2023 IEEE 30th International Conference on High Performance Computing, Data and Analytics Workshop (HiPCW) | 979-8-3503-8378-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/HIPCW61695.2023.00020

Analysis of MPI Communication Time for Distribution of Repartitioned Data

John-Paul Robinson*, Ke Fan†, Sidharth Kumar†
*University of Alabama at Birmingham, jpr@uab.edu
†University of Illinois Chicago, {kfan23, sidharth}@uic.edu

***Abstract*—Repartitioning in a parallel setting can be defined as the task of redistributing data across processes based on a newly imposed grid/layout. Repartitioning is a fundamental problem, with applications in domains that typically involve computation on tiles (blocks/patches) of varying resolution, for example, while creating multiresolution data formats in in-situ mode (such as the JPEG format and its variants). This paper explores the performance and tradeoffs of different ways to perform the data redistribution phase. We explore a greedy scheme that aims to minimize data movement while compromising on load balancing and a balanced scheme that aims to create a balanced load across processes while compromising on data movement. For both schemes, we further compare per-patch (staggered data transfer) and per-rank (aggregated data transfer) communication patterns to measure the impact of buffer size on MPI point-to-point communication performance. We conclude that the reduced data movement of the greedy scheme leads to reduced transfer times during redistribution. We further conclude that the per-patch communication pattern outperforms per-rank communication.**

***Index Terms*—data movement, data layout, load-balancing**

I. Introduction

Repartitioning entails the task of redistributing existing data across the available set of processes based on a new grid/layout [3]. There are different approaches to distributing the new patches across the available set of processes. Figure 1 introduces the data repartitioning problem and highlights a key challenge: how to assign shared-patches *(SP)* to ranks.

The *Initial State* image tile shows the global data set evenly distributed across four processes, labeled P0-3 and color-coded to highlight patch spanning and placement in later steps. The *Imposed Patch Grid* tile shows the Initial State with a patch division logically imposed across the data set indicating patch boundaries now shared across ranks, labeled p0-8. The logical patches need to be arranged as contiguous data assigned to specific ranks for further computation. The main challenge in assigning patches to ranks is how to deal with patches that span processes: shared-patches *(SP)* p1, p3, p4, p5, and p7. Fully-contained patches *(FCP)* are fully contained within a process after repartitioning (p0, p2, p6 and p8) and remain with their original rank to minimize data movement. The *Balanced* and *Greedy* image tiles highlight the two patch distribution schemes explored, leading to even and uneven patch data distribution for subsequent compute stages. Greedy placement assigns the SP to the rank that already holds the largest part of the SP, at the cost of imbalanced workload distribution. The balanced placement algorithm seeks to maintain an even distribution of patches to maintain computational workload balance for subsequent patch computations [2].

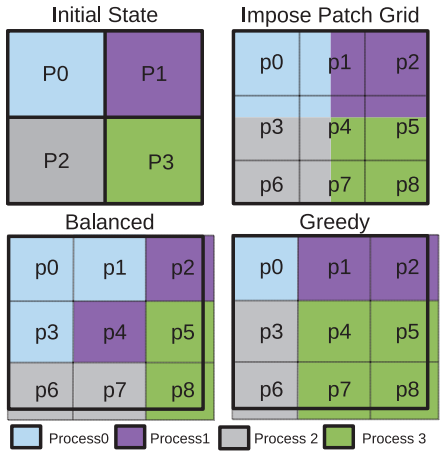


Fig. 1: Data Repartitioning Problem: reassigning shared patches resulting from new grid/layout to ranks. Color indicates process owner of data before and after repartitioning.

II. Emperical evaluation

A. HPC platform

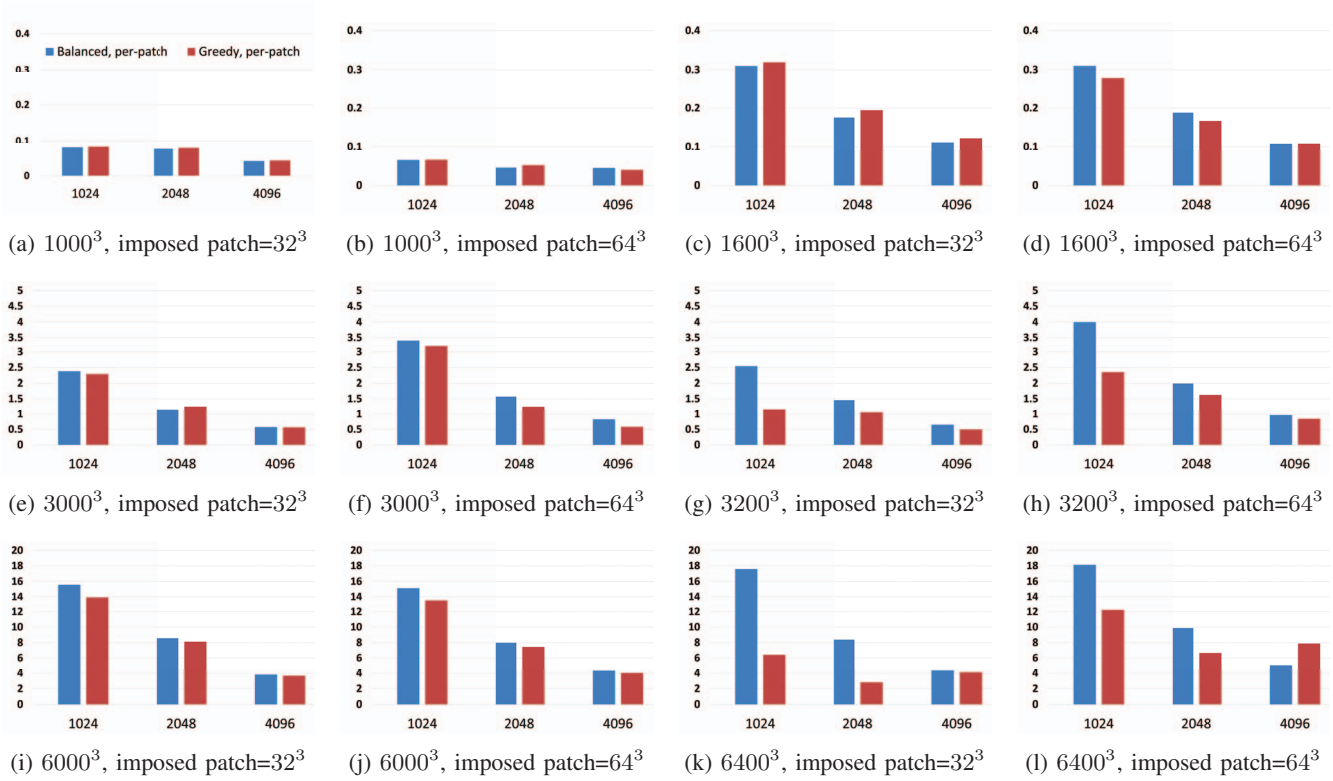
All our experiments are performed on the Theta Supercomputer [1] at the Argonne Leadership Computing Facility(ALCF). Theta is a Cray machine with a peak performance of 11.69 petaflops, 281,088 compute cores, 843.264 TiB of DDR4RAM, 70.272 TiB of MCDRAM, and 10PiB of online disk storage. The supercomputer has a Dragonfly network topology and a Lustre filesystem.

B. Experiment setup

We evaluate the efficacy of our four data repartitioning strategies: (a) greedy, per-rank, (b) greedy, per-patch, (c) balanced, per-rank, and, (d) balanced, per-patch. We measure the performance of the four strategies by recording their time to completion. We evaluate the performance of the four strategies using synthetic micro-benchmarks, that simulate real-world application scenarios.

The greedy and balanced patch placement strategies are used to assign the logical patches to their defined rank as described in the Introduction. The per-patch and per-rank communication algorithms define how the patch buffers are

2770-0135/23/$31.00 ©2023 IEEE
DOI 10.1109/HiPCW61695.2023.00020



(a) 1000^3, imposed patch=32^3 (b) 1000^3, imposed patch=64^3 (c) 1600^3, imposed patch=32^3 (d) 1600^3, imposed patch=64^3

(e) 3000^3, imposed patch=32^3 (f) 3000^3, imposed patch=64^3 (g) 3200^3, imposed patch=32^3 (h) 3200^3, imposed patch=64^3

(i) 6000^3, imposed patch=32^3 (j) 6000^3, imposed patch=64^3 (k) 6400^3, imposed patch=32^3 (l) 6400^3, imposed patch=64^3

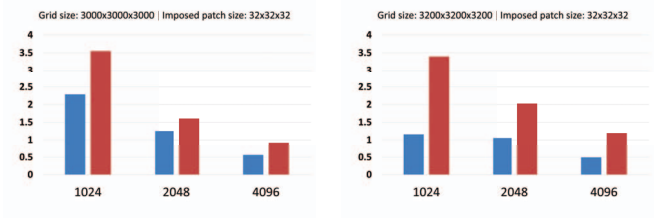
Fig. 2: Timing for repartitioning for different workloads. Blue bar is the balanced, per-patch scheme and red bar is for greedy, per-patch. X-axis for all graphs shows three process counts, 1024, 2048 and 4096. Y-axis for the graphs is time in seconds.

exchanged between ranks. Per-patch communication issues non-blocking, point-to-point MPI calls for each patch. Per-rank communication packs all patches destined for a rank into a per-rank buffer for which the rank then issues a single non-blocking, point-to-point MPI call to transfer. This exploration is motivated to measure the potential impact of smaller message sizes in the per-patch approach on communication time.

III. Results

Figure 2 shows the results for balanced and greedy placement with per-patch communication for all 12 experiments. We observe the expected benefits of strong scaling as process counts increase and the data exchange burden of individual tasks diminishes. These two methods perform similarly for global data sets sizes below $3200^3 \times 8bytes \approx 250GB$. This confirms that using balanced patch placement to facilitate workload balance for downstream tasks is an effective strategy for data sets below that size [2]. For larger data sets, greedy placement significantly reduces communication time, suggesting opportunities for early start to computation to offset the inherent data imbalance. At 4096 rank counts, performance is similar in all cases. We conclude this results from the small patch communication workload required of each rank.

We observed per-patch communication consistently outperformed the per-rank communication, highlighted in Figure 3 for two data set sizes of 3000^3 and 6000^3 for the imposed patch size of 32^3. Per-patch communication benefits from MPI



(a) 3000^3, imposed patch=32^3 (b) 3200^3, imposed patch=32^3

Fig. 3: Per-patch vs. Per-rank comm. Blue is per-patch, red is per-rank. Per-patch consistently outperforms per-rank.

requests available for transfer as patches are processed. In contrast, per-rank buffers delay the start of MPI patch exchange until rank buffer packing is complete (after reading all memory). Future work will explore the use of MPI partitioned communication to facilitate partial per-rank buffer transfers.

References

[1] Fahey et al. Theta and mira at argonne national laboratory. In *Contemporary High Performance Computing*, pages 31–61. CRC Press, may 2019.

[2] Ke Fan et al. Load-balancing parallel i/o of compressed hierarchical layouts. In *2021 IEEE 28th International Conference on High Performance Computing, Data, and Analytics (HiPC)*. IEEE, dec 2021.

[3] Sidharth Kumar et al. Efficient data restructuring and aggregation for i/o acceleration in PIDX. In *2012 International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, nov 2012.