Tallack Graser
HW 3
CIS 410

Data Reorganization

1) The purpose of data reorganization is to ensure that your program is performing at the best it can be. This is because the performance bottleneck is due to data movement and not just computation. How do we ensure that our data is moving in the most efficient way possible? We must make sure that it is organized in a way such that the data is access and moved in the most cost efficient way.

However, taking the time to reorganize the data may take a long time, so long in fact that reorganizing the data so that it can be used more efficiently that it might actually slow the program down. However, by using a parallel pattern to reorganize the data we can lower the overhead of the reorganization, and use data reorganization more often without worrying of decreasing our performance time.

2)

a) The memory block that is housing the attributes of this struct can only be accessed by one thread at a time. Hence, accessing the data will cause latency in our program.

b) This will obviously cause bad performance. While one thread is accessing the information, the other two threads will be waiting.

c) You would resolve this problem by partitioning (a pattern that goes hand in hand with the stencil pattern) the block in such a way that all three elements can be accessed individually, allowing all the threads to be computing information simultaneously.

3) Pretty much addressed in the previous question.. reorganizing the data by partitioning it in some way is a very helpful way to prepare for a stencil computation, because it eliminates thread blocking in many cases.

Stencil Pattern

1)

a) Lets say we are working with a given thread. As described in the question, each thread is in charge of a 500X500X500 chunk of data. This means that any cells that are in direct contact with the outermost cells in this chunk are ghost cells to this thread.

b)

1,000/250=4----→meaning there will be 4x4x4 threads needed.
Hence, we will need 64 threads

1,000/125=8-----→meaning there will be 8x8x8 threads needed.
Hence, we will need 512 threads

c)
First lets look at the data:

# of threads	Size of chunk	# of ghost cells	Ratio: ghost cells/non-ghost cells
8	500^3	12,048,064	.012
64	250^3	24,192,512	.024
512	125^3	48,772,096	.048

# of ghost cells computed by [(X)^3]-(size of chunk)](#of threads), where X is (the length of one of the edges+2) of the given chunk you're looking at.

If you look at the above table its easy to conclude that the smaller of chunk you use, the more ghost cells you have. From the class slides, this poses a trade off. The higher the ghost cells/non-ghost cells ratio is the less communications and more independence you will have, however, you will have more redundant computation and will use more memory. In regard to computation, it might now take longer (with more threads and smaller ghost cells per thread) since we must be accessing memory more often, and must wait and update the ghost cells more often.

2)By using a larger/deeper halo you are achieving two major things: allowing for less communication, and gaining more independence. However, the larger/deeper you make your halo, the less threads you will be able to use (which will be slower than finding the proper amount of threads to use). This means that you must find a happy medium, by using the proper amount of parallelism and to do the computation in a reasonable time, without completely taxing your memory.

3) In order to parallelize this nested for loop we must implement the recurrence pattern. That is, we must find a separating hyper plane and compute through the date in a parallel direction to this hyper plane.