the simulation. This is an advantage over the material model plugin discussed next which restricts the code to using a single material model. However, it is important to note that performance is poor when dealing with large numbers of particles, as the implementation still requires some optimisation, or when using a highly-adaptive mesh, as load balancing does not take into account the number of particles and instead only considers the number of degrees of freedom that are related to the cell count.

The plugin has been designed to accept many different parameter options permitting access to many of the results reported by `Perple_X` and the wrapper code. Results for any of the possible solution phases, including the melt, are available and properties such as the composition, molar amount and volume fraction may all be specified. It would be straightforward to extend the list of available properties to include any that are reported as output from MEEMUM.

One particularly noteworthy set of options permitted by the plugin are `Extract_melt` and `Melt_extraction_threshold`. These permit an investigation of fractional melting of the material. If `Extract_melt` is set to `true` then whenever the melt volume fraction exceeds the specified threshold, typically around 5%, then the melt is removed from the simulation, altering the bulk composition accordingly. If this option is enabled, then the plugin will automatically report on the amount of moles and composition of both the melt that is extracted and the melt that is present within the bulk material. Relative properties such as the volume fraction of the extracted melt are not tracked because they are non-physical quantities.

## B.2 Material model plugin

Despite its many benefits in performance and simplicity, the particle property plugin is incompatible with a study of two-phase flow which is desirable for a realistic model of melt behaviour. This is because two-phase flow requires that the composition of the melt be advected separately to the composition of the residue, one being solid and the other a fluid. In ASPECT, particles can be set to either advect with the melt field *or* with the solid material, but not both. In this work we avoided this problem by pivoting to a compositional field-based approach and implementing a new *material model* plugin.

Compositional fields are extra fields in ASPECT whose motion is solved by one of the four equations used in ASPECT:

$$\frac{\partial c_i}{\partial t} + \mathbf{u} \cdot \nabla c_i = q_i.$$

The compositional fields  $c_i$  are advected with the flow velocity  $\mathbf{u}$  with an optional *reaction term*  $q_i$  to handle reactions between the fields. Although similar to particles in that they are advected with a velocity field, compositional fields are able to be advected with the melt velocity field as well as with the usual, solid velocity field which is what makes two-phase flow possible.

For our work, several compositional fields were introduced, each representing a given chemical component for either the melt or the residue, as well as an additional compositional field called the *porosity* that must be included for models with melt transport and represents the volume fraction of the melt. In our investigations, given that the `Perple_X` model we were using specified four chemical components, we used nine compositional fields: four for the melt, four for the residue, and the porosity.

An unfortunate downside to this approach is that the parameter files become significantly more complex as the compositional fields must be specified by name and must correspond exactly to the compounds specified in the `Perple_X` problem file. Examples demonstrating the correct usage of the plugin may be found inside the repository.

In ASPECT, the material model is responsible for calculating the various coefficients, such as the viscosity, density and specific heat, that are required in order to solve the main equations. Reactions between compositional fields are controlled by specifying reaction terms (zero by default). The principal method of any material model plugin is called `evaluate()` which is called multiple times at every time step to calculate the coefficients. These coefficients are often evaluated at multiple quadrature points inside of each cell.

The material model plugin that was implemented inherits from the `melt_simple` material model found in the main ASPECT repository. This choice was made because writing a material model from scratch that implements partial melting was found to be very difficult leading to catastrophic convergence errors. Instead, the new model simply extends the `evaluate()` method and overrides the `melt_fractions()` method. In both of these methods, the chemical composition is extracted from ASPECT's compositional fields and passed to the `Perple_X` wrapper along with the pressure and temperature. The result from the calculation is then used to either report on the volume fraction of melt or update the compositional fields.

The decision to use the `melt simple` model as the base over the very similar `melt global` model was arbitrary and the submitted code could easily be adapted to inherit from that model instead.

For models with melt migration, the processes of melting and freezing typically occur on a much faster time scale than even the flow of the melt itself. In order to decouple the two processes the *operator splitting* scheme is used, where for every advection time step, multiple reaction time steps occur. A consequence of this method is that there are a great many calls to `Perple_X` but this is necessary for a realistic model.

Finally, so as to resolve the melt reaction and advection processes with a higher resolution than the solid material, the `composition threshold` mesh refinement strategy is used to ensure that areas where melt is present are refined to a greater extent.

### B.3 Additional plugins

In addition to the plugins discussed above, *postprocessor* and *initial composition* plugins were also written. The postprocessor plugin is called `perplex cache statistics` and it permits analysis of the cache in the `Perple_X` wrapper. The initial composition plugin is called `perplex composition` and it makes sure that the compositional fields representing the melt and residue compositions (for two-phase flow) are initialised correctly using the initial `Perple_X` data file for the bulk composition.

### B.4 Testing

Unit testing is difficult to do in ASPECT, as evidenced by the fact that very few unit tests actually exist in the main repository, due to the fact that plugins are tightly integrated with the rest of the simulation. Most plugins inherit from the `SimulatorAccess` class in order to access information from the rest of the simulation such as time step number or to interact with other plugins. Such tight coupling makes unit testing of single pieces of functionality practically impossible.

Instead of using unit tests, ASPECT relies upon *benchmarks* for evidence of its accuracy. These involve running simulations with a known answer, for example the temperature change over a phase transition, and verifying that the ASPECT code converges to the correct answer. The decision was made to not include any benchmarks with the code because they require detailed domain-specific knowledge and were impractical to include given the time constraints of the project.

The other form of testing used in ASPECT is much less informative. The `tests` directory in the repository contains hundreds of parameter files, each very quick to run and only a few time steps in duration. Also stored are the output files produced when running each of the parameter files. This allows for a very simple test to verify that the code runs as expected by simply rerunning the model setups and checking that the outputs have not changed. In ASPECT the process has been cleverly automated and relies on utilities such as `diff` to determine a pass or fail. The implementation of this is rather complicated so in our work we rely upon making manual comparisons of the outputs. A few simple tests of this nature are included in our repository.

## III RESULTS

In the following, the functionality and performance of the libraries will be demonstrated using two model setups run in ASPECT (Table 1). The scripts necessary to reproduce the results, including the plotting ones, are available at <http://github.com/cward97/miscada-report>.

The first model setup to be investigated was a generic scenario (henceforth referred to as the “closed box” scenario) in which an enclosed amount of material advects inside a box. Temperature fluctuations at the base of the model, specified in the parameter file, create convection currents transporting hot plumes of mantle material upwards. As they rise towards the surface, the reduction in pressure causes partial melting. Some screenshots of the setup running with both plugin types are shown in Figure 3. They show melt forming at the head of the two mantle plumes where the temperature is greatest and also disappearing as the plume disperses and the temperature drops. At the end of the simulation, the temperature field is mostly mixed and the hot material has almost entirely risen above the colder material that was initially at the surface.



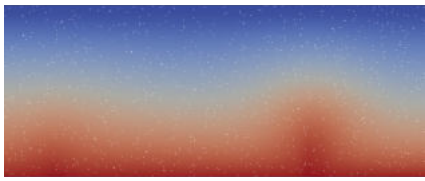
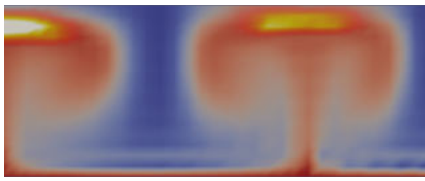
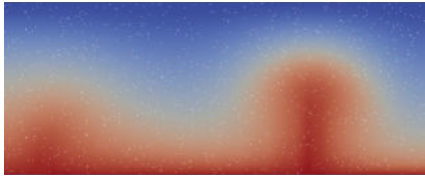
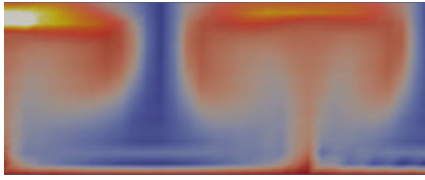
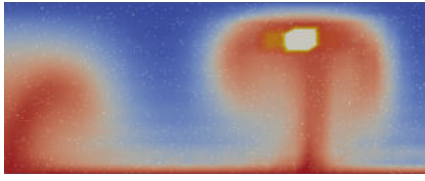
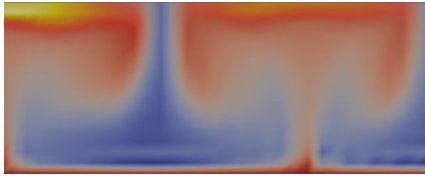
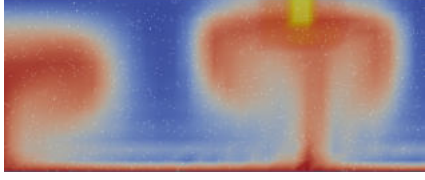
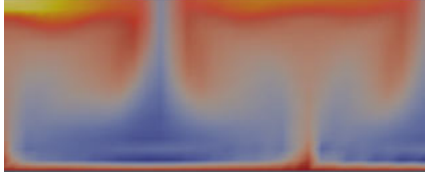
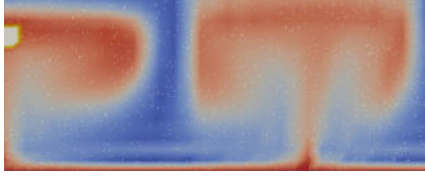
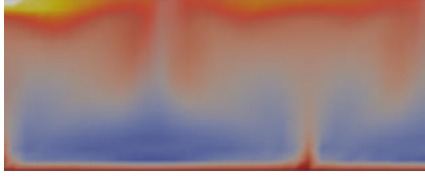
<i>Time (yr)</i>	<i>Particle property plugin</i>	<i>Material model plugin</i>
677 115		
800 992		
958 796		
$1.0513 \times 10^6$		
$1.280 82 \times 10^6$		

Figure 3: Example runs of the closed box model setup with both plugin types. 2000 particles were used when using the particle property plugin and none were used in the material model one. The temperature is represented by the red to blue colour map and the melt is yellow and white. Whilst the simulation was run for 3 million years, only the period of 600 000 years is shown because it shows the principal behaviour. Before this the velocities of the particle property plugin are too low to show any dispersion behaviour, and at later times the plumes disperse and the melt disappears.

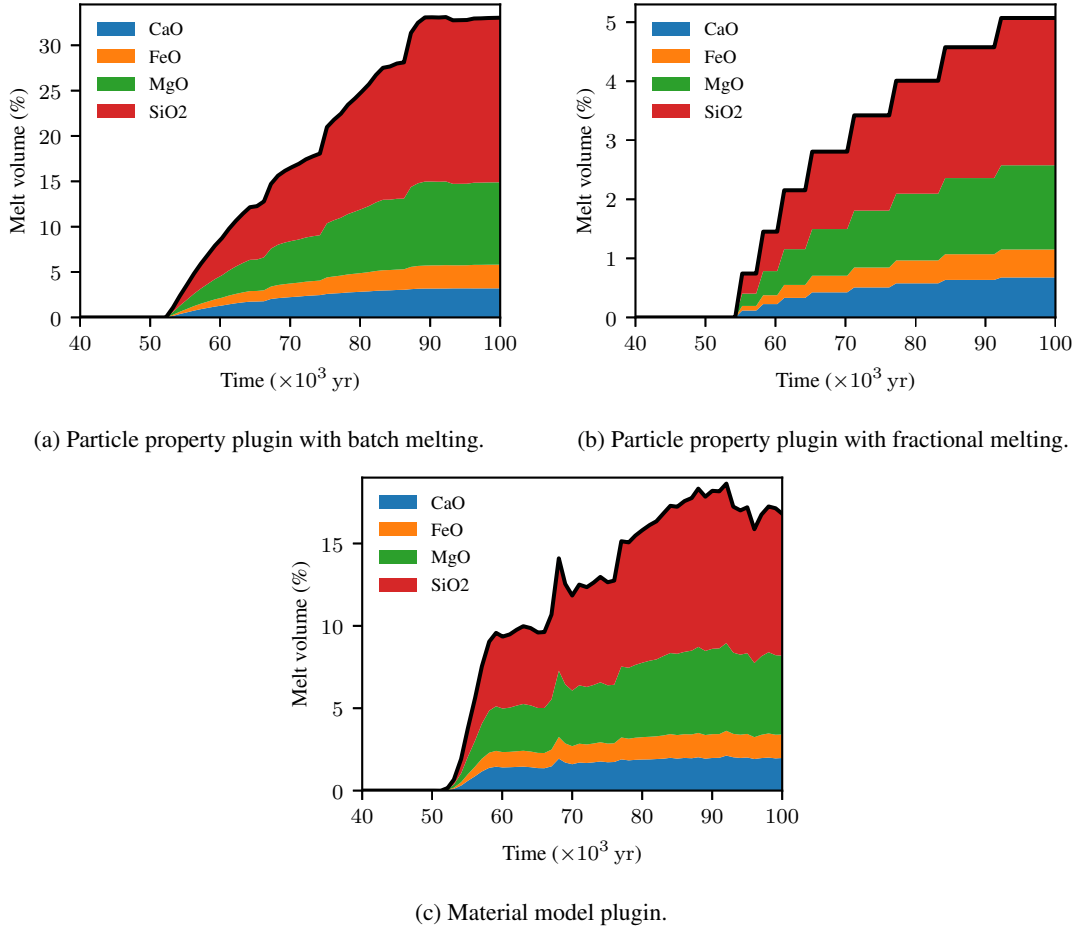


Figure 4: Average melt amount and compositional information recorded by the particles during the decompression event model setup for the different plugin types. The first 40 000 years are omitted because no melt is present. Note that subfigure (b) the extracted melt and the melt amount are recorded instead of melt volume because the melt volume is unknown for the extracted melt.

<i>Parameter</i>	<i>Closed box</i>	<i>Decompression event</i>
Height (km)	120	120
Width (km)	300	1
Surface pressure (Pa)	0	0
Surface temperature (K)	293	293
Bottom temperature (K)	1873	1873
Temperature profile	Linear	Linear
Temperature boundary conditions	Fixed at base	Fixed at base
Temperature perturbations at base	Present	Not present
Boundary velocities	Tangential	$1 \text{ m yr}^{-1}$ upwards
Reference bulk viscosity (Pa s)	$1 \times 10^{18}$	$1 \times 10^{18}$
Reference shear viscosity (Pa s)	$1 \times 10^{18}$	$1 \times 10^{18}$
Thermal viscosity exponent	4	4
Thermal bulk viscosity exponent	4	4
Perple_X data file	Simplified KLB-1	Simplified KLB-1

Table 1: Parameter values used in the two model setups. A detailed description of the meaning of the different parameters may be found in the ASPECT manual. More information regarding the Perple\_X data file may be found in Section IVA.

When implemented using the particle property plugin, particles were added to the simulation tracking the melt volume. Since batch melting was used, the chemical composition that matched the bulk composition in the *Perple\_X* data file stayed fixed throughout. An initial mesh refinement level of 5 was used and adaptive mesh refinement was disabled.

For the material model plugin, no particles were required and instead the melt volume could be plotted directly from the porosity compositional field. The composition initially also matched that found in the *Perple\_X* data file, but was able to change over time as the melt and residue advected separately. An initial mesh refinement level of 4 was used, but adaptive mesh refinement was included in order to better resolve the areas where melt is present.

The second scenario is a less general model aimed at modelling individual “decompression events” during which a single hot column of material rises and begins to melt. The amount of melt is tracked by 11 particles, initially placed at the base of the model, that move upwards at the same velocity as the temperature field (specified as  $1 \text{ m yr}^{-1}$ ) and keep track of the melt amount and composition that forms as the pressure reduces. The simulation ran for 100 000 years as this was enough time for the particles, moving at  $1 \text{ m yr}^{-1}$  in a column 120 km high, to approach the surface without being removed.

An advantage to this model setup is that it is a more intuitive way to study the impact of fractional melting. The closed box setup is designed to run continuously, and so frequently removing the melt from the simulation will quickly make the results unrealistic because the extracted melt is never reintroduced. In contrast, the decompression event setup has a fixed endpoint so the fact that the melt is not reintroduced is not an issue.

The results are displayed in Figure 4. In all cases, as the particles migrate upwards and the pressure decreases, the material begins to partially melt leading to an increase in the melt volume. The chemical composition of the melt is also shown and appears to be similar between the various models tested.

Interestingly, only about half as much melt is formed by the material model plugin compared to the particle property one. One possible explanation for this effect would be depletion. Since in the material model the melt advects separately to the residue (and the particles advect with the residue), the bulk composition changes over time, taking away the more volatile components and reducing its melt productivity.

## A Performance analysis

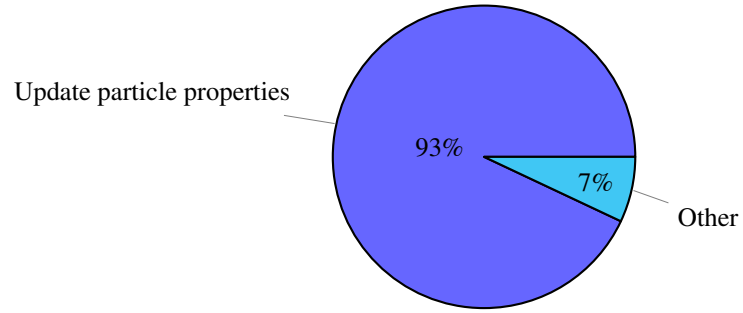
Having proved the feasibility of the code to make physically sensible results, the next stage is to measure the performance of the two plugins. In order to do this, the closed box model setup was used with varying parameters. This model was selected instead of the decompression event model because it was simpler, since it would not need to deal with particles or material being removed from the simulation.

A control setup was used for the particle property plugin, in which the same number of particles was tracked by the simulation but the specific *Perple\_X* properties were not tracked. Unfortunately, no control scenario could be created for the material model plugin. This was because the parametrised melt fraction equation in `melt_simple` produced a lot more melt than observed in *Perple\_X* for the same initial conditions. This resulted in much more adaptive mesh refinement and also the solvers took a lot longer to converge to the correct value making any sort of run time comparison impossible.

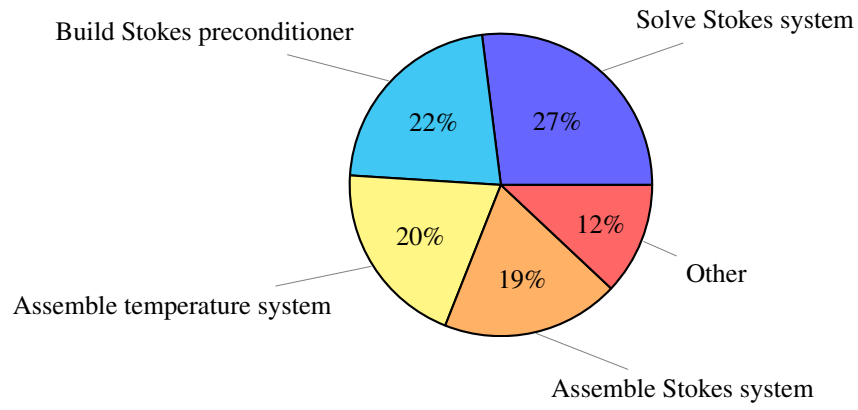
In the following analyses, only a single run was used rather than the average of multiple runs. This approach was chosen because this report only aims to identify general trends in the data and not make precise performance measurements. In all cases though, care was taken to “hot-start” every calculation whereby a dummy run was done before taking any measurements to try and mitigate library load times and other similar effects.

### A.1 Run time

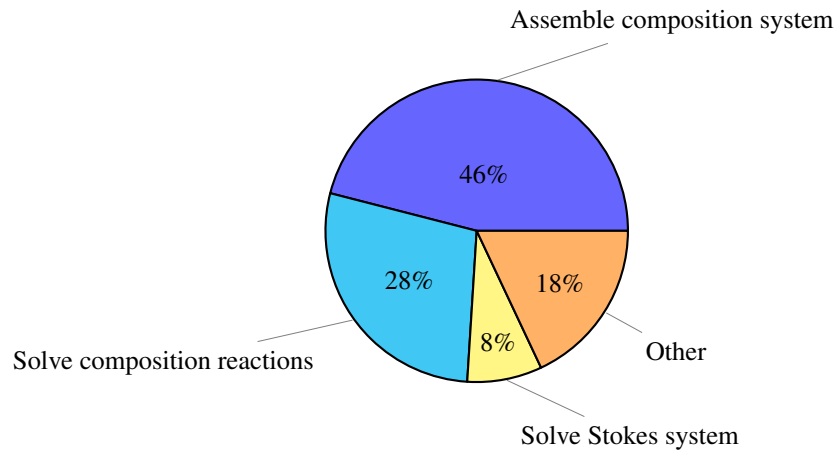
An accurate comparison of the run times for the particle property and material model plugins is difficult to achieve because they work in totally different ways. That notwithstanding, it is certainly the case that the particle property plugin runs more quickly than the material model plugin. Running both plugins with the same input files reveals that the latter plugin takes roughly twice as long to run as the former. Breakdowns of the various parts of the runtime are displayed in Figure 5.



(a) Particle property plugin.



(b) Particle property plugin control.



(c) Material model plugin.

Figure 5: A comparison of the breakdown of runtime spent in the different elements of ASPECT between the particle property and material model approaches to tracking the composition.

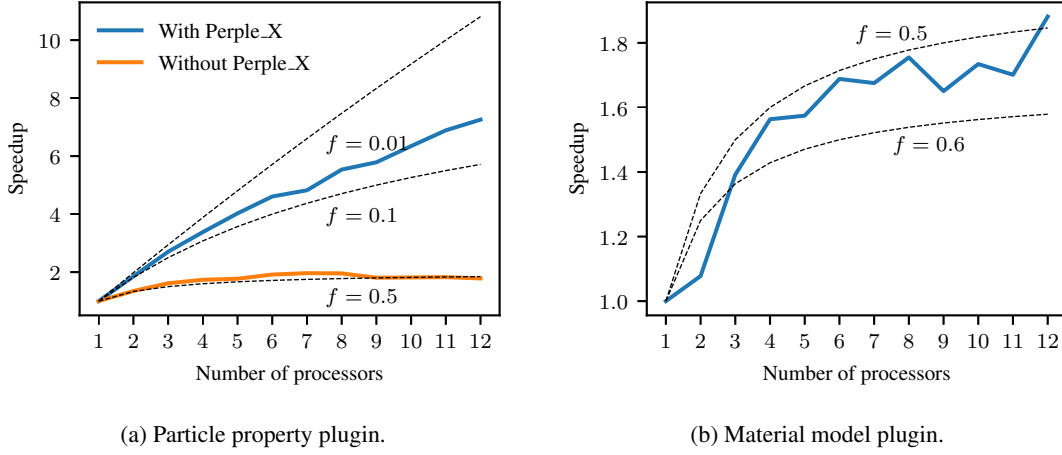


Figure 6: Speedup of the plugins compared with theoretical constraints. Note that the problem size has remained fixed for each run.

## A.2 Scaling

Of primary importance to the performance of the plugins is how well they scale with the number of processors. The greatest number of processors that was used in this work was 12, but geodynamical simulations have processor counts that often run into the thousands of cores and good scaling performance is essential in order to take advantage of them.

A common metric for measuring scaling performance is the *speedup*  $S$ . This is defined as:

$$S(p) = \frac{t(1)}{t(p)},$$

where  $t(1)$  is the runtime for the problem on a single processor and  $t(p)$  the runtime on  $p$  processors. Naturally the optimal speedup achievable as  $p$  increases is  $p$  itself (i.e. doubling the number of cores should halve the compute time). However, this is never achievable in reality as some parts of the code will always need to be run sequentially rather than in parallel. There will also be additional overhead to consider that derives from increasing parallelism and this often scales with  $p$ .

Speedup curves may be predicted with Amdahl's Law (Amdahl 1967):

$$t(p) = ft(1) + (1 - f) \frac{t(1)}{p}.$$

It simply states that the fraction of the code that must be run sequentially,  $f$ , will experience no speedup and the fraction of the code that can be run in parallel,  $1 - f$ , will experience the optimal speedup of  $p$ . The 'law' is a major simplification and does not account for any parallel-induced overhead, but it serves as a useful technique for modelling  $f$ .

The speedups for each plugin are shown in Figure 6 and they are plotted against the curves predicted by Amdahl's Law for given values of  $f$ . For the particle property plugin (Figure 6a) one sees that  $f$  lies between 0.01 and 0.1, which is very near optimal and means that significant speedup is observed as the number of processors increases. Conversely, the control model setup shown in the same plot has  $f$  roughly equal to 0.5.

Turning our attention to the material model plugin (Figure 6b), the speedup is much less pronounced and  $f$  is shown to lie approximately between 0.5 and 0.6.

## A.3 Load balancing

When running a program in parallel, it is important not only that the code be divided into pieces that can run on each processor, but also that the divisions be made equally in order to avoid processors idling unnecessarily.

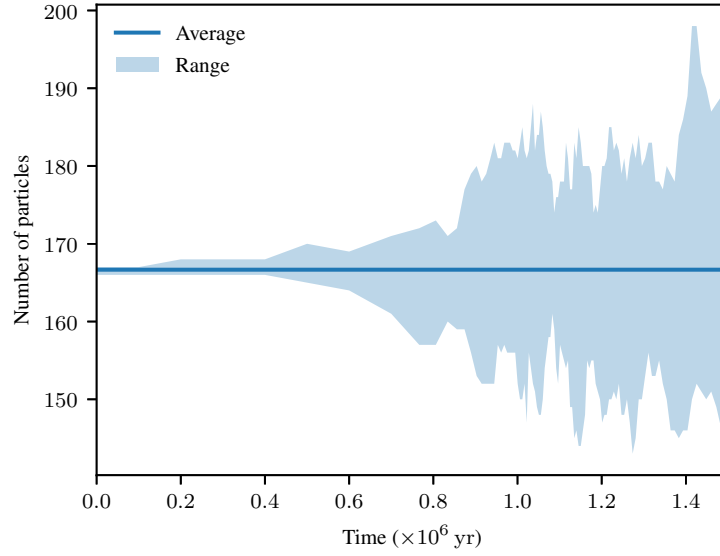


Figure 7: Load balancing across processors for the particle property plugin. The data shown reflect the number of particles per process and the range indicates the maximum and minimum number of particles for each. The data were collected using 2000 particles on 12 MPI processes.

The load balancing information for the particle property plugin is shown in Figure 7. At first the particles are almost perfectly uniformly distributed about the domain and have very low velocities, leading to a very small difference between the maximum and minimum number of particles per process. However, as the plumes begin to form at around 500 000 years (see Figure 3 for screenshots), the particle velocities increase and the uniformity breaks down. At the end of the simulation there is an imbalance of approximately 30 % between the number of particles per processor. The particle distribution will follow a normal distribution and so this increase is simply a consequence of the normal curve gradually smoothing out. Due to the fact that the model setup does not permit material to leave the simulation, the number of particles was unchanged and so the average number of particles per process was constant.

No results were reported for the material model plugin because, as no adaptive mesh refinement was found to have occurred during the simulation, the load was perfectly balanced between the cells throughout.

#### A.4 Cache usage

Cache utilisation for the `Perple_X` wrapper was recorded for both plugins over a range of tolerances and the results are shown in Figure 8. In all cases, the first time step has a significantly lower hit rate due to mandatory misses (i.e. the cache is empty so there are no results to reuse). This is followed by a spike to a peak value for that tolerance, likely caused by the fact that the initial velocity in the entire domain is close to zero and therefore the  $p$ - $T$ - $X$  conditions are unchanged between time steps resulting in a cache hit. Around 500 000 years drops in the hit rate can be observed. This time corresponds with the point where turbulent flow becomes more obvious and the particle velocities increase leading to greater variation in the pressure and temperature causing cache misses.

Although this general trend may be observed for both plots, the hit rate for the material model plugin is extremely high at around 99.5 %. This is due to the large number of iterations as a consequence of using the operator splitting method. At each time step, so as to effectively model the rapid reaction rate of the partially melting substance, a great many evaluations are made of the material model with very tiny variations in the  $p$ - $T$ - $X$  conditions, leading to a very high rate of cache reuse, even for high tolerances.

A final observation from the figure may be made regarding the particle property plugin. Right at the end of the simulation a jump can be seen in the cache hit rate. This may be explained by observing that there is also a sudden drop in the time step duration, meaning that the particles have displaced less and their respective  $p$ - $T$ - $X$  conditions will not have changed much.

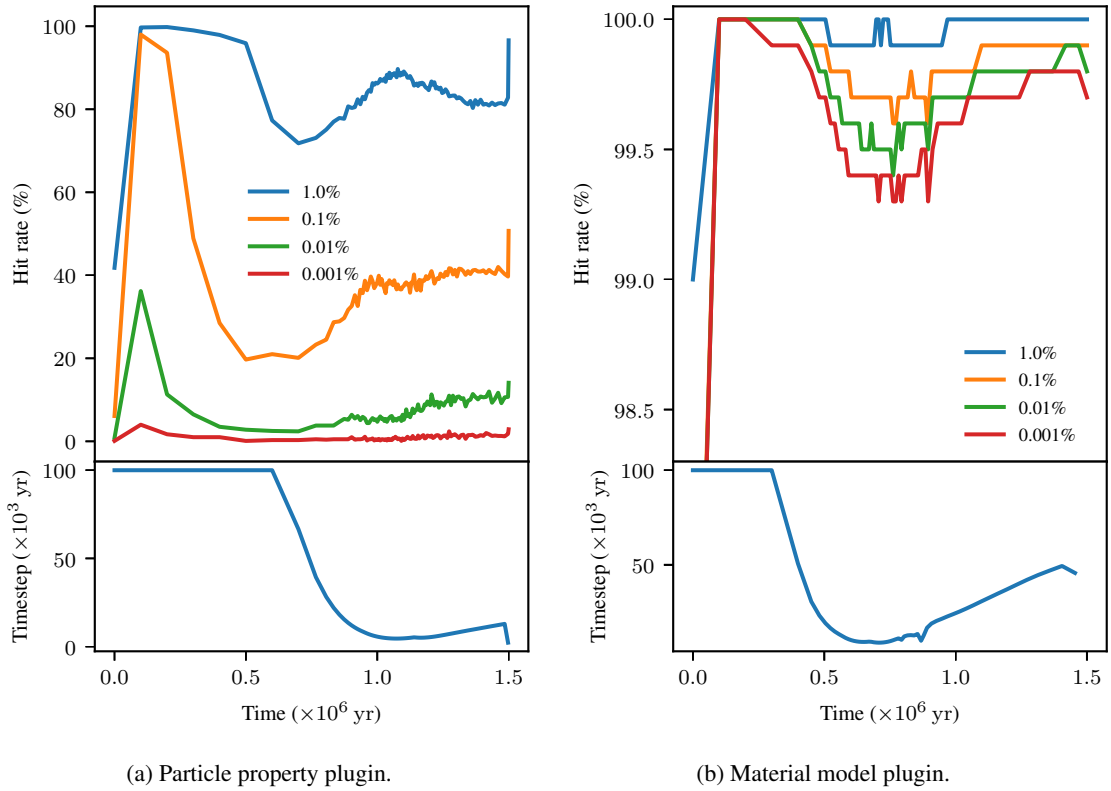


Figure 8: Cache utilisation and time step size against time.

## IV DISCUSSION

### A *Perple\_X* data file accuracy

For the *Perple\_X* problem definition file it was important to use a close approximation of the chemical composition of the upper mantle. The file initially selected was created by Debaditya Bandyopadhyay and modelled KLB-1 peridotite (Takahashi 1986) in order to reproduce the results from Holland, Green, et al. (2018). Unfortunately, the problem file is fairly complex so any single minimisation takes several seconds to complete, posing problems for running many such simulations in ASPECT. The complexity is due to there being many chemical components used in the file, and these massively increase the number of pseudocompounds used by the linear solving algorithm. In addition, a further, more critical, problem was encountered when we tried running the material model with this data set. It turns out that, every time MEEMUM is called, data is added to a static array which slowly fills up<sup>3</sup>. The array fills up much faster for the complex KLB-1 data file than for simpler problems and so after a short while the array overflows and the program crashes.

No fix could be found for this problem within the time constraints of the project, so we were forced to perform the experiments using a simpler data file, initially intended just for testing (see Table 2 for a comparison of bulk compositions of the two). This simpler file was based upon the KLB-1 data set, so as to remain as consistent as possible with the mantle region of interest, but the composition components with the lowest concentrations, including water, were omitted. This resulted in a data file that could be solved by MEEMUM in a fraction of the time and was also free from the array overflow error. Unfortunately, having constructed the file out of a desire for fast performance rather than scientific validity, the results reported above cannot be considered particularly realistic.

To improve the accuracy we present three options: (a) use a slightly more accurate data file that still has relatively few pseudocompounds; (b) use an advanced data set such as KLB-1 but limit the number

<sup>3</sup>The array causing the overflow error is called `sco` and is defined inside of `perplex_parameters.h`.

<i>Chemical component</i>	<i>KLB-1 (mol)</i>	<i>Simplified KLB-1 (mol)</i>
K <sub>2</sub> O	0.01	-
Na <sub>2</sub> O	0.25	-
SiO <sub>2</sub>	38.46	38.50
Al <sub>2</sub> O <sub>3</sub>	1.77	-
CaO	2.82	2.82
MgO	50.53	50.50
FeO	5.88	5.88
TiO <sub>2</sub>	0.07	-
Cr <sub>2</sub> O <sub>3</sub>	0.11	-
O <sub>2</sub>	0.05	-

Table 2: Bulk composition of the KLB-1 and simplified KLB-1 data files. Only the four most abundant components are used in the latter.

of calls to *Perple\_X* to restrict the rate at which the array fills up; or (c) alter either the C++/Fortran interface code or the *Perple\_X* source directly to avoid the overflow issue entirely. We propose option (b) as the best solution. Option (a), while improving the results, will still lead to an inaccurate solution and is fundamentally limited. Option (c) is highly complex and it is preferable not to alter the original *Perple\_X* source code (as discussed in Section A.1). By comparison, option (b) will not affect the validity of the results, assuming that the rate of evaluations is not too coarse, and has the potential to dramatically increase the performance by reducing the number of calls to *Perple\_X*.

Some specific examples of how this could be achieved are: (a) introduce a  $p$ - $T$  threshold, that may be a function, that only asks to evaluate *Perple\_X* when there is likely to be melt present; or (b) add an additional option to the material model so that *Perple\_X* is evaluated at specified times, for example every 10 000 years. Option (b) mirrors the approach taken by the particle property plugin although such an approach could cause numerical instabilities with the continuous compositional fields, which are designed to accommodate small, smooth changes rather than large jumps in value.

Also of concern regarding the accuracy of the phase information reported is the validity of the cache implementation. The relative tolerances investigated were chosen arbitrarily to demonstrate the various rates of cache utilisation and the actual effect of the cache on the results is unknown. To have more confidence in the results, it would be useful to investigate the variations between the actual and reused results with a varying tolerance to ensure that they converge. Also, the actual logic involved in determining a cache hit is extremely basic and there is a lot of scope for possible improvements. This could involve, for instance, having different tolerances for each of  $p$ ,  $T$  and  $X$  as the phase functions are likely to have greater sensitivities to some of these conditions than others. Another point of consideration is the fact that, as the number of chemical components increases, the likelihood of a cache hit will decrease, making the results in Figure 8 problem-dependent rather than general.

## ***B Particle property plugin performance***

Performance with the particle property plugin is generally excellent. Figure 5a shows that the majority of the runtime is spent in the “Update particle properties” phase. This is good because it implies that most of the computation time is spent solving *Perple\_X* calculations which, given that the *Perple\_X* performance has not been optimised and that the rest of the ASPECT simulation is intended to be quick-to-run, is where the bulk of the computation time is expected to be. The scalability of the code is also very good. The fraction of the code that must be run sequentially (Figure 6a) is tiny, meaning that adding additional processors is highly effective at increasing performance.

Despite this, the load balancing is a concern. Figure 7 shows that the longer a simulation runs, the greater the imbalance in load between processors, potentially reducing the program’s efficiency. A possible solution to this would be to modify the mesh refinement criteria used by ASPECT. To do this one can specify `particle density` as one of the options to the mesh refinement `Strategy` option inside the input parameter file. This will refine the mesh in regions with a high particle density, increasing the number of degrees of freedom in those regions, and thus ensuring that they are spread more evenly



across processors.

Implicit in our approach to the load balancing is the assumption that the distribution of particles is proportional to the distribution of the computational workload and that therefore an imbalanced number of particles handled by each process will lead to a reduction in performance. This was the reasoning used to justify reporting the number of particles per process instead of the number of cells. Given that the majority of the time spent in the simulation is in solving for the particle properties (Figure 5a), this would seem to be a reasonable assumption.

However, the presence of the cache is likely to have a large impact on this. If there are many particles in one cell then they are more likely to make use of the cache, in fact increasing performance. It would be straightforward to investigate the effect of the cache on the load balancing. At present, the `perplex cache statistics` postprocessor simply sums the number of cache hits and misses across processors, but this could easily be adapted to report statistics on a per-processor basis.

### ***C Material model plugin performance***

In stark contrast to the particle property plugin, the material model plugin is plagued by poor performance and scalability. The runtime breakdown (Figure 5c) shows that much more time is spent in the “Assemble composition system” stage rather than in the next largest “Solve composition reactions” stage. Since `Perple_X` should not be involved in the former stage, this suggests that there is a significant overhead with this plugin, likely due to the added complexity resulting from dealing with large numbers of compositional fields. Furthermore, if one also looks at Figure 6b, one can see that over half the code is estimated to be running serially and this means that it experiences low levels of speedup with increasing parallelism. Put together, these results suggest that the material model plugin is not a serious candidate for further research with ASPECT when modelling more complex compositions or at a higher resolution, either of which would increase the aforementioned overhead.

However, the plugin is the only truly accurate way of modelling two-phase flow and as such should not be discarded. One simple thing that could be done to possibly improve performance would be to average the evaluations over the quadrature points of each cell. This would potentially reduce the number of calculations necessary by a factor of 4 or more, but given the high cache utilisation would likely have a negligible impact. It would also have no impact on the most expensive serial component of the code. Another thing to look at is the number of `Perple_X` evaluations that are being made. Figure 8b shows that the cache utilisation is extremely high. This suggests that: (a) the cache is essential to the performance of the material model; and (b) there are a lot of unnecessary calculations being done. The number of evaluations could be brought down by increasing the size of the time steps used in the operator splitting scheme.

### ***D Next steps***

Thus far we have focused on the issues existing in the code and their solutions. We will now present our suggestions for ways in which the code could be meaningfully extended and improved.

Starting with the `Perple_X` wrapper, one meaningful improvement would be to add robustness to the code. Currently, only very basic checks (e.g. that the temperature is non-negative) are performed when the code is executed, but the code still occasionally fails and, because standard output is disabled, gives no explanation as to why. This is obviously problematic and reduces the likelihood of the software receiving widespread use in the academic community.

Another problem facing the code is the lack of rigorous testing. At present the code only ‘seems’ to work and produces results that at first glance look to be reasonable (e.g. partial melting occurring as the material depressurises). In order to have confidence in the accuracy of the results, we recommend that several benchmark model setups be created verifying that the code reproduces results found in the literature. The decompression event model setup, being a straightforward, discrete event, might serve as a good starting point for such analysis.

In order to resolve the performance issues present in the material model plugin, we recommend the creation of an entirely new one. Rather than tracking all of the chemical components as distinct compositional fields this plugin would use just a single dynamic compositional field, the porosity, in order to track the flow of the melt. The value of the porosity would be calculated by the `Perple_X` wrapper

and the composition, rather than being advected with the melt, should be either read from the `Perple_X` data file, or declared as static compositional fields. The latter would be advantageous as it would permit a changing composition through the domain. This suggested method will unfortunately no longer perfectly model the changing melt composition, but it is still clearly improved from any parametrised melt function and the performance should be sufficient for more detailed investigations.

Finally, we encourage that the code be shared with the developers of ASPECT who may find it useful in their work. In particular it would be helpful to contact R. Myhill and J. Dannberg because they are involved with the existing `Perple_X` integration and two-phase flow implementation in ASPECT respectively.

## V CONCLUSIONS

In this work, we have described our interface code permitting `Perple_X` to be integrated into other C++ programs, and we have demonstrated the code through implementing both particle property and material model plugins within ASPECT. Some example scenarios have been investigated and the code’s performance has been measured to determine its viability for increasing levels of computational complexity and parallelism.

We have identified that the particle property plugin scales excellently with increasing parallelism and runs comparatively quickly with simple but powerful options for controlling performance. In contrast, the material model plugin, whilst permitting a study of two-phase flow, suffers from dramatic amounts of overhead and scales poorly with increasing parallelism.

Finally, several suggestions have been made to improve the code both in performance and accuracy and some useful extension work has been explained.

## VI ACKNOWLEDGEMENTS

CW would like to thank Jeroen van Hunen and Marion Weinzierl for their support with this project. The computations in this paper made use of the facilities of the Hamilton HPC Service of Durham University.

## References

- Amdahl, G. M. (1967). “Validity of the single processor approach to achieving large scale computing capabilities”. In: AFIPS spring joint computer conference. IBM Sunnyvale, California. DOI: 10.1145/1465482.1465560.
- Bangerth, W., R. Hartmann, and G. Kanschat (2007). “deal.II—A general-purpose object-oriented finite element library”. In: *ACM Transactions on Mathematical Software* 33.4. DOI: 10.1145/1268776.1268779.
- Bouilhol, P., V. Magni, J. van Hunen, and L. Kaislaniemi (2015). “A numerical approach to melting in warm subduction zones”. In: *Earth and Planetary Science Letters* 411, pp. 37–44. DOI: 10.1016/j.epsl.2014.11.043.
- Connolly, J. A. D. (2005). “Computation of phase equilibria by linear programming: A tool for geodynamic modeling and its application to subduction zone decarbonation”. In: *Earth and Planetary Science Letters* 236.1, pp. 524–541. DOI: 10.1016/j.epsl.2005.04.033.
- (2009). “The geodynamic equation of state: What and how”. In: *Geochemistry, Geophysics, Geosystems* 10.10. DOI: 10.1029/2009GC002540.
- (2017). “A primer in gibbs energy minimization for geophysicists”. In: *Petrology* 25.5, pp. 526–534. DOI: 10.1134/S0869591117050034.
- Cottaar, S., T. Heister, I. Rose, and C. Unterborn (2014). “BurnMan: A lower mantle mineral physics toolkit”. In: *Geochemistry, Geophysics, Geosystems* 15.4, pp. 1164–1179. DOI: 10.1002/2013GC005122.
- Dannberg, J. and T. Heister (2016). “Compressible magma/mantle dynamics: 3-D, adaptive simulations in ASPECT”. In: *Geophysical Journal International* 207.3, pp. 1343–1366. DOI: 10.1093/gji/ggw329.

- Freeburn, R., P. Bouilhol, B. Maunder, V. Magni, and J. van Hunen (2017). “Numerical models of the magmatic processes induced by slab breakoff”. In: *Earth and Planetary Science Letters* 478, pp. 203–213. DOI: 10.1016/j.epsl.2017.09.008.
- Ghiorso, M. S. and R. O. Sack (1995). “Chemical mass transfer in magmatic processes IV. A revised and internally consistent thermodynamic model for the interpolation and extrapolation of liquid-solid equilibria in magmatic systems at elevated temperatures and pressures”. In: *Contributions to Mineralogy and Petrology* 119.2, pp. 197–212.
- Holland, T. J. B., E. C. R. Green, and R. Powell (2018). “Melting of Peridotites through to Granites: A Simple Thermodynamic Model in the System KNCFMASHTOCr”. In: *Journal of Petrology* 59.5, pp. 881–900. DOI: 10.1093/petrology/egy048.
- Holland, T. J. B. and R. Powell (1998). “An internally consistent thermodynamic data set for phases of petrological interest”. In: *Journal of Metamorphic Geology* 16.3, pp. 309–343. DOI: 10.1111/j.1525-1314.1998.00140.x.
- Kaislaniemi, L. (2015). *PrplxWrap*. GitHub. URL: <https://github.com/larskaislaniemi/PrplxWrap>.
- Kaislaniemi, L., J. Van Hunen, and P. Bouilhol (2018). “Lithosphere Destabilization by Melt Weakening and Crust-Mantle Interactions: Implications for Generation of Granite-Migmatite Belts”. In: *Tectonics* 37.9, pp. 3102–3116. DOI: 10.1029/2018TC005014.
- Katz, R. F., M. Spiegelman, and C. H. Langmuir (2003). “A new parameterization of hydrous mantle melting”. In: *Geochemistry, Geophysics, Geosystems* 4.9. DOI: 10.1029/2002GC000433.
- King, S. D., A. Raefsky, and B. H. Hager (1990). “ConMan: vectorizing a finite element code for incompressible two-dimensional convection in the Earth’s mantle”. In: *Physics of the Earth and Planetary Interiors* 59, pp. 195–207. DOI: 10.1016/0031-9201(90)90225-M.
- Kronbichler, M., T. Heister, and W. Bangerth (2012). “High accuracy mantle convection simulation through modern numerical methods: High accuracy mantle convection simulation”. In: *Geophysical Journal International* 191.1, pp. 12–29. DOI: 10.1111/j.1365-246X.2012.05609.x.
- Magni, V., P. Bouilhol, and J. van Hunen (2014). “Deep water recycling through time”. In: *Geochemistry, Geophysics, Geosystems* 15.11, pp. 4203–4216. DOI: 10.1002/2014GC005525.
- Mayne, M. J., J. F. Moyen, G. Stevens, and L. Kaislaniemi (2016). “Rcrust: a tool for calculating path-dependent open system processes and application to melt loss”. In: *Journal of Metamorphic Geology* 34.7, pp. 663–682. DOI: 10.1111/jmg.12199.
- Moresi, L., S. Zhong, and M. Gurnis (1996). “The accuracy of finite element solutions of Stokes’ flow with strongly varying viscosity”. In: *Physics of the Earth and Planetary Interiors* 97, pp. 83–94.
- Myhill, R. (2018). *PERPLEX INTEGRATION WITH ASPECT*. URL: <https://github.com/geodynamics/aspect/tree/master/contrib/perplex>.
- Powell, R., T. J. B. Holland, and B. Worley (1998). “Calculating phase diagrams involving solid solutions via non-linear equations, with examples using THERMOCALC”. In: *Journal of Metamorphic Geology* 16.4, pp. 577–588. DOI: 10.1111/j.1525-1314.1998.00157.x.
- Stixrude, L. and C. Lithgow-Bertelloni (2011). “Thermodynamics of mantle minerals - II. Phase equilibria”. In: *Geophysical Journal International* 184.3, pp. 1180–1213. DOI: 10.1111/j.1365-246X.2010.04890.x.
- Takahashi, E. (1986). “Melting of a dry peridotite KLB-1 up to 14 GPa: Implications on the Origin of peridotitic upper mantle”. In: *Journal of Geophysical Research* 91 (B9), pp. 9367–9382. DOI: 10.1029/JB091iB09p09367.