**Task 1**: 3C’s are three types of cache misses, a cache miss means the data must be accessed from the slow main memory therefore the number of 3Cs that occur can be used as a benchmark for performance.

* **Compulsory** or cold misses are start up misses. Cache are empty on start up therefore the number of compulsory misses are equal to the number of caches. It can be somewhat mitigated by pre-fetching, a technique a gathering data before it is needed.
* **Conflict** misses occur when data that was previously in the cache was evicted for new data. This can be mitigated by increasing the number of cache lines (associability) of the the cache, however that comes with the price of a slower cache (more work to find data). It can be measured by comparing misses with the target associality to full associative.
* **Capacity** misses are when the cache is not large enough to store all the important data for a program. The simple solution is to increase the size of the cache, though this leads to longer access times. It is measured as the remainder after subtracting the compulsory and conflict misses from the total misses.

**Task 2**: The claim that stochastic (variance driven) and statistical (mean driven) models provide insight to program performance seems well supported enough. However the source for the claim that these models do not provide insight on how to improve the program seems suspect. The source for that claim is focus on deterministic models versus non-deterministic models. It also seems to directly contradict the claim in some places by asserting that the non-deterministic model is effective for addressing certain parallel programing behaviors. Most of these source do seem focused on measuring performance and not on actually improving it, so the claim feels at least moderately supported.

**Task 3**: Amdahl’s Law certainly seems well known enough with over 5000 citations. Amdahl’s law seems to basic come down to this: the more of the program that can be parallelized the more it benefits from increasing processors. Also the upper bound on speed up comes from the amount of work that must be done in serial. This claim is accurate for fixed problem size but in reality as more resources become available problem (often data sets) sizes become larger, increasing the proportion of the problem that is parallelizable.

**Task 4,5,6**: In order to achieve peak performance the roofline model looks at a series of bounds then prioritizes them in the following order. It focus first on achieve maximum computational performance then on reducing the necessary operation complexity to reach that maximum performance.

1. Improve ILP (instruction level parallelism) or apply SIMD (single instruction, multiple data). ILP is the basic concept of allowing a compiler to parallelize as much sequential code as possible. The paper doesn’t go into much depth on the ways to achieve this except to brief mention that on superscalar processors (multiple instruction per clock cycle) one can unroll loops (calculate multiple iterations of the loop simultaneously). SIMD is when addition is applied to multiple data in a single operation, for example reducing the brightness of the pixels in an image (all the rgb values being reduced by a constant amount). Instances of code that can use SIMD are not as common as ILP.
2. To take full advantage of the floating-point capabilities of the computer and thus reach maximum performance, the code needs to have a significant number of its instruction be floating-point operations. Additionally, ideally there will be an even split of multiplication and addition operations as many computers have equal amounts of adders and multipliers.
3. Restructure loops for unit stride accesses. A unit stride is structuring an array such that elements are only one “unit” away from each other. It increases the locality of the memory and allows for better prefetching.
4. Ensure memory affinity: make sure that relevant data goes onto the same chip as the processor responsible for the thread which needs that data. This ensures that the processor doesn’t need to access data that is stored on another microprocessor chip.
5. To reach highest performance the program must do prefetching itself in addition to relying on the hardware. In a loop a basic calculation on how much to prefetch is (penalty/iteration time). Assuming this number is greater than zero, there will always be misses at the beginning of the loop.

In Figure 2c you can see the that as the steps are taken in order, first we raise the “flat” (computational speed) part of the ceiling, then when we address memory bandwidth we push the “pitched” part to the left reducing the necessary computation complexity needed to hit peak performance.

**Task 7**: Dynamic operational intensity that increases with respect to problem size is more complicated than fixed operational intensity. Cache performance is directly related to operational intensity. Applying 3Cs to operational intensity:

* Compulsory misses set the min memory traffic and max operational intensity.
* Avoiding conflict misses reduces memory traffic via array padding and cache line addressing.
* Capacity misses can be mitigated via non-allocating store instructions. - This prevents unnecessary cache overwriting, reducing traffic.

Kernel operational intensity should be optimized before anything else.

**Task 8**: The four multicore computers for demonstrating the roofline model are:

*Intel Xeon:* (dual) Front side bus (complicated) connecting north bridge chip and off-chip memory control unit (MCU). Front side bus can consume up to half of bandwidth (limitation) - a snoop filter can help mitigate this limitation. Uses cache

*Opteron X4*: MCU on chip. High peak flop performance. L3 cache on chip. “Glueless multichip system”. L

*Sun UltraSPARC T2+:* MCU on chip. Twice as many cores as others. Multi-threaded 8x per core. Highest bandwidth due to dual memory controller units. Incapable of Multiply/Add balancing - reduces effectiveness of roofline model for some tests.

*IBM Cell*: MCU on chip. Highest clock rate at 3.2GHz. Unusual heterogeneous design with 8 unique synchronized elements and local memories. Challenged by porting programs. Flop performance better because of DMA use.

High bandwidth yields lower operational intensity required at the ridge point. The kernels applied will be the “Seven Dwarves” numerical methods which are significant to computational intensity despite varying technological growth or parallelization. These methods tie old algorithms to the future and invite extensive cache optimization. For this demonstration, the methods specifically help with load balancing.

**Task 9**: The four methods in question are:

*Sparse Matrix Vector Multiplication (SpMV)*: Solving y = Ax where A is a sparse matrix, and x and y are dense vectors. Flop performance is an issue. Memory should be optimized for all computers looking at ridgepoint.

*Lattice-Boltzmann Magnetohydrodynamics (LB- MHD)*: High operational intensity. Irregular memory access like SpMV. x86 helps with Flop performance. (T2+) does this best not needing memory optimization.

*Stencil:* (Imagine Kinex toys) Construct a point from information from its neighbors. High operational intensity.

*3D Fast Fourier Transform:* Recursive Fourier transform. Operational intensity is a function of problem size. Capacity is the biggest issue for *IBM* *Cell* with no cache.Issues for the computers include memory behavior, SIMD instructions, compiling, prefetching, and DMA scheduling.

**Task 10:** Categorizing the results by computer:

*Sun T2+*: Lowest ridge point. Easy to program due to large bandwidth and simple cores. Best optimization through compiler and good threading. Not enough cache space for stencil - conflict misses often. Even bound between memory and computation.

*Intel Xeon*: Highest ridgepoint, lowest unoptimized roofline area. Difficult to program - complicated code in C with many SIMD instructions. Almost entirely memory-bound.

*Opteron X4*: Second highest ridge point. Very similar to Xeon with less complexity in memory behavior without dual busses. Almost entirely memory-bound.

*IBM Cell*: Second lowest ridge point. Compiler needs the help of assembly language to exploit SIMD instructions (limitation, complexity). Complicated memory behavior with sparse addressing (limitation). DMA does prefetching work (positive). Even bound between memory and computation.

**Task 11**: The key concepts of Kernel optimization are:

*Memory Affinity*: Prevent fetching from another socket in DRAM memory.

*Long unit-stride accesses*: Structure loops for max prefetching and TLB miss reduction.

*Software Prefetching*: Max simplification of memory behavior analyzing cache space, local memory space and loop complexity.

*Reduce Conflict Misses*: Array padding for memory affinity to improve cache hit rate.

*Unroll and Reorder Loops*: for memory affinity, improve code elegance, reduce register use/pressure, assist SIMD instructions.

*“SIMD-ize” Code*: SSE intrinsics to augment x86 implementation.

“*Compress data structures (SpMV only)*: optimize sub-block size for available bandwidth.

**Task 12**: The main benefits, despite belief, of the roofline model are:

* Accounting for caching and prefetching.
* Variable operational intensity via cache size can be regulated.
* Long memory latency in prefetching and some hardware implementations accounted for via memory optimization.
* Integer vs. float complexity mitigated by converting to integer in some cases.
* Multi-core improves unicore roofline modeling.
* A diversity of kernels can be encompassed on one model.
* Multi-core modeling encompasses cache use and misses well.
* A diverse range of programs outside of floating point are supported.
* DRAM use can be bypasses via cache and memory behavior manipulation - it is an optimization option.

Short Summary: The Roofline Model offers optimization suggestions for a variety of computer scientists. It is superior to variance and mean driven models through validation of the 3Cs: compulsory misses which determine minimum memory traffic and maximum operational intensity where the cache is unpopulated are optimized with prefetching – this is the starting point of optimization; conflict misses caused by excessive cache overwriting are considered with cache size and memory hardware optimization – with consideration of memory affinity as well; and capacity misses where cache and local memory cannot support the size of a program can be optimized with non-allocating storing, traffic reduction and cache augmentation. The model considers Amdahl’s law that parallelization benefits are limited to the ratio of parallel to series of the program. When using the model, some steps to “raise the roof” are improving instruction level parallelism, balancing floating point capability, unroll loops to unit stride access for memory affinity, pre-fetch with memory affinity, and checking for guaranteed loop misses. The “roof” is the current limitation of the computer in question given a kernel is applied. If the kernel is limited by a flat roof, computational optimization is needed. If limited by the slope, memory optimization is needed. Lower roofs must be achieved before higher ones typically in the order of unit stride and TLP, memory affinity and ILP, then prefetching and floating point balance. To show the effectiveness of this model, four computers created four roofs, then four kernels provide a variety of optimization benchmarks for each. There is a variety of memory and computational hardware employed, so each roof is different and the same problem applied to each yields a different benchmark. Among the hardware highlights are front side busses to MCU, L3 cache level, multithreading, DMA use, and unique synchronization designs. The kernels applied are the Sparse Matrix Vector Multiplication, Lattice-Boltzmann Magnetohydrodynamics, stencil, and 3D fast Fourier transform. These are four of the “Seven Dwarves” numerical methods determined to be useful in computational intensity optimization regardless of quick technological growth in the future. These computers were found to be mostly memory bound with only two requiring significant computational optimization to “raise the roof”. This is determined to be caused by the complexity of dual busses, DMA, memory management, and limited prefetching. Some recommendations for these computers given their ridge points are applying SSE intrinsics to x86 implementation and optimizing sub-block sizing for the unique bus and memory systems. It can be seen that this model appropriately accounts for variable cache and memory size, prefetching, memory latency, floating point complexity, program and kernel diversity, multi-core use, and DRAM use.