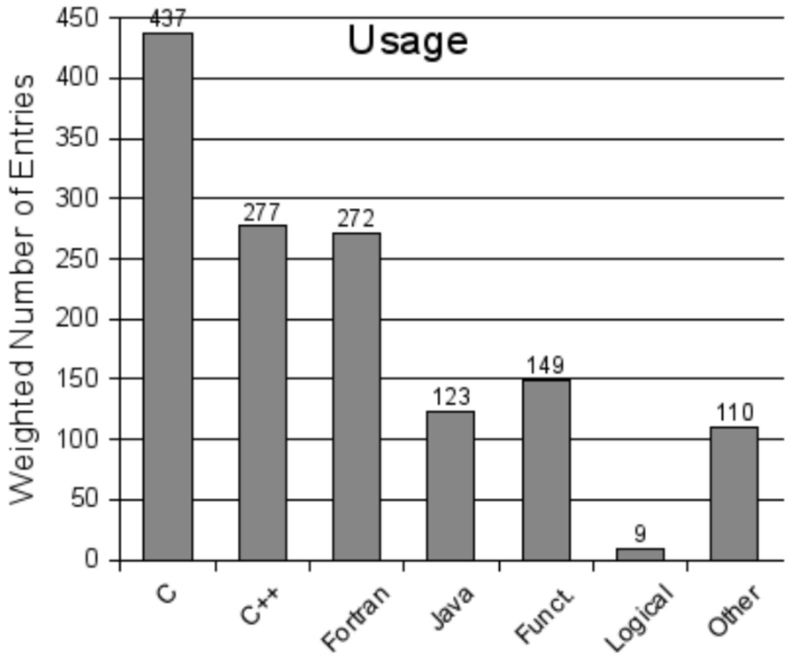
MPI practice

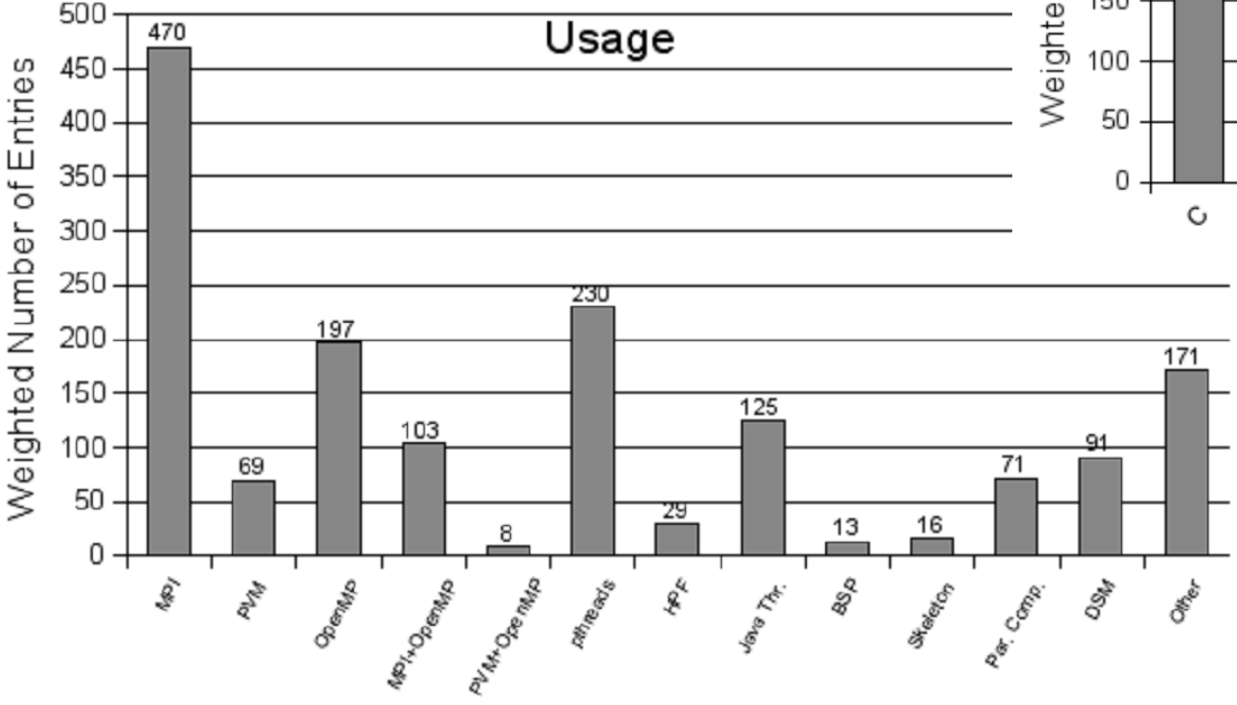
Aleksey Doludenko

Role and place of parallel languages and libraries

* - по данным опроса
Suess M, Leopold C. Observations on the Publicity and Usage of Paral-
lel Programming Systems and Languages: A Survey Approach. (2007)



The most popular parallel programming languages *



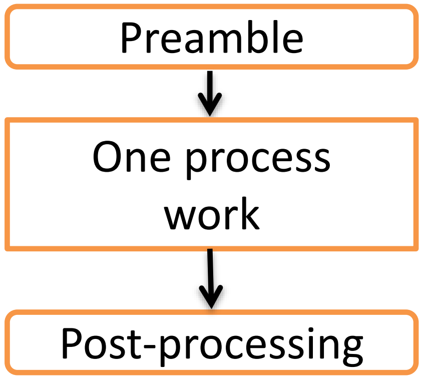
The most popular parallel programming libraries *

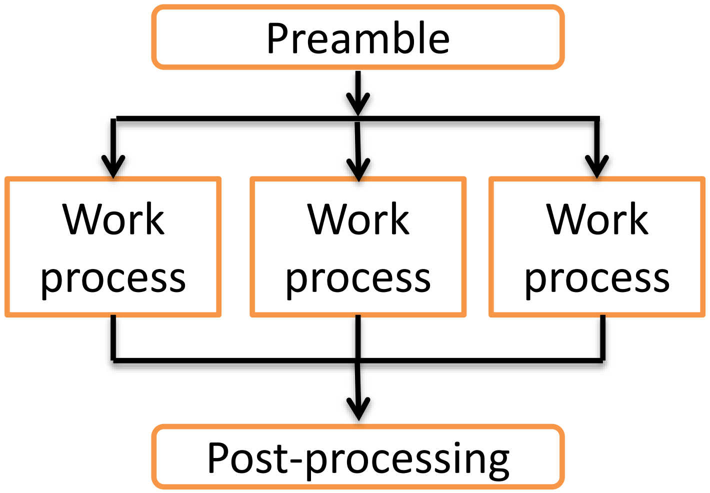
History and overview

- The message passing interface effort began in the summer of 1991. The draft MPI standard was presented at the Supercomputing '93 conference in November 1993. After a period of public comments, which resulted in some changes in MPI, version 1.0 of MPI was released in June 1994.
- MPI is a language-independent communications protocol used to program parallel computers.
- MPI is not sanctioned by any major standards body; nevertheless, it has become a de facto standard for communication among processes that model a parallel program running on a distributed memory system.

Concepts

- A process is an instance of a computer program that is being executed. A computer program is a passive collection of instructions; a process is the actual execution of those instructions.





Cluster. Host and logins

- **Login:**
  pd891XY,
  XY is the number
- **Password:**
  ...
- **Host**
  calc.cod.phystech.edu

Some Linux commands

- ls - view files in the current directory
- pwd - view the path to the current directory
- mkdir directory_name - make a new directory
- rmdir directory_name - remove the directory
- cd directory_name - go to the directory
- cd / - go to the root directory
- cd .. - go to a higher level
- chmod 755 filename - set file execution permissions
- ssh –p port login @ remote_machine_name - logging into a remote machine via ssh (Secure Shell), default port equals to 22

Some vi editor commands

- vi myfile.c – create a new or open the old file
- i – command after which you can enter some text
- press Esc, :wq, press Enter – write to the file and exit editor
- :q! – press to exit without saving

1st program, part 1

```
#include<stdio.h>
#include<stdlib.h>
#include<mpi.h>          //   mpi header file

int main(int argc, char *argv[]){
        int i;
        int array[10];
        int myrank, size;
        MPI_Status Status;    // mpi data type
```

1st program, part 2

```
    /* MPI programs start with MPI_Init; all 'N' processes exist
thereafter */
    MPI_Init(&argc, &argv);

    /* find out how big the world of processes is */
   MPI_Comm_size(MPI_COMM_WORLD, &size);

    /* and this processes' rank is */
    MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
```

1st program, part 3

```
 /* At this point, all processes are running equivalently, the rank
distinguishes the roles of the processes in the program,    with
rank 0 often used specially... */

  printf("I am %d of %d\n", myrank, size);

    /* MPI programs end with MPI Finalize*/

  MPI_Finalize();
  return 0;
}
```

Compilation and running

Before compiling, type in the command line

module add mpi/openmpi4-x86_64

Compilation:

mpicc   file_name.c

(by default the executable is named "a.out")

Running:

mpirun   -np   num_of_processes ./a.out

(num_of_processes is the number!)

1st program, part 4

/* At this point, all processes are running equivalently, the rank distinguishes the roles of the processes in the program, with rank 0 often used specially... */

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

1st program, part 5

```
if (myrank == 1){

    /* receive from rank 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 ");
}
```

1st program, part 6

```
/* The MPI program must end using the MPI Finalize function */


  MPI_Finalize();
  return 0;
}
```

Compilation and running

Compilation:

```
mpicc    file_name.c
```

(by default the executable is named "a.out")

Running:

```
mpirun    -np    num_of_processes ./a.out
```

(num_of_processes is the number!)

Some software design

```
 if (myrank == 0){
     for (i = 1; i < size; i++){
          /* sending messages to processes with ranks i: */
          MPI_Send(&buf[i*N/size], N/size, MPI_INT, i, i,
                                                  MPI_COMM_WORLD) ;
     }
}

if (myrank != 0){
        /* each of the processes receives a message from the
process rank 0*/
        MPI_Recv (&buf[0], N/size, MPI_INT, 0, myrank,
                              MPI_COMM_WORLD,  &Status);
}
```

Determining the running time of a parallel program

double MPI_Wtime(void) – returns the astronomical time in seconds (real number) since some point in the past. The difference between the returned values will show the operating time of this section.

Sample:

```
double begin, end, total;
begin = MPI_Wtime();
....
end = MPI_Wtime();
total = end – begin;
```