Introduction to parallel computing. Report and project preparation

December 15, 2023

1 Introduction

The assessment is based on the project quality+homeworks+presentation. Below, we describe what you need to do concerning the project report (which is the most important part, check the weights of make in the presentation of the course on Moodle).

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).

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 will be assigned to you.

2 Fundamental routine and runtime

2.1 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 *cublasT gemmBatched* [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/.

2.2 Parallel Sorting Algorithms

The goal of this project is to parallelize a sorting algorithm using both shared memory (OpenMP) and distributed memory (MPI) paradigms. 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 OpenMP version.
4. Analyze the scalability of the MPI 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 Scientific Computing

3.1 Sparse Matrix-Vector Multiplication on RISC-V Based Distributed-Memory Clusters

This project proposes optimizing Sparse Matrix-Vector Multiplication (SpMV) on distributed-memory clusters utilizing RISC-V processors. 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 RISC-V's features and parallel capabilities of distributed architectures for enhanced SpMV performance. Focus areas include efficient data distribution, communication optimization, and exploitation of RISC-V's modularity. 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 RISC-V 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/`

3.2 Parallelized Solution of 1D Heat Equation

The goal of this project is to parallelize the numerical solution of the 1D heat equation using both shared memory (OpenMP) and distributed memory (MPI) paradigms. The 1D heat equation describes the evolution of temperature in a one-dimensional rod over time.

The 1D heat equation is given by:

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$$

where $u(x, t)$ is the temperature distribution, α is the thermal diffusivity, x is the spatial coordinate and t is the time.

The project must include the following tasks:

1. Implement a sequential code for solving the 1D heat equation using finite difference methods (explicit or implicit method) as a baseline.
2. Parallelize the code using OpenMP directives to distribute the computation across multiple threads. Consider parallelizing the spatial loop.
3. Parallelize the code using MPI to distribute the computation across multiple processes. Each process should handle a portion of the spatial domain.
4. Measure and compare the execution time of the sequential, OpenMP, and MPI versions for different spatial and temporal resolutions.
5. Analyze the achieved speedup and efficiency in the parallel implementations.
6. Evaluate the scalability of the MPI code by varying the number of processes and observing the impact on execution time.
7. Try different load balancing strategies in the OpenMP code to optimize thread utilization and explore asynchronous communication and overlapping computation and communication in the MPI code.

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 [Smi85, Wil06].

4 AI and Data Analytics

4.1 Betweenness Centrality Algorithms

This project proposal focuses on the parallelization of Betweenness Centrality (BC) [Bra01], a key metric in graph theory used to measure the importance of vertices within a graph. BC computation is computationally intensive, particularly for large-scale graphs, making it a prime candidate for parallelization. The objective is to develop and implement efficient parallel algorithms for BC computation on distributed systems, aiming to significantly reduce computation time and enhance scalability.

The challenge lies in effectively decomposing the BC computation task for parallel execution while minimizing communication overhead and balancing workload across multiple nodes. The project will explore strategies for graph partitioning, distributing the workload of shortest path calculations, and aggregating partial BC scores from distributed computations. These strategies must be carefully designed to ensure the correctness of the final BC values while maximizing parallel efficiency.

The project involves a balanced mix of theoretical algorithmic design and practical implementation. Approximately 50% of the effort will be dedicated to the development of parallel BC algorithms, focusing on optimizing data structures and communication patterns for distributed environments. The remaining 50% will involve hands-on implementation and performance tuning in a parallel computing environment, using programming frameworks suitable for distributed systems, such as MPI (Message Passing Interface).

Success will be measured by benchmarking the parallel BC implementation against traditional single-node BC computations, with metrics including speedup, scalability, and computational efficiency. This project is ideal for students with interests in graph algorithms, parallel computing, and distributed systems, offering valuable experience in tackling computational challenges in graph analytics using parallelization techniques.

For this project, you can consider GPUs. In case you want to implement it in GPU, we consider only shared-memory systems. For the graph partitioning it is recommended the application of 2D decomposition [CHY05]. For the dataset, please consider SNAP repository `https://snap.stanford.edu/data/` Other simpler algorithms can be considered (e.g., BFS [CHY05]).

4.2 Scale dot product attention

This project proposal aims to parallelize the computation of scaled dot-product attention, a crucial component of transformer models [VSP+17] widely used in natural language processing and other

AI applications. Scaled dot-product attention, responsible for capturing dependencies irrespective of their distance in input sequences, is computationally intensive and becomes a bottleneck in large-scale models or datasets. The goal is to develop and implement a parallel algorithm that efficiently computes scaled dot-product attention, significantly improving the performance and scalability of transformer models.

Algorithm 1 Scaled Dot-Product Attention

```
Require: $Q$ (Query matrix)
Require: $K$ (Key matrix)
Require: $V$ (Value matrix)
Require: $d_k$ (Dimension of Key matrix)
Require: $mask$ (optional)
  $score \leftarrow \frac{QK^T}{\sqrt{d_k}}$
  if mask is not None then
    $score \leftarrow score + mask$
  end if
  $attentionWeights \leftarrow \mathrm{softmax}(score, \mathrm{axis} = -1)$
  $output \leftarrow attentionWeights \times V$
  return $output$
```

The primary challenge is to design a parallelization strategy that effectively distributes the computation of attention mechanisms across multiple processors while maintaining the accuracy and integrity of the model. The project will explore techniques for decomposing attention calculations, optimizing data transfer and synchronization between processors, and minimizing latency in multi-node environments. Special attention will be paid to memory optimization and reducing the computational redundancy inherent in attention mechanisms.

The project will involve a mix of theoretical and practical work. Approximately 60% of the effort will focus on coding and implementation, which includes developing parallel algorithms and optimizing them for specific hardware architectures like GPUs or distributed clusters. The remaining 40% will be dedicated to theoretical aspects, including algorithmic design, analysis of parallel efficiency, and scalability.

Success will be measured by benchmarking the parallelized attention mechanism against traditional, non-parallel implementations, with a focus on metrics such as computation time, scalability, and resource utilization efficiency. This project is well-suited for students and researchers interested in parallel computing, machine learning, and specifically transformer architectures. The expected outcome is a significant enhancement in the computational efficiency of transformer models, contributing to the advancement of AI applications that rely on large-scale language models.

References [RCHG+23].

4.3 k-Means

The main goal of this project is to implement and parallelize the k-Means clustering algorithm. k-Means is an unsupervised classification (clustering) algorithm that groups objects into k distinct, non-overlapping subsets (clusters) based on their characteristics. The clustering is performed by minimizing the sum of distances between each object and the centroid of its cluster. The Euclidean distance is usually used. The algorithm consists of three steps:

1. Initialization: the number of clusters, k, is fixed, and k data points are set in the data space as initial centroids, e.g., by choosing them randomly
2. Assignment of elements to each cluster: each data is assigned to its nearest centroid
3. Updating centroids: once the data is distributed, the position of the centroid of each cluster is updated, taking as new centroid the position of the average of the objects belonging to that cluster

Steps 2 and 3 are repeated until it converges, i.e., the centroids do not move or move below a threshold distance in each step, or a maximum number of iterations is reached.

The project must include the following tasks:

1. Describe the problem in detail, explaining carefully the parallel approach that you have used.
2. Measure and compare the execution time of the serial, OpenMP, and MPI versions for different dataset sizes and dimensions.
3. Analyze the achieved speedup and efficiency in the parallel implementations, discussing scalability issues and possible optimizations.
4. Experiment with load balancing strategies in the OpenMP version to optimize thread utilization.
5. Explore asynchronous communication and overlapping computation and communication in the MPI version.

The project will be divided into 50% coding, focusing on algorithm implementation and performance tuning, and 50% theory, involving analysis of algorithmic design and performance.

References [AV06, LY09].

4.4 Parallel distributed Local Sensitive Hashing

This project proposal focuses on the parallelization of Locality-Sensitive Hashing (LSH) [JMN+21], a widely used algorithm in large-scale similarity search and data retrieval tasks. LSH is instrumental in efficiently identifying similar items in high-dimensional data spaces by hashing them into buckets with a high probability of containing similar items. However, as data scales grow, the computation becomes increasingly intensive, making parallel processing essential for timely and efficient performance.

The primary goal is to develop a parallel LSH algorithm that significantly reduces computation time and resource consumption while maintaining or improving accuracy in similarity search tasks. The project involves designing parallelization strategies that effectively distribute hashing and bucketing operations across multiple processors or nodes. Special attention will be paid to optimizing data partitioning, minimizing inter-processor communication, and balancing workload to enhance overall efficiency.

The project is structured with an approximately 70% focus on practical implementation and 30% on theoretical analysis. The implementation phase involves coding the parallel LSH algorithm, likely using GPU programming with CUDA. The theoretical phase includes the design and analysis of the parallel LSH algorithm, ensuring effective load balancing and minimal communication overhead.

Success will be measured through benchmarks comparing the parallel LSH implementation with traditional, sequential LSH algorithms in terms of speed, scalability, and accuracy in similarity search tasks or other parallel implementations. This project is ideal for students in computer science with an interest in data mining, machine learning, and parallel computing. The expected outcome is an efficient, scalable parallel LSH algorithm that can handle large-scale data, significantly contributing to the fields of data retrieval and big data analytics.

For testing the performance of Locality-Sensitive Hashing (LSH) algorithms, a representative dataset can be found in the ANN datasets repository on GitHub at the following URL `https://ann-benchmarks.com/`.

4.5 Parallel Graph Algorithms

In this project, the objective is to parallelize graph traversal algorithms using both shared memory (OpenMP) and distributed memory (MPI) paradigms. The focus will be on parallelizing graph traversal algorithms such as breadth-first search (BFS) and depth-first search (DFS).

The project must include the following tasks:

1. Implement the sequential BFS and DFS algorithms to obtain a baseline for performance comparison.
2. Parallelize both algorithms using OpenMP for shared memory systems and using MPI for distributed systems.
3. Measure and compare the execution time of the serial, OpenMP, and MPI versions for different graph sizes.

4. Analyze the achieved speedup and efficiency in the parallel implementations.
5. Try different load balancing strategies in the OpenMP version to optimize thread utilization.
6. Evaluate the scalability of the MPI version by varying the number of processes and observing the impact on execution time. Explore asynchronous communication and overlapping computation and communication in the MPI version.

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

References [CLRS09, BM11, KR87, RK96].

4.6 Parallelizing Traffic Flow Simulation using Cellular Automaton

The main purpose of this project is to parallelize a cellular automaton model to simulate traffic flow. Cellular automata are discrete, computational models widely used to simulate complex systems. In this case, we aim to model and parallelize the dynamics of traffic flow on a road network using a cellular automaton.

The project must include the following tasks:

1. Implement the sequential and the parallel versions.
2. Measure and compare the execution time of the serial and parallel versions for different road grid sizes and traffic densities.
3. Analyze the achieved speedup and efficiency in the parallel implementation.
4. Evaluate the scalability of the parallel version by varying the number of threads or processes and observing the impact on execution time.
5. Experiment with load balancing strategies in the parallel version to optimize thread or process utilization.
6. Explore fine-grained and coarse-grained parallelization approaches.

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 [Tre13].

4.7 Parallel Image Processing

In this final project, the objective is to parallelize image processing operations, for example, image filtering algorithms, using both shared memory (OpenMP) and distributed memory (MPI) paradigms.

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 OpenMP for shared memory systems, distributing the workload among available threads.
4. Parallelize the image filtering algorithm using MPI for distributed memory systems, distributing the workload among different processes.

5. Evaluate the speedup achieved by the parallel image filtering algorithm using OpenMP for different image sizes and numbers of threads and assess the impact of load balancing on performance in the OpenMP version.

6. Analyze the scalability of the parallel image filtering algorithm on the clusters using different numbers of processes in the MPI version.

7. Implement strategies for optimizing performance, such as overlapping communication and computation, in the MPI 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].

References

[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.

[AV06] D. Arthur and S. Vassilvitskii. How slow is the k-means method? In *Proceedings of the Twenty-Second Annual Symposium on Computational Geometry*, pages 144–153, New York, NY, USA, 2006. Association for Computing Machinery.

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

[BM11] A. Buluç and K. Madduri. Parallel breadth-first search on distributed memory systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. Association for Computing Machinery, 2011.

[Bra01] Ulrik Brandes. A faster algorithm for betweenness centrality. *Journal of mathematical sociology*, 25(2):163–177, 2001.

[CHY05] Edmond Chow, Keith Henderson, and Andy Yoo. Distributed breadth-first search with 2-d partitioning. Technical report, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2005.

[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.

[JMN$^+$21] Omid Jafari, Preeti Maurya, Parth Nagarkar, Khandker Mushfiqul Islam, and Chidambaram Crushev. A survey on locality sensitive hashing algorithms and their applications. *arXiv preprint arXiv:2102.08942*, 2021.

[KR87] Vipin Kumar and V. Rao. Parallel depth first search. part ii. analysis. *International Journal of Parallel Programming*, 16:501–519, 1987.

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

[LY09] D. Liu and J. Yu. Otsu method and k-means. In *2009 Ninth International Conference on Hybrid Intelligent Systems*, volume 1, pages 344–349, 2009.

[RCHG$^+$23] Luka Ribar, Ivan Chelombiev, Luke Hudlass-Galley, Charlie Blake, Carlo Luschi, and Douglas Orr. Sparq attention: Bandwidth-efficient llm inference, 2023.

[RK96] V. Rao and Vipin Kumar. Parallel depth first search, part i: Implementation. *International Journal of Parallel Programming*, 16, 1996.

[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.

[Smi85] G.D. Smith. *Numerical Solution of Partial Differential Equations: Finite Difference Methods.* Oxford applied mathematics and computing science series. Clarendon Press, 1985.

[Tre13] M. Treiber. *Traffic Flow Dynamics: Data, Models and Simulation.* Springer, 2013.

[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.

[VSP$^+$17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[Wil06] P. Wilkinson. *Parallel Programming: Techniques And Applications Using Networked Workstations And Parallel Computers, 2/E.* Pearson Education, 2006.