Composable GPU programming

GPUs -- what are they?

- Basic model: SIMD, SPMD, MIMD;
  - blocks of PUs with single PC, local memory (synchronous); warps
  - many blocks (asynchronous), VRAM
- discontinuities/constraints from hardware implementation of memory access;
  - next-generation hardware likely to mediate this to make programmability more orthogonal

GPUs -- what are they?

Revenge of the PRAM?

- Basic model: SIMD, SPMD, MIMD;
  - blocks of PUs with single PC, local memory (synchronous); warps
  - many blocks (asynchronous), VRAM
- discontinuities/constraints from hardware implementation of memory access;
  - next-generation hardware likely to mediate this to make programmability more orthogonal

Programming GPUs

- CUDA: C-like language for general-purpose programming with code generated for GPUs
    - previously: OpenGL for graphics programming
    - coming up: OpenCL (compute language)
- foo<<m, n, k>> (args)
    - Execute foo with implicit argument i, j (block, PU) selecting from arguments
    - Care required when accessing memory: out of sequence accesses sequentialized!

GPU language projects

- Data parallel Haskell:
  - Programming flat PRAM level
  - Nested/compositional programming
    - map (map f) (xss)
- Obsidian: Combinator language for generating CUDA code
  - explicit synchronization
  - choosing threads, mapping to blocks

How to exploit?

- Performance: If you have a data parallel problem, formulate it using scan, map, fold, permute on bulk data (arrays), have it shipped out to a GPU!
- If you can't figure out how to do that, do not expect magic from your compiler.

Qualities

- Obsidian good candidate for capturing two-level model (synchronous blocks and asynchronous sets of blocks) and implementing APRAM model
  - Excellent scan implementations
- Data parallel Haskell good model for programming APRAM model and for compositional abstraction on top of that
  - NESL with h.o. functions, polymorphism

Requirements

- Need a robust performance model: NESL at PRAM level, sth else lower;
- Need to stay in the same programming model when engineering/tuning code
- Need a robust programming model (sw/hw) -- small changes shouldn't lead to unpredicatable radical changes in performance.

(End)