Aditya Dhulipala
EE/CSCI 451, Spring 2015
Homework 1 Submission

1.

1. **Superscalar processor** – A processor which has the ability to execute multiple instructions in the same clock cycle.
2. **Row major layout** – Data layout ordering wherein contiguous blocks of memory are loaded with contiguous rows of the data (matrix) (as opposed to column major layout where contiguous columns are stored)
3. **Cache pollution** – The situation wherein data fetched into the cache is unnecessary for the execution of the program, i.e. cache could've been used to load useful data to hide memory latency instead.
4. **Instruction level parallelism** – Parallelism obtained by executing sequence of low-level instructions simultaneously through the use of hardware & software techniques such as dynamic instruction issue (hardware) or compilers(software).
5. **Cache line** – Unit of data fetched from main memory to cache
6. **Data dependency** – The issue where the result of one particular instruction may be required to execute some subsequent instruction.
7. **Very Long Instruction Word (VLIW) Processor** – Processors that can execute a group of instruction packed into a single long instruction word. (Used to exploit instruction level parallelism by resolving dependencies through software techniques).
8. **Spatial locality** – A computation-centric view of memory access wherein the consecutive words in memory are used by successive instructions
9. **Single instruction multiple data** – A parallel computing architecture where one instruction is executed by all the processing elements on all the data (each processing element works on one piece of the data). This execution is performed in lockstep operation.
10. **Implicit parallelism** – Parallelism inherent in the program because of the nature of operations/computations being performed.

2.

1. For the given symmetric multiprocessing system, effective memory access time is

$$0.8 * 10 + 0.1 * 100 + 0.1 * 400 = 58ns$$

This means the systems takes $58ns$ to fetch 1 word from memory (including cache). Assuming we perform 1 $operation/word$, the peak computation rate is

$$\frac{1}{58} * 10^9 = 17.24MFLOPS$$

2. For the given cache hit ratio in a single processor system, effective memory access time is

$$0.7 * 10 + 0.3 * 100 = 37ns$$

Assuming the system performs 1 $operation/word$ as before, the peak computation rate is

$$\frac{1}{37} * 10^9 = 27.02MFLOPS$$

$\therefore$ The fractional computation is rate is 17.24/27.02

3. 1. Latency of DRAM is 100 cycles, i.e.

$$= 100 * \textit{clock cycle of processor} = 100 * 1ns$$

To execute the instruction in the for-loop once we need to fetch 2 words. One word of matrix A and one of B.

$$\begin{aligned}
\textit{Fetch one word of A} &\rightarrow 100ns \\
\textit{Fetch one word of B} &\rightarrow 100ns \\
\textit{Execute 1 multiply-add operation, i.e. 2 FLOPs} &\rightarrow 1ns \\
\textit{2 FLOPs in } 201ns & \\
\implies \textit{peak achievable performance} &= \frac{2}{201} * 10^9 FLOPS \\
&= 9.95 MFLOPS \\
&\approx 10 MFLOPS
\end{aligned}$$

2. The processor fetches four words in each memory cycle. So, the processor fetches 2 words of matrix A and 2 words of B in one cycle.

$$\begin{aligned}
\textit{Fetch 2 words of A and 2 words of B} &\rightarrow 100ns \\
\textit{Execute 2 multiply-add operation, i.e. 4 FLOPs} &\rightarrow 1ns \\
\textit{4 FLOPs in } 101ns & \\
\implies \textit{peak achievable performance} &= \frac{4}{101} * 10^9 FLOPS \\
&= 39.60 MFLOPS \\
&\approx 40 MFLOPS
\end{aligned}$$

4. The latency of DRAM is $100 * 0.5ns = 50ns$. Assume that the cache is of size $16KB$. This is enough to fit the vector. The processor first reads the vector from DRAM into the cache.
This takes $4K * 50ns = 200\mu s$
We don't need to consider this into the computation rate since this is a one-time operation. Henceforth, the vector data will be loaded directly from the cache.

$$\begin{aligned}
\textit{Fetch 2 words of matrix} &\rightarrow 50ns \\
\textit{Fetch 2 words of vector} &\rightarrow 0.5ns
\end{aligned}$$

The processor issues both the above instruction at the same time since it can issue 4 instructions in each cycle.
This means that the DRAM latency hides the cache latency,
i.e. effective latency to load operands is 50ns.

$$\begin{aligned}
\textit{Execute 2 multiply-add operations, i.e. 4 FLOPs} &\rightarrow 0.5ns \\
\textit{4 FLOPs in } 50.5ns & \\
\implies \textit{peak achievable performance} &= \frac{4}{50.5} * 10^9 FLOPS \\
&= 79.20 MFLOPS \\
&\approx 80 MFLOPS
\end{aligned}$$

5. The latency of DRAM is $100 * 1ns = 100ns$.
The cache is not big enough to fit $4K * 4K$ words.
Since we have two multiply-add units , we need to fetch 2 pairs of words for each computation cycle.
Since the matrices are laid out in row-major order, for each computational step we fetch one row of

matrix A and two rows of matrix B and keep only required column of matrix B. We don't need to fetch the entire rows here; but only 2 words of each row.

$$\begin{aligned} &\textit{Fetch 2 words of } A \rightarrow 100ns \\ &\textit{Fetch 2 words of } B \rightarrow 200ns \end{aligned}$$

Since the processor can issue four instructions in each cycle, both the above requests are issued in the same cycle. Since we're fetching 2 rows of matrix B we pay 200ns in latency cost. This effectively means that the total cost to fetch all required operands in 200ns, i.e. the latency to fetch operands of B hides that of A.

$$\begin{aligned} \textit{Execute 2 multiply-add operations, i.e. } 4 &\rightarrow 1ns \\ 4 \textit{ FLOPs in } 201ns & \\ \implies \textit{peak achievable performance} &= \frac{4}{201} * 10^9 FLOPS \\ &= 19.9 MFLOPS \\ &\approx 20 MFLOPS \end{aligned}$$

6. The processor clock is 1ns. This implies that cache latency is 1ns.
The DRAM latency is 100 cycles, i.e. 100ns.
The processor can execute 4 instructions in each cycle.

The total execution time of the program is equal to:-
time to load w words from DRAM to cache + time to retrieve w words from the cache k times during the execution of the program + computation time of the program.
Since the cache line is 1 word wide, only one word is loaded from DRAM to cache in each memory cycle. $\therefore$ time to load w words is $100w$ ns.

The processor can issue 4 instructions in each cycle. Let's assume that the program is structured that the processor can execute on 4 words of the program in each cycle. This means that the processor fetches 4 words from the cache in each cycle. This happens k times for each word, i.e. data reuse.
$\therefore$ time to fetch data from cache to processor is $\frac{w}{4} * k\ cycles$. This is equal to $wk/4$ ns.

The computation time of the program is given to be n cycles, i.e. n ns.

$\therefore$ total execution time is $100w + (wk/4) + n$ ns.

7. Differences, Advantages/Disadvantages between shared address space and message passing models.

shared address space	**message passing**
There is one big memory bank that all processors read and write.	Each processor has it's own memory space.
processing elements communicate through global variables	processing elements communicate through exchanging *send & receive* messages
Easy to program; difficult to ensure correctness and avoid race conditions	Difficult to implement but offers more control and flexibility to program
Using shared-address it is easy to emulate message passing model by partitioning memory for each processor; processors communicate by writing in each other's partition	Not very easy to emulate shared-address model using message passing because accessing remote memory is very costly