Bitwise Reproducible Execution of Unstructured Mesh Applications

Bálint Siklósi, István Reguly, Gihan R Mudalige

Pázmány Péter Catholic University
Faculty of Information Technology and Bionics

Oct, 2020

Overview

- Introduction
- OP2-DSL
- Reproducible strategies
- Results

Introduction

- Problem:
  - IEEE-754 standard for floating point representation
  - Correct behaviour, but comes with roundings. $\rightarrow$ non-associativity
  - The order of calculations, usually relaxed in a parallel environment, affects the results
- According to google's built in calculator:

$$
\begin{aligned}
&((((((((((1000000000 + 0.001) + 0.001) + 0.001) + 0.001) + 0.001) \\
&\qquad + 0.001) + 0.001) + 0.001) + 0.001) + 0.001) = 1000000000.0100005 \\
&1000000000 + (0.001 + 0.001 + 0.001 + 0.001 + 0.001 + 0.001 \\
&\qquad + 0.001 + 0.001 + 0.001 + 0.001) = 1000000000.01
\end{aligned}
$$

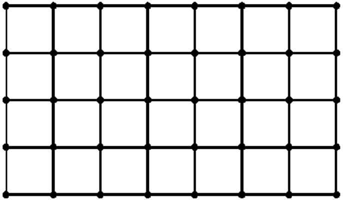
Motivation

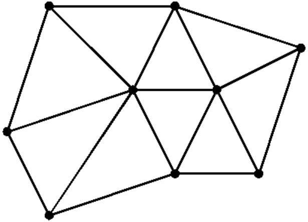
- In some industries exact reproducibility is very important, due to regulatory requirements:
    - aircraft turbine design
    - algorithmic trading, checked by regulators
- Debugging
    - Reproduce errors
    - Compare outputs

Approaches

- Other solutions:
  - ReproBLAS project's binned representation $\rightarrow$ $5n$ to $9n$ floating point operations overhead
  - Lulesh $\rightarrow$ only for boundary/halo values
- Our solution:
  - Reproducible ordering for indirect increments
  - Reproducible reductions
  - Reproducibility even when running on different numbers of MPI processes

PDEs on structured and unstructured grids

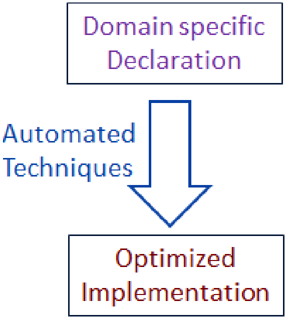




- Structured grids
  - Logical indexing with implicit connectivity
  - Easy to parallelise, including on GPUs
- Unstructured grids
  - A collection of nodes, edges, etc., with explicit connections - e.g. mapping tables define connections from edges to nodes
  - Much harder to parallelise
  - For many interesting cases, unstructured meshes are the only tool capable of delivering correct

One approach to develop future proof HPC applications is the use of domain specific high-level abstractions (HLAs)

- Provide the application developer with a domain specific abstraction
    - To declare the problem to be computed
    - Without specifying its implementation
    - Use domain specific constructs in the declaration
- Create a lower implementation level
    - To apply automated techniques for translating the specification to different implementations
    - Target different hardware and software platforms
    - Exploit domain knowledge for better optimisations on each hardware system

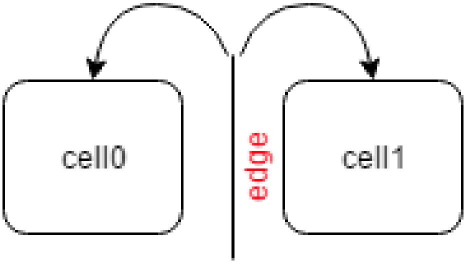


OP2

- Open Source project
- OP2 based on OPlus (**O**xford **P**arallel **L**ibrary for **U**nstructured **S**olvers), developed for CFD codes on distributed memory clusters
- Separate high level description from parallel implementation
- Looks like a conventional library, but uses code transformations (source to source translator) to generate parallel codes
- Support application codes written in C++ or FORTRAN

OP2 loop over edges

```
void res(double* edge,
         double* cell0,
         double* cell1){
   *cell0 += *edge;
   *cell1 += *edge;
}
```




```
op_par_loop(res,"residual_calculation", edges,
  op_arg(dedges, -1, OP_ID, 1, "double", %OP_READ),
  op_arg(dcells, 0, pecell, 1, "double", OP_INC),
  op_arg(dcells, 1, pecell, 1, "double", OP_INC));
```

Generated code for the example loop

```
void res(double* edge,double* cell0,double* cell1) {
    *cell0 += *edge; *cell1 += *edge; }

void op_par_loop_res(char const *name, op_set set,
                      op_arg arg0, op_arg arg1,
                      op_arg arg2) {

  int nargs = 3; op_arg args[3] = {arg0, arg1, arg2};
  int exec_size = op_mpi_halo_exchanges(set, nargs, args);

  for ( int n = 0; n < exec_size; n++ ){
    if (n == set->core_size) op_mpi_wait_all(nargs, args);

    int map0idx = arg0.map_data[n * arg0.map->dim + 0];
    int map1idx = arg0.map_data[n * arg0.map->dim + 1];
```

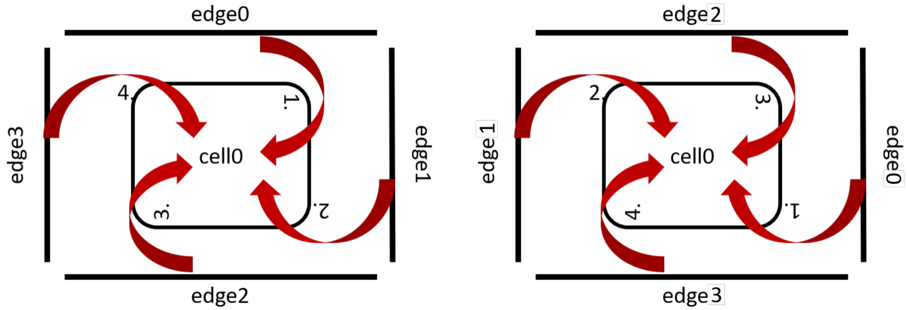
Kernel function

Number of arguments

Static code

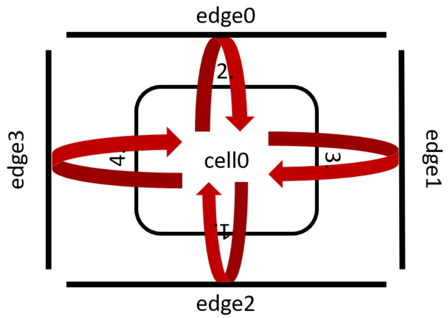
Prepare indirect accesses

Nonreproducible indirect increments



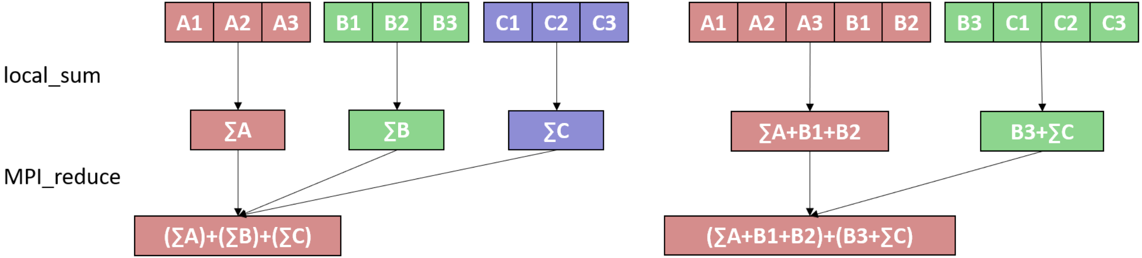
The associative laws of algebra do not necessarily hold for floating-point numbers
$\rightarrow e0 + e1 + e2 + e3 \neq e1 + e3 + e0 + e2$

Reproducible indirect increments - temporary array method

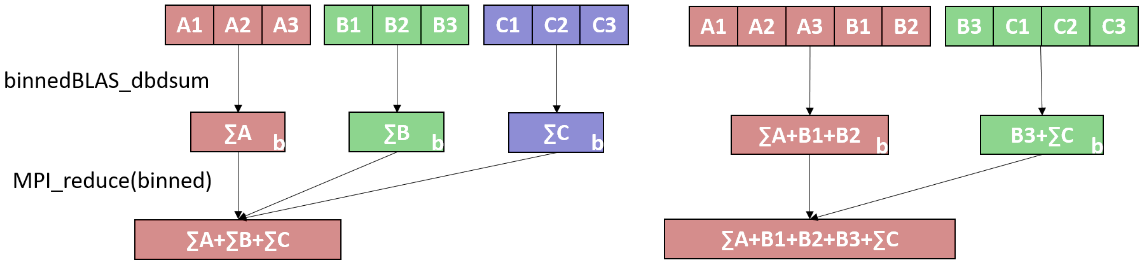


1. Iterate through the edges, calculate increments and store them in temporary array
2. Iterate through the cells and collect the increments using the global indexing of the neighbouring edges

Reproducible reduce over MPI



Reproducible reduce over MPI



Reproducible indirect increments - coloring method

- There are applications, with RW access, not just increment $\rightarrow$ the kernel must be executed, temporary storage is not enough
- We apply a regular coloring scheme to define order
- Problem: how to achieve same color orders each time?
- Solution 1: Color the full mesh on one process, save it, and next time distribute it during loading. $\rightarrow$ DONE
- Solution 2: Develop an algorithm which generates same colors in a distributed graph. $\rightarrow$ TODO
  - Trivial solution exists. Is there a better one?

Experimental setup

- CPU related environment
  - Intel Xeon Gold 6226R CPU@2.90GHz 16 processes per core
- GPU related environment
  - dgx-station with 4 Nvidia V100 GPUs

- Test applications
  - **Airfoil**, a standard finite volume CFD benchmark code
    - Mesh sizes: 45k, 720k, 2.8M
  - **Aero**, a finite element 2D nonlinear steady potential flow simulation.
    - Mesh sizes: 180k, 1.6M, 6M
  - **MG-CFD**, a multigrid, finite-volume CFD mini-app
    - Mesh sizes: 1M, 8M

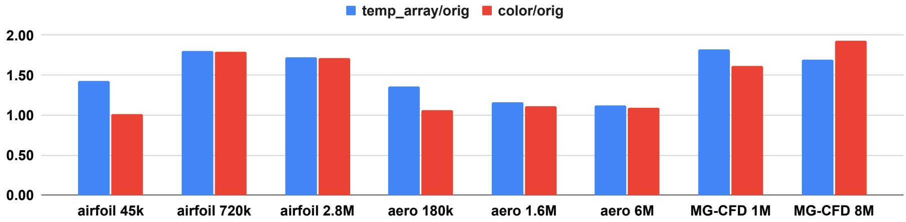


Figure: Measured slowdown effect of the generated sequential reproducible MPI version compared to the original non-reproducible

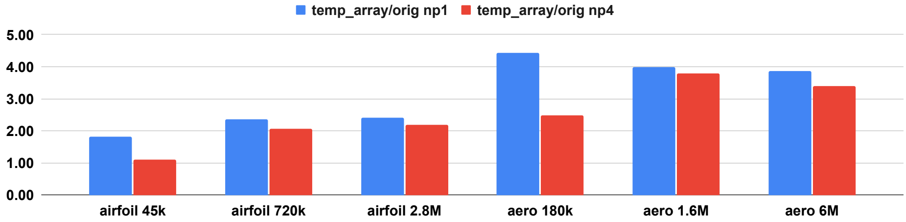


Figure: Measured slowdown effect of the generated cuda reproducible MPI version compared to the original non-reproducible

Contact

- **OP-DSLs**: `https://op-dsl.github.io/`
- **OP2**: `https://github.com/OP-DSL/OP2-Common`
- siklosi.balint@itk.ppke.hu

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.