Configuring Block/Grid Dimensions

Goal: Maximize throughput *and* utilization of SMs on GPU

CUDA terms “grid”, “block”, “warp”, “thread”, and “SM” are used here; the OpenCL equivalents are “NDRange”, “work group”, “wavefront”, “work item”, and “CU”, respectively.

Dimensionality of grid and block

- Decide based on the application whether these should be 1D, 2D, or 3D.
- Remember that the number of threads in a block is the product of the block size in each dimension.

General Strategies

- Do adequate work per thread to amortize overhead.
- The grid should be sufficiently large to get multiple blocks per SM.

  If there *are* enough blocks, whether a SM can actually have more than one block resident and executing at a time depends on its resource requirements including (i) shared memory per block, (ii) registers per block, and (iii) threads per block.

  If the number of blocks is fewer than number of SMs, the GPU may be able to execute another kernel at the same time on a different set of SMs if it is on another stream (CUDA) or queue (OpenCL).
- The number of threads in a block must be a multiple of the warp size (typically 32 on today’s GPUs). Further, it should be at least (*k* * warp size) where *k* is the number of warp schedulers on each SM.

Numerical Targets

- Occupancy of a SM (see book, pp. 416-417)

  occupancy = numberWarpsOnSM / maxWarpsPerSM *(target: occupancy → 1)*

  Denominator is just maxThreadsPerSM/warpSize.

  Numerator is influenced by resource requirements such as those mentioned above.

Caveats

- Some parts of the formula given in the book (reproduced below) and its implementation assume some data whose values may be difficult to obtain.
- There are CUDA – OpenCL differences in terms of how and what data can be queried as well. (We will see a sample of these differences shortly.)
- Hence these formulas and code examples represent *rough initial guidelines*.

One Implementation of the numerical targets for Block size

$$threadsPerBlock = \min\begin{pmatrix} numWarpSchedulers * warpSize \\ regsPerBlock \big/ regsPerThread \\ sharedMem \big/ sharedMemPerThread \\ maxThreadsPerSM \end{pmatrix}$$

- Notes:
    - “*regsPerBlock*” is maximum allowed registers per block – a GPU-specific limit.
    - “*sharedMem*” is maximum shared memory allowed by the GPU per block.
- Then round *threadsPerBlock* down, if necessary, to multiple of warp size
- Then compute total number of blocks (or global work size) using `ceil(`*totalNumThreads*/*threadsPerBlock*`)`.
- See/run Barlas’ `executionConfHeur.cu`.
    - ➔ This code will fail if the supplied parameters are such that the loop condition is initially false.
    - ➔ A reminder that this is just one idea of a starting point for this type of runtime kernel configuration!