Data Alignment Summary

Liam Godin

2021-03-01

Memory Access Granularity

- Programmers see memory as a simple array of bytes
- Processors access memory in chunks of bytes (2, 4, 8, 16, even 32)
  - These chunk sizes are called memory access granularity

Alignment

- If a processor's memory access granularity causes a misalignment when reading from memory, the processor will have to do more work, accessing the memory more times than if it were aligned correctly

Lazy Processors

- In order to handle an unaligned memory access, a processor has to read the first chunk of the access, shift out bytes that were not requested by the access, do the same with the second chunk, then combine the results into a register
- Some processors will not do this, especially older processors. Some are specifically designed to not be able to handle unaligned memory accesses.

Speed

- Speed penalties are massive for unaligned accesses
- decreases in speed are much more apparent when more bytes are accessed at once

Atomicity

- atomic instructions on unaligned addresses do not work, as the code required to run in order to access the location destroys the atomicity of the operation

Altivec

- Altivec does not throw an exception about an unaligned address, and instead ignores the issue, continuing to operate on the wrong address.

Structure Alignment

- Oftentimes, compilers will pad the size of a struct in order to allow them to reside at aligned addresses

Conclusion

- Unaligned memory accesses can destroy performance due to expensive alignment exception handling
- atomically storing to an unaligned address can cause your application to lock up
- Passing an unaligned address to Altivec can cause corruption of data or yielding incorrect results