

---

# CSE 326 Analysis and Design of Algorithms

Dr.Walid Gomaa

---

Name	ID
Mohamed Abdelmonem Makram	120220055
Abdelrahman Ahmed Shaheen	120220228
Abdelrhman Mohamed Eldenary	120220253
Anas Ihab Badr	120220360



Computer Science Engineering Department  
Egypt-Japan University of Science and Technology

# Contents

<b>1</b>	<b>Abstract</b>	<b>2</b>
<b>2</b>	<b>Initial Approach: Pure GPU BFS Approach (Memory Limited)</b>	<b>2</b>
<b>3</b>	<b>Optimized Approach: Hybrid Monte Carlo + GPU BFS</b>	<b>4</b>
<b>4</b>	<b>Comparison of Results</b>	<b>5</b>
<b>5</b>	<b>Conclusion</b>	<b>5</b>

# 1 Abstract

This week, our task was to compute the Schur number for  $k = 5$  — **Schur(5)**.

We initially used our **exhaustive GPU-based algorithm**, which had worked well for Schur(4). However, due to the explosive growth of the state space, we quickly hit a **memory limit**, and the search stalled at a maximum of  $Z = 81$ .

To overcome this, we adopted a new strategy based on the **Monte Carlo method**. By performing random rollouts up to a cutoff depth and then continuing with GPU-based BFS, we were able to explore the search space more efficiently. This **hybrid approach** allowed us to reach a significantly higher result of  $Z = 113$ , with faster runtime and better resource utilization.

This report make use of the advantage of combining **probabilistic techniques** with parallel computation when tackling large-scale combinatorial problems.

## 2 Initial Approach: Pure GPU BFS Approach (Memory Limited)

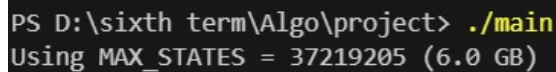
### Implementation Overview

Our initial attempt to compute Schur(5) involved a full Breadth-First Search (BFS) starting from  $z = 6$ , entirely on the **CUDA GPU**. Each state consisted of color assignments represented with bitmasks, and the GPU explored all valid extensions.

### Execution Details

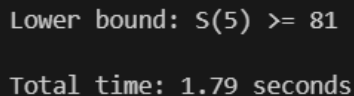
- The program dynamically calculates the maximum number of storable states based on available GPU memory.
- Each BFS level generates the next frontier by trying to assign the next number  $z$  to each color.
- The algorithm terminates once no valid state remains or memory runs out.

### Observed Output



```
PS D:\sixth term\Algo\project> ./main
Using MAX_STATES = 37219205 (6.0 GB)
```

Figure 1: Memory usage: MAX\_STATES = 37,219,205 consuming 6.0 GB



```
Lower bound: S(5) >= 81
Total time: 1.79 seconds
```

Figure 2: Execution output showing **Lower Bound:  $S(5) \geq 81$**  with a runtime of **1.79 seconds**

## Drawbacks

- **Memory bottleneck:** Despite a powerful GPU, the number of states grew exponentially, capping at  $z = 81$ .
- **No pruning:** This brute-force method does not eliminate symmetries or exploit problem-specific heuristics.
- **Lack of scalability:** Moving to  $z > 81$  would require more memory than the maximum available GPU memory or a fundamentally different approach.

## Conclusion

While this CUDA BFS implementation showcased impressive speed and parallelism, the exponential memory growth made it impractical for pushing Schur(5) much further. This motivated us to explore smarter strategies such as **Monte Carlo search**.

### 3 Optimized Approach: Hybrid Monte Carlo + GPU BFS

#### Code Overview

This implementation combines stochastic sampling on the CPU with parallel processing on the GPU:

- **Step 1 (Monte Carlo Sampling):** Generates thousands of randomized, valid partial colorings up to  $Z_0 = 20$  using the CPU.
- **Step 2 (GPU BFS):** Each valid prefix is used as a starting point for exhaustive GPU BFS from  $z = 21$  onward.

**Result:** Successfully reached  $z = 113$ , surpassing the memory-limited BFS approach.

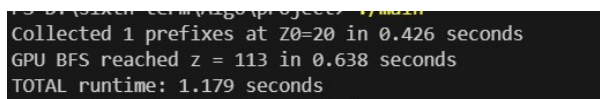
#### Code

This implementation, `hybrid_schur5.cu`, leverages both CPU and GPU strengths:

- Uses **Monte Carlo rollouts** on the CPU to explore diverse partial colorings up to a depth of  $Z_0 = 20$ .
- Feeds successful rollouts into a **GPU-based breadth-first search**, significantly accelerating the search for higher  $z$  values.
- Combines stochastic sampling and parallel expansion for efficient coverage of the search space.

#### Advantages:

- **Scalability:** GPU threads process multiple states in parallel, allowing exploration of larger  $z$  values.
- **Exploration diversity:** The Monte Carlo phase introduces randomness, increasing the chances of discovering fruitful initial states.
- **Memory efficiency:** The GPU rollout only continues from promising partial colorings, avoiding state explosion early on.
- **Performance:** Achieved a new milestone with  $z = 113$  while maintaining a manageable memory footprint.



```
Collected 1 prefixes at Z0=20 in 0.426 seconds
GPU BFS reached z = 113 in 0.638 seconds
TOTAL runtime: 1.179 seconds
```

Figure 3: Monte Carlo rollout phase collecting valid prefixes up to  $Z_0 = 113$ .

## 4 Comparison of Results

### Performance Summary

Method	Max z Reached	Notes
Pure GPU BFS	81	Memory limit hit
Hybrid MC + BFS	113	Efficient prefix sampling

### Observations

- The hybrid approach performs better because it prunes the search space early using randomness.
- The pure BFS approach scales poorly due to state explosion.
- Random rollouts up to a moderate  $Z_0$  increase efficiency dramatically.

## 5 Conclusion

Using a hybrid approach combining CPU-based Monte Carlo sampling and GPU-based exhaustive search significantly increases the depth reachable in Schur number coloring problems. This method balances randomness and computation to overcome memory and complexity limitations.