NeoRodinia: Evaluation of High-Level Parallel Programming Models and Compiler Transformation for GPU Offloading

Xinyao Yi
xyi2@uncc.edu
University of North Carolina at Charlotte
Charlotte, North Carolina, USA

Anjia Wang
awang15@uncc.edu
University of North Carolina at Charlotte
Charlotte, North Carolina, USA

Yonghong Yan
yyan7@uncc.edu
University of North Carolina at Charlotte
Charlotte, North Carolina, USA

ABSTRACT

NeoRodinia is a comprehensive benchmark suite developed from Rodinia, containing 23 real-world applications and 5 microbenchmarks. It addresses the limitations of Rodinia by optimizing OpenMP GPU offloading programs and introducing OpenACC variants. The evaluation involves thorough performance assessments on various hardware architectures and compilers, measuring execution time and memory usage. These evaluations offer valuable insights into parallel programming models and compiler selection, guiding optimization efforts and helping developers, especially beginners to make informed decisions.

KEYWORDS

GPU, benchmarks, OpenMP, OpenACC, CUDA, evaluation, compiler

1 INTRODUCTION

Systems equipped with GPU accelerators feature complex heterogeneous and deep memory systems, necessitating proficient and accurate utilization by programmers to fully utilize the GPU's parallel capabilities. Parallel programming models, such as OpenMP, OpenACC and CUDA provide different styles and parallelism patterns of APIs to help users achieve efficient and portable parallel programming, yet challenges such as load imbalance and high communication delay are still prominent in applications. Additionally, compiler support also exhibits significant variation in terms of execution efficiency.

Benchmarks play an important role in parallel computing, offering examples to help users understand complexities, guide performance optimization, and evaluate heterogeneous systems and compiler effectiveness. NeoRodinia includes Rodinia's original 23 CUDA benchmarks and 5 additional microbenchmarks. We have developed corresponding optimized OpenMP Offloading and OpenACC Offloading versions for each benchmark. We also evaluated the performance of various parallel models and compilers, aiming to offer valuable insights into optimization and compiler selection.

We also introduced a 3-tier optimization model labeled P1, P2, and P3 (P for parallel), each with specific objectives and methods, demonstrated using microbenchmarks. P1 is the basic optimization that aims to balance the load and reduce memory access. P2 refers to advanced optimization including scheduling, block size, and dynamic/static optimizations. The P3 level represents the most challenging optimization strategies that involve tiling, vectorization, and manual code optimizations.

2 BACKGROUND AND MOTIVATION

2.1 Rodinia Benchmark Suite

The Rodinia benchmark suite includes 23 real-world applications and kernels for multi-core CPU and GPU platforms, covering various domains, and is implemented in parallel languages like CUDA, OpenCL, and OpenMP. Several studies evaluated Rodinia's performance across various platforms and have expanded it to support new features within programming models. Che et al. characterized Rodinia's performance on NVIDIA GeForce GTX480 [1]. Misra et al. evaluated Rodinia kernels "LUD" and "HotSpot" on Intel's x86-based Xeon Phi [2]. And Svedin et al. benchmarked the A100 GPU, comparing it to four previous GPU generations to quantify performance expectations [3].

2.2 Motivation

There are some limitations to Rodinia although it is popular. For instance, the OpenMP variant mostly offers CPU versions of programs, lacking optimized GPU versions. There is no official OpenACC variant and the existing unofficial versions barely work. Moreover, it lacks the evaluation of using different compilers. The varying compiler support can lead to inconsistent performance results, also sometimes introducing unpredictability errors.

To address the limitations, we proposed three objectives for developing NeoRodinia: optimizing and expanding Rodinia with OpenMP and OpenACC offloading benchmarks, serving as a platform for compiler evaluation, and providing valuable educational resources for developers, including five user-friendly microbenchmarks.

Meanwhile, we observed a lack of standardized optimization guidance, resulting in many optimization processes relying on developers' experience. Therefore, we introduced the 3-tier model not only to guide performance programming but also to standardize and improve traceability in optimization processes.

3 PRELIMINARY RESULTS

NeoRodinia consists of 23 real-world applications from Rodinia, along with 5 newly added microbenchmarks which are summarized in the table 1.

Applications	CUDA	OMP-CPU	OMP-GPU	OpenACC
Leukocyte	Existed	Existed	Added	Added
Heart Wall	Existed	Existed	Added	Added
MUMmerGPU	Existed	Existed	Added	Added
CFD Solver	Existed	Existed	Added	Added
LU Decomposition	Existed	Existed	Added	Added
HotSpot	Existed	Existed	Added	Added
Back Propagation	Existed	Existed	Added	Added
Needleman-Wunsch	Existed	Existed	Added	Added
Kmeans	Existed	Existed	Added	Added
Breadth- First Search	Existed	Existed	Added	Added
SRAD	Existed	Existed	Added	Added
Streamcluster	Existed	Existed	Added	Added
Particle Filter	Existed	Existed	Added	Added
PathFinder	Existed	Existed	Added	Added
Gaussian Elimination	Existed	Added	Added	Added
k-Nearest Neighbors	Existed	Existed	Added	Added
LavaMD	Existed	Existed	Added	Added
Myocyte	Existed	Existed	Added	Added
B+ Tree	Existed	Existed	Added	Added
GPUDWT	Existed	Added	Added	Added
Hybrid Sort	Existed	Added	Added	Added
Hotspot3D	Existed	Existed	Added	Added
Huffman	Existed	Added	Added	Added
AXPY	Added	Added	Added	Added
MatVec Mul	Added	Added	Added	Added
MatMat Mul	Added	Added	Added	Added
Sum	Added	Added	Added	Added
stencil	Added	Added	Added	Added

Table 1: Applications in NeoRodinia

3.1 Optimization for Stream Cluster

3.1.1 OpenMP Offloading Version. Stream cluster calculates point membership based on weight and distance to batch points. In the kernel, data transfers are frequent except for point coordinates, which are copied only if they change. Fig. 1 displays the optimized code. The optimization includes a function `center_is_stable()` on line 4 to check for changed input coordinates before data transfers. The `critical` directive is replaced with the more efficient `atomic` directive on line 14.

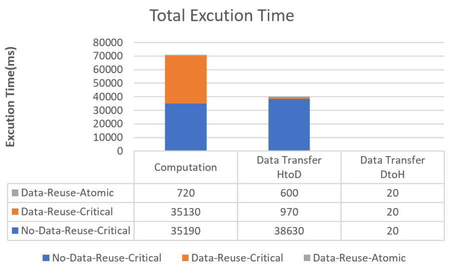
```
1  #pragma omp target data map(alloc:d_coord[0: chunksize * dim])
2  { ...
3     // only if necessary
4  #pragma omp target update to(d_coord[0:num*dim]) while(!center_is_stable()){
5  #pragma omp target teams distribute parallel for
6    map (to:d_points [0:chunksize], center_table [0: chunksize])
7    map(tofrom:switch_membership [0: chunksize], lower[0:stride * (nproc+1)])
8    num_teams(1024) num_threads(512) reduction(+:cost_of_opening_x)
9  for (int i = 0; i < num_points; i++) {
10  ... // compute distance between points
11  if (x_cost < current_cost) {
12  ...  } else {
13     int assign = d_points[i].assign;
14     #pragma omp atomic //The original version uses critical directive
15     lower[center_table[assign]] += current_cost - x_cost;
16   } ... } } }
```

Figure 1: Optimized OpenMP GPU Version of Stream Cluster

3.1.2 OpenACC Version. We use specific OpenACC directives instead of directly use `kernel` directive to address limitations in the compiler and runtime support. The directives we use basically achieve the same functions as using OpenMP, but due to the limitations of the OpenACC compiler and runtime support, we cannot achieve the same performance as using OpenMP.

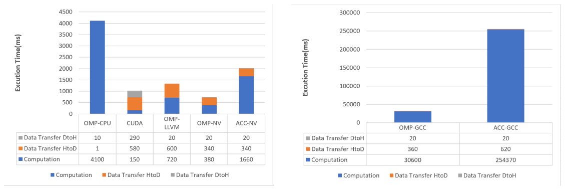
3.1.3 Experimental Results. The experimental results presented in Fig. 2 compare three versions of OpenMP: an initial version without optimization, a version using `center_is_stable()` to enable data reuse, and a version utilizing `atomic` directive. The results demonstrate that data multiplexing significantly reduces the data transmission time, while `atomic` directives lead to a reduction in intermediate results computation.

Fig. 3 presents a comparison of different models and compilers. The results indicate that the CUDA version achieves the highest



	Computation	Data Transfer HtoD	Data Transfer DtoH
▨ Data-Reuse-Atomic	720	600	20
🟧 Data-Reuse-Critical	35130	970	20
🟦 No-Data-Reuse-Critical	35190	38630	20

Figure 2: Kernel execution time of stream cluster. It's compiled by LLVM. The number of input points is 64k.



	OMP-CPU	CUDA	OMP-LLVM	OMP-NV	ACC-NV
▨ Data Transfer DtoH	10	290	20	20	20
🟧 Data Transfer HtoD	1	580	600	340	340
🟦 Computation	4100	150	720	380	1660

	OMP-GCC	ACC-GCC
▨ Data Transfer DtoH	20	20
🟧 Data Transfer HtoD	360	620
🟦 Computation	30600	254370

(a) (b)

Figure 3: Kernel execution time of stream cluster using different parallel models and compilers, a) total execution time for LLVM, NVCC and NVC compilers, b) total execution time for gcc compiler. OMP-CPU version is compiled by LLVM using 40 cores. For all GPU versions, the number of teams and threads is specified to make each thread only handle one loop iteration. The number of input points is 64k.

calculation efficiency, but incurs significant data transmission overhead. The OpenMP offloading version compiled with nvc delivers the best overall performance.

3.2 3-Tier Model

We use matrix-vector multiplication to demonstrate our 3-tier optimization model. In the P1 level, we leveraged the `parallel for` directive and data-sharing clauses `private` and `shared` to manage threads and minimize memory access. Moving on to the P2 level, we employed the `schedule(static, 64)` clause to implement static thread scheduling. At the P3 level, we achieved vectorization. Our experimental results for an input size of 10240*10240 demonstrated execution times of 43.6ms, 36.5ms, and 16.79ms for the three respective levels. These results closely align with our expectations: as we delve deeper into these optimization strategies, we consistently observe a progressive improvement in system performance.

REFERENCES

[1] Shuai Che, Jeremy W Sheaffer, Michael Boyer, Lukasz G Szafaryn, Liang Wang, and Kevin Skadron. 2010. A characterization of the Rodinia benchmark suite with comparison to contemporary CMP workloads. In *IEEE International Symposium on Workload Characterization (IISWC'10)*. IEEE, 1–11.

[2] Goldi Misra, Nisha Kurkure, Abhishek Das, Manjunatha Valmiki, Shweta Das, and Abhinav Gupta. 2013. Evaluation of rodinia codes on intel xeon phi. In *2013 4th international conference on intelligent systems, modelling and simulation*. IEEE, 415–419.

[3] Martin Svedin, Steven WD Chien, Gibson Chikafa, Niclas Jansson, and Artur Podobas. 2021. Benchmarking the nvidia gpu lineage: From early k80 to modern a100 with asynchronous memory transfers. In *Proceedings of the 11th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies*. 1–6.