GPU Computing - Architecture and Programming 25/26 WiSe **(Submission: October 22, 2025)**

Exercise #1

Du, Qilong
Yu, Mengbi
Cao, Qingyang

Exercise 1.1 Reading [10 points]

Read the following two papers and provide reviews as explained in the first lecture (see slides):

- Leslie G. Valiant. 1990. *A bridging model for parallel computation.* Commun. ACM **33**, 8 (August 1990), 103–111.
- John Montrym and Henry Moreton. 2005. *The GeForce 6800.* IEEE Micro **25**, 2 (March 2005), 41–51.

Solution 1.1:

Valiant's 1990 paper remains a cornerstone of parallel computing because it formalized how algorithmic parallelism and hardware performance can be connected through a simple analytical model. The **Bulk Synchronous Parallel paradigm** anticipates the structure of today's GPU computing: computation organized in kernels or "supersteps," separated by **data exchange** and **synchronization**. By abstracting **communication** and **barrier** costs with only two parameters, (g, L), the BSP model offers both theoretical clarity and practical guidance for designing scalable programs. Despite its simplifying assumptions, it has proven remarkably durable, shaping later frameworks such as MPI, MapReduce, and GPU kernels. BSP elegantly explains **why GPUs succeed**: they tolerate latency through massive parallel slackness and predictable synchronization.

Montrym and Moreton's paper records the moment when GPUs became truly programmable parallel processors. By combining high-throughput vector units with programmable control, the **GeForce 6800** defined the modern GPU blueprint and initiated the convergence between graphics and computation. It illustrates how **architectural specialization can evolve into generality**: a design once aimed at pixels ultimately laid the foundation for data-parallel scientific computing. And in the context of GPU Computing, it stands as the practical realization of the theoretical models introduced by Valiant—demonstrating that large-scale parallelism **can be engineered effectively in silicon**.

Exercise 1.2 Amdahl's Law [10+6 points]

1. Derive Amdahl's Law.
2. Comment on the correctness of this law regarding real applications. Is it accurate in its prediction, or reporting too high or too low performance? Provide examples for the last two cases.
3. (a) Calculate the overall speedup using Amdahl's Law
   - $P = 0.5$ (50% of the program can be parallelized).
   - $N = 4$ (number of processors).

(b) Calculate the overall speedup if the number of processors is increased to 8.

- $P = 0.2$ (20% of the program can be parallelized).
- $N = 8$ (number of processors).

(c) Compare the results. What can you conclude from this comparison?

Solution 1.2:

1. Derivation of Amdahl's Law

Assume that the baseline serial execution time is normalized to 1. Split the work into a serial fraction s and a parallelizable fraction p with

$$s + p = 1.$$

With N parallel execution units $\Rightarrow$

- the serial part still takes time s (cannot be parallelized).
- the parallel part is ideally divided across N units, taking time p/N (no overheads).

$\Rightarrow$ the parallel execution time is $t_{\text{par}} = s + \frac{p}{N}$, and speed-up is the ratio of baseline time to parallel time:

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

and take the upper bound as $N \to \infty$: $a_{\max} = \lim_{N\to\infty} \frac{1}{s+\frac{p}{N}} = \frac{1}{s}$.

2. Comment on the law

The validity of Amdahl's Law in real applications : Amdahl's Law is **theoretically correct** under its assumptions: a fixed problem size, perfect load balancing, and no communication or synchronization overhead. However, these assumptions rarely hold in real systems. In practice, the law often **overestimates achievable performance** because it neglects costs introduced by parallelization itself—such as thread management, data transfer, and synchronization delays.

Overestimation example: In GPU-accelerated matrix multiplication, theoretical speed-up might be $a = 1/(s + p/N)$ with $s \approx 0.01$. For $N = 1000$, this predicts nearly $100\times$ acceleration. Yet in practice, host-device memory transfers and kernel launch overheads reduce speed-up to about $30\times$–$40\times$. The law reports **too high** performance because the effective serial portion is larger than measured in pure computation.

Underestimation Example: Conversely, when the problem size grows with N—for instance, in climate simulations or particle dynamics—parallel efficiency may remain high even for many cores. Here, Amdahl's Law is **too pessimistic**. Gustafson's Law captures this better by scaling the workload with the available parallel resources.

3. Calculation

(a) Let $p = 0.5$ (50% of the program can be parallelized) and $N = 4$ processors:

$$a = \frac{1}{(1-0.5) + \frac{0.5}{4}} = \frac{1}{0.5 + 0.125} = \frac{1}{0.625} = 1.6.$$

Hence, the overall speed-up is $1.6\times$.

(b) Let $p = 0.2$ (20% of the program can be parallelized) and $N = 8$ processors:

$$a = \frac{1}{(1-0.2)+\frac{0.2}{8}} = \frac{1}{0.8+0.025} = \frac{1}{0.825} \approx 1.21.$$

Hence, the overall speed-up is approximately $1.21\times$.

(c) Comparison and conclusion : although the number of processors increases from 4 to 8, the speed-up decreases from 1.6 to 1.21 because the parallel fraction p is smaller. This indicates an Amdahl's key message: performance is limited by the serial portion $(1-p)$, not by the number of processors N. Increasing N only helps when p is large.

Exercise 1.3 Willingness to present [13 points]

- Reading (section: 1.1)
- Amdahl's law (section: 1.2)