



Technology for a better society

Johan Seland

GPU Algorithm Design

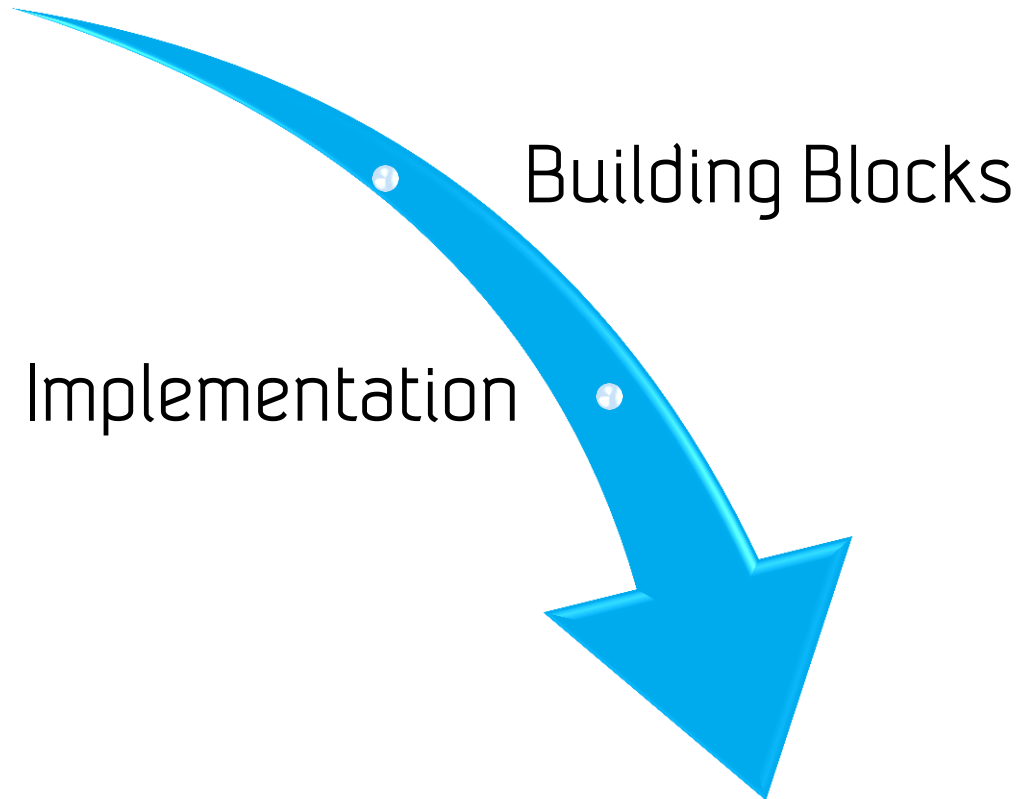
USIT Course Week

16th November 2011

Programmers waste enormous amounts of time thinking about, or worrying about, the speed of noncritical parts of their programs, and these attempts at efficiency actually have a strong negative impact when debugging and maintenance are considered. We *should* forget about small efficiencies, say about 97% of the time: **premature optimization is the root of all evil**. Yet we should not pass up our opportunities in that critical 3%.

Donald Knuth

Prerequisites



Benchmark and
Validation

Prerequisites



Decision Framework for Optimization

- Establish a clear goal for optimization
 - Performance
 - Power
 - Scalability
 - Accuracy
- Who is the "customer"?
 - What is their HW/SW stack?
 - Fast double precision only on compute GPUs
 - What is their skill-level
 - Will they need to maintain GPU code?

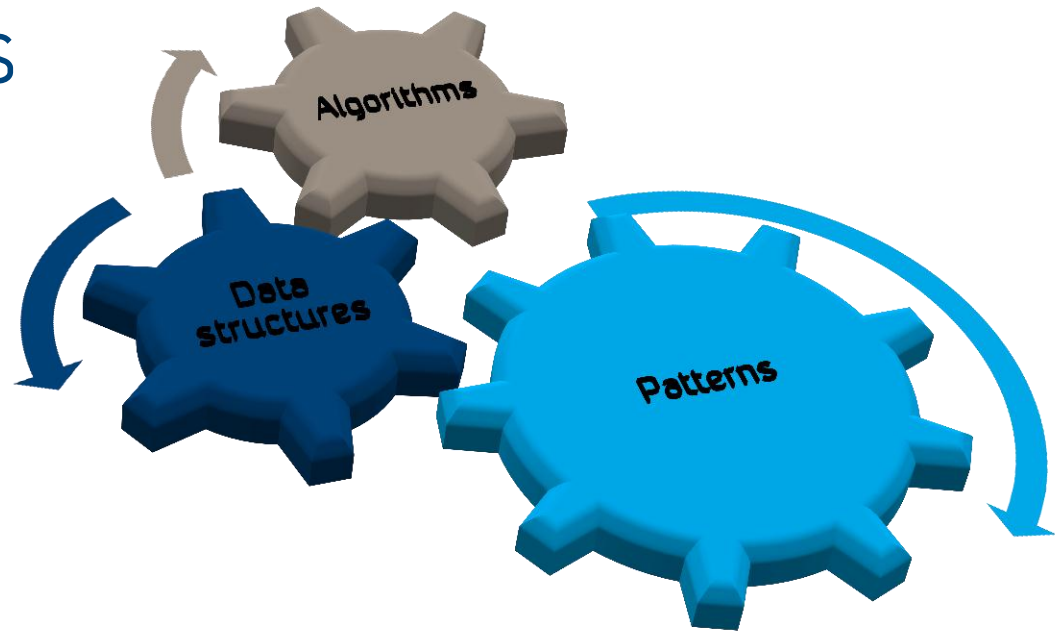
Validation and Benchmarking

- Establish validation suite
 - Bit-exact "gold" standard does not work
 - TDD makes a lot of sense
- Establish benchmark suite
 - Measure effective bandwidth
 - Measure scalability
- Benchmarking **MUST** include validation
 - Commit to VC after every run?
- Prototype CPU version of GPU algorithm?

Parallel Floating-Point

- Floating point **IS NOT** associative
 - $(a * b) \neq (b * a)$
 - Parallel algorithms will not be bit-equal to serial version
 - Runtime scheduling is not bit-repeatable
- x87 FP uses 80-bit internally
- GPU fast-path is not fully IEEE compliant
- GPU will automatically use Fused Multiply and Add (FMAD)

Building Blocks



The Thirteen Dwarves

- **Computational Dwarf**: An algorithmic method that captures a pattern of
 - **Computation**
 - **Communication**
- Inspired by Phil Colella
 - Seven numerical methods for science and engineering
- Common computational patterns of current and future interest
- There is also other parallel pattern catalogues



Dwarves 1-7

Dwarf	Description	Limitation
Dense Matrix	Data are dense matrices or vectors. Uses unit stride lookups.	Compute limited
Sparse Matrix	Many zero values. Datasets stored in compressed formats.	50% compute, 50% memory bandwidth
Spectral (FFT)	Data in the frequency domain. Typically several butterfly stages	Memory latency limited
N-Body	Interactions between many discrete points.	Compute limited
Structured Grid	Regular grids. Points and grids are updated together.	Memory bandwidth limited
Unstructured Grid	Irregular grids, typically involves several levels of memory reference	Memory latency limited
MapReduce/Monte Carlo	Calculations depend on repeated random trials. Embarrassingly parallel.	Problem dependent

Dwarves 8-13

Dwarf	Description	Limitation
Combinatorial Logic	Implemented with logical functions and stored state.	CRC Problems bandwidth; Crypto compute
Graph Traversal	Visits many nodes in a graph by following successive edges	Memory latency
Dynamic Programming	Computes a solution by solving simpler overlapping sub-problems	Memory latency
Backtrack and Branch+Bound	Finds an optimal solution by dividing region into subdomains and prune.	?
Construct Graphical Models	Constructs graphs that represent variables as nodes and edges.	?
Finite State Machine	A system defined by states. Transitions defined by inputs	Nothing helps!

How to use Dwarfs

- Your problem likely maps into one or more dwarfs
- Identify your problem:
 - Will you be compute or bandwidth bound?
 - Research literature
- Reference implementations exists for most cases

Common GPU algorithmic building blocks

- Parallel reductions
 - Sum, min, max etc.
 - Typically $\log n$ -passes
- Scan – Prefix-sums - Histogram-Pyramids
 - For building (sparse) datastructures
- Sorting
 - Radix-Sort
 - Bitonic sort
- Datastructures
 - Skip-lists
 - Nested, sparse structures

Libraries

- Does existing libraries provide a (partial) solution?
 - Is license acceptable?
- "Drop-in"-libraries can synchronize too much

Some available libraries

CUFFT	Fast Fourier Transform
CUBLAS	Dense Linear Algebra
CULA	LAPACK interface
CUSPARSE	Sparse Linear Algebra
CUSP	Linear Algebra Graph Computations
CURAND	Random Number Generation
NPP – Nvidia Perf. Primitives	Image and Signal Processing
CUDA Video Decoder/Encoder	H.264/MPEG-2 video coding
THRUST	STL like algorithms

These libraries have various licenses

Implementation



What GPUs are

- GPUs are parallel co-processors, not accelerators
- 10 threads don't matter; 10 000 threads do
 - Starting threads is very lightweight
- Divide & conquer!
- Some irregularity is ok if common case is regular

What GPUs are NOT

- NOT: Flat Multiprocessor
 - Global synchronization is not cheap
 - Atomics, mutexes etc.
 - Global memory access times are expensive
 - I.e: not an SMP
- NOT: Distributed Processors
 - Distributed computing is a different setting
 - I.e: not a small MPI cluster

Effective algorithm design

- Expose fine-grained parallelism
 - Needs 1000s of threads for full utilizations
- Maximize on-chip work
 - On-chip memory is orders of magnitude faster
 - PCIe transfer expensive
- Minimize "local" execution divergence
 - SIMT execution of threads in 32-threads warps/wavefronts
- Minimize memory divergence
 - Coalesced (vector) load/store across warp

Two regimes of parallel tasks

- Many independent fine-grained tasks
 - Assign one task to each thread
 - Coordination mostly at kernel boundaries
 - Little use of shared memory
- Collection of coordinated parallel tasks
 - Assign one task to each thread block
 - Heavy use of shared memory
 - Common case in divide and conquer algorithms

Parameterize Algorithm

- Facilitate benchmarking over:
 - Number of threads
 - Number of thread blocks
 - Various sizes of datasets
 - Different GPU platforms
- "Policy"-based design helps
 - Separate communication and computation

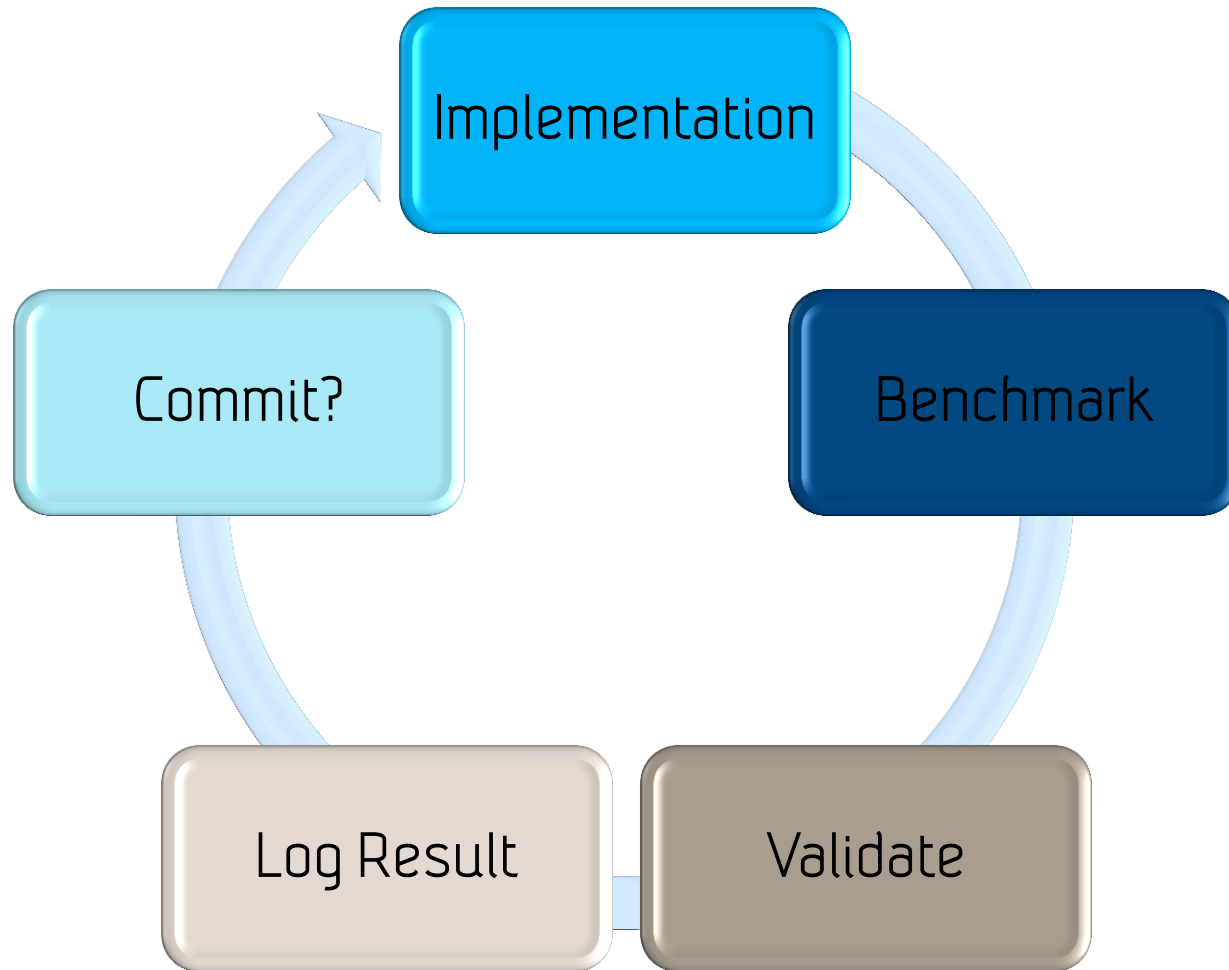
Benchmarking and Validation



Best Practices

- Run repeatable benchmark on realistic datasets
 - Easy to optimize for a given dataset
- **VALIDATE results automatically**
- Benchmark memory access separately
- Use existing tools

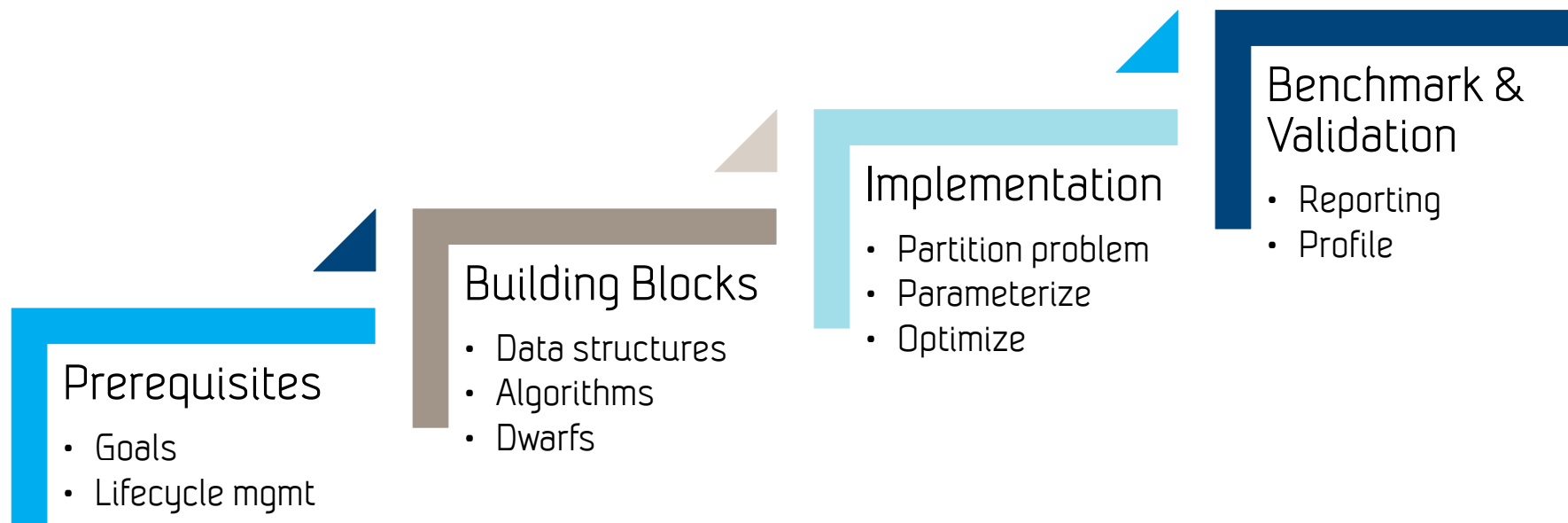
TDD-Like Cycle



GPU Speedup reporting

- Speedup compared to unoptimized CPU implementation is irrelevant
 - Superlinear speedups are very rare
- Measure resource utilization
 - Observed bandwidth
 - Observed FLOPS
 - Observed efficiency
- Domain specific measures
 - Millions of cells per second etc.

Conclusion



Further reference

- [The Landscape of Parallel Computing Research: A View from Berkeley](#)
- [OpenCL and the 13 Dwarfs](#)
- Patterns for Parallel Programming
- [GPU Algorithm Design Video Series](#)

