Exam Assignments V08

chengchengguo 183090

1. Explain the **naming conventions** for **intrinsic functions**.

   (_*<vector_size>*_*<operation>*_*<suffix>*)

- *<vector_size>* specifies the size of the returned vector; **mm** stands for 128-bit vectors (_ SSE), **mm256** for 256-bit vectors (_ AVX, AVX2), and **mm512** for 512-bit vectors (_ AVX-512)
- *<operation>* describes the **operation** performed by the intrinsic function, for example: add, sub, mul, etc.
- *<suffix>* identifies the data type of the function's primary arguments (**data type used within the input vectors**); **ps** is for **float**, **pd** for **double**, and **ep***<int_type>* is for **integer** data types: epi32 for signed 32-bit integer, epu16 for unsigned 16-bit integer, etc.

2. What do the metrics **latency** and **throughput tell you about** the **performance** of an **intrinsic function?**

Latency and **Throughput** are two metrics that provide **information** about the **performance of an intrinsic**.
Latency is the **number of cycles** that an intrinsic takes **until its result** is **available (**and can be used for further computations).
Throughput, on the other hand, specifies how many **cycles** it takes to **start the next intrinsic of the same kind**. Here, in each clock cycle we can start two independent **fmadd** operations.

3. How do **modern processors** realize **instruction-level parallelism**?

instruction-level parallelism → several instructions are evaluated simultaneously (on different functional units) (not mix with SIMD. **instruction-level parallelism** means that **several instructions** can be executed **simultaneously on the same CPU core**.)
out of order→ the order in which instructions execute need not correspond to their ordering in the machine-level program
branch prediction → guessing whether or not a branch will be taken
speculative execution → executing operations before knowing whether or not the branch prediction was correct

Instruction Level Parallelism is achieved when multiple operations are performed in single cycle, that is done by either executing them simultaneously or by utilizing gaps between two successive operations that is created due to the latencies.

Now, the decision of when to execute an operation depends largely on the compiler rather than hardware. However, extent of compiler's control depends on type of ILP architecture where information regarding parallelism given by compiler to hardware via program varies. The classification of ILP architectures can be done in the following ways –

1. **Sequential Architecture :**
   Here, program is not expected to explicitly convey any information regarding parallelism to hardware, like superscalar architecture.
2. **Dependence Architectures :**
   Here, program explicitly mentions information regarding dependencies between operations like dataflow architecture.
3. **Independence Architecture :**
   Here, program gives information regarding which operations are independent of each other so that they can be executed instead of the 'nop's.[1]

4. How may **loop unrolling** affect the **execution time** of compiled code?

Loop unrolling is a **loop transformation technique** to **exploit instruction-level parallelism**. In loop unrolling the **loop's body is replicated** and the logic that controls the number of **iterations performed is adjusted**.
But, unrolling **increases compile time**, **increases program size**, needs **additional space** in the **instruction cache**. The unrolled loop body may have a **greater demand for registers** than the original loop body.

5. What does a **high IPC** value (instructions per cycle) mean in terms of the **performance of an algorithm**?

By **dividing the number of instructions by the cycles**, the number of instructions per cycle (**IPC**) is computed.
A **high IPC** indicates efficient CPU utilization (→ **instruction-level parallelism**).

[1] Instruction Level Parallelism - GeeksforGeeks