Advanced Multicore Programming

Dr Andrew Moss

[1]BTH Karlskrona

September 26, 2011

Performance Models

Last lecture covered the practical approach (what we can measure)

- Now we look at what *should* happen
- You have the tools now to decide if it does

Why would you need to decide for yourself?

- Performance of a GPU is a black-box (creator won't tell you)
- Performance of a GPU is complex (creator can't tell you)

Henry Wong, Misel-Myrto Papadopoulou, Maryam Sadooghi-Alvandi, and Andreas Moshovos
Demystifying GPU Microarchitecture through Microbenchmarking
ISPASS 2010
http://www.stuffedcow.net/research/cudabmk

Simplified model

Sources

nVIDIA Corporation.
nVIDIA OpenCL Best Practices Guide.
http://developer.download.nvidia.com/compute/cuda/3_2_prod/toolkit/docs/OpenCL_Best_Practices_Guide.pdf

David B. Kirk, Wen-mei W. Hwu
Programming Massively Parallel Processors.
978-0-12-381472-2

Check bandwidth first

Peak theoretical bandwidth

- Find this from the card specification (i.e manually)
- $(1107\text{Mhz} \cdot 10^6 \cdot \frac{512}{8} \cdot 2)/10^9 = 141.6$ GB/sec (GTX-280)
- 1.107 Ghz
- 512-bit bus
- 2 is because the bus is DDR

Effective bandwidth

- $\frac{B_r + B_w}{t}$ where B_r/B_w is Bytes Read / Written, t is time

Provides a limit on performance, avoid wasting time in optimisation

- If the limit is too low. . . transfer fewer bytes

Workgroups

Kernels are executed across grids (ranges of IDs)

- Each ID in the range is associated with a single thread
- There is a unit of organisation between threads and grids

Workgroups are collections of associated threads

- The idea is that you organise threads with similar memory accesses into a workgroup.
- Actual units of despatch inside the GPU

For example the GTX-280 uses SMs with 8 scalar processors per SM

- Pipelining takes two clock cycles per instruction despatch
- Basic unit of despatch is 16 threads
- nVIDIA refer to this as a *half-warp*
- Some other instruction scheduling and memory access issues mean that 32 threads is smallest general scheduling unit (*warp*)

Warps

Each thread inside a warp executes in lock-step on an SM

- When we say that you need multiple threads to hide latency...
- We really mean that you need multiple warps of threads
- Similar unit on AMD hardware is called a *wavefront*, but uses 64 threads

Every instruction has some latency associated with it

- Time to calculate arithmetic
- Time to store results into the register bank
- nVIDIA refer to both of these as “arithmetic latency” (§4.3 in Best Practices)
- Requires at least 6 warps per SM to hide

Compute Density

CGMA : Compute to Global Memory Access ratio
Measure (global) memory access

- How many bytes are read/write per kernel execution

Measure how much arithmetic inside the kernel

- CUDA toolkit provides some tools to help
- OpenCL doesn't
- Either way you will need to estimate how FLOPs

CGMA = FLOPs : bytes transfered

- Number of bytes, not the number of floats
- GTX-280 has max (theoretical) peak of 367 GFLOP/s
- GTX-280 has max (global) bandwidth of 86.4 GB/s
- Thus max CGMA is about 4:1

Compute Density II

If your kernel has a CGMA less the peak for the device

- Your kernel will be memory-bound, rather than compute-bound
- To increase performance, do more work on less data

This requires a set of techniques for restructuring kernels

- These are not general-purpose (across all kernel functions)
- Specific types of problems will allow specific transformations

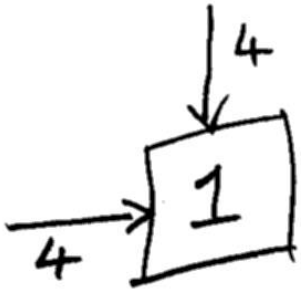
One common type of problem

- Computed over a 2D grid
- Each cell in the grid is independent of other results
- But, intermediate values are reused
- (Read only) Sharing, but no dependencies (Read/Write)
- Obvious example: matrix multiplication
- Restructuring technique is *tiling*

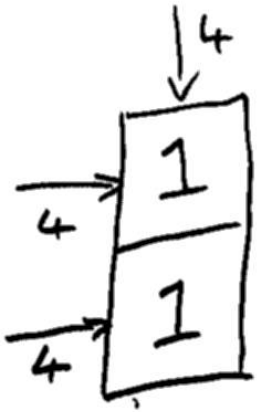
Tiling

Each cell has some number of inputs and some amount of computation

- One instance of the kernel can write to multiple outputs
- Can combine several computations
- Amortisize the memory costs

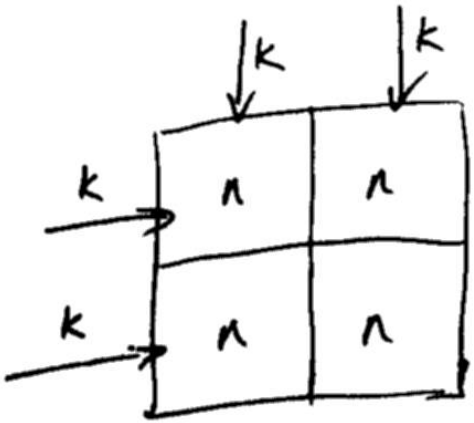


CGMA = 1:8 ($\frac{1}{32}$ of peak)



CGMA = 1:6 ($\frac{1}{24}$ of peak)

Tiling II



CGMA = whn:$(w + h)k$

- n is FLOPs to calculate output
- k is measured in bytes
- $w \times h$ is the tile size

Tiling III

Attempt to maximise CGMA ratio ($whn : (w + h)k$)

- As k and n are fixed, implies maximise tile size $w \times h$
- But larger tiles impiles fewer threads
- Less parallelism and less latency hiding

Need to maintain enough parallelism to achieve performance

- Tiling working by merging computations into a thread
- Opposite approach is splitting threads into multiple computations
- One general target is chains of associative operators

Splitting

For example, an inner kernel may be

$$\sum_{i}^{n} x_i \cdot y_i$$

Addition is both associative and commutative, so we can rearrange

$$\sum_{i}^{\frac{n}{2}} x_i \cdot y_i + \sum_{i=\frac{n}{2}}^{n} x_i \cdot y_i$$

Can split into anything up to n pieces

- Add pieces together at the end...

Optimisation trade-off

Splitting increases the number of threads at work

- In exchange for a final merging stage

Tiling increases the locality of work in a thread

- In exchange for fewer threads

If they are combined together

- Should see a trade-off between the two
- Large set of parameters (tile width, tile height, number of splits. . .)
- Looking for an optimal combination

Approach

When the model is a good description of the behaviour

- Work analytically (pushing around symbols and manipulating equations)

When the model is a poor description of the behaviour

- Resort to empirical measurement

When we have no model

- Build one

Most real optimisation problems require mixing these three steps...

Summary

Lots of constants

- Performance is highly dependent on tuning to current architectures
- No good way to automatically find those constants

“Simple” scheduling model is too complex

- If your “rule of thumb” is a spreadsheet. . .
- Too imprecise — abstracts memory access completely

“Real” scheduling behaviour is too complex

- Limited options; profiling and tuning
- Need to balance local processing, local storage, memory bandwidth. . .

When it works you get 1000x the performance of a general purpose processor