Parallel Computing with GPUs

Performance
Part 3 – Occupancy

The University Of Sheffield.

Dr Paul Richmond

http://paulrichmond.shef.ac.uk/teaching/COM4521/

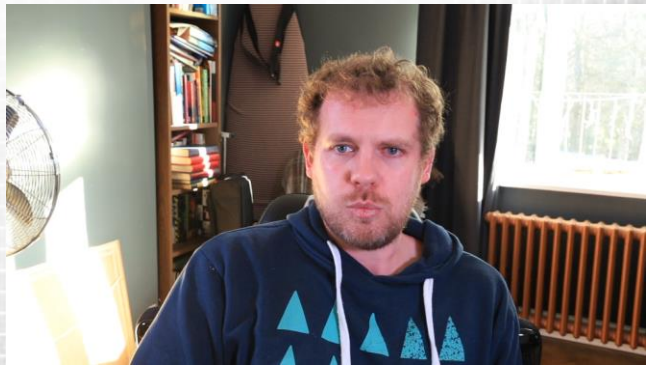
This Lecture (learning objectives)

- Occupancy
  - Define occupancy as a metric of resource utilisation
  - Explain the implications of occupancy on performance
  - Demonstrate methods to maximise occupancy
  - Examine the impact of thread block size on occupancy and performance

Occupancy

- Occupancy is the ratio of active warps on an SMP to the maximum number of active warps supported by the SMP
  - Occupancy = Active Warps / Maximum Active Warps

- Why does it vary?
  - Resources are allocated at thread block level and resources are finite
  - Multiple thread blocks can be assigned to a single Streaming Multi Processor
  - Your occupancy might be limited by either
    1. Number of registers
    2. Shared memory usage
    3. Limitations on physical block size



Why is occupancy important

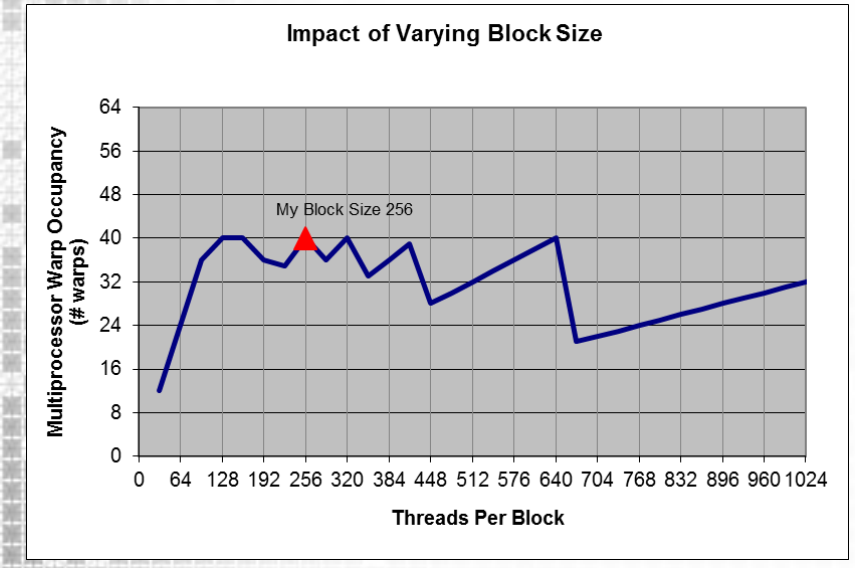
- Implications of Increasing Occupancy
  - Memory bound code
    - Higher occupancy will hide memory latency
    - If bandwidth is less than peak then increasing active warps might improve this
  - Compute bound code
    - Will not improve performance
  - 100% occupancy not required for maximum performance
    - Instruction throughput might be optimal
    - Memory bandwidth might be fully saturated

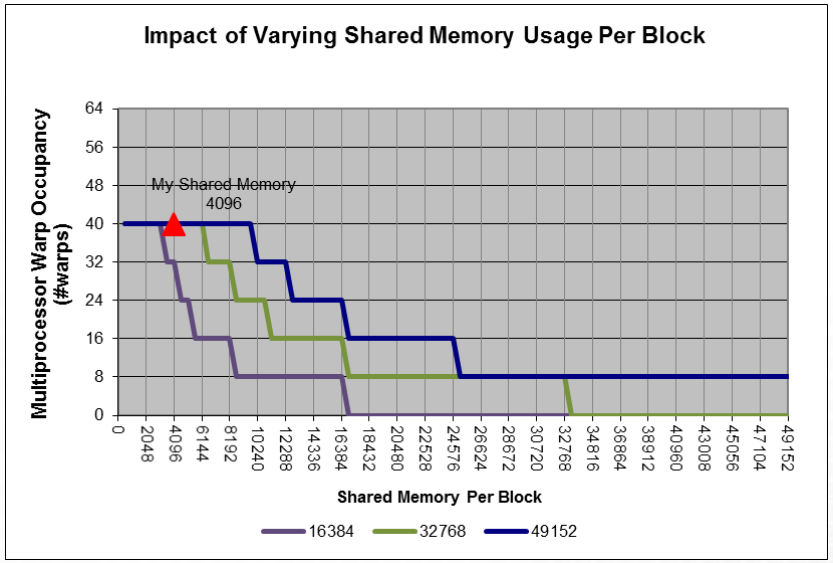
Occupancy and Thread Block Size

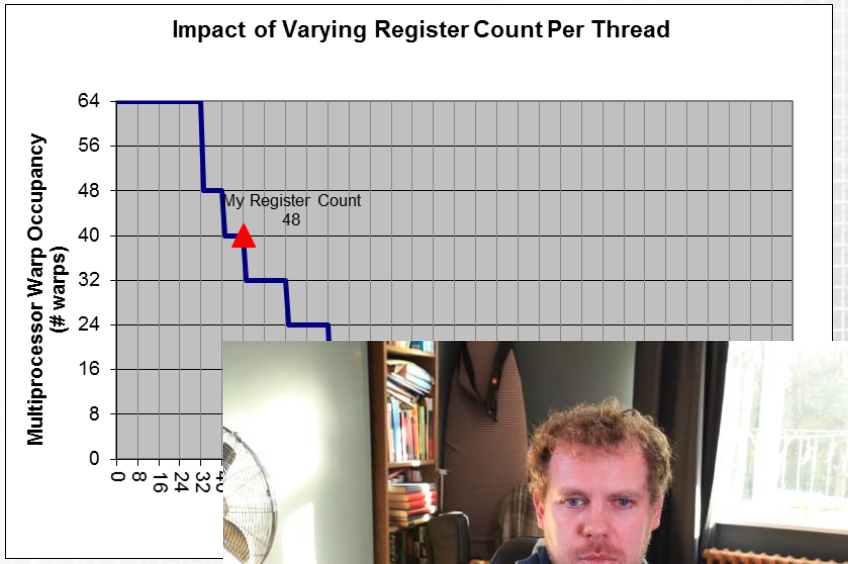
- Thread Block Limitations
  - Always a factor of 32 (warp size)
- Changing the thread block size will change occupancy
  - If thread block size is too small
    - There is a fixed limit on the number of active thread blocks per SM (16 in Kepler, 32 in Maxwell)
  - If thread block is too big
    - Not enough resources for another block
    - Block is stalled until enough resource is available
- The relationship between thread block size and occupancy is non linear
  - Complex interplay between resources

Occupancy Calculator

- The CUDA Occupancy calculator is available for download
  - http://developer.download.nvidia.com/compute/cuda/CUDA_Occupancy_calculator.xls
  - By giving a Compute Capability, Threads per block usage registers per thread and shared memory per block occupancy can be predicted.
  - It will also inform you what factor is limiting occupancy (registers, SM, block size)



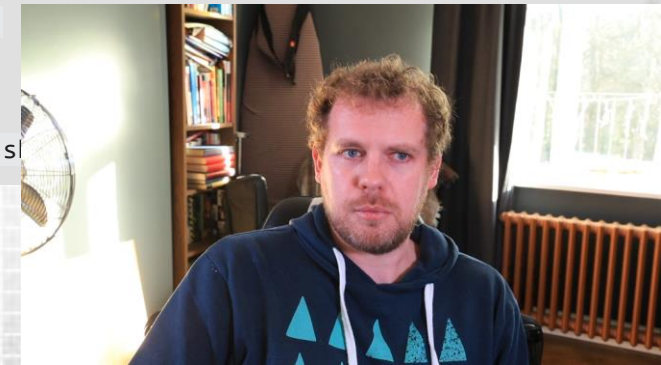




Occupancy Calculator

- How do I know what my SM usage per block is?
  - You either statically declared it or dynamically requested it as a kernel argument
- How do I know what my register usage is?
  - CUDA build rule Device Properties-> Verbose PTX Output = Yes
    - i.e. nvcc –ptxas-options=-v

```
1>------ Build started: Project: Lab06, Configuration: Debug Win32 ------
1>  Compiling CUDA source file kernel.cu...
1>
1>  E:\Lab06>"nvcc.exe" -gencode=arch=compute_35,code=\"sm_35,compute_35\" --use-local-env --cl-
version 2013 -ccbin "C:\MSVS12.0\VC\bin"  -I"C:\CUDA\v7.0\include" -I"C:\CUDA\v7.0\include"  -G   -
-keep-dir Debug -maxrregcount=0 --ptxas-options=-v --machine 32 --compile -cudart static -Xptxas -
dlcm=ca  -g   -D__CUDACC__ -DWIN32 -D_DEBUG -D_CONSOLE -D_MBCS -Xcompiler "/EHsc /W3 /nologo /Od
/Zi /RTC1 /MDd  " -o Debug\kernel.cu.obj "E:\Lab06\kernel.cu"
1>  ptxas info    : 0 bytes gmem
1>  ptxas info    : Function properties for cudaGetDevice
1>      8 bytes stack frame, 0 bytes spill stores, 0 bytes spill loads
1>  ptxas info    : Function properties for cudaFuncGetAttributes
1>      8 bytes stack frame, 0 bytes spill stores, 0 bytes spill loads
1>  ptxas info    : Function properties for cudaOccupancyMaxActiveBlocksPerMultiprocessorWithFlags
1>      24 bytes stack frame, 0 bytes spill stores, 0 bytes spill loads
1>  ptxas info    : Function properties for cudaDeviceGetAttribute
1>      16 bytes stack frame, 0 bytes spill stores, 0 bytes spill loads
1>  ptxas info    : Compiling entry function '_Z9addKernelPiS_S_' for 'sm_35'
1>  ptxas info    : Function properties for _Z9addKernelPiS_S_
1>      0 bytes stack frame, 0 bytes spill stores, 0 bytes spill loads
1>  ptxas info    : Used 6 registers, 332 bytes cmem[0]
1>  ptxas info    : Function properties for cudaMalloc
1>      8 bytes stack frame, 0 bytes spill stores, 0 bytes spill loads
1>  ptxas info    : Function properties for cudaOccupancyMaxActiveBlocksPerMultiprocessor
1>      16 bytes stack frame, 0 bytes spill stores, 0 bytes spill loads
1>  kernel.cu
1>  Lab06.vcxproj -> E:\Lab06.exe
1>  copy "C:\CUDA\v7.0\bin\cudart*.dll" "E:\Lab06\Debug\"
1>  C:\CUDA\v7.0\bin\cudart32_70.dll
1>  C:\CUDA\v7.0\bin\cudart64_70.dll
1>          2 file(s) copied.
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 s
```



Intelligent Launching

- Since CUDA 6.5 It is possible to launch block sizes to maximise occupancy (using the Occupancy API)
  - This does not guarantee good performance!
  - However: Usually a good compromise
- `cudaOccupancyMaxPotentialBlockSize`: will find best block size and minimum grid size
  - Actual grid size must be calculated

```
int blockSize;
int minGridSize;
int gridSize;

cudaOccupancyMaxPotentialBlockSize(&minGridSize, &blockSize, MyKernel, 0, 0);
gridSize = (arrayCount + blockSize - 1) / blockSize; //round up
MyKernel <<< gridSize, blockSize >>>(d_data, arrayCount);
```

Static SM size

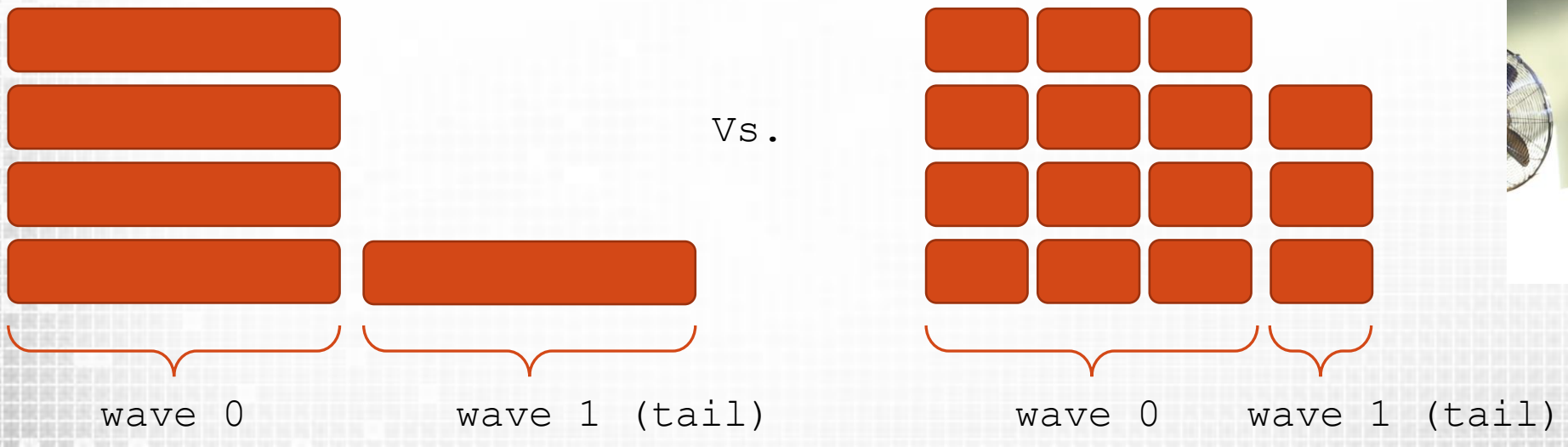
Occupancy SDK for Shared Memory

- What if SM use varies depending on block size?

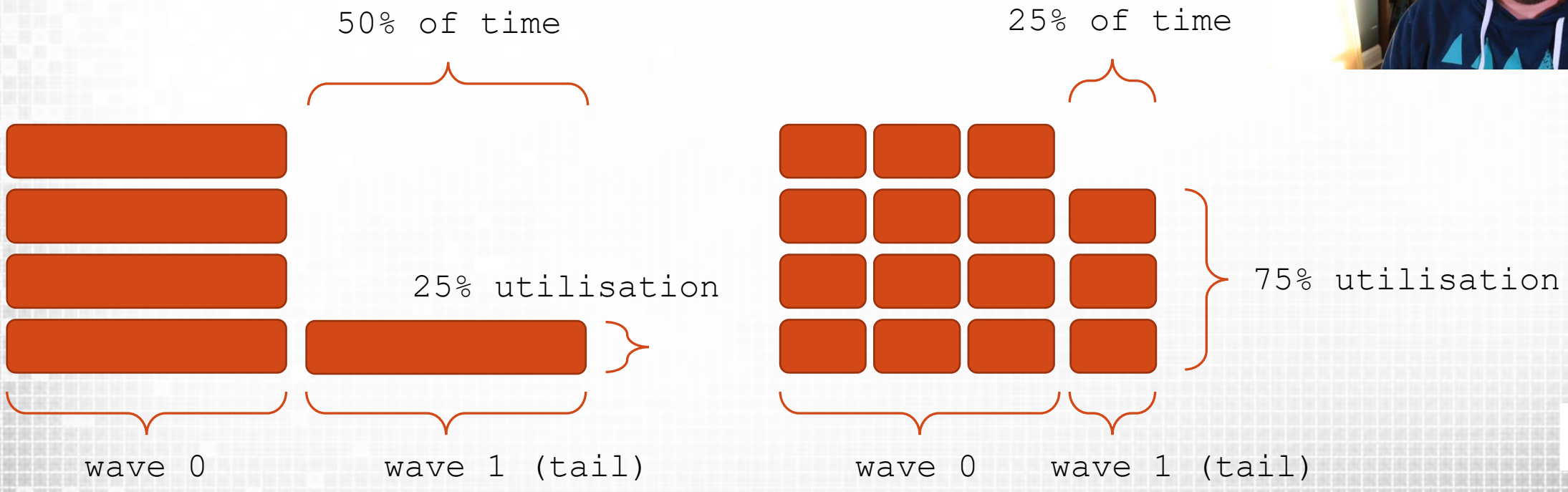
```
int SMFunc(int blockSize){
  return blockSize*sizeof(int);
}



void launchMyKernel(int *d_data, int arrayCount)
{
  int blockSize;
  int minGridSize;
  int gridSize;

  cudaOccupancyMaxPotentialBlockSizeVariableSMem(&minGridSize, &blockSize, MyKernel, SMFunc, 0);
  gridSize = (N + blockSize - 1) / blockSize;
  MyKernel <<< gridSize, blockSize, SMFunc(gridSize) >>>(d_data, arrayCount);
}
```

Other considerations for block sizes

- Waves and Tails
  - A Wave is a set of thread blocks that run concurrently on the device
    - Grid launch may have multiple waves
  - A Tail is the partial thread block left as a result of dividing problem size by thread block dimensions
- Performance Implications
  - Larger thread blocks sizes may result in inefficient execution

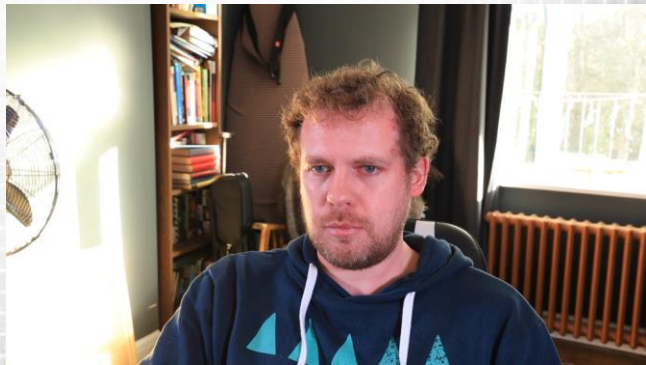


Other considerations for block sizes



Summary

- Occupancy
  - Define occupancy as a metric of resource utilisation
  - Explain the implications of occupancy on performance
  - Demonstrate methods to maximise occupancy
  - Examine the impact of thread block size on occupancy and performance



Acknowledgements and Further Reading

- Cache line sizes
- GPU Performance Analysis
  - http://on-demand.gputechconf.com/gtc/2012/presentations/S0514-GTC2012-GPU-Performance-Analysis.pdf
- Waves and Tails (http://on-demand.gputechconf.com/gtc/2012/presentations/S0514-GTC2012-GPU-Performance-Analysis.pdf)
- How to enable use of L1 cache
  - http://acceleware.com/blog/opt-in-L1-caching-global-loads
- Better Performance at Lower Occupancy
  - http://nvidia.fullviewmedia.com/gtc2010/0922-a5-2238.html