CS3210

Parallel Computing

Changes from Monday in Green

Lab 4

Mon (4pm)
Tues (2pm)

Admin
Updates

- Assignment 2 due yesterday morning, 11am
  - Late penalty: 5% per day
  - If you have submitted, download your submission and check its contents are correct

- Mid-term and Lab 2 grades released on LumiNUS
  - Mid-term comments available on **`softmark.io`** (check your NUS email for the link to the marked script)
  - If you have questions/want to dispute the grading, write in to Prof via email ASAP - mid-term grades finalised after 22 Oct

Admin
Roadmap

- Please use only the machines assigned to your lab pair
  - Use both assigned machines with your MPI programs

- Today's lab
  - Part 1: Compiling and Running MPI Programs
  - Part 2: Mapping MPI Processes to Processors
  - Part 3: Inter-process Message-Passing

Admin

Lab 4 Submission Instructions

- Due this Friday (23 Oct), 11.59pm
  - Each student submits their own PDF writeup (indep. work)
  - Name it A0123456X.pdf
  - No need to include your programs in writeup
  - List the hostnames of the machines you used

- Contents
  - Responses to ex6, ex9 and ex10 (max 3 paragraphs total)
  - Relevant performance measurements for ex6 with your observations and a short analysis

Introduction

Message-Passing Interface

- Message-Passing Interface (MPI) is a message-passing library standard
  - MPI is a specification for how implementing libraries should *behave*
  - Actual MPI implementations: OpenMPI, MPICH, etc.

- Lab machines have OpenMPI v2.1.1 installed
  - Conforms fully to the MPI-3 standard (released 2012)
  - Documentation: **https://www.open-mpi.org/doc/v2.1/**
  - For following labs and Assignment 3, please test your programs on the lab machines

Part 1

MPI Programs

- Comprises multiple processes cooperating via MPI calls
  - Each process identified by its **rank** (non-negative integer)
  - Use rank to specify sender/recipient of messages or control what each process executes

- Processes belong to communicators
  - A **set** of processes (**MPI_Group**) with an associated context
  - By default, all processes in global communicator **MPI_COMM_WORLD**
  - Can split (see **MPI_Comm_split**) or create new ones (see **MPI_Comm_create**) for targeted communication

Part 2

Specifying Host Nodes

- Terminology
  - Remote node/host: another machine on the network
  - Slot: a reserved spot for an MPI process on a node (does not necessarily correlate to a CPU core!)

- To specify hosts to launch processes on with `mpirun`
  - Explicitly: use **`-H`** flag followed by comma-delimited list of hostnames, repeat to use multiple slots on that host
  - More scalable: use **`--hostfile`** (or **`--machinefile`**) flag with name of hostfile
  - Each line: **`<hostname>( slots=<number to use>)?`**

Part 2

MPI Process Mapping

- By default, 1 slot accepts only 1 MPI process
  - Run with more processes than slots: **`--oversubscribe`**
  - A slot is not bound to any CPU core - process can migrate

- By default, MPI exhausts all slots on a host before proceeding to the next host in the hostfile
  - Use **`--map-by <option>`** to change mapping policy
  - If hostfile does not specify slots on a node, MPI assumes it is the number of physical cores on that node
  - Use **`--map-by node`** to assign processes to slots from different hosts in a round-robin (load balancing) fashion

Part 2

MPI Process Ranking

- By default, rank is assigned by socket (similar to slot)
  - Use **`--rank-by <option>`** to change ranking policy
  - More granular control: provide rankfile with **`--rankfile`**
  - Rankfiles: text files specifying which host the i-th ranked process executes on, and which logical core(s) they should be bound to
  - Each line of format:
  - **`rank <i>=<host> slot=(<socket>:<core list>)+`**
  - e.g. **`rank 0=xgpc5 slot=0:0,0:2-3,0:5,0:7`**
  - **`<core list>`** here refers to physical core indexes - can correspond to more than one logical core if CPU has SMT

Part 2

MPI Process Binding

- MPI can bind a process to a set of logical cores with an affinity mask
  - Binding: controls which CPU cores the process can execute on (vs. mapping - which is where the process starts)
  - By default, if # of processes $N \leq 2$, MPI binds to cores
  - By default, if $N > 2$ and not oversubscribed, MPI binds to the socket (all cores on it), else MPI does not bind
  - Use **`--bind-to <option>`** to change binding policy
  - Affinity mask: i-th LSB = 1 if process bound to i-th logical core (0-indexed)
  - Use **`lstopo -p`** (list topology) to see logical core layout

Part 3

Message-Passing Semantics

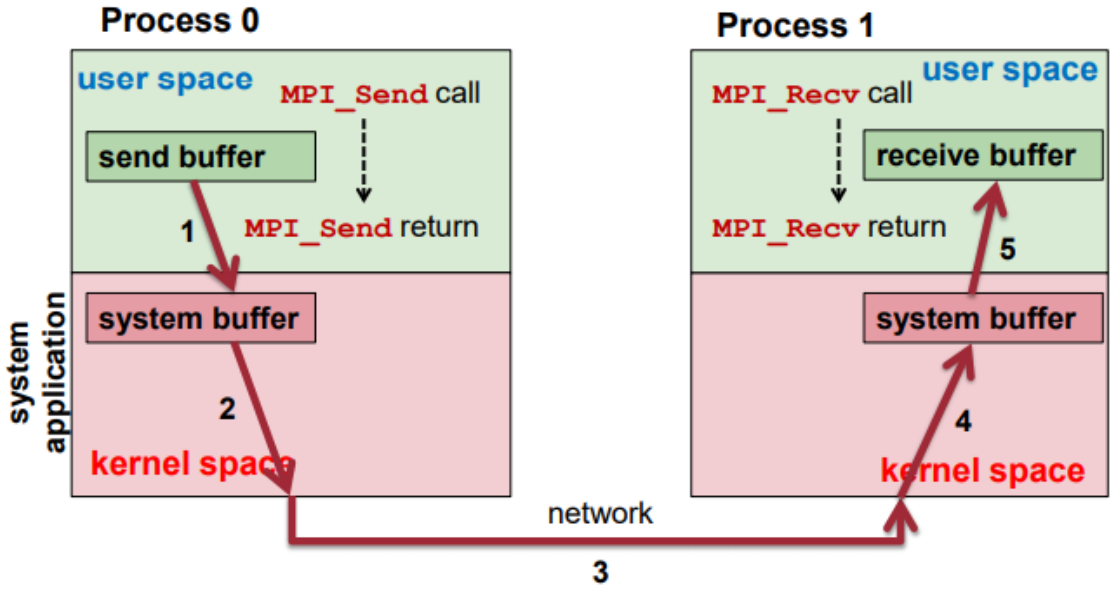
- Local view (blocking?) vs. global view (synchronous?)
  - Can mix blocking and non-blocking operations
  - Incorrect use can lead to deadlock
  - More details: Lecture 9 on Message-Passing Programming

	Synchronous	Asynchronous
Blocking	`MPI_SSend` `MP_SRecv`	`MPI_Send` `MPI_Recv`
Non-blocking	`MPI_ISSend` `MP_ISRecv`	`MPI_ISend` `MP_IRecv`

Part 3

Message-Passing Buffers

- MPI runtime may use additional system buffers
  - MPI program is **secure** if correctness of program does not depend on assumptions about properties of MPI runtime



Part 3

Message Ordering

- Message ordering (delivery) guarantees between a particular sender S and receiver R
  - Messages from S to R delivered in FIFO order
- No guarantees between messages arriving from different senders (or messages to different receivers)
- Interested to know more?
  - Covered in CS4231: Parallel and Distributed Algorithms

CS3210

Parallel Computing

Thank you! Any questions?

Lab 4

Mon (4pm)
Tues (2pm)

bit.ly/cs3210-t04-qn