Computer Architecture - CS 301

Rishit Saiya - 180010027, Assignment - 8

October 31, 2020

1

Temporal Locality
Temporal locality refers to the reuse of specific data, and/or resources, within a relatively small time duration. The processor tends to access memory locations that have been used recently. So the instruction is kept in cache memory such that it can be fetched easily and takes no time in searching for the same instruction. It is generally exploited by keeping recently used instruction and data values in cache memory and by exploiting a cache hierarchy.

Let us consider a stack where, whenever we access an address, we bring it to the top of the stack. The distance from the top till where element was found is defined as the stack distance. It actually counts the frequency of reuse of the addresses, thus its an example of temporal locality. Most stack distances are very low and thus have high temporal locality in general.

Example:
Let's say that in a case, we are recursively updating a variable. Consider the following code snippet:

```
int sum = 0;
    for (int i = 0; i < 10; i++){
        sum += i;
    }
```

In here, we update the variable sum, continuously and hence this demonstrates Spatial Temporal Locality case.

Spatial Locality
Spatial locality (also termed data locality[3]) refers to the use of data elements within relatively close storage locations. The processor tends to execute and involve a number of memory locations that are clustered. So, the data elements (instructions) are used which are relatively close in storage locations. It is generally exploited by using larger cache blocks and

by incorporating prefetching mechanisms (fetching items of anticipated use) into the cache control logic.

Let us consider maintaining a sliding window of the last K memory addresses. The i^{th} address distance as the difference in the memory addresses of the i^{th} memory access, and the closest address in the set of last K memory access. This shows the similarity in addresses. The address distances within $\pm$ 25 bytes are most likely to be accessed frequently and hence they have high spatial locality.

Example:

Now, let us consider a case where we want to read array elements from array **arr**. If we want to read values from arr[i], where i is index, such memory locations are nearby (spaced with 4 gaps of byte). Thus, this demonstrates Spatial Locality case.

2

If the registers are removed from any given architecture, then we are compelled to store the operands of the instructions in the processors' memory or other processing unit's memory. However, we know that Memory involved operations are quite slower than the operations performed by registers. The process of operand fetching the memory and then writing, involves more time and computations.

A processing unit operates on register contents at the rate of more than one operations per cycle but such multiple operations are not possible in case of memory. So, if only memory is used with no availability of caches (which register provide), then the execution time is naturally going to increase.

The registers provide a faster computing interim internal calculations of processing units and if for such calculations, if only memory is involved then the time computation will be verbose for programs as mentioned above.

So, No, such a processor would be very slow if only memory is used as registers are very fast. So such a processor is not recommended.