|  |
| --- |
|  |
| **Arrays v Pointers, Compiler Generated Code Optimization** |
| Brandon Chin |
|  |
| **CSC342 - Instructor: Prof. Izidor Gertner** |
| **4/13/2015** |

|  |
| --- |
|  |

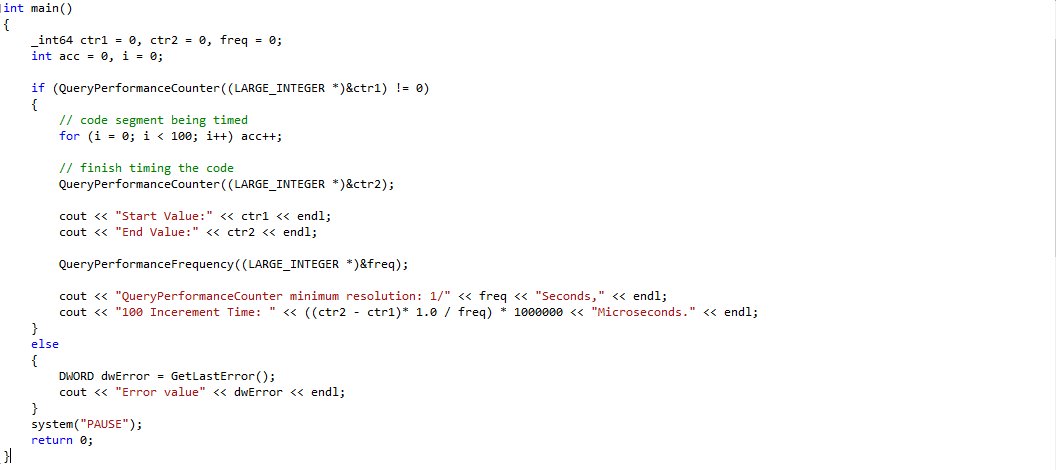
**Objective:**

We are going to create compiler generated assembly code for two separate functions. One which clears an array of some arbitrary size using an index approach, and the other which clears an array of some arbitrary size using a pointer approach. Then we will modify this compiler generated assembly code in order to optimize the duration of the function's execution. Finally, we will organize and compare our results before and after optimization of both methods.

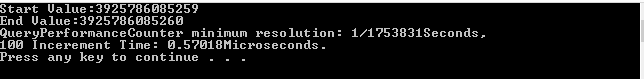
**Code Timer**

Let's first test the code that will measure the duration of a specific segment of execution. The following code is provided to us:





*(CodeTimer, main.cpp)*



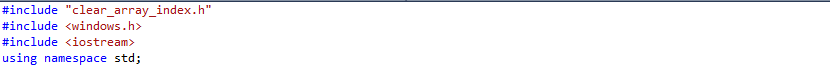
*(CodeTimer, main.cpp - output)*

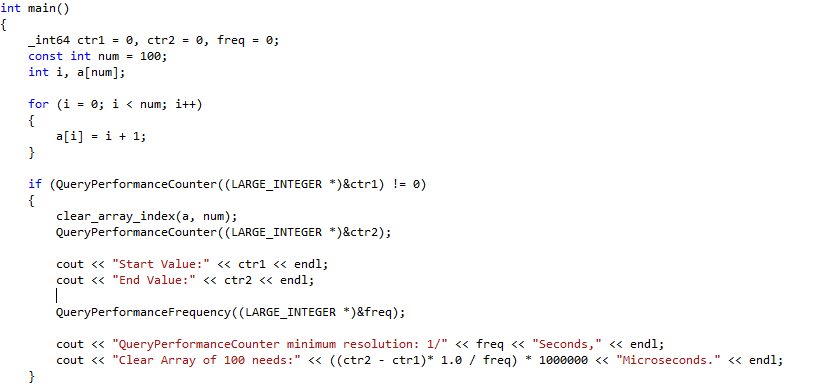
**Clear Array Using Index Approach**

Now we will utilize the code timer that was demonstrated above to analyze the duration of the following clear\_array\_index function. The function takes an array of any integer size, and overwrites all of its values to be 0. This is done by using another variable which will serve as both a counter to control the for loop and an index for each element in the array. At each iteration of the loop, the counter will increment by one, thus advancing to the next element in the array.

 *(clear\_array\_index.h)*

*(clear\_array\_index.cpp)*

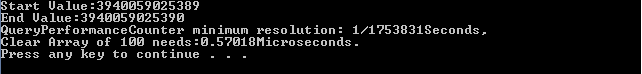




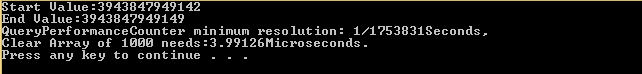


*(ClearArrayIndex, main.cpp)*

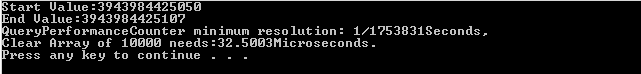
***Array Size: 100***



***Array Size: 1000***



***Array Size: 10000***

******

***Array Size: 100000***

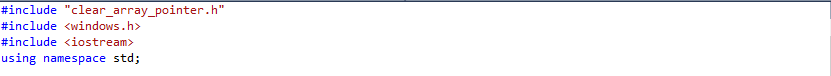


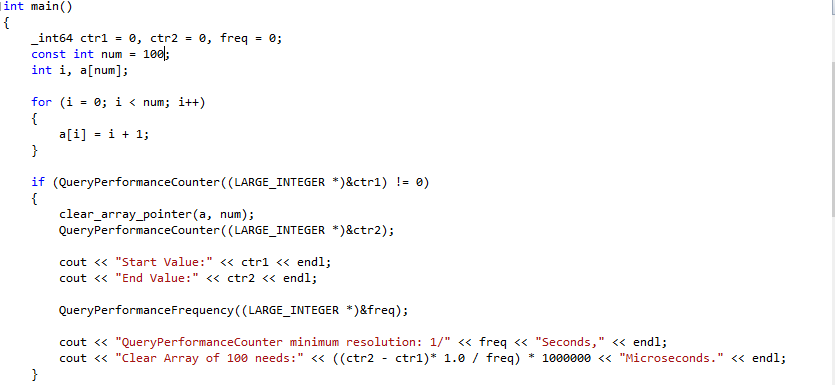
**Clear Array Using Pointer Approach**

This time, we will use the code timer to analyze the duration of the following clear\_array\_pointer function. The function once again takes an array of any integer size, and overwrites all of its values to be 0. However, it does so by using pointers instead of indexing over the course of the loop.

**** *(clear\_array\_pointer.h)*

**** *(clear\_array\_pointer.cpp)*

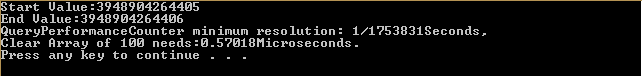
****

****

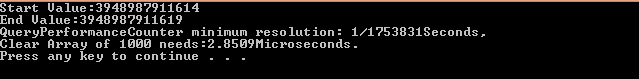
****

*(ClearArrayPointer, main.cpp)*

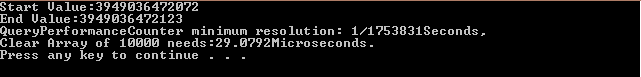
***Array Size: 100***

******

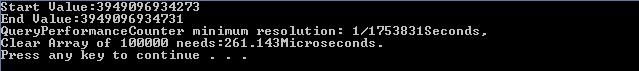
***Array Size: 1000***

******

***Array Size: 10000***

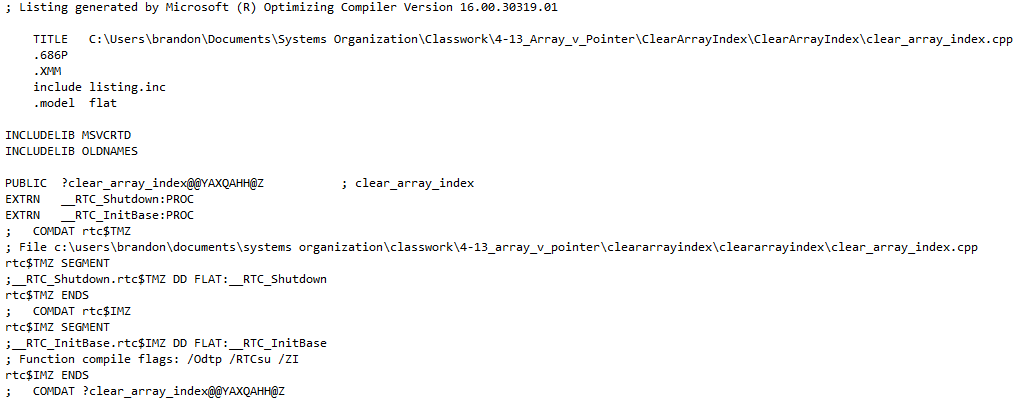
******

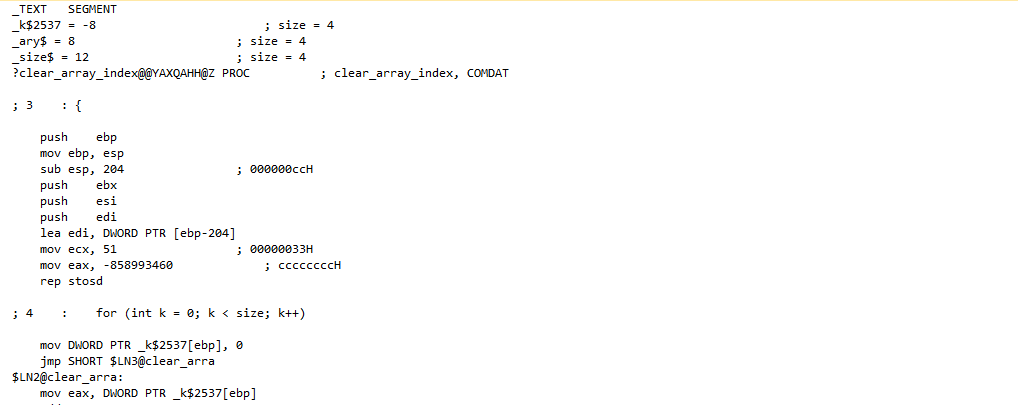
***Array Size: 100000***

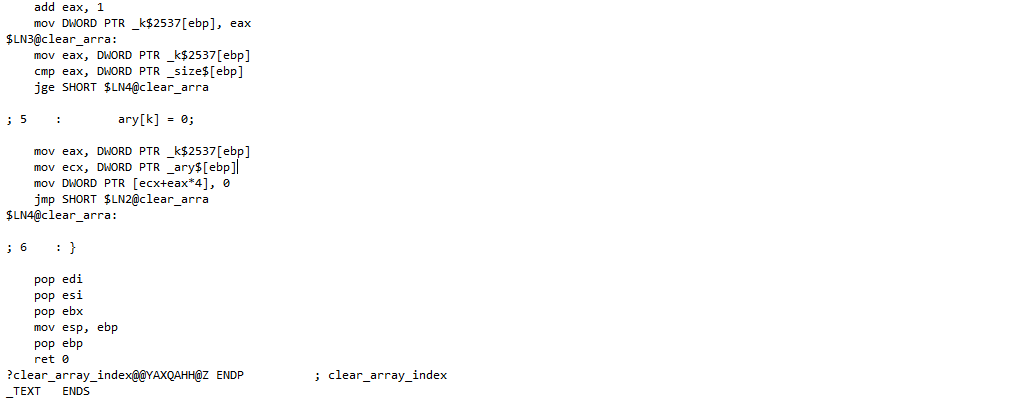


**Compiler Generated Assembly Code**

***Clear Array Index - Assembly***

******

******

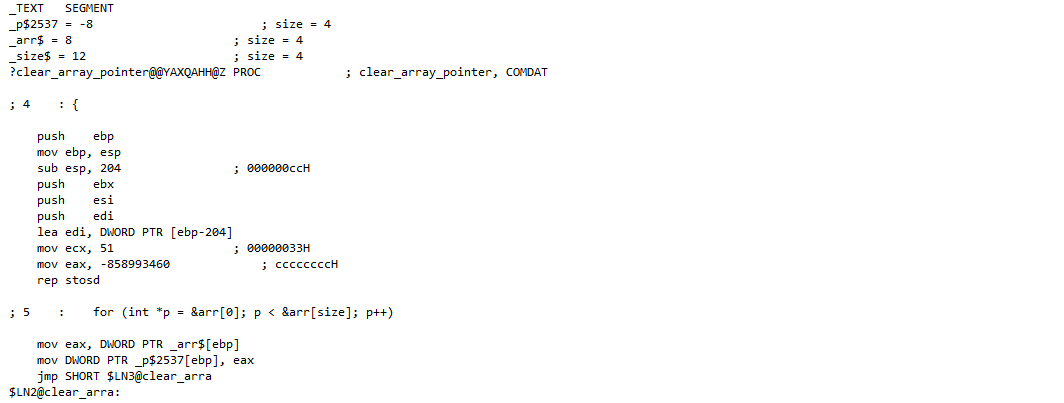
******

******

*(clear\_array\_index.asm)*

***Clear Array Pointer - Assembly***

******

******

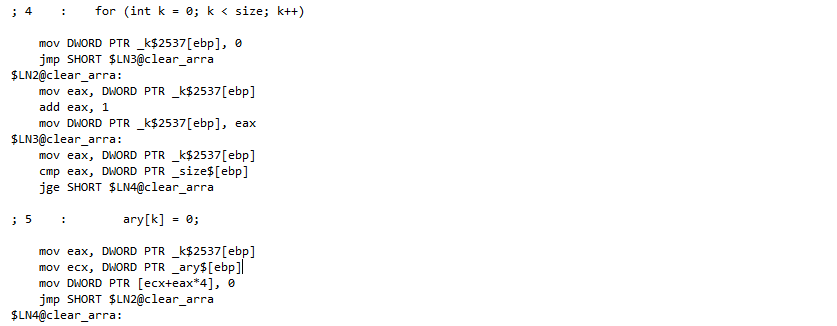
******

******

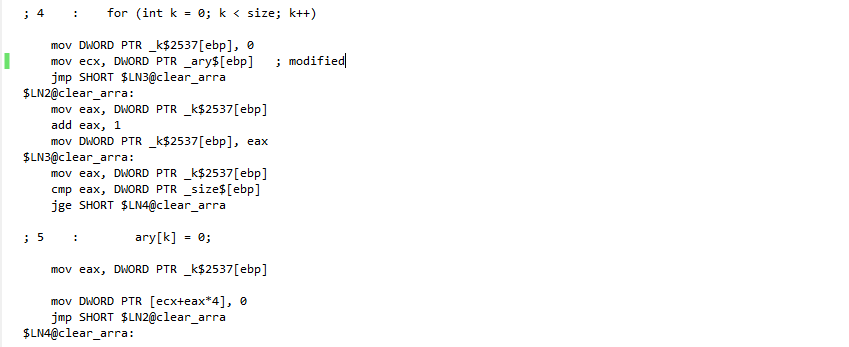
*(clear\_array\_pointer.asm)*

**Clear Array Using Index Approach - Optimization**

***Before Optimization:***

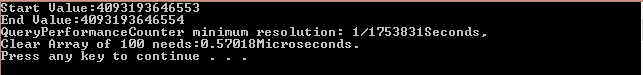
******

***After Optimization:***

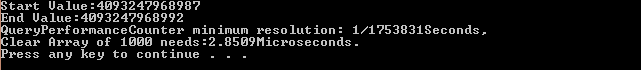


Originally, the register ECX is assigned the same address stored on stack at every instance of the loop. In order to optimize this, I have modified the code by initializing the register ECX before the loop begins. That way, this assignment only needs to be executed once.

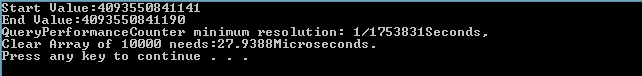
***Array Size: 100***

******

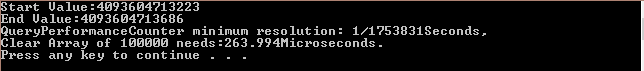
***Array Size: 1000***

******

***Array Size: 10000***

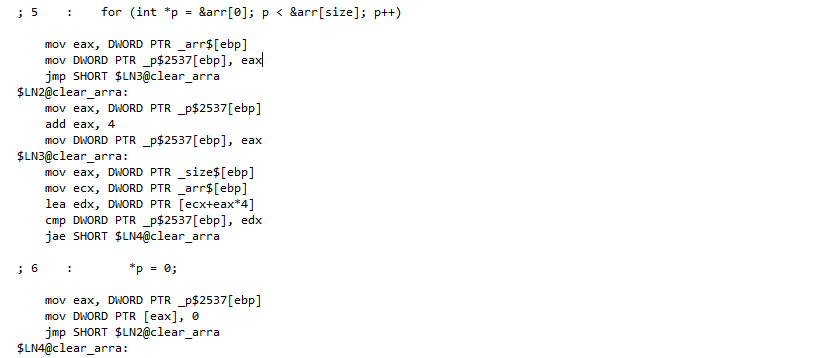
******

***Array Size: 100000***

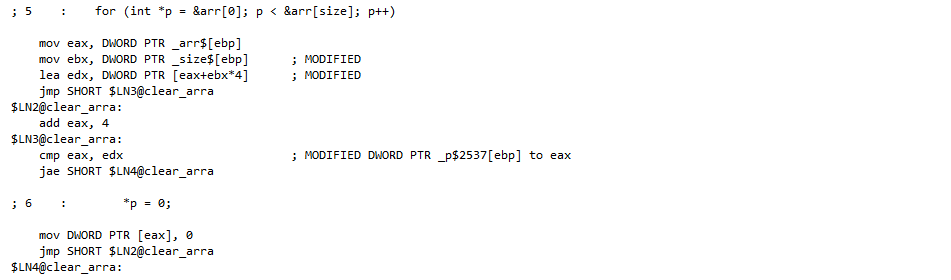
******

**Clear Array Using Pointer Approach - Optimization**

***Before Optimization:***

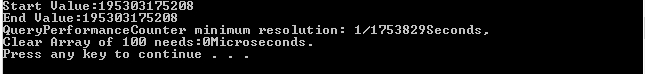
******

***After Optimization:***

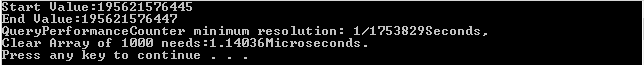
******

In order to optimize this code, a lot of statements are removed. Many registers are assigned values unnecessarily because these values do not change. There are also cases where the same value is written twice, once on stack and once again in registers. I have modified the code to accommodate for these inefficiencies.

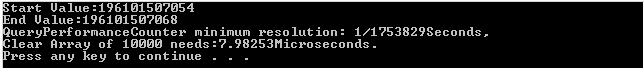
***Array Size: 100***

******

***Array Size: 1000***

******

***Array Size: 10000***

******

***Array Size: 100000***

******

**Results**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Size N** | | | |
| **100** | **1000** | **10000** | **100000** |
| **Index (Before Optimization)** | 0.57018 | 3.99126 | 32.5003 | 336.406 |
| **Index (After Optimization)** | 0.57018 | 2.85090 | 27.9388 | 263.994 |
| **Pointer (Before Optimization)** | 0.57018 | 2.85090 | 29.0792 | 261.143 |
| **Pointer (After Optimization)** | 0.00000 | 1.14036 | 7.98253 | 88.9482 |

*(Table #1)*

*(Graph #1)*