Journal of
open research software

# `Portage`: A Modular Data Remap Library for Multiphysics Applications on Advanced Architectures

SOFTWARE METAPAPER

]u[ ubiquity press

ANGELA HERRING
CHARLES FERENBAUGH
CHRISTOPHER MALONE
DANIEL SHEVITZ
EVGENY KIKINZON
GARY DILTS
HOBY RAKOTOARIVELO
JAN VELECHOVSKY
KONSTANTIN LIPNIKOV
NAVAMITA RAY
RAO GARIMELLA

**Author affiliations can be found in the back matter of this article*

## ABSTRACT

`Portage` is a scalable and extensible remap library for numerical simulations. It supports state-of-the-art remap schemes for meshes and particles in 2D and 3D up to a second-order accuracy. `Portage` ensures critical properties such as local/global conservation and bounds preservation for mesh remap. It enables multi-material field remap through a dedicated plugin, and leverages the hybrid parallelism exposed by advanced architectures using multi-processing and multi-threading.

CORRESPONDING AUTHOR:
**Angela Herring**
Lagrangian codes, Los Alamos National Laboratory, USA
*angelah@lanl.gov*

KEYWORDS:
remap; conservative interpolation; particle remap; meshing; ALE

TO CITE THIS ARTICLE:
Herring A, Ferenbaugh C, Malone C, Shevitz D, Kikinzon E, Dilts G, Rakotoarivelo H, Velechovsky J, Lipnikov K, Ray N, Garimella R 2021 `Portage`: A Modular Data Remap Library for Multiphysics Applications on Advanced Architectures. *Journal of Open Research Software,* 9: 26. DOI: *https://doi.org/10.5334/jors.338*

## (1) OVERVIEW

### INTRODUCTION

Remap is the transfer of numerical fields from a computational domain to another. It is said to be conservative when some extensive quantity is preserved during this transfer. For instance, one may want to preserve the material mass while remapping the density field defined on a source mesh to a target mesh. Remap is necessary for:

- interpolating fields from a distorted mesh to an improved one in an indirect **Arbitrary Lagrangian-Eulerian** simulation (ALE in short).
- linking internal and external fields in a multi-physics simulation pipeline.
- interpolating data from different numerical codes.

Conservative schemes have long been of particular interest for **ALE** simulations [1]. In such simulations, the mesh is allowed to evolve in time along with the material such as depicted in ***Figure 1***. In that case, the mesh is smoothed to prevent cells distorting or tangling, and all fields computed on the old mesh are remapped to the new one. Remap schemes such as advection-based and intersection-based remaps [2] are often integrated within **ALE** hydrodynamics codes such as `FLAG` [3]. They are also useful for other multi-physics applications such as `Amanzi` [4] (to assimilate scattered input data from observation sources), and for code-to-code linking problems such as in `Ingen` [5]. However, this tight integration has led to a proliferation of remap schemes that cannot be easily shared between simulation codes. To address this issue, standalone conservative remap software has been developed such as the closed source `Overlink` [6] from Lawrence Livermore National Laboratory, the legacy `REMAP3D` code [7] from Los Alamos National Laboratory, or the globally conservative open-source `DTK` code [8] from Oak Ridge National Laboratory.

`Portage` is currently the only actively developed open source library that performs locally conservative remap. It provides a lightweight and extensible interface that can easily be customized and integrated into simulation codes. `Portage` supports general polyhedral mesh fields remap up to a second-order accuracy, while preserving integral quantities of interest and numerical bounds. It supports remap between particle fields as well, and provides means to perform mesh remap using the particle remap engine. `Portage` is designed to scale to thousands of cores on distributed architectures through **MPI** and **OpenMP** (using Nvidia's **Thrust** wrapper).

### IMPLEMENTATION AND ARCHITECTURE

#### Features

`Portage` supports three types of remap:

- **Intersection-based** remap is a conservative scheme that relies on exact intersection of source and target meshes. It first identifies the candidate source cells that may potentially overlap each target cell. It then computes two moments of intersection (volume and centroid) between each target cell and overlapping source cells (***Figure 2***). Finally, it interpolates the target cell value from the candidate source cells values using the moments of intersection as weights [2].
- **Advection-based** remap is a conservative scheme specifically designed for meshes with the same topology but with different node positions. As described earlier, this need arises from **ALE** hydrodynamic simulations when the mesh is slightly smoothed to prevent cell distortion induced by the Lagrangian fluid motion. Here, the remap is formulated as an advection or fluxing of integral quantities in/out of each cell through its faces. Any quantity that is fluxed out of a cell is added into one of its neighbors, so the method is intrinsically conservative. In this algorithm, the interpolation weights are deduced from the flux volumes, which is less expensive – but less accurate – than the previous remap scheme [9].
- **Particle** remap is a specific scheme for point clouds. In this method, source fields are reconstructed by means of local regression [10]. Here a shape function is attached to each source point (**scatter** form) or each target point (**gather** form). The algorithm first identifies the source points included in the support of the shape

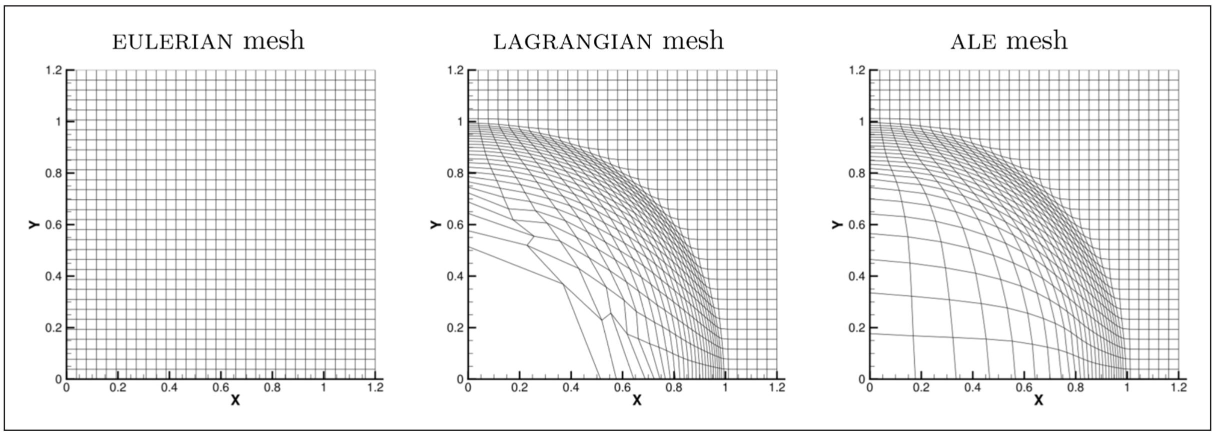


**Figure 1** Mesh deformation in an ALE simulation [1].

function of a target point (***Figure 3***) which are included in the zone delimited by the user-defined **smoothing lengths** which control the number of points used for the local regression. It then computes the weights by evaluating the shape function and its derivatives on each point. Finally, it approximates the value on each target point using those weights. Despite its high accuracy, this remap method is not conservative.

Each step can be processed in parallel with the granularity of a single point or cell.

## Design

`Portage` has a modular design. It relies on extensive `C++` templating of all remap steps, allowing client codes to extend, adapt or replace them by customized ones. Besides, most of its core methods are designed to have no side-effects to ease their parallelization and their individual reuse. `Portage`'s components and their interactions are given in ***Figure 4***.

`Portage` takes the source and target domains along with fields data as inputs, and then outputs remapped fields on the target domain. Here a domain can be

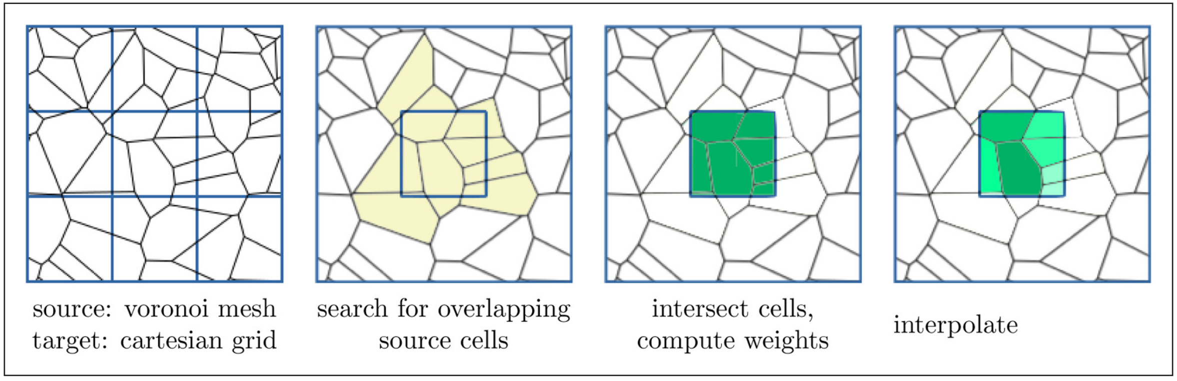


**Figure 2** Illustration of intersection-based remap.

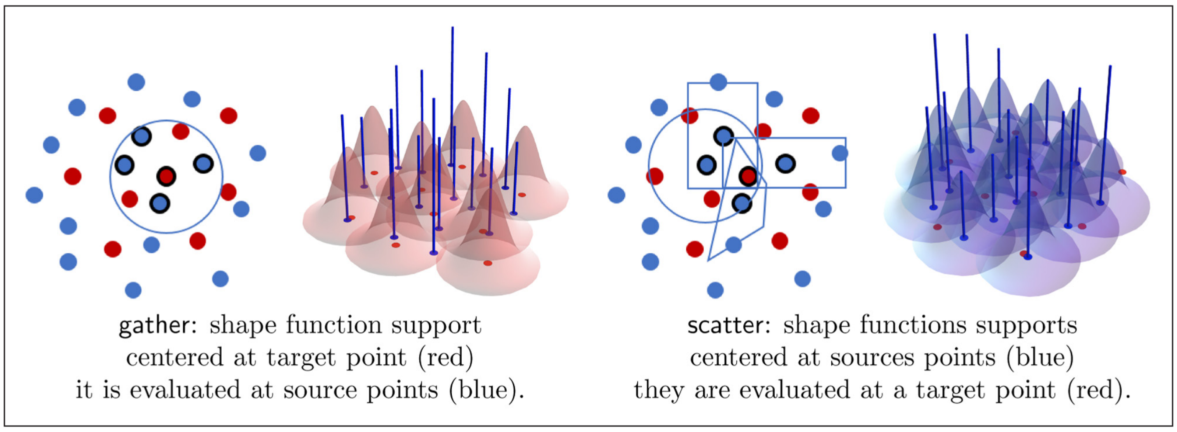

gather: shape function support
centered at target point (red)
it is evaluated at source points (blue).
scatter: shape functions supports
centered at sources points (blue)
they are evaluated at a target point (red).

**Figure 3** Illustration of particle-based remap.

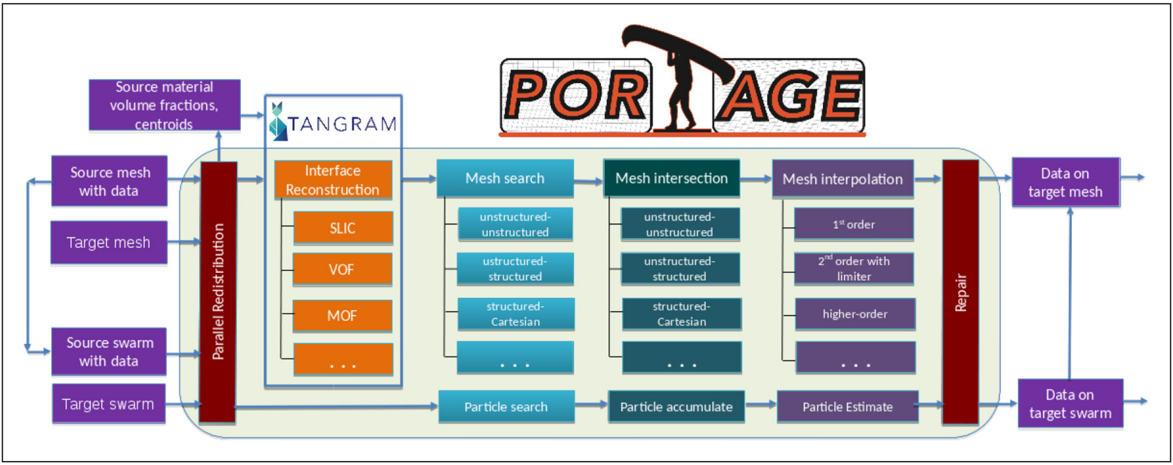


**Figure 4** Portage software design and workflow.

a mesh or a point cloud. For multi-material fields, it requires the material volume fractions on the source domain as depicted in ***Figure 5***, and which corresponds to the proportion of each material on each cell. The remap workflow consists of six stages:

1. **Redistribution:** this optional step is only necessary for distributed domains with a mismatch between the source and target partitions. In that case, some source entities (points or cells) are reassigned among **MPI** ranks such that each target subdomain is overlapped by the corresponding source subdomain. This eliminates the need for communications in the remaining steps.
2. **Interface reconstruction:** this optional step is only required for multi-material fields and is performed by a dedicated plugin called `Tangram`. It recovers the interface between different materials by computing the material polygons on each source cell, given their volume fractions and, optionally, their centroids for a second-order remap accuracy.
3. **Search:** this step identifies and retrieves the source entities that are necessary to interpolate the value of a given target entity. The algorithm depends on the remap scheme:
   - *intersection*: collects the source cells that may overlap the target cell.
   - *advection*: collects the source cell itself and a subset of its neighbors.
   - *particle*: collects the source points included in the support of the shape function of a target point in **scatter** form, and vice-versa for **gather** form.
4. **Computation of weights:** this step computes the contribution weights of each identified source entity to reconstruct the value on a given target entity. Again, the algorithm depends on the remap scheme:
   - *intersection*: computes the moments of intersection (volume and centroids) of each candidate source cell that overlaps the target cell.
   - *advection*: computes the moments of each swept polyhedron (volume and centroids) formed by the displacement of each face of the source cell.
   - *particle*: computes and accumulates the values of the shape functions and their derivatives on each point given by the search step.
5. **Interpolation:** this step reconstructs the target entity values by interpolating them using the computed weights. For mesh remap, the gradient of the source field is required to achieve a second-order accurate reconstruction. It is computed in `Portage` by a least-squares method. Here, values can be limited using **Barth-Jespersen**'s limiter [11], except at domain boundaries because boundary conditions are not yet supported. For particles, we use the term **estimation** as recovered values may pass *near* the data not necessarily *through* it.
6. **Repair:** this step is only necessary in case of mismatch between source and target mesh boundaries. Here, remapped values are fixed to enforce the conservation of integral quantities. `Portage` exposes three options to fix partially overlapped cells:
   - *constant-preserving*: no field value perturbations but not conservative.
   - *locally-conservative*: conservative but perturbations may occur: constant fields may not remain constant.
   - *shifted-conservative*: conservative with minimal perturbations but values are shifted: constant field remains constant but with a different value.

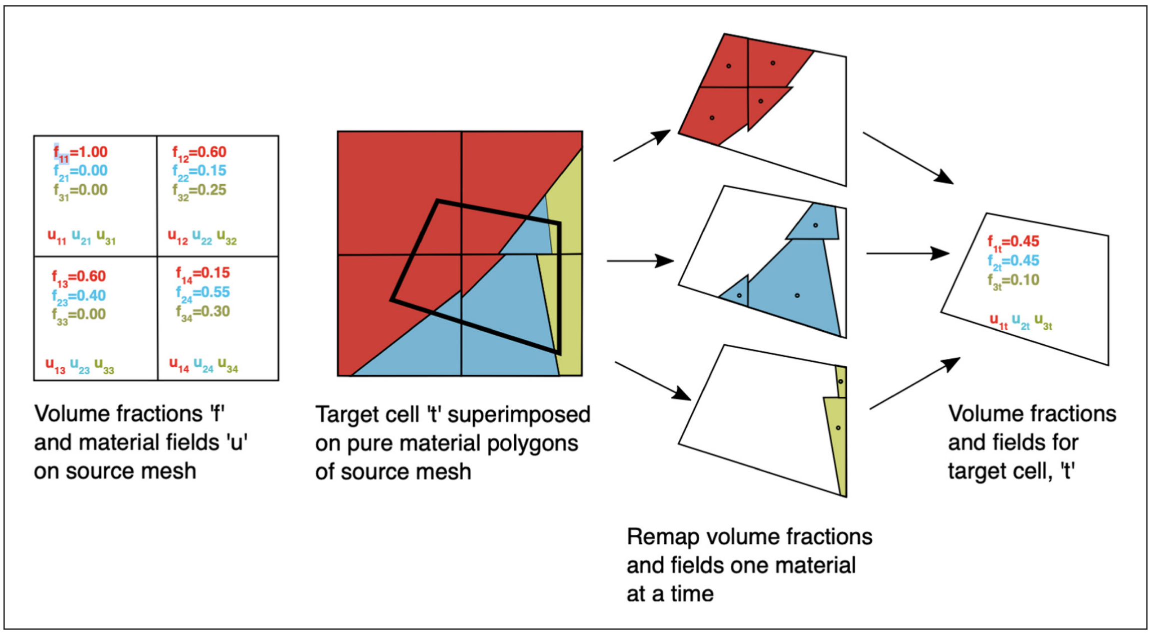

$f_{11}$=1.00
$f_{21}$=0.00
$f_{31}$=0.00
$u_{11}$ $u_{21}$ $u_{31}$
$f_{12}$=0.60
$f_{22}$=0.15
$f_{32}$=0.25
$u_{12}$ $u_{22}$ $u_{32}$
$f_{13}$=0.60
$f_{23}$=0.40
$f_{33}$=0.00
$u_{13}$ $u_{23}$ $u_{33}$
$f_{14}$=0.15
$f_{24}$=0.55
$f_{34}$=0.30
$u_{14}$ $u_{24}$ $u_{34}$
Volume fractions 'f' and material fields 'u' on source mesh
Target cell 't' superimposed on pure material polygons of source mesh
Remap volume fractions and fields one material at a time
$f_{1t}$=0.45
$f_{2t}$=0.45
$f_{3t}$=0.10
$u_{1t}$ $u_{2t}$ $u_{3t}$
Volume fractions and fields for target cell, 't'

**Figure 5** Additional step involved in multi-material remap.

It is also possible to extrapolate values to empty cells in the target mesh.

### Driver

A driver is the interface that exposes the remap capabilities to the client simulation code. Writing a driver allows client codes to mix, match, or extend specialized remap components for their particular needs. `Portage` comes with a few drivers to ease the design of custom ones and several apps to show common remap use cases. Each driver is templated on core components (*interface reconstruction, search, weight computation, interpolation*) for each remap method (*intersection, advection, particle*) and on mesh type. If the simulation code provides a mesh with a set of queries that conforms with the mesh wrapper interface, then no data recopy is involved. `Portage` embeds five built-in drivers:

- **uberdriver:** an easy to use mesh remap class.
- **coredriver:** a low-level mesh driver that allows finer control on remap steps.
- **mmdriver:** a legacy monolithic mesh remap driver.
- **driver_swarm:** a dedicated particle remap driver.
- **driver_mesh_swarm_mesh:** a mesh remap driver that relies on particle kernels.

A basic example of a single-material mesh remap using coredriver methods (using default parameters where possible) is given in ***Listing 1***. A list of available options for remap components is given in ***Table 1***. Each of them is templated on source and target domains as well as field entity types.

## SCALABILITY

`Portage` is designed for high performance computing clusters. It relies on both **MPI** and **OpenMP** to leverage the hybrid parallelism exposed by such architectures. Here we present some scaling results on a simple multi-material problem in ***Figure 6***. Tests are run on a cluster formed by **256** dual-socket nodes (**Intel Broadwell** with **18** cores per-socket at **2.1** Ghz). Here we consider a cell-centered three-material field remap with **3D** cartesian grids and a simple t-junction material distribution on the domain. The source and target grids have **$40^3$** and **$120^3$** cells respectively. To ease memory pressure, we set a single **MPI** rank per node and **16** threads per rank explicitly pinned on cores using `KMP_AFFINITY=granularity=core,compact`.

The total execution time and the remap time are depicted in black and red respectively. The time spent on material interface reconstruction – which is only

```cpp
using namespace Portage;

std::vector<std::string> fields = {"density", "temperature"};

/* the type of the driver to be used for remap templated on:
 * . the problem dimension.
 * . the field entity type: node or cell.
 * . the type of the mesh with a basic set of topological queries,
 * . the type of mesh state which contains fields to be remapped. */
using Remap = CoreDriver<dim, entity, Mesh, State>;

/* create the driver and pass the source and target meshes and states */
Remap remap(source_mesh, source_state, target_mesh, target_state);

/* 1. retrieve the list of source candidate cells that may overlap each target cell
 * 2. compute the weights by intersecting target cells with their candidate cells.
 * 3. check for boundaries mismatch and cache per—cell volume of intersection. */
auto candidates = remap.search<SearchKDTree>();
auto weights = remap.intersect_meshes<IntersectR2D>(candidates);
bool mismatched = remap.check_mismatch(weights);

/* 4. compute gradient of the current field on the source mesh,
 * it can be replaced by any user defined gradient computation.
 * 5. interpolate the current field on the target mesh.
 * 6. fix values for conservation using default parameters. */
for (auto&& field : fields) {
  auto grad = remap.compute_source_gradient(field);
  remap.interpolate_mesh_var<double,Interpolate_2ndOrder>(field, weights, &grad)
  if (mismatched) { remap.fix_mismatch(field, field); }
}
```

**Listing 1** Example of using a driver for remap.

| STEP | ALGORITHM | DESCRIPTION |
|---|---|---|
| SEARCH | **SearchSimple** | search using bounding-box (2D) |
| | **SearchKDTree** | search using a k-d tree |
| | **SearchSimplePoints** | basic quadratic search for particles |
| | **SearchPointsByCells** | linear search for particles using a virtual cell |
| WEIGHTS | **IntersectR*n*D** | compute exact *n*-polytopes intersection |
| | **IntersectSweptFace** | compute moments of advected regions |
| | **Accumulate** | evaluate and sum shape functions/derivatives |
| | | *options: shape kernels and geometry, basis, estimators* |
| INTERPOLATE | **Interpolate_1stOrder** | first-order interpolation of mesh values |
| | **Interpolate_2ndOrder** | second-order interpolation with limiters |
| | **Estimate** | *n*-order approximation of particle values |

**Table 1** Driver options for remap steps.

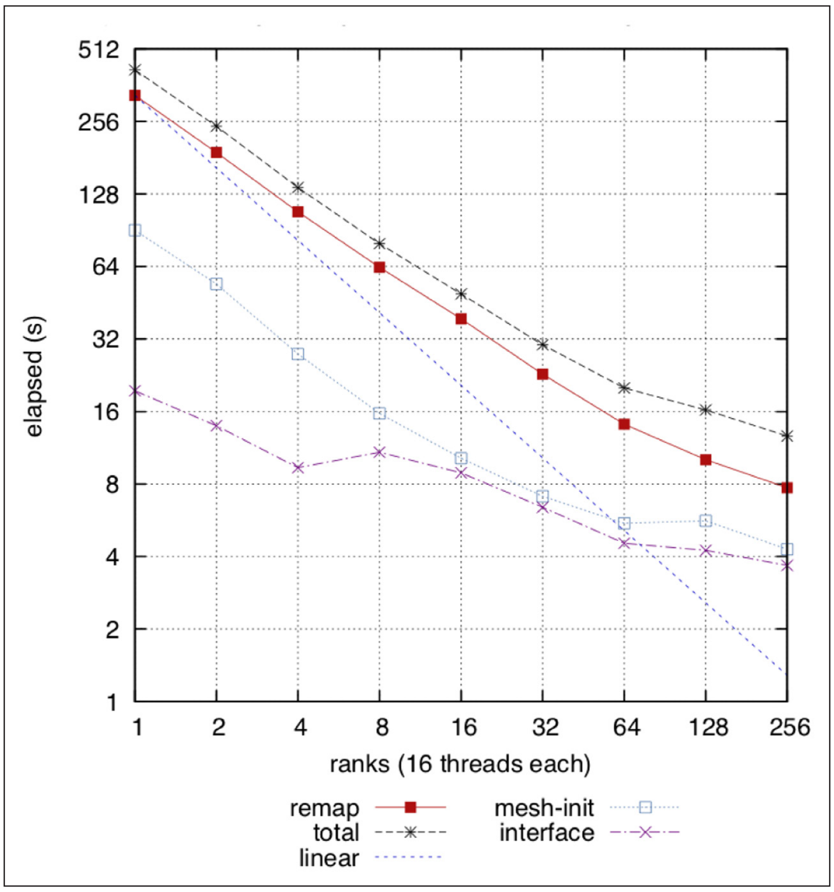


**Figure 6** Scaling of multi-material remap in a hybrid parallel setting.

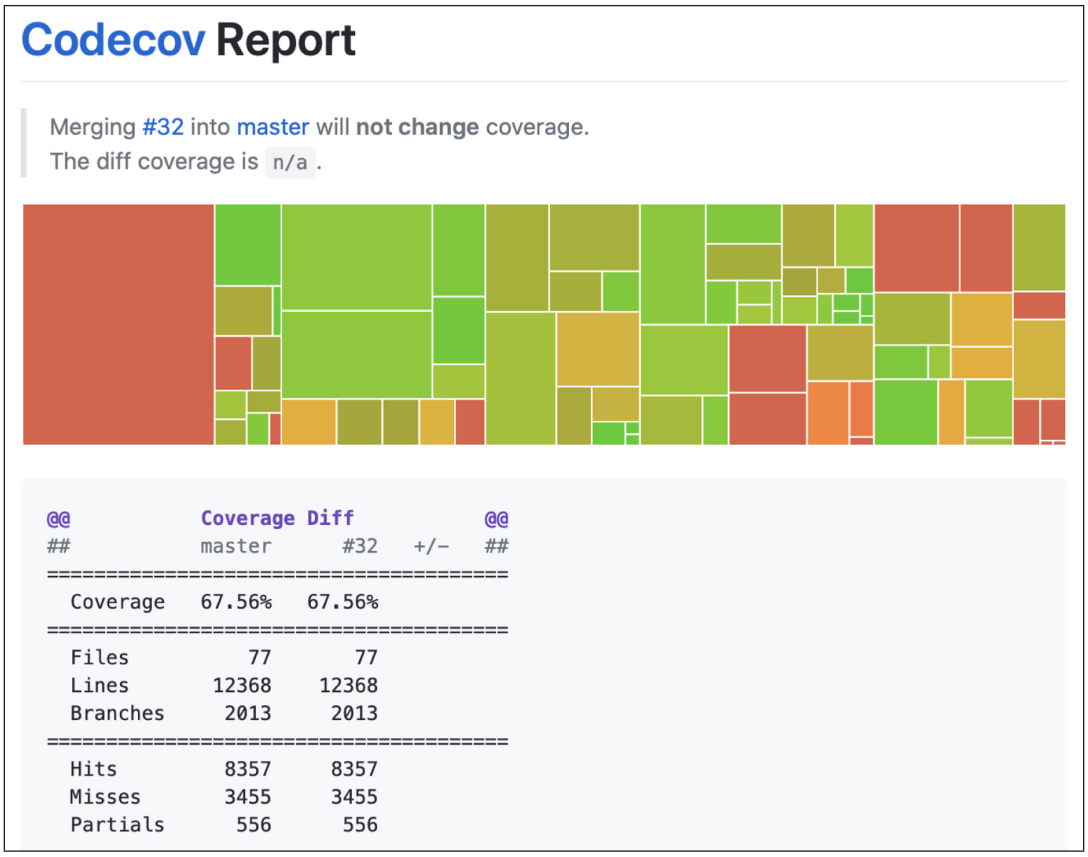
performed on multi-material cells – is shown in purple. Here, the workload per rank is impacted by the uneven distribution of multi-material cells. Despite the workload imbalance, a reasonable scaling is still achieved.

## QUALITY CONTROL

`Portage` is tested on Linux with **Gnu** and **Intel** compilers. It provides over **200** unit and functional tests as part of a **Travis** continuous integration setup using the **Github** workflow. In particular, they ensure that remap algorithms:

- are bounds preserving,
- provide the expected order of accuracy,
- are conservative.

The code coverage in the latest release is **67%** as shown in ***Figure 7***.



Codecov Report

Merging #32 into master will not change coverage.
The diff coverage is `n/a`.

```
@@           Coverage Diff          @@
##           master      #32   +/-   ##
=======================================
  Coverage   67.56%   67.56%
=======================================
  Files          77       77
  Lines       12368    12368
  Branches     2013     2013
=======================================
  Hits         8357     8357
  Misses       3455     3455
  Partials      556      556
```

**Figure 7** Code coverage in latest release.

## (2) AVAILABILITY

### OPERATING SYSTEM

`Portage` is designed for high performance computing clusters. Hence it is primarily targetted to **Linux**.

### PROGRAMMING LANGUAGE

`Portage` is written in `C++14`.

### ADDITIONAL SYSTEM REQUIREMENTS

None.

### DEPENDENCIES

minimal:

- `r3d`: exact polytope intersection. *https://github.com/devonmpowell/r3d*
- `cinch`: build utilities and options. *https://github.com/laristra/cinch*
- `wonton`: mesh wrappers and helpers. *https://github.com/laristra/wonton*

optional:

- `jali`: distributed mesh infrastructure. *https://github.com/lanl/jali*
- `tangram`: interface reconstruction. *https://github.com/laristra/tangram*
- `lapack`: linear algebra kernels. *http://www.netlib.org/lapack*
- `thrust`: a wrapper for OpenMP. *https://github.com/thrust/thrust*
- `tcmalloc`: fast memory allocation. *https://github.com/google/tcmalloc*

### LIST OF CONTRIBUTORS

All contributors are or were affiliated with **Los Alamos National Laboratory**.

- **current:** Angela Herring, Christopher Malone, Daniel Shevitz, Evgeny Kikinzon, Hoby Rakotoarivelo, Jan Velechovsky, Konstantin Lipnikov, Navamita Ray and Rao Garimella.
- **previous:** Brendan Krueger, Charles Ferenbaugh, Christopher Sewell, Gary Dilts, Onděj ěrtíķ, Michael Rogers and Rachel Ertl.

### SOFTWARE LOCATION

***Archive***

***Name:*** Portage
***Persistent identifier:*** *https://github.com/laristra/portage/releases*
***Licence:*** BSD
***Publisher:*** Angela Herring
***Version published:*** 2.2.3
***Date published:*** 27/04/2020
***DOI:*** *10.5281/zenodo.4571000*

***Code repository***

***Name:*** Portage

***Persistent identifier:*** https://github.com/laristra/portage
***Licence:*** BSD
***Date published:*** 01/09/2017
***Support:*** We will use the GitHub "issues" feature as well as email (*portage@lanl.gov*) to maintainers for support.

## LANGUAGE

English.

## (3) REUSE POTENTIAL

`Portage` is an extensible and mesh-agnostic library. Its unique design allows it to be re-used in a variety of applications such as:

- field remap in **ALE** simulations,
- multi-physics code-to-code field remap,
- operator-split intra-code linking.

`Portage` is actively developed, supported and continuously released. Bugs and feature requests can be notified using the issue tracker on **Github**, as well as any question related to the software. User support may be reached by email at *portage@lanl.gov*. We welcome community contributions through pull requests.

## ACKNOWLEDGEMENTS

We thank Benjamin Bergen, Brian Jean, Christoph Junghans, Davis Herring, Marc Charest and Mikhail Shashkov from **Los Alamos National Laboratory** for their help.

## FUNDING STATEMENT

This work is supported by the U.S. Department of Energy (DoE) through Los Alamos National Laboratory which is operated by Triad National Security, LLC for the National Nuclear Security Administration of the DoE under contract 89233218CNA000001.

## COMPETING INTERESTS

The authors have no competing interests to declare.

## AUTHOR AFFILIATIONS

**Angela Herring**
Lagrangian codes, Los Alamos National Laboratory, USA

**Charles Ferenbaugh**
Applied computer science, Los Alamos National Laboratory, USA

**Christopher Malone**
Lagrangian codes, Los Alamos National Laboratory, USA

**Daniel Shevitz**
Applied computer science, Los Alamos National Laboratory, USA

**Evgeny Kikinzon**
Applied computer science, Los Alamos National Laboratory, USA

**Gary Dilts**
Computational physics and methods, Los Alamos National Laboratory, USA

**Hoby Rakotoarivelo**
Applied mathematics and plasma physics, Los Alamos National Laboratory, USA

**Jan Velechovsky**
Eulerian codes, Los Alamos National Laboratory, USA

**Konstantin Lipnikov**
Applied mathematics and plasma physics, Los Alamos National Laboratory, USA

**Navamita Ray**
Applied computer science, Los Alamos National Laboratory, USA

**Rao Garimella**
Applied mathematics and plasma physics, Los Alamos National Laboratory, USA

## REFERENCES

1. **Hirt C, Amsden A, Cook JL.** An arbitrary Lagrangian-Eulerian computing method for all flow speeds. *Journal of Computational Physics*. 1974; 14(3): 227–253. DOI: *https://doi.org/10.1016/0021-9991(74)90051-5*
2. **Barlow A, Maire P-H, Rider W, Rieben R, Shashkov M.** Arbitrary Lagrangian-Eulerian Methods for Modeling High-Speed Compressible Multimaterial Flows. *Journal of Computational Physics*. 2016; 322: 603–665. DOI: *https://doi.org/10.1016/j.jcp.2016.07.001*
3. **Burton D.** Lagrangian Hydrodynamics in the FLAG Code. *Tech. rep. LA-UR-07-7547*. Los Alamos National Laboratory; 2007.
4. **Painter S, Coon E, Atchley A, Berndt M, Garimella R, Moulton D, Svyatskiy D, Wilson C.** Integrated surface/subsurface permafrost thermal hydrology: Model formulation and proof-of-concept simulations. *Water Resources Research*. 2016; 52. DOI: *https://doi.org/10.1002/2015WR018427*
5. **Herring A.** Automated Engineering-Physics link in Ingen. *Tech. rep. 2014–26185*. Los Alamos National Laboratory; 2015.
6. **Grandy J.** Conservative Remapping and Region Overlays by Inter-secting Arbitrary Polyhedra. *Journal of Computational Physics*. 1999; 148(2): 433–466. DOI: *https://doi.org/10.1006/jcph.1998.6125*
7. **Dukowicz J, Padial N.** Remap3D: a conservative three-dimensional remapping code. *Tech. rep. LA-12136-MS*. Los Alamos National Laboratory; 1991.
8. **Slattery SR, Wilson PPH, Pawlowski RP.** The Data Transfer Kit: A Geometric Rendezvous-Based Tool for Multiphysics Data Transfer. *International Conference on Maths and Comput. Methods Applied to Nuclear Science & Engineering*. 2013; 5–9.
9. **Zalesak S.** Fully multidimensional ux-corrected transport algorithms for uids. *Journal of Computational Physics*. 1979; 31(3): 335–362. DOI: *https://doi.org/10.1016/0021-9991(79)90051-2*

10. **Dilts G.** Estimation of Integral Operators on Random Data. *Tech. rep. LA-UR-17-23408*. Los Alamos National Laboratory; 2017.

11. **Barth T, Jespersen D.** The design and application of upwind schemes on unstructured meshes. *27th Aerospace Sciences Meeting*; 1989. DOI: *https://doi.org/10.2514/6.1989-366*

TO CITE THIS ARTICLE:
Herring A, Ferenbaugh C, Malone C, Shevitz D, Kikinzon E, Dilts G, Rakotoarivelo H, Velechovsky J, Lipnikov K, Ray N, Garimella R 2021 `Portage`: A Modular Data Remap Library for Multiphysics Applications on Advanced Architectures. *Journal of Open Research Software,* 9: 26. DOI: *https://doi.org/10.5334/jors.338*

Submitted: 26 June 2020 Accepted: 12 March 2021 Published: 28 September 2021