Programming for performance - Assignment 2

Abhay Kumar Dwivedi
22111001

September 5, 2022

1 Solution

Given array accesses :

A(i, i - j) = A(i, i - j - 1) - A(i + 1, i - j) + A(i - 1, i - j - 1)

true dependency	anti dependency	output dependency
1	2	0

Here, there are three reads in array A and one write in the same array. We can't have dependency between read - read accesses so the 3 reads are independent and there is no dependency among them.
Also, We have one write on A which is going to have dependency with each one of the read on array A.

Let's name the three read as :
R1 : A(i, i - j - 1)
R2 : A(i + 1, i - j)
R3 : A(i - 1, i - j - 1)

And since we have only one write let's denote it with W i.e.,
W : A(i, i - j)

Case - 1 : Dependency between W and R1

Using delta test we will equate the corresponding subscript of the array,

for the first subscript,
$i = i + \Delta i$
$\Delta i = 0$

for the second subscript,
$i - j = (i + \Delta i) - (j + \Delta j) - 1$
$i - j = (i - j) + (\Delta i - \Delta j) - 1$

$\Delta j = \Delta i - 1$
$\Delta j = -1$

Distance vector = (0, -1)
but we can't have first non - zero value as negative, which means the dependency is not true and is anti dependency.
Hence, **distance vector will be (0, 1)**

Case - 2 : Dependency between W and R2

Using delta test we will equate the corresponding subscript of the array,

for the first subscript,
$i = i + \Delta i + 1$
$\Delta i = -1$

for the second subscript,
$i - j = (i + \Delta i) - (j + \Delta j)$
$i - j = (i - j) + (\Delta i - \Delta j)$
$\Delta j = \Delta i$
$\Delta j = -1$

Distance vector = (-1, -1)
but we can't have first non - zero value as negative, which means the dependency is not true and is anti dependency. Hence, **distance vector will be (1, 1)**

Case - 3 : Dependency between W and R3

Using delta test we will equate the corresponding subscript of the array,

for the first subscript,
$i = i + \Delta i - 1$
$\Delta i = 1$

for the second subscript,
$i - j = (i + \Delta i) - (j + \Delta j) - 1$
$i - j = (i - j) + (\Delta i - \Delta j) - 1$
$\Delta j = \Delta i - 1$
$\Delta j = 0$

Distance vector = (1, 0)
In this case the dependency is true dependency. And, **distance vector will be (1, 0)**

2 Solution

The Question is to count inversion in an array using a given number of threads.

Use this command to compile the file : **g++ fileName.cpp -pthread** (In windows).
Then execute - **a.exe n p**
where n is number of elements in the array and p is the number of thread.

Approach : A thread will come take the current index value and increase index by 1 and find inversion related to curIndex - 1 and rest of the elements after it i.e. if curIndex = 2 and size of array = 10 then the thread will come increase curIndex by 1 to make it 3 and calculate inversion on the basis of element at index (curIndex - 1) = (3 - 1) = 2 relative to index 3 to index 9.

How mutual exclusion is guaranteed for index variable :
Before working on index the thread will lock mutexIndex variable using mutex lock and then check whether index is not overflowed, if yes then exit else store the value of index in a local variable increase index by 1 and unlock the variable then perform the logic.

Note : curIndex is stored in a local variable because it is accessed after getting unlocked to initialize the loop variable and it may get change by another thread just after unlocking it before intializing loop variable. The reason behind unlocking it is that we are not writing anything in the array so we can access array in parallel and unlocking it soon increase more code to be executed in parallel by threads.

How mutual exclusion is guaranteed for numOfInversion variable :
Before increasing the value of numOfInversion by any arbitrary thread it locks the variable using mutex then update it and unlocks it to be used by other threads as well.

Performance of the code :

Array Size	Thread Count	T_{mte}	T_{ste}	T_{ste} / T_{mte}
50000	10	1.516000 s	9.57600 s	6.31662 (speed up)
2000	2	0.016000 s	0.01500 s	0.93750 (speed down)
10000	5	0.078000 s	0.0.37500 s	4.80769 (speed up)

Case - 1 :
Number of inversion from multi threaded Execution : 628935689
Time taken by multi threaded execution is : 1.516000 seconds

Number of inversion from single threaded Execution : 628935689
Time taken by single threaded inversion count methodis : 9.57600 seconds

Multithread execution performs **6.31662** better than single threaded execution !!

Case - 2 :
Number of inversion from multi threaded Execution : 1002418

Time taken by multi threaded execution is : 0.016000 seconds

Number of inversion from single threaded Execution : 1002418
Time taken by single threaded inversion count method is : 0.01500 seconds

Multithread execution performs **0.93750** better than single threaded execution !!

Case - 3 :
Number of inversion from multi threaded Execution : 24903415
Time taken by multi threaded execution is : 0.078000 seconds

Number of inversion from single threaded Execution : 24903415
Time taken by single threaded inversion count methodis : 0.37500 seconds

Multithread execution performs **4.80769** better than single threaded execution !!

Point to note : If the array is not huge then the single threaded function does better than the multithreaded version (As in second case) as there is overhead to create and manage thread in case of multithreaded execution.

3 Solution

As per the question we have to implement producer consumer problem for arbitrary values of number of producers, size of buffer and number of consumers.

Use this command to compile the file : **g++ fileName.cpp -pthread** (In windows).
Then execute - **a.exe p b w**
where p is number of producers, b is size of buffer and w is the number of consumers.

Flow of the program :

- Starting from the main, I have created producer and consumer vectors(array) as per the given command line arguements. Also as per b value, size of buffer is updated.
- **Producer Routine** :
  - It has a while(1) loop which ensures that the producer function will run forever till any break condition inside the loop body mets.
  - If the file has not ended, producer will try to lock the mutexBuffer, if locked will wait, else it will continue to execute.
  - If the buffer is full, the thread will sleep using pthread_cond_wait() function and the consumer when consumes an item will signal the producer back using pthread_cond_broadcast() function to start producing .
  - There are a number of checks of whether the file has ended at a lot of place as a thread maybe locked before the file got ended by another producer will be terminated using those conditions.
  - If a producer has successfully avoided all the condition check, then it will take take one number from the file and push it to the buffer.
  - The mutual exclusion is guaranteed using mutex variables mutexBuffer(to lock buffer) and mutexFileInput(to lock file from which we are taking input).
- **Consumer Routine** :
  - It has a while(1) loop which ensures that the Consumer function will run forever till any break condition inside the loop body mets.
  - If file has ended and buffer is empty as well, consumer thread will be terminated by the condition checks within the consumer function.
  - A consumer thread will try to lock mutexBuffer, if locked already will wait, else will continue to execute.
  - If the buffer is empty, the thread will sleep using pthread_cond_wait() function and the producer when produces an item will signal the consumer back using pthread_cond_broadcast() function to start consuming.
  - After avoiding all the condition checks, the thread will pop an item from the buffer and check whether it is prime or not, If prime will append it to the output file

which contains all the prime numbers from the number in input file.

– The mututal exclusion is guaranteed using mutex variables mutexBuffer(to lock buffer) and mutexFilePrime(to lock the while where we are writing prime numbers).

4 Solution

As per the question we have to implement multi-threaded key value library using pthreads.

Use this command to compile the file : **g++ fileName.cpp -pthread** (In windows).
Then execute - **a.exe n**
where n is the number of operations.

Flow of the program :

- Starting from the main, I took n(number of operations) from the command line arguements.
- **Main Routine** :
  - The main function executes the logic of the manager thread.
  - Also there is a helper method in which we basically put operations data to test the program.
  - First I have created worker threads in the main function and then implemented the logic of the manager thread before joining the worker thread so that both worker and manager run in parallel.
  - After creating worker threads, there is a loop which is responsible for enqueue of the jobs in the job.
  - In every iteration of enqueue it calls enqueue function which basically checks whether the queue has crossed the size of 8, if it has then we discard that operation or else we push it into the job queue.
  - The mutual exclusion in the job buffer is guaranteed using mutex variable mutexJobBuffer(to lock buffer).
  - It also has a broadcast method which signals the worker thread in case those are waiting as there must be no jobs in the jobq.
- **Worker Routine** :
  - It has a while(1) loop which ensures that the Consumer function will run forever till any break condition inside the loop body mets.
  - If jobq is empty and all the operations are already inserted into the jobq, consumer thread will be terminated by the condition checks within the consumer function.
  - A consumer thread will try to lock mutexJobBuffer, if locked already will wait, else will continue to execute.
  - If the jobq is empty, the worker thread will sleep using pthread_cond_wait() function and the manager when pushes a job will signal the worker back using pthread_cond_broadcast() function to start operating on the remaining jobs.
  - After avoiding all the condition checks, the worker thread will pop a job from the jobq and operate on it depending upon the type of operation it is.

– The mututal exclusion is guaranteed using mutex variables mutexKVStore(to lock the key value library) and mutexJobBuffer(to lock the jobq).