Group 1: GPUs are a bad approach

1. not enough parallelism inherent in our code
2. Amdahl's law - won't get a very big speedup
3. extra 1-2 CPU sockets gets you same performance - avoid retraining and refactoring code
4. Put effort into tuning CPU code instead of writing CUDA
5. Caching (larger for CPUs) - GPU register approx same time as L1 cache
6. spend money on the CPU, not on rewriting the code and retraining the coders
7. avoid the bottleneck of moving data back and forth between CPU and GPU
8. Getting GOOD performance on GPU takes much expert

Group 4: GPUs are all we need

1. easier to manufacture multiple small simple cores
2. "Semantic" argument: they'll all be some kind of a GPU in future anyway, CPU --- > gradually GPU
3. CPU advanced functionality less used anyway, so why keep putting money there?
4. Power - GPU is much more sustainable
5. Why cares about retraining - hire new grads : O
6. More interesting to program
7. Thinking in parallel - we do this all the time easily - cooking fried rice example
8. Amdahl's law - the "typical" workload is changing, to data intensive & parallel

Group 2: CPUs SHOULD change to take into account acceleration

1. GPUs DO exist, so CPUs should match what is left
2. PCI bus is a big bottleneck -> "zero copy"
3. "makes sense" to match the hw to the workload
4. CPU improvements have matched "typical workload" but that is changing
5. Specialization - GPUs offer great speedup for anything parallelizable. CPU now needs to be specialized for the non-parallelizable parts. These things do not help: cache, branch prediction, pipeline, SSE
6. Simplify CPU design -> better manuf.
7. Small datasets - GPUs are poor choice

Group 3: CPUs should NOT change to take into account acceler'n

1. Not everything is parallelizable
2. In "redefining" paper: Cost of moving the data CPU-GPU not taken into account
3. Loss of high-level abstraction from GPUs (e.g. LISP) -> they will never do everything
4. Legacy code is a huge factor
5. Integrated chipset technical challenges not yet solved