Laborator 4 & 5 - Tanislav Cosmin-Gabriel, Grup 237

**Implementation details**

The program runs all the tests automatically from a statically specified configurations list, printing the test configuration and the average time it took to execute over the course of multiple runs.

Each configuration consists of a number of polynomials to generate, the max number of polynomial members to generate (each polynomial will have a different random number of members between 1 and the max number of members), the max power for each member (each member will have a different random power between 0 and the max power), the implementation to use, 0 for threaded, and 1 for sequential, the number of producer threads to use, the number of consumer threads to use, and the type of locking to use with the polynomial linked list used for storing the members.

If the configuration’s number of polynomials, max number of polynomial members, or the max power of each element differ from the previous used configuration, the polynomial files are regenerated.

All the streams are opened and the implementation specified in the configuration is ran multiple times, adding the times for each run to a sum. After all runs are done, the average time of a run is calculated.

**Threaded implementation**

A queue is created with streams.

Number of producer threads are created, and each thread will pop a stream from the queue until the queue is empty. The threads will read all the members from a polynomial stream and add them to the members queue.

Number of consumer threads are created, and each thread will pop a member from the members queue until the queue is empty and the queue is marked as closed. The thread will add the member to the polynomial linked list.

**Iterative implemetation**

All streams and read in order, adding each member read from the stream to the polynomial linked list.

**Queue implementation**

The queue can be created from a vector, and a closed flag can be passed.

By default, the queue is not closed.

The queue can be marked as closed, operation which will wake up all waiting threads using the conditional variable.

An element can be pushed to the back of the queue. The push back operation grabs the queue’s lock and adds the element to the inner data structure. The first thread waiting for data is notified using a conditional variable.

An element can be popped from the front of the queue. The pop operation grabs the queue’s lock, and, in a loop, checks if the queue is not empty, in which case the loop is exited, checks if the queue is closed, in which case the function returns true to tell the caller that the queue closed, and, in other cases, waits for a signal from the conditional variable. When the signal comes, the loop repeats, checking again if data is available, or if the queue closed. When the loop exited, the element at the front of the queue is returned.

**Polynomial linked list implementation**

The polynomial members present in the polynomial can be retrieved.

Internally, the linked list has a sentinel value at the head, which simplifies the addition process.

A polynomial member can be added to the polynomial.

If the linked list is told to use a global locking method, the global lock is grabbed at the beginning of the addition process**,** and only released after the addition process is done. If the linked list is told to use an individual locking method, the lock of each iterated node is locked, and released after all the operations with the node is done.

We start with the current node being the sentinel node, and all operations are done on the node after the current one**,** to make it easier to insert a node without storing the previous node.

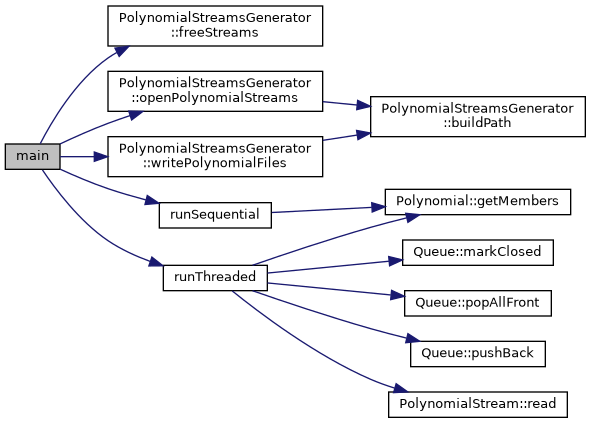
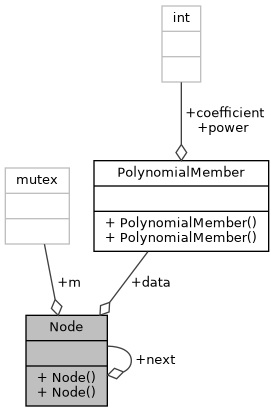
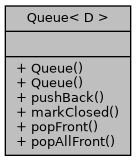
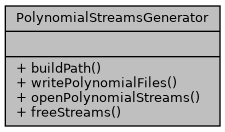
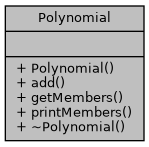
First, we check if the node after the current one is null, in which case we must have reached the end of the list, so we add a new node and populate it with the given member.

If the node after the current one has a power smaller than the member to be added**,** then we must insert the member between the current one and the node after the current one.

If the node after the current one has a power equal to the member to be added’s power, then we add the coefficient of the member to be added to the coefficient of the node after the current one.

If none of these cases were valid, then we make the node after the current one, the current node.

**Classes**



**Results (average over 100 runs)**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Polynomials** | **Max members** | **Max member power** | **Implementation** | **Producer threads** | **Cosumer threads** | **Locking type** | **Time (microseconds)** |
| 10 | 50 | 1000 | thread | 1 | 3 | global | 89 |
| 10 | 50 | 1000 | thread | 1 | 5 | global | 197 |
| 10 | 50 | 1000 | thread | 1 | 7 | global | 188 |
|  |  |  |  |  |  |  |  |
| 10 | 50 | 1000 | thread | 2 | 2 | global | 57 |
| 10 | 50 | 1000 | thread | 2 | 4 | global | 105 |
| 10 | 50 | 1000 | thread | 2 | 6 | global | 206 |
|  |  |  |  |  |  |  |  |
| 10 | 50 | 1000 | thread | 3 | 1 | global | 82 |
| 10 | 50 | 1000 | thread | 3 | 3 | global | 279 |
| 10 | 50 | 1000 | thread | 3 | 5 | global | 192 |
|  |  |  |  |  |  |  |  |
| 10 | 50 | 1000 | thread | 1 | 3 | individual | 67 |
| 10 | 50 | 1000 | thread | 1 | 5 | individual | 589 |
| 10 | 50 | 1000 | thread | 1 | 7 | individual | 169 |
|  |  |  |  |  |  |  |  |
| 10 | 50 | 1000 | thread | 2 | 2 | individual | 65 |
| 10 | 50 | 1000 | thread | 2 | 4 | individual | 96 |
| 10 | 50 | 1000 | thread | 2 | 6 | individual | 760 |
|  |  |  |  |  |  |  |  |
| 10 | 50 | 1000 | thread | 3 | 1 | individual | 60 |
| 10 | 50 | 1000 | thread | 3 | 3 | individual | 399 |
| 10 | 50 | 1000 | thread | 3 | 5 | individual | 191 |
|  |  |  |  |  |  |  |  |
| 10 | 50 | 1000 | sequential | - | - | none | 2 |
|  |  |  |  |  |  |  |  |
| 5 | 100 | 10000 | thread | 1 | 3 | global | 58 |
| 5 | 100 | 10000 | thread | 1 | 5 | global | 99 |
| 5 | 100 | 10000 | thread | 1 | 7 | global | 128 |
|  |  |  |  |  |  |  |  |
| 5 | 100 | 10000 | thread | 2 | 2 | global | 57 |
| 5 | 100 | 10000 | thread | 2 | 4 | global | 101 |
| 5 | 100 | 10000 | thread | 2 | 6 | global | 142 |
|  |  |  |  |  |  |  |  |
| 5 | 100 | 10000 | thread | 3 | 1 | global | 66 |
| 5 | 100 | 10000 | thread | 3 | 3 | global | 468 |
| 5 | 100 | 10000 | thread | 3 | 5 | global | 332 |
|  |  |  |  |  |  |  |  |
| 5 | 100 | 10000 | thread | 1 | 3 | individual | 56 |
| 5 | 100 | 10000 | thread | 1 | 5 | individual | 115 |
| 5 | 100 | 10000 | thread | 1 | 7 | individual | 176 |
|  |  |  |  |  |  |  |  |
| 5 | 100 | 10000 | thread | 2 | 2 | individual | 61 |
| 5 | 100 | 10000 | thread | 2 | 4 | individual | 101 |
| 5 | 100 | 10000 | thread | 2 | 6 | individual | 149 |
|  |  |  |  |  |  |  |  |
| 5 | 100 | 10000 | thread | 3 | 1 | individual | 70 |
| 5 | 100 | 10000 | thread | 3 | 3 | individual | 101 |
| 5 | 100 | 10000 | thread | 3 | 5 | individual | 135 |
|  |  |  |  |  |  |  |  |
| 5 | 100 | 10000 | sequential | - | - | none | 2 |

**Conclusions**

The sequential implementation seems to be faster in the requested test cases, probably because of lock contention.