NEPTUNE DSL workshop 8 April 2021

1 Agenda

- Chair / intro (Wayne Arter, UKAEA)
- `OP2` / `OPS` and NEPTUNE work (Gihan Mudalige, Warwick)
- What makes a good DSL? (Will Saunders, UKAEA)
- TBA (Exeter / Imperial)
- 10-minute break, followed by general discussion

(The above was subsequently modified by removal of the break and inclusion of spur-of-the-moment presentations by most of the external attendees.)

2 Minutes

The meeting was opened by Wayne Arter (chair), who showed the agenda, followed by announcements:

- There is new material on the NEPTUNE documents GitHub repository; a new branch `y2end` contains the presentations / minutes of the March progress meeting. There is still scope for contract holders to check this as the branch has yet to be merged.
- The next RAG progress meeting is scheduled for 10-11am of Thursday 29 April (see email of 29/3 - which hopefully everybody received). Volunteers are requested for presentations 11am-12pm of the same day. A Doodle poll for a regular fortnightly slot is to follow.
- This meeting is being recorded for minute-taking purposes.

WA continued with a slide indicating the general proposed software strategy: spectral / hp methods with UQ, that is made attractive to three broad classes of user:

1. Engineer or physicist using the code as a 'black box', e.g. a plasma physicist.
2. High-level programmer using e.g. Python or Julia.
3. Writer of new problem-specific code in e.g. C++.

The code needs to be modifiable and extensible to stand the test of 30 years' future use. WA cited matlab and NAG as models and said these might have even more grades of potential user.

WA then displayed slides of a set of equations governing the scrape-off layer (SOL) dynamics, after F. Riva (single-fluid), then showed the 13-moment model after Zhdanov (fluid model extended to allow for low collisionality); he gave a brief explanation of these e.g. replacing pressure stress tensor in a fluid by a higher-order dynamical completion. This was to illustrate the complexity of the equation systems describing the SOL plasma.

2.1 Bridging the complexity gap in exascale simulation software development through high-level DSLs, Gihan Mudalige (Warwick)

This was an overview of many years' work by GM. A motivation for the work was given: the need for parallelizable code capable of working on a diverse hardware landscape: a large 'zoo' of accelerator types and many different programming methods (OpenMP, SIMD, CUDA, ROCm / HIP, MPI, PGAS ...). Open standards attempting to keep up with the range of hardware and complicated by a lack of overall consensus and companies' vested interests in optimizing code for their own hardware; also the issue of large legacy codes (Fortran was mentioned).

Raising the level of abstraction to solve the problem of a varied hardware landscape: GM explained that a classical compiler has two basic themes: analysis (syntax, semantics, ... , polyhedra) and synthesis (parallelization, tiling, vectorization) (WA asked for clarification of 'polyhedra' in this context: it means things like cache-blocking and tiling). The problem is really that for a general program the compiler cannot easily optimize in terms of layouts and memory spaces (e.g. if running a computation on an unstructured mesh). The proposed solution is to raise the level of abstraction from that of a general programming language to e.g. a communication skeleton (`OP2 / OPS`) and further to a specific numerical method e.g. `Firedrake`. With a narrower problem ambit (e.g. just FEM), there is scope to re-use a known set of optimizations (or sets, given that there are multiple target platforms). This would work as a domain-specific API (the 'contract' with the user) being embedded in e.g. Python or C++. A quote from Mike Giles was cited: `OP2 / OPS` 'straitjacket' the user and prevent them from writing bad code. So one has a scheme in which a problem is declared by the user and a set of automated routines produce an optimized implementation. Examples of handling unstructured mesh code (which looked to be finite-difference based) were shown; the problem here was sorting out data races if multiple edges tried to update the same node.

A nice application structure diagram was shown and it was made clear that e.g. `OP2` acts as a parser: it is a source-to-source translator and it outputs human-readable code, which then goes into a general compiler and can be used with general debugging / profiling tools (g++, gdb, Allinea MAP). The DSL layer here handles automatic parallelization, load-balancing, checkpointing, and runtime (JIT) compilation.

The talk necessarily accelerated here due to time constraints; some `SYCL` examples were shown (a useful citation is a 2021 publication by GM, Jarvis, Powell, Owenson, Reguly `https://warwick.ac.uk/fac/sci/dcs/people/gihan_mudalige/op2-mgcfd.pdf`).

Mention was given to the ASIMOV project: re-engineering existing codes with `OP2`; e.g. one which is 50k lines of Fortran with over 300 parallel loops.

It was emphasized that getting the correct abstraction (i.e. higher-level description) is the main thing and that this has more mileage in it than the particular implementation using the technology of the moment (this paraphrases a quote from Alfred Aho and Jeffrey Ullman).

WA asked how to minimize the workload of a future re-engineering exercise such as in ASIMOV; GM replied that NEPTUNE is in a good position as it starts code from scratch - just get the abstractions correct.

Patrick Farrell asked whether the DSL acts at compile or runtime. GM replied that historically it has been done at compile time due to e.g. not having the compiler available on the HPC compute nodes, but some stuff can be done at runtime e.g. `Firedrake` does loop nesting and tiling.

2.2 What makes a good DSL, Will Saunders (UKAEA)

WS advised he would provide a more high-level discussion, starting with some (ubiquitous!) examples of DSLs e.g. LaTeX, SQL, HTML, APIs. Desirable properties are:

- Ease of use.
- Communication: allows accurate problem description and allows enforcing of explicit standards from third parties e.g. a publisher's LaTeXconventions.
- Abstractions to give separation of concerns (e.g. LaTeXusers are agnostic as to *how* exactly their PDF is generated).

Two types of DSLs were considered: 1) External, implemented by a specific interpreter or compiler (e.g. LaTeX, SQL, `Make`), which are flexible but involve the hard work of writing a compiler; 2) Embedded, which extend an existing host language (i.e. an API) (e.g. `SYCL`, `UFL` - Unified Form Language). The latter allows the use of the host ecosystem, but restricts the DSL to use the lexicon of the host language (e.g. Python does not support overloading the assignment operator).

WS presented three characteristics of a 'good' DSL:

- Provides the correct abstraction to describe domain tasks (concurring with GM's preceding talk).
- Ease of use i.e. intuitive for domain user.
- Composable, as DSLs are rarely used in isolation.

WS finished by discussing the question of what we want from a DSL:

- This is really an open question for all levels of our (prospective) user community.
- Separation of concerns - a hierarchy.
- Performance portability over likely HPC targets.
- Offering interoperability between components i.e. acting as a gluing language.

2.3 Discussion of `Nektar++` from a DSL standpoint (my title), Spencer Sherwin

SS explained that `Nektar++` did not initially use DSLs. He gave then an overview of the structure of the code in the latest version (refined in the light of knowledge gained writing earlier spectral / hp codes). Currently the developers are pushing a top-level Python interface (also, I'd add that the xml session file is a DSL of sorts).

In a discussion between SS and WA it became clear that the fluids community had relatively little need for a complex DSL because fluid equations are fairly standardized things (in contrast to the wide range of models used in fusion). WA emphasized that the new terms added to fusion models (e.g. sources) had the potential to cause a wide range of numerical issues, and so it was clear that these might need a relatively deep integration with the source code.

2.4 `UFL` / `Firedrake` (my title), Patrick Farrell

The question of what happens if the DSL is too restrictive to add a certain piece of new physics was raised by SS; PF explained that `Firedrake` circumvents this issue by allowing pieces of code from other languages to be included (via 'escape hatches'), so there can be C++ or Python 'bolt-ons'. PF showed then some slides taken from a FEM theory course he teaches showing the use of `UFL` in `Firedrake` (one very nice feature is a simple API to generate regular meshes - not present in `Nektar++`). This showed how easy it is to specify weak-form PDEs e.g. `G = inner(grad(u), grad(v))*dx - inner(f,v)*dx` then `Solve(G==0,u,bc)`. He then showed a more complex linear elasticity example and explained that these toy problems could nevertheless be scaled to billions of degrees of freedom of ARCHER. WA asked to see an example of some bolt-on code (PF prepared some - see 2.6 below).

2.5 DSLs in `BOUT++` (my title), Ben Dudson

BD's experience with coming to existing code and finding discrepancies between the code and the documentation, as well as fusion physics' lack of completely-specified models, led to the aim to make it easy to add new physics to `BOUT++`. He explained the code structure i.e. method-of-lines time integration with all the physics in a module that computes the time-derivatives, and gave an example of the `BOUT++` DSL: `ddt(n) = -vE_Grad(n,phi)+Div_par(Jpar)+2*DDZ(n) / R_c`. WA asked whether the next bit of code shown was C++ but BD replied it is an external DSL that is interpreted at runtime; apparently the DSL evolved from a simple configuration file and now it can e.g. parse complex mathematical formulae but it was unfortunately *not* Turing-complete, as a result of this evolution.

BD explained how the code had been made performant and cited work by Joseph Parker (2018) vectorizing the kernel inner loops (subsequently, the bottleneck became the elliptic inversion). Another problem overcome was that of too many small loops not parallelizing efficiently on GPUs (need more work per unit of loaded data to get the efficiency). Showed example of merging outer loops. This became hard to debug, and adding debugging framework code wrecked the performance, so an idea was borrowed from `SYCL` - unsafe but lightweight wrappers and doing the runtime checks outside the loop. There was a brief discussion about the need for manual vectorization, as compilers cannot do this well, and interaction with threading only complicates things.

2.6 General discussion

PF showed `Firedrake` with bolt-on code snippets e.g. example of converting non-periodic mesh to periodic in loopy syntax (https://documen.tician.de/loopy/) and the other as a normal C function to exploit tensor product structure. The kernels (either plain C or loopy) are inputs along with sets, maps and access descriptors to PyOP2. PyOP2 is a large Python code integral to `Firedrake` which generates the C source code for a shared library which is written to disk and then compiled with a C compiler of choice (usually GCC).

WA raised a couple of points:

- From BD talk: it was clear that not all plasma physicists will know about FEM.
- `BOUT++` uses method of lines, and also elliptic solvers: what if we needed to solve coupled elliptic problems? BD explained that such things could be transformed into something that can be solved.

SS raised issue of whether a DSL could encourage good practice - clearly with a DSL the added flexibility means more scope to get nonsense results out (though, presumably, VVUQ techniques would flag up bad calculations). WA agreed that is a big question ... SS mentioned that in finite difference, the only adjustable parameter is the global refinement (whereas, in finite element, one can locally refine the mesh, or do p-refinement; WA mentioned it had to be done dynamically as shocks can form during simulations).

SS asked whether the user could be warned if they had introduced a term that was likely to cause the simulation to fail; WA responded that the equations for fusion tended not the be that bad in this regard (second derivatives, collisions and transport) and that the difficulty really came from the sheer number of different terms and possible species. PF asked what a warning (or straitjacketing) system would mean in practice given that it might reduce overall freedom (e.g. clearly LaTeXallows the user to write mathematically incorrect equations). SS replied that his intent was to help the user understand how simulation results are affected by the various solver options. PF put it that it should be made easy for the user to do things they 'should' be doing (i.e. use the interface to 'nudge' users in the right direction). BD agreed, saying that this was the thinking behind `BOUT++`. PF and SS agreed that error checking by means of the method of manufactured solutions was useful. PF then said his concern with `UFL` was that it is designed for FEM, not plasmas: he plans to talk to BD about what equations are needed for NEPTUNE and also the wider cross-cutting ExCALIBUR themes. Specifically, BD asked whether `UFL` can handle 5D and 6D phase spaces (i.e. including velocity space), to which PF replied that `UFL` can handle this but the solvers (`Firedrake`) currently cannot.

WA said it seemed many people were happy with `UFL` but many in the plasma community are unfamiliar with FEM; he thought some in the community will have an equation they wish to solve, so good if they can use the DSL to implement it on their own (separation of concerns between the equation and the FEM used to solve it). WA asked whether the NEPTUNE community should put effort behind `UFL`, given that most `UFL` users are not in the fusion field. BD said `UFL` was promising. PF added that `UFL` was becoming the language of choice in the FEM community and is good for comparing FEM runtimes.

BD asked about methods for transforming higher-level mathematics into the weak form used in `UFL`. PF cautioned that there are many possible variational forms for the same mathematical equations, different Sobolev spaces etc., so automating this is probably impossible. WA questioned whether something like this existed - biased to providing a 'robust' solver in all cases - PF unsure, but added that a least-square coercive approach would be robust but conditioning and performance would always be suboptimal (WA agreed and there was an astrophysical application by Wiegelmann which might be admitted to be at least 100 times slower). PF opined that the approach should be to try to 'automate out' the computer programmer and not the mathematician or the physicist.

WA raised the issue that the fusion equations may become very large and prone to errors / typos in implementation; PF agreed that it might be much quicker to implement the equations in *Firedrake* that for the mathematician to try to debug the equation system by inspection.

WA said one goal was to educate the user community about aspects of FEM, therefore expose some of the options (e.g. element order, basis type). PF agreed and said we should try to give enough education that users can avoid common pitfalls. SS mentioned some of the options in FEM and added that people like the strong form and the nodal basis (as 'easier to think about'). PF mentioned that not all possible discretizations are stable, citing compatibility conditions. WA mentioned that workers on the European Boundary Code project are getting good results with Discontinuous Galerkin, as this is typically more stable than classical Galerkin - SS added that there is a large literature on DG stability, Riemann fluxes etc.

WA wrapped up the session, stating that this meeting was more about discussion than reaching firm conclusions (and added that we have yet to define a DSL). GM made the point that there is a higher-level maths / physics interface as well as a lower-level hardware abstraction layer dealing with loops as its input (GM's work concerns the latter of these two). PF mentioned that `Firedrake` takes a `UFL` input and produces computer code output so acts to separate concerns. WA added that there is a great deal of scope for additional physics / more species in our equations - is there a need for more software in consequence? WA asked whether new functionality can be added to `OP2` if we need (e.g. PIC codes); GM said he had yet to try PIC codes and that these need their own abstractions; it was mentioned that Steven Wright has worked on PIC via Kokkos, so a discussion to be had there.

WA concluded by saying that discussion between grantees is encouraged although UKAEA would like to be kept in the loop; also to let him know if the grantees thought that UKAEA's organising another session on DSLs would be of benefit.

3 Attempt at Conclusion (written after the meeting by the minute-taker)

It became clear that there are really *two* separate issues here: the DSL as a user interface and the DSL as a hardware abstraction API.

The DSL as a user interface inherits all the usual problems of interface design e.g. learning curve, flexibility versus complexity ... The usual holy grail of the code being able to determine internally the optimal parameters for performing a given calculation (e.g. mesh resolution, level of p-refinement and many, many more) seems something to strive toward, rather than representing an attainable goal. The use of 'escape hatches' as demonstrated by PF seems a very good idea. `UFL` seems a reasonable early candidate on which to base something, though it does not currently do everything we might want (e.g. strong form) and it relies on at least some user knowledge of FEM.

The hardware abstraction part seems to be a matter of working out the loop structure and implementing this using e.g. `OP2` or considering direct implementations in e.g. `SYCL`. The hope is that these standards will render time-consuming performance tuning, e.g. manual vectorization, unnecessary.