Bitwise Reproducible Execution of Unstructured Mesh Applications

Bálint Siklósi
3in Research Group
Faculty of Information Technology and Bionics
Pazmany Peter Catholic University,
Esztergom, Hungary
siklosi.balint@itk.ppke.hu

István Z Reguly
Faculty of Information Technology and Bionics
Pazmany Peter Catholic University,
Budapest, Hungary

Gihan Mudalige
Department of Computer Science
University of Warwick
Coventry, United Kingdom

Abstract

Engineering applications use floating point arithmetic which are not associative according to the IEEE specifications. In a parallel environment, this usually means the application becomes unreproducible due to the non-deterministic ordering of operations. In this paper we present work on generating a method for unstructured mesh applications to provide bitwise reproducibility between separate runs, even if they are started with different number of MPI processes. We implement our work in the OP2 domain-specific library, which provides an API that abstracts the solution of unstructured mesh computations, and demonstrate how the whole process can be automated without intervention from the user. We carry out the performance analysis of our method applied to two applications: a simple finite volume application, and a more complex finite element code that uses a conjugate-gradient solver. We show a $2.37\times$ to $1.49\times$ slowdown on these applications as a price for full bitwise reproducibility.

Introduction

- Problem:
  - IEEE-754 standard for floating point representation
  - Correct behaviour, but comes with roundings. → non-associativity
  - The order of calculations, usually relaxed in a parallel environment, affects the results
- Motivation:
  - In some industries exact reproducibility is very important, due to regulatory requirements
    - aircraft turbine design
    - algorithmic trading
- Other solutions:
  - ReproBLAS project's binned representation [1] → $5n$ to $9n$ floating point operations overhead
  - Lulesh → only for boundary/halo values
- Our solution:
  - Reproducible ordering for indirect increments
  - Reproducible reductions
  - Reproducibility even when running on different numbers of MPI processes

The OP2 domain specific language

The OP2 (Oxford Parallel library for Unstructured mesh solvers) project is developing an open-source framework for the execution of unstructured grid applications on clusters of GPUs or multi-core CPUs.

```
1  /* ----- elemental kernel function in res.h ------*/
2  void res(const double *edge,
3           double *cell0, double *cell1 ){
4    //Computations, such as:
5    cell0 += *edge; *cell1 += *edge;
6  }
7  /* ---------- in the main program file -----------*/
8  // Declaring the mesh with OP2 sets
9  op_set edges = op_decl_set(numedge, "edges");
10 op_set cells = op_decl_set(numcell, "cells");
11 // mppings -connectivity between sets
12 op_map edge2cell = op_decl_map(edges, cells,
13                    2, etoc_mapdata,"edge2cell");
14 // data on sets
15 op_dat p_edge = op_decl_dat(edges,
16                 1,"double",edata,"p_edge");
17 op_dat p_cell = op_decl_dat(cells,
18                 4,"double",cdata,"p_cell");
19 // OP2 parallel loop declaration
20 op_par_loop(res,"res", edges,
21   op_arg_dat(p_edge,-1,OP_ID    ,4,"double",OP_READ),
22   op_arg_dat(p_cell, 0,edge2cell,4,"double",OP_INC ),
23   op_arg_dat(p_cell, 1,edge2cell,4,"double",OP_INC));
```

Figure 1: Specification of an OP2 parallel loop

Reproducible indirect increments

On Figure 2 two example incrementing orders can be seen on a single cell, by executing through the `edges` set by using an `edge_to_cells` mapping. Since the associative laws of algebra do not necessarily hold for floating-point numbers, $cell0 = e0 + e1 + e2 + e3 \neq cell0 = e1 + e3 + e0 + e2$. This situation might happen over MPI, when OP2's partition algorithm produces different local IDs for every elements. Thus we had to implement a fixed execution order into OP2, to solve this issue. Our method consists of two main parts:

- OP2 needs to determine a consistent fixed ordering
- The code automatically generated for the application needs to use this order

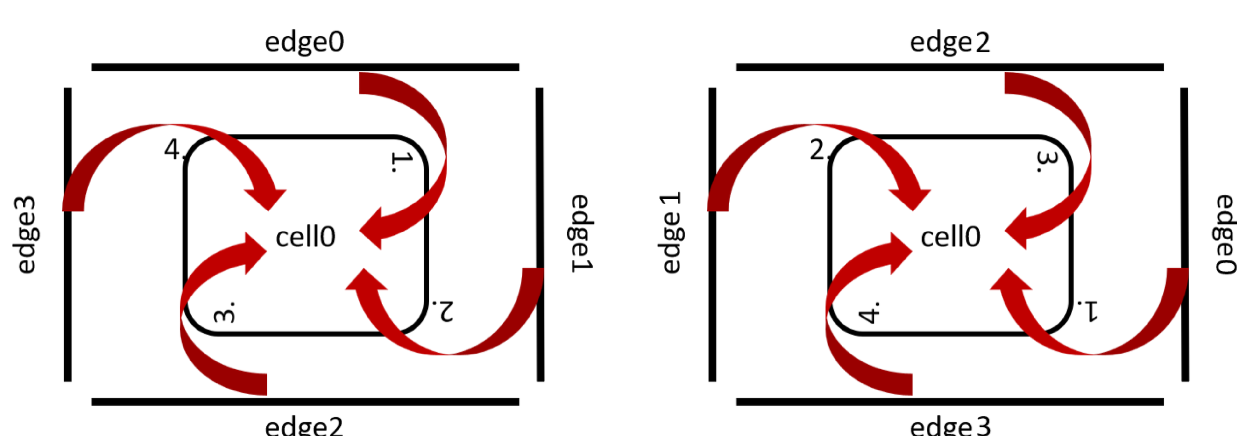

Figure 2: Example orders of incrementing a cell in airfoil.

Or solution can be seen at [2]. Our main idea is to swap the execution of the elements. From iterating through edges and increment cells, we iterate through cells and ask for increments from each edges with a fixed global order and apply those on the cell. This idea can be seen on Figure 3.

Obviously in implementation, this produces some overheads in computational time and memory usage. All edges are executed as they were executed originally, but their results are stored in a temporary array. After that an extra loop must be executed to collect all the local increments.

This method works with other type of mappings, not just with `edges_to_cells`.

An other difficulty of combining reproducibility with MPI is reducing into a single variable. To solve this issue, we introduce again a temporary storage for all local increments, and then we use ReploBLAS's `double_binned` structure which can apply all increments reproducibly.

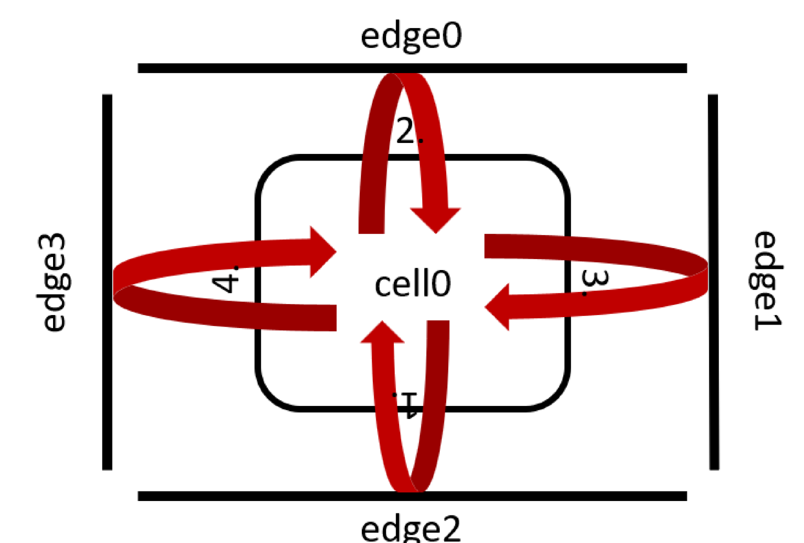

Figure 3: Swapped structure of execution.

Results

Our results are tested on the ARCHER supercomputer on two mini-applications: (1) Airfoil, a standard finite volume CFD benchmark code and (2) Aero, a finite element 2D nonlinear steady potential flow simulation. Both meshes contain 2880000 nodes and 5757200 edges.

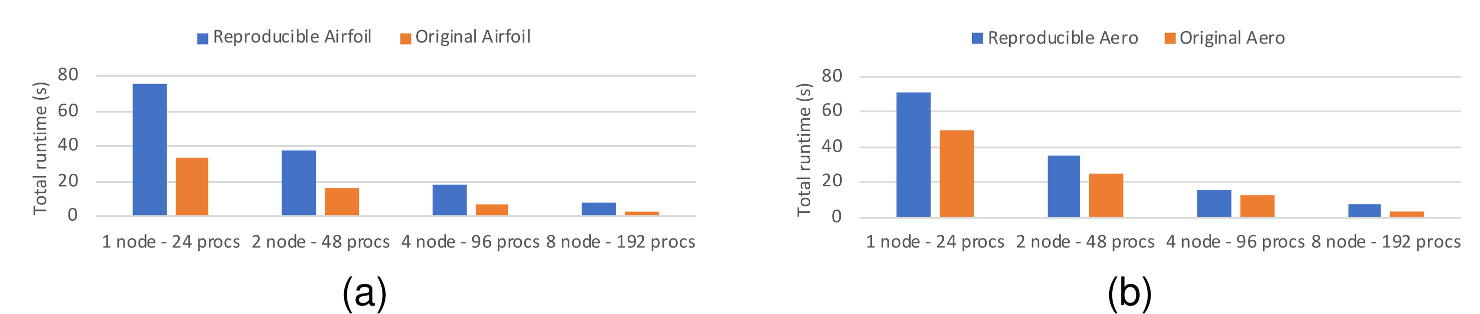

Figure 4: Measured full runtimes of the (a) Airfoil and the (b) Aero applications. On the Airfoil, there is an average of $2.37\times$ slowdown, and a $1.49\times$ on the Aero.

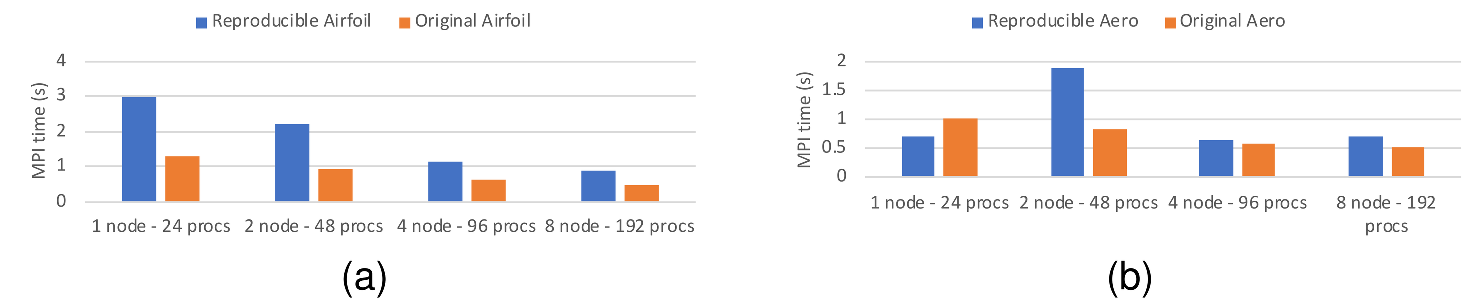

Figure 5: Measured MPI runtimes of the (a) Airfoil and the (b) Aero applications. On the Airfoil, there is an average of $2.14\times$ slowdown, and a $1.37\times$ on the Aero.

As it was expected, reproducibility comes with a significant slowdown effect due to the extra memory movement involved, although if the application is computationally more intensive then the runtime difference decreases. In terms of bandwidth, there is a $1.4\times$ decrease measured on the Airfoil application, due to increased irregularity in memory accesses.

Forthcoming Research

Future work involves extending this method to OpenMP and CUDA. During this work, a new type of problem came up. If a kernel is not just incrementing a variable, but reads and rewrites it, then the kernel call from one edge must be executed, not just the increment temporarily stored. This problem needs a solution to be able to really execute the kernel calls in a predefined fixed order. Solving this issue, we already introduced a simple model: if we can create a coloring for all executed elements which satisfies the property at equation 1 for all types of partition, then executing the elements used by those colours we can achieve full reproducibility.

$$\forall i \in V, \forall j, k \in V | (i,j) \in E, (i,k) \in E : j < k \rightarrow color[(i,j)] < color[(i,k)] \quad (1)$$

References

[1] James Demmel, Peter Ahrens, and Hong Diep Nguyen. Efficient reproducible floating point summation and blas. Technical Report UCB/EECS-2016-121, EECS Department, University of California, Berkeley, Jun 2016.

[2] OP-DSL: The Oxford Parallel Domain Specific Languages, 2015. https://op-dsl.github.io.

Acknowledgements

Project no. PD 124905 has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the PD_17 funding scheme. The research has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013, Thematic Fundamental Research Collaborations Grounding Innovation in Informatics and Infocommunications) and by the Thematic Excellence Program of the Hungarian Ministry for Innovation and Technology.