HYPER Challenge 2021

Zürich Research Center

Huawei Technologies Switzerland

November 24, 2021

Challenge timeline

- Complete the team registration here[1] by 1st February 2022.
- Download the challenge starter kit code here[2] and datasets here[3] and code away!

  The challenge takes place in two rounds:

  - In the 1st round, all registered teams must submit their solutions by 9th January 2022. We publish the leaderboard with rankings based on the quality score of each team's latest submission.
  - In the 2nd round, all teams may update their solutions (based on their performance in the 1st round) by 6th February 2022. We publish the final leaderboard and announce the winners in **three categories**: **Quality** (3,000 CHF), **Performance** (2,000 CHF), and **Innovation** (2,000 CHF).
- The award ceremony takes place on 24th February 2022.
- All deadlines are 11:59 PM UTC-12:00 (Anywhere on Earth).

1 Motivation

Deep learning with neural networks has revolutionized the field of artificial intelligence. The state-of-the-art networks are ever-growing in size and the largest ones such as GPT-3 [1] and Megatron-LM [2] contain billions of parameters. Furthermore, current networks are becoming sparse in the search for new neural network topologies. Thus, a strong need exists for efficient parallel algorithms that handle the large scale of dense and sparse neural networks. One of the main challenges in writing such algorithms is to automatically partition neural networks, which is necessary as their size largely exceeds the storage available on a single GPU or TPU device. This problem becomes even harder when considering the network and machine parameters, the sizes of tensors passed between layers, and the computational load of each layer.

There are many different partitioning algorithms. The ones developed for **hypergraphs**, which are higher-order generalizations of graphs, have been historically used to model problems as complex as circuit partitioning for Very Large Scale Integration (VLSI) circuits [3], sparse matrix partitioning [4], and biological networks analysis [5]. They may also be leveraged for deep learning by effectively modelling a sparse neural network as a hypergraph, which has recently been shown in the context of sparse neural network inference [6]. In the HYPER Challenge 2021, we focus on novel algorithms for **hypergraph partitioning**, in the hope of simultaneously improving the quality of solutions in vastly different domains, including deep learning.

[1] https://huawei.eyevip.ch/events/hyperchallange/en/
[2] https://challenge.huaweirc.ch/HYPER_starter_kit.zip
[3] https://challenge.huaweirc.ch/HYPER_PUBLIC.zip

2 What is a hypergraph anyways?

A graph consists of a set of vertices and a set of edges, which represent relationships between a pair of vertices. A **hyperedge** generalizes the concept of an edge to model an all-to-all connection between a subset of vertices. Similarly, a **hypergraph** generalizes the notion of a graph by using hyperedges to model relationships between vertices.

Formally, a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ consists of a set of vertices $\mathcal{V}$ and a set of hyperedges $\mathcal{E}$ where every hyperedge $e \in \mathcal{E}$ is a set of vertices, i.e., $e \subseteq \mathcal{V}$. Empty hyperedges are not allowed. We focus only on unweighted and undirected hypergraphs. Let us give an example.

Example. Let hypergraph $\mathcal{H}_A = (\mathcal{V}_A, \mathcal{E}_A)$ consist of vertices $\mathcal{V}_A = \{v_1, v_2, v_3, v_4, v_5\}$ and hyperedges $\mathcal{E}_A = \{e_1 = \{v_1, v_3\}, e_2 = \{v_2, v_3\}, e_3 = \{v_4\}, e_4 = \{v_3, v_4, v_5\}\}$. Figure 1 visualizes $\mathcal{H}_A$ by representing a vertex as a circle and a hyperedge as a square and connecting each hyperedge to the vertices it contains. Figure 2 draws $\mathcal{H}_A$ as a bipartite graph, observing that any hypergraph may be viewed as such.

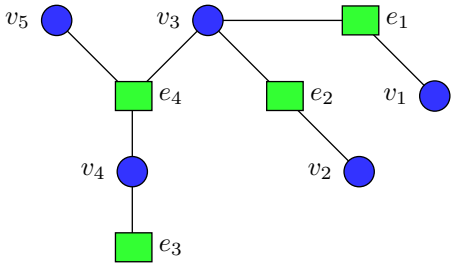

Figure 1: A visualization of hypergraph $\mathcal{H}_A$.

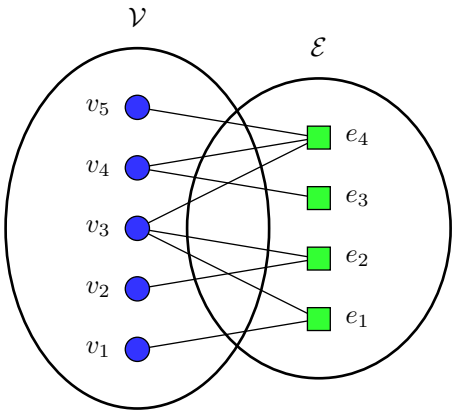

Figure 2: The bipartite graph representation of hypergraph $\mathcal{H}_A$.

3 The challenge: How to partition hypergraphs?

Let $\Pi_{\mathcal{V}}^K = \{\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K\}$ denote a K-partition[4] of vertex set $\mathcal{V}$; we typically refer to $\Pi_{\mathcal{V}}^K$ as a partition of hypergraph $\mathcal{H}$ when $\mathcal{V}$ is its vertex set. A partition $\Pi_{\mathcal{V}}^K$ is ε-**balanced** when the size of each **part** $\mathcal{V}_i$ is at most

$$\left\lfloor (1+\varepsilon)\frac{|\mathcal{V}|}{K} \right\rfloor$$

for all $1 \leq i \leq K$. One possible 2-partition of $\mathcal{V}_A$ is $\Pi_{\mathcal{V}_A}^2 = \{\{v_1, v_4\}, \{v_2, v_3, v_5\}\}$ which we visualize in Figure 3. This partition is 0.2-balanced, which we may verify by checking that the largest part $\mathcal{V}_2$ is not greater than $\lfloor (1+0.2)5/2 \rfloor = 3$.

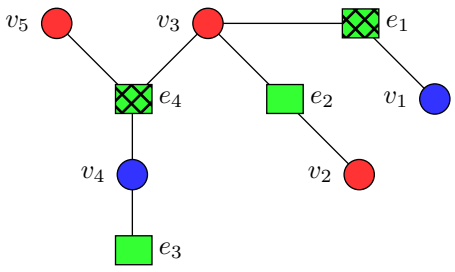

Figure 3: A visualization of 2-partition of hypergraph $\mathcal{H}_A$ consisting of parts $\mathcal{V}_1 = \{v_1, v_4\}$ (blue vertices) and $\mathcal{V}_2 = \{v_2, v_3, v_5\}$ (red vertices). The crossed edges contain vertices that belong to different parts.

[4] A partition of a set is collectively exhaustive ($\cup_{i=1}^K \mathcal{V}_i = \mathcal{V}$), and mutually exclusive ($\mathcal{V}_i \cap \mathcal{V}_j = \emptyset$ for all $i \neq j$).

Given a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, a number of parts $K \geq 1$, and an **imbalance ratio** $\varepsilon > 0$, the **hypergraph partitioning problem** is the task of finding an ε-**balanced** $\Pi_{\mathcal{V}}^K$ that minimises an objective cost function, which we define next.

Assume that **connectivity** λ_e of hyperedge e is the number of different parts of $\Pi_{\mathcal{V}}^K$ to which the vertices it contains belong to, i.e., $\lambda_e = \#\{e \cap \mathcal{V}_i \neq \emptyset \,|\, i \in \{1, \ldots, K\}\}$. We define the **cost function** using a metric known in the literature [7] as the $(\lambda-1)$ metric or the connectivity metric:

$$C(\Pi_{\mathcal{V}}^K) = \sum_{e \in \mathcal{E}} (\lambda_e - 1). \tag{1}$$

Looking at visualization in Figure 3 again, for each hyperedge count the number of distinct colors connected to it and subtract one; the cost is the sum of the results. For example, the cost of partition $\Pi_{\mathcal{V}_A}^2$ is 2 due to non-unit connectivities of hyperedges e_1 and e_4 in $\mathcal{E}_A$. In order to get a load-balanced 2-partition of a hypergraph with 5 vertices like $\Pi_{\mathcal{V}_A}^2$, ε should be at least 0.2. Commonly, ε is a small number that depends on $|\mathcal{V}|$ and K and may vary from 1% (0.01) up to 20%.

Challenge: Your task is to approximately solve the above partitioning problem. That is, given a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, a positive integer K and a small real number $\varepsilon > 0$, compute a ε-balanced partition $\Pi_{\mathcal{V}}^K$ that minimises cost $C(\Pi_{\mathcal{V}}^K)$ as defined in Equation (1).

For $K = 2$, this hypergraph partitioning problem is NP-Complete [8], while for general K the decision version of this problem (does there exist a balanced K-partition with cost less than c?) is NP-complete [3, Ch. 6]. Thus, your approach should follow an approximation or heuristic algorithm. Please refrain from developing a brute-force approach as the evaluation of your submission must complete in a limited time.

4 File Formats

A hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ can be represented as a binary 0/1 matrix. The columns correspond to the vertices and the rows correspond to the hyperedges. Each vertex is identified by its column index. The (i, j)-th entry equals 1 if $v_j \in e_i$ and equals zero otherwise. Figure 4 is the matrix representation of hypergraph $\mathcal{H}_A$.

$$\begin{array}{c} \\ e_1 \\ e_2 \\ e_3 \\ e_4 \end{array} \begin{array}{c} \begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & v_5 \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \end{pmatrix} \end{array}$$

Figure 4: An $|\mathcal{E}_A| \times |\mathcal{V}_A|$ matrix that represents hypergraph $\mathcal{H}_A$.

Input file format of a hypergraph. As the resulting matrix is sparse, we store it using the Matrix Market (MM) file format [9]. This format contains only the **one-based coordinates** of the nonzero entries. Specifically, each row contains two positive integers i and j that implies that the (i, j)-th entry is non-zero.

Example. The following are the contents of the Matrix Market file that corresponds to the hypergraph $\mathcal{H}_A$ of Figure 1:

```
%%MatrixMarket matrix coordinate pattern general
4 5 8
1 1
1 3
2 2
2 3
3 4
4 3
```

```
4 4
4 5
```

The first line of the MM format contains the number of rows (hyperedges), the number of columns (vertices), and the number of nonzero entries separated by a whitespace character. Note that the number of lines in the file must equal the number of nonzero entries excluding any optional comment lines and the header line. Please make use of the existing parsers for the MM format in order to avoid any mistakes. We provide a code skeleton in Python which uses the SciPy MM parser provided in the scipy.io module.

Output file format of a K-partition. The vertex partition $\Pi_{\mathcal{V}}^{K}$ is stored as a sparse matrix with the number of rows equal to the number of vertices $|\mathcal{V}|$ and the number of columns equal to the number of parts K, using the same file format as the input. Here, a nonzero is given by tuple (i, j) encodes that vertex v_i (corresponding to column index i in the input file) is assigned to part $\mathcal{V}_j$.

Example. The following is the content of a valid output file for partition $\Pi_{\mathcal{V}_A}^{2}$ (shown in Figure 3) of hypergraph $\mathcal{H}_A$:

```
%%MatrixMarket matrix coordinate pattern general
%
5 2 5
1 1
3 2
5 2
2 2
4 1
```

5 Getting started

To get started, you can clone the Git repository containing the code skeleton from here[5] and start developing your approach. You can develop your solution on the provided **development hypergraphs** using our evaluation script. The partitions produced by your code should have a lower cost on average than partitions produced by the baseline that uses a random partition of the vertices.

Development dataset. We provide a set of hypergraphs to you and your team to develop your solution. The file format is specified in the previous section. The hypergraphs come from various domains: knowledge representation, circuit simulation, artificial intelligence, structured and unstructured finite element methods, linear programming, web, electromagnetics, and social interactions.

For further testing as well as for contestants exploring learning-based solutions, a number of small optimally bi-partitioned ($K = 2$) hypergraphs, which were derived from sparse matrices, may be found in a sparse matrix dataset compiled by Prof. Bisseling at Utrecht University here[6].

Evaluation (hold-out) dataset. A hold-out dataset containing several hypergraphs will be used to evaluate and rank the submissions. It will not be available until the challenge has finished to avoid overfitting / fine-tuning the algorithm on these instances.

How to submit your solution

The submission process is done via a dedicated webpage which will be communicated only to registered participants. Your submission must consist of:

- A folder named src contains your code.

[5] https://challenge.huaweirc.ch/HYPER_starter_kit.zip
[6] https://webspace.science.uu.nl/~bisse101/Mondriaan/Opt/

- Step-by-step instructions to run your code. For example, if your solution is based in Python, you should provide a 'requirements.txt' file or a conda environment file. On the other hand, if your solution is written in a compiled language, you should provide a system setup, for example using docker and instructions to set up the Docker image that runs your submission and outputs the partition output file. We provide an example using docker in the starter kit. If it is not possible for the organizers to build/run your submission, you will be disqualified.
- A concise explanation of your solution.

Your submission must adhere to the following requirements:

1. You must provide your submission as a zipped file named <**team-name**>**.zip**. Maximum size on the submission is 5MB.
2. Your partitioning executable, e.g., 'hg_tools split' below, must accept **three** input arguments: (1) the absolute path of the input hypergraph, (2) the maximal load imbalance ε, and (3) the number of parts in the partition K.

```
#!/bin/bash

hg_tools split data/sample.mtx --K 2 --epsilon 0.1

# hg_tools must create a "sample.2.output.mtx" file
```

3. Partitioner must produce an output file named **hypergraph_name.K.output** where K is the number of parts specified, e.g., the output file *sample.2.output* should be written by the program above.
4. Although the provided code skeleton is written in Python, you can use any language you prefer. In particular, submissions competing on the Performance Prize (see below for different prizes) may benefit from compiled languages. The Matrix Market format parsers are available in various programming languages. For C language, the MM_IO[7] library provides functions to read and write MM files. For more information and other parsers, see the MatrixMarket[8] webpage.

The easiest way to get started is to develop your code on the baseline submission that we provide in the starter package. You may test the quality of your algorithm using the given quality script that returns the cost and average load imbalance (in percentage) of a partition. In summary, we provide the following:

1. A starter kit code in Python that samples a uniformly random ε-balanced K-partition.
2. The quality script that returns cost C (Equation 1) of a given K-partition of a hypergraph and checks that it is ε-balanced.
3. A Docker container with the Python starter kit that encapsulates the environment ensuring reproducibility of results.

Given the example hypergraph and a random 2-partition, you can check the validity of the partition using the starter kit as follows:

```
#!/bin/bash

hg_tools check data/sample.mtx sample.2.output.mtx

# <...truncated ...>
# ********************************
# * Partition imbalance : 0.2
# ********************************
# * Partition cost      : 2
# ********************************
```

[7] https://math.nist.gov/MatrixMarket/mmio-c.html
[8] https://math.nist.gov/MatrixMarket/index.html

Rules and Guidelines

1. Please note that only submissions via registered email addresses will be considered.
2. Teams may consist of up to **four** members. Each team should have a *single* registered email address.
3. Please make sure that your code always returns an ε-balanced K-partition of the input hypergraph for different values of ε and K. If a requested K-partition of a hold-out hypergraph is not ε-balanced, your submission will be penalised according to the quality score.
4. Your code must run on a single shared-memory machine; distributed-memory code will not run on multiple nodes. We provide a two-socket machine using 22-core Intel Xeon Gold 6238T 1.9 GHz processors running Ubuntu 20.04.3 LTS with g++ 9.3.0 compiler. The machine has 96 GB DRAM memory per socket.
5. Submissions that exceed the time limit, which will be fixed according to the number of contestants, will be disqualified.
6. If your code fails to run due to improper/lack of instructions, your submission will be disqualified.

6 Scoring and Prizes

Each hold-out hypergraph $i \in \{1, \ldots, n\}$ will be partitioned using your code for several values of K, namely $\{K_{i,1}, \ldots, K_{i,d(i)}\}$. We now describe the quality and performance scores we compute for each submission and the set of prizes for this competition.

Quality score. For a given hypergraph i and number of partitions K, let $C_r(i, K)$ denote the cost attained on dataset i by a fully random K-partition obeying load imbalance ratio ε, and let $C(i, K)$ denote the cost attained by your code on the same dataset provided it satisfies load balancing, and otherwise leave cost $C(i, K)$ undefined. Let $H = \bigcup_{i=1}^{n} \{(i, K_{i,1}), \ldots, (i, K_{i,d(i)})\}$ be the set of pairs of hypergraphs and the values of K for which they will be tested, and furthermore let $S \subset H$ be the subset consisting of those pairs for which your method satisfies load balancing. Assuming S is nonempty, we define the contestants score according to

$$\sum_{(i,K) \in S} \log \frac{C_r(i, K)}{C(i, K)}.$$

Thus, we sum the logarithm of the relative cost w.r.t. a random partition over those hypergraph-K pairs on which your method satisfies load balancing. The score achieved by the random partitioner itself is thus 0, by definition. For every $(i, K) \in S$ such that $C(i, K) > C_r(i, K)$, the (i, K)-th term in the score is negative, and thus submissions with a cost worse than that of a uniformly random partition are *penalised*.

Performance score. For the performance prize, only the top **five** ranked teams according to the quality score are considered. Labelling these teams by $m \in \{1, \ldots, 5\}$, their *performance score* is computed somewhat analogously to the quality score. Specifically, for $(i, K) \in H$ (where H is as in the previous paragraph), let $M(i, K) \subset \{1, \ldots, 5\}$ be the subset of these teams whose solution satisfies load balancing on hypergraph i with K partitions, and for $m \in M(i, K)$ let $T(m, i, K)$ be the run-time of team m's algorithm on hypergraph i with K partitions (in all other cases we leave $T(m, i, K)$ undefined). Now, for team $m \in \{1, \ldots, 5\}$, we define their performance score according to

$$\sum_{\substack{(i,K) \in H \text{ s.t.} \\ m \in M(i,K)}} \log \frac{\max\{T(\mu, i, K) \,|\, \mu \in M(i, K)\}}{T(m, i, K)}.$$

Provided that $M(i, K)$ contains at least two teams for at least one $(i, K) \in H$, the performance prize is given to the team with the highest performance score. If this is not the case, the performance score is equal to 0 for all 5 teams and the organizers reserve the right to devise a different rule (which will be made public) to determine the performance prize winner.

Prizes. The following three prizes will be awarded. Every team is eligible for all prize categories.

1. **Quality Prize** (3,000 CHF) will be awarded to the team with the highest quality score.
2. **Performance Prize** (2,000 CHF) will be given to the team with the highest performance score.
3. **Innovation Prize** (2,000 CHF) will be awarded to the most innovative heuristics. The selection criteria here will be in the judgment of the competition committee. To be eligible in this category, your approach should not mirror prior work.

Good luck and be a Champion!

References

[1] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, K. Melanie, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[2] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[3] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. Wiley–Teubner, Chichester, U.K., 1990.

[4] Ü. V. Çatalyürek and C. Aykanat. Hypergraph partitioning. In David A. Padua, editor, *Encyclopedia of Parallel Computing*, pages 871–881. Springer, 2011.

[5] S. Klamt, U.-U. Haus, and F. Theis. Hypergraphs and cellular networks. *PLOS Computational Biology*, 5(5):1–6, 05 2009.

[6] F. Pawłowski, R. H. Bisseling, B. Uçar, and A. N. Yzelman. Combinatorial tiling for sparse neural networks. In *2020 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7, 2020.

[7] Ü. V. Çatalyürek and C. Aykanat. *PaToH: A Multilevel Hypergraph Partitioning Tool, Version 3.3*. Bilkent University, Department of Computer Engineering, PaToH is available at https://www.cc.gatech.edu/~umit/software.html, 1999.

[8] Timon E. Knigge and Rob H. Bisseling. An improved exact algorithm and an NP-completeness proof for sparse matrix bipartitioning. *Parallel Computing*, 96:102640, 2020.

[9] R. F. Boisvert and K. A. Remington. *The Matrix Market Exchange Formats: Initial design*, volume 5935. US Department of Commerce, National Institute of Standards and Technology, 1996.