**Rajat Sethi – CPSC 3300 – Project 2**

**Loop Optimization Techniques:**

* Loop Peeling [1]:
  + Loop peeling is the act of optimizing a loop by removing unnecessary variables and indices outside of the loop.
  + For example, imagine the following code block:

int j = 11;

for (int i = 8; i > 0; i--) {

arr2[i] = arr[i] + arr[j];

j = i + 2;

}

* + Through peeling, the code block is re-written to remove the ‘j’ variable.

arr2[8] = arr[8] + arr[11]

for (int i = 7; i > 0; i--) {

arr2[i] = arr[i] + arr[i + 2];

}

* + Overall, this process helps optimize a bit of time since the computer doesn’t need to create another variable in memory and risk a few cache misses.

* Loop Fusion [2]:
  + Loop fusion is the act of fusing two separate loops with the same conditions together.
  + For example, imagine the following code block:

for (int i = 0; i < 20; i++) {

printf(“%d\n”, arr[i]);

}

for (int i = 0; i < 20; i++) {

printf(“%d\n”, arr2[i]);

}

* These two loops can be easily combined to make one large loop of the same conditions, as shown with this re-write.

for (int i = 0; i < 20; i++) {

printf(“%d\n”, arr[i]);

printf(“%d\n”, arr2[i]);

}

* While these types of optimizations aren’t particularly significant, they can still be found in several locations and accumulate to a larger time save.
* Loop Unrolling [3]:
  + For some loops, it is more efficient to have the commands repeated several times instead of branching back to the start. For example, imagine the following code block.

for (int i = 0; i < 3; i++) {

printf(“One More Print\n”);

}

* + Instead of using a for loop, the compiler saves a MIPS instruction by unrolling the loop, creating the following code block.

printf(“One More Print\n”);

printf(“One More Print\n”);

printf(“One More Print\n”);

* + With this change, the for loop is removed, but the result remains the same in assembly. In fact, there is a slight boost because there is no branch instruction anymore, so a minor optimization does occur here.

**Loop Carried Dependencies [4]:**

In a loop, it is often possible for a data-flow dependence to occur, namely the RAW dependency. For example, imagine a loop with the body.

arr[i + 1] = arr[i] + x;

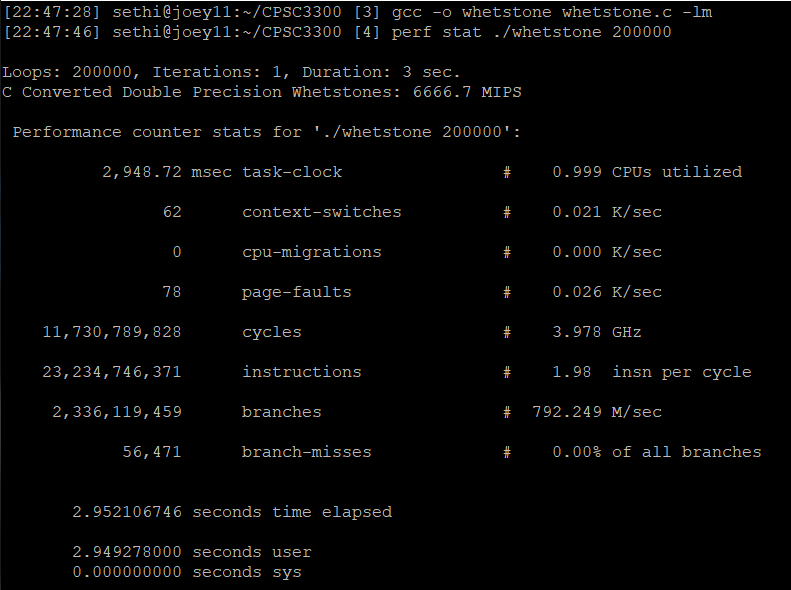
In the first iteration, the value at arr[i + 1] is written to. In the next iteration, arr[i + 1] is read, creating a RAW hazard. This forces the processor to stall for every iteration, avoiding the RAW dependency at the cost of time.

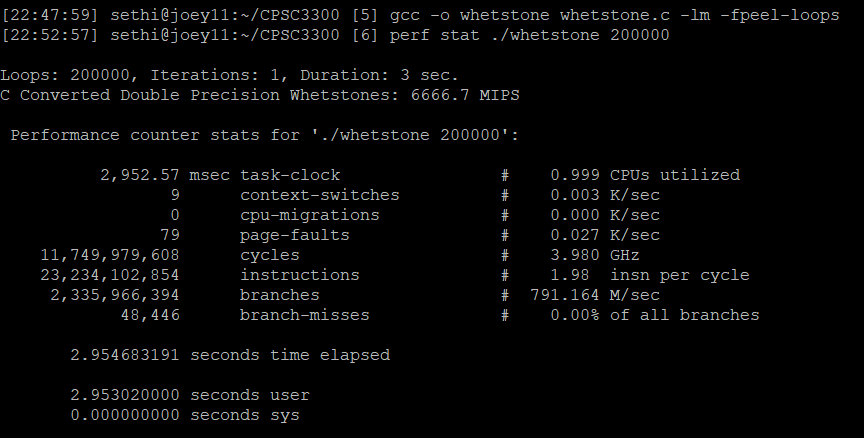
**Array-Element Aliasing [5]:**

In a loop, passing a pointer as a value can cause slowdowns, since the compiler has problems determining whether the object passed in is a variable or an array. For example, a function with header “foo(int \* object)” could be passing in either an integer pointer or integer array. Because of this ambiguity, loops that use this “object” cannot try optimizations because it doesn’t understand the context of the pointer. The program does not want to accidentally access memory it should not, simply because it thought an integer value was an array.

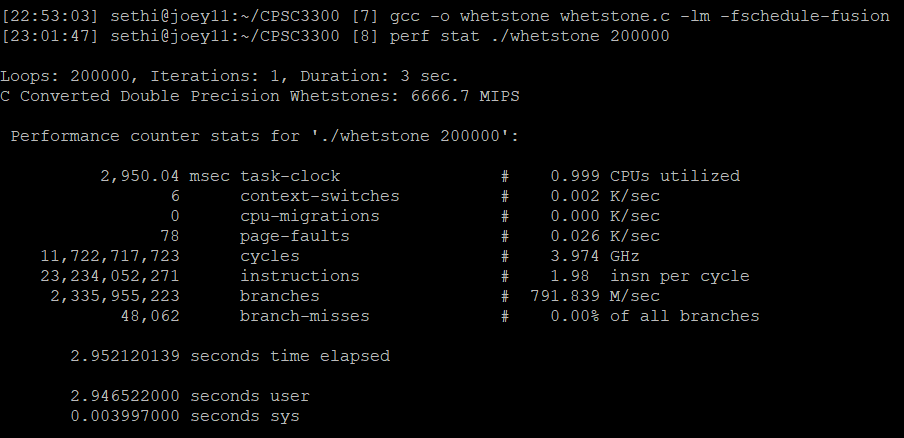
**Running Optimizations [6]:**

* Control Case: -O0 default optimization.

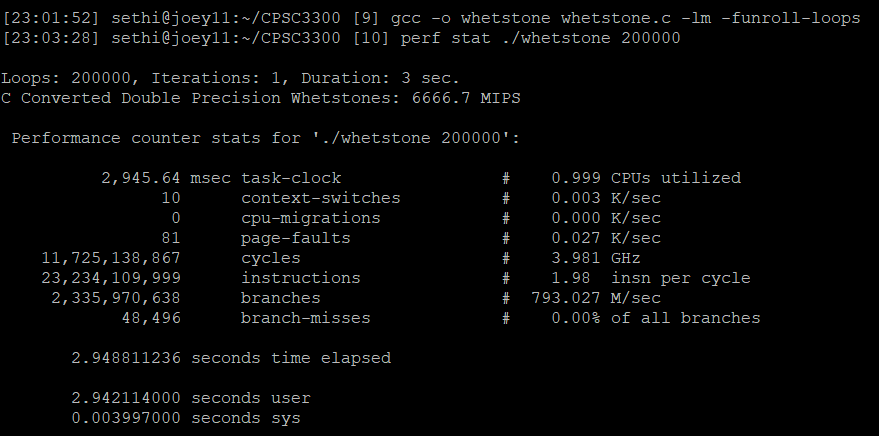


Case 1: Loop Peeling 

Case 2: Loop Fusion



Case 3: Loop Unrolling



**IEEE Citations:**

[5] *3.7 Aliasing and Parallelization - Oracle Solaris Studio 12.3: C User's Guide*, 01-Jan-2012. [Online]. Available: https://docs.oracle.com/cd/E24457\_01/html/E21990/bjafa.html. [Accessed: 06-Apr-2021].

[1] Litong Song and K. M. Kavi, "A technique for variable dependence driven loop peeling," Fifth International Conference on Algorithms and Architectures for Parallel Processing, 2002. Proceedings., Beijing, China, 2002, pp. 390-395, doi: 10.1109/ICAPP.2002.1173607.

[2] *3.7.2 Loop Fusion (Sun Studio 12: C User's Guide)*. [Online]. Available: https://docs.oracle.com/cd/E19205-01/819-5265/bjaey/index.html. [Accessed: 06-Apr-2021].

[3] “Loop Unrolling,” *GeeksforGeeks*, 19-Feb-2018. [Online]. Available: https://www.geeksforgeeks.org/loop-unrolling/. [Accessed: 06-Apr-2021].

[4] *OpenStax CNX*. [Online]. Available: https://cnx.org/contents/u4IVVH92@5.2:Qv3tWJ-B@3/Loop-Carried-Dependencies. [Accessed: 06-Apr-2021].

[6] *Optimize Options (Using the GNU Compiler Collection (GCC))*. [Online]. Available: https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html. [Accessed: 06-Apr-2021].