BSG Manycore QuickSort Implementation

This README provides an overview of the QuickSort implementation for the BSG Manycore architecture.

Overview

This kernel implements a QuickSort algorithm to sort an array of KeyValuePair structures on the BSG Manycore hardware. The sorting is based on the 'data' field of the KeyValuePair structure.

Key Components

1. `KeyValuePair`: A structure containing an 'id' and a 'data' field.
2. `swap`: Helper function to swap two KeyValuePair elements.
3. `partition`: Function to partition the array for QuickSort.
4. `quick_sort`: Recursive QuickSort implementation.
5. `kernel_sort`: Main kernel function that initializes the sorting process.

Function Descriptions

swap

Swaps two KeyValuePair elements.

partition

Partitions the array around a pivot for the QuickSort algorithm.

quick_sort

Implements the recursive QuickSort algorithm.

kernel_sort

The main entry point for the sorting kernel. It performs the following steps:

1. Initializes hardware barriers for synchronization.
2. Copies the input array to the output array.
3. Performs QuickSort on the output array.
4. Synchronizes all threads after sorting.

Usage

To use this kernel:

1. Ensure the BSG Manycore environment is properly set up.
2. Compile the kernel for the BSG Manycore architecture.
3. Load the compiled kernel onto the BSG Manycore hardware.
4. Provide input data as an array of KeyValuePair structures.
5. Retrieve the sorted output from the hardware.

Performance Considerations

- The sorting is performed by a single thread (__bsg_id == 0) to avoid synchronization issues.
- Hardware barriers are used to ensure proper synchronization between threads.
- The kernel includes debugging statements for performance analysis.

Limitations

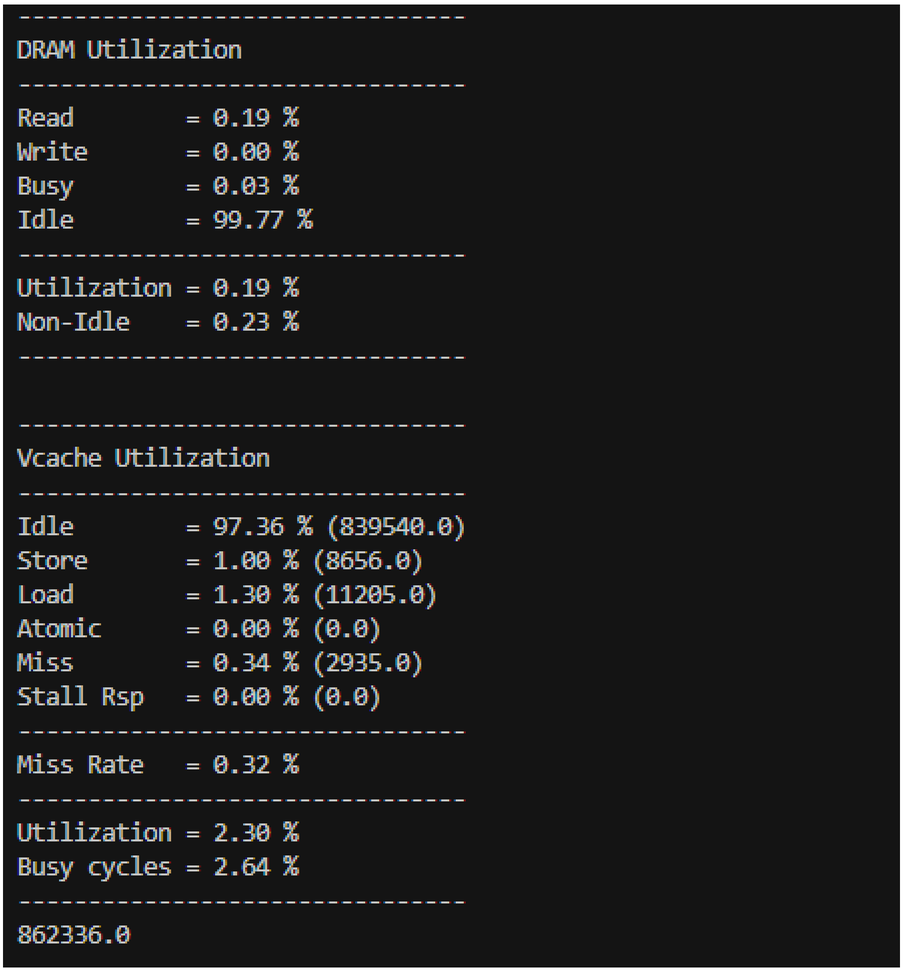
- The current implementation may not fully utilize the parallel capabilities of the BSG Manycore architecture.
- Performance may vary based on input size and data distribution.

Future Improvements

- Implement a parallel sorting algorithm to better utilize the BSG Manycore's capabilities.
- Optimize memory access patterns for improved performance.
- Add error handling and input validation.

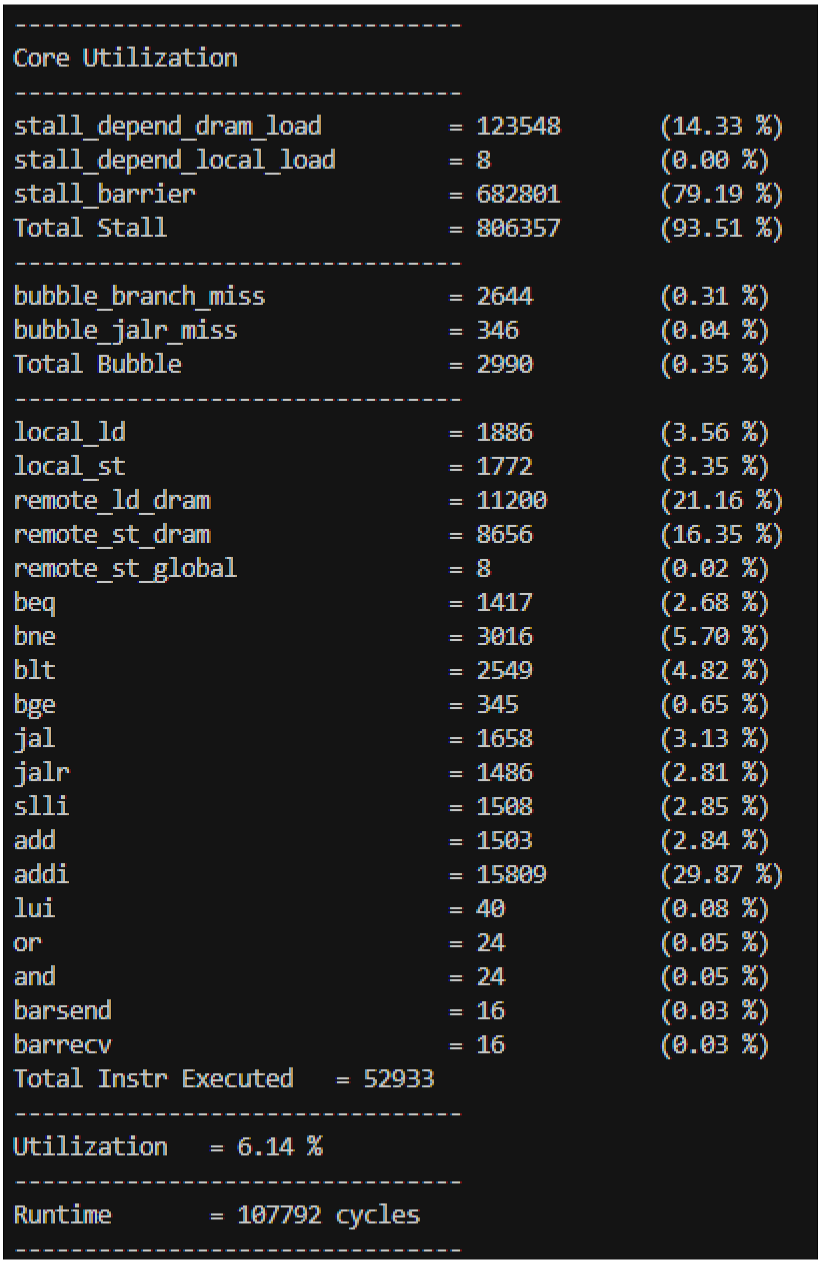
For further assistance or to report issues, please contact

For 2^8 : Runtime cycles = 107792 cycles



```
--------------------------------
DRAM Utilization
--------------------------------
Read        = 0.19 %
Write       = 0.00 %
Busy        = 0.03 %
Idle        = 99.77 %
--------------------------------
Utilization = 0.19 %
Non-Idle    = 0.23 %
--------------------------------


--------------------------------
Vcache Utilization
--------------------------------
Idle        = 97.36 % (839540.0)
Store       = 1.00 % (8656.0)
Load        = 1.30 % (11205.0)
Atomic      = 0.00 % (0.0)
Miss        = 0.34 % (2935.0)
Stall Rsp   = 0.00 % (0.0)
--------------------------------
Miss Rate   = 0.32 %
--------------------------------
Utilization = 2.30 %
Busy cycles = 2.64 %
--------------------------------
862336.0
```



```
--------------------------------
Core Utilization
--------------------------------
stall_depend_dram_load          = 123548       (14.33 %)
stall_depend_local_load         = 8            (0.00 %)
stall_barrier                   = 682801       (79.19 %)
Total Stall                     = 806357       (93.51 %)
--------------------------------
bubble_branch_miss              = 2644         (0.31 %)
bubble_jalr_miss                = 346          (0.04 %)
Total Bubble                    = 2990         (0.35 %)
--------------------------------
local_ld                        = 1886         (3.56 %)
local_st                        = 1772         (3.35 %)
remote_ld_dram                  = 11200        (21.16 %)
remote_st_dram                  = 8656         (16.35 %)
remote_st_global                = 8            (0.02 %)
beq                             = 1417         (2.68 %)
bne                             = 3016         (5.70 %)
blt                             = 2549         (4.82 %)
bge                             = 345          (0.65 %)
jal                             = 1658         (3.13 %)
jalr                            = 1486         (2.81 %)
slli                            = 1508         (2.85 %)
add                             = 1503         (2.84 %)
addi                            = 15809        (29.87 %)
lui                             = 40           (0.08 %)
or                              = 24           (0.05 %)
and                             = 24           (0.05 %)
barsend                         = 16           (0.03 %)
barrecv                         = 16           (0.03 %)
Total Instr Executed    = 52933
--------------------------------
Utilization    = 6.14 %
--------------------------------
Runtime        = 107792 cycles
--------------------------------
```