Data vs Task Parallelism

Working definition:

- Data parallel computation: parallelism is applied by performing the same operation to different items of data at the same time; the amount of parallelism grows with the size of the data.
- Task parallel computation: parallelism is applied by performing distinct computations – or tasks – at the same time. Since the number of tasks is fixed, the parallelism is not scalable.
MPI

Tutorial at Lawrence Livermore National Labs: 
https://computing.llnl.gov/tutorials/mpi/

Library

To run MPI programs use mpirun or mpiexec and pass it the number of processes to run

MPI uses a communicator to handle the inter-process communication. For the most part we will be using the predefined MPI_COMM_WORLD
**MPI_Init()**

```c
int MPI_Init(int *argc, char **argv, int *errcode);
```

// Initialize MPI

// Number of command line arguments

// Command line arguments

**Arguments:**

- Number of command line arguments.
- Command line arguments.

**Notes:**

This routine must be called in every MPI process before any other MPI routine is called. It is an error to call this routine more than once in a process unless a subsequent MPI_Finalize() is called.

**Return value:**

An MPI error code.
MPI_Finalize()

int MPI_Finalize();

Notes:
This routine should be the last MPI routine called in each process, and it should only be invoked after all other MPI routines have completed. In particular, any pending communication operations should complete before this routine is called.

Return value:
An MPI error code.
MPI_Comm_Size()

```c
int MPI_Comm_Size(
    MPI_Comm comm,
    int *size,
);
```

**Arguments:**

- The communicator of interest.
- A pointer to the size, whose target will contain the number of tasks in the specified communicator.

**Notes:**

This routine obtains the number of processes in a communicator.

**Return value:**

An MPI error code.
MPI_Comm_Rank()

```c
int MPI_Comm_Rank(
    MPI_Comm comm,
    int *rank,
);
```

Notes:

This routine obtains a process’ rank within a communicator.

Arguments:

- The communicator of interest.
- A pointer to the rank, whose target will contain the rank of the specified communicator.

Return value:

An MPI error code
Simple Example

- MPI_Init
- Determine which process we are
- Do some work, totally ignoring our process rank
- MPI_Finalize
MPI_Send() is a blocking send operation in MPI, used to send data to another process. The function signature is as follows:

```c
int MPI_Send( void *buffer, int count, MPI_Datatype type, int dest, int tag, MPI_Comm *comm );
```

- **Arguments:**
  - `void *buffer`: The address of the data to send.
  - `int count`: The number of data elements to send.
  - `MPI_Datatype type`: The type of data elements to send.
  - `int dest`: The ID of the destination process.
  - `int tag`: The tag to distinguish this message.
  - `MPI_Comm *comm`: An MPI communicator.

- **Arguments:**
  - The address of the data to send.
  - The number of data elements to send.
  - The type of data elements to send.
  - The ID of the process that should receive this message.
  - A message tag that distinguishes this message from others that may be sent to the same process.
  - The MPI communicator to use.

- **Notes:**
  - This routine sends data to another process. This routine has blocking semantics, which means that the routine does not return until the message has been sent. MPI_Isend() is a non-blocking version of the send operation; it takes a seventh parameter of type MPI_Request that is used to differentiate this send from other invocations of MPI_Isend() when waiting for completion.

- **Return value:**
  - An MPI error code.
MPI_Recv()