Distributed and Parallel Computing

Lecture 2

Alan P. Sexton

University of Birmingham

Autumn 2019

Measuring speed

Most CPU clocks can not get better than 10ms accuracy

- At 3GHz, 10ms = 30,000,000 cycles, 1 to 5 cycles (approx) per instruction, so 10ms is maybe 10,000,000 instructions
- Code which really takes 15ms, reported as either 10ms or 20ms - large inaccuracy
  - e.g. 10ms $\pm$ 10ms
- Instead run it 100 times: 1,500ms, reported as one of 1,490ms, 1,500 or 1510
  - e.g. 1,500ms $\pm$ 10ms for 100 iterations $\Rightarrow$ 14.9ms $\pm$ 0.1ms
- BUT... lots going on in the computer
  - `ps -ef`
- The longer a piece of code runs, the higher the probabilities of resource contention, context switching etc., which interferes with accurate speed measurement
- **DO NOT INCLUDE ANY I/O IN YOUR TIMINGS** (unless specifically timing I/O)
- The first time you run a piece of code, the timings might be significantly out of line with following re-reruns. Why?

Posix Threads - pthreads

A language independent parallel execution model (but currently only implemented in C)

- Thread management
- Mutexes
  - Mutually exclusive lock on a variable
- Condition variables
  - Lock until a condition becomes true
- Read/Write Locks
  - Allows multiple readers OR a single writer
- Barriers
  - Threads which hit the barrier (code location) have to wait until all threads in the group get to the barrier
- Docs: Lots of tutorials online, Linux man pages: e.g. `man pthreads`, `man pthread_create`, etc.

Basics of pthreads

- `pthread_attr_init()` gets default thread attributes and stores them in a `pthread_attr_t` struct (e.g. stack size, and "detach state")
- `pthread_attr_setdetachstate()` sets the detach state attribute value, which can be *joinable* or *detached*, in a `pthread_attr_t` struct
- `pthread_create()` creates a new thread with the attributes defined in the given `pthread_attr_t` struct, that starts running on the function specified with the argument specified and stores the thread id in the specified variable
- `pthread_exit()` terminates the current thread. If the thread is joinable, its return value parameter is available to any other thread that calls `pthread_join()` on this thread id.
- `pthread_join()` waits for the specified joinable thread to terminate and gets the return value from it.

Switch to nsight and view code

Performance of imflipP

Intel Core i5-4210M, 2 cores, 4 threads

# Threads	H	V
1	4.8	5.3
2	2.7	2.9
3	3.5	3.8
4	2.9	3.1
5	3.1	3.4

DRAM Access Patterns

Rule	Ideal Values	Description
Granularity	8-64 B	Size transferred in a single read/write. Reading small sizes is very inefficient
Locality	1-4 KB	If consecutive accesses are too far from each other, they force the row buffer to be flushed, triggering a new DRAM read
L1, L2 Caching	64-256 KB	If the total set of bytes read/written repeatedly is within a small region, then they will stay in the cache so accesses after the first will be MUCH faster for the same thread
L3 Caching	8-20 MB	If the total set of bytes read/written repeatedly is within a small region, then they will stay in the cache so accesses after the first will be MUCH faster for ALL the cores

adapted from [Soyata]

Performance of imflipP and imflipPM

Intel Core i5-4210M, 2 cores, 4 threads

# Threads	H	V	I(H)	W(V)
1	4.8	5.3	2.9	0.20
2	2.7	2.9	1.7	0.14
3	3.5	3.8	1.9	0.18
4	2.9	3.1	1.7	0.17
5	3.1	3.4	1.8	0.22

SRAM recommendations

To take advantage of caching, write your programs so that:

- Each thread access 32KB data regions repetitively
- Try to confine broader access to 256KB if possible
- Try to confine cumulative data accesses with L3$
- If you must exceed this, make sure that there is heavy usage of L3$ before exceeding beyond this region

[Soyata]