GPU Computing

Exercise Sheet 01

Maximilian Richter, Jan Kränzke, Markus Everling,
Nicolas Schledorn, Olga Sergeyeva, Xel Pratscher

October 2024

1 Reading

1.1 John Montrym and Henry Moreton. 2005. The GeForce 6800. IEEE Micro 25, 2 (March 2005), 41-51.

1.1.1 Primary Contribution:

The paper presents the design of the GeForce 6800 GPU, detailing its novel architecture and features (for 2005) like programmable vertex and fragment processors, with an emphasis on the interplay between programmability, parallelism, and memory characteristics to achieve efficient real-time rendering comparable to cinematic quality. The authors highlight the challenges of handling complex graphical scenes, including shadow volumes, high-dynamic-range rendering, and increasing demands for realism, which require high precision calculations and efficient memory utilization. They explain how the GeForce 6800 addresses these challenges through its parallel processing capabilities, specialized functional units for tasks like rasterization and texture filtering, and a flexible memory subsystem that supports various DRAM types and incorporates lossless compression.

1.1.2 Key Insights:

A crucial takeaway is the growing importance of programmability in GPUs, enabling developers to implement custom algorithms and accelerate diverse rendering techniques beyond traditional fixed-function pipelines. The authors stress the need for latency tolerance and efficient data handling due to the high memory bandwidth demands of modern graphics processing, showing how the GeForce 6800's architecture, with its multiple independent processors and optimized data paths, tackles these issues. The paper effectively illustrates the shift in GPU design from fixed-function hardware to programmable platforms capable of handling complex computations and contributing to advancements in areas beyond traditional graphics rendering, such as general-purpose computing and simulations.

1.1.3 Our Opinion:

The paper was remarkably accurate in predicting the direction of GPU development. Its emphasis on programmability, parallelism, and memory bandwidth proved to be surprisingly accurate. GPUs have become increasingly programmable, with sophisticated shader models and general-purpose computing capabilities (CUDA or OpenGL) becoming central to their functionality, as the authors anticipated. The rise of deep learning and AI further underscores the paper's foresight, as these fields heavily rely on the parallel processing power of GPUs. However, the paper might have underestimated the extent to which GPUs would evolve beyond graphics rendering. Today, GPUs play a crucial role in diverse fields like scientific computing, data analysis, and cryptocurrency mining, highlighting a broader impact than the authors may have initially envisioned as many of the technologies of today were not yet fully developed at the time of writing this article.

I would accept the paper.

1.2 Leslie G. Valiant. 1990. A bridging model for parallel computation. Commun. ACM 33, 8 (August 1990), 103-111.

1.2.1 Primary Contributions:

This paper introduces the Bulk-Synchronous Parallel (BSP) model as a way to bridge the gap between parallel software and hardware, similar to how the von Neumann model serves sequential computing. The author argues that a unifying model like BSP is crucial for the widespread adoption of parallel computing, providing a standard for both hardware designers and software developers. The paper presents theoretical arguments and results to demonstrate the BSP model's efficiency in handling various aspects of parallel computation, such as memory management, algorithm design, and communication

1.2.2 Key Insights:

The key insight is the importance of "parallel slackness" in achieving optimal efficiency on the BSP model. This means having more virtual processors than physical processors, allowing the system to schedule and pipeline tasks effectively. The author emphasizes that this slack enables efficient memory management through hashing and facilitates the simulation of concurrent memory accesses. The paper also highlights the flexibility of the BSP model, allowing programmers to control memory and communication explicitly for improved performance when necessary.

1.2.3 Our Opinion:

The paper was quite visionary in its identification of the need for a bridging model to facilitate parallel computing. The BSP model, with its emphasis on bulk-synchronous operation, has indeed influenced the design of parallel algorithms and programming models. For example, consider how Google Maps finds shortest routes. It divides the map into a grid, where each grid cell contains precomputed shortest paths. This is like assigning portions of a problem to different processors, embodying the "parallel slackness" concept. When you request a route, Google Maps combines these precomputed paths, similar to how processors synchronize and exchange data in the BSP model. This demonstrates how BSP provides a framework for understanding parallel computation. Its principles are reflected in libraries like OpenMP and MPI (I am however not a big fan of Intel or MPI for that matter). However, the paper's predictions about communication costs haven't entirely held. Advancements in network bandwidth and interconnects(100+ Gbit/s Infiniband were just unthinkable in 1990) have mitigated the need for strict bulk-synchronous operations in some cases.

2 Amdahl's law

(1) Amdahl's law states that a program, of which a fraction p can be parallelized and a fraction $s := 1 - p$ must be run serially, has the following upper bound on the runtime speedup a when executed using N processors:

$$a = \frac{1}{(1-p) + \frac{p}{N}}$$

It can be derived as follows:

Let S_1 denote the runtime of the program when executed on one processor, and S_N denote the runtime when executed on N processors. The speedup then is $a = \frac{S_1}{S_N}$.

Since only the parallelizable part p of the program benefits from the extra processors, only that portion of the runtime becomes faster by a factor of N, so $S_N = sS_1 + \frac{p}{N}S_1 = S_1\left(s + \frac{p}{N}\right)$. Thus

$$a = \frac{S_1}{S_N} = \frac{S_1}{S_1\left(s + \frac{p}{N}\right)} = \frac{1}{s + \frac{p}{N}} = \frac{1}{(1-p) + \frac{p}{N}}$$

(2) Generally, Amdahl's law can be very useful for roughly estimating the theoretical speedup from parallelizing a process. However, there are applications that overshoot or undershoot the theoretical bound given by the law:

- The law does not take into consideration the overhead from dispatching to different processors and memory synchronization. Certain memory access patterns can lead to effects such as False sharing, which dramatically reduce the performance of the process, in the worst case making it run slower than the baseline serial process.
- In some cases, distributing the work across many processors can lead to superlinear speedup, which exceeds the upper bound stated by Amdahl. One example of this is when a data set, distributed across many processors, fits fully into the processors' caches, whereas a single processor would have to spend a lot more time waiting on memory accesses.

(3) (a) $p = \frac{1}{2}, N = 4$. The speedup given by Amdahl's law is

$$a = \frac{1}{\frac{1}{2} + \frac{1}{8}} = \frac{1}{\frac{5}{8}} = \frac{8}{5} = 1.6$$

(b) $p = \frac{1}{5}, N = 8$. The speedup given by Amdahl's law is

$$a = \frac{1}{\frac{4}{5} + \frac{1}{40}} = \frac{1}{\frac{33}{40}} = \frac{40}{33} \approx 1.21$$

(c) Even though in (b) the number of processors is double that in (a), the overall speedup is much lower, 21% compared to 60%. This suggests that the parallelizable fraction p plays a much bigger role in the overall speedup than the number of cores. In fact, with $p = 0.2$, the best possible speedup would be $\frac{1}{1-0.2} = 1.25$, whereas with $p = 0.5$, the best possible speedup would be 2.

3 Willingness to Present

- Reading: True
- Amdahl's Law: True