6.172 Project 1
Team Members – Natalia Suarez, Adriano Hernandez

Design Overview

We redesigned the staff implementation of rotating bits by splitting up our algorithm into two parts: an "inner" part that rotates a 64x64-bit block and an "outer" part that rotates 64x64-bit blocks within a larger image (using the inner part).

See Figure 1 for an illustration of the outer algorithm. We use two temporary buffers to help rotate blocks as it allows us a place to hold them while we move blocks around. They also give us better internal rotation cache performance since their bytes are consecutive in memory. They are all pre-allocated and consecutive in memory.

On a high level, the outer algorithm traverses the array similarly to the "snailspeed" staff solution. We pick blocks on the top left quadrant, rotate them and place them into their final locations, rotate and place the overwritten blocks onto their corresponding final locations, and so on in a four-block loop. For images with odd numbers of blocks per side, *N*, we use the top left rectangle whose width is *ceiling(N / 2)* and whose height is *floor(N / 2)* and rotate it onto the top right rectangle whose width is the previous rectangle's height and whose height is the previous rectangles width (and so on). For these odd images we rotate the block at the very center independently at the end.

The inner part of the algorithm occurs while the 64-bit block is in the buffer. While in the buffer, it is rotated internally. Subsequently, the outer rotation places it in its final location.

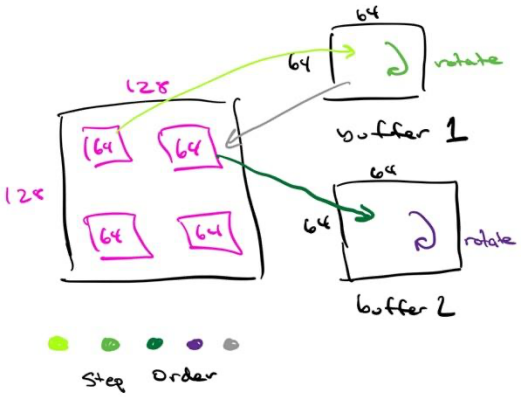

Figure 1

The algorithm we implemented for the internal rotation is similar to one suggested in 6.172 recitation, but with a slight modification to the order and shift size (as suggested on Piazza):

1. Row r is shifted r to the left (ex. row 2 is shifted 2 bits to the left)
2. Column j is shifted j down (ex. column 2 is shifted 2 bits down)
3. Row r is shifted r + 1 to the left (ex. row 2 is shifted 3 bits to the left)

These shifts are effectively "rotations" (that wrap) as described in recitation. We converted the blocks into big endian form before rotating and then back afterwards. In big endian form it was easy to use 64-bit machine words to shift left (by bit shifting to the left and taking the or of the leftmost bits shifted to the right a corresponding amount). We used the divide and conquer algorithm from recitation to efficiently shift the bit-columns down.

Final Release Plans

We spent a lot of time creating a testing framework to help test the correctness of our code, so we hope that this allows us to test future implementations faster and save us time. For example, we exhaustively test all 64x64 bit images that have a single one in them.

We are also pleased with the current state of our implementation. We expect it to run bug-free for correctness and feel as though it has a good basis to allow for further optimization. We believe that we have sufficiently optimized the internal rotation to the point where our current bottleneck is the outer rotation.

We are considering increasing the size of our blocks, ensuring that proper vectorization is being used, and exploring different algorithms to optimize for cache hits. Overall, cache performance seems to be the most important metric for us to optimize for. We will keep portability in mind as we continue to optimize (ex. cache size can vary).

Help Acknowledgment

We want to acknowledge the 6.172 recitation slides for helping us start off with a base algorithm for our internal rotation along with the 6.172 teaching staff for help with debugging and algorithm questions.

We also want to thank our Piazza classmates. We thank the poster(s) of *Proof of RCR* (note @256) for helping us redesign our algorithm, and we thank the poster(s) of *Potentially Useful Bitmaps* (note @245) for an invaluable addition to our 64x64-bit correctness testing repertoire.

Work Log

		Duration		
		Adriano	Natalia	
September 15th	2:00 PM	1	1	Outlined our thoughts on the team contract
September 15th	7:00 PM		2	Created team contract
September 16th	8:00 AM	1	1	Read the handout carefully and wrote down some reminders.
September 16th	8:00 PM	2	2	We thought about how to quickly do 64x64 rotations and designed an algorithm.
September 17th	8:00 AM	1		Worked on outer rotation block-rotation
September 19th	5:00 PM		2	Worked on inner 64x64 rotation on simple algorithm described in class
September 20th	8:00 AM	2		Worked on creating a backup slower version of the inner 64x64 rotation
September 20th	8:00 PM		1	Worked on updating algorithm to use new divide and conquer algorithm from recitation
September 21st	4:00 PM		2	Finished implemeting the new divide and conquer algorithm but was not working
September 21st	8:00 AM	2		Worked on the backup 64x64 rotation; did not fix, but added a ton of asserts.
September 21st	6:00 PM	2		At OH getting help with 64x64. Added various tests in my stash branch (adriano/stash).
September 21st	7:30 PM		2	At OH, added endian swap to divide and conquer algorithm which fixed most bugs
September 21st	11:00 PM	2		Fixed some bugs and added more tests. The tests are not exercised. Worked on block rotates.
September 22nd	5:30 PM	2.5	2.5	Debugged our implementation at OH. Went from buggy to Tier 39.
September 22nd	10:00 PM		2	Worked on beta write up