BGPQ: A Heap-Based Priority Queue Design for GPUs

Yanhao Chen  
cy827@cs.rutgers.edu  
Rutgers, The State University of New Jersey  
New Brunswick, New Jersey, USA

Yuwei Jin  
yj243@scarletmail.rutgers.edu  
Rutgers, The State University of New Jersey  
New Brunswick, New Jersey, USA

Fei Hua  
huafei90@gmail.com  
Rutgers, The State University of New Jersey  
New Brunswick, New Jersey, USA

Eddy Z. Zhang  
eddy.zhengzhang@gmail.com  
Rutgers, The State University of New Jersey  
New Brunswick, New Jersey, USA
Priority Queues

• Priority queues are fundamental data structures.
  • Insert and DeleteMin Operations.
  • Search problem: A* search algorithm
  • Graph problem: Dijkstra’s algorithm
  • Branch-and-Bound problem: 0/1 Knapsack
Concurrent Priority Queues

• Concurrent priority queues for **multi-core CPUs** have been well studied.
  • Heap based: Nageshwara and Kumar [21], Hunt et al.[14], ...
  • Skip-List based: Sundell and Tsigas[27], Linden and Jonsson [16], ...
  • Others: MD-List[30], CBPQ [3], ...

• Very few studies on the concurrent priority queues for **many-core GPUs**.
  • Heap based: He et al.[12]
  • Skiplist based: Moscovici et al. [20]
Concurrent Priority Queue Design Choices

Heap Based Priority Queue

4

8

6

16

9

15
Concurrent Priority Queue Design Choices

Heap Based Priority Queue

- Computation Complexity: $O(\log N)$
- Space Complexity: $N + O(1)$
Concurrent Priority Queue Design Choices

**Heap Based Priority Queue**

- Computation Complexity: $O(\log N)$
- Space Complexity: $N + O(1)$
- Limitation: Scalability
**Concurrent Priority Queue Design Choices**

**Heap Based Priority Queue**

- Computation Complexity: O(logN)
- Space Complexity: N + O(1)
- Limitation: Scalability

**Skip-List Based Priority Queue**

- Computation Complexity: O(logN)
- Space Complexity: N + O(1)
- Limitation: Scalability
Concurrent Priority Queue Design Choices

**Heap Based Priority Queue**
- Computation Complexity: $O(\log N)$
- Space Complexity: $N + O(1)$
- Limitation: Scalability

**Skip-List Based Priority Queue**
- Computation Complexity: $O(\log N)$
- Space Complexity: $O(N) + O(1)$
Concurrent Priority Queue Design Choices

Heap Based Priority Queue

- Computation Complexity: $O(\log N)$
- Space Complexity: $N + O(1)$
- Limitation: Scalability

Skip-List Based Priority Queue

- Computation Complexity: $O(\log N)$
- Space Complexity: $O(N) + O(1)$
- Dynamic memory management.
Concurrent Priority Queue Design Choices

Heap Based Priority Queue

- Computation Complexity: $O(\log N)$
- Space Complexity: $N + O(1)$
- Limitation: Scalability

Skip-List Based Priority Queue

- Computation Complexity: $O(\log N)$
- Space Complexity: $O(N) + O(1)$
- Dynamic memory management.
Concurrent Priority Queue Design Choices

• Parallelism Exploitation
  • Most existing approaches exploit only task parallelism.
  • Very few studies have exploited data parallelism.
    • CPU: Deo and Prasad [8]
    • GPU: He et al.[12], Moscovici et al. [20]

• Operation collaboration
## Concurrent Priority Queue Design Choices - Summary

<table>
<thead>
<tr>
<th></th>
<th>CPU Implementations</th>
<th>GPU Implementations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>STSL [27]</td>
<td>LJSL [16]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P-Sync [12]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BGPQ</td>
</tr>
<tr>
<td><strong>Data Structure</strong></td>
<td>Heap</td>
<td>Linked List + Skip-List</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Skip-List</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Skip-List</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Skip-List</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Skip-List</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Heap</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Heap</td>
</tr>
<tr>
<td><strong>Data Parallelism</strong></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>Task parallelism</strong></td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td>O</td>
<td>O</td>
</tr>
<tr>
<td><strong>Operation collaboration</strong></td>
<td>X</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>Memory Efficiency</strong></td>
<td>O</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>O</td>
</tr>
</tbody>
</table>

STSL: Sundell and Tsigas’s skip-list based implementation.
LJSL: Linden and Jonsson’s skip-list based implementation.
Spray List: a relaxed skip-list based implementation.
P-Sync: He et al.’s GPU heap-based implementation.
BGPQ Design Choices

• We proposed a **Batched-based GPU Priority Queue** (BGPQ).
• BGPQ uses the **Heap** as the underlying data structure.
• BGPQ exploits both task and data parallelism.
• Thread collaboration methods are used in BGPQ for better scalability.
• BGPQ is the first linearizable GPU priority queue implementation.
BGPQ Overview

• Each node contains exactly $k$ keys.
  • Except root and buffer.
• The smallest key in the node is larger than the largest key of its children’s.
BGPQ Insert Operation

- Insert keys are sorted first and then merged with the root node.
  - Smallest keys are placed back into the root.
BGPQ Insert Operation

- Insert keys are sorted first and then merged with the root node.
  - Smallest keys are placed back into the root.
  - Root still contains smallest keys in the heap.

```
Insert keys  
2 4
0 1 1 2
2 2 3 4
4 5 5 6
6 8 8 9
```

Buffer node
3
BGPQ Insert Operation

- Insert keys are sorted first and then merged with the root node.
  - Smallest keys are placed back into the root.
  - Root still contains smallest keys in the heap.
- If the buffer can hold all insert keys, updated insert keys will be placed in the buffer.
BGPQ Insert Operation

• Insert keys are sorted first and then merged with the root node.
  • Smallest keys are placed back into the root.
  • Root still contains smallest keys in the heap.
• If the buffer can hold all insert keys, updated insert keys will be placed in the buffer.
• Otherwise, an insert heapify will be triggered.
  • Insert heapify traverses the path from the root to the target.
BGPQ Insert Operation

• Insert keys are sorted first and then merged with the root node.
  • Smallest keys are placed are placed back into the root.
  • Root still contains smallest keys in the heap.
• If the buffer can hold all insert keys, updated insert keys will be placed in the buffer.
• Otherwise, an insert heapify will be triggered.
  • Insert heapify traverses the path from the root to the target.
BGPQ Insert Operation

- Insert keys are sorted first and then merged with the root node.
  - Smallest keys are placed back into the root.
  - Root still contains smallest keys in the heap.
- If the buffer can hold all insert keys, updated insert keys will be placed in the buffer.
- Otherwise, an insert heapify will be triggered.
  - Insert heapify traverses the path from the root to the target.
BGPQ Insert Operation

- Insert keys are sorted first and then merged with the root node.
  - Smallest keys are placed back into the root.
  - Root still contains smallest keys in the heap.
- If the buffer can hold all insert keys, updated insert keys will be placed in the buffer.
- Otherwise, an insert heapify will be triggered.
  - Insert heapify traverses the path from the root to the target.
BGPQ DeleteMin Operation

- If the root contains enough keys to delete, they are retrieved directly.
BGPQ DeleteMin Operation

- If the root contains enough keys to delete, they are retrieved directly.
- Otherwise, if the root becomes empty, a deleteMin heapify is triggered.
BGPQ DeleteMin Operation

- If the root contains enough keys to delete, they are retrieved directly.
- Otherwise, if the root becomes empty, a deleteMin heapify is triggered.

![Diagram of a binary heap with delete keys highlighted and buffer node at the bottom right]
BGPQ DeleteMin Operation

• If the root contains enough keys to delete, they are retrieved directly.
• Otherwise, if the root becomes empty, a deleteMin heapify is triggered.
Evaluation

• We compare BGPQ with:
  • Four CPU baselines
    • Intel Thread Building Blocks - TBB [29]
    • Chunk Based Priority Queue - CBPQ [3]
    • Skip-List based Priority Queue - LJS [16] and SprayList [1]
  • Four Xeon E7-4879 processors, 1TB memory, 2.4 GHz, 80 threads

• One GPU baseline
  • He et al. – P-Sync [12]

• Nvidia TITAN X GPU w/ 28 SMs
Evaluation - BGPQ Performance

(a) Insert operation

(b) DeleteMin operation

(c) Performance w.r.t thread block numbers
Evaluation - BGPQ Performance

![Graph showing Mean BGPQ Speedup for CPU and GPU with TBB, SprayList, CBPQ, LJS, and P-Sync. The graph compares Insertion & DeleteMin Random Keys for 1M, 8M, and 64M keys.]

- TBB: Gray bar for 1M Keys, Orange for 8M Keys, Blue for 64M Keys.
- SprayList: Blue bar for 1M Keys, Orange for 8M Keys, Gray for 64M Keys.
- CBPQ: Blue bar for 1M Keys, Orange for 8M Keys, Gray for 64M Keys.
- LJS: Orange bar for 1M Keys, Blue for 8M Keys, Gray for 64M Keys.
- P-Sync: Blue bar for 1M Keys, Orange for 8M Keys, Gray for 64M Keys.

50th International Conference on Parallel Processing (ICPP)
August 9-12, 2021 in Virtual Chicago, IL
Evaluation – A* searching

A* Search for Routing Planning

- TBB
- SprayList
- LJSR

50th International Conference on Parallel Processing (ICPP)
August 9-12, 2021 in Virtual Chicago, IL
Conclusion

• We study existing concurrent priority queue implementations and their corresponding GPU friendliness.

• We present BGPQ, a GPU-friendly priority queue implementation.

• We compare the performance of BGPQ with 4 CPU baselines and 1 GPU baseline and show significant speedup.
Question?