Utrecht University

B3CC: Concurrency

16: Conclusion

Ivo Gabe de Wolff

Final exam!

- Final exam
  - Tuesday 30 January @ 13:30
  - Olympos Hal 2
  - Mix of multiple choice and open questions
  - Covers material from second half of the course:
    - From lecture 9 (Parallelism) through lecture 15 (Work & Span)
  - You don't need to write Accelerate code
    - But you may be asked to design a parallel algorithm in terms of the parallel patterns
  - Remindo has a calculator, no physical calculators allowed

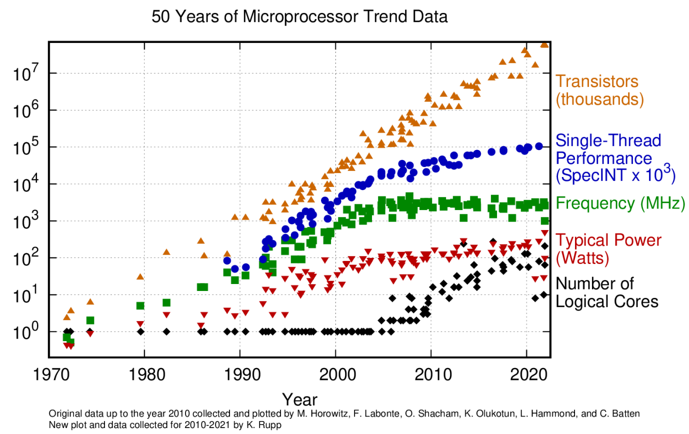
https://www.instagram.com/p/CZIPjkijxj1

Brief course summary

What?

Parallelism & Concurrency

Why?



Where?

- Three kinds of code:
  - Gameplay simulation
    - Models the state of the game world as interacting entities
  - Numeric computation
    - Physics, collision detection, path finding, scene graph traversal, etc.
  - Shading
    - Pixel & vertex attributes; runs on the GPU

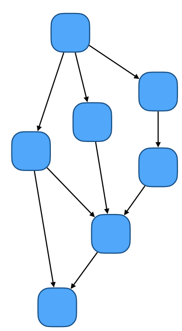
Sekiro: Shadows Die Twice, FromSoftware

How?

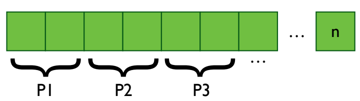
	Game Simulation	Numeric Computation	Shading
Languages	C++, scripting	C++	GC, HLSL
CPU Budget	10%	90%	n/a
Lines of Code	250.000	250.000	10.000
FPU Usage	0.5 GFLOPS	5 GFLOPS	500 GFLOPS
Concurrency/ Parallelism	**STM**	**SIMD**	**GPU**

Tim Sweeney: *The Next Mainstream Programming Language*, POPL 2006

Kinds of parallelism



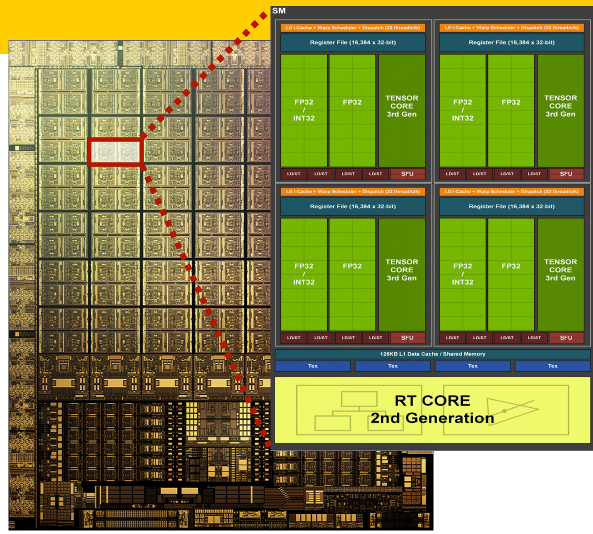
Task parallelism



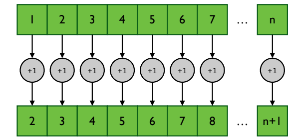
Data parallelism

GPGPU

- How the parallel patterns we have talked about map to GPU code
- Difference between CPU and GPU
  - What each is designed for; strengths and weaknesses
  - What the GPU programming model (CUDA) is designed for

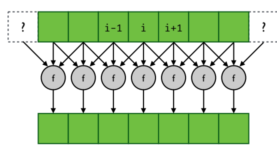


Patterns: map



- Apply a function to every element of an array, independently
- This one is (hopefully) straightforward…

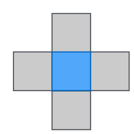
Patterns: stencil



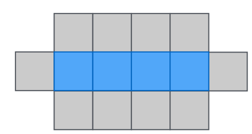
- A map with access to the surrounding neighbourhood
- What are the difficulties/limitations?
  - The ghost region (halo) and how/why to use it
  - Optimisations (tiling, strip mining, etc.)

Stencil optimisations

- Use a different kernel for the interior and border regions
  - In the gaussian blur example of a 512x512 pixel image, 98% of the pixels do not require in-bounds checks
- Optimise data locality & reuse through tiling
  - *Strip mining* is an optimisation that groups elements in a way that avoids redundant memory access and aligns accesses with cache lines



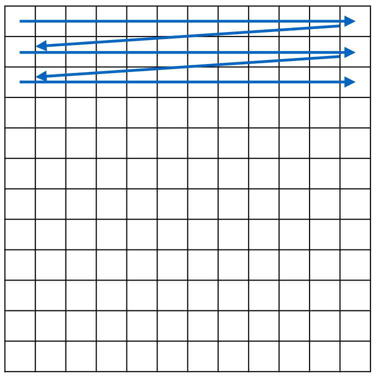
4 x (5 reads + 1 write)



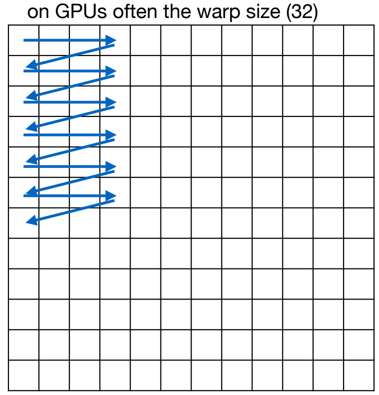
14 reads + 4 writes

Stencil optimisations

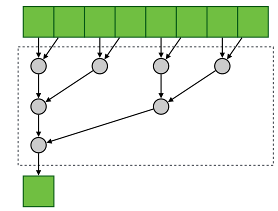
Without tiling
- When handling row 0, row 1 is loaded in cache.
- First values of row 1 may already be out of cache, when handling row 1



With tiling
- Previously loaded row is still in cache
- Tile width is usually a power of 2, on GPUs often the warp size (32)

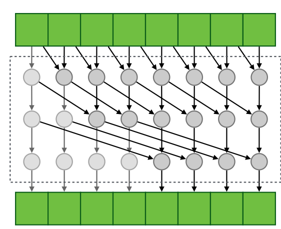


Patterns: fold



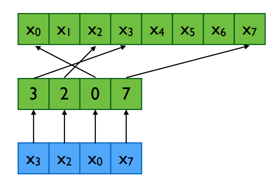
- Reduce an array to a summary value
- How to implement this in parallel
  - What kinds of restrictions are necessary?
  - What additional restrictions can be leveraged to improve it further?

Patterns: scan



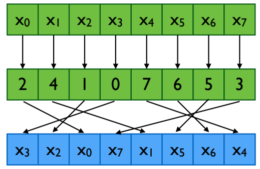
- All partial reductions of an array
- Varieties
  - Inclusive vs. exclusive etc.
- Parallel implementation
  - Restrictions, etc.

Patterns: gather



- Parallel random read
- Implications for memory access patterns
  - Optimisations for special cases (e.g. transpose), like tiling
  - Implications for the GPU, caches, etc.

Patterns: scatter



- Parallel random write
- How to handle collisions in the index permutation function
  - Performance implications of collisions, false sharing, etc.
  - Scatter vs. gather

https://en.wikipedia.org/wiki/False_sharing

Patterns: gather vs scatter

- Random reads (gather) are slower than structured reads
- Random writes (scatter) are slower than structured writes
- This problem is larger for scatter,
  as the processor needs to perform more synchronization between cores
- In general, use gather instead of scatter if both are possible

Patterns

- You should be able to:
  - Give examples for each pattern
  - Recognise these patterns and where they can be used
    - e.g. given a problem description, give an implementation in terms of these patterns
    - Use Accelerate code, pseudocode or an explanation in text
      - Especially for the latter, make sure your explanation is concrete

Work & Span

- We analysed the performance of algorithms using the *work* and *span*:
  - **Work =** T_1
    How long to execute on a single processor
  - **Span =** T_∞
    How long to execute on an infinite number of processors
    - The longest dependence chain / critical path length / computational depth
    - Example: $O(\log n)$ for summing an array

Efficient & optimal

- The parallelisation *overhead* of an algorithm is its work divided by the cost of the best sequential algorithm
  - For this parallel scan we have to put $O(n \log n)$ work into something which can be done sequentially in linear $O(n)$ time: the overhead is logarithmic
  - A parallel algorithm is:
    - *Efficient* when the span is poly-logarithmic and the overhead is also poly-logarithmic
    - *Optimal* when the span is poly-logarithmic and the overhead is constant

Master Theorem

- The master theorem provides a solution to recurrence relations of the form
  - For constants $a \geq 1$ and $b > 1$ and f asymptotically positive

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

- The master theorem has three cases:

Recursion dominates	Both contribute	*f* dominates
If $f(n) = O\left(n^{\log_b a - \epsilon}\right)$ for some $\epsilon > 0$, then $T(n) = \Theta\left(n^{\log_b a}\right)$	If $f(n) = \Theta\left(n^{\log_b a}\right)$, then $T(n) = \Theta\left(n^{\log_b a} \log n\right)$	If $f(n) = \Omega\left(n^{\log_b a + \epsilon}\right)$ for some $\epsilon > 0$, and $af\left(n/b\right) \leq cf\left(n\right)$ for some $c < 1$ for all n sufficiently large, then $T(n) = \Theta\left(f\left(n\right)\right)$

https://en.wikipedia.org/wiki/Master_theorem_(analysis_of_algorithms)

Analysis of parallel algorithms

- You should be able to:
  - Compute the work and span given a problem description/code
  - Compare parallel algorithms
    - Efficient & optimal
    - Parallel speedup (Amdhal vs. Gustafson-Baris)

Questions?

Finally...

- Please fill out the Thermometer survey!
  - All constructive feedback is welcome
  - https://caracal.uu.nl/35916/Respond

Photo by Claudio Piccolo