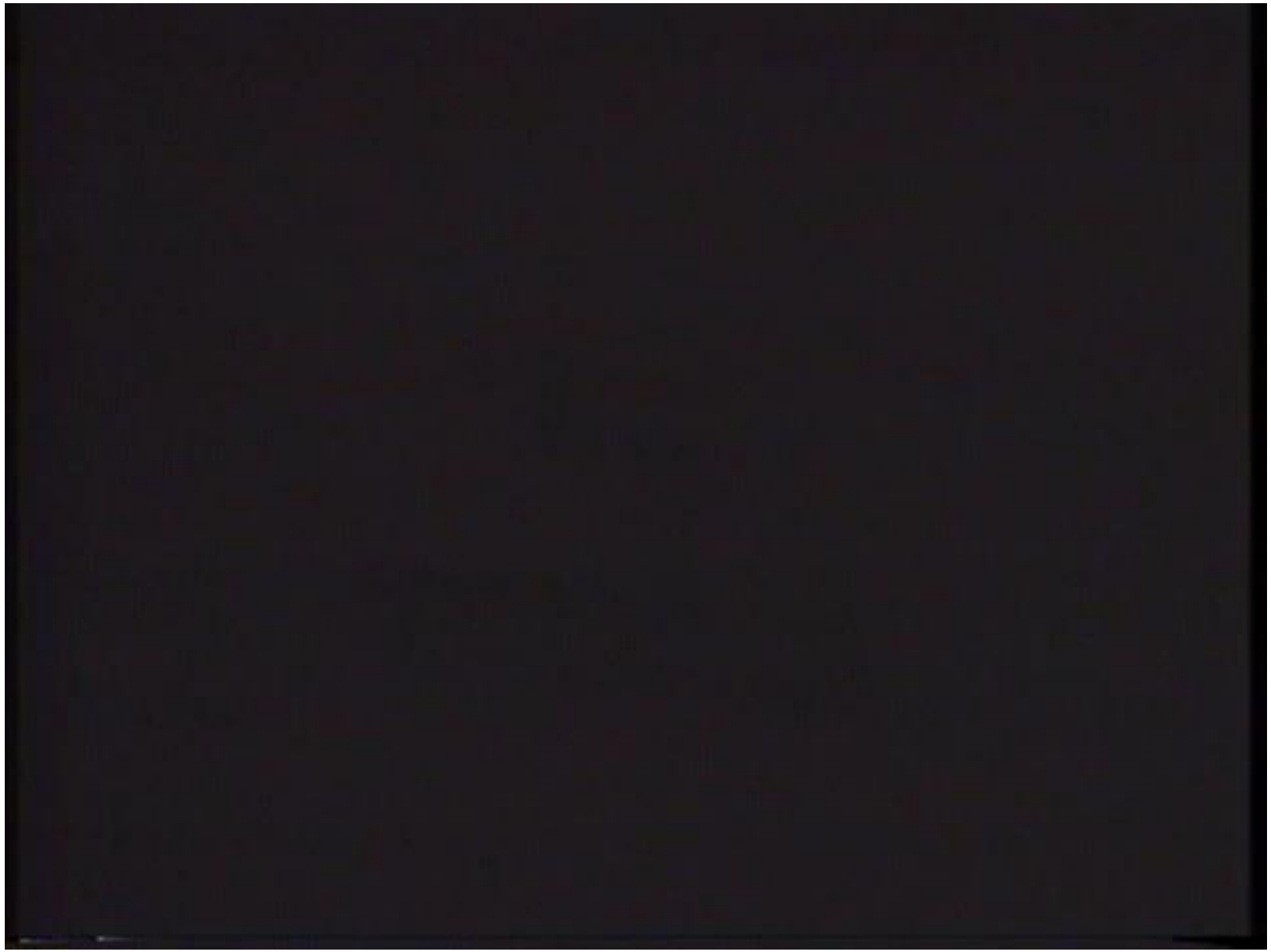
arm

GPUs Intro

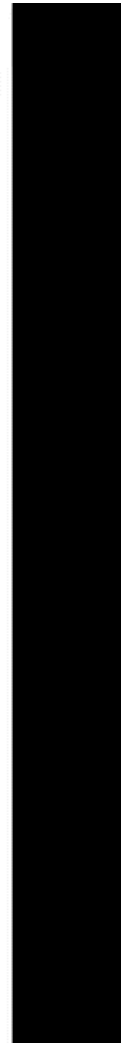
Based on the material *"Arm GPU Architecture course"*

GPUs

GPU Evolution



GPU Evolution 2021



GPU Evolution

- GPUs originally developed as specialized hardware for graphics computation
  - Now used as programmable accelerators for highly data-parallel workloads
- GPU hardware evolution closely tied to evolution in usage patterns
  - Desire for improved visual effects driven by games
  - More realism, more effects, more screen resolution, more frames per second
- Originally a fixed-function pipeline, programmability was added gradually to many stages.
  - Now the bulk of the GPU is a programmable data-parallel architecture.
  - Some fixed-function hardware remains for graphics.
  - New hardware being added to cope with modern workloads, e.g., machine learning

GPU Design Principles

- CPU design is about making a single thread run as fast as possible.
  - Pipeline stalls and memory accesses are expensive in terms of latency.
  - So increased logic was added to reduce the probability/cost of stalls.
  - Use of large cache memories to avoid memory misses
- GPU design is about maximizing computation throughput.
  - Individual thread latency not considered important
  - GPUs avoid much of the complex CPU pipeline logic for extracting ILP.
  - Instead, each thread executes on a relatively simple core with performance obtained through parallelism.
    - Single instruction, multiple threads
- Computation hides memory and pipeline latencies.
- Wide and fast bandwidth-optimized memory systems

SIMT vs SIMD

SIMT

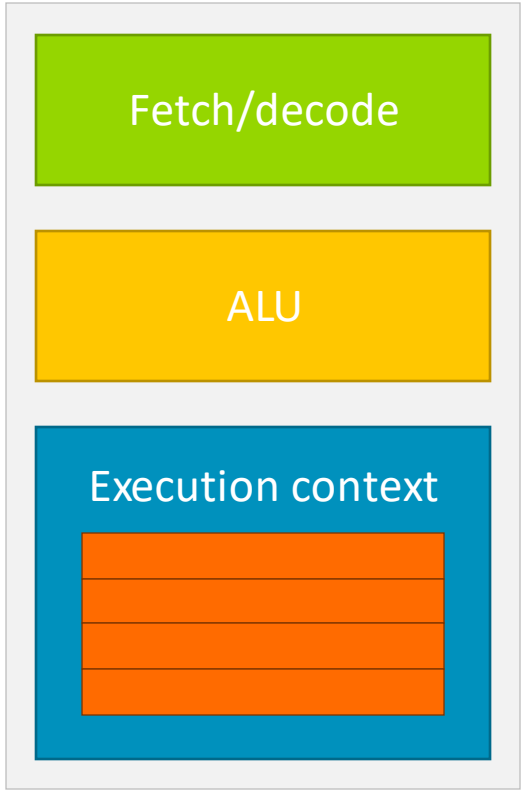
- Single instruction, multiple threads
- Takes advantage of data-level parallelism
- Can be considered a constrained form of multithreading
- Many threads each with their own state
  - Operating on scalar registers
  - With their own local memory

SIMD

- Single instruction, multiple data
- Takes advantage of data-level parallelism
- Can be considered a constrained form of vector processing
- One thread operating on vector registers

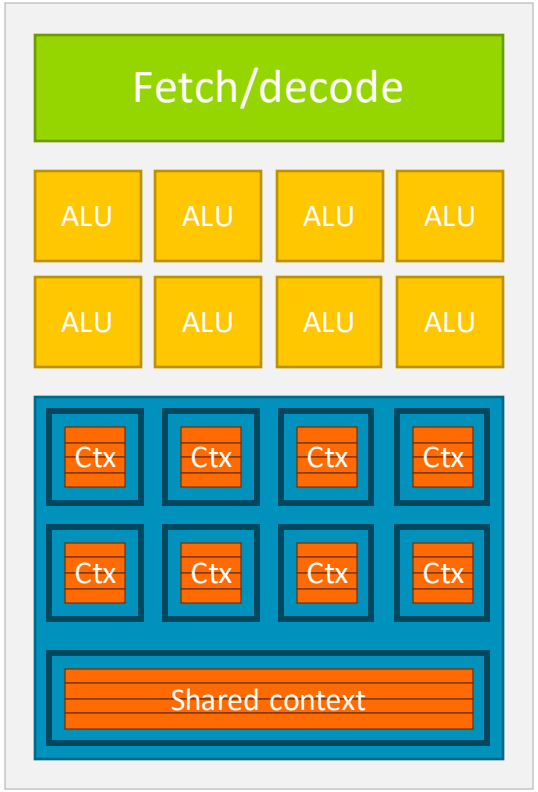
One Processing Element

- A single thread runs on a simple processing element.
- Short pipeline
- The execution context consists of the thread's state.
  - E.g., registers and local memory
- But fetch and decode are costly.



Multiple Processing Elements

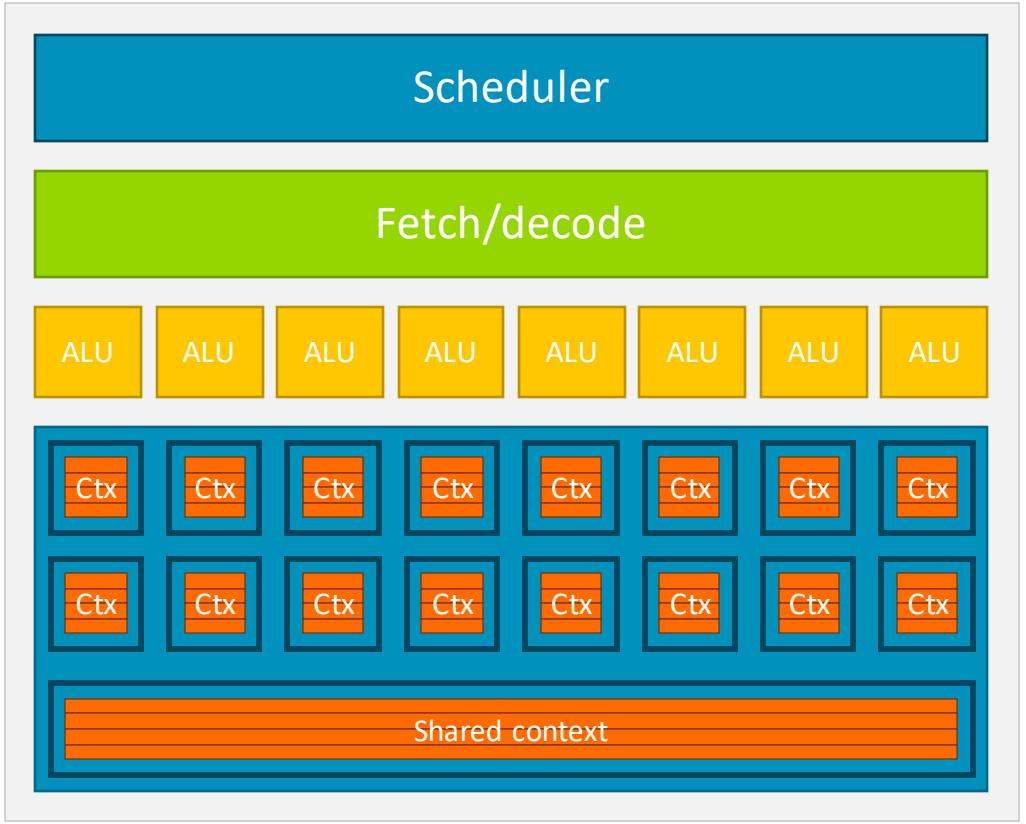
- SIMT execution of threads
- Cost of fetch and decode spread across all threads in a work group (OpenCL terminology)
- Each thread has its own context.
  - And some shared context
- Multiple functional units for parallelism
  - One per thread for cheap units (e.g., simple ALU)
  - Fewer expensive units than threads (e.g., sqrt)
- However, stalls are costly.



arm

Minimizing Stalls

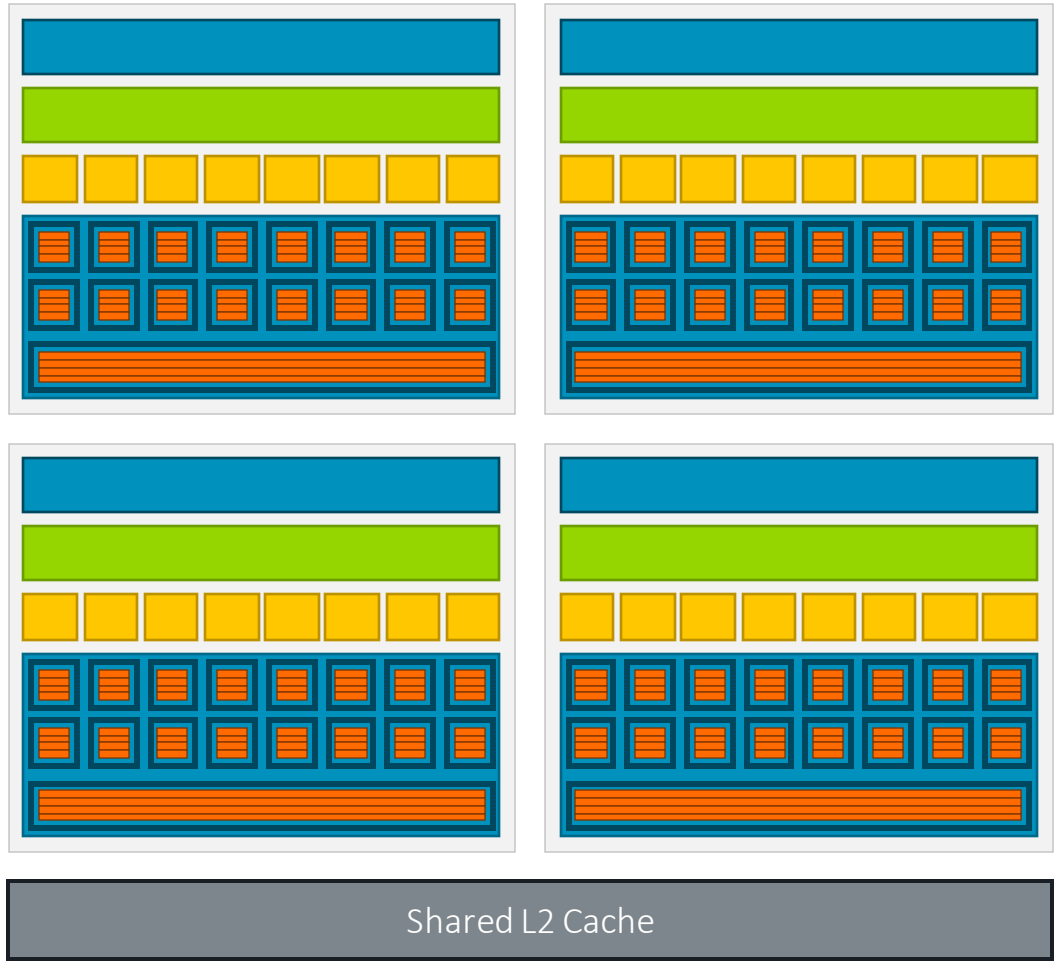
- Include support for multiple work groups
  - Each is independent of all others.
  - And this is guaranteed by the compiler.
  - Which means there is no fixed ordering of groups.
- A scheduler chooses work groups to run.
  - Maintains a list of ready work groups
  - Makes a choice each cycle
  - Some GPUs can schedule more than one work group per cycle.
  - Hides latency when a work group stalls
- This system is sometimes called a shader core.



arm

Scaling Out

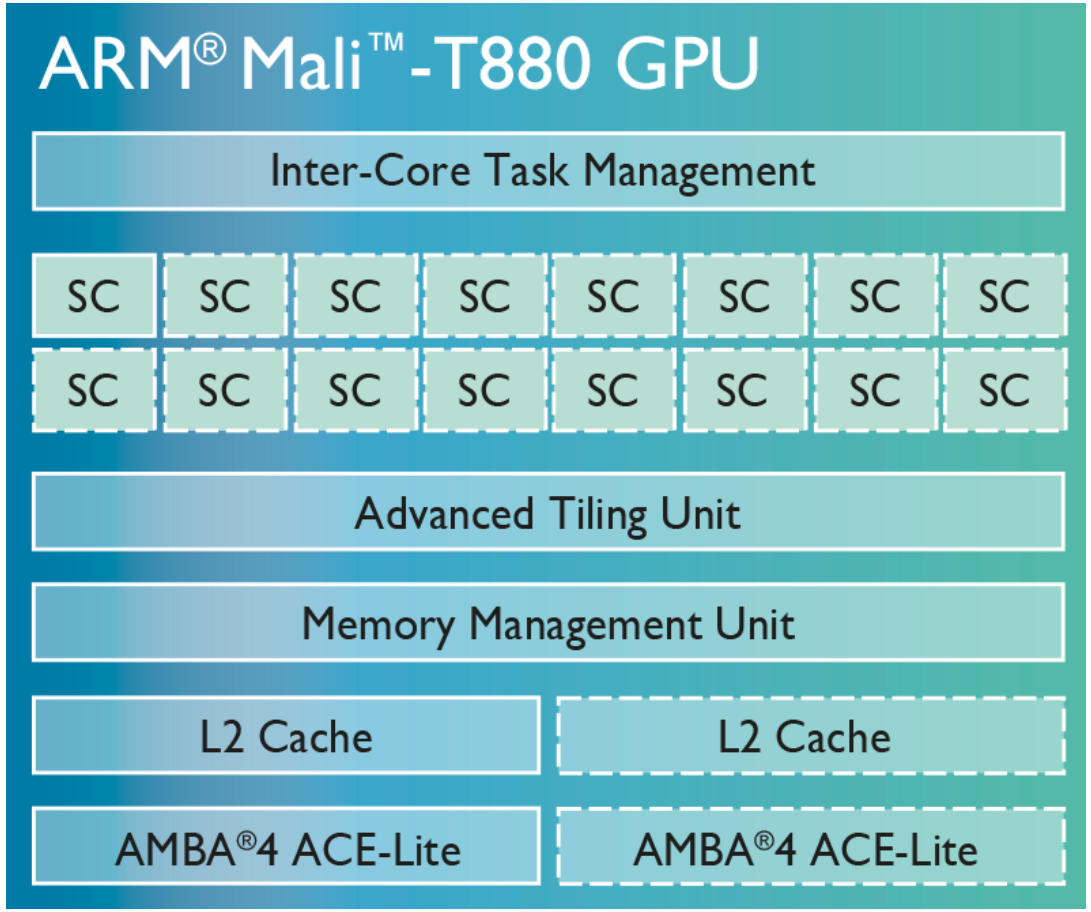
- Multiple instances of each shader core provided together
  - Each independent of the others
  - Each processes a subset of the work groups
- Massively increases parallelism
- Memory hierarchy provided, too
  - Shared L2 cache reduces memory bandwidth requirements.
  - Smaller caches local to each multithreaded core



arm

Case Study: Mali T880 GPU

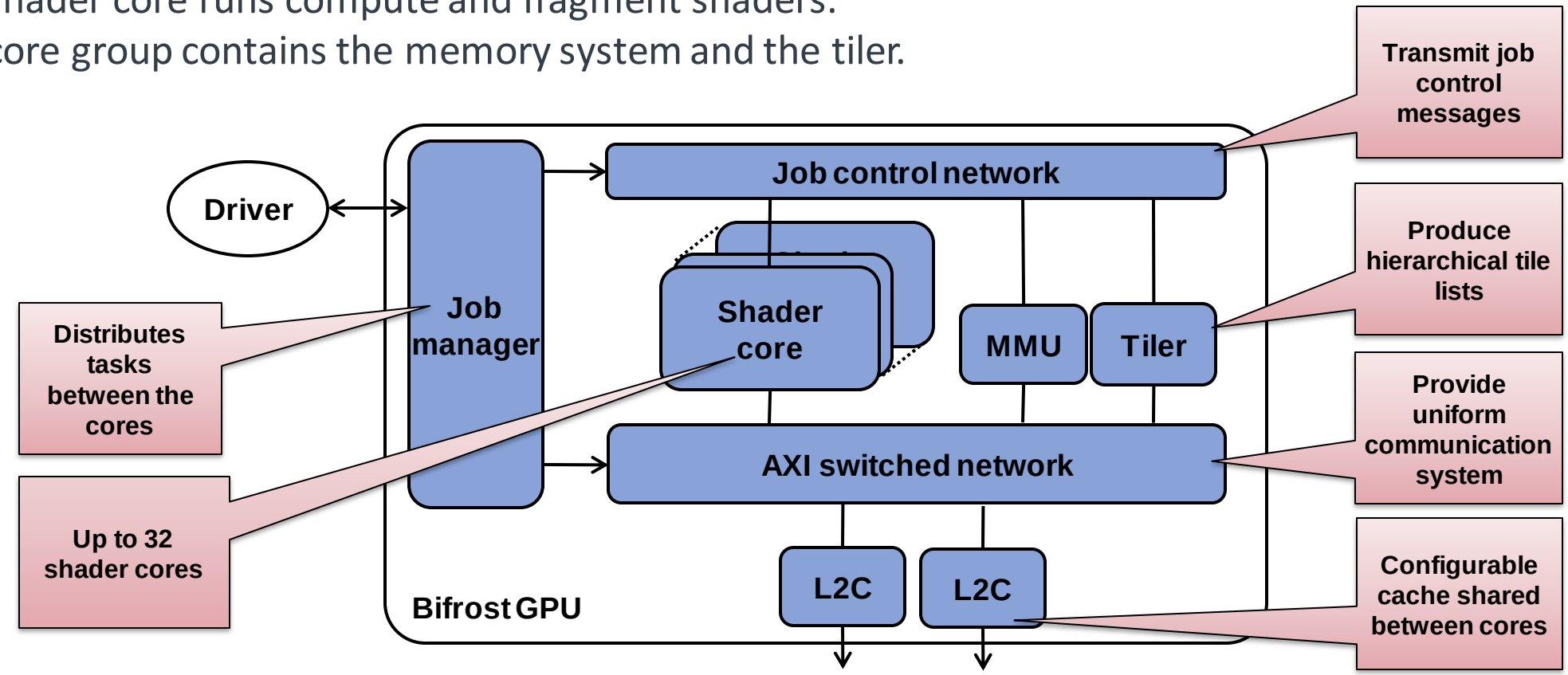
- Up to 16 shader cores (SC)
  - Each core supports multiple threads and operations.
- Performance
  - 30.6G FLOPS at 900 MHz
- API support
  - OpenGL ES 1.1, 1.2, 2.0, 3.0, 3.1
    OpenCL 1.1, 1.2
    DirectX 11 FL11_2
    RenderScript
- Usage in SoCs
  - Exynos 8890, Helio X20 (MT6797), Kirin 950
- Used for both graphics processing and high-performance computing



arm

Case Study: Arm Mali Bifrost GPU Architecture

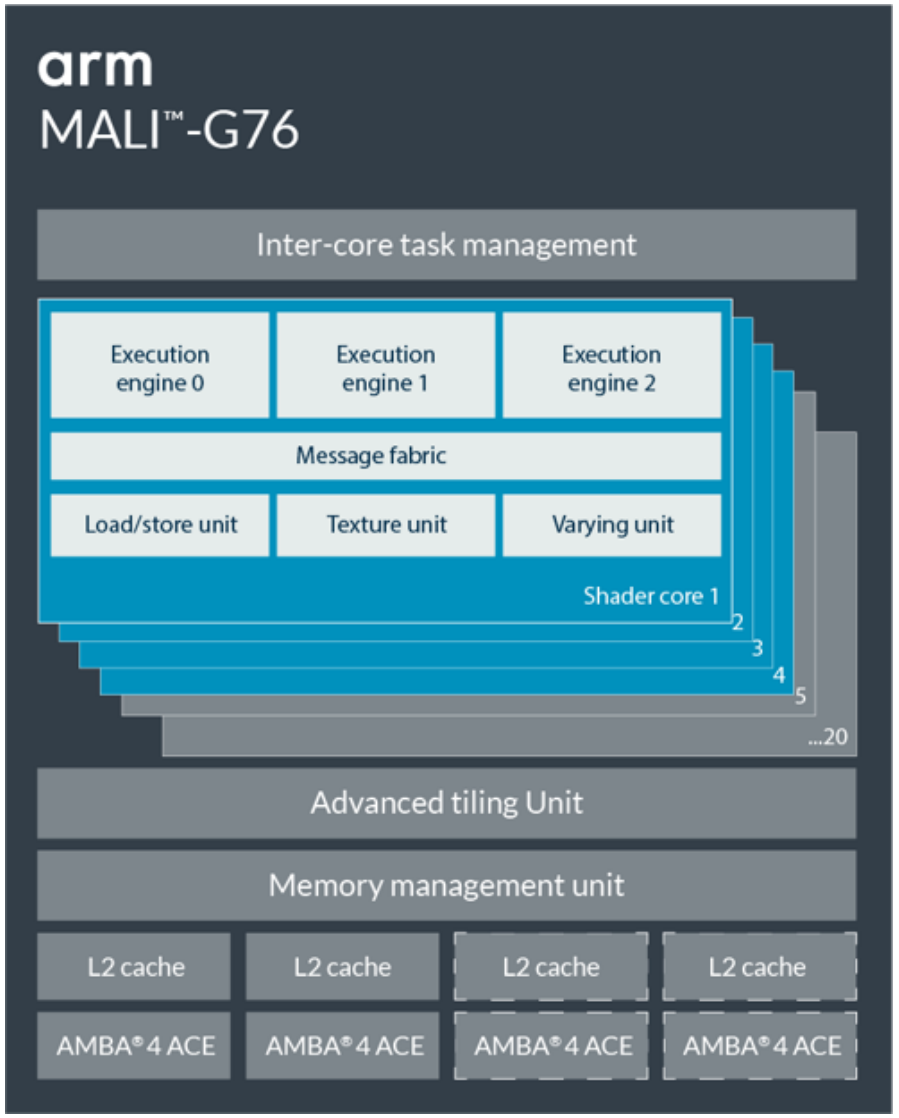
- Mali Bifrost GPU consists of 3 main blocks or groups.
  - The job manager interacts with the driver and controls the GPU HW.
  - The shader core runs compute and fragment shaders.
  - The core group contains the memory system and the tiler.



arm

Case Study: Arm Mali-G76 GPU

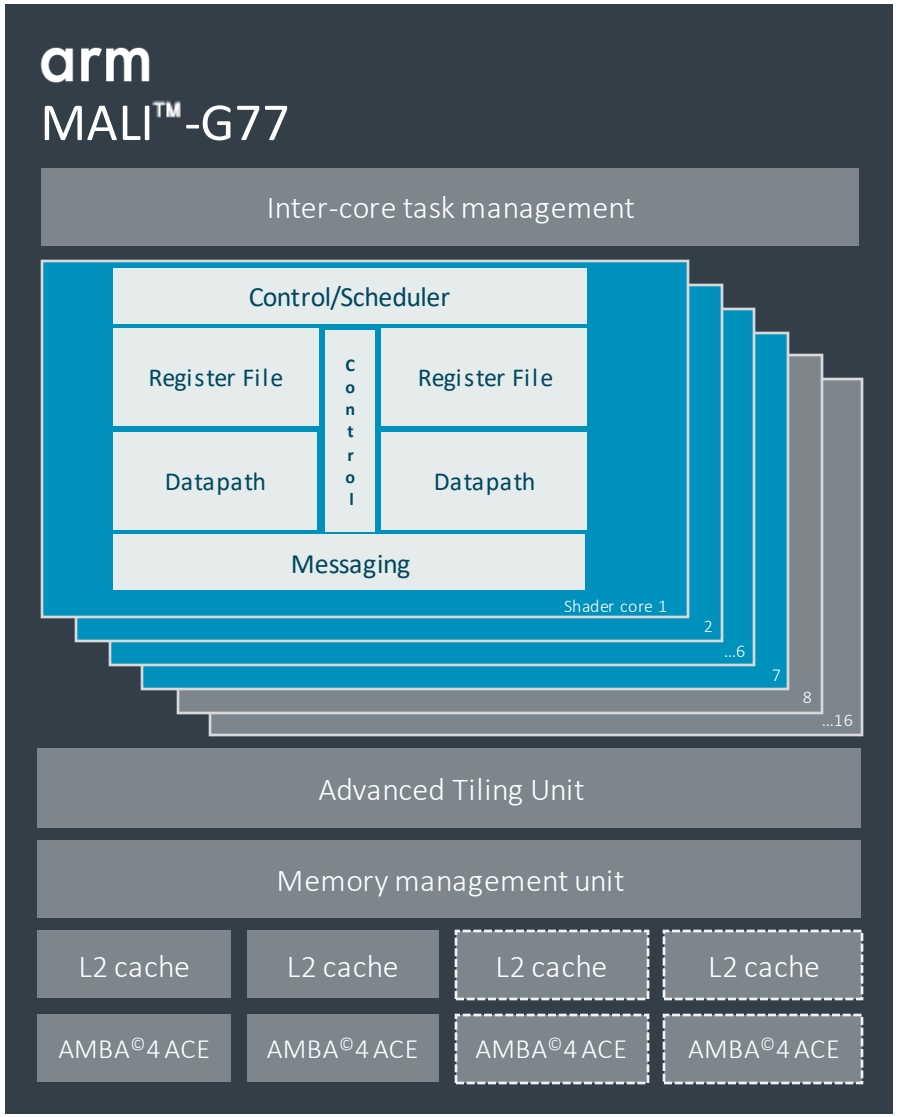
- Third generation of the Bifrost architecture
- Maximum 20 shader cores (SC)
  - Wider execution engines with double the number of lanes
- Performance
  - Complex graphics and machine-learning workloads
- API support
  - OpenGL ES 1.1, 2.0, 3.1, 3.2
    OpenCL 1.1, 1.2, 2.0 Full profile
    Vulkan 1.1
- Shared L2 cache with 2 or 4 slices



arm

Case Study: Arm Mali-G77 GPU

- First generation of the Valhall architecture
  - Warp-based execution model
  - New instruction set with operational-equivalence to Bifrost
  - Dynamic instruction scheduling in hardware
- Configurable 7 to 16 shader cores
- Single execution engine per shader core
- Configurable 2 to 4 slices of L2 cache
  - 512 KB to 4 MB in total
- Texture mapper – 4 texture element (texel) per cycle
- Supports Vulkan and OpenCL



Summary

- General-purpose CPUs optimize scalar single-threaded performance
  - But there are many domains where data-level parallelism is common.
  - Other architectures can exploit this efficiently.
- GPUs exploit a different form, SIMT, with massive parallelism.
  - Simple cores but multithreading and multiprocessing hide latency