Friedrich-Alexander-Universität
Technische Fakultät

Computer Architecture Practical Exercise

8 CUDA Jacobi 2D

Kenan Gündogan[1] **Philipp Gündisch**[1]

[1]Friedrich-Alexander Universität Erlangen-Nürnberg, Chair of Computer Science 3 (Computer Architecture)

January 22, 2024

Task 9.1

2D Jacobi CUDA

- Reuse code from STREAM benchmark
- Implement an `update_grid()` function as a CUDA kernel
- Allocate and initialize the grids on host memory
- Copy the grids to the GPU (see `cudaMalloc` and `cudaMemcpy`)
- Keep the grids in the GPU memory until time is up
- Copy back the final result (**check results** and measure transfer time)
- Calculate the performance with and without copy overhead
- **Benchmark over the runtime** while keeping the grid size fixed at 3 GiB

```
// Measure for min_runtime_us with 100ms, 1s and 10s
for(runs = 1u;  actual_runtime_us < min_runtime_us; runs=runs<<1u) {
    ...
}
```

CUDA Kernels

Hints

- Update your C implementation with the CUDA kernel call
- Consider the asynchronous nature of kernel calls when measuring the runtime
- Assign each thread to a single grid cell (*blockIdx, threadIdx, blockDim*)
- Use a two dimensional mapping
  - `threadsPerBlock.{x,y}, blocksPerGrid.{x,y}`
  - Consider the boundaries for the maximum number of threads per thread block
  - Details can be found in the technical datasheet of the GPU

Appendix: Checklist

Performance Optimization (1/2)

During the timeline of this class new bullet points will be added. Recently added entries are bold.

- Compiling
  - Choice of the compiler (`icc`)
  - Compiler flag to optimize aggressively (e.g. `-O3`)
  - Compiler flag to adapt for specific hardware (e.g. `-xHost`)
- Programming Techniques (if applicable)
  - Use `#define` and `const` instead of variables
  - Data type aware programming
  - Use aligned memory (e.g. `_mm_malloc()` or `posix_memalign()`)
  - Consecutive address iteration
  - Variable declarations outside of loops
  - Reduce function calls
  - Use intrinsics (to utilize SIMD)
  - Cache aware programming (Spatial Blocking)
  - Prefetcher aware programming (L1 Cache Blocking)

Appendix: Checklist

Performance Optimization (2/2)

During the timeline of this class new bullet points will be added. Recently added entries are bold.

- Measurement
  - Reasonable benchmark time
  - Reasonable benchmark workload
  - Reduce interference factors to a minimum
  - **GPU: Consider memory transfer overhead**
- Optimization Process
  - Check assembler code while optimizing
  - Check performance gains while optimizing
  - Use profiling tools
  - Ensure correctness of code
  - Optimize iteratively
  - Optimize single core performance first
  - Parallelize your code on the CPU first