**Key points of 1.1-1.3:**

1. Von Neumann architecture: a design with an undivided memory that stores both programs and data, and a processing unit that executes the instructions, operating on the data in ‘fetch, execute, store cycle’.
2. How an instruction is processed: Instruction fetch, Instruction decode, Memory fetch, Execution, Write back to memory.
3. Several ways to improve performance of dealing with floating point data: have separate addition and multiplication units to execute independent instructions simultaneously; have a *Fused Multiply-Add(FMA)* unit that execute in the same amount of time as a separate addition or multiplication and combine with pipelining.
4. Pipelining: the addition and multiply units are pipelined, enabling a stream of independent operations to be performed at an asymptotic speed of one result per clock cycle.

Traditional FPU: producing n results takes is clock cycle time, l is the number of stages. The rate at which results are produced is ..

Pipelined FPU: , s is a setup cost. We also denote

.

1. Instruction level Parallism(ILP): execute instruction in parallel that don’t depend on each other.
2. Memory hierarchies (in general, the faster, the smaller):

Main memory: slow but the largest.

Cashes: consist of on-chip L1 and L2, and may have off-chip L3. Finding data in cache is called cache hit, and cache miss otherwise.

Registers: very fast, very expensive.

1. Latency: the delay between the processor issuing a request for a memory item, and the item actually arriving.

Bandwidth: the rate at which data arrives at its destination, after the initial latency is overcome. , where is latency and is inverse of bandwidth.

1. Cache misses: first time reference of date; capacity cache miss; conflict misses where one data is mapped to the same cache location as another.
2. Cache line: the smallest unit of data to move, typically 64/128 bytes long.
3. Cache mapping: the way that main memory address of item is mapped to an address in cache.

Direct mapping: such as takes from each memory address the last 16 bytes and use it as the address in cache.

k-way associate cache: a data item can go to any of k cache locations.

**Questions:**

1. How does an FMA actually work? Why it can execute as efficiently as addition or multiplication.
2. In page 31, it is mentioned that each four iterations of loop involve 8 transfers of elements in a, it seems that bringing the cache line back into memory isn’t counted here. Is it as expensive to bring cache line back to memory as bringing it to cache?

**Exercises:**

1.2. For Linked triads, since it is possible to feed multiplication pipe into addition pipe without data going back to memory first, so , where s is setup costs, l1 is stages needed for multiplication, and l2 is the computation stages in addition(equal to 3), therefore .

1.8. If the mapping is done by using the first 16 bytes as the cache address, then obviously a[0][i], a[1][i] and a[2][i] will be stored in different cache addresses, so there will be no overwriting. Besides, since a cache line holds for four words, only transferring a[0][0] and a[1][0] will be sufficed for the first four loops, so there will be no problem for this program. However, fix i, it maps a[i][j] to same cache address for all j, so problems will be caused for other programs. Such as the one we compute a[0][i]+a[0][i+4] in a loop.

He Lyu

09/03/2018