**Deep Bodra**

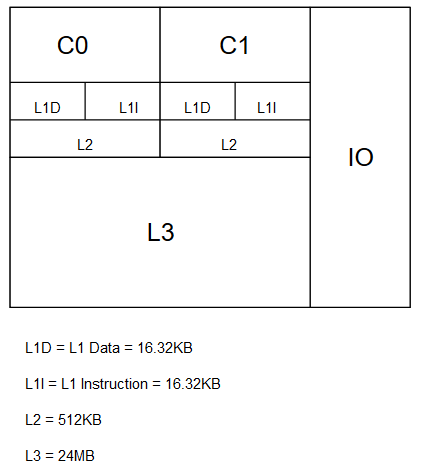
**5801 1841**

**DBI Week 1 (1/13 - 1/17)**

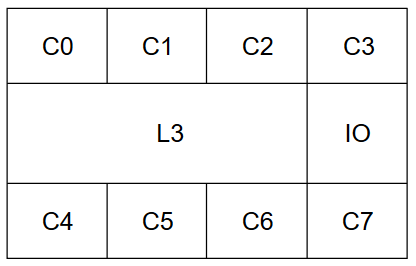
**Detailed explanation of the topics covered in class**

**1/14/2020**

1. Introduction
   1. Moore’s Law: The number of resistors on a dye doubles every 18 months
   2. Heat generated ∝(No of resistors)3
2. The revolution of 2002-03
   1. 2 processors on the same chip
   2. Clock speed: 3.6GHz on workstation
   3. 32 bit and 64 bit architectures were introduced



1. The revolution of 2012-13



* 1. All cores C0-C7 share the same cache memory (L3)
  2. It has an advantage that if a core C0 operates on data D1 that was fetched from the secondary memory then it will be cached in L3
  3. As a result any request for D1 by the other cores will not go to the secondary memory but will be served from L3 instead.
  4. This speeds up the overall memory access time
  5. But this method has the problem of **cache coherency**.
  6. Since the cores are sharing the same memory L3, if data in L3 changes then all the cores must be notified.
  7. If not then the copy of that piece of data in L1 or L2 might become dirty (in simple terms all the cores may have different versions of the same chunk of data)
  8. **This is what makes multi cores slower than a computer with a single core for some programs**
  9. For eg. Merge sort is faster to parallelize than Quick sort

1. Intel vs AMD
   1. Itanium
      1. It is a 64 bit processor launched by Intel in 2000
      2. It was based on the idea that processor optimizations like “Out of Order execution” is the job of a compiler and not the processor
      3. This put a lot of burden on the compilers
      4. Its cost was $20K
   2. Opteron
      1. It is a 64 bit processor launched by AMD in 2005
      2. It was faster and even cheaper than Itanium
2. NUMA architecture
   1. It is collection of NUMA nodes with each node consisting of core and memory
   2. It has 2d nodes where d denotes the number of nodes to which each NUMA node is connected to
   3. The nodes that are connected to each other have a shared memory
   4. The C library “prefres” helps to schedule our process on a desired NUMA node
3. Turbo boost
   1. Instead of using all the cores for processing, all but one cores are shut down and they are used for heat dissipation.
   2. This improves the performance/speed
4. Virtual Memory
   1. Without virtual memory a process can be executed only of there is enough main memory available
   2. Also there is no security mechanism and processes can wander into the address space of other processes.
   3. These are the problems that Virtual Memory avoids
   4. Secondary memory is divided into fixed sized chunks called Pages and the Main memory is divided into fixed sized chunks called Frames
   5. The size of the page and frame is decided by the manufacturer and is generally kept the same.
   6. Page table is stored in the main memory which maps the virtual address to the physical address
   7. CPU generates virtual address and the page table is accessed to get the corresponding physical address
   8. If the physical address is invalid then the page will be loaded from the secondary memory
   9. There is an additional caching mechanism used called the Translation Lookaside Buffer which caches the page table entries for faster access
   10. This is flushed at every context switch

**1/16/2020**

1. Large Pages

|  |  |  |
| --- | --- | --- |
| Main memory size | Page size | # of pages |
| 1 TB (230 B) | 4KB (212 B) | 218 |
| 1 TB (230 B) | 2MB | 512 times less pages |

|  |
| --- |
| Always Unmapped page |
| CODE |
| STATIC DATA |
| HEAP |
| Unmapped area |
| STACK |

* 1. The 1st page is always left unmapped to catch poorly typed code like dereferencing a NULL pointer
  2. CODE: Executable code
  3. STATIC DATA: Variables
  4. HEAP: grows downwards to higher addresses
  5. STACK: grows upwards to lower addresses
  6. Unmapped area: This area can be used either by HEAP/STACK or by memory mapped functions (mmap/munmap)

1. mmap and munmap
   1. mmap
      1. It creates a mapping in the virtual address space of the calling process
      2. This function can be used
         1. To increase the page size by asking it to allocate memory in the multiples of 2. It may not always succeed
         2. To allocate a chunk of shared memory between processes for faster memory access
   2. munmap
      1. It deallocates the memory that was allocated by mmap

----------------------------------------------------------------------------------------------------------------------------------------

Did you know?

strace: Can be used to find the system calls used by your program.

----------------------------------------------------------------------------------------------------------------------------------------

--THANK YOU--