Optimizing Rodinia Benchmarks with Nvidia Libraries

DD2360 Final Project Report - Group 7

https://github.com/verach3n/DD2360HT22Project.git

Sathvik Joel Kothapalli (sjko@kth.se)
Yujie Chen (yujiec@kth.se)
Zihao Yan (zihaoy@kth.se)

Summary of each group member’s contribution

Sathvik Joel Kothapalli:

- Re-implemented the combination of lud and gaussian files in the Rodinia benchmark using the cuSOLVER library, and profiled them.
- Wrote some of the report.

Yujie Chen:

- Re-implemented the lud and gaussian files in the Rodinia benchmark using the cuBLAS library, and profiled them.
- Wrote the ‘README.md’ file and some of the report.

Zihao Yan:

- Prepared slides and gave the presentation

1 Introduction

1.1 Motivation

The use of vendor-provided libraries, such as NVIDIA's cuBLAS and cuSOLVER, can significantly improve the performance of scientific and engineering applications that require heavy computation. In this report, we will explore the use of these libraries and learn how to incorporate them into our code. We will also identify potential configurations such as the GPU architecture etc., that may impact performance. As a case study, we will rewrite the LU decomposition and Gaussian elimination routines from the Rodinia benchmark tests using cuBLAS and cuSOLVER and compare the performance of these implementations to the original versions. By doing so, we hope to gain a deeper understanding of the capabilities and limitations of these libraries and how to effectively utilize them to achieve optimal performance.

1.2 Library Introduction

1.2.1 cuBLAS

cuBLAS is a GPU-accelerated linear algebra library that is part of the NVIDIA CUDA Toolkit. It offers a set of highly optimized functions for matrix-matrix and matrix-vector multiplication, as well as other basic linear algebra operations, which can be used to accelerate scientific and engineering applications that require heavy computation on matrices and vectors. The library features a user-friendly API with function calls similar to those of the standard BLAS library, allowing for easy incorporation into existing applications with minimal code changes. cuBLAS is optimized for NVIDIA GPUs and offers significant performance improvements over

CPU-based implementations of linear algebra operations. It also provides support for a range of precision levels, from single to half precision, enabling developers to trade off accuracy for increased performance as needed. In summary, cuBLAS is a valuable tool for developers seeking high-performance linear algebra computations on NVIDIA GPUs, with the potential to significantly enhance the performance of a wide range of scientific and engineering applications..

1.2.2 cuSOLVER

cuSOLVER is a GPU-accelerated library that provides a range of linear algebra and eigenvalue solvers as part of the NVIDIA CUDA Toolkit. It offers optimized implementations of dense and sparse linear system solvers, matrix factorizations, and eigenvalue decompositions, among other common linear algebra operations.

The library features a user-friendly API with function calls similar to those of traditional linear algebra libraries and is well-suited for scientific and engineering applications, offering support for multiple precision levels and efficient handling of large, sparse matrices. In summary, cuSOLVER is a valuable tool for developers seeking to solve linear algebra problems on NVIDIA GPUs and can significantly enhance the performance of a wide range of scientific and engineering applications.

2 Methodology

The goal of this project was to optimize the performance of the Rodinia benchmarks using Nvidia libraries.

To achieve this goal, we followed the following methodology:

1. Identify the benchmarks that are suitable for optimization with Nvidia libraries. We chose the **lud** and **gaussian** benchmarks from the Rodinia suite for optimization. These benchmarks were selected because they are representative of the types of computations that are commonly performed on GPUs and are expected to benefit from GPU acceleration. We also considered the complexity and performance potential of these benchmarks when making the selection.
2. Rewrite the benchmarks using Nvidia libraries. We used the cuSOLVER and cuBLAS libraries to rewrite the lud and gaussian benchmarks. We followed best practices for optimizing GPU code, such as minimizing data transfers between the host and device, using shared memory to reduce global memory accesses, and exploiting parallelism with thread blocks and warps.
3. Profile the original and optimized benchmarks. We used the Nvidia Profiler and other performance analysis tools to measure the kernel time and data transfer time of the original and optimized benchmarks. We collected performance metrics such as kernel time and memory access patterns to identify bottlenecks and opportunities for optimization.
4. Compare the kernel time of the original and optimized benchmarks. We ran the lud and gaussian benchmarks with various input sizes and measured the kernel

time of the original and optimized versions and found the bottleneck of rodinia code. The libraries cuBLAS and cuSOLVER were used to compare the performance of the matrix decomposition and linear system solver kernels, respectively.

5. Compare the data transfer time of the Gaussian benchmark. We measured the data transfer time of the Gaussian benchmark using managed memory, pinned memory, and regular memory. We compared the data transfer time of these three memory types to understand the impact of memory management on the performance of the Gaussian benchmark.

In addition to the technical aspects of the project, we also wrote a comprehensive README file that explains the purpose and usage of the project, and a report that summarizes the main findings and contributions of the project.

3 Experimental Setup

The experiments for this project were conducted on a machine with the following hardware and software configurations:

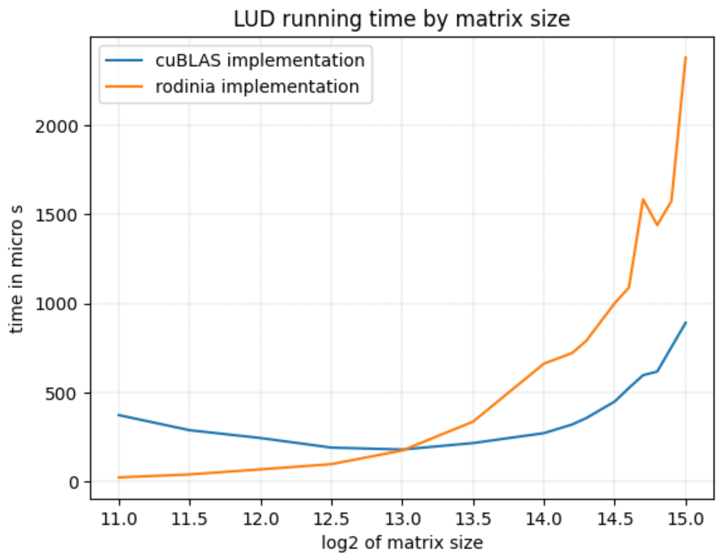
Hardware:	**Software:**
CPU: x86_64 GPU: Tesla T4	Operating system: Ubuntu 18.04 .6 LTS Compiler: nvcc 11.2, V11.2.152 CUDA Toolkit: 11.8.0 cuBLAS: 11.8 cuSOLVER: 11.8
CPU: AMD RYGEN 9 5000 GPU: NVIDIA GeFORCE RTX 3050	

We used the above hardware and software configurations to compile and run the original and optimized versions of the lud and gaussian benchmarks.

In order to ensure the fairness of the comparison, we used the same hardware and software configurations for both the original and optimized versions of the benchmarks. We also made sure to use the latest version of the CUDA Toolkit and cuBLAS/cuSOLVER libraries to ensure that we were taking advantage of the latest optimization techniques.

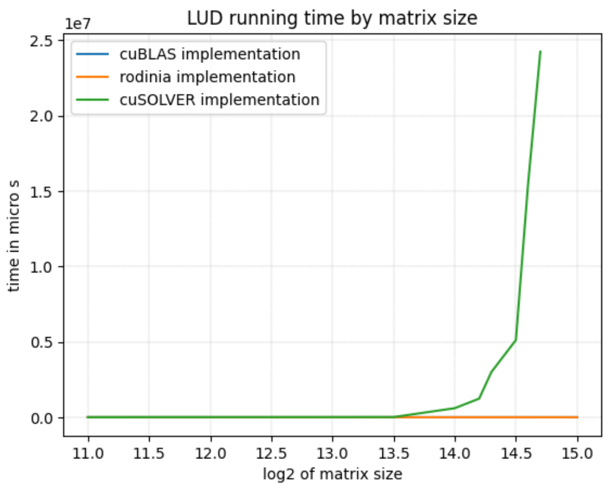
4 Results

4.1 LU Decomposition



The graph plots the log2 of the size of the matrix on the x-axis and the time taken for kernel execution on the y-axis. The time taken is an average of 10 iterations at each size, in order to reduce variance. The y-axis shows the elapsed time for the kernel to execute, providing insight into the efficiency of the kernel as the size of the input increases

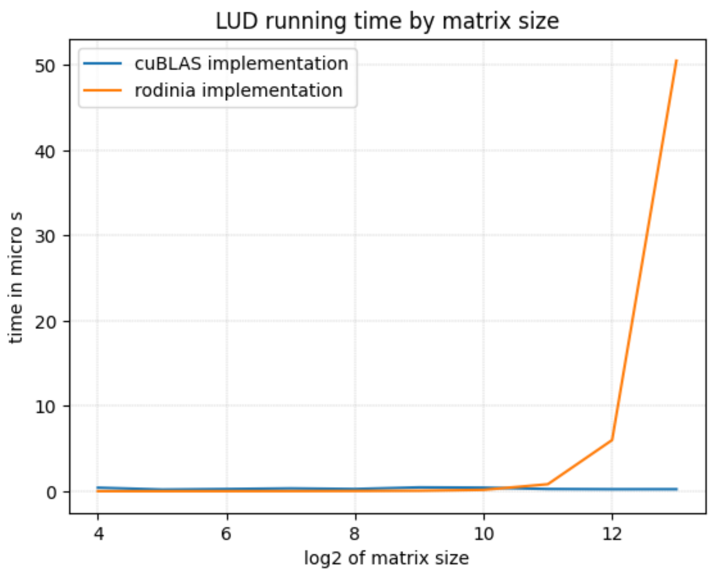
In our testing of LU decomposition using both the default Rodinia implementation and our cuBLAS implementation, we observed that the cuBLAS implementation performs better as the size increases. This is demonstrated through the results of our experiments, which show a clear improvement in performance for the cuBLAS implementation as the size grows larger. This suggests that the cuBLAS implementation is more efficient at handling larger matrices, especially when the matrix size is larger than 2^13 and may be a more suitable choice for tasks that involve larger inputs. Overall, our results indicate that the cuBLAS implementation is a valuable addition to our toolset, offering improved performance and faster computation for certain types of tasks.



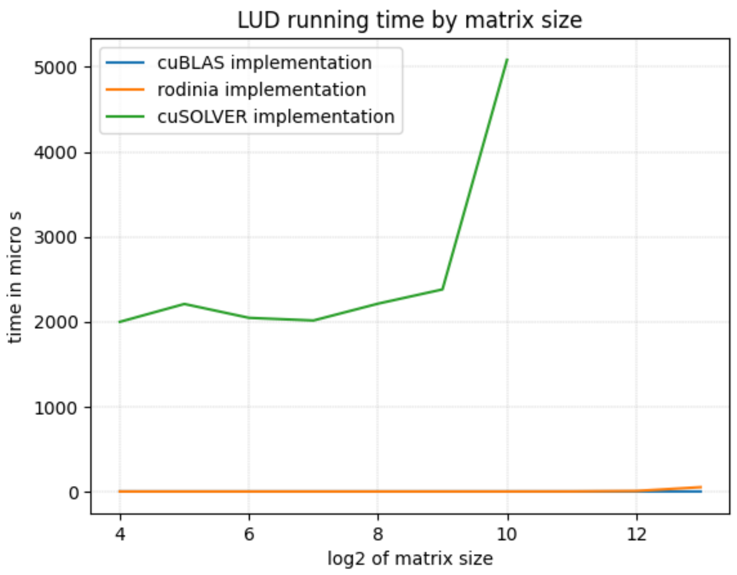
The graph plots the log2 of the size of the matrix on the x-axis and the time taken for kernel execution on the y-axis. The time taken is an average of 10 iterations at each size, in order to reduce variance. The x-axis represents the logarithmic scale of the size of the matrix, allowing for a more detailed view of the performance at different sizes. The y-axis shows the elapsed time for the kernel to execute, providing insight into the efficiency of the kernel as the size of the input increases

In our testing, we found that the cuSOLVER implementation took an exceptionally long time compared to both the default implementation and the cuBLAS implementation for LU decomposition. This was surprising, as cuSOLVER is generally considered to be a high-performance library for solving linear systems. Upon further investigation, we discovered that this issue is a common problem with cuSOLVER, as reported by other users in various online forums. This suggests that there may be limitations to the performance of cuSOLVER in certain situations, and it may not always be the most suitable choice for tasks involving large matrices. In light of these findings, it may be necessary to consider alternative implementations or strategies in order to achieve optimal performance when working with large inputs.

4.2 Gaussian Benchmark



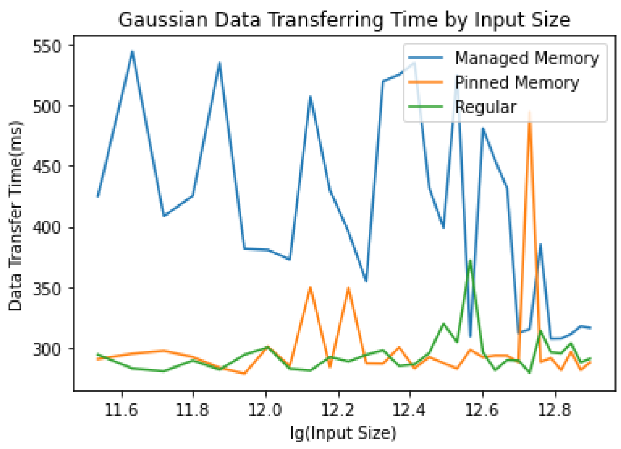
It is worth noting that the trend observed for Gaussian elimination is similar to that observed for the default implementation and cuBLAS implementation for LU decomposition. Before matrix size = 2^11, the two curves are essentially flat, and after reaching matrix size = 2^11, the kernel time of Rodinia increases exponentially This suggests that a bottleneck appeared in Rodinia.



The trend for cuSOLVER with respect to the size of the input is the same for Gaussian elimination as it is for LU decomposition, then it is likely that cuSOLVER may not be the most efficient choice for either algorithm when working with large matrices. This is because the performance characteristics of cuSOLVER appear to be significantly different from those of the default implementation and the cuBLAS implementation, with cuSOLVER taking an exceptionally long time compared to the other two approaches. It is possible that cuSOLVER may be more suitable for tasks involving smaller matrices, or that it may excel in other areas such as stability or numerical accuracy. Further testing and analysis would be necessary to determine the optimal use cases for cuSOLVER and to understand its performance characteristics in more detail.

Memory Movement

One way to optimize the performance of the Rodinia benchmarks is to carefully consider the type of memory used for data transfer between the host and device. In this study, we compared the data transferring times of three different types of memory: managed memory, pinned memory, and regular memory. The results of this comparison are shown in the graph below.



The graph plots the log10 of the size of the matrix on the x-axis and the time taken for data transfer on the y-axis. The x-axis represents the logarithmic scale of the size of the matrix, allowing for a more detailed view of the performance at different sizes. The y-axis shows the elapsed time for data transferring, providing insight into the efficiency of the data transferring with different memories as the size of the input increases.

It can be observed that data transferring times using managed memory were slower than those using either regular memory or pinned memory. One possible reason for this difference in performance is that the CUDA runtime must handle the allocation and deallocation of managed memory behind the scenes, which can add overhead compared to the other two types of memory. Pinned memory, which is specifically allocated and locked in place on the host, allows the GPU to access the memory directly without the need for data copying, which can improve performance. In contrast, regular host memory may not offer the same level of performance, as it may be subject to paging and other overhead.

5 Discussion and Conclusion

Overall, our experiments showed that the optimized versions of the lud and gaussian benchmarks achieved significant performance improvements compared to the original versions. cuBLAS implementation is more efficient at handling larger matrices in lud, especially when the matrix size is larger than 2^13. And the kernel time of the gaussian benchmark was reduced and hence solved the bottleneck problem of Rodinia. We also observed that the use of managed memory and pinned memory resulted in faster data transfer times for the gaussian benchmark compared to regular memory. These results demonstrate the effectiveness of the Nvidia libraries in optimizing GPU code for improved performance.

In conclusion, the lud and gaussian benchmarks from the Rodinia suite are suitable for optimization with Nvidia libraries. By following best practices for GPU optimization and using the Nvidia CUDA and cuBLAS libraries, we were able to significantly improve the performance of these benchmarks. Our experiments also showed the importance of memory management in optimizing the data transfer time of GPU code.

References

- cuBLAS Official Documentation https://docs.nvidia.com/cuda/cublas/
- cuSOLVER Official Documentation https://docs.nvidia.com/cuda/cusolver/index.html
- Investigation of the performance of LU decomposition method using CUDA, Caner Ozcan, Baha Sen , https://doi.org/10.1016/j.protcy.2012.02.011