Programming Machine Learning Algorithms for HPC

- Fundamentals of HPC and SLURM

Dr. Pierrick Pochelu and Dr. Oscar J. Castro-Lopez

Fundamentals of HPC

Introduction

HPC

- High-Performance Computing: A field of computing focused on aggregating computational power to solve large, complex problems much faster than standard computing.

- HPC systems often consist of clusters or supercomputers that perform parallel processing, making them essential for tasks like scientific simulations, data analysis, and machine learning.

- In parallel computing, parallelization divides a task into smaller parts to be processed simultaneously, often to improve speed and efficiency. Here are the most common types of parallelization

Data Parallelism

- **Description**: In data parallelism, the same operation is applied to different subsets of the data simultaneously. Each processor handles a portion of the data independently.
- **Example**: Image processing, where each part of the image is processed by a separate processor.
- **Best For**: Applications that perform the same operation on a large data set, like matrix computations or batch data processing in machine learning.

Task Parallelism

- Description: In task parallelism, different tasks or functions are executed in parallel, often on separate processors or threads. Each task can perform a unique operation on the same or different data.
- Example: A web server handling multiple client requests simultaneously or processing various stages of a pipeline concurrently.
- Best For: Applications with distinct tasks that can be done in parallel, like different stages of an algorithm or independent services in microservices architecture.

Pipeline Parallelism (Pipeline Processing)

- Description: Pipeline parallelism divides a task into stages, where each stage is processed in sequence, and each stage is handled by a different processor or thread. As soon as one stage is completed, the data is passed to the next stage.
- Example: Assembly lines in manufacturing or stages of data processing in deep learning (e.g., one stage handles data loading, another handles processing, and a third stage performs training).
- Best For: Applications with sequential stages that can overlap, such as data processing pipelines or streaming data applications.

Hybrid Parallelism

- **Description**: Hybrid parallelism combines multiple types of parallelism (often data and task parallelism) to maximize efficiency. This approach is used to leverage the strengths of different types of parallelization within a single application.
- **Example**: Scientific computing applications, where different parts of a problem (e.g., computation-heavy tasks and data-heavy tasks) are parallelized differently.
- **Best For**: Large-scale applications that require both data and task parallelism, especially on clusters or supercomputers.

Bit-level Parallelism

- **Description**: In bit-level parallelism, the hardware processes multiple bits of data simultaneously. It's primarily a hardware-level approach, where the CPU processes data in larger chunks (e.g., 32-bit, 64-bit).
- **Example**: Arithmetic operations on large integers or signal processing in digital circuits.
- **Best For**: Low-level hardware optimization to improve processing speed.

Loop Parallelism

- **Description**: Loop parallelism focuses on parallelizing loops within a program, where iterations are executed in parallel rather than sequentially.
- **Example**: Running a loop that performs a similar operation on each element of an array, where each iteration can be assigned to a different processor.
- **Best For**: Computational loops in numerical simulations, statistical calculations, or vectorized operations in scientific applications.

Domain Decomposition (Spatial Parallelism)

- **Description**: The problem's domain (e.g., physical space in simulations) is divided into smaller sub-domains, and each processor works on a sub-domain. Often used in simulations that rely on spatial divisions.
- **Example**: Computational fluid dynamics simulations, where each processor handles a section of the physical simulation space.
- **Best For**: Applications like physics simulations, weather forecasting, and other scientific computing tasks that can be broken down into independent regions.

SLURM

Basic usage

SLURM

- **Slurm** (Simple Linux Utility for Resource Management): An open-source workload manager used to schedule and manage jobs on HPC clusters. It allocates resources to users, queues job requests, and ensures efficient resource utilization, handling job priorities, dependencies, and distributing tasks across available computing nodes.

SLURM

Main tasks of SLURM:

1. **Cluster Structure**
2. **Job Submission**
3. **Resource Allocation and Scheduling**
4. **Job Execution**
5. **Job Monitoring and Management**
6. **Completion and Cleanup**

Allocating a job

Allocating a batch job **sbatch**:

- sbatch is used to submit a job script for later execution. The script will typically contain one or more srun commands to launch parallel tasks.

Allocating an interactive real-time job **salloc:**

- salloc is used to allocate resources for a job in real time. Typically this is used to allocate resources and spawn a shell. The shell is then used to execute srun commands to launch parallel tasks.

Useful commands

- **squeue** reports the state of jobs or job steps. It has a wide variety of filtering, sorting, and formatting options. By default, it reports the running jobs in priority order and then the pending jobs in priority order.
- **sstat** is used to get information about the resources utilized by a running job or job step.
- **scancel** is used to cancel a pending or running job or job step. It can also be used to send an arbitrary signal to all processes associated with a running job or job step.

Example of batch job

```
#!/bin/bash -l
#SBATCH -J my_batch_job                 # Job name
#SBATCH -N 1                            # Number of nodes
#SBATCH --ntasks-per-node=1             # Number of tasks per node
#SBATCH --output=aipaca_test_4.txt      # Name of output file
#SBATCH -c 7                            # Cores assigned to each tasks
#SBATCH --time=0-48:00:00               # Time to execute the script
#SBATCH -p gpu                          # Partition
#SBATCH --mail-user your.email@uni.lu   # Mail to send notifications
#SBATCH --mail-type BEGIN,END,FAIL      # Type of notifications to send

# Load modulues
module load lib/UCX/1.9.0-GCCcore-10.2.0-CUDA-11.1.1

# Load virtual environment
conda activate tcc

# Run your code
python large_generate_data_cnn2d.py
```

Example of an interactive job

- `salloc –t 30 –N 1 –c 7 –p gpu --ntasks-per-node=1`