AMD: AMD Graphic Core Next: Low Power High Performance Graphics & Parallel Compute.

Introduces a new low power high performance architecture called GCN, graphics core next. Is designed to be good at both graphics and computing tasks to encourage developers to develop heterogeneous applications that utilize AMD’s GPU/CPU.

AMD has previously used a VLIW5/4 design with its streaming processors. The 5 indicates the number of math units and the designs excel at many operations in parallel by breaking them up into ‘wavefronts’. A wavefront is a group of 64 pixels/values and instructions to be executed against them.[Difference adv/disadv of VLIW5 vs VLIW4?] (*)

VLIW is good for graphics but not very good for computing, so AMD wants to find a way to integrate the hard part of computing on a CPU offload that responsibility to the GPU.

In the most recently iteration AMD is opts not to use the VLIW designs and instead use a non-VLIW SIMD. The advantage to VLIW SIMD is that it is good with TLP whereas VLIW is good with ILP. (*)

Problem with VLIW include that is complex for optimization, highly dependent on the compiler, and VLIW is hard to schedule ahead of time because there is no dynamic scheduling during execution. But we have to note that all of these deficiencies are only bad for a VLIW processor doing compute tasks but aren’t too much of an issue for VLIW doing graphics tasks (*), main take away is that statically scheduling is bad (when there are dependencies?)

They placed VLIW with a 16-wide SIMD vector processor capable of many different integer and float operations. They did something similar to this before with Cayman but the difference here is that you can do dynamic scheduling with the new CU - compute unit in the GCN.

The reason why this may not have been done before is that designing a dynamic hardware schedule takes die space that would normally be used for functional units, it’s a trade off. Dynamic schedule is useful because you can better schedule and work around dependencies. In the VLIW worst case, something scheduled is completely dependent and blocking the instructions before and after and must be run on its own. The paper notes that GCN is NOT an OOO architecture as within a wavefront the instructions need to be executed in order. But they key is that the CU and SIMDs can choose different wavefront to work on.

Instead of VLIW slots going unused because of dependencies, independent SIMDs can be given entirely different wavefronts to work on. Apparently compiling also becomes easier because of this because it does not have to worry about scheduling and the underlying ISA Is simpler and makes debugging easier (*)

They have also introduced this unit called the scholar unit that executes one-off mathematical operations, i.e. independent operations go to the scalar unit to not waste SIMD time. This means integer options like branches and jump. (*Why useful?) IT improves the latency for control flow operations. Executes 1 IPC so 4 instructions can execute over the time it takes one wavefront to be completed on a SIMD. AMD also introduces the idea of ACE’s, async compute engines so
that they can accept work and dispatch it to the CU’s for processing and there are multiples ACEs on a GPU as the GCN is designed to concurrently work on several tasks. Although OOO, the ACEs have the ability to prioritize tasks so they can be completed in a different order than what they were received. This lets the GNS free resources free up tasks as early as possible rather than keeping them in a nearly-finished state (*) and is similar to how modern in order CPUs do multi-tasking.

On the graphics side there is just information about the PRT, partially resident textures that allow part of textures to be loaded to memory so large textures can be used without a performance hit. GCN implements the underlying features to support C++ and other languages and complex instructions for virtual functions, recursions, and pointers. This is so that developers can use high level languages to code on the GPU. The ISA is also made to support unified memory to allow coders to target the cpu and gpu as programs can reference memory anywhere although the performance hit for accessing off CPU memory is large.