Extending just-in-time compilation for OP2

Nathan Dunne - u1604486

October 6, 2019

Problem Statement

OP2 is an Embedded Domain Specific Language for solving unstructured mesh based applications. It exists to provide a level of abstraction to scientists and domain application developers, allowing utilisation of the high performance benefits of advancements in hardware, without requiring them to maintain an high level of knowledge on any new technologies and architectures. As the high-level application written using the OP2 API no longer requires any platform specific optimisations, it can be used on any platform supported by OP2; saving the time and financial costs of rewritting the application with different hardware optimisations.

One way to improve the performance of an application such as the unstructured meshes targetted by OP2 is using Just-In-Time (JIT) compilation, whereby performance sensitive sections of the application are re-compiled at runtime to apply additional optimisations once the parameters of execution are known. The performance gain must be sufficient to offset the extra time taken to recompile in order for this technique to be effective. There is exisiting work implemented already for some JIT optimisations, however there is still space for further performance gain. The aim of this project is to implement additional optimisations in this area, such as replacing library operatuibs with constants, hard coding loop bounds, and applying loop fission/fusion.

There are a number of High Performance Computing applications already written in the OP2 api, from smaller example projects to production applications. Since the API itself will not be altered, it will be easy to apply my additions to the OP2 translator, and determine whether there is permformance benefit using benchmarks.

Project Goals

- To gain an understanding of the existing work in the OP2 library, and how it benefits unstructured mesh applications.

- Apply my knowledge of optimisation techniques to implement and test further JIT optimisations.
  - There are a number of optimisations possible, so the time to implement each one will determine how many can be included in the project
  - The primary optimisation will be replacing time consuming operations with constants where possible. Ideally the project will also extend to the optimisation of loops.

- To benchmark the performance of existing OP2 applications with and without the additions, and determine whether it provides benefit.

Methods

Research

In order to contribute to the OP2 project, I will first read a number of papers on the op-dsl homepage to ensure I understand the existing work, and am able to provide useful addition to the project. The papers suggested by my supervisor where:

Version Control

The OP2 project is Open Source, and hosted on github. I will work on a branch of the repository, and hopefully the completed work can be merged once it is complete.

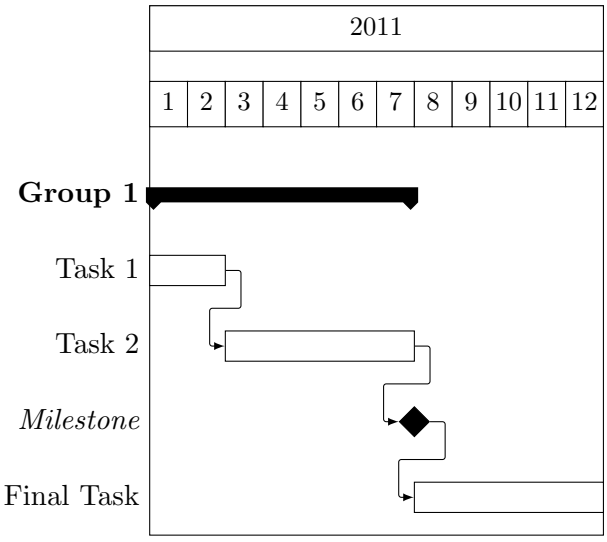
Development

As any progress made will be pushed to the remote github branch, I am not limited to a single workstation or system to work on. Depending on convenience I will either be using my personal laptop.
This laptop does have a cuda enabled graphics card, and I also have access to a remote machine with a hpc grade nvidia card should there be any issues with using my device.

Benchmarking

Initally and throughout development I will be using the airfoil[1] example program written using the OP2 api as a testbed for my work. This should be small enough as to be easy to understand, but not so simple that it does not adequately exercise my additions to the OP2 librarary.
Once progress has been made, there are more complex OP2 applications available to assess what performance benefits can be provided.

Timetable



References

[1] G.R. Mudalige, I Reguly, M.B. Giles. Airfoil Example Guide Airfoil Example Guide, 2013

[2] G.R. Mudalige, I. Reguly, M.B. Giles, C. Bertolli and P.H.J. Kelly. OP2: An Active Library Framework for Solving Unstructured Mesh-based Applications on Multi-Core and Many-Core Architectures. In Proceedings of Innovative Parallel Computing (InPar), 2012