SCAN STACK

Scan Stack: A Search-based Concurrent Stack for GPU

This report presents the implementation details of the Scan Stack, a search-based concurrent stack designed for efficient GPU operations, by Aditya Azad, Sparsh Sehgal, and Vraj Patel.

Understanding the Base Structure of Scan Stack

- **One-Dimensional Array Structure**
  The Scan Stack utilizes a one-dimensional array, optimizing GPU memory access.
- **GPU-Friendly Implementation**
  This structure facilitates efficient memory access patterns for GPU threads.
- **Initialization Method**
  The init() function initializes the stack's array, marking cells as EMPTY.
- **Key Constants for Operations**
  Constants EMPTY (-1) and INVALID (-2) manage stack states during operations.
- **Push and Pop Operations**
  Specific methods for adding and removing elements from the Scan Stack are included.

Understanding Stack Operations: Push & Pop

An overview of push and pop operations in stacks

Push Operation

Inserts an integer value into the stack by scanning for an empty cell and using atomicCAS for safe insertion.

Pop Operation

Removes and returns the top value from the stack, marking the cell as invalidated after extraction.

Contention Handling

If insertion fails, the system backs off and rescans for a valid position to ensure data integrity.

Cell Validation

Ensures that invalid cells are re-validated when appropriate, maintaining stack consistency.

Effective Contention Handling Techniques

- **Contention in Stack Access**
  Multiple threads accessing the stack can lead to performance issues due to contention.

- **Scanning Approach**
  This method allows threads to independently locate the stack's top, reducing contention.

- **CAS Operation Handling**
  Failed Compare-And-Swap (CAS) operations trigger a backoff and rescan to minimize conflicts.

- **Using Invalid Cells**
  Invalid cells help prevent race conditions during push and pop operations on the stack.

- **Warp Elimination Strategy**
  A lightweight scheme that operates entirely within a CUDA warp to reduce contention.

- **Block Elimination Strategy**
  A more robust backoff method that spans the entire block, enhancing contention management.

Performance Characteristics of Scan Stack

Understanding factors influencing Scan Stack performance

- Elimination technique boosts throughput
  - Significantly enhances throughput, particularly with mixed push and pop operations.
- Lowest-area optimization reduces overhead
  - Lowers scanning overhead, especially beneficial for larger stack sizes.
- Scalability with operation counts
  - Implementation demonstrates strong scalability as operation counts increase.
- Granularity affects performance
  - Interval size for lowest-area scans impacts trade-off between overhead and accuracy.
- Thread block size plays a role
  - Thread block size influences the effectiveness of the elimination technique.
- Balanced operation mix enhances efficiency
  - Mixing push and pop operations creates more elimination opportunities.
- Stack size influences overhead
  - The overall capacity of the stack significantly impacts scanning overhead.

Overcoming Stack Implementation Challenges

Key issues faced during stack implementation process

The Step Over Problem

Threads scanning for the stack's top can pass each other, causing issues.

Avoiding Race Conditions

Concurrent access must be managed to keep the stack valid and stable.

Balancing Access Methods

Finding a balance between elimination processes and direct access was complex.

Memory Coalescing Optimization

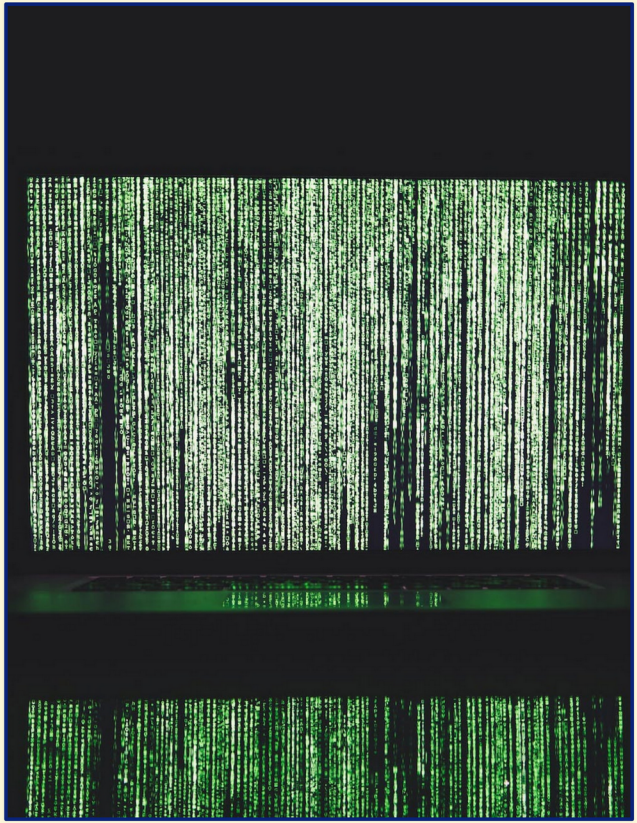
Effective memory access patterns needed careful code structuring for optimization.

Enhancing Future Work in Data Systems

- **Dynamic LA GRANULARITY Adjustment**
  Implement dynamic changes in LA GRANULARITY based on stack conditions.

- **Alternative Elimination Strategies**
  Explore different methods for data elimination to improve efficiency.

- **Support for Multiple Data Types**
  Extend capabilities to handle various data types for broader applications.



THANKS

Efficient Concurrent Stack for GPUs

Leveraging GPU Features for Stack Operations

- Efficient Concurrent Stack Implementation
  - Our Scan Stack provides a concurrent stack tailored for GPU applications.
- Leverages GPU Features
  - Utilizes memory coalescing and warp execution model for performance optimization.
- Non-Blocking Access Techniques
  - Employs non-blocking methods to manage concurrent access challenges effectively.
- Search-Based Scan Stack
  - Features a fully functional implementation of the search-based Scan Stack.

Efficient Concurrent Stack for GPUs

Leveraging GPU Features for Stack Operations

- Lowest-Area Optimization
  - Incorporates lowest-area optimization to enhance the efficiency of top location.
- Contention Reduction Techniques
  - Utilizes elimination techniques to minimize contention among operations.
- Scalability with Operations
  - Designed to scale efficiently as the number of operations increases.
- Array-Based Concurrent Structures
  - Demonstrates effective implementation of array-based concurrent data structures on GPUs.
- High-Throughput Solutions
  - Combines operations and techniques for scalable high-throughput stack operations.