GPU Computing.
Projects and Final Report

May 24, 2024

1 Introduction

The assessment is based on the project quality+homeworks+presentation. Below, we describe what you need to do concerning the project report.

A group of two students (up to) can choose one of the proposed projects.

Students must send the report via email **by three days before** the exam **a report**. The report must satisfy the following requirements.

1. Authors: Student Name, Surname, ID, email
2. Maximum 4 pages (references do not count!!!)
3. References.
4. Platform and computing system description.
5. GIT and instruction for the reproducibility
6. Contribution: a section where each student describes his/her contribution.
7. Format: use IEEE conference template (2-column format, main text 10pt).

Disclaimer, if the report does not match one or more requirements it will NOT be evaluated.

1.1 Report organization

Recommendations on how to organize the report. Each report can contain the following sections (On top of the mandatory sections):

Abstract

- Summary of the project, highlighting key objectives, methods, and findings.

Introduction of the problem and importance

- Describe the background and importance of the problem your project addresses.
- Explain the relevance of the project in the context of parallel computing.
- Outline the main objectives of your project.

State-of-the-art

- Review current research and developments related to your project.
- Discuss existing solutions and their limitations.
- Identify the gap your project aims to fill.

Contribution and Methodology

- Elaborate on the unique contributions of your project.
- Describe the methodology used in your project, including algorithms, data structures, and parallelization techniques.
- Discuss the challenges faced and how they were addressed.

Experiments and System Description

- Detailed description of the computing system and platform.
- Relevant specifications or configurations (e.g., libraries and programming toolchains).
- Description of the experimental setup, procedures, and methodologies used in the project.
- Discussion on how experiments are designed to test the hypotheses or achieve the objectives.

Results and Discussion

- Presentation of results.
- Analysis and interpretation in context.
- Comparison with the state-of-the-art.

Conclusions

- Summary of key findings and contributions.
- Discussion on impact and future work

Contribution

- Individual contribution descriptions by each student.

GIT and Instructions for Reproducibility

- GIT repository link the point to the code and the detailed instructions for reproducing results (README).

This year the presentation part is not necessary.

2 Project list for who complete all the deliverables

2.1 Fundamental Algorithm

This project consists of extending your deliverables based on Matrix transposition with further optimizations and problem variations. All the deliverable must be included in only 1 document to deliver by the date of the exam. The different problems are related to exploiting advanced technologies for single-GPUs, dealing with sparse matrices or dealing with very large matrices that requires more than one GPU.

2.1.1 Dense Matrix Transposition

We assume that you have implemented a tile-based transposition. The algorithm is limited by the data movement, so you need to improve the access pattern and reducing the latency. Possible optimization should consider warp-based routine (shuffle) and Tensor Memory Accelerator (link) or other technologies like cooperative groups and dynamic parallelism.

The goals are:

- Design a newer tile-based algorithm by using one of the technologies described above.
- Compute the effective bandwidth and presenting the improvement of the various kernels in terms of bandwidth wrt the theoretical peak.
- Compare your code with vendor's library (e.g., CuBLAS).

2.1.2 Sparse Matrix Transposition

This project requires to design an efficient algorithm to transpose a sparse matrix. Specifically the matrix should be highly sparse, namely the number of zero element is more that 75% of the whole $(n \times n)$ elements. The implementation should emphasize:

- storage format for storing sparse matrices (for example, compressed sparse row);
- the implementation to perform the transposition;
- a comparison against vendors' library (e.g., cuSPARSE);
- dataset for the benchmark (compare all the implementation presented by selecting at least 10 matrices from suite sparse matrix collection https://sparse.tamu.edu/);

As usual, the metric to consider is the effective bandwidth.

2.1.3 Multi-GPU Matrix Transposrition

The goal of the project is to develop an efficient matrix transposition algorithm utilizing multiple GPUs to achieve high performance and scalability. The project aims to understand and implement advanced parallel computing techniques to handle large datasets. This project should emphasize Multi-GPU strategy.

1. Divide a dense matrix into sub-matrices to distribute across multiple GPUs.
2. Implement data partitioning and distribution strategies.
3. Handle GPU-GPU communication efficiently using MPI (cuda-aware) or NCCL.
4. Show the strong scaling up to 4 GPUs.

3 Project list

The abstract contains general information to catch your interest. Discuss in detail the objectives and further details with the Professor or the tutor who will be assigned to you.

3.1 Fundamental routine and runtime

3.1.1 Batched Matrix Multiplication on CUDA Tensor Cores

This project proposes the optimization of Batched matrix multiplication algorithms using CUDA and Tensor Cores. The ability to compute many (typically small) matrix-matrix multiplies at once, is known as batched matrix multiply. The objective is to leverage the parallel processing capabilities of GPUs and the specialized matrix computation features of Tensor Cores to achieve high performance in a large number of batched matrix operations.

The challenge lies in efficiently mapping the multiple batches matrices to fully utilize the architecture of modern GPUs, particularly focusing on the efficient use of shared memory and minimizing communication overhead across the memory hierarchy. Special attention will be given to the alignment and distribution of data batches through dense blocks to optimize Tensor Core utilization. The algorithm should take a list of blocks, identified by 4 coordinates. Each block may have a different size therefore fore the scheduling and the workload balance are particularly challenging. The experiments should consider state-of-the-art libraries such as *cublasgemmBatched* [1] and CUTLASS and different configurations of the batched GEMM (number of small matrices, distribution of the sizes etc...).

Effort distribution is anticipated as 70% coding, focusing on CUDA kernel development and performance tuning, and 30% theory, involving algorithmic design and performance analysis.

The project aims to demonstrate significant performance improvements over traditional GPU-based matrix multiplication methods. Success will be measured by benchmarking against existing implementations.

References [ATD19, VD08].

[1] https://developer.nvidia.com/blog/cublas-strided-batched-matrix-multiply/.

3.1.2 Parallel Sorting Algorithms

The goal of this project is to parallelize a sorting algorithm over GPU (CUDA). Sorting is a fundamental operation in computer science, and enhancing its performance through parallel computing techniques is a crucial aspect. The project must include the following tasks:

1. Implement the sequential, and the parallel versions and compare them.
2. Evaluate the speedup achieved by the parallel sorting algorithm for different array sizes and numbers of threads.
3. Assess the impact of load balancing on performance in the CUDA version.
4. Analyze the scalability of the GPGPU parallel sorting algorithm on the cluster with varying numbers of processes.
5. Implement a dynamic load balancing strategy for the parallel sorting algorithm and analyze its impact on performance.
6. Explore strategies for optimizing performance, such as parallelizing specific stages of the sorting algorithm.

The project will be divided into 70% coding, focusing on algorithm implementation and performance tuning, and 30% theory, involving analysis of algorithmic design and performance.

References [Akl85, CLRS09].

3.1.3 Investigation of memory pool, benchmark and evaluation

Many algorithms and workloads require to iterate operations. The number of iteration often is not predictable. A specific subroutine may require to allocate memory on different size for each iteration. The cost of the memory allocation is particularly expensive especially if you use cudaHostAlloc. This project requires to design a synthetic benchmark to evaluate different approaches and in particular asynchronous memory allocations. The use of advanced profiler tools is required.

References CUDA Programming Guide, CUDA API Reference Guide

3.2 Scientific Computing

3.2.1 Minimum Weight Perfect Matching (MWPM)

A matching in a graph G = (V, E) is a subset of edges M such that no two elements of M have common endpoints. For an unweighted graph the objective is typically to compute a matching of maximum cardinality. In an edge weighted graph the objective is to compute a matching such that the sum of the weights of the elements in M is maximum. The sparse blossom algorithm is an efficient algorithm for finding minimum-weight paths between detection events in a detector graph, developed by Adam Bouland and Craig Gidney to solve the decoding problem in quantum error correction. It is a variant of the classic blossom algorithm that uses a layered forest data structure to represent the connectivity of the graph and efficiently identify and contract odd-length cycles called blossoms. The algorithm is called ”sparse” because it works well for sparse graphs and has a worst-case time complexity of $O(n^2 \log n)$ and a worst-case space complexity of O(n), where n is the number of vertices in the graph. The algorithm has several advantages over the original blossom algorithm, including its exactness, suitability for parallelization, and ability to handle sparse graphs efficiently.

The project should also discuss the differences wrt to other algorithm like Suitor.

References [MH14]

3.2.2 Sparse Matrix-Vector Multiplication on multi-GPU

This project proposes optimizing Sparse Matrix-Vector Multiplication (SpMV) on distributed-memory clusters utilizing multi-GPU. SpMV is pivotal in scientific computations but faces challenges due to memory bandwidth and irregular access patterns in sparse matrices. The aim is to leverage multi-GPU’s features and parallel capabilities of distributed architectures for enhanced SpMV performance.

Focus areas include efficient data distribution, communication optimization, and exploitation of multi-GPU. The project involves developing parallel algorithms, performance tuning, and benchmarking against existing solutions. Anticipated outcomes include significantly improved SpMV efficiency and scalability, contributing to high-performance computing advancements. This endeavor is suitable for students interested in parallel computing and computer architecture, offering insights into emerging processor technologies and distributed computing paradigms. For the matrix partitioning it is recommended the usage of Metis or ParMeTIS [KSK97]. The access to a multi-GPU cluster will be provided.

References [GMFC21, RSI23, KSK97] The project will be divided into 60% coding, involving parallel algorithm development and performance tuning, and 40% theory, focusing on algorithmic design and performance analysis. For the dataset, please consider Suite Sparse Matrix Collection repository `https://sparse.tamu.edu/`

3.3 AI and Data Analytics

3.3.1 Clustering algorithms on GPUs

Iterative clustering is a technique used in unsupervised machine learning to cluster similar data points together. It involves iteratively partitioning the data points into groups, based on their similarity, until a satisfactory clustering is achieved.

The iterative clustering process typically starts with an initial set of cluster centres, which can be randomly selected or obtained through some other means. Then, each data point is assigned to the nearest cluster centre based on some distance metric, such as Euclidean distance or cosine similarity (pseudocode gives an example).

Algorithm 1 Iterative clustering

```
Input: set of vectors $V$, threshold $\tau$, number of centroids per iteration $k$ and a distance $d_J(\cdot,\cdot)$
Output: set of clusters of vectors $\mathcal{C}$
$\mathcal{C} \leftarrow \emptyset$
while $V \geq k$ do
    $C_1 \leftarrow$ new $,\ldots, C_k \leftarrow$ new        // initialize $k$ new empty clusters
    select $c_1,\ldots,c_k \in V$        // select $k$ centroids
    for $v \in V$ do
        $j \leftarrow argmin_{1\leq i\leq k}\hat{d}_J(c_i, v)$        // find closest centroid $c_j$
        if $\hat{d}_J(c_j, v) \leq \tau$ then
            // if $c_j$ is close enough
            $V \leftarrow V \setminus v$        // ... move $v$ to $C_j$
            $C_j \leftarrow C_j \cup v$
    $\mathcal{C} = \mathcal{C} \cup C_1,\ldots,C_k$        // add new clusters to $\mathcal{C}$
for $v \in V$ do
    // add remaining vectors as singleton clusters
    $\mathcal{C} \leftarrow \mathcal{C} \cup \{v\}$
return $\mathcal{C}$
```

In this project, we want to investigate the advantages and the algorithmic problems of using GPU parallelism to speeding up iterative clustering. Additionally, students are encouraged to explore estimators and heuristics for optimizing the selection of comparison similarity patterns and centroid choices.

We recommend developing the new implementations using OpenACC. By doing so, students can give more attention to investigating algorithmic challenges rather than focusing on low-level implementation details.

References [LBN$^+$22]

3.3.2 Simple implementation of local sensitive hashing algorithm

Short description: Finding near points inside a huge point's dataset becomes computationally harder as the dataset size grows. Given a P dataset of n points and the goal of finding all the couples within a certain distance α, solving this problem by comparing all the couple of points has a complexity of $O(n^2)$ operations. The local sensitive hash (LSH) are randomised dimension reduction functions which allows us to solve the Nearest Neighbor Search with a smaller computational complexity. The simplest LSH consists in generating random hyperplanes that partition the dataset; all the points that lie on the same partition subset have much more probability to be near.

This project has the aim of implementing one or more GPU LSH algorithms and comparing It with naive and sequential algorithms.

References [IM98]

3.3.3 Graph Neural Network from scratch

A graph neural network (GNN) belongs to a class of artificial neural networks for processing data that can be represented as graphs. The first part of the project involves the writing of a small library (or set of classes) that perform operations on dense and sparse COO matrices. The second part involves writing an end-to-end library for GNN training and inference.

Evaluate your code on a simple network dataset e.g. Zachary's Karate Club. Find the most expensive function/routine and parallelize it.

References [Afl, Men] `https://github.com/HicrestLaboratory/rnwzd-sparse-matrices`

3.3.4 Parallel Image Processing

In this final project, the objective is to parallelize image processing operations, for example, image filtering algorithms, using shared memory (CUDA).

The first step consists of implementing an image-filtering algorithm. To do it, for each pixel in the output image, apply the convolution operation by multiplying the pixel values in the local neighborhood of the corresponding pixel in the input image with the corresponding filter coefficients and summing up the results. To deal with the parallel algorithm, it is necessary to implement a suitable boundary-handling strategy to handle edge pixels, as they require special treatment during the convolution operation. Some well-known strategies are zero-padding, mirror padding, or wrapping around the image.

The project must include the following tasks:

1. Read the input image from a file.
2. Implement a serial image filtering algorithm using the given convolution filter.
3. Parallelize the image filtering algorithm using CUDA for shared memory systems, distributing the workload among available threads.
4. Evaluate the speedup achieved by the parallel image filtering algorithm using CUDA for different image sizes and numbers of threads and assess the impact of load balancing on performance in the CUDA version.
5. Analyze the scalability of the parallel image filtering algorithm on the clusters using different numbers of processes in the CUDA version.
6. Implement strategies for optimizing performance, such as overlapping communication and computation, in the CUDA version and explore the use of different convolution filter sizes and characteristics to assess algorithm adaptability.

The project will be divided into 60% coding, involving parallel algorithm development and performance tuning, and 40% theory, focusing on algorithmic design and performance analysis.

References [BFRR01, GW18].

3.4 LLM Optimization

This final project aims to optimize LLM routines using GPGPU programming with OpenACC or CUDA. Given the reference code available at `https://github.com/karpathy/llm.c`, students need to benchmark the baseline code, identify the most time-consuming routines, and implement parallel optimizations.

The project must include the following tasks:

1. Detailed benchmark and bottleneck analysis of the baseline code.
2. Accurate algorithm description and motivation for the designed optimizations.
3. Implementation of at least one or two routine optimizations.
4. Achieved speed-up analysis.

The evaluation will be based on the following criteria: 50% performance analysis, 30% coding, and 20% theoretical understanding.

References

[Afl] Omar Aflak. Neural Network from scratch in Python — towardsdatascience.com. https://towardsdatascience.com/math-neural-network-from-scratch-in-python-d6da9f29ce65. [Accessed 06-05-2024].

[Akl85] S.G. Akl. *Parallel Sorting Algorithms*. Notes and reports in computer science and applied mathematics. Academic Press, 1985.

[ATD19] Ahmad Abdelfattah, Stanimire Tomov, and Jack Dongarra. Fast batched matrix multiplication for small sizes using half-precision arithmetic on gpus. In *2019 IEEE international parallel and distributed processing symposium (IPDPS)*, pages 111–122. IEEE, 2019.

[BFRR01] T. Braunl, S. Feyrer, W. Rapf, and M. Reinhardt. *Parallel Image Processing*. Springer, 2001.

[CLRS09] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. Computer science. MIT Press, 2009.

[GMFC21] Constantino Gómez, Filippo Mantovani, Erich Focht, and Marc Casas. Efficiently running spmv on long vector architectures. In *Proceedings of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 292–303, 2021.

[GW18] R.C. Gonzalez and R.E. Woods. *Digital Image Processing*. Pearson, 2018.

[IM98] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing (STOC '98)*, pages 604–613, New York, NY, USA, 1998. Association for Computing Machinery.

[KSK97] George Karypis, Kirk Schloegel, and Vipin Kumar. Parmetis: Parallel graph partitioning and sparse matrix ordering library. 1997.

[LBN$^+$22] Paolo Sylos Labini, Massimo Bernaschi, Werner Nutt, Francesco Silvestri, and Flavio Vella. Blocking sparse matrices to leverage dense-specific multiplication. In *2022 IEEE/ACM Workshop on Irregular Applications: Architectures and Algorithms (IA3)*, pages 19–24, 2022.

[Men] Amal Menzli. Graph Neural Network and Some of GNN Applications: Everything You Need to Know. https://neptune.ai/blog/graph-neural-network-and-some-of-gnn-applications. [Accessed 06-05-2024].

[MH14] Fredrik Manne and Mahantesh Halappanavar. New effective multithreaded matching algorithms. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, pages 519–528, 2014.

[RSI23] Alexandre Rodrigues, Leonel Sousa, and Aleksandar Ilic. Performance modelling-driven optimization of risc-v hardware for efficient spmv. In *International Conference on High Performance Computing*, pages 486–499. Springer, 2023.

[VD08] Vasily Volkov and James W Demmel. Benchmarking gpus to tune dense linear algebra. In *SC'08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–11. IEEE, 2008.