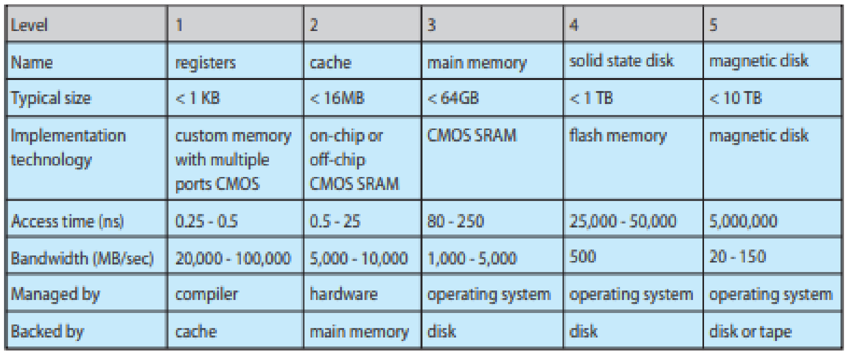
Susanna Kyler REU - Feb 8, 2018

CPU Architecture

Memory hierarchy

- The key to proper memory use is *locality.* Good locality is shown when programs “access the same set of data items over and over again”(aka Temporal locality) or primarily “access sets of nearby data”, also called Spacial locality (pg 531, link).
- Below is a graphic which I believe displays the hierarchy more effectively than listing out the same information from fastest to slowest (pg 28, link) →



Level	1	2	3	4	5
Name	registers	cache	main memory	solid state disk	magnetic disk
Typical size	< 1 KB	< 16MB	< 64GB	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS SRAM	flash memory	magnetic disk
Access time (ns)	0.25 - 0.5	0.5 - 25	80 - 250	25,000 - 50,000	5,000,000
Bandwidth (MB/sec)	20,000 - 100,000	5,000 - 10,000	1,000 - 5,000	500	20 - 150
Managed by	compiler	hardware	operating system	operating system	operating system
Backed by	cache	main memory	disk	disk	disk or tape

- There is volatile and nonvolatile storage. Volatile means the content is lost when the power is removed. Permanent storage requires in nonvolatile storage.

Multiple Cores

- The term multicore (also called parallel systems, or multiprocessor systems) means there are “two or more processors in close communication, sharing the computer bus and sometimes the clock, memory, and peripheral devices” (pg 14, link)
- Multicores have 3 advantages: increased speed, lower cost - by sharing processes and data instead of storing data on multiple disks, and increased reliability.
- There are two types of multicores, *asymmetric multiprocessing*(ASMP) and *symmetric multiprocessing* (SMP). ASMP has each processor doing a specific task and another processor controls the system. In SMP each processor performs all tasks, each has their own registers and cache, but all share physical memory.

Fine Grain Parallelism:

- Granularity is the task size. *Fine Grain Parallelism* means doing a large number of tasks. SIMDs (single instruction multiple data) are a type of parallel computers that “perform the same operation on multiple data points simultaneously” (link). *FMA* is a fused multiply add instruction.

GPU Architecture

Memory hierarchy

- Global Memory is the “main” memory of the GPU and has a global scope and lasts for the duration of the host allocation (link). “Global memory is separate hardware from the GPU” (slide 11, link). This the slowest from of IO. Shared Memory is the fastest and located in the single streaming multiprocessor. Its scope is the block.

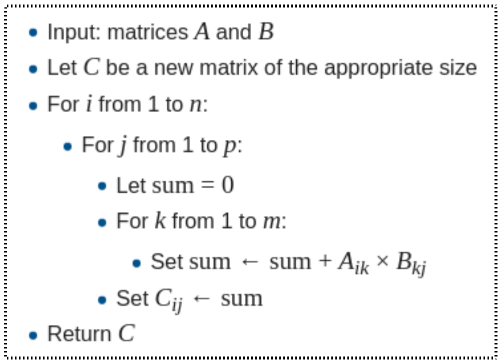
Coarse grain processor

- *Coarse grain processors* have data communicated infrequently. The GPU's Global Scheduler manage coarse grain parallels at the thread block level. This reads info/ "issues thread blocks" to the *streaming multiprocessor* (SM), also called nodes. *CUDA* (Compute Unified Device Architecture) is "a parallel computing platform and programming model" (slide 5, link).

Algorithms

Matrix multiplication algorithms are used in multicore systems. It takes time n^3 to multiply n*n matrices. The fastest algorithm for matrix multiplication is unknown.

The Iterative Algorithm: matrix A= (n*m), matrix B=(m*p) : C = AB = size(n*p)



```
• Input: matrices $A$ and $B$
• Let $C$ be a new matrix of the appropriate size
• For $i$ from 1 to $n$:
    • For $j$ from 1 to $p$:
        • Let sum = 0
        • For $k$ from 1 to $m$:
            • Set sum ← sum + $A_{ik} \times B_{kj}$
        • Set $C_{ij}$ ← sum
• Return $C$
```

- If we assume the inputs are square matrices size (n*n) then the run time   is order of (n^3).

- Cache Behavior: order of the for loops in the algorithm should be swapped depending on what you want to do because they change performance based on their memory access patterns / cache use.

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$$

The *Divide and Conquer Algorithm* uses "block partitioning" for matrices shaped $2^n * 2^n$. And its complexity is $T(n) = 8T(n/2) + \Theta(n^2)$.

Non Square Matrices split the matrix into two equal parts. "The recursive algorithm is cache oblivious". Cache misses are bound by :

$$\Theta\left(m + n + p + \frac{mn + np + mp}{b} + \frac{mnp}{b\sqrt{M}}\right)$$

eel computer :

CPU:

# of processors	: 12
model name	: Intel(R) Xeon(R) CPU E5-1650 v4 @ 3.60GHz
size	: 2842MHz
cache size	: 15360 KB
cache_alignment	: 64
address sizes	: 46 bits physical, 48 bits virtual
Each processor has 6 cpu cores	
memory size	: 15 Gi B

GPU:

NVIDIA Corporation GP106	
width	: 64 bits
clock	: 33MHz
capabilities	: vga_controller bus_master cap_list rom
Resources:	irq:39 memory:f2000000-f2ffffff memory:d0000000-dfffffff

memory: e0000000-e1ffffff ioport:1000(size=128) memory:f3080000-f30fffff
CUDA Version 8.0.61