Contents

Background

In the previous meeting , we implemented an image blur algorithm.We've modified the code to resolve I/O bottlenecks, and we are now faced with the need to optimize the computation as the CPU is now the bottleneck. Hardware Acceleration using the FPGA capabilities is now an option to consider.

Converting Software Code to Hardware Accelerators on FPGAs

FPGA Background

FPGAs are integrated circuits that can be reconfigured after manufacturing. They consist of an array of configurable logic blocks (CLBs), interconnected by programmable interconnects. This allows designers to implement custom hardware logic, tailored to specific applications.

Early Days: ASIC Prototyping and Debugging (1980s - early 1990s)

- **ASIC Limitations:** Before FPGAs became widespread, Application-Specific Integrated Circuits (ASICs) were the dominant technology for custom hardware. ASICs are designed for a very specific purpose and are manufactured in large quantities. However, they have a major drawback: once manufactured, they can't be changed. Any errors in the design or changes in requirements would necessitate a costly and time-consuming redesign and remanufacturing process.

- **FPGAs as Prototyping Tools:** FPGAs emerged as a solution to this problem. Their reconfigurability allowed designers to prototype their ASIC designs on FPGAs.[2] This meant they could:

  - **Verify functionality:** Test the logic and behavior of their design in a real hardware environment before committing to an expensive ASIC fabrication.
  - **Debug errors:** Identify and fix bugs in the design by simply reprogramming the FPGA, avoiding costly respins of the ASIC.[3]
  - **Emulate performance:** Get an early estimate of the performance of their ASIC design.
- **Early FPGA Characteristics:** Early FPGAs had limited capacity and performance compared to ASICs. They were primarily seen as a tool for prototyping and low-volume production.

Growth and Expansion: Beyond Prototyping (mid-1990s - 2000s)

- **Increased Capacity and Performance:** As FPGA technology advanced, their capacity and performance increased significantly. This made them suitable for a wider range of applications beyond just prototyping.
- **New Applications:** FPGAs started to be used in various applications such as:
  - **Telecommunications:** Implementing custom communication protocols and signal processing.[4]
  - **Networking:** Building network interface cards and routers.
  - **Industrial control:** Implementing real-time control systems.[5]
- **Emergence of EDA Tools:** The development of sophisticated Electronic Design Automation (EDA) tools, such as synthesis and place-and-route software, made it easier to design complex circuits on FPGAs.

The Rise of Acceleration: High-Performance Computing (2010s - Present)

- **Meeting Demanding Performance Needs:** With the increasing demand for high-performance computing in areas like data centers, artificial intelligence, and scientific computing, FPGAs found a new role as hardware accelerators.[6]
- **Offloading Computation from CPUs:** FPGAs are now used to offload computationally intensive tasks from CPUs, providing significant speedups for specific workloads.[7]
- **Cloud Computing and Data Centers:** Major cloud providers like Microsoft and Amazon have integrated FPGAs into their data centers to accelerate various applications.[8]
- **High-Level Synthesis (HLS):** The development of HLS tools has further simplified FPGA development by allowing programmers to use high-level languages like C/C++ to design hardware.[9]

FPGAs started as a tool to aid in ASIC development and debugging, providing a flexible and cost-effective way to prototype and verify designs. However, **continuous advancements in technology have transformed them into powerful hardware accelerators** capable of meeting the demanding performance needs of modern applications. They are now an integral part of high-performance computing, cloud computing, and various other fields.

Observation:

FPGA with a clock speed of 100 MHz can outperform a CPU with a clock speed of several GHz.
Here's why FPGAs can achieve significant acceleration despite lower clock speeds:

Parallelism:

This is the key. CPUs execute instructions sequentially, one after another. FPGAs, on the other hand, can perform many operations simultaneously. Imagine a CPU as a single worker doing tasks one by one, while an FPGA is like an entire factory with many workers performing different parts of a task at the same time. This massive parallelism compensates for the lower clock speed.

Custom Hardware:
When you program an FPGA, you're essentially designing a custom hardware circuit tailored to your specific task. This means that the operations are implemented directly in hardware, rather than being translated into instructions that the CPU has to decode and execute. This direct hardware implementation is much more efficient.

Pipelining:
FPGAs can be designed to use pipelining, where different stages of a computation are performed concurrently, like an assembly line. This increases the throughput of the system, even if the individual operations take the same amount of time.

Specialized Logic:
FPGAs are made up of configurable logic blocks that can be arranged to perform specific operations very efficiently. For example, if you need to perform a lot of multiplications, you can configure the FPGA to have dedicated multipliers, which are much faster than performing multiplications on a general-purpose CPU.

In summary:
FPGAs accelerate tasks not by having a higher clock speed, but by:

- Performing many operations in parallel
- Implementing custom hardware circuits
- Using pipelining to increase throughput
- Having specialized logic for specific operations

This makes them particularly well-suited for tasks that can be parallelized, such as digital signal processing, image processing, and cryptography.

Guidelines for Converting Software to Hardware

Profiling and Analysis:

- **Identify Bottlenecks:** Use profiling tools to pinpoint performance-critical sections in the software code.
- **Analyze Data Flow:** Understand the data dependencies and parallelism within the algorithm.

Algorithm Transformation:

- **Expose Parallelism:** Restructure the algorithm to maximize parallelism.
- **Optimize Data Structures:** Choose appropriate data structures for hardware implementation.
- **Fixed-Point Arithmetic:** Consider using fixed-point arithmetic to reduce hardware complexity.

Hardware Design:

- **Hardware Description Languages (HDLs):** Use Verilog to describe the hardware logic.
- **High-Level Synthesis (HLS):** Use HLS tools to automatically generate HDL code from high-level languages like C/C++.
- **Design for Pipelining:** Implement pipelining to increase throughput.
- **Memory Management:** Design efficient memory interfaces for data access.

Verification and Testing:

- **Simulation:** Use simulation tools to verify the functionality of the hardware design.
- **Emulation:** Use FPGA-based emulation platforms for faster verification.
- **Hardware Testing:** Test the implemented design on the target FPGA hardware.

The host CPU & Accelerator (like a GPU, FPGA, or specialized ASIC)

The relationship between the two in terms of data movement, and algorithm flow is crucial for understanding how heterogeneous computing systems work.

1. Host CPU's Role:

- **Orchestration:** The host CPU is the master controller. It manages the overall execution of the program, including:
  - Initializing the accelerator.

- Allocating memory on both the host and the accelerator.
    - Transferring data between the host and the accelerator.
    - Launching kernels (the code executed on the accelerator).[2]
    - Synchronizing execution and retrieving results.
- **Sequential Code:** The host CPU typically executes the sequential parts of the algorithm, which may include:
    - Preprocessing data.[3]
    - Postprocessing results.
    - Control flow logic that determines when and how to use the accelerator.

2. Accelerator's Role:

- **Parallel Execution:** The accelerator excels at executing data-parallel portions of the algorithm. These are often computationally intensive loops or operations that can be performed independently on multiple data elements simultaneously.
- **Kernel Execution:** The code that runs on the accelerator is called a *kernel*. The host CPU launches kernels on the accelerator, and the accelerator executes them in parallel.

3. Data Movement:

- **Host to Accelerator:** Data required by the kernel must be transferred from the host's main memory to the accelerator's memory. This is often the most significant bottleneck in heterogeneous computing.
- **Accelerator to Host:** After the kernel completes execution, the results must be transferred back from the accelerator's memory to the host's memory.
- **Data Transfer Mechanisms:** Different accelerators use different mechanisms for data transfer:
    - **DMA (Direct Memory Access):** Allows the accelerator to access host memory directly without CPU intervention.
- **Data Transfer Overhead:** Minimizing data transfer is critical for performance. Techniques like:
    - **Data locality:** Keeping data on the accelerator for as long as possible.
    - **Coalesced memory access:** Arranging data in memory to optimize transfers.
    - **Overlapping computation and communication:** Transferring data while the accelerator is still computing.

4. Algorithm Flow:

The typical flow of an algorithm using a CPU accelerator is as follows:

1. **Host Initialization:** The host CPU initializes the accelerator and allocates necessary memory.
2. **Data Transfer (Host to Accelerator):** The host CPU transfers the input data to the accelerator's memory.
3. **Kernel Launch:** The host CPU launches the kernel on the accelerator.
4. **Accelerator Execution:** The accelerator executes the kernel in parallel.
5. **Data Transfer (Accelerator to Host):** The accelerator transfers the results back to the host's memory.
6. **Host Postprocessing:** The host CPU processes the results.

questions:

1. What is the primary bottleneck in the image blur algorithm after the I/O optimizations?
2. Why are FPGAs considered suitable for accelerating the image blur algorithm?
3. Explain the concept of "parallelism" in the context of FPGAs and how it contributes to their performance advantage.
4. What are the key stages involved in converting software code to hardware on an FPGA?
5. How does the clock speed of an FPGA compare to a CPU, and why can an FPGA still achieve superior performance?
6. Describe the role of the host CPU in a heterogeneous computing system with an FPGA accelerator.
7. What are the critical factors in minimizing data transfer overhead between the host CPU and the FPGA?

8. Write a c code, the code should have an M*N byte image as an input, the code should printout the image a word at a time (4 bytes), while duplicating the value of the last raw and line + pedding with zero if there are missing bytes for a full word.
   - For example if the input is 2x2 byte
   - {0,1,
   - 2,3}
   - The output should be 3 words
   - {0,1,1,0}
   - {2,3,3,0}
   - {2,3,3,0}
9. Modify your code to add a 2nd printout of words,
   - the 2nd print out should plot the same result, starting with the 2nd line.
     - {2,3,3,0}
     - {2,3,3,0}
   - The 2 printouts should be interleaved (first print, 2nd print, first print, 2nd print)