**MiniGPU**  
SoC notes

-The exploitation of the GPUs vast floating-point throughput as a means of speeding up certain elements of our software

-multi- and many-core computation will be a significant area of games-related research for years to come

-we focus on CUDA as it is the most straightforward API through which to implement GPU computation without completely abstracting the GPU hardware

-The principles discussed in this lecture series, however, map to all contemporary GPU computation APIs, as the issues faced in deploying code to the GPU do not change with vendor.

Kepler CUDA hardware architecture

Maxwell-architecture

These units share the L2 Cache and, through that, access to the VRAM (analogous to system memory when programming for the GPU).

An SMX features 192 single-precision cores and 64 double-precision, along with 32 special function units (SFUs units optimised for common mathematical functions). You will note also the memory architecture. 48KB of Read-Only Data Cache, and 64KB of memory labelled ”Shared Memory/L1 Cache”.

This 64KB is a pool of memory that you can, through the CUDA API, control to favour one or the other (L1 Cache, or Shared Memory) 16KB L1 and 48KB Shared; 16KB Shared and 48KB L1; or, 32KB of each. Shared Memory is a store for variables that can be accessed and updated by any core in the SMX, at any time. The L1 Cache pool is a shared cache pool which is used by every core in an SMX.

CUDA is a C-styled language that permits the deployment of programs on the GPU. CUDAs syntax is relatively straightforward