Tutorial 1
Parallel Computer Architecture

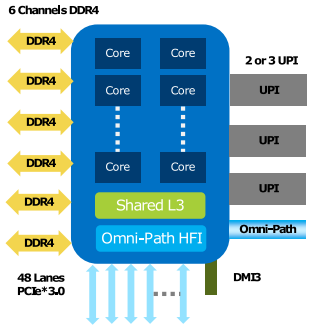
CS3210 – 20/21 Semester 1

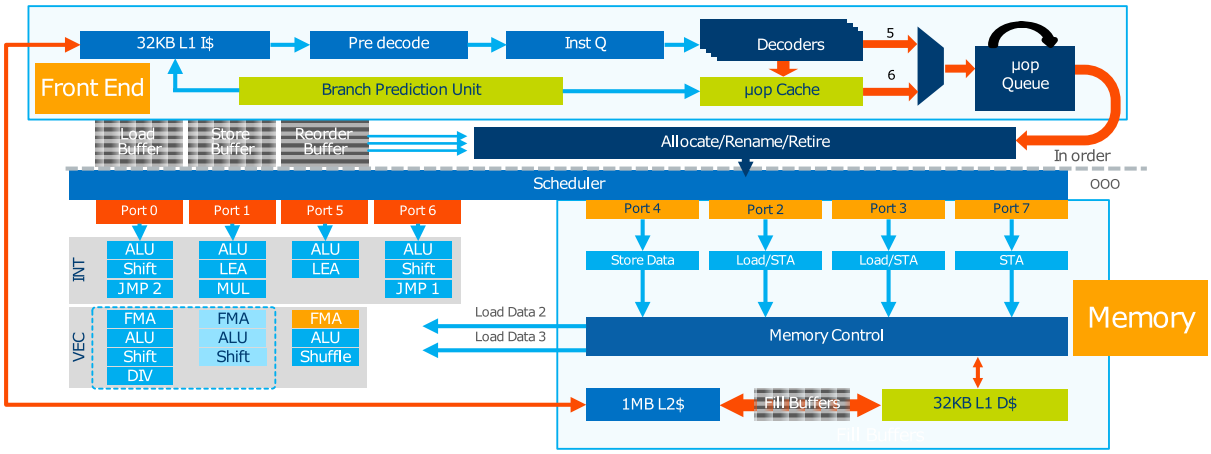
Learning Outcomes

1. Recall the various levels of parallelism
2. Understand and apply the Flynn's taxonomy of computer architectures
3. Understand the various ways memory can be organized
4. Set-up and run basic parallel applications on a compute cluster

Part 1: Tutorial on Parallel Computing Architectures

1. **(Levels of Parallelism)** A simplified diagram of the Intel Xeon Scalable processor family and its micro-architecture diagram is reproduced below





Briefly discuss and identify the various forms of parallelism that are supported by this processor chip.

Useful information on Intel Xeon Scalable

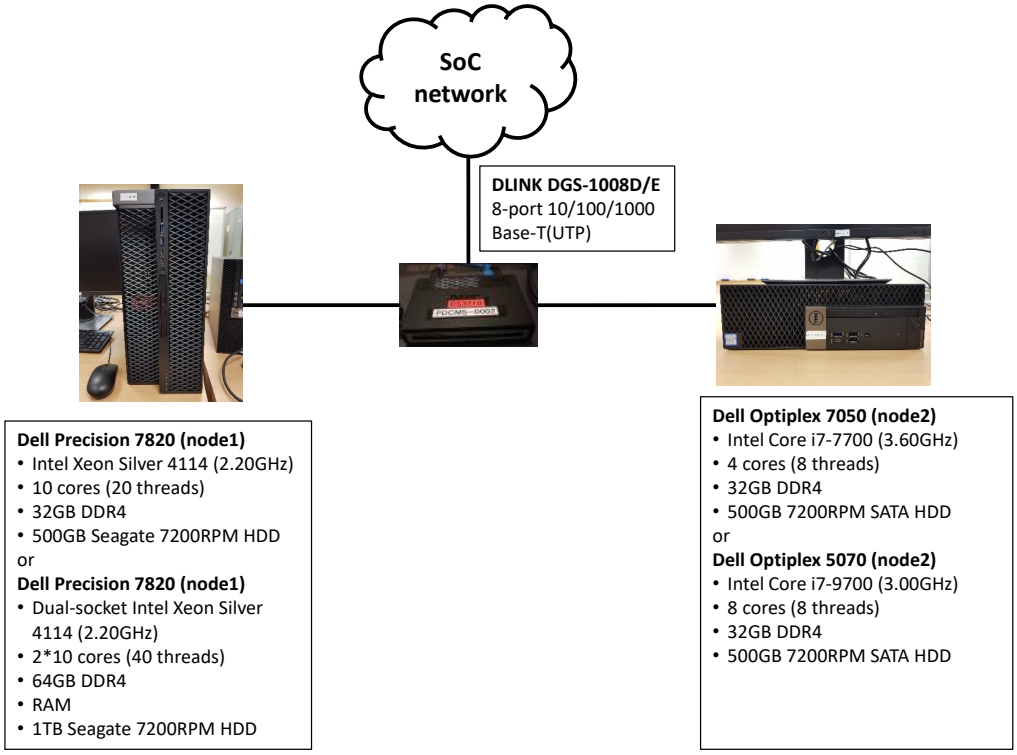
- `https://software.intel.com/en-us/articles/intel-xeon-processor-scalable-family-technical-overview`
- `https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/xeon-scalable-platform-brief.pdf` (page 9)

2. **(Flynn Taxonomy)** For each of the following scenarios, classify them according to Flynn's taxonomy (i.e. SISD, SIMD, MISD, MIMD) and provide a brief (say, one line) justification
   (a) A personal computer from the 1980s
   (b) A laptop with a multi-core processor
   (c) Intel's AVX instruction set
   (d) A uniprocessor (i.e. single core) with pipelining (ILP)
   (e) Students attempting the same exam in an exam hall
3. **(Memory + Multi-core Architecture)** For each of the following questions, indicate whether the statement is true or false. Briefly justify your answer.
   (a) Shared-memory system implies a UMA architecture.
   (b) On a NUMA architecture, the actual location of data (i.e. on which node) has no impact for a distributed shared-memory program.
   (c) In the hierarchical design of multicore architecture, the memory organization is hybrid (distributed-shared memory)
4. **(Processes and Threads)** For each of the following questions, indicate whether the statement is true or false. Briefly justify your answer.
   (a) A semaphore can be used to replace a mutex without affecting the correctness of a program.
   (b) Implementing an algorithm using multiple threads **always** runs faster than implementing the same algorithm using multiple processes.

Part 2: Laboratory Assignment

In this section, we learn how to setup and connect multiple computers into a cluster. This would allow us to run parallel programs/applications on multiple nodes/machines effortlessly.

Workbench Cluster Diagram



The diagram above shows the arrangement of the 2 machines in our workbench parallel cluster. The nodes are designated as follows:

Node	Type
Node 1	Xeon Silver 4114 or Dual-socket Xeon Silver 4114
Node 2	Intel Core i7-7700 or Intel Core i7-9700

Step 1: Identification of Nodes

To start configuring the machines in the cluster, the nodes need to be identified first. The hostname and the type of node are shown below.

- `soctf-pdc-001` - `soctf-pdc-008`: Xeon Silver 4114
- `soctf-pdc-009` - `soctf-pdc-016`: Intel Core i7-7700
- `soctf-pdc-018` - `soctf-pdc-019`: Dual-socket Xeon Silver 4114
- `soctf-pdc-020` - `soctf-pdc-021`: Intel Core i7-9700

Each pair of students will be assigned two nodes to use during this lab. Write the hostname in the corresponding blanks below (or take a note of them):

Node	Type	Hostname
Node 1	Xeon Silver 4114 or Dual-socket Xeon Silver 4114	
Node 2	Intel Core i7-7700 or Intel Core i7-9700	

Step 2: Configuring SSH Access

To allow two machines to work as a cluster (access from from one to another), we need to configure SSH key-based authentication. This allows a node to connect to another node without the use of passwords.

Check network connectivity

Firstly, we have to ensure that both machines are reachable over the network. We can use the **ping** command to check their "visibility".

- On **node1**, run the following command:

  `$ ping -c 5 <hostname of node2>`
- Similarly, on **node2**, we can run the following command:

  `$ ping -c 5 <hostname of node1>`

Generating SSH key pairs

Ensure that you're logged into the compute nodes under your own user ID!

Next, we generate the SSH authentication keys, which be used in place of your system password when connecting (through SSH) to another node in the cluster.

First, we generate a SSH public/private key pair:
`$ ssh-keygen`
(Press Enter when prompted for the directory & password)

This command generates two files: `id_rsa` and `id_rsa.pub` in the ~/.ssh folder. `id_rsa` is the private key and `id_rsa.pub` the public key.

Now we need to ensure that the other node knows which key is authorized to login to the machine. Hence, we need to copy the **public key** to the other node.

- If you are currently utilising **node 1**, run the following:

  `$ ssh-copy-id <your user ID>@<hostname of node 2>`
- If you are currently utilising **node 2**, run the following:

  `$ ssh-copy-id <your user ID>@<hostname of node 1>`
- **Example:** `$ ssh-copy-id 1111Z@soctf-pdc-999`

You be prompted for your user password. `ssh-copy-id` logs into the other host, copies the public keys to the other host, and configures them to grant access by adding them to the `~/.ssh/authorized_keys` file. Only the public key is copied to other machines. The private key should never be copied to another machine.

Verifying password-less access

After that's done, we need to verify that the other node can be accessed without a password from the node that hosts the private key. A simple way to check is to SSH into the machine:

```
$ ssh <your user ID>@<the other node's hostname>
```

If everything is successfully set up, you should not be prompted for the password. Exit the remote SSH session by running the `exit` command.

Given that both nodes have the same user ID and almost similar hostnames, refrain from (routinely) SSH-ing to the other nodes, as you might accidentally perform your work from another machine, miss files, and interfere with others work.

Note that you can access the lab machines from Sunfire. You can connect through SSH to Sunfire from home (`ssh your_soc_account@sunfire.comp.nus.edu.sg`), and you can SSH into any of the lab machines from Sunfire (provided that the lab machine is powered on and booted in Ubuntu). This means that you can access the lab machines from home.

When accessing the lab machines from home, check what others user are connected and running their programs. Execution of other programs will interfere with the performance that you observe for your own programs.

Step 3: Testing cluster with an MPI program

In this step, we run a simple MPI program on the new cluster. Download `hello.c` program from LumiNUS and save it to a **new folder T1** in your home directory (not on Desktop).

First, compile `hello.c` on your current machine (in T1 folder):

```
$ mpicc hello.c -o hello
```

Next, copy the binary to the other node, in a newly created folder T1:

```
$ ssh <user-id>@<other node hostname> 'mkdir -p T1'
$ scp hello <user-id>@<other node hostname>:~/T1
```

To run the MPI program on the cluster, MPI needs to know which nodes it needs to run on. This is specified in a file called the `hostfile`. Create a file named `hostfile` in the same directory.

In the hostfile, add the list of hostnames of nodes, like so:

```
<hostname of node 1>
<hostname of node 2>
```

Execute the parallel program on your machine like so:

```
$ mpirun -hostfile hostfile -np 28 hello
```

The output of `hello` should appear like so:

```
Hello world from process 0 of 28 running on <node 1>
...
Hello world from process 27 of 28 running on <node 2>
```

You can vary the number of independent processes running the parallel program by modifying the number supplied to the `np` argument.

Congratulations, you have successfully set up a working parallel system!