**Parallel Programming Skills**

* **Identifying the components on the raspberry PI B+ (5pt).**
  + Looking at the PI with birds-eye view and the USB ports facing to the right
    - CPU/RAM at the bottom of the GPIO pins
    - Ethernet controller to the left of the USB ports
    - USB ports to the top of ethernet port
    - Ethernet port to the right of audio port
    - Audio port to the left of camera ribbon cable connector
    - Camera connector to the right of HDMI port
    - HDMI port to the right of power supply port
    - Display ribbon connector to the left of CPU/RAM and top of power port
* **How many cores does the Raspberry Pi’s B+ CPU have (5pt)?**

Raspberry PI 3 module B+ has a Quad-Core CPU

* **List three main differences between X86 (CISC) and ARM Raspberry PI (RISC). Justify you answer and use your own words (do not copy and paste) (8pt).**
  + X86 is a CISC (Complex Instruction Set Computing) processor while ARM is a RISC (reduced Instruction Set Computing. The biggest difference between the two is the instruction sets. CISC processors have bigger instruction sets with more features (while RISC has 100 or less instructions), and they give memory access to complex instructions. Because of this, x84 processors has more operations and addressing mode but less registers than ARM.
  + X86 processors uses the little-endian format (storing data from right to left). The old version (before v3) of ARM processors also uses little-endian, but as new versions roll out, it switched to big-endian (storing data from left to right) with a switchable endianness feature.
  + ARM processors use instructions that can only operate on registers (while X86 can use both register and memory) and uses the load/store model to access the memory. For example, if we want to increment a value on a specific memory address on ARM, we first need to load it to the register, increment it, then store it back to the memory from the register. In X86, we can directly access the memory and increment the value since instructions can both operate on registers and memory.
* **What is the difference between sequential and parallel computation and identify the practical significance of each (6pt)?**
  + Sequential computation executes programs on a single processor at a time while a parallel computation executes programs on multiple processors at a time
  + In sequential computing, programs are broken into series of instructions that will be sequentially executed one after another by the CPU with only one instruction being executed at any given time.
  + In parallel computing, programs are broken into parts of instructions that will then be broken down into series of instructions that each different processor will then execute in coordination with each other.
* **Identify the basic form of data and task parallelism in computational problems (5pt).** 
  + Data Parallelism
    - Computation is applied to multiple data items
    - Amount of parallelism is proportional to input size (causes huge amount on potential parallelism
    - Gives programmers flexibility in writing scalable (with input size) parallelism programs in a way that the program should be using all the available parallelism.
  + Task Parallelism
    - Parallelism is organized around tasks instead of data.
    - Work should be balanced, and all the work should contribute to the result in some way or form
    - Task parallelism does not scale well as data parallelism
* **Explain the differences between processes and threads (6pt).**
  + Processes:
    - Abstraction of a running program
    - Does not share memory with each other
    - Single-core CPU operates on one process at a time while multi-core CPU can operate on more processes at a time (concurrency)
  + Threads:
    - Lightweight process allowing a single executable to be decomposed to smaller independent parts
    - All thread shares common memory of the process they belong to
    - The operating system will schedule threads on separate available cores/CPU
* **What is OpenMP and what is OpenMP pragmas (3pt)?** 
  + OpenMP
    - Industry standard since the end of the 1990s with native support with GCC compilers
    - Uses implicit multithreading model where the library creates and manages threads
    - Makes programmer’s task simpler and less error-prone
    - Uses Thread Pool pattern to concurrent execution control
    - Initializes group of threads (pool of threads) for programs, and the threads will execute concurrently during portion of the code when told so by the programmer
  + OpenMP pragmas
    - An alternative to pthreads (a low-level thread package) when writing programs for shared-memory, multicore hardware
    - Compiler directives that allows the compiler to generate threaded code instead of having the programmer generating and managing it (like in pthread).
    - Uses fork/join and single program, multiple data (two primary patterns used as program structure implementation strategies.
* **What applications benefit from multi-core (list four) (4pt)?** 
  + Database and web servers
  + Compilers
  + Multimedia and scientific applications
  + Applications with thread-level parallelism
* **Why Multicore (why not single core, list four) (4pt)?**
  + Hard to increase single-core clock frequencies
  + Deeply pipelined circuits
    - Heat and speed of light problems
    - Difficult design and verification wit large design team needed
    - Server farms need expensive AC units
  + Most new applications use multithreading
  + Trending in computer architecture (we are shifting towards parallelism)