A practical partitioner for distributed simulations on sparse dynamic domains using optimal transport

JOEL WRETBORN, Wētā FX and University of Waterloo, Sweden
MARCUS SCHOO, Wētā FX, New Zealand
NOH-HOON LEE, Wētā FX, South Korea
CHRISTOPHER BATTY, University of Waterloo, Canada
ALEXEY STOMAKHIN, Wētā FX, USA

This work addresses the challenges of distributing large physics-based simulations often encountered in the visual effects industry. These simulations, based on partial differential equations, model complex phenomena such as free surface liquids, flames, and explosions, and are characterized by domains whose shapes and topologies evolve rapidly. In this context, we propose a novel partitioning algorithm employing *optimal transport*—which produces a power diagram—and designed to handle a vast variety of simulation domain shapes undergoing rapid changes over time. Our *Power partitioner* ensures an even distribution of computational tasks, reduces inter-node data exchange, and maintains temporal consistency, all while being intuitive and artist-friendly. To quantify partitioning quality we introduce two metrics, the surface index and the temporal consistency index, which we leverage in a range of comparisons on real-world film production data, showing that our method outperforms the state of the art in a majority of cases.

CCS Concepts: • **Computing methodologies** → **Distributed computing methodologies**; **Computer graphics**.

Additional Key Words and Phrases: partitioning, load balancing, power diagrams, optimal transport, sparse dynamic domains

ACM Reference Format:
Joel Wretborn, Marcus Schoo, Noh-hoon Lee, Christopher Batty, and Alexey Stomakhin. 2026. A practical partitioner for distributed simulations on sparse dynamic domains using optimal transport. *ACM Trans. Graph.* 45, 2, Article 20 (January 2026), 14 pages. https://doi.org/10.1145/3787521

1 Introduction

In the context of distributed simulation, the role of a partitioner is to allocate the computational workload across multiple machines (a.k.a. nodes or *ranks*) to maximize performance. Since the exact computation and communication costs are rarely known up front, the *optimal* allocation is typically defined by a set of heuristic objectives, such as ensuring an equitable distribution of computational tasks to prevent overburdening individual nodes (a.k.a. *load balancing*) and reducing the communication overhead required for inter-rank data exchange. The inherent conflicts among these goals—where enhancing one may detract from another—pose significant challenges.

Fig. 1. *Fighting kids.* For a scene from *Avatar: The Way of Water* (top) a sparse particle-in-cell splashing water simulation with surface tension and viscosity (bottom) [Stomakhin et al. 2023] and peak total memory consumption of ~500GB has been split into 8 ranks (indicated by colors) using our Power partitioner. Top image from AVATAR: WAY OF WATER ©2022 20th Century Studios, Inc. All rights reserved. Bottom image © Wētā FX Ltd.

Our work focuses on simulation applications tailored for the visual effects (VFX) industry. These simulations are grounded in physics, utilizing partial differential equations (PDEs) to model intricate phenomena, such as liquids or smoke. They are typically discretized using Cartesian grids or hybrid particle-in-cell methods. The focus is placed primarily on the aesthetic quality and visual richness of the output, capable of captivating audiences in film, gaming, or other media, rather than raw computational accuracy. Consequently, practitioners tend to employ computationally less heavy lower-order methods compared to those used in traditional engineering applications. As simulations scale to higher resolutions, practitioners reach for distributed techniques when a simulation no longer fits in working memory (typically ~256–512 GB).

Since the defining feature of VFX simulations is the emphasis on visually striking and dynamic outputs, the simulation domains

Authors' Contact Information: Joel Wretborn, joel@wbn.se, Wētā FX and University of Waterloo, Stockholm, Sweden; Marcus Schoo, mschoo@wetafx.co.nz, Wētā FX, Queenstown, New Zealand; Noh-hoon Lee, nlee@wetafx.co.nz, Wētā FX, Seoul, South Korea; Christopher Batty, christopher.batty@uwaterloo.ca, University of Waterloo, Waterloo, Canada; Alexey Stomakhin, st.alexey@gmail.com, Wētā FX, Mililani, USA.

ACM 1557-7368/2026/1-ART20
https://doi.org/10.1145/3787521

tend to exhibit highly dynamic behavior, characterized by sparsely distributed topologies that evolve rapidly over time. Such rapid evolution places additional importance on minimizing the communication costs specifically associated with restructuring the domain from one frame to the next, as opposed to the costs of ghost transfers of adjacent neighbor data in a statically partitioned domain.

Below we give a concise summary of the partitioner requirements we identified to meet the unique demands of our application.

Equitable work distribution. The workload assigned to each rank should be as uniform as possible. Since accurately predicting the actual computational effort up front is not straightforward, many approaches rely on approximate or proxy metrics, such as the number of grid cells (a.k.a. *voxels*) or particles assigned to a given rank, which we adopt in the current work.

Minimized ghost transfers. In PDE-based simulations, inter-rank ghost data transfers are governed by spatial locality: transfers occur at the border layers, where data regions from different ranks meet spatially. Minimizing the border size reduces the amount of data exchanged, lowering overhead and improving efficiency.

Temporal coherence. A critical consideration in the VFX context is the need for temporal coherence, which entails minimizing substantial shifts in the distribution of computational work between consecutive simulation frames, thereby curtailing the volume of inter-rank data movement.

Artist friendliness. The partitioner must prioritize simplicity, transparency, and reliability, ensuring that artists in the visual effects industry can focus on creativity rather than technical adjustments. It should operate seamlessly out of the box, requiring no manual tuning or deep understanding of distributed systems.

In order to understand the shortcomings of current state-of-the-art partitioning algorithms for sparse dynamic domains, to be discussed in §3, we first present in §2 the mathematical underpinnings of our domain structure. To tackle the outlined challenges we propose a novel partitioning algorithm based on regularized optimal transport. It maps *buckets* (≡ units of computational work, see §2) to ranks, modeled as points in the simulation world space, by minimizing the total transportation distance of work. The resulting partitioning turns out to be a *power diagram* with rank positions as sites, and in combination with Lloyd's algorithm [Lloyd 1982] we can ensure a balanced distribution and minimal neighbor transfers while maintaining temporal coherence without any user intervention. We introduce this *Power partitioner* in §4, which we compare to existing methods in §5 and §6.

2 Sparse domains and partitioning

Sparse grids are ubiquitous in computer graphics and VFX applications [Bojsen-Hansen et al. 2021; Bridson 2003; Museth 2013; Setaluri et al. 2014]. Instead of allocating a full tensor-product (or dense) grid, sparse grids employ a hierarchical structure that selectively includes grid points based on their proximity to the region of interest, drastically reducing the memory required while maintaining sufficient spatial coverage. A popular implementation choice, especially for uniform grids, is a sparse-block structure [Lesser et al. 2022; McAdams et al. 2011; Wang et al. 2005; Zhu et al. 2010], where the grid domain is represented as an array of axis-aligned non-overlapping dense cubic blocks of voxels (see Fig. 2), which we adopt

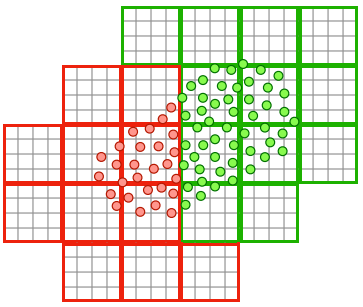

Fig. 2. *Sparse grid domain.* An illustration of a 2D sparse grid simulation domain, composed of blocks / buckets (large squares), each of which contains 4×4 voxels (small squares) and, in this example, some particles (small circles). The coloring depicts a partitioning of the buckets into 2 ranks. © Wētā FX Ltd.

in this work. In addition to representing a unit of minimal spatial coverage, such blocks simultaneously double as the units of parallel work. Smaller blocks allow for precise coverage of detailed regions of interest and enhanced parallelism, but handling the resulting large number of blocks can become cumbersome. Conversely, larger blocks simplify management by reducing their total count, though they may not exactly conform to the simulation domain causing unnecessary padding, and are also harder to partition evenly. The literature suggests that the optimal block size lies between $4 \times 4 \times 4$ and $8 \times 8 \times 8$ grid voxels [Lesser et al. 2022; McAdams et al. 2011; Zhu et al. 2010].

Below we give formal definitions related to our sparse domain representation. We will assume that our simulations happen in 3D space, and will set $d = 3$ unless stated otherwise.

Definition 2.1. A *region of interest* is a time-varying subset $\Omega_t \subset \mathbb{R}^d$, with $t \in \mathbb{R}$ representing time, where we wish to track the evolution of some (discretized) functions representing visual, physical, or geometric features, such as smoke density, temperature, liquid signed distance fields, particles, and/or other properties.

Definition 2.2. A *bucket* $\boldsymbol{b}$ is a half-open, cubic, axis-aligned region of space $\mathbb{R}^d$, which will represent a block in our sparse-block grid structure. The buckets are assumed to be chosen such that the *set of all buckets* B gives an overlap-free tiling of the whole of $\mathbb{R}^d$.

Definition 2.3. A *bucketization* is a function $\mathcal{B} : \mathbb{R}^d \to B$ which maps any point $\boldsymbol{x} \in \mathbb{R}^d$ to the bucket $\boldsymbol{b} \in B$ that contains it.

Definition 2.4. A *neighborhood* function $\mathcal{N} : B \to 2^B$ defines the list of neighboring buckets[1], for any bucket $\boldsymbol{b} \in B$. In general the neighborhood concept does not necessarily imply proximity in the physical space, but rather describes the interaction pattern within the simulated system at hand, which may become rather complicated for long range forces. In the case of PDEs, however, the neighborhood corresponds to immediately adjacent buckets.

Definition 2.5. A *domain description* $(\mathcal{B}, \mathcal{N})$ is a pairing of a bucketization and a neighborhood function.

Definition 2.6. A *uniform grid domain description* is defined by choosing a bucket size $\Delta > 0$ and setting

$$\mathcal{B}_\Delta(\boldsymbol{x}) = \left\lfloor \frac{\boldsymbol{x}}{\Delta} \right\rfloor, \tag{1}$$

which is to say that the set B is composed of cubic voxels of a uniform grid with spacing Δ. The neighborhood function considers

[1] 2^B is the set of all subsets of B.

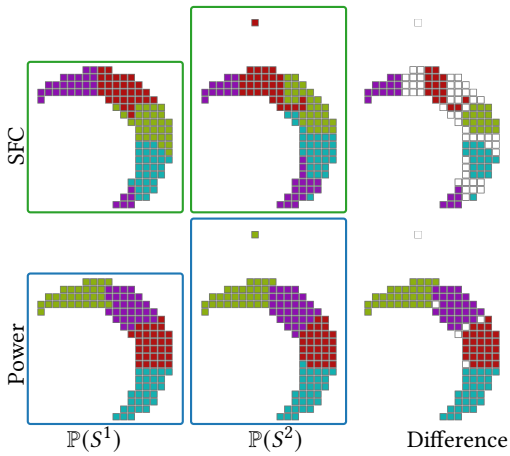

Fig. 3. *Partitioning consistency.* Left column: Visualizations of SFC and Power partitionings on a crescent-shaped domain S^1 for 4 ranks. Middle column: A single new bucket $\boldsymbol{b}$ is added, yielding $S^2 = S^1 \cup \boldsymbol{b}$, and the domain is re-partitioned. Right column: The difference between $\mathbb{P}(S^1)$ and $\mathbb{P}(S^2)$ is visualized by marking buckets that changed ranks white. The significant shift in the SFC partitioning is due to the change in the domain bounds. © Wētā FX Ltd.

two buckets to be neighbors if the Euclidean distance between them is zero, meaning their closures share a face, an edge, or a corner.

Definition 2.7. A *bucket subset* is a (finite) collection $S \in 2^B$. The simulation domain, which is the area where the volumetric data is allocated for computation, is specified via a bucket subset, which we will refer to as the *(simulation) domain subset* $D_t = \{\cup_{\boldsymbol{x}\in\Omega_t} \mathcal{B}(\boldsymbol{x})\}$.

Definition 2.8. A *padding* function $\mathcal{T} : 2^B \to 2^B$ is defined as

$$\mathcal{T}(S) = \{\boldsymbol{b} \cup \mathcal{N}(\boldsymbol{b}), \boldsymbol{b} \in S\}, \tag{2}$$

and dilates a given domain subset by a single layer of buckets.

Definition 2.9. A *partitioning* is a function that maps buckets of a given bucket subset S to ranks, $\mathcal{P} : S \to [R]$, where $[R] \equiv \{0 \dots R-1\}$ and R is the total number of ranks. The *partition* corresponding to rank r is the subset of buckets that has been assigned to that rank, specifically $P_r = \{\boldsymbol{b} \in S : \mathcal{P}(\boldsymbol{b}) = r\}$. If not otherwise stated we will assume that a partitioning is computed on the domain subset, i.e., $S = D_t$. We refer to the algorithm used to produce a partitioning $\mathcal{P}$ of a bucket subset S as a *partitioner* $\mathbb{P}$, so that $\mathbb{P}(S) = \mathcal{P}$.

Given a measure of the computational work $W_{\boldsymbol{b}}$ per bucket $\boldsymbol{b}$ the computational work for the partition P_r associated with rank r is $W_r = \sum_{\boldsymbol{b}\in P_r} W_{\boldsymbol{b}}$. We are specifically interested in partitioners that would (1) produce the minimal variation of work W_r across all ranks, (2) for each partition P_r, minimize the number of neighbor buckets $|\mathcal{T}(P_r) \setminus P_r \cap D_t|$ that do not belong to it, and (3) keep partitions coherent across time as D_t evolves.

Remark. The definition of the domain description in Def. 2.5 admits more general shapes (e.g., an adaptive octree) than the uniform grid we define with Eq. 1, without any required changes to the partitioner algorithm we outline in §4.

3 Existing methods

There are a number of partitioners available in the literature.

Graph-based partitioners. Graph-based partitioners model the computational domain as a graph: the vertices represent computational units—such as buckets in our sparse grid structure—while edges encode spatial dependency relationships derived from the neighborhood function $\mathcal{N}$. The partitioning problem is then cast as a graph-cut optimization task, where the goal is to minimize the number of edges connecting different ranks while balancing the computational load, represented by vertex weights $W_{\boldsymbol{b}}$. The widely adopted METIS [Karypis and Kumar 1998] and ParMETIS [Schloegel et al. 2003] employ multilevel recursive bisection or k-way partitioning strategies to achieve these aims.

For sparse and irregular domain topologies these methods produce close-to-optimal cuts. However, small changes in the domain subset often cause drastic changes in the partitioning, resulting in low temporal coherence across frames. This sensitivity arises because graph-based methods operate purely on connectivity and are largely oblivious to the domain's world-space geometry, making it difficult for them to preserve spatial consistency over time. In VFX applications, where the domain subset can change substantially between frames, our testing shows poor performance for these kinds of methods (see §5).

Space filling curve (SFC) methods. A number of methods use geometric heuristics that lead to reasonable approximations of optimal partitionings. SFCs [Borrell et al. 2020; Tsuzuki and Aoki 2016], such as the Hilbert curve or Morton (a.k.a. Z-order) curve, provide a computationally efficient way to map multi-dimensional data into a one-dimensional sequence. A popular claim is that this mapping preserves locality, which means nearby points in the original multi-dimensional space stay close together in the 1D ordering. This in turn implies that when dividing work (i.e., buckets) among ranks using this 1D ordering, the communication overhead is minimized. Alas, the starting claim above is only approximate: while small, there is a nonzero chance of encountering non-local patterns in the mapping, occurring in places where different coils of an SFC meet spatially. In practice, this leads to SFC partitioners producing visually prominent artefacts that compromise locality. Additionally, the 1D bucket ordering varies rapidly with changes in the bounding box of the domain subset, leading to poor temporal coherence properties for domains with fast-changing topologies.

To illustrate the last two points consider Fig. 3 (top-left) where a crescent-shaped domain is partitioned with a Hilbert SFC (see §5 for algorithmic details). The purple partition has buckets mainly located in the top left, but also a few scattered buckets at the bottom, demonstrating poor locality. Then, when a single bucket is introduced outside of the current domain (Fig. 3, top-middle), the domain subset bounding box changes, causing a drastic change in the partitioning. Although seemingly contrived, this case is remarkably common in VFX simulations: a water splash can send single droplets flying into the air, causing temporally incoherent partitions.

Voronoi methods. Voronoi methods (e.g., [Koradi et al. 2000]) are intuitive and powerful, aligning well with our goals of locality and

work balance. However, traditional implementations, such as standard centroidal Voronoi tessellation, face challenges in dynamic VFX simulations. Constructing explicit Voronoi meshes, as done by Fu et al. [2017b], adds computational overhead and implementation complexity as buckets must be classified into cells. Importantly, when $W_{\boldsymbol{b}}$ is spatially nonuniform, load balancing becomes increasingly inaccurate since Voronoi partitions depend only on geometry and cannot adapt cell sizes to heterogeneous work distributions. It may also not be possible to directly partition a domain of an arbitrary shape, instead requiring embedding into a convex hull, leading to even more overhead and meshes with overly stretched elements.

Our Power partitioner was heavily inspired by and is an improvement over Voronoi methods. It produces Voronoi-like structures, hence preserving all of the desired locality properties, while also being capable of equally partitioning domains of arbitrary shapes and spatially varying work functions. Conceptually, the Power partitioner occupies a middle ground between graph-based and purely geometric heuristic methods. The regularized optimal transport formulation (§4.1) can be viewed as a continuous analogue of graph partitioning, replacing the discrete edge-cut minimization with an optimization based on world-space distances between ranks and vertices. However, it does not explicitly minimize communication volume, but instead offers a practical balance between load equality, spatial compactness, and temporal coherence. Thus for domains with stable connectivity, graph-based methods can achieve lower ghost communication overhead, albeit with reduced temporal stability. Table 1 gives a qualitative summary of how SFCs (Hilbert curve), graph-based methods (METIS), and our Power partitioner score on different partitioning properties. We evaluate these partitioners more quantitatively in §5.

There exist other partitioning strategies best-suited to simulation domains that are compact, box-shaped, or essentially static. Given our stated objectives, we explicitly do not consider methods that are tailored to such limited domain shapes—for example static partitioning techniques [Irving et al. 2006; Wang et al. 2020] or axis-aligned plane-cutting techniques [Qiu et al. 2022; Surmin et al. 2015]. Although plane-cutting has been commonly used in VFX applications ([Bailey et al. 2015; Flores and Horsley 2009; Lait 2016]), it only works well when there is a "preferred" simulation direction. For highly dynamic simulations, like the fluid simulation highlighted in Fig. 1, such a direction is not always possible to specify. Additionally, plane-cutting techniques often require the simulation operator to specify up front the number of cutting planes as well as their directions, which conflicts with our previously stated goal of *artist friendliness*. Flexible recursive variants of plane-cutting (analogous to kd-trees) are classical in computational fluid dynamics [Berger and Bokhari 1987], under the names orthogonal recursive bisection or recursive coordinate bisection; however, such methods are known to produce much lower quality partitions than the modern graph-based schemes discussed earlier [Bruaset and Tveito 2005].

Another example is the speculative partitioning technique by Shah et al. [2018], who account for the temporal aspect of partitioning by predicting the future region of interest via a low-resolution simulation run concurrently with the main simulation. Since many of the features we are interested in are scale-dependent—for example surface tension effects (Fig. 1) or chemical combustion (Fig. 9)—a low-resolution simulation is seldom predictive of the actual future simulation state.

Table 1. Qualitative evaluation of properties of different partitioners.

Partitioner	Speed	Work balance	Locality	Temporal
Graph-based	★★	★★★★	★★★★	★
Hilbert curve	★★★★	★★★★	★★	★★
Power diagram	★★	★★★★	★★★	★★★★

We emphasize that our approach harmonizes well with general load-balancing frameworks, such as those of Kale and Krishnan [1993] and Pearce et al. [2012], which utilize on-the-fly self-profiling tools to further enhance the partitioning quality with real-time data; however, we consider those techniques to be complementary to the partitioner presented here and outside the scope of our work.

Optimal transport in graphics. We end this section by highlighting the increased use of optimal transport across computer graphics. One of the earliest examples is the work of de Goes et al. [2012], who showed how to use optimal transport for blue noise sampling. Since then there has been a variety of work applying optimal transport to discrete domains ([Mandad et al. 2017; Solomon 2018; Solomon et al. 2015]) and to physics simulation ([de Goes et al. 2015; Qu et al. 2022, 2023; Wretborn et al. 2025; Zhai et al. 2020]).

4 The Power partitioner

The need for a new partitioner arose when we observed SFC partitionings on very dynamic scenes (e.g., Fig. 1) or sparse non-convex domains (e.g., the "River" shape in Fig. 10), where individual rank partitions would visibly flicker across neighboring frames or exhibit severe non-locality. See also the supplementary video.

Our goal was to develop a new technique that would improve on both the locality and temporal coherence of existing methods. To that end, we introduce a novel partitioner based on regularized optimal transport and power diagrams, hence the name "Power partitioner". This approach produces geometric structures, similar to Voronoi cells, that can be used to create high quality partitionings, while also being computationally efficient. In this section we first describe the application of regularized optimal transport to our problem domain (§4.1), and then define our partitioner and discuss implementation considerations (§4.2).

4.1 Regularized power diagrams

We will leverage regularized power diagrams as our spatial heuristic of choice, which can be viewed through the lens of optimal transport. For a primer on optimal transport we recommend the review by Peyré and Cuturi [2019].

We use $\boldsymbol{x}_{\boldsymbol{b}} \in \mathbb{R}^d$ to denote the fixed reference position assigned to a given bucket $\boldsymbol{b} \in B$, and similarly associate a position $\boldsymbol{x}_r \in \mathbb{R}^d$ with each rank $r \in [R]$. In summation expressions, we will omit indicating set membership for brevity where the scope is obvious from context, e.g., $\sum_{\boldsymbol{b}}$ will imply summation over all $\boldsymbol{b} \in D_t$. We will also omit the subscript t when the time is irrelevant or clear from context. Let $\mathbf{C} : [R] \times D \to \mathbb{R}_+$ be the so-called *cost function* where each entry $\mathbf{C}_{r\boldsymbol{b}}$ is the squared distance[2] between a rank

[2]The squared 2-norm will create power cells, which are attractive for their spatial compactness, but other cost functions can be chosen [Peyré and Cuturi 2019, Sec. 5.2].

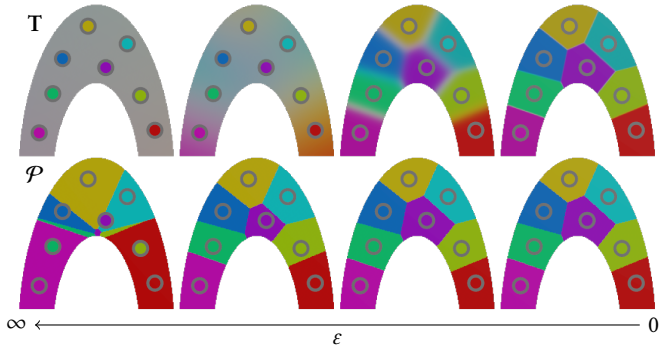

Fig. 4. *Effect of regularization on Power partitioning.* The coupling $\mathbf{T}$ (top) and resulting partitioning $\mathcal{P}$ (bottom) is visualized for varying ε on a non-convex domain. Ranks are drawn as circles with color $\boldsymbol{\gamma}_r$. In the top row, the color is computed on buckets as $\boldsymbol{\gamma}_{\boldsymbol{b}} = \left(\sum_r \boldsymbol{\gamma}_r \mathbf{T}_{r\boldsymbol{b}}\right) / \left\|\sum_r \boldsymbol{\gamma}_r \mathbf{T}_{r\boldsymbol{b}}\right\|_2$. In the bottom row the color is computed as $\boldsymbol{\gamma}_{\boldsymbol{b}} = \boldsymbol{\gamma}_{\mathcal{P}(\boldsymbol{b})}$. © Wētā FX Ltd.

and a bucket, $\mathbf{C}_{r\boldsymbol{b}} = \|\boldsymbol{x}_r - \boldsymbol{x}_{\boldsymbol{b}}\|_2^2$. We define the *desired workload* per rank as $L = \sum_{\boldsymbol{b}} W_{\boldsymbol{b}}/R$. Then there exists an *optimal coupling* $\mathbf{T} : [R] \times D \to \mathbb{R}_+$ that satisfies

$$\mathbf{T} = \operatorname*{argmin}_{\mathbf{T}} \sum_{r,\boldsymbol{b}} \mathbf{T}_{r\boldsymbol{b}} \mathbf{C}_{r\boldsymbol{b}} - \varepsilon \mathcal{H}(\mathbf{T}), \tag{3}$$

$$\text{subject to} \sum_{\boldsymbol{b}} \mathbf{T}_{r\boldsymbol{b}} = L \ \forall r, \tag{4}$$

$$\sum_{r} \mathbf{T}_{r\boldsymbol{b}} = W_{\boldsymbol{b}} \ \forall \boldsymbol{b}, \tag{5}$$

where $\varepsilon \in \mathbb{R}_+$ is a user-defined regularization parameter and $\mathcal{H}(\mathbf{T})$ is the discrete entropy, $\mathcal{H}(\mathbf{T}) = -\sum_{r,\boldsymbol{b}} \mathbf{T}_{r\boldsymbol{b}} \left(\log \mathbf{T}_{r\boldsymbol{b}} - 1\right)$.

The coupling matrix $\mathbf{T}$ represents a joint assignment of work between ranks and buckets, encoding $R \times |D|$ unknowns. These are constrained by prescribed marginal distributions: each rank must receive a total workload of L, while each bucket's entire work $W_{\boldsymbol{b}}$ must be assigned across the ranks[3]. The objective function balances the transportation cost $\sum_{r\boldsymbol{b}} \mathbf{T}_{r\boldsymbol{b}} \mathbf{C}_{r\boldsymbol{b}}$—favoring proximity between ranks and buckets—with an entropic regularization term $\varepsilon \mathcal{H}(\mathbf{T})$. This regularization induces strict convexity, ensuring the existence and uniqueness of the optimal solution $\mathbf{T}$, in contrast to unregularized optimal transport, where multiple optima may exist. This uniqueness is a desirable feature for our problem: it makes partitioning more temporally stable, because it reduces the likelihood of toggling between quite distinct but equal-quality solutions as the domain evolves.

The geometric interpretation of $\mathbf{T}$ is an approximate power diagram with rank positions as sites and diffuse edges/faces that turn progressively sharper as $\varepsilon \to 0$; see the top row in Fig. 4. Notably, ε has units of length and can be viewed as a characteristic width of the diffuse transitions between power cells. The *work center of mass* for a rank can be computed via the power kernel [Qu et al. 2022, Eq. 6] as

$$\boldsymbol{c}_r = \frac{1}{L} \sum_{\boldsymbol{b}} \mathbf{T}_{r\boldsymbol{b}} \boldsymbol{x}_{\boldsymbol{b}} \ \forall r. \tag{6}$$

[3]We temporarily assume here that $W_{\boldsymbol{b}}$ (i.e., one bucket's work) may be split among multiple ranks.

The specific choice of the entropic term given in Eq. 3 implies that the solution takes the form [Peyré and Cuturi 2019, Prop. 4.3]

$$\mathbf{T}_{r\boldsymbol{b}} = q_r e^{-\mathbf{C}_{r\boldsymbol{b}}/\varepsilon} q_{\boldsymbol{b}} \ \forall r, \boldsymbol{b}, \tag{7}$$

for unknown variables q_r and $q_{\boldsymbol{b}}$, defined for each rank and bucket separately. This reduces the number of unknowns to $R+|D|$, making computing a solution computationally tractable.

4.1.1 Sinkhorn iterations. Equation 7 allows for an efficient solution procedure to find $\mathbf{T}$ by performing so-called *Sinkhorn iterations*. Let $s = 0, 1, \ldots$ be an incrementing index denoting the current iteration. We perform interleaved projections by substituting Eq. 7 into the load and work constraints Eqs. 4-5 as

$$q_r^{s+1} = \frac{L}{\sum_{\boldsymbol{b}} e^{-\mathbf{C}_{r\boldsymbol{b}}/\varepsilon} q_{\boldsymbol{b}}^s} \ \forall r, \tag{8}$$

$$q_{\boldsymbol{b}}^{s+1} = \frac{W_{\boldsymbol{b}}}{\sum_r e^{-\mathbf{C}_{r\boldsymbol{b}}/\varepsilon} q_r^{s+1}} \ \forall \boldsymbol{b}. \tag{9}$$

Equations 8-9 can each be viewed as one matrix-vector multiplication and one vector-vector componentwise division, and thus a single Sinkhorn iteration has complexity $O(R|D|)$. The iterations will converge linearly, and as stopping criterion we use the largest relative deviation from Eq. 4,

$$\max_r \left| \frac{\sum_{\boldsymbol{b}} \mathbf{T}_{r\boldsymbol{b}}^s}{L} - 1 \right| < \tau_{\text{sinkhorn}}, \tag{10}$$

for some user defined threshold τ_{sinkhorn}. We construct $\mathbf{T}^s$ by evaluating Eq. 7 using q_r^s and $q_{\boldsymbol{b}}^s$. Although $\mathbf{C}$ can be computed on the fly from the rank and bucket positions, we found it beneficial to precompute $e^{-\mathbf{C}/\varepsilon}$ and store it in a $R \times |D|$ matrix.

4.1.2 Log-domain Sinkhorn iterations. Smaller regularization parameters ε produce sharper power diagrams but lead to poor convergence [Peyré and Cuturi 2019, Remark 4.7]; in particular, the calculation $\mathbf{C}/\varepsilon$ may overflow numerically. We address this by detecting when overflow is likely to occur (§4.2.1), in which case we instead use the following log-domain computations,

$$p_r^{s+1} = p_r^s + \varepsilon \log L + \operatorname*{smoothmin}_{\boldsymbol{b}} \left(\mathbf{C}_{r\boldsymbol{b}} - p_r^s - p_{\boldsymbol{b}}^s, \varepsilon \right) \ \forall r, \tag{11}$$

$$p_{\boldsymbol{b}}^{s+1} = p_{\boldsymbol{b}}^s + \varepsilon \log W_{\boldsymbol{b}} + \operatorname*{smoothmin}_{r} \left(\mathbf{C}_{r\boldsymbol{b}} - p_{\boldsymbol{b}}^s - p_r^{s+1}, \varepsilon \right) \ \forall \boldsymbol{b}, \tag{12}$$

for the log-domain Sinkhorn variables $p_\star = \varepsilon \log q_\star, \star \in \{r, \boldsymbol{b}\}$ and smooth minimum function

$$\operatorname*{smoothmin}_{\star}(\boldsymbol{y}_\star, \varepsilon) = -\varepsilon \log \sum_{\star} e^{-\boldsymbol{y}_\star/\varepsilon}. \tag{13}$$

Equations 11-12 are the log-domain equivalents of Eqs. 8-9 and are stable for arbitrary $\varepsilon > 0$ given a good initial guess [Peyré and Cuturi 2019, Remark 4.23]; we use $p_r^0 = 0$ and $p_{\boldsymbol{b}}^0 = \min_r \mathbf{C}_{r\boldsymbol{b}}$ which guarantee that the iterations do not overflow. The coupling can be reconstructed via

$$\mathbf{T}_{r\boldsymbol{b}} = e^{\frac{p_r + p_{\boldsymbol{b}} - \mathbf{C}_{r\boldsymbol{b}}}{\varepsilon}} \ \forall r, \boldsymbol{b}, \tag{14}$$

which is the log-domain equivalent of Eq. 7.

Algorithm 1 Regularized Power partitioner.

Input: $W_{\boldsymbol{b}}, \boldsymbol{x}_{\boldsymbol{b}}, \boldsymbol{x}_r$ (optionally coarsen; §4.2.4, §4.2.5)
Output: $\mathcal{P}, \boldsymbol{x}_r$

$L \leftarrow \sum_{\boldsymbol{b}} W_{\boldsymbol{b}}/R$
for Lloyd iteration $l = 1, 2, \ldots$ **do**
  $\mathbf{C}^l_{r\boldsymbol{b}} \leftarrow \|\boldsymbol{x}_r - \boldsymbol{x}_{\boldsymbol{b}}\|_2^2$ (compute cost function; §4.1)
  Compute ε^l (§4.2.2)
  Compute $\mathbf{T}^l$ (Sinkhorn iterations; §4.1, §4.2.1)
  $\mathcal{P} \leftarrow \operatorname{argmax}_r \mathbf{T}^l_{r\boldsymbol{b}}$ (compute partition, Eq. 15)
  $\boldsymbol{x}_r \leftarrow \boldsymbol{c}_r$ (update rank centroids, Eq. 6)
  if $\max_r \lambda_{\mathcal{P}}(r) < \tau_{\text{load}}$ **then** (§4.2.3)
    break
  end if
end for

4.2 A partitioner from regularized power diagrams

Equipped with the optimal coupling $\mathbf{T}$ (Eqs. 3-5) we define the partitioning as

$$\mathcal{P}(\boldsymbol{b}) = \operatorname*{argmax}_r \mathbf{T}_{r\boldsymbol{b}}. \tag{15}$$

Equation 15 assigns each bucket to the single rank with which it is coupled most, and since $\mathcal{P}$ is a function on D it is a valid partitioning according to Def. 2.9. However, Eq. 15 converts the optimal coupling from a continuous to a discrete function, and as such the constraints Eqs. 4-5 may no longer be satisfied. Let $\mathbf{T}^{\mathcal{P}} : [R] \times D \to \mathbb{R}_+$ be the function that evaluates to $W_{\boldsymbol{b}}$ if $r = \mathcal{P}(\boldsymbol{b})$ and 0 otherwise, and consider each equation with respect to $\mathbf{T}^{\mathcal{P}}$.

By construction $\sum_r \mathbf{T}^{\mathcal{P}}_{r\boldsymbol{b}} = W_{\boldsymbol{b}}$, since each bucket is assigned to exactly one rank; thus $\mathbf{T}^{\mathcal{P}}$ satisfies Eq. 5. The rank load equality constraint, Eq. 4, is in general not satisfied, $\sum_{\boldsymbol{b}} \mathbf{T}^{\mathcal{P}}_{r\boldsymbol{b}} \neq L$, because the "rounding" induced by Eq. 15 can cause significant deviations of $\mathbf{T}^{\mathcal{P}}$ from $\mathbf{T}$. This is clearly true for sufficiently large values of ε, as in the limit $\varepsilon \to \infty$ the optimal coupling $\mathbf{T}$ is the maximal entropy solution where every bucket is equally coupled to every rank. In practice the errors become significant when the ratio $\mathbf{C}/\varepsilon$ gets small. Thus, we prefer partitionings where ranks are located close to the center of their respective power cell, and a small regularization parameter. For the former, we adopt Lloyd's algorithm [Lloyd 1982] and iteratively move rank sites to the work center of mass (Eq. 6) in an outer loop. The resulting full partitioning algorithm is outlined in Alg. 1. The effect of the regularization parameter on the partitioning of Eq. 15 is visualized in the bottom row of Fig. 4.

4.2.1 Check for overflow. Numerical overflows in $\mathbf{C}/\varepsilon$ are problematic when—due to the exponentiation in Eq. 7—they cause *all* $\mathbf{T}_{r\boldsymbol{b}} = 0$ for a particular bucket $\boldsymbol{b}$, rendering Eq. 15 meaningless. The element in $\mathbf{C}$ that is most likely to induce such scenarios is determined by the bucket that is furthest away from its closest rank, since such a bucket is comparatively far from *all* ranks. We denote that value

$$\Gamma(\mathbf{C}) = \max_{\boldsymbol{b}} \left(\min_r \mathbf{C}_{r\boldsymbol{b}} \right). \tag{16}$$

Numerical issues are likely if $e^{-\Gamma(\mathbf{C})/\varepsilon}$ is below the precision of the floating point type. We use 64-bit precision for all of our operations and fall back to log-domain Sinkhorn iterations if $e^{-\Gamma(\mathbf{C})/\varepsilon} < 10^{-12}$ (Eqs. 11-12, 14), otherwise we use standard Sinkhorn iterations (Eqs. 7-9).

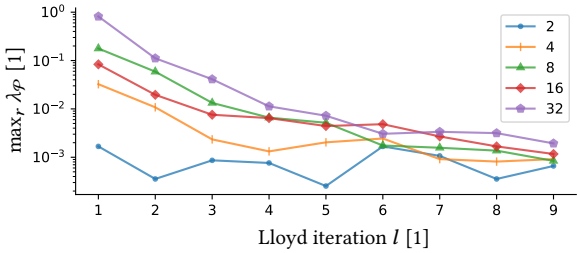

Fig. 5. *Power partitioner convergence plot.* We study the convergence of the Lloyd iteration loop in Alg. 1. The domain subset is a box with $\lfloor \sqrt[3]{10^4 R} \rfloor$ buckets on all sides to keep the size of each partition P_r roughly equal. We set $\tau_{\text{load}} = 0$ to ensure the break in Alg. 1 is never triggered and we vary the number of ranks $R = \{2, 4, 8, 16, 32\}$. © Wētā FX Ltd.

Regular Sinkhorn iterations are significantly faster to compute than their log-domain counterparts due to the per-component exponentiation in Eq. 13, so keeping the above threshold small is beneficial. We have chosen the threshold value 10^{-12} conservatively and have not experienced convergence issues in any of our testing, indicating that it may be possible to reduce the threshold further.

4.2.2 Choosing the regularization parameter. As seen previously, large values of the regularization parameter ε tend to produce suboptimal load balancing, while lower ones may lead to numerical overflows and reduced performance of the Sinkhorn algorithm. To work around these shortcomings, we propose to gradually decrease ε with Lloyd iterations. Specifically, we define the regularization parameter ε^l for the first Lloyd iteration $l = 1$ in terms of Γ (Eq. 16), and then successively lower it by a fixed factor $\eta \in (0, 1)$ so that $\varepsilon^1 = \Gamma(\mathbf{C}^1)/E$ and $\varepsilon^l = \eta\varepsilon^{l-1}$ for some user-defined parameter $E \in \mathbb{R}_+$. This way we start by making crude estimates of the rank positions, moving them into the ballpark of the solution quickly, and then refine them to achieve better load balancing with more precise iterations at lower values of ε. We found $E = 10$ and $\eta = 2/3$ to work well in practice, and used those values for all of our examples.

4.2.3 Partitioner convergence criteria. We stop the Lloyd iteration loop when a target load index τ_{load} has been reached (see §5.1 for an explanation of our partitioning metrics). All our examples use $\tau_{\text{load}} = 0.01$, which typically takes 1–3 Lloyd iterations to reach. We test the convergence to equal load between ranks of our method in Fig. 5.

4.2.4 Initialization. We estimate the computational work $W_{\boldsymbol{b}}$ following Qiu et al. [2022]. For hybrid particle-in-cell simulations (Figs. 1 and 6; all other examples are purely volumetric), we set $W_{\boldsymbol{b}}$ equal to the number of particles in the bucket; for volumetric simulations, we set a constant workload $W_{\boldsymbol{b}} = 1$ for all buckets.

Bucket positions $\boldsymbol{x}_{\boldsymbol{b}}$ are determined by picking a random (but temporally constant) position within the extent of the bucket. For domains where the optimal coupling $\mathbf{T}$ produces (close to) axis-aligned power cells for a given rank r, we have found this strategy beneficial, as opposed to, e.g., choosing $\boldsymbol{x}_{\boldsymbol{b}}$ to be the center of each bucket. Using bucket centers, all buckets $\boldsymbol{b}$ along the border of such

Algorithm 2 Partitioner timestep overview.

Input: $D_t^0, \mathbb{P}$
Output: $D_{t+\Delta t}^0$

$\mathcal{P}_t \leftarrow \mathbb{P}(D_t^0)$ (partition domain; Alg. 1)
for algorithm **in** {advection, emission, rasterization, …} **do**
  $D_t^{i+1} \leftarrow \text{algorithm}(D_t^i)$
  Extend $\mathcal{P}_t$ via Eq. 17 (assign nearest rank; §4.2.6)
end for
$D_{t+\Delta t}^0 \leftarrow D_t^{\text{last}}$

a power cell r would have $\mathbf{T}_{r\boldsymbol{b}}$ roughly equal to each other, making it difficult for individual buckets to change rank and thus causing poor convergence of Alg. 1. The added positional randomness helps avoid this kind of degeneracies.

Rank positions $\boldsymbol{x}_r$ are initialized on the first simulation frame by picking a random $\boldsymbol{x}_{\boldsymbol{b}}$ in the domain subset. For subsequent frames, we "warm-start" the Lloyd iterations by using the rank positions of the previous timestep, which facilitates temporal coherence across timesteps.

4.2.5 Coarsening for efficiency. Partitioning domains with a large number of buckets can be time consuming, and it is often possible to reduce the problem complexity by coarsening D without adversely affecting the partitioning quality. We present one such coarsening procedure here, by introducing a coarser uniform grid (Eq. 1), $\mathcal{B}_{\Delta_{\text{coarse}}}(\boldsymbol{x})$, for $\Delta_{\text{coarse}} = \Delta \cdot k$. Here, k (typically 2 or 3) merges k^3 fine buckets into one coarse unit, with k chosen to target a manageable bucket count. For example, we target $|D^{\text{coarse}}| \approx 64\,000$ for the coarsened domain in Fig. 10. For each coarse bucket, we sum the work from its fine constituents and average their positions, and store these as new sets $\{W_{\boldsymbol{b}}^{\text{coarse}}\}$ and $\{\boldsymbol{x}_{\boldsymbol{b}}^{\text{coarse}}\}$. The coarsened sets are used as inputs to Alg. 1, and the resulting partitioning is mapped back to the original bucket structure.

4.2.6 Managing dynamic domains. Our partitioner is intended to be run once per timestep. In practice, however, the domain subset may also change at any point during the computation of a single timestep. When and how depends on the particular simulation type and scenario; for our fluid simulator changes may happen during algorithms for emission, advection, or rasterization of boundaries.

Consider a domain subset D_t^0 and associated partitioning $\mathcal{P}_t = \mathbb{P}(D_t^0)$ for a particular simulation time t, where the superscript ^{0} indicates that D_t^0 is the *initial* domain subset for that time t. An algorithm may then change the domain subset by removing or adding buckets, creating D_t^1, D_t^2, and so on. It is too costly to fully re-partition every time a change in the domain happens; instead, assuming such incremental changes are modest, we take a heuristic approach and assign any new bucket to the rank to which it is closest,

$$\mathcal{P}_t(\boldsymbol{b}) = \operatorname*{argmin}_r \|\boldsymbol{x}_r - \boldsymbol{x}_{\boldsymbol{b}}\|_2, \text{ for } \boldsymbol{b} \notin D_t^0. \tag{17}$$

Removing buckets from D_t^0 can be done without any change to the partitioning, since a partitioning on D_t^0 is also a valid partitioning on any subset $S \subset D_t^0$. At the next simulation time $t+\Delta t$ we initialize the domain subset with the last domain subset from the previous time, $D_{t+\Delta t}^0 = D_t^{\text{last}}$, and compute a new partitioning $\mathcal{P}_{t+\Delta t} = \mathbb{P}(D_{t+\Delta t}^0)$. An overview of this procedure is shown in Alg. 2.

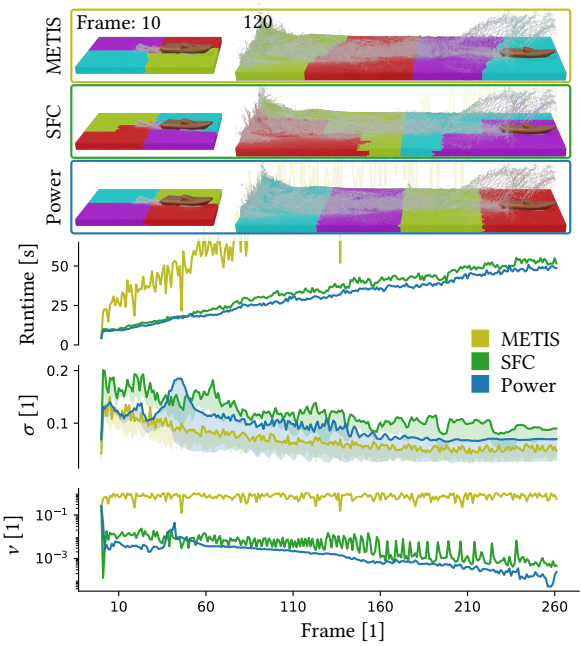

Fig. 6. *Boat wake.* A hybrid particle-in-cell fluid simulation around a moving boat is run on 4 machines, the domain expanding as the simulation progresses. Simulated particles are colored according to their rank for the different partitioners; on the last frame there are 169M particles in total (top). Runtime: METIS was more than 3× slower than Power due its temporal incoherence and the plot is clipped by the partitioning visualizations. Surface index: $\max_r \sigma$ is drawn as a solid line. The shaded colored region represents $[\min_r \sigma, \max_r \sigma]$. © Wētā FX Ltd.

5 Evaluation

The end goal in designing a good partitioner is to reduce the runtimes for a certain class of simulations. Performance losses may come from multiple sources, such as sub-optimal locality, poor temporal coherency, or the cost of re-partitioning. In addition, the quality of partitionings derived from spatial heuristics (like SFC and our Power partitioner) depends on the geometric shape of the domain, emphasizing the need to evaluate partitioners on domains representative of the application. Consequently, we introduce a number of partitioning metrics in §5.1 which we evaluate and relate to runtimes on a few simulation scenarios typical of VFX in §5.3. Then, to ensure the domain shapes were not cherry-picked, we also perform comparisons of the metrics with different partitioners on a large corpus of pre-computed fluid simulation cache sequences.

5.1 Metrics

To evaluate the effectiveness of partitioners in satisfying our goals, we define several quantitative metrics.

Load index. Following the equitable work objective, we would like each rank to receive $1/R$ of the total workload $\sum_{\boldsymbol{b}} W_{\boldsymbol{b}}$, which we have previously introduced as the desired workload L. We thus define the *load index* $\lambda_{\mathcal{P}} : [R] \to \mathbb{R}_+$ for a given partitioning $\mathcal{P}$ and

Table 2. *Dynamic averages.* We present the data for all dynamic domain experiments. Surface index is presented by computing $\max_r \sigma$ for each frame, averaged across all frames. Temporal index ν is averaged across all frames. *Runtime* is the total runtime for the simulation. Values in parenthesis $(\cdot)$ are computed by dividing with the same metric for the Power partitioner. Performance metrics for these simulations are summarized in Tab. 5.

			SFC			**METIS**			**Power (ours)**		
	R	$\lvert D_t \rvert$	σ	ν	Runtime	σ	ν	Runtime	σ	ν	Runtime
Boat wake	4	[10.4K, 671K]	0.120 (1.27×)	0.006 (1.83×)	2h29m (1.12×)	0.070 (0.74×)	0.736 (210.98×)	7h26m (3.35×)	0.094	0.003	2h13m
Flamethrower	4	[150, 207K]	0.205 (1.67×)	0.044 (2.56×)	5h11m (1.22×)	0.093 (0.75×)	0.737 (42.70×)	7h10m (1.69×)	0.123	0.017	4h14m
Turntable	1	[397K, 599K]	-	-	49m34s	-	-	49m34s	-	-	49m34s
	2	[397K, 599K]	0.025 (1.25×)	0.036 (2.25×)	35m20s (1.12×)	0.013 (0.65×)	0.544 (34.41×)	43m25s (1.38×)	0.020	0.016	31m34s
	4	[397K, 599K]	0.052 (1.31×)	0.078 (2.45×)	21m52s (1.18×)	0.030 (0.75×)	0.746 (23.54×)	30m7s (1.63×)	0.040	0.032	18m28s
	8	[397K, 599K]	0.136 (1.52×)	0.205 (5.54×)	18m33s (1.51×)	0.082 (0.92×)	0.858 (23.16×)	19m38s (1.60×)	0.089	0.037	12m18s
	16	[397K, 599K]	0.266 (1.97×)	0.328 (6.19×)	13m33s (1.44×)	0.122 (0.90×)	0.901 (16.97×)	13m36s (1.45×)	0.135	0.053	9m24s
	32	[397K, 599K]	0.395 (2.04×)	0.503 (7.45×)	10m14s (1.29×)	0.162 (0.84×)	0.939 (13.92×)	9m18s (1.18×)	0.193	0.067	7m54s
Fight scene	8	[32.5K, 369K]	0.214 (1.48×)	0.318 (3.55×)	-	0.192 (1.33×)	0.668 (7.46×)	-	0.145	0.089	-
Corpus	[2, 32]	[10k, 1.2M]	0.155 (1.83×)	0.041 (9.34×)	-	0.081 (0.82×)	0.569 (505.41×)	-	0.100	0.008	-

rank r as a relative deviation from L

$$\lambda_{\mathcal{P}}(r) = \left| \frac{\sum_{\boldsymbol{b} \in \mathcal{P}_r} W_{\boldsymbol{b}}}{L} - 1 \right|. \tag{18}$$

Assuming $W_{\boldsymbol{b}}$ reflects the actual computation needed to process a given bucket, a load index close to zero ensures the work is split evenly across machines.

Surface index. Any nontrivial parallelizable simulation involves accessing some form of neighbor data, which in a distributed setting must be communicated between machines. We desire a metric to reflect the costs associated with such communication, which generally occurs at the borders between partitions. Consider the set of neighboring buckets to the buckets of rank r that are still part of the domain subset D_t, that is $(\mathcal{T}(P_r) \setminus P_r) \cap D_t$. We define the *surface index* $\sigma_{\mathcal{P}} : [R] \to \mathbb{R}_+$ as

$$\sigma_{\mathcal{P}}(r) = \frac{|(\mathcal{T}(P_r) \setminus P_r) \cap D_t|}{|P_r|}, \tag{19}$$

which measures the number of buckets that need to be communicated to rank r relative to the number of buckets it owns.

Temporal consistency index. The other major source of communication arises when the partitioning is recomputed between timesteps in response to changes in the shape of the domain subset, leading to reshuffling of data ownership among ranks. To measure this effect, we define the *consistency index* $\nu : (\mathcal{P}^1, \mathcal{P}^2, S) \to \mathbb{R}_+$ between partitionings $\mathcal{P}^1$ and $\mathcal{P}^2$ on a given bucket subset S by counting the relative number of buckets in S that changed rank,

$$\nu(\mathcal{P}^1, \mathcal{P}^2, S) = \frac{|\{\boldsymbol{b} \in S : \mathcal{P}^1(\boldsymbol{b}) \neq \mathcal{P}^2(\boldsymbol{b})\}|}{|S|}. \tag{20}$$

A value of zero implies that the partitioning remained unchanged and hence no data needed to be transferred between ranks; larger values correspondingly reflect increased transfer costs.

In practice, we are interested in tracking the consistency of a partitioning across time, or more specifically, between subsequent simulation timesteps. In our implementation, re-partitioning $\mathcal{P}_t = \mathbb{P}(D_t^0)$ is performed at the start of every simulation step on D_t^0 and is compared to the previous partitioning $\mathcal{P}_{t-\Delta t} = \mathbb{P}(D_{t-\Delta t}^0)$ extended to $D_{t-\Delta t}^{\text{last}} = D_t^0$, where any new buckets had already been added according to §4.2.6. We set $S = D_t^0 = D_{t-\Delta t}^{\text{last}}$, as both configurations have this same bucket set allocated in memory, and hence the metric value would be indicative of the transfer costs. A visualization of the temporal consistency is shown in Fig. 3.

5.2 Comparison to other partitioners

We perform comparisons w.r.t. METIS and SFC, which are common state-of-the-art partitioning techniques. Relevant details on how to fit these into our sparse domain framework are presented below. In particular, we handle dynamic bucket creation and deletion as in §4.2.6, with the exception that instead of rank positions $\boldsymbol{x}_r$ in Eq. 17 we use the average bucket position $\boldsymbol{x}_r = \sum_{\boldsymbol{b} \in P_r} \boldsymbol{x}_{\boldsymbol{b}} / |P_r|$ which is computed and stored immediately after every call to the partitioner.

SFC. To partition a sparse grid domain D_t with an SFC, the domain is embedded into a bounding cube. It is then subdivided into $n \times n \times n$ sub-boxes, to which the SFC is applied, where n is a power of 2. Typically, 1024 suffices for our applications. Algorithmically this means assigning each sub-box a unique index from 0 to $n^3 - 1$, corresponding to its position along the curve—a procedure well-established for Hilbert, Morton (Z-order), and other SFCs—so that all sub-boxes are ordered in a 1D array of length n^3. Mapping the buckets of the domain D_t to the sub-boxes based on proximity in turn orders the former in a 1D array as well. This 1D array of buckets is then divided into R contiguous intervals for distribution across ranks. Tsuzuki and Aoki [2016] provide further details of this approach. We used Hilbert curves for all SFC tests that we performed.

METIS. We leverage METIS by building a graph G of the domain, using D_t as vertices and $\mathcal{N}$ to create edges. We then pass G on to `METIS_PartGraphRecursive` [Karypis 2013], which returns a graph partitioning that we map back to D_t.

5.3 Application experiments

In the following sections we will use $\mathcal{P} = \{\text{SFC, METIS, Power}\}$ as labels for partitionings created by the respective partitioner. We will omit presenting the load index (Eq. 18), since all partitioners produce approximately even loads across ranks (i.e., $\max_r \lambda_r < \tau_{\text{load}} = 0.01$ for all examples). Unless stated otherwise we always report the largest surface index $\max_r \sigma_{\mathcal{P}}(r)$ (Eq. 19). In what follows, we summarize the experiments we performed and report the resulting data, deferring observations and discussions until §6.

All simulations were run on machines with 64 physical CPU cores and 256 GB of available RAM, with hyperthreading disabled, using OpenMP to parallelize across all physical cores. The machines are connected via a multi-level tree network topology, providing

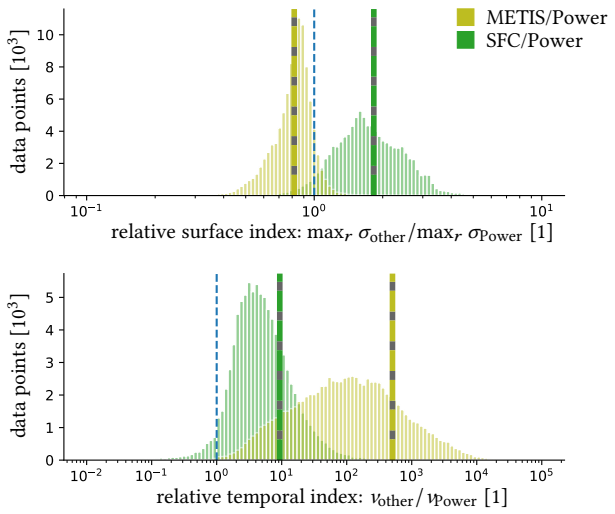

Fig. 7. *Simulation corpus.* We run METIS, SFC, and our Power partitioner on 133 simulation data caches. For each frame in each cache we compute a partitioning $\mathcal{P} = \{\text{METIS, SFC, Power}\}$ and report *relative* metrics METIS/Power (olive) and SFC/Power (green) as a data point on a histogram. Top: the relative maximum surface index (Eq. 19) per frame is recorded. Bottom: the relative temporal index (Eq. 20) per frame is recorded. The mean of each histogram is drawn as a dashed vertical line; the dashed blue line corresponds to the Power partitioner and can be used as a guide to determine how it compares against the other two methods. On average, Power outperforms both SFC (9.34×) and METIS (505×) in temporal index; for surface index, Power outperforms SFC (1.83×) but is mildly outperformed by METIS (0.82×). © Wētā FX Ltd.

roughly 20 Gbps full-duplex bandwidth and inter-node round-trip latency in the range of 0.1-0.5 ms.

Weak scaling on static domains. We choose three analytical domain topologies intended to loosely resemble common simulation scenarios: a meandering river, a smoke plume, and an ocean surface; see Fig. 10. The shapes are defined by analytical signed distance fields, and the region of interest $\Omega_t \equiv \Omega$ is defined as the set of points within the zero isocontour (i.e., values less than or equal to zero). We manually picked a uniform grid size Δ such that $|P_r| \approx 70\text{k}$ for every simulation. For a simple representative simulation, we perform a volumetric fluid simulation using the incompressible Euler equations, consisting of a pressure projection—solved using Successive Over-Relaxation (SOR) with a fixed number (100) of iterations—and semi-Lagrangian advection [Stam 1999] with trilinear interpolation and first order Runge-Kutta backtracing, which we run for 10 timesteps. Each simulation step is initialized with random velocity values to induce greater variety, each simulation receiving the same random inputs. All metrics and timings are reported in Tabs. 3 and 4.

Strong scaling on a dynamic domain. We perform a strong scaling test on a rapidly rotating rectangular box, as shown in Fig. 8, using the same volumetric fluid solver as for the weak scaling test above. The changing domain was designed to stress test temporal consistency of different partitioners. Throughout the simulation $|D_t| \approx 500\text{k}$ (the number varies depending on the orientation). All metrics and timings are reported in Tabs. 2 and 5.

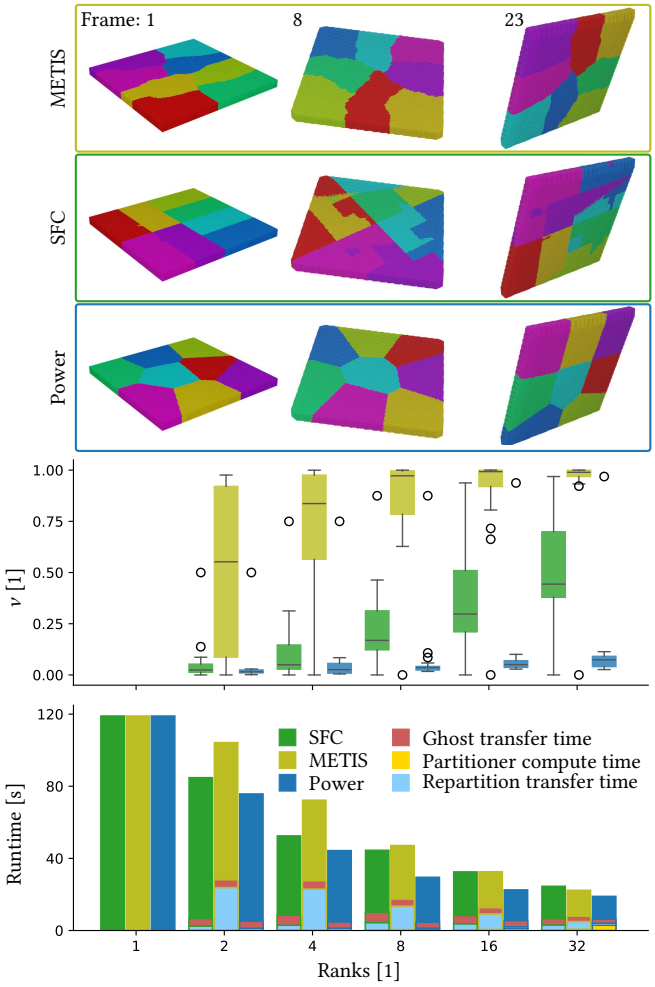

Fig. 8. *Turntable.* A rectangular domain is rotated rapidly for 24 frames, where we perform a Poisson solve followed by an Eulerian advection step. Partitionings for frames 1, 8, and 23 are shown (top); METIS produces similar partitionings every frame, but they shift in time; SFC creates non-local partitionings when the domain is non-axis aligned (see frames 8 and 23); Power (ours) produces local and temporally stable partitionings. The temporal index ν is recorded for every frame in a box plot (middle), together with the total runtime (bottom). © Wētā FX Ltd.

Simulation corpus. We run the partitioners on a corpus of 133 fluid cache sequences—including smoke, fire, and explosions from actual film productions—constituting a total of 102 580 unique frames of volumetric and particle data. For each cache sequence we identified the frame with the most buckets, $\max_t |D_t|$, and chose the number of ranks $R = \max_t |D_t|/50\text{k}$ rounded to the nearest multiple of 2^n ranks for the entire sequence. For each frame+partitioner combination we calculated the temporal index and recorded the maximum surface index across all ranks. To remove the noise and scaling effects associated with the wide variety of domain sizes and topologies and hence vastly different magnitudes of indices, we calculated relative values, SFC/Power and METIS/Power, for the corresponding indices and organized them in a histogram, see Fig. 7. Aggregate values are displayed in Tab. 2.

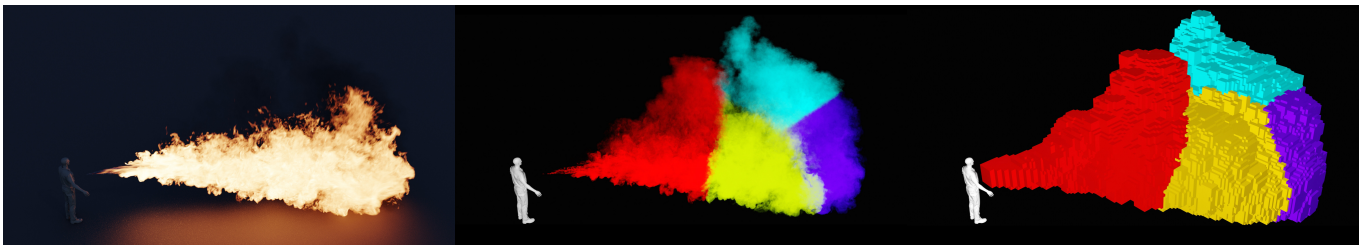

Fig. 9. **Flamethrower.** A volumetric combustion simulation (left) was run on 4 machines with our Power partitioner. Here fire/smoke density (middle) and sparse simulation domain buckets (right) are colored by rank. © Wētā FX Ltd.

Fight scene. We highlight one particularly challenging cache sequence from the corpus: a water simulation shot from *Avatar: The Way of Water*, shown in Fig. 1. In this extremely chaotic scenario, the Power partitioner outperforms both METIS and SFC in both surface and temporal metrics (see Tab. 2).

Boat wake and flamethrower. Lastly, we compare the different partitioners on two common VFX simulation scenarios. First, a moving boat on flat open water is simulated with a hybrid particle-in-cell method [Fu et al. 2017a], gradually expanding the simulation domain to capture the produced wake (Fig. 6). Second, we use a physically based combustion model [Edholm et al. 2023] to create a moving flamethrower (Fig. 9). Runtimes and metrics are reported in Tab. 2.

6 Discussion

We have evaluated our novel Power partitioning algorithm, based on optimal transport, in a range of challenging scenarios, demonstrating significant advancements over other state-of-the-art partitioning methods for sparse dynamic domain configurations with runtime speedups of up to 70% (e.g., "Boat wake", Tab. 2).

Temporal coherence. When averaging over the simulation corpus (Fig. 7), the Power partitioner has superior temporal properties compared to both SFC (9.34×) and METIS (505×). Notably, METIS reshuffles, on average, 57% of the domain subset every simulation step (see Tab. 2), rendering it impractical for sparse dynamic domains.

Data transfers related to re-partitioning can cause significant performance losses. For example, in Fig. 8 there are multiple instances where the re-partitioning transfer time is longer than the total ghost (i.e., neighbor) transfer time for the same timestep, when using METIS. Additionally, in the boat wake example (Fig. 6), both SFC and Power simulate significantly faster than METIS due to their low temporal consistency index, all while having a higher average surface index. Importantly, re-partitioning, in contrast to ghost transfers, is difficult to perform asynchronously, making it a blocking part of the simulation code.

The high temporal coherence of our method is the result of two key aspects. The first is the regularization term in Eq. 3, which ensures there is a *unique* (regularized) optimal coupling $\mathbf{T}$, and the second is our warm-start procedure for $\boldsymbol{x}_r$. The progressive reduction of ε (§4.2.2) during the Lloyd iterations ensures quick convergence to optimal coupling which, when converted to a partitioning via Eq. 15, satisfies the target load threshold (§4.2.3). The resulting partitioning yields power cells that evolve smoothly in time, even when the simulation domain undergoes large topological changes. The convergence study in Fig. 5 supports the reliability of our method, although we lack a formal proof of convergence. While we limit Lloyd iterations l to 10, convergence typically occurs within 1–3 iterations to our target load threshold, $\tau_{\text{load}} = 0.01$. In order for $\mathcal{P}$ to be a good approximation of $\mathbf{T}$ (via Eq. 15) we set $\tau_{\text{sinkhorn}} = 0.5\tau_{\text{load}}$ (Eq. 10) in all of our examples.

One beneficial side-effect of the Lloyd iterations in Alg. 1 is that partitions tend approximately towards centroidal Power cells, although only indirectly, since the termination condition is based solely on τ_{load}. It is possible to introduce an additional criterion that explicitly constrains $\|\boldsymbol{x}_r - \boldsymbol{c}_r\|$, but doing so typically increases the number of Lloyd iterations. While more centroidal Power cells can yield slightly improved spatial locality, as their shapes tend to be more compact, in our experiments the resulting reduction in transfer time was marginal and did not justify the additional iteration cost.

Spatial locality. METIS produces individual partitionings with the highest spatial locality across all of our tests, as indicated by a low surface index (Tabs. 2-3). This is not surprising: METIS receives topological information of the domain via $\mathcal{N}$, which both Power and SFC lack. Despite this, the Power partitioner is able to partition many challenging and non-convex scenarios close to optimally, such as the crescent (Fig. 3) and river (Fig. 10, left).

Importantly, the Power partitioner produces much more robust partitionings compared to SFC. Consider the still-frame partitionings with SFC in Fig. 8 (2nd row from top). Since the SFC is embedded into a bounding cube, small rotations of the domain can have catastrophic effects on the partitioning quality.

Note that the relationship between surface index and the amount of ghost data transfers is nontrivial in practice. It depends on the implementation details of the simulator, since the data-access patterns of individual algorithms, as well as the bucket data memory layout, determine what data must be transferred and when. We do not attempt to derive an exact relationship in this work, and instead provide the timing breakdowns in Tabs. 5-4.

Performance. The computation time of the Power partitioner deteriorates as $R \times |D|$ grows, caused by the dense matrix-vector products in the Sinkhorn solve (Eqs. 8-9) or log-domain Sinkhorn solve (Eqs. 11-12). We have found that a simple coarsening procedure (§4.2.5) alleviates the problem (Fig. 10), which ensures partitioning takes less than 2% of simulation time even for $R = 32$ and $|D| \approx 2\text{M}$ (see exact numbers in Tabs. 3 and 4). Interestingly, the coarsened Power partitioner sometimes yields a lower surface index than the

non-coarsened version (Fig. 10), which we believe is due to coarsening creating beneficial neighbor patterns. In principle, one could adapt the coarsening level based on the regularization parameter, selecting a bucket size that matches the effective "blurring" of the optimal coupling induced by ε. Since the scale of our typical problems is within $R < 10$ and $|D| < 1$M, we have not found coarsening to be a requirement in practice and we leave such explorations of coarsening strategies to future work.

The scaling, both strong (Fig. 8) and weak (Fig. 10), of our solvers is suboptimal by supercomputing standards. We attribute this partly to ghost transfer communication, which was implemented with blocking calls to ensure we could accurately measure time spent in communication vs computation. Asynchronously overlapping communication and computation, however, is unlikely to hide all communication costs, since many algorithms commonly used in VFX are memory bound. Therefore, the need for a high-quality partitioner remains relevant.

7 Conclusion

Our proposed partitioning algorithm, leveraging optimal transport to generate a power diagram based on the distribution of work in the domain, demonstrates significant advancements in distributing large-scale physics-based simulations for VFX applications. We evaluated our method on real-world production scenarios, which feature highly dynamic domain topologies on sparse and non-convex domains. To analyze the quality of a partitioning we introduced the *surface index* and the *temporal consistency index*, and analyzed our method's effect on simulation runtimes. The study in Fig. 7 indicates major improvements in temporal index over SFC and METIS. The surface index is improved considerably over SFC, while being slightly worse than for METIS. Overall, our proposed method strikes a solid balance on the qualitative objectives we stated at the outset, generally outperforming both METIS and SFC in total runtime.

References

Dan Bailey, Harry Biddle, Nick Avramoussis, and Matthew Warner. 2015. Distributing liquids using OpenVDB. In *ACM SIGGRAPH 2015 Talks*. ACM, New York, NY, USA.

Berger and Bokhari. 1987. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Trans. Comput.* C-36, 5 (May 1987), 570–580.

Morten Bojsen-Hansen, Michael Bang Nielsen, Konstantinos Stamatelos, and Robert Bridson. 2021. Spatially Adaptive Volume Tools in Bifrost. In *ACM SIGGRAPH Talks*.

Ricard Borrell, Guillermo Oyarzun, Damien Dosimont, and Guillaume Houzeaux. 2020. Parallel SFC-based mesh partitioning and load balancing. *arXiv* (July 2020).

Robert Edward Bridson. 2003. *Computational aspects of dynamic surfaces*. Ph. D. Dissertation. Stanford, CA, USA.

Magnus Bruaset and Aslak Tveito (Eds.). 2005. *Numerical solution of partial differential equations on parallel computers* (2006 ed.). Springer, Berlin, Germany.

Fernando de Goes, Katherine Breeden, Victor Ostromoukhov, and Mathieu Desbrun. 2012. Blue noise through optimal transport. *ACM Trans. Graph.* 31, 6 (Nov. 2012).

Fernando de Goes, Corentin Wallez, Jin Huang, Dmitry Pavlov, and Mathieu Desbrun. 2015. Power particles: an incompressible fluid solver based on power diagrams. *ACM Trans. Graph.* 34, 4 (July 2015), 1–11.

John Edholm, Alexey Stomakhin, Rahul Deshprabhu, David Caeiro Cebrian, Florian Hu, and Caitlin Pope. 2023. Fire and Explosions in Avatar: The Way of Water. In *ACM SIGGRAPH 2023 Talks (SIGGRAPH '23)*.

Lucio Flores and David Horsley. 2009. Underground cave sequence for Land of the Lost. In *SIGGRAPH 2009: Talks*. ACM, New York, NY, USA.

Chuyuan Fu, Qi Guo, Theodore Gast, Chenfanfu Jiang, and Joseph Teran. 2017a. A polynomial particle-in-cell method. *ACM Trans. Graph.* 36, 6 (Nov. 2017), 1–12.

Lin Fu, Xiangyu Y Hu, and Nikolaus A Adams. 2017b. A physics-motivated Centroidal Voronoi Particle domain decomposition method. *J. Comput. Phys.* 335 (April 2017).

Geoffrey Irving, Eran Guendelman, Frank Losasso, and Ronald Fedkiw. 2006. Efficient simulation of large bodies of water by coupling two and three dimensional techniques. In *SIGGRAPH '06*. ACM Press, New York, New York, USA.

Laxmikant V Kale and Sanjeev Krishnan. 1993. CHARM++. In *Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications*. ACM, New York, NY, USA.

George Karypis. 2013. METIS Manual: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices. https://github.com/KarypisLab/METIS/blob/master/manual/manual.pdf. Accessed: 2025-4-13.

George Karypis and Vipin Kumar. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.* 20, 1 (Jan. 1998), 359–392.

R Koradi, M Billeter, and P Güntert. 2000. Point-centered domain decomposition for parallel molecular dynamics simulation. *Comput. Phys. Commun.* 124 (Feb. 2000).

Jeff Lait. 2016. Inside houdini's distributed solver system. In *ACM SIGGRAPH 2016 Talks*. ACM, New York, NY, USA.

Steve Lesser, Alexey Stomakhin, Gilles Daviet, Joel Wretborn, John Edholm, Noh-Hoon Lee, Eston Schweickart, Xiao Zhai, Sean Flynn, and Andrew Moffat. 2022. Loki: a unified multiphysics simulation framework for production. *ACM Trans. Graph.* 41, 4 (July 2022), 1–20.

S Lloyd. 1982. Least squares quantization in PCM. *IEEE Trans.* 28, 2 (March 1982).

Manish Mandad, David Cohen-Steiner, Leif Kobbelt, Pierre Alliez, and Mathieu Desbrun. 2017. Variance-minimizing transport plans for inter-surface mapping. *ACM Trans. Graph.* 36, 4 (Aug. 2017), 1–14.

Aleka McAdams, Yongning Zhu, Andrew Selle, Mark Empey, Rasmus Tamstorf, Joseph Teran, and Eftychios Sifakis. 2011. Efficient elasticity for character skinning with contact and collisions. *ACM Trans. Graph.* 30, 4 (July 2011), 1–12.

Ken Museth. 2013. VDB: High-resolution sparse volumes with dynamic topology. *ACM Trans. Graph.* 32, 3 (June 2013), 1–22.

Olga Pearce, Todd Gamblin, Bronis R de Supinski, Martin Schulz, and Nancy M Amato. 2012. Quantifying the effectiveness of load balance algorithms. In *Supercomputing*.

Gabriel Peyré and Marco Cuturi. 2019. Computational Optimal Transport: With Applications to Data Science. *Foundations and Trends® in Machine Learning* 11 (2019).

Yuxing Qiu, Samuel Temple Reeve, Minchen Li, Yin Yang, Stuart R Slattery, and Chenfanfu Jiang. 2022. A sparse distributed gigascale resolution Material Point Method. *ACM Trans. Graph.* (Nov. 2022).

Ziyin Qu, Minchen Li, Fernando De Goes, and Chenfanfu Jiang. 2022. The power particle-in-cell method. *ACM Trans. Graph.* 41, 4 (July 2022), 1–13.

Ziyin Qu, Minchen Li, Yin Yang, Chenfanfu Jiang, and Fernando De Goes. 2023. Power plastics: A hybrid Lagrangian/Eulerian solver for mesoscale inelastic flows. *ACM Trans. Graph.* 42, 6 (Dec. 2023), 1–11.

Kirk Schloegel, George Karypis, and Vipin Kumar. 2003. Graph partitioning for high-performance scientific simulations. In *Sourcebook of parallel computing*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 491–541.

Rajsekhar Setaluri, Mridul Aanjaneya, Sean Bauer, and Eftychios Sifakis. 2014. SPGrid: a sparse paged grid structure applied to adaptive smoke simulation. *ACM Trans. Graph.* 33, 6 (Nov. 2014), 1–12.

C Shah, D Hyde, H Qu, and P Levis. 2018. Distributing and load balancing sparse fluid simulations. *Comput. Graph. Forum* 37, 8 (Dec. 2018), 35–46.

Justin Solomon. 2018. Optimal Transport on Discrete Domains. *arXiv* (Jan. 2018).

Justin Solomon, Fernando de Goes, Gabriel Peyré, Marco Cuturi, Adrian Butscher, Andy Nguyen, Tao Du, and Leonidas Guibas. 2015. Convolutional wasserstein distances. *ACM Trans. Graph.* 34, 4 (July 2015), 1–11.

Jos Stam. 1999. Stable fluids. In *SIGGRAPH '99*. ACM.

Alexey Stomakhin, Steve Lesser, Joel Wretborn, Sean Flynn, Johnathan Nixon, Nicholas Illingworth, Adrien Rollet, Kevin Blom, and Douglas Mchale. 2023. Pahi: A Unified Water Pipeline and Toolset *(DigiPro '23)*.

Igor Surmin, Alexei Bashinov, Sergey Bastrakov, Evgeny Efimenko, Arkady Gonoskov, and Iosif Meyerov. 2015. Dynamic load balancing based on rectilinear partitioning in particle-in-cell plasma simulation. Springer.

Satori Tsuzuki and Takayuki Aoki. 2016. Effective dynamic load balance using space-filling curves for large-scale SPH simulations on GPU-rich supercomputers. In *2016 7th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)*. IEEE.

Huamin Wang, Peter J Mucha, and Greg Turk. 2005. Water drops on surfaces. *ACM Trans. Graph.* 24, 3 (July 2005), 921–929.

Xinlei Wang, Yuxing Qiu, Stuart R Slattery, Yu Fang, Minchen Li, Song-Chun Zhu, Yixin Zhu, Min Tang, Dinesh Manocha, and Chenfanfu Jiang. 2020. A massively parallel and scalable multi-GPU material point method. *ACM Trans. Graph.* 39, 4 (Aug. 2020).

Joel Wretborn, Alexey Stomakhin, and Christopher Batty. 2025. A unified multi-scale method for simulating immersed bubbles. *Comput. Graph. Forum* (April 2025).

Xiao Zhai, Fei Hou, Hong Qin, and Aimin Hao. 2020. Fluid Simulation with Adaptive Staggered Power Particles on GPUs. *IEEE Trans. Vis. Comput. Graph.* 26, 6 (June 2020), 2234–2246.

Yongning Zhu, Eftychios Sifakis, Joseph Teran, and Achi Brandt. 2010. An efficient multigrid method for the simulation of high-resolution elastic solids. *ACM Trans. Graph.* 29, 2 (March 2010), 1–18.

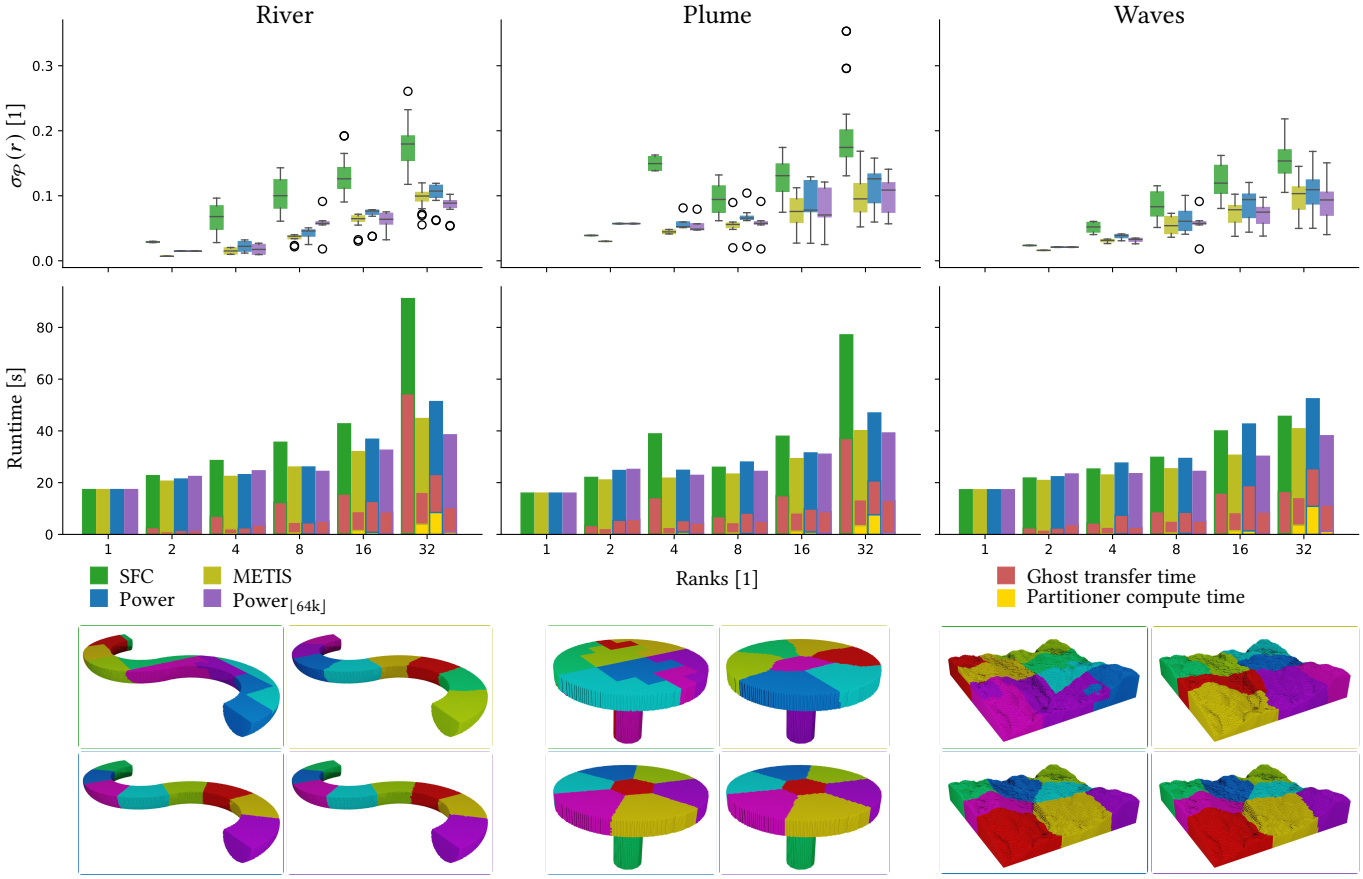

Fig. 10. *Weak scaling.* Results from the River, Plume, and Waves weak scaling examples are shown, comparing different partitioning algorithms on a static domain, where we perform a Poisson solve followed by an Eulerian advection step. We provide the surface index $\sigma_{\mathcal{P}}(r)$ (Eq. 19) from the final frame (top row), as well as performance numbers (middle row) from the frame with the lowest total runtime (out of 10 frames). We explicitly mark the time spent computing the partitioning and time spent transferring ghost data. Since the domain is static, we do not provide the temporal consistency index. For a visual comparison we also visualize the partitionings for 8 machines; the color of the image border indicates what algorithm was used. Performance metrics for these simulations are summarized in Tab. 4. © Wētā FX Ltd.

Table 3. *Static averages.* We present the data for the static domain experiments in Fig. 10, picking the frame with the shortest runtime out of 10 frames. Surface index is presented by computing $\max_r \sigma$. *Runtime* is the runtime for the frame. Values in parentheses $(\cdot)$ are computed by dividing by the same metric for the Power$_{\lfloor 64k \rfloor}$ partitioner.

			SFC		**METIS**		**Power (ours)**		**Power$_{\lfloor 64k \rfloor}$ (ours)**	
	R	$\lvert D_t \rvert$	σ	Runtime	σ	Runtime	σ	Runtime	σ	Runtime
River	1	69.6K	-	17s (1.00×)	-	17s (1.00×)	-	17s (1.00×)	-	17s
	2	141K	0.030 (2.02×)	22s (1.01×)	0.007 (0.47×)	20s (0.92×)	0.015 (1.00×)	21s (0.96×)	0.015	22s
	4	281K	0.096 (3.57×)	28s (1.16×)	0.020 (0.76×)	22s (0.91×)	0.032 (1.19×)	23s (0.94×)	0.027	24s
	8	453K	0.143 (1.61×)	35s (1.46×)	0.040 (0.45×)	26s (1.07×)	0.051 (0.57×)	26s (1.07×)	0.089	24s
	16	1.09M	0.192 (2.56×)	42s (1.31×)	0.072 (0.95×)	31s (0.98×)	0.079 (1.05×)	36s (1.13×)	0.075	32s
	32	2.14M	0.261 (2.54×)	1m31s (2.37×)	0.120 (1.17×)	44s (1.16×)	0.121 (1.18×)	51s (1.33×)	0.102	38s
Plume	1	60.7K	-	15s (1.00×)	-	15s (1.00×)	-	15s (1.00×)	-	15s
	2	126K	0.039 (0.68×)	22s (0.88×)	0.030 (0.52×)	21s (0.84×)	0.057 (1.00×)	24s (0.98×)	0.057	25s
	4	235K	0.163 (2.06×)	38s (1.70×)	0.048 (0.61×)	21s (0.95×)	0.081 (1.03×)	24s (1.09×)	0.079	22s
	8	453K	0.132 (1.48×)	25s (1.07×)	0.090 (1.01×)	23s (0.96×)	0.103 (1.16×)	27s (1.15×)	0.089	24s
	16	921K	0.174 (1.55×)	37s (1.22×)	0.112 (1.00×)	29s (0.95×)	0.127 (1.13×)	31s (1.02×)	0.113	31s
	32	1.85M	0.353 (2.50×)	1m17s (1.97×)	0.168 (1.19×)	40s (1.02×)	0.157 (1.11×)	46s (1.20×)	0.141	39s
Waves	1	70.8K	-	17s (1.00×)	-	17s (1.00×)	-	17s (1.00×)	-	17s
	2	136K	0.024 (1.11×)	21s (0.93×)	0.016 (0.75×)	20s (0.89×)	0.021 (1.00×)	22s (0.95×)	0.021	23s
	4	273K	0.061 (1.73×)	25s (1.08×)	0.033 (0.96×)	22s (0.98×)	0.041 (1.18×)	27s (1.18×)	0.035	23s
	8	453K	0.115 (1.30×)	29s (1.22×)	0.073 (0.82×)	25s (1.04×)	0.101 (1.14×)	29s (1.21×)	0.089	24s
	16	1M	0.162 (1.66×)	39s (1.32×)	0.102 (1.05×)	30s (1.01×)	0.119 (1.22×)	42s (1.41×)	0.098	30s
	32	2.05M	0.218 (1.44×)	45s (1.20×)	0.145 (0.96×)	40s (1.07×)	0.168 (1.11×)	52s (1.37×)	0.151	38s

Table 4. *Static performance breakdown.* Breakdown of memory usage, communication volume, and runtime for static-domain simulations (no changes in the domain subset over time). Each row corresponds to the simulation and partitioner configuration listed in Table 3. All values are reported for rank 0. RAM usage is reported as the maximum over the simulation; all other values are taken from the frame with the shortest *Runtime*. Ghost data includes both sent and received data, and the corresponding transfer time includes rank synchronization. Values in parentheses $(\cdot)$ are computed by dividing by the same metric for the Power partitioner.

	R	RAM (GB)		Runtime (s)		Ghost data (GB)		Ghost transfer and sync (s)		Partitioner compute (s)	
River	1	14.53	(1.00×)	17.32s	(1.00×)	-		-		-	
SFC	2	8.85	(1.00×)	22.71s	(1.06×)	1.93	(1.96×)	2.62s	(1.72×)	14ms	(1.31×)
METIS		8.73	(0.98×)	20.57s	(0.96×)	0.46	(0.46×)	1.04s	(0.68×)	57ms	(5.20×)
Power		8.89	(1.00×)	21.40s	(1.00×)	0.98	(1.00×)	1.53s	(1.00×)	11ms	(1.00×)
$\text{Power}_{\lfloor 64k \rfloor}$		9.01	(1.01×)	22.39s	(1.05×)	0.98	(1.00×)	1.57s	(1.02×)	15ms	(1.38×)
SFC	4	9.14	(1.00×)	28.54s	(1.24×)	1.79	(0.90×)	6.99s	(2.92×)	28ms	(0.49×)
METIS		9.00	(0.99×)	22.41s	(0.97×)	0.71	(0.36×)	1.98s	(0.82×)	198ms	(3.44×)
Power		9.10	(1.00×)	23.08s	(1.00×)	2.00	(1.00×)	2.39s	(1.00×)	58ms	(1.00×)
$\text{Power}_{\lfloor 64k \rfloor}$		9.21	(1.01×)	24.57s	(1.06×)	1.62	(0.81×)	3.42s	(1.43×)	51ms	(0.89×)
SFC	8	9.44	(1.00×)	35.62s	(1.37×)	3.90	(1.24×)	12.22s	(2.91×)	63ms	(0.39×)
METIS		9.42	(0.99×)	26.03s	(1.00×)	2.49	(0.79×)	4.11s	(0.98×)	570ms	(3.47×)
Power		9.47	(1.00×)	26.05s	(1.00×)	3.15	(1.00×)	4.20s	(1.00×)	164ms	(1.00×)
$\text{Power}_{\lfloor 64k \rfloor}$		11.90	(1.26×)	24.35s	(0.93×)	4.85	(1.54×)	4.75s	(1.13×)	111ms	(0.67×)
SFC	16	10.90	(0.99×)	42.73s	(1.16×)	9.75	(1.90×)	15.47s	(1.37×)	130ms	(0.16×)
METIS		9.79	(0.89×)	31.98s	(0.87×)	1.96	(0.38×)	7.32s	(0.65×)	1.51s	(1.83×)
Power		11.02	(1.00×)	36.79s	(1.00×)	5.14	(1.00×)	11.32s	(1.00×)	825ms	(1.00×)
$\text{Power}_{\lfloor 64k \rfloor}$		10.24	(0.93×)	32.55s	(0.88×)	4.79	(0.93×)	8.27s	(0.73×)	250ms	(0.30×)
SFC	32	11.38	(0.86×)	1m31s	(1.78×)	10.63	(1.42×)	54.07s	(4.25×)	266ms	(0.03×)
METIS		11.26	(0.86×)	44.78s	(0.87×)	6.15	(0.82×)	12.33s	(0.97×)	3.94s	(0.46×)
Power		13.16	(1.00×)	51.33s	(1.00×)	7.50	(1.00×)	12.73s	(1.00×)	8.49s	(1.00×)
$\text{Power}_{\lfloor 64k \rfloor}$		11.07	(0.84×)	38.48s	(0.75×)	6.45	(0.86×)	9.01s	(0.71×)	986ms	(0.12×)
Plume	1	13.75	(1.00×)	15.96s	(1.00×)	-		-		-	
SFC	2	8.58	(0.99×)	22.03s	(0.89×)	2.27	(0.67×)	3.43s	(0.63×)	14ms	(1.56×)
METIS		8.30	(0.96×)	21.03s	(0.85×)	1.75	(0.52×)	2.24s	(0.41×)	59ms	(6.74×)
Power		8.63	(1.00×)	24.70s	(1.00×)	3.39	(1.00×)	5.40s	(1.00×)	9ms	(1.00×)
$\text{Power}_{\lfloor 64k \rfloor}$		8.72	(1.01×)	25.13s	(1.02×)	3.39	(1.00×)	5.73s	(1.06×)	11ms	(1.25×)
SFC	4	10.14	(0.73×)	38.85s	(1.57×)	9.77	(3.32×)	14.18s	(3.24×)	25ms	(0.03×)
METIS		8.05	(0.58×)	21.73s	(0.88×)	2.29	(0.78×)	2.57s	(0.59×)	171ms	(0.18×)
Power		13.81	(1.00×)	24.78s	(1.00×)	2.94	(1.00×)	4.37s	(1.00×)	961ms	(1.00×)
$\text{Power}_{\lfloor 64k \rfloor}$		13.75	(1.00×)	22.82s	(0.92×)	2.72	(0.92×)	4.07s	(0.93×)	191ms	(0.20×)
SFC	8	9.27	(0.77×)	25.95s	(0.93×)	6.65	(1.17×)	6.81s	(0.95×)	47ms	(0.13×)
METIS		8.19	(0.68×)	23.32s	(0.83×)	3.07	(0.54×)	4.05s	(0.56×)	531ms	(1.51×)
Power		12.03	(1.00×)	27.96s	(1.00×)	5.67	(1.00×)	7.19s	(1.00×)	351ms	(1.00×)
$\text{Power}_{\lfloor 64k \rfloor}$		11.90	(0.99×)	24.35s	(0.87×)	4.85	(0.86×)	4.75s	(0.66×)	111ms	(0.32×)
SFC	16	9.23	(0.84×)	37.94s	(1.21×)	6.18	(0.90×)	14.86s	(1.76×)	99ms	(0.10×)
METIS		8.36	(0.76×)	29.30s	(0.93×)	3.20	(0.47×)	7.03s	(0.83×)	1.28s	(1.29×)
Power		10.95	(1.00×)	31.49s	(1.00×)	6.85	(1.00×)	8.44s	(1.00×)	992ms	(1.00×)
$\text{Power}_{\lfloor 64k \rfloor}$		10.90	(1.00×)	31.00s	(0.98×)	6.63	(0.97×)	7.49s	(0.89×)	519ms	(0.52×)
SFC	32	11.75	(1.07×)	1m17s	(1.64×)	23.12	(2.78×)	36.84s	(2.96×)	208ms	(0.03×)
METIS		9.00	(0.82×)	40.08s	(0.85×)	4.31	(0.52×)	10.00s	(0.80×)	3.39s	(0.44×)
Power		10.99	(1.00×)	46.93s	(1.00×)	8.33	(1.00×)	12.43s	(1.00×)	7.63s	(1.00×)
$\text{Power}_{\lfloor 64k \rfloor}$		10.21	(0.93×)	39.20s	(0.84×)	8.08	(0.97×)	11.89s	(0.96×)	684ms	(0.09×)
Waves	1	14.61	(1.00×)	17.26s	(1.00×)	-		-		-	
SFC	2	8.97	(0.90×)	21.78s	(0.98×)	1.50	(1.12×)	2.50s	(1.04×)	14ms	(0.64×)
METIS		9.03	(0.90×)	20.83s	(0.94×)	1.02	(0.76×)	1.64s	(0.68×)	55ms	(2.58×)
Power		10.01	(1.00×)	22.28s	(1.00×)	1.33	(1.00×)	2.41s	(1.00×)	21ms	(1.00×)
$\text{Power}_{\lfloor 64k \rfloor}$		9.98	(1.00×)	23.35s	(1.05×)	1.33	(1.00×)	3.77s	(1.56×)	18ms	(0.87×)
SFC	4	9.79	(1.00×)	25.27s	(0.92×)	2.76	(1.04×)	4.39s	(0.61×)	30ms	(0.15×)
METIS		9.53	(0.97×)	22.95s	(0.83×)	2.06	(0.78×)	2.60s	(0.36×)	192ms	(0.96×)
Power		9.82	(1.00×)	27.57s	(1.00×)	2.64	(1.00×)	7.21s	(1.00×)	200ms	(1.00×)
$\text{Power}_{\lfloor 64k \rfloor}$		9.62	(0.98×)	23.46s	(0.85×)	2.25	(0.85×)	2.64s	(0.37×)	71ms	(0.35×)
SFC	8	10.97	(0.97×)	29.80s	(1.01×)	5.01	(1.97×)	8.71s	(1.11×)	55ms	(0.14×)
METIS		10.83	(0.96×)	25.34s	(0.86×)	4.46	(1.75×)	4.57s	(0.58×)	587ms	(1.51×)
Power		11.30	(1.00×)	29.36s	(1.00×)	2.54	(1.00×)	7.81s	(1.00×)	389ms	(1.00×)
$\text{Power}_{\lfloor 64k \rfloor}$		11.90	(1.05×)	24.35s	(0.83×)	4.85	(1.91×)	4.75s	(0.61×)	111ms	(0.28×)
SFC	16	13.06	(0.99×)	39.98s	(0.94×)	6.31	(0.92×)	15.86s	(0.96×)	120ms	(0.08×)
METIS		12.86	(0.98×)	30.61s	(0.72×)	4.89	(0.72×)	6.99s	(0.42×)	1.45s	(1.02×)
Power		13.14	(1.00×)	42.65s	(1.00×)	6.82	(1.00×)	16.58s	(1.00×)	1.43s	(1.00×)
$\text{Power}_{\lfloor 64k \rfloor}$		13.33	(1.01×)	30.20s	(0.71×)	5.82	(0.85×)	8.08s	(0.49×)	197ms	(0.14×)
SFC	32	17.74	(0.98×)	45.63s	(0.87×)	7.80	(1.21×)	16.35s	(1.20×)	261ms	(0.02×)
METIS		17.63	(0.98×)	40.81s	(0.78×)	3.07	(0.48×)	10.66s	(0.78×)	3.59s	(0.33×)
Power		18.02	(1.00×)	52.40s	(1.00×)	6.43	(1.00×)	13.67s	(1.00×)	10.84s	(1.00×)
$\text{Power}_{\lfloor 64k \rfloor}$		17.97	(1.00×)	38.13s	(0.73×)	5.57	(0.87×)	9.79s	(0.72×)	1.17s	(0.11×)

Table 5. *Dynamic performance breakdown.* Breakdown of memory usage, communication volume, and runtime for dynamic-domain simulations. Each row corresponds to the simulation and partitioner listed in Table 2. All values are reported for rank 0. RAM usage is reported as the maximum over the simulation; all other values are averaged over the number of frames. Ghost data and repartition data includes both sent and received data, and are separated to highlight the cost of spatial locality (ghost data) versus temporal coherence (repartition data). Synchronization of ranks is included in the ghost transfer time. Values in parentheses $(\cdot)$ are computed by dividing by the same metric for the Power partitioner.

	frames	R	**RAM (GB)**		**Runtime (s/frame)**		**Ghost data (GB/frame)**		**Ghost transfer and sync (s/frame)**		**Repartition data (GB/frame)**		**Repartition transfer (s/frame)**		**Partitioner compute (s/frame)**	
Boat wake																
SFC	262	4	49.79	(1.23×)	34.22s	(1.12×)	2.81	(1.90×)	4.93s	(1.54×)	0.07	(1.60×)	255ms	(1.67×)	18ms	(0.44×)
METIS			65.50	(1.62×)	1m42s	(3.35×)	1.68	(1.14×)	8.70s	(2.72×)	18.35	(422.80×)	59.57s	(389.41×)	105ms	(2.60×)
Power			40.50	(1.00×)	30.48s	(1.00×)	1.48	(1.00×)	3.21s	(1.00×)	0.04	(1.00×)	153ms	(1.00×)	40ms	(1.00×)
Flamethrower																
SFC	240	4	14.74	(1.18×)	1m17s	(1.22×)	9.76	(1.80×)	21.04s	(2.02×)	0.20	(3.14×)	1.04s	(2.52×)	82ms	(0.11×)
METIS			20.33	(1.63×)	1m47s	(1.69×)	6.58	(1.21×)	14.93s	(1.43×)	7.54	(117.62×)	30.62s	(74.48×)	605ms	(0.79×)
Power			12.51	(1.00×)	1m3s	(1.00×)	5.44	(1.00×)	10.43s	(1.00×)	0.06	(1.00×)	411ms	(1.00×)	763ms	(1.00×)
Turntable	25	1	57.76		1m58s	(1.00×)	-		-		-		-			
SFC		2	42.82	(1.27×)	1m24s	(1.12×)	5.46	(1.36×)	4.18s	(1.07×)	0.76	(2.04×)	2.38s	(2.06×)	53ms	(0.77×)
METIS			83.40	(2.47×)	1m44s	(1.38×)	3.49	(0.87×)	4.24s	(1.09×)	9.52	(25.49×)	23.56s	(20.37×)	297ms	(4.28×)
Power			33.73	(1.00×)	1m15s	(1.00×)	4.00	(1.00×)	3.91s	(1.00×)	0.37	(1.00×)	1.16s	(1.00×)	69ms	(1.00×)
SFC		4	24.49	(1.13×)	52.50s	(1.18×)	4.63	(1.15×)	5.57s	(1.65×)	0.40	(1.15×)	2.84s	(2.34×)	56ms	(0.55×)
METIS			49.82	(2.30×)	1m12s	(1.63×)	3.38	(0.84×)	4.54s	(1.34×)	6.63	(19.19×)	22.59s	(18.60×)	436ms	(4.31×)
Power			21.62	(1.00×)	44.36s	(1.00×)	4.03	(1.00×)	3.38s	(1.00×)	0.35	(1.00×)	1.21s	(1.00×)	101ms	(1.00×)
SFC		8	14.63	(1.11×)	44.53s	(1.51×)	4.77	(1.34×)	5.75s	(1.78×)	0.46	(2.70×)	4.32s	(4.52×)	54ms	(0.15×)
METIS			29.96	(2.26×)	47.15s	(1.60×)	3.16	(0.88×)	3.78s	(1.17×)	3.65	(21.15×)	13.05s	(13.67×)	560ms	(1.56×)
Power			13.23	(1.00×)	29.55s	(1.00×)	3.57	(1.00×)	3.23s	(1.00×)	0.17	(1.00×)	955ms	(1.00×)	359ms	(1.00×)
SFC		16	10.49	(1.33×)	32.54s	(1.44×)	3.21	(1.29×)	4.55s	(1.36×)	0.38	(3.82×)	3.69s	(3.73×)	53ms	(0.05×)
METIS			15.97	(2.02×)	32.66s	(1.45×)	2.53	(1.02×)	3.35s	(1.00×)	1.98	(19.72×)	8.61s	(8.69×)	644ms	(0.57×)
Power			7.89	(1.00×)	22.56s	(1.00×)	2.48	(1.00×)	3.35s	(1.00×)	0.10	(1.00×)	991ms	(1.00×)	1.13s	(1.00×)
SFC		32	7.20	(1.07×)	24.57s	(1.29×)	2.85	(1.54×)	3.46s	(1.55×)	0.29	(3.66×)	3.00s	(2.55×)	53ms	(0.02×)
METIS			9.72	(1.44×)	22.35s	(1.18×)	1.91	(1.03×)	2.95s	(1.32×)	1.00	(12.78×)	4.24s	(3.61×)	759ms	(0.26×)
Power			6.74	(1.00×)	18.97s	(1.00×)	1.85	(1.00×)	2.24s	(1.00×)	0.08	(1.00×)	1.18s	(1.00×)	2.89s	(1.00×)