Введение в MPI (практика)

Лектор — Долуденко Алексей Николаевич

План семинара

1. Работа на **кластере**
2. Базовые концепции **MPI**
3. Простая **программа**
4. **Запуск** простой программы
5. Посылка сообщений. Elapsed time

Доступ к кластеру

```
ssh USER@calc.cod.phystech.edu
```

Элементарные команды Linux

- посмотреть файлы в текущем каталоге
- посмотреть путь текущего каталога
- сделать новый каталог
- удалить каталог
- удалить файл
- перейти в каталог
  - в корневой каталог
  - на уровень выше
- установить права файла на исполнение
- заход на удаленную машину по протоколу ssh (Secure Shell)

Элементарные команды Linux

- **ls** - посмотреть файлы в текущем каталоге
- **pwd** - посмотреть путь текущего каталога
- **mkdir** имя_каталога - сделать новый каталог
- **rmdir** имя_каталога - удалить каталог
- **rm** имя_файла - удалить файл
- **cd** имя_каталога - перейти в каталог
  - **cd** / в корневой каталог
  - **cd** .. на уровень выше
- **chmod** 755 имя_файла - установить права файла на исполнение
- **ssh** –p порт логин@имя_удаленной_машины - заход на удаленную машину по протоколу ssh (Secure Shell)

Элементы редактора vi

- **vi** myfile.c – создать новый или открыть старый файл
- клавиша i – перейти в режим ввода текста
- клавиша Esc – вернуться в режим набора команд
- **:wq** – записать в файл и выйти
- **:q!** - выйти без сохранения

Установка MPI (локально)

- Ubuntu / Mint: `sudo apt install libopenmpi-dev openmpi-doc`
- Mac OS: `brew install open-mpi`

На Ubuntu доступны man-страницы по всем функциям MPI

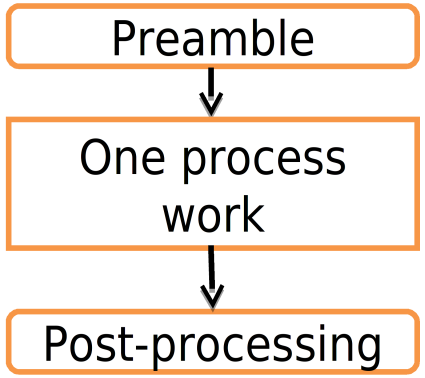
Также man-страницы доступны здесь:
https://www.open-mpi.org/doc/current/

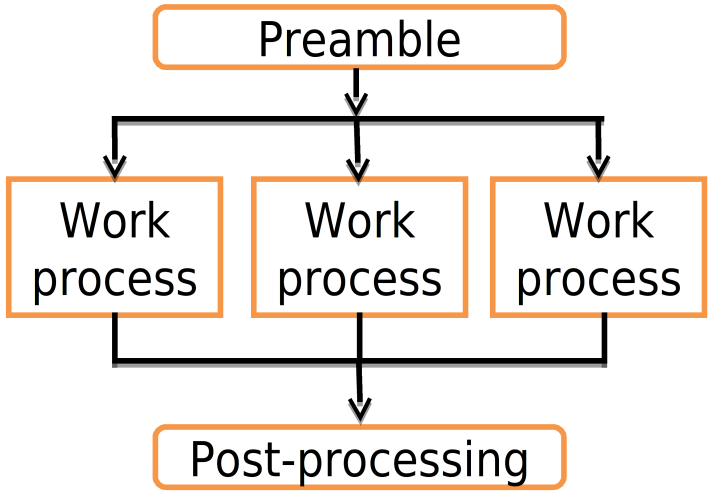
История и обзор

- MPI = Message Passing Interface.
- Разработка MPI 1.0: с 1991 по **1994**
- MPI — **протокол** обмена сообщениями + набор интерфейсов на C, C++ и Fortran
- Де-факто стандарт для **суперкопьютеров** с **распределённой памятью**
- Разработка MPI ведётся сообществом

Основные понятия

- Процесс — компьютерная программа в стадии своего выполнения. Программа есть набор инструкций; процесс является активным выполнением этих инструкций.
- В MPI единицей исполнения является **процесс** (сравни: thread).





Простая программа (1)

```
#include<stdio.h>
#include<stdlib.h>
#include<mpi.h>

int main(int argc, char *argv[]){
   int myrank, size;
   int array[10];
   MPI_Status status;
```

Простая программа (2)

```
/* MPI_Init запускает несколько процессов */
MPI_Init(&argc, &argv);

/* size — общее число процессов */
MPI_Comm_size(MPI_COMM_WORLD, &size);

/* у каждого процесса собственный уникальный rank */
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
```

Простая программа (3)

```
/* Все процессы работают одновременно, а номера процессов
позволяют распределить задания в рамках модели single
program multiple data. Процесс с rank 0 обычно выделяется
для распределения задач */

    printf("I am %d of %d\n", myrank, size);
    MPI_Finalize();
    return 0;
}
```

Запуск на кластере

```
module add mpi/openmpi4-x86_64
```

Подготовить терминал для запуска MPI

```
mpicc main.c
mpicxx main.cpp
```

Скомпилировать MPI-программу

```
mpiexec -np N ./a.out
```

Исполнить MPI-программу

```
sbatch -n N ./run.sh
```

Отправить задачу в очередь Slurm для исполнения

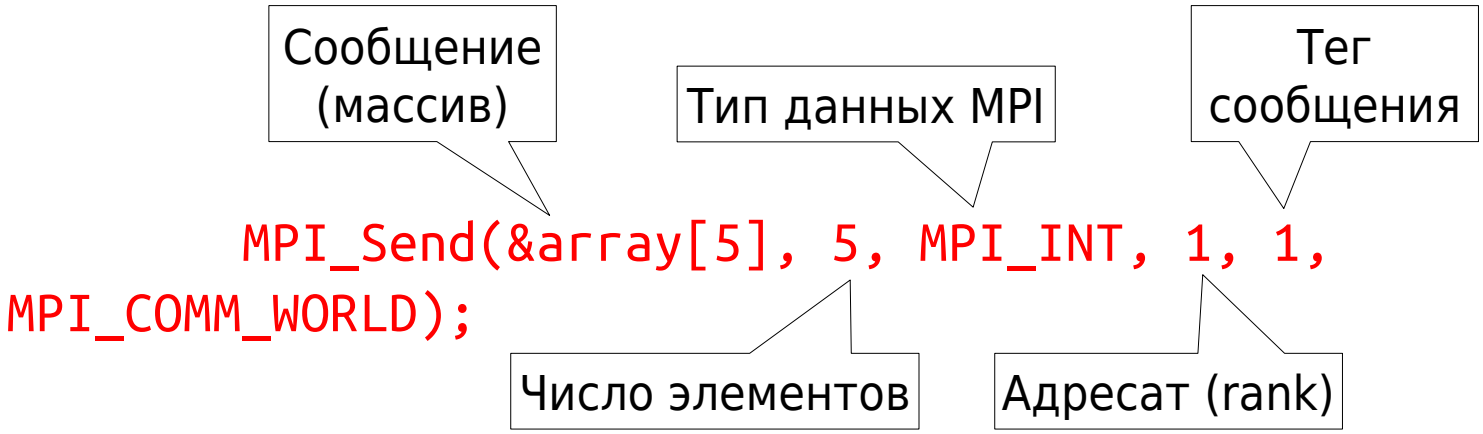
sbatch-файл

```
#!/usr/bin/env bash
#
#SBATCH --ntasks=8
#SBATCH --cpus-per-task=1
#SBATCH --job-name=my_job
#SBATCH --output=out.txt
#SBATCH --error=error.txt
mpiexec ./MpiHelloWorld
```

Посылка сообщений

```
    printf("I am %d of %d\n", myrank, size);
    if (myrank == 0) {
        for (i = 0; i < 10; i++) {
            array[i] = i;
        }
        /* пересылка процессу с номером 1 */
        MPI_Send(&array[5], 5, MPI_INT, 1, 1,
MPI_COMM_WORLD);
    }
```

Посылка сообщений



Посылка сообщений

```
if (myrank > 0) {
    /* получение от процесса с номером 0: */
    MPI_Recv(array, 5, MPI_INT, 0, 1,
MPI_COMM_WORLD, &status);
    for (i = 0; i < 5; i++){
        printf("%d ", array[i]);
    }
    printf("\n");
}
```

Посылка сообщений

```
    MPI_Finalize();
    return 0;
}
```

Elapsed time

double MPI_Wtime(void)

время в секундах от некоторого момента в прошлом.

Гарантируется, что часы **монотонные**.

```
double elapsed = MPI_Wtime();
… // some code
elapsed = MPI_Wtime() - elapsed;
```

Полезные материалы

Примеры кода на MPI (и не только)

https://github.com/akhtyamovpavel/ParallelComputationExamples

Man-страницы

https://www.open-mpi.org/doc/current/

Стандарт MPI

https://www.mpi-forum.org/docs/mpi-2.1/mpi21-report.pdf