Flexible and scalable particle-in-cell methods with adaptive mesh refinement for geodynamic computations

Rene Gassmöller[1], Harsha Lokavarapu[1], Eric Heien[2], Elbridge Gerry Puckett[3] and Wolfgang Bangerth[4]

[1]Department of Earth and Physical Sciences, University of California, Davis

[2]Computational Infrastructure for Geodynamics, University of California, Davis

[3]Department of Mathematics, University of California, Davis

[4]Department of Mathematics, Colorado State University, Fort Collins

Key Points:

- Particle-in-cell methods require new algorithms when used with finite element calculations on dynamically partitioned, adaptively refined, unstructured meshes.
- We present approaches for particle generation, sorting, and hybrid load balancing in hierarchically refined finite celement computations.
- We show scalability and applicability of the developed methods for problems in computational geodynamics.

Corresponding author: Rene Gassmöller, `rene.gassmoeller@mailbox.org`

Abstract

Particle-in-cell (PIC) methods couple mesh-based methods for the solution of continuum mechanics problems with the ability to advect and evolve properties on particles. PIC methods have a long history and numerous applications in geodynamic modeling. However, they are historically either implemented in sequential codes, or in parallel codes with structured, statically partitioned meshes. Yet, today's codes increasingly use adaptive mesh refinement (AMR) of unstructured coarse meshes, dynamic repartitioning, and scale to thousands of processors. Optimally balancing the work per processor for a PIC method in these environments is a difficult problem, and many existing implementations are not sufficient for this task. Thus, there is a need to revisit these algorithms for future applications.

Here we describe challenges and solutions to implement PIC methods in the context of large-scale parallel geodynamic modeling codes that use dynamically changing meshes. We also provide guidance for how to address bottlenecks that impede the efficient implementation of these algorithms and demonstrate with numerical tests that our algorithms can be implemented with optimal complexity and that they are suitable for large-scale, practical applications. We provide a reference implementation in ASPECT (Advanced Solver for Problems in Earth's convection), an open source code for geodynamic modeling built on the DEAL.II finite element library.

1 Introduction

Most methodologies to numerically solve flow problems are based on a continuum description in the form of partial differential equations, and include the finite element, finite volume, and finite difference methods. On the other hand, it is often desirable to couple these methods with discrete, "particle" approaches for a number of applications. These include, for example, visualization of flows, tracking interfaces and origins, or tracking the history of material. Use cases and discussions of computational methods can be found as far back as *Harlow* [1962] and are often referred to as *particle-in-cell* (PIC) methods.

Different implementations of such methods can be found in the geodynamic literature [*Poliakov and Podladchikov*, 1992; *Moresi et al.*, 2003; *Gerya and Yuen*, 2003; *McNamara and Zhong*, 2004; *Popov and Sobolev*, 2008; *Thielmann et al.*, 2014], but almost all of these methods were developed for either structured meshes and/or sequential computations. However, over the past two decades adaptive finite element methods have demonstrated that they are vastly more accurate than computations on uniformly refined meshes [*Carey*, 1997; *Ainsworth and Oden*, 2000; *Bangerth and Rannacher*, 2003; *Kronbichler et al.*, 2012; *Heister et al.*, 2017], and have been successfully combined with PIC methods in other fields [*Wallstedt and Guilkey*, 2010; *Adams et al.*, 2015; *Balay et al.*, 2018; *Almgren et al.*, 2013]. While many parts of existing particle-in-cell algorithms can still be used in this context, a number of new algorithmic challenges arise. The present contribution is therefore primarily an assessment of possible algorithms when implementing particle methods for computational geodynamics in the following two situations:

1. Unstructured, hierarchically refined quad-/octree, 2D/3D meshes that change dynamically and potentially utilize higher order polynomial mappings to represent curved geometries;
2. Large parallel computations that run on thousands of cores, using tens of millions of cells, and billions of particles.

Specifically, we will discuss the following components, along with an assessment of their practical performance:

1. Parallel generation of particles in unstructured meshes;
2. Treatment of particles as they cross cell and processor boundaries;
3. Treatment of particles during mesh refinement and coarsening, including appropriate load balancing.

Other components in our reference implementation use well understood algorithms: We use standard C++-containers as data structures; integrate the particle trajectories using Forward-Euler, Runge-Kutta 2 or 4 integration schemes with higher order accuracy in space and time; store variable scalar, vector, or tensor-valued properties on particles; and transfer information between particles and mesh using simple arithmetic or harmonic cell averaging schemes, or least-squares projections. Massively parallel output capability is provided by the VTK [*Schroeder et al.*, 2006] and HDF5 [*Folk et al.*, 1999] data formats. As our manuscript is focussed on the particular difficulties of combining particle methods with adaptive finite element computations, we do not discuss traditional difficulties of particle methods, such as memory locality or particle clustering, as these have already been addressed elsewhere [*Mellor-Crummey et al.*, 2001; *Wang et al.*, 2015].

We provide a reference implementation of the presented methods in the geodynamic modeling code ASPECT [*Kronbichler et al.*, 2012; *Bangerth et al.*, 2017a; *Heister et al.*, 2017], and include most of the discipline-independent methods in the deal.II finite element library [*Bangerth et al.*, 2007; *Arndt et al.*, 2017], thus making them available for a variety of applications and scientific disciplines. Given that our implementation is based on deal.II, we will henceforth only consider quadrilateral and hexahedral, hierarchically refined meshes, which are balanced by a 2:1 refinement ratio between neighboring cells. Exploiting these assumptions allows us to optimize our algorithms, but we believe that generalizations to other situations are often straightforward.

2 Computational methods

2.1 Parallel particle generation

The first step in using particles in mesh-based solvers is their creation on all involved processors, and depending on their purpose, initial particle distributions may vary widely. Two broad classes of initial distributions come to mind:

Random particle positions. Randomly chosen particle locations are often used in cases where particles represent the values of a field; e.g., the origin and movement of a specific type of material. In these cases, one is not interested in prescribing exact initial particle locations, and randomly chosen locations are acceptable. The probability distribution, $\rho(\mathbf{x})$, from which locations are drawn is often chosen as uniform over the domain. Alternatively, one can use a higher particle density in regions of interest, for example to better resolve steep gradients, which can be interpreted as equivalent to AMR in mesh-based methods.

We propose the following algorithm, running on each processor:

1. Compute and store local cell weights as integral of $\rho(\mathbf{x})$ over each local cell.
2. Compute the global sum of the local cell weight integrals.
3. Compute the local number of particles as ratio between local and global weight integral times global number of particles.
4. Compute the local starting particle index based on the partial sum of local number of particles of all processes with lower rank.

5. Either: Compute the number of particles per cell by randomly drawing cells K according to their weight repeatedly and tallying up how many times each cell was selected.
6. Or: Compute the number of particles per cell according to their share of the local integral of $\rho(\mathbf{x})$
7. Generate local particles in each cell K by drawing random locations inside its axes-parallel bounding box B_K until we find a position in K (see Supplementary Text S1 for details).

Apart from the two global reductions to determine the global weight and the local start index, all of the operations above are local to each processor. Thus, the overall run time for generating particles is proportional to the number of particles on the process with the largest number of particles, i.e., of optimal complexity in the global number of particles and, if the number of particles per process is balanced, also in the number of processes. However, this balancing is often not the case in practice (see Section 2.3).

We note that our algorithm yields a number of particles on each process that is deterministic. Consequently, the distribution of particles is not entirely random. However, in practice we find this does not matter for sufficiently many particles.

Prescribed particle locations An alternative to the random arrangements of particles is to exactly prescribe initial locations, either algorithmically (e.g., a regular grid), or by reading locations from a file. Surprisingly, for distributed unstructured meshes this case is more computationally expensive than randomly generated particle locations.

Let us assume that the initial positions of all particles are given in an array $\{\mathbf{x}_k\}$, $k = 1 \ldots N$. Then for each particle one has to find its surrounding cell, which in the worst case, is of complexity global number of particles times local number of cells. This is because, for general unstructured meshes, we can not predict whether a given particle's location lies inside the locally owned cells without searching through all cells. This limits the usefulness of the algorithm to moderate numbers of particles. However, the algorithm can be accelerated by checking whether a particle's location lies inside the bounding box of the locally owned cells, before checking each cell.

On hierarchically refined meshes, one can alternatively find the cell K by finding the coarse level cell in which it is located, and then recursively searching through its children. This reduces the complexity to the global number of particles times the logarithm of the local number of cells. However, it only works if child cells occupy the same volume as their parent cell; this condition is often not met when using nonlinear polynomial mappings to represent curved geometries.

In the paragraphs above we assume that the particle positions are known in the global coordinate system, and we have to search for the surrounding cell. If however, the particle coordinates are known in the *local* cell coordinate system (e.g. the center), then the algorithm is much simpler. A loop over all cells and all local particle coordinates that are then mapped into the real space [as used in *Puckett et al.*, 2017] will generate the particles in the optimal order, and will be cheap.

2.2 Transport between cells and subdomains

PIC codes in geodynamics contain a time integration in which one computes a velocity field (usually on some grid), and then moves the particles with the flow field. To parallelize these computations the grid is usually fully distributed, which means each process only knows about local cells and one layer of “ghost” cells around the local domain. Thus, after each particle movement the new particle location is either inside its original cell K or in a different cell K'. To be able to transfer data between particle and grid we then need to find its new cell that may be owned by the same processor or a different one. The challenge, in the context of adaptive, distributed meshes lies in constructing algorithms that can efficiently search for the new surrounding cells of particles, as well as potentially transfer the particle to a different processor. In practice, communication patterns that cover the exchange of particles between processes that own adjacent parts of the mesh are often sufficient to implement, i.e., using point-to-point messages. In particular, this is possible if the (ODE) time step is chosen such that the CFL number is less than or equal to one, because then particles travel no more than one cell diameter in each step. Following these arguments our reference implementation employs the following algorithm, executed for each particle that is not in its old cell:

1. Search for the locally known current cell K'.
2. If K' is owned by the current process, mark the particle as being in K'.
3. If K' is in a ghost cell owned by the process p, mark the particle for transmission to p.

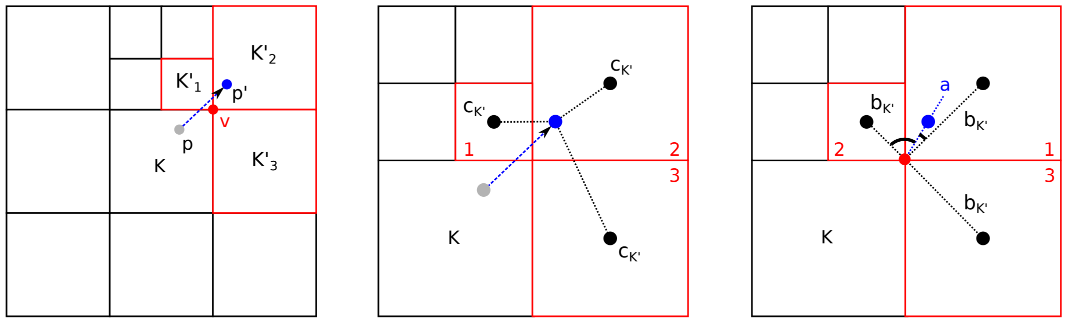


Figure 1. In 2:1 balanced quadtree meshes finding the new cell K' for a particle that has left its old cell K is a nontrivial problem. Limiting the search to the cells that contain the vertex of the old cell that is closest to the new particle position (left panel) reduces the search cost. Note that sorting neighbor cells according to angle between a and $b_{K'}$ (right panel, see main text for definitions) correctly predicts the search order (red numbers) in most cases, while a simpler criterion like particle–cell–center distance mispredicts the new cell (center panel).

4. If K' cannot be found, mark particle for deletion.

After the algorithm has finished, (1) all particles marked for transmission are communicated to their neighbors that now own them, (2) all particles that have been lost or communicated are removed from local storage, and (3) all particles with a new cell association (local or communicated) are reinserted into local storage. This bulk handling is advantageous, since particles of the same cell tend to move into the same neighbor cells, and a collective insertion reduces copies and reallocation of memory.

The vast majority of particles remain in the current cell, end up in a new local cell (option 2), or a cell owned by another process (option 3). A few cases, however, do not fall in these categories. First, the ODE integrator error during particle movement can carry a particle over a processor boundary, and out of the one-cell ghost layer. Second, the integrator error can transport a particle across a geometry boundary, after which it is not contained in any cell. For a benchmark model setup (see Supplementary Dataset S1) we have found that only a negligible fraction of particles is lost because of these two mechanisms. As expected, an explicit Euler scheme loses significantly more particles than the RK2 methods, while decreasing the time step significantly reduces the loss. Given the small overall loss and added computational expense to reduce the timestep, dropping particles that fall out of bounds (option 4) seems like a reasonable approach to us.

The algorithm above requires finding the cell a particle is in now (Step 1). As discussed in Section 2.1, without additional information this requires $O(N_{\text{local cells}})$ operations, all of which are expensive. Furthermore, because many of the local particles cross to a different cell, this step is not of optimal – i.e., $O(N)$ for N particles – complexity. While the tree structure of the mesh makes it possible to implement global tree-search algorithms with logarithmic complexity in the number of cells [*Isaac et al.*, 2015], we found that the algorithm spends the majority of its works on determining whether a particle is inside a cell K', i.e., inverting the mapping of K' for the position of the particle. Since in our application the vast majority of particles only cross from one cell to its neighbors, we can accelerate the global algorithms significantly by first searching all neighbor cells in an order that makes it likely that we find the correct one early. Only the very small fraction that does not end up in a neighbor then requires an expensive search over all cells. We note that for problems without this local property other algorithms might be more appropriate [e.g. *Mirzadeh et al.*, 2016; *Burstedde*, 2018].

Following some experimentation, we found that the following strategy to pre-sort local neighbor cells works best (see also Fig. 1): Let $\mathbf{p}'$ be the particle's current position, K the known previous cell of the particle, $\mathbf{v}$ be the vertex of K closest to $\mathbf{p}'$, and $\mathbf{c}_{K'}$ be the center of the potential new cell K', which is a vertex neighbor of K adjacent to vertex $\mathbf{v}$ (see Fig. 1). Let $\mathbf{a} = \mathbf{p}' - \mathbf{v}$ be the normalized vector from the closest vertex of K to the particle, and $\mathbf{b}_{K'} = \mathbf{c}_{K'} - \mathbf{v}$ be the normalized vector from the closest vertex to the center of cell K'. Then we search through all K' in the order of descending scalar product $\mathbf{a} \cdot \mathbf{b}_{K'}$ (Fig. 1, right panel). In other words, cells with a center in the direction of the particle movement are checked first. This algorithm is somewhat similar to the one proposed by [*Capodaglio and Aulisa*, 2017], with the difference that we know our particle is in one of the neighbors of the old cell, and we therefore search through a sorted list of neighbor cells, instead of along a computed search path through multiple cells. While there are corner cases in which our algorithm fails to find the new cell in the first try, in practice more than 98% of the particles moving to a new cell in the models discussed in Section 3 are found immediately. The rest of the particles is found in at most 2 (2D) or 4 (3D) searches, except if the particle left the immediate neighbors of the old cell as discussed above. Simpler criteria – like searching by distance between particle and cell center (Fig. 1, middle panel) – fail more often, in particular for adaptively refined neighbors.

If particles crossed a process boundary they are communicated to neighboring processes in two steps. First, two integers are exchanged between every neighbor and the current process, representing the number of particles that will be sent and received. In a second step every process transmits the serialized particle data and receives its respective data from its neighbors. This allows us to implement all communications as non-blocking point-to-point MPI transfers, only generating $O(1)$ transmissions and $O(N_{\text{local particles}})$ data per process. Since we already determined which ghost cell contains this particle on the old process, we also transmit this information. Because ghost cells are guaranteed to exist on the owning process we thus avoid another search for the enclosing cell on the new process.

2.3 Handling adaptively refined, dynamically changing meshes

In the current context, adaptively refined, dynamically changing meshes present two particular challenges.

Mesh refinement and repartitioning Typically, refinement and coarsening happens in two steps: First, cells are refined or coarsened separately on each process, and particles are distributed to the children of their previous cell (upon refinement), or are merged to the parent of their previous cell (upon coarsening). The second step of mesh adaptation consists of redistributing the resulting mesh to achieve an efficient parallel load distribution [*Burstedde et al.*, 2011; *Bangerth et al.*, 2011]. To keep this process simple we append the serialized particle data to other data already attached to a cell (such as vertex locations and values of field based solution variables), and transmit all data at the same time. We can therefore utilize existing software for parallel mesh handling [*Burstedde et al.*, 2011], which uses well-optimized bulk communication patterns, and thereby avoid sending particles individually or having to re-join particles with their cells.

Load balancing The mesh repartitioning discussed in the previous paragraph is designed to redistribute work equally among all available processes. For mesh-based methods, this typically means equilibrating the number of cells each process “owns”. On the other hand, in the context of PIC methods for adaptive meshes, the number of particles per cell frequently ranges from zero to a few hundred. Consequently, the described process leads to unbalanced workloads during particle-related parts of the code. Conversely, rebalancing the mesh to equilibrate the number of particles leaves the mesh-based algorithms with unbalanced workloads. Both situations reduce the overall parallel efficiency of the code.

The only approach to restore perfect scalability is to partition cells differently for the mesh-based and particle-based parts of the code. On the other hand, one can not avoid transporting all mesh and particle data during these rebalancing steps, because each phase of the algorithm might require all data from the other. Consequently, the amount of data that has to be transported twice per time step is significant.

In practice, some level of imbalance can often be tolerated. One can work with the following compromise solutions:

1. *Repartition the mesh according to the combined particle and cell load ("Balanced repartitioning").* Instead of estimating the workload of each cell during the rebalancing step as constant (pure mesh-based methods) or proportional to the number of particles in a cell (pure particle-based methods), one can estimate it as an appropriately weighted sum of the two. The resulting mesh is optimal for neither of the two phases, but is better balanced than either of the extremes (see Section 3.2 and Supporting Figure S1).
2. *Ignore imbalance.* As long as the number of particles is small one may simply ignore the imbalance. A typical case is when particles are only used to output information for a few specific points of interest, e.g. an accumulated strain profile through a subducting slab.
3. *Adjust particle density to mesh during particle generation.* The particle density can be chosen to follow the mesh resolution, if the region of highest mesh resolution is known in advance. This is most useful for tracking pre-existing interfaces. The higher particle density close to the interface then not only increases the accuracy in regions of interest, but it also improves parallel efficiency and scalability.
4. *Adjust mesh to particle density.* Instead of prescribing the particle density following the mesh, the mesh resolution can also be adjusted to the particle distribution. As in the previous alternative, the alignment of mesh and particle density yields better parallel efficiency and scaling.
5. *Adjust particle density to mesh by particle population management.* In cases of a priori unknown regions of high mesh density it can be necessary to manage the particle density actively during the model run. This includes removing particles from

regions with high particle density or adding particles in regions of low density. If done appropriately, the result will be a mesh where the average number of particles per cell is managed so that it remains approximately constant.

While the last three approaches lead to better scalability, they may of course not suit the problem one originally wanted to solve. On the other hand, generating additional particles upon refinement of a cell, and thinning out particles upon coarsening, is a common strategy in existing codes [*Popov and Sobolev*, 2008; *Leng and Zhong*, 2011]. We also note that while load balancing is particularly important for dynamically changing adaptive meshes, it is also beneficial for uniform meshes if the particle distribution happens to be non-uniform.

3 Scalability

To verify our claims of performance and scalability, we show that our algorithms scale well to typical model sizes in computational geodynamics. Technical information about the used hardware, and the definition of the timing events is provided in Supplementary Text S2. Additional benchmarks confirming the correctness of the implemented advection schemes is provided in Supporting Text S3, and Figure S1.

3.1 Uniform meshes

We first show scalability using a two-dimensional benchmark case with a static and uniformly refined mesh. We employ a circular-flow setup in a spherical shell, with no flow across the boundary. Particles are distributed randomly with uniform density (see Fig. 2, top left), and are advected using a RK2 integration scheme.

The top row of Fig. 2 shows excellent weak and strong scaling over at least three orders of magnitude of model size. For a fixed problem size (strong scaling), we use a mesh with $786,432 = 12 \cdot 256^2$ cells and $1.536 \cdot 10^7$ particles. Increasing the number of processes from 12 to 12,288 shows an almost perfect decrease in wall time for all operations, despite the rather small problem each process has to deal with for large numbers of processes. Note that the scaling of the “Exchange particles” event is likely specific to the used network topology and probably shows the transition from a large-throughput large-latency mode of transfer to a small message-size small-latency transfer.

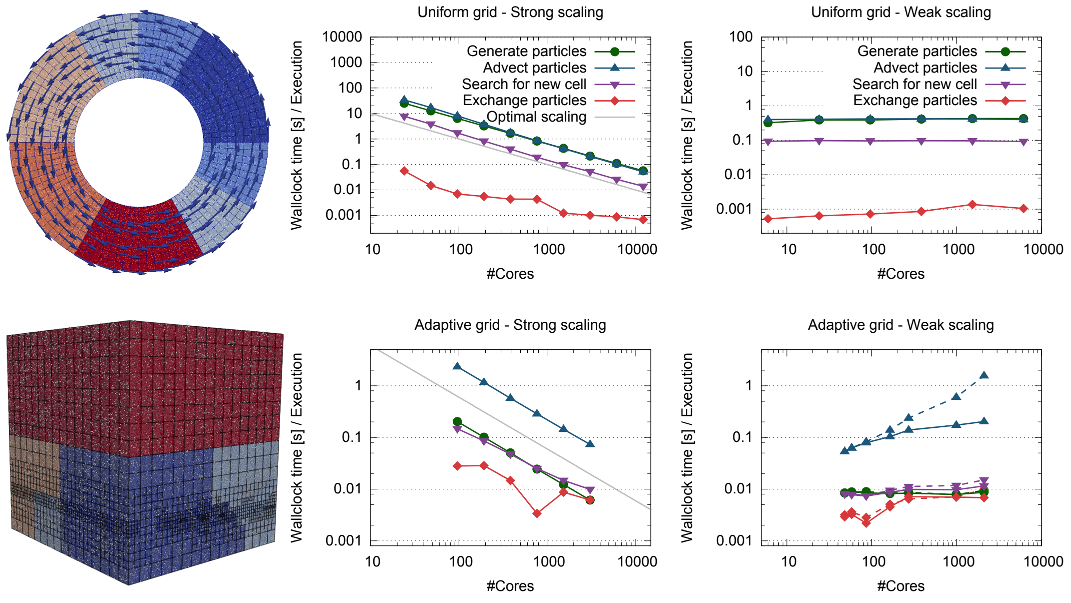


Figure 2. Scaling of algorithms. Top row: Results for a uniformly refined mesh. Bottom row: Results for an adaptively refined mesh. Left column: Model geometry and initial parallel partition. Center column: Strong scaling for a constant number of cells and particles. Top right: Weak scaling for a uniform mesh with a constant number of cells and particles per process. Bottom right: Weak scaling for an adaptive mesh with a fixed (though increasingly unbalanced) number of cells and particles per process. The dashed models use the common cell load balancing, while the solid models use balanced repartitioning as described in Subsection 2.3.

Keeping the number of cells and particles per core fixed and increasing the problem size and number of processes accordingly (weak scaling, Fig. 2, top right panel), the wall-clock time stays constant between 6 and 6,144 processes. In this test each process owns 512 cells and $1.0 \cdot 10^4$ particles. Each refinement step leads to four times as many cells, and consequently processes. 6,144 cores was the last multiple to which we had access for timing purposes. Results again show excellent scalability, even to large problem sizes, in this case approximately 3 million cells and 61 million particles.

3.2 Adaptively refined meshes

Discussing scalability for *adaptive* meshes is more complicated because increasing level of refinement does not create a predictable number of cells. We apply the same particle distribution and integration as for the uniform mesh case, but use a model setup based on the benchmarks presented in [*van Keken et al.*, 1997], extended to three spatial dimensions. Specifically, we use a rectangular domain $[0, 0.9142] \times [0, 1] \times [0, 1]$ that contains a sharp non-horizontal interface separating a less dense lower layer from a denser upper layer. The shape of the interface then leads to a Rayleigh-Taylor instability. For the strong scaling tests, we create an adaptive mesh of at most 256^3 cells, retaining fine cells only in the vicinity of the interface. This mesh consists of approximately 1,000,000 cells, and we generate approximately 30 million, uniformly distributed particles, and run this setup on increasing numbers of processors.

The results in Fig. 2 show that *strong* scaling for the adaptive grid case is nearly as good as for the uniform grid case, decreasing the total runtime essentially linearly from 96 to 3,072 cores. The small worse-than-linear component of the cell-search algorithm seems to be related to the imbalance between particles and cells that will be further discussed in the weak scaling results, but since this part is one order of magnitude cheaper than the particle advection it will only limit the scalability beyond 10,000 cores. As for the uniform mesh the "Exchange particles" algorithm shows some variations, likely caused by the interaction between the allocated compute nodes and the network topology used for the tests. Because this scaling test actually solves for the Stokes solution on the finite element mesh we are more restricted in the number of possible model sizes compared to the synthetic test for uniform meshes above. Increased memory consumption excludes very small core numbers and limited scaling of the Stokes solver for very small number of degrees of freedom per core limits the maximum number of cores. Nevertheless, 100 to 3000 cores is the most common model size for our application and increasing or decreasing the model size has not revealed significant changes to the scaling behavior outside of the here presented range.

Setting up weak scaling tests requires further consideration. Since we can not predict the number of cells for a given number of mesh refinements, we use a 16^3 mesh and adaptively refine it a variable number of times taking note of the resulting numbers of cells. We then run this model series with increasing number of cores to keep the number of cells per process approximately constant at 550 cells per process. Each of the models uses ≈ 25 times as many particles as cells, uniformly distributed across the domain.

The weak scaling results are more difficult to interpret than the strong scaling case. In a first series, we only strive to balance the number of cells per process (Option 2 in Subsection 2.3). However, because the particle density is constant while cell sizes increasingly vary, the imbalance in the number of particles per process grows with the size of the model. This is easily seen in the bottom left panel of Fig. 2 in which all four processes own the same number of cells, but vastly different volumes and consequently numbers of particles. Therefore, the run time for some parts of the algorithm – in particular for particle advection – grows as the model size increases (dashed lines, bottom right panel of Fig. 2).

As discussed in Section 2.3, this effect can be addressed by balancing cell and particle numbers. The solid lines in the bottom right panel of Fig. 2 show that with appropriately chosen weights, the increase in runtime can be reduced from a factor of 30 to a factor of 4. To achieve this, we introduce a cost factor W for each particle. The total cost of each cell in load balancing is then one (the cost of the field-based methods per cell) plus W times the number of particles in this cell. $W = 0$ implies that we only consider the number of cells for load balancing, whereas $W = \infty$ only considers the number of particles. In practice, one will typically choose $0 \leq W < 1$; for realistic applications, we found $W = 0.01$ to be adequate. On the other hand, computational experiments suggest that it is not important to *exactly* determine the optimal value since the overall runtime varies only weakly in the vicinity of the minimum (see Supporting Figure S1).

4 Example application: Convection in the Earth's mantle

We illustrate the applicability of our algorithms to realistic applications by modeling compressible Stokes flow in the Earth's mantle constrained by known movements of the tectonic plates at the surface for the past 250 million years. The equations we solve and the model setup are identical to a previously published model [*Heister et al.*, 2017], but enhanced by adding 4.8 million particles, which are used to track material movement over time. The particles are generated randomly with a uniform distribution, are integrated with a RK2 integration scheme, and in order to enforce balanced parallel workloads we limit the maximum number of particles per cell to 25 and remove additional particles dynamically during the model run. Therefore, at the final time regions with coarse cells have a lower particle density than finely resolved regions (see right panel of Fig. 3). As the number of particles is relatively small, it was not necessary to use balanced repartitioning to improve load balancing. Material properties such as density and heat capacity are computed from a database for basaltic and harzburgitic rocks, following [*Nakagawa et al.*, 2009], and the viscosity is based on a published viscosity model incorporating mineral

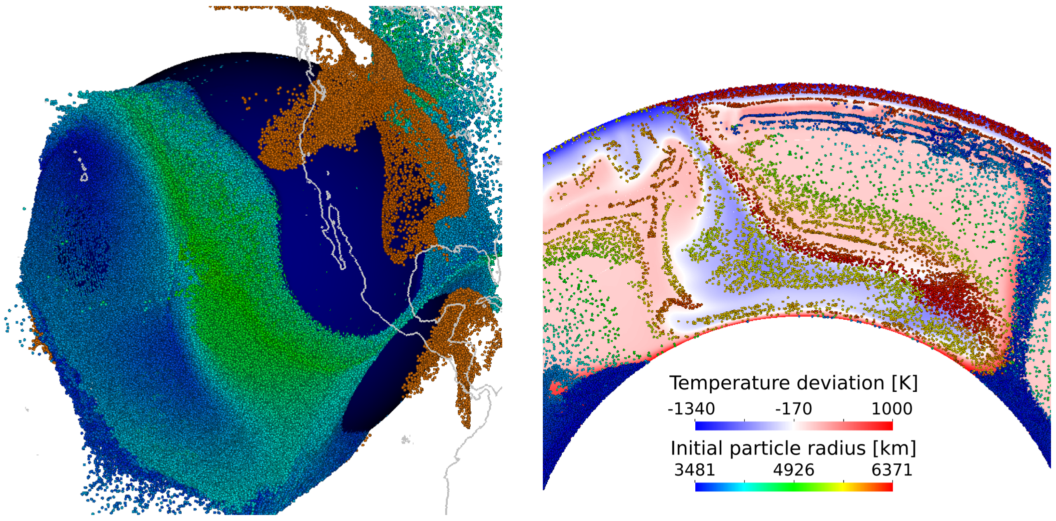


Figure 3. Illustration of a 3D mantle convection model with particles. Left: Subducting plates below the Western United States (brown particles) push material at the core-mantle boundary (dark blue sphere) towards the west. Only a selection of particles is shown, and each is colored by the distance from its initial position (blue: small to green: large). Right: Vertical slice through the subduction zone. All particles close to the slice are shown, and they are colored by the radius of their initial position (red: surface; blue: core-mantle boundary).

physics properties, geoid deformation, and seismic tomography [*Steinberger and Calderwood*, 2006]. The prescribed surface velocities use reconstructions of past plate movement on Earth [*Seton et al.*, 2012].

In the first time steps of this example model (before the number of particles is influenced by particle deletion) particle advection takes approximately 2.0 s per time step, particle cell-search requires 1.4 s per time step, particle generation is a one time process requiring 6.7 s, and particle communication was negligible, compared to a total time per time step of 26 s. A linear extrapolation to a larger number of particles (e.g. 20 per cell, as needed for active particles) would suggest a total particle cost of about 50 % of the total runtime, although this is highly simplified as for more particles a balanced repartitioning strategy could save significant amounts of runtime.

Fig. 3 shows a part of the example model, the present-day state of the Farallon subduction zone below the Western United States. Particles that are initially close to the core-mantle boundary are colored by the displacement they have experienced. This reveals that the Farallon slab (orange) has primarily pushed the easternmost material. Particles in the

Central Pacific have not moved significantly, illuminating the limited influence of the West Pacific subduction zones.

5 Conclusions

In this article, we have presented strategies for implementing PIC methods in computational geodynamic problems that use unstructured adaptive meshes. We have described our algorithms for the parallel generation of particles including both random and prescribed particle locations, and how utilizing information about the neighbors of cells can efficiently help to predict the owning cell of a particle. We discussed different load balancing techniques during mesh repartitioning and explained how balanced repartitioning can improve scalability significantly even in the presence of imbalanced workloads such as the ones that occur when combining unstructured AMR and PIC methods. Finally, we have documented in scaling tests and application examples that the expected optimal complexities can indeed be realized in practice. While there is certainly room for optimization in the presented algorithms, we are convinced that the present state allows for useful combination of unstructured AMR and PIC techniques in geodynamic modeling codes. Our implementation is freely available as part of the ASPECT and DEAL.II software.

Acknowledgments

All models were computed with the open-source software ASPECT [*Bangerth et al.*, 2017b, `http://aspect.geodynamics.org`] published under the GPL2 license, and the necessary data to reproduce the models is included in the supplementary material. We thank the Computational Infrastructure for Geodynamics (http://geodynamics.org) which is funded by the National Science Foundation under awards EAR-0949446 and EAR-1550901.

R. Gassmöller and W. Bangerth were partially supported by the National Science Foundation under award OCI-1148116 as part of the Software Infrastructure for Sustained Innovation (SI2) program; and by the Computational Infrastructure in Geodynamics initiative (CIG), through the National Science Foundation under Award No. EAR-0949446 and The University of California – Davis. E. G. Puckett was supported by the National Science Foundation under Award ACI-1440811 as part of the SI2 Scientific Software Elements (SSE) program.

The computational resources were provided by the North-German Supercomputing Alliance (HLRN) as part of the project bbk00003 "Plume-Plate interaction in 3D mantle flow – Revealing the role of internal plume dynamics on global hot spot volcanism".

References

Adams, M., P. O. Schwartz, H. Johansen, P. Colella, T. J. Ligocki, D. Martin, N. Keen, D. Graves, D. Modiano, B. Van Straalen, et al. (2015), Chombo software package for amr applications-design document, *Tech. rep.*

Ainsworth, M., and J. T. Oden (2000), *A Posteriori Error Estimation in Finite Element Analysis*, John Wiley and Sons.

Almgren, A. S., J. B. Bell, M. J. Lijewski, Z. Lukić, and E. Van Andel (2013), Nyx: A massively parallel amr code for computational cosmology, *The Astrophysical Journal*, *765*(1), 39.

Arndt, D., W. Bangerth, D. Davydov, T. Heister, L. Heltai, M. Kronbichler, M. Maier, J.-P. Pelteret, B. Turcksin, and D. Wells (2017), The `deal.II` library, version 8.5, *Journal of Numerical Mathematics*, doi:10.1515/jnma-2016-1045.

Balay, S., S. Abhyankar, M. F. Adams, J. Brown, P. Brune, K. Buschelman, L. Dalcin, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, D. A. May, L. C. McInnes, R. T. Mills, T. Munson, K. Rupp, P. Sanan, B. F. Smith, S. Zampini, H. Zhang, and H. Zhang (2018), PETSc users manual, *Tech. Rep. ANL-95/11 - Revision 3.9*, Argonne National Laboratory.

Bangerth, W., and R. Rannacher (2003), *Adaptive Finite Element Methods for Differential Equations*, Birkhäuser Verlag.

Bangerth, W., R. Hartmann, and G. Kanschat (2007), deal.II – a general purpose object oriented finite element library, *ACM Trans. Math. Softw.*, *33*(4), 24.

Bangerth, W., C. Burstedde, T. Heister, and M. Kronbichler (2011), Algorithms and data structures for massively parallel generic adaptive finite element codes, *ACM Trans. Math. Softw.*, *38*(2).

Bangerth, W., J. Dannberg, R. Gassmöller, T. Heister, et al. (2017a), ASPECT: Advanced Solver for Problems in Earth's ConvecTion, User Manual, doi:10.6084/m9.figshare.4865333, doi:10.6084/m9.figshare.4865333.

Bangerth, W., J. Dannberg, R. Gassmöller, T. Heister, , et al. (2017b), Aspect v1.5.0 [software], doi:http://doi.org/10.5281/zenodo.344623.

Burstedde, C. (2018), Parallel tree algorithms for AMR and non-standard data access, *ArXiv e-prints*.

Burstedde, C., L. C. Wilcox, and O. Ghattas (2011), `p4est`: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees, *SIAM J. Sci. Comput.*, *33*(3), 1103–1133, doi:10.1137/100791634.

Capodaglio, G., and E. Aulisa (2017), A particle tracking algorithm for parallel finite element applications, *Computers & Fluids*, *159*, 338–355.

Carey, G. F. (1997), *Computational Grids: Generation, Adaptation and Solution Strategies*, Taylor & Francis.

Folk, M., A. Cheng, and K. Yates (1999), HDF5: A file format and I/O library for high performance computing applications, in *Proc. ACM/IEEE Conf. Supercomputing (SC'99)*.

Gerya, T. V., and D. A. Yuen (2003), Characteristics-based marker-in-cell method with conservative finite-differences schemes for modeling geological flows with strongly variable transport properties, *Physics of the Earth and Planetary Interiors*, *140*(4), 293–318.

Harlow, F. (1962), *The particle-in-cell method for numerical solution fo problems in fluid dynamics*.

Heister, T., J. Dannberg, R. Gassmöller, and W. Bangerth (2017), High accuracy mantle convection simulation through modern numerical methods – II: realistic models and problems, *Geophys. J. Int.*, *210*(2), 833–851, doi:https://doi.org/10.1093/gji/ggx195.

Isaac, T., C. Burstedde, L. C. Wilcox, and O. Ghattas (2015), Recursive Algorithms for Distributed Forests of Octrees, *SIAM Journal on Scientific Computing*, *37*(5), C497–C531, doi:10.1137/140970963.

Kronbichler, M., T. Heister, and W. Bangerth (2012), High accuracy mantle convection simulation through modern numerical methods, *Geophysics Journal International*, *191*, 12–29.

Leng, W., and S. Zhong (2011), Implementation and application of adaptive mesh refinement for thermochemical mantle convection studies, *Geochemistry, Geophysics, Geosystems*, *12*(4), doi:10.1029/2010GC003425, q04006.

McNamara, A. K., and S. Zhong (2004), Thermochemical structures within a spherical mantle: Superplumes or piles?, *Journal of Geophysical Research*, *109*(B7), 1–14, doi:10.1029/2003JB002847.

Mellor-Crummey, J., D. Whalley, and K. Kennedy (2001), Improving memory hierarchy performance for irregular applications using data and computation reorderings, *Interna-*

tional Journal of Parallel Programming, *29*(3), 217–247.

Mirzadeh, M., A. Guittet, C. Burstedde, and F. Gibou (2016), Parallel level-set methods on adaptive tree-based grids, *Journal of Computational Physics*, *322*, 345–364, doi:10.1016/J.JCP.2016.06.017.

Moresi, L., F. Dufour, and H. B. Muhlhaus (2003), A Lagrangian integration point finite element method for large deformation modeling of viscoelastic geomaterials, *J. Comp. Ph.*, *184*, 476–497.

Nakagawa, T., P. J. Tackley, F. Deschamps, and J. A. Connolly (2009), Incorporating self-consistently calculated mineral physics into thermochemical mantle convection simulations in a 3-D spherical shell and its influence on seismic anomalies in Earth's mantle, *Geochemistry, Geophysics, Geosystems*, *10*(3).

Poliakov, A., and Y. Podladchikov (1992), Diapirism and topography, *Geophysical Journal International*, *109*(3), 553–564.

Popov, A. A., and S. V. Sobolev (2008), SLIM3D : A tool for three-dimensional thermo-mechanical modeling of lithospheric deformation with elasto-visco-plastic rheology, *Physics of the Earth and Planetary Interiors*, *171*, 55–75, doi:10.1016/j.pepi.2008.03.007.

Puckett, E. G., D. L. Turcotte, Y. He, H. Lokavarapu, J. M. Robey, and L. H. Kellogg (2017), New numerical approaches for modeling thermochemical convection in a compositionally stratified fluid, *Physics of the Earth and Planetary Interiors*, doi:https://doi.org/10.1016/j.pepi.2017.10.004.

Schroeder, W., K. Martin, and B. Lorensen (2006), *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*, 3rd ed., Kitware, Inc.

Seton, M., R. Müller, S. Zahirovic, C. Gaina, T. Torsvik, G. Shephard, a. Talsma, M. Gurnis, M. Turner, S. Maus, and M. Chandler (2012), Global continental and ocean basin reconstructions since 200Ma, *Earth-Science Reviews*, *113*(3-4), 212–270, doi:10.1016/j.earscirev.2012.03.002.

Steinberger, B., and A. R. Calderwood (2006), Models of large-scale viscous flow in the Earth's mantle with constraints from mineral physics and surface observations, *Geophysical Journal International*, *2*, 1461–1481, doi:10.1111/j.1365-246X.2006.03131.x.

Thielmann, M., D. A. May, and B. J. P. Kaus (2014), Discretization errors in the hybrid finite element particle-in-cell method, *Pure and Applied Geophysics*, *171*, 2165–2184.

van Keken, P. E., S. D. King, H. Schmeling, U. R. Christensen, D. Neumeister, and M.-P. Doin (1997), A comparison of methods for the modeling of thermochemical convection, *Journal of Geophysical Research: Solid Earth*, *102*(B10), 22,477–22,495, doi:10.1029/97JB01353.

Wallstedt, P., and J. Guilkey (2010), A weighted least squares particle-in-cell method for solid mechanics, *International Journal for Numerical Methods in Engineering*, *85*(13), 1687–1704.

Wang, H., R. Agrusta, and J. Hunen (2015), Advantages of a conservative velocity interpolation (cvi) scheme for particle-in-cell methods with application in geodynamic modeling, *Geochemistry, Geophysics, Geosystems*, *16*(6).

Figure 1.

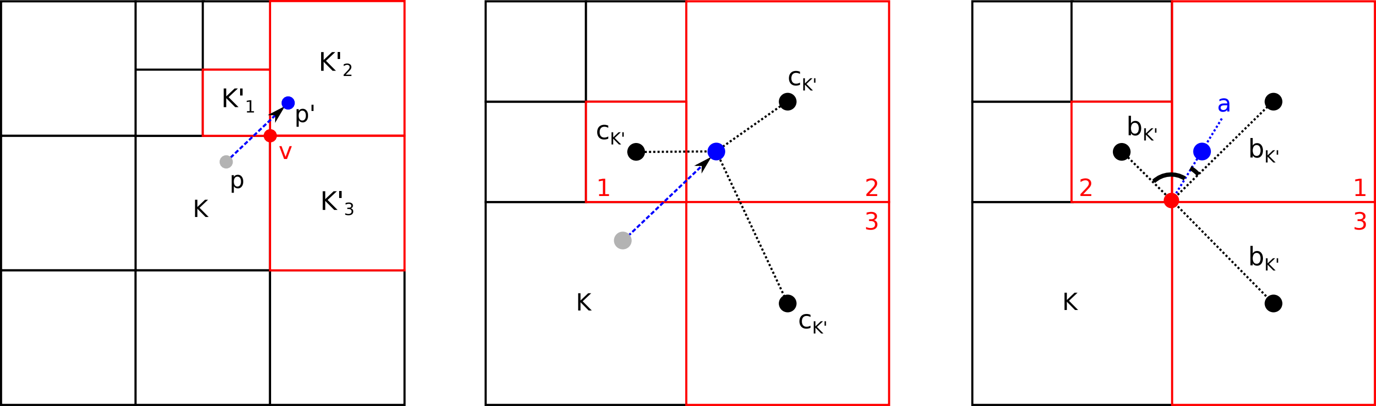
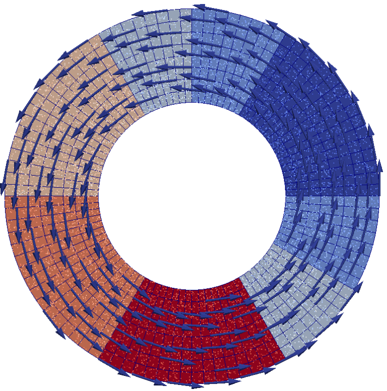
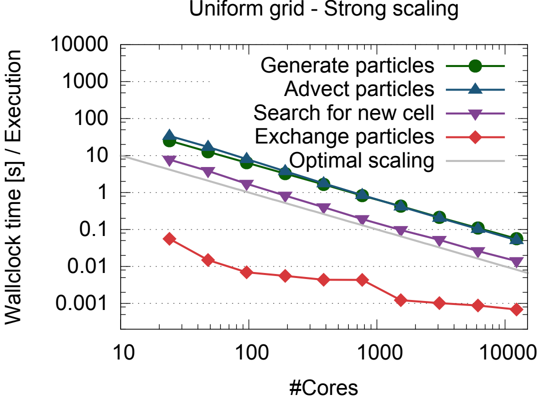
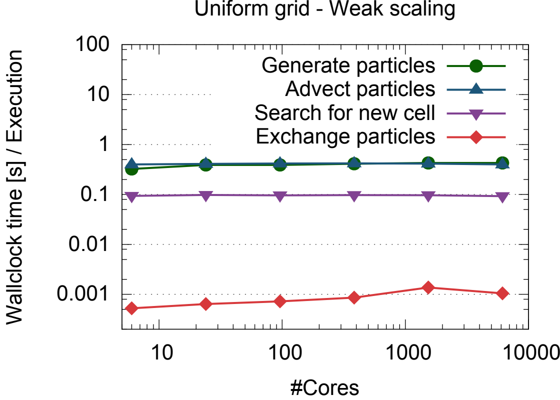
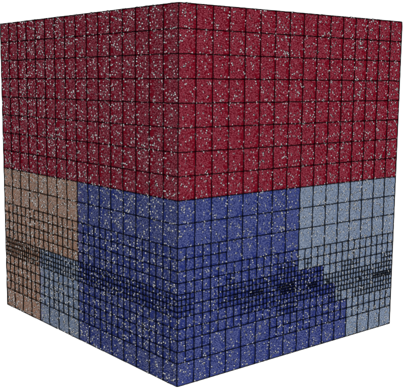

K'₂
K'₁
p'
v
p
K
K'₃
c_K'
c_K'
1
2
3
K
c_K'
a
b_K'
b_K'
2
1
3
b_K'
K

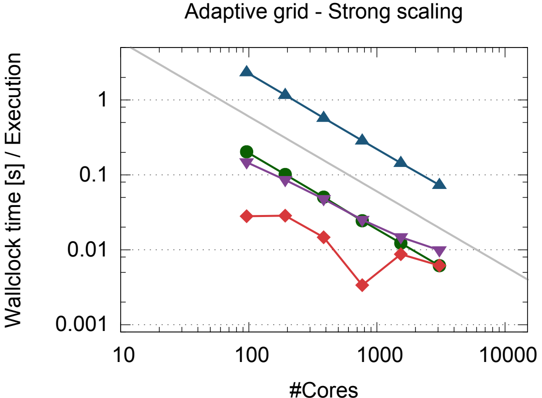
Figure 2.




Uniform grid - Strong scaling
Wallclock time [s] / Execution
10000
1000
100
10
1
0.1
0.01
0.001
10
100
1000
10000
#Cores
Generate particles
Advect particles
Search for new cell
Exchange particles
Optimal scaling


Uniform grid - Weak scaling
Wallclock time [s] / Execution
100
10
1
0.1
0.01
0.001
10
100
1000
10000
#Cores
Generate particles
Advect particles
Search for new cell
Exchange particles




Adaptive grid - Strong scaling
Wallclock time [s] / Execution
1
0.1
0.01
0.001
10
100
1000
10000
#Cores

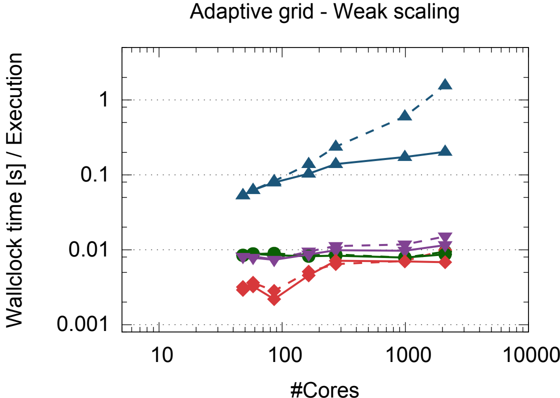
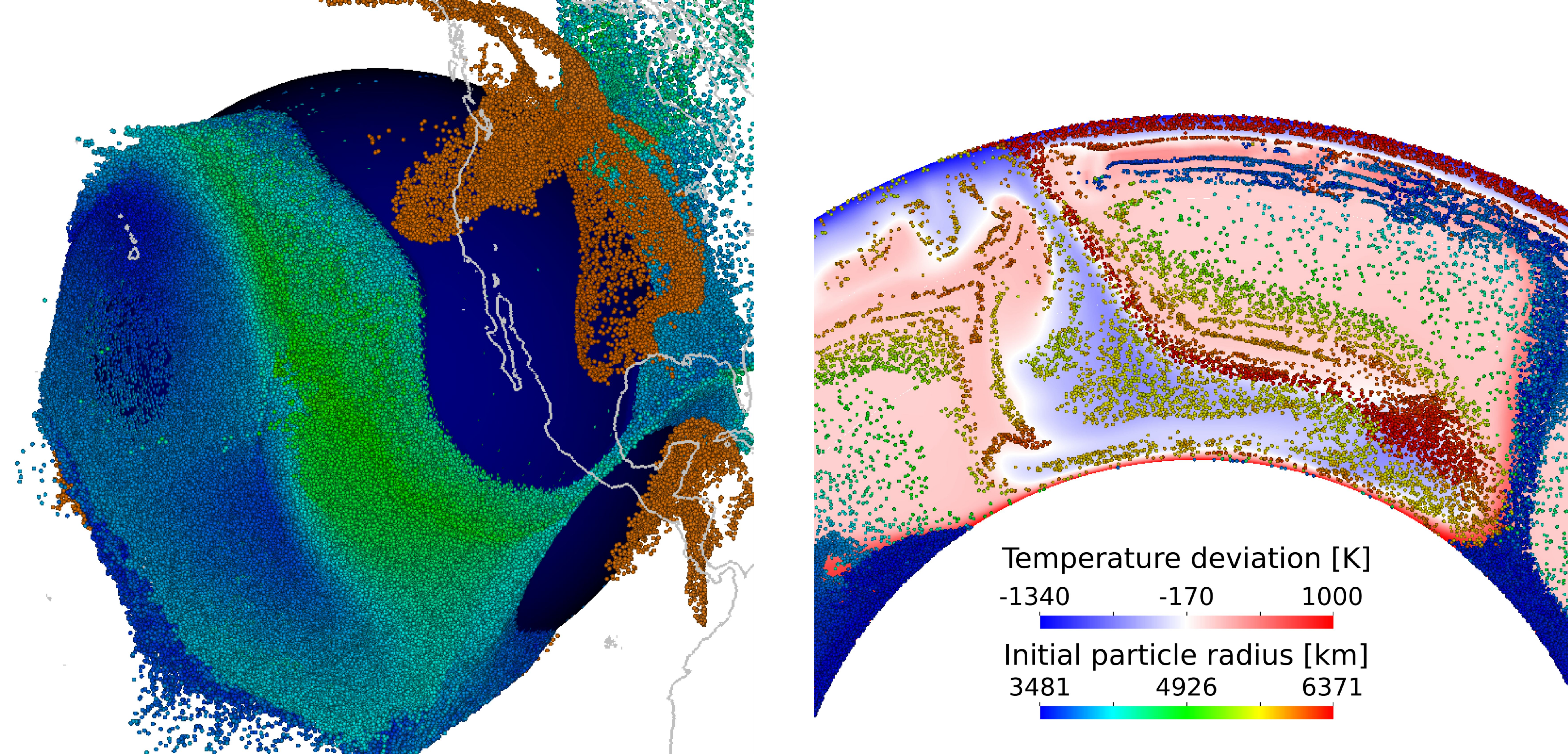

Adaptive grid - Weak scaling
Wallclock time [s] / Execution
1
0.1
0.01
0.001
10
100
1000
10000
#Cores

Figure 3.


Temperature deviation [K]
-1340
-170
1000
Initial particle radius [km]
3481
4926
6371