GIK INSTITUTE
OF ENGINEERING SCIENCES AND TECHNOLOGY

2025/02

Student Name: Abdullah Ejaz Janjua

Reg: 2023038

Assignment No: 01

Instructor: Waheed Ahmad

FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Ghulam Ishaq Institue of Engineering Sciences and Technology, Topi, Swabi

Part A - Trends, Cost & Performance Indicators 25 marks

1. **Explain bandwidth vs latency with one hardware example for each.**

   - **Bandwidth:** The total amount of work done in a given time. It's essentially the throughput of the system. For example, disk bandwidth indicates the amount data a disk can transfer per second.
   - **Latency:** The time between start and completion of an event. It's essentially the response time of the system. For example, disk access latency indicates the time it takes for the disk to retrieve or store data after a request is made.

2. **Manufacturing Cost: A 30 cm wafer costs $2000 and yields 640 dies. If the die yield is 78.7% and packaging/test yield is 95%, compute the cost per packaged IC. Show steps and formulae.**

   We have the following information:

   - **Diameter of Wafer:** 30cm
   - **Cost of Wafer:** $2000
   - **Dies per Wafer:** 640
   - **Die Yield:** 78.7%
   - **Packaging/test yield:** 95%
   - **Cost of testing die:** $1 (Assumed from slides)
   - **Cost of Packaging & Final Die:** $1 (Assumed from slides)

   As we already have Die Yield and Dies Per Wafer, we simply have to compute Cost of Die to compute Cost of IC.
   First, we compute Cost of Die as:

$$\text{Cost of Die} = \frac{\text{Cost of Wafer}}{\text{Dies per Wafer} \times \text{Die Yield}}$$

$$= \frac{2000}{640 \times 0.787}$$

$$= \frac{2000}{503.68}$$

$$= \$3.97$$

   Now, we compute the cost of IC as:

$$\text{Cost of IC} = \frac{\text{Cost of } \{\text{Die + Testing Die + Packaging \& Final Die}\}}{\text{Final Test Yield}}$$

$$= \frac{3.97 + 1 + 1}{0.95}$$

$$= \frac{5.97}{0.95}$$

$$= \$6.28$$

Thus, our cost of per package IC ends up to total of **$6.28**.

3. **Historical rule-of-thumb: "Bandwidth grows at least as the square of the improvement in latency." Interpret and give a plausible reason (architectural or technological) for why this trend has held.**

   The historical rule of thumb, "Bandwidth grows at least as the square of the improvement in latency," suggests that as latency decreases, the system's ability to handle more work increases. This makes sense because, as jobs take less time to complete (i.e., latency decreases), bandwidth naturally increases, since the system can accomplish more work in the same time frame. The reason this trend has held (i.e, latency lagging behind bandwidth) is both architectural and technological. The scaling of semiconductor processes results in faster transistors and a greater number of them on a chip. While the increased speed and number of transistors boost bandwidth, the larger number of transistors also increases the distance between components, which limits the reduction in latency.

Part B - Availability & Dependability Scenario 25 marks

Your campus service runs on either (i) a single 12-core SMP server or (ii) a 3-node cluster with 4 cores each over Gigabit Ethernet. Discuss how each option behaves if one core/node fails, and outline one strategy per option to maintain service quality.

In (i), when a single core fails, the remaining 11 cores are still functional. In this instance, the workload will be divided between the remaining active cores through load balancing, maintaining service quality. In (ii), when a node fails, two nodes are still active. In this case, the system could either activate a backup node or divide the workload between the remaining active nodes, maintaining service quality.. In both systems, there will be some performance degradation.

Part C – Amdahl's Law 25 marks

1. **For parallel = 0.90 and cores ∈ {1, 2, 4, 8}, compute speedup S() and efficiency E(). Present as a table. Briefly comment on the diminishing returns.**

   Speed-up is defined as:

$$S = \frac{1}{f + \frac{1-f}{p}}$$

   Where, p is the number of cores, T_1 is the execution time on 1 processor, T_p is the execution time on p processors, and f is the fraction of a program that is sequential. Efficiency is defined as:

$$E = \frac{S}{p}$$

   As the parallel region $(1 - f)$ is 0.9 then the serial region is f is 0.1.

   As shown in Table 1, Speed-up increases with the number of cores but at a decreasing rate. For example, going from 1 to 2 cores gives a speed-up of 1.81, but going from 4 to 8 cores only gives a speed-up of 4.70. This is the diminishing return phenomenon. Efficiency decreases as the number of cores increases. This happens because, as the number of cores grows, the serial portion of the program remains at 0.1.

Number of Cores	Speed-Up	Efficiency
1	$\frac{1}{0.1+\frac{0.9}{1}} = 1.00$	$\frac{1.00}{1} = 1.00$
2	$\frac{1}{0.1+\frac{0.9}{2}} = 1.81$	$\frac{1.81}{2} = 0.9$
4	$\frac{1}{0.1+\frac{0.9}{4}} = 3.07$	$\frac{3.07}{4} = 0.76$
8	$\frac{1}{0.1+\frac{0.9}{8}} = 4.70$	$\frac{4.70}{8} = 0.58$

Table 1: Comparison of Speed-Up and Efficiency Relative to Number of Cores.

2. **Given parallel region = 0.92, what is the minimum core-count p needed to achieve S() ≥ 6? Show algebraic steps.**

$$S = \frac{1}{f + \frac{1-f}{p}} \geq 6$$

$$\frac{1}{0.08 + \frac{0.92}{p}} \geq 6$$

$$0.08 + \frac{0.92}{p} \leq \frac{1}{6}$$

$$\frac{0.92}{p} \leq \frac{1}{6} - 0.08$$

$$\frac{0.92}{p} \leq 0.0867$$

$$p \geq \frac{0.92}{0.0867}$$

$$p \geq 10.61$$

Thus, we need a minimum of **11** cores to achieve a speedup of ≥ 6.

3. **Two teams optimize different parts: Team A increases parallel from 0.90 → 0.93 at p = 8; Team B adds cores 8 → 16 at parallel=0.90. Which yields higher speedup? Justify numerically**

For parallel = 0.90 and cores 8 we have::

$$\begin{aligned} S &= \frac{1}{f + \frac{1-f}{p}} \geq 6 \\ &= \frac{1}{0.1 + \frac{0.9}{8}} \\ &= 4.70 \end{aligned}$$

For team A, we have speed-up:

$$\begin{aligned} S &= \frac{1}{f + \frac{1-f}{p}} \geq 6 \\ &= \frac{1}{0.07 + \frac{0.93}{8}} \\ &= 5.3 \end{aligned}$$

For team B, we have:

$$\begin{aligned} S &= \frac{1}{f + \frac{1-f}{p}} \geq 6 \\ &= \frac{1}{0.1 + \frac{0.90}{16}} \\ &= 6.4 \end{aligned}$$

Team B's approach of doubling the cores gives the highest speed-up.

Part D – OpenMP Programming 25 marks

1. **Parallel Sum with Reduction**

   - Initialize array A[i] = i % 100 (or random).
   - Compute a sequential sum (baseline).
   - Compute a parallel sum using #pragma omp parallel for reduction(+:sum).
   - Measure wall time with omp_get_wtime().
   - Run with threads T ∈ 1, 2, 4, 8 (or up to your CPU's max). Record time, speedup, efficiency in results/measurements.csv.

   Refer to `code/omp_assignment_d_q1.c` for the solution to this. Time, speedup, efficiency can be found in `results/measurements.csv`.

2. **Scheduling Experiment**

   - Create a synthetic workload loop with varying work per iteration (e.g., inner loop dependent on i).
   - Compare schedules: static, dynamic, guided (keep chunk sizes reasonable).
   - Report which schedule performs best and why (load balance vs overhead).

   Static Scheduling: This type of scheduling divides the iterations into fixed-size chunks and assigns each thread these chunks. However, if the work per iteration varies significantly, meaning that some threads finish work earlier than others leading to poor load balance. This has low overhead as it determines the iteration distribution at compile-time. The static scheduling type is appropriate when all iterations have the same computational cost.

   Dynamic Scheduling: In this, each thread requests a chunk of iterations when it becomes available until there are no more chunks available. This approach provides better load balancing but has high overhead because it dynamically distributes the iterations at runtime. The dynamic scheduling type is appropriate when the iterations require different computational costs.

   Guided Scheduling: This scheduling is similar to dynamic scheduling. The difference with the dynamic scheduling type is in the size of chunks. It starts with large chunk sizes and gradually reduces the chunk size as the loop progresses. This approach aims to balance the load while minimizing overhead.

The best schedule depends on the specific problem and the characteristics of the workload. In my assumed workload, the work per iteration varies significantly due to the 100%(i + 1) inner loop, making dynamic or guided scheduling potentially more suitable than static scheduling. On my machine (specifics mentioned in system_info.txt file), at fixed thread, guided performas slightly better.

You can find the code in file `code/omp_assignment_d_q2.c` file.