

# Riviera HPC SBATCH Documentation

---

## 1 Job Arrays

### 1.1 Use Cases

Running multiple instances of the same job with different arguments without a need to communicate between jobs.

### 1.2 Examples

```
1  #!/bin/bash
2  #SBATCH --job-name=array-example
3  ##SBATCH --array=1-5 # Submits a job array with index values between 1 and 5
4  #SBATCH --array=1,3,5,7 # Submits a job array with index values of 1,3,5,7
5  ##SBATCH --array=1-7:2 # Submits a job array with index values between 1 and 7 with steps
   ↳ of 2 (1,3,5,7)
6  ##SBATCH --array=1-5%2 # Submis a job array with index values between 1 and 5 but limits
   ↳ the number of simultaneously running tasks for this job array to 4
7  #SBATCH --partition=short-cpu
8  #SBATCH --output=%A/out_%a.out # The output file will be in a folder with the name jobId
   ↳ and will have the form out_arrayIndex
9  #SBATCH --error=%A/error_%a.err # The error file will be in a folder with the name jobId
   ↳ and will have the form error_arrayIndex
10 #SBATCH --ntasks=1
11 #SBATCH --time=00:05:00
12
13 module load python39
14
15 srun python3 ~/examples/array-example/array-example.py $SLURM_ARRAY_JOB_ID
   ↳ $SLURM_ARRAY_TASK_ID # Calls a python script with the arguments jobId and arrayIndex
```

Listing 1: Job Array SBATCH File Example

## 2 MPI

### 2.1 Use Cases

Process based parallelization where the different process can communicate with each other by passing messages. It works on both distributed and shared memory systems.

## 2.2 Examples

### mpi-binary-search.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4  #include <mpi.h>
5  #define ARRAY_SIZE 10
6  int binarySearch(int arr[], int key, int begin, int end) {
7      int mid_point = (begin + end) / 2;
8      if(arr[mid_point] == key) return mid_point;
9      else if(abs(begin - end) == 1) return -1;
10     else if(key > arr[mid_point]) return binarySearch(arr, key, mid_point + 1, end);
11     else return binarySearch(arr, key, begin, mid_point - 1);
12     return -1;
13 }
14 void insertionSort(int arr[], int n) {
15     int i, j, key;
16     for(i = 1; i < n; ++i) {
17         key = arr[i];
18         j = i - 1;
19         while(j >= 0 && key < arr[j]) {
20             arr[j + 1] = arr[j];
21             j = j - 1;
22         }
23         arr[j + 1] = key;
24     }
25 }
26 int main(int argc, char** argv) {
27     static int arr[ARRAY_SIZE];
28     time_t t;
29     int i;
30     size_t n = sizeof(arr)/sizeof(arr[0]);
31     MPI_Status status;
32     MPI_Init(&argc, &argv);
33     int pid;
34     MPI_Comm_rank(MPI_COMM_WORLD, &pid);
35     int number_of_processes;
36     MPI_Comm_size(MPI_COMM_WORLD, &number_of_processes);
37     if (pid == 0) {
38         srand((unsigned) time(&t));
39         for( i = 0 ; i < n ; ++i ) arr[i] = rand() % 50;
40         int index, i;
41         int elms;
```

```

42     srand((unsigned) time(&t));
43     int key = rand() % 50;
44     elms = ARRAY_SIZE / number_of_processes;
45     if (number_of_processes > 1) {
46         for (i = 1; i < number_of_processes - 1; i++) {
47             index = i * elms;
48             MPI_Send(&key, 1, MPI_INT, i, 0, MPI_COMM_WORLD);
49             MPI_Send(&elms, 1, MPI_INT, i, 0, MPI_COMM_WORLD);
50             MPI_Send(&arr[index], elms, MPI_INT, i, 0,
51                 ↪ MPI_COMM_WORLD);
52         }
53         index = i * elms;
54         int elements_left = ARRAY_SIZE - index;
55         MPI_Send(&key, 1, MPI_INT, i, 0, MPI_COMM_WORLD);
56         MPI_Send(&elements_left, 1, MPI_INT, i, 0, MPI_COMM_WORLD);
57         MPI_Send(&arr[index], elements_left, MPI_INT, i, 0,
58             ↪ MPI_COMM_WORLD);
59     }
60     insertionSort(arr, elms);
61     index = binarySearch(arr, key, 0, elms);
62     if (index == - 1)
63         for (i = 1; i < number_of_processes; i++) {
64             MPI_Recv(&index, 1, MPI_INT, MPI_ANY_SOURCE, 0,
65                 ↪ MPI_COMM_WORLD, &status);
66             if (index == -1) printf("the key %d is not in the
67                 ↪ array\n", key);
68             else printf("the key %d is found in the array\n", key);
69         }
70     else printf("the key %d is found in the array\n", key);
71 } else {
72     int key = 0;
73     MPI_Recv(&key, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
74     int recv = 0;
75     MPI_Recv(&recv, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
76     int buffer[recv];
77     size_t n = sizeof(buffer)/sizeof(buffer[0]);
78     MPI_Recv(&buffer, recv, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
79     insertionSort(buffer, n);
80     int index = binarySearch(buffer, key, 0, n);
81     MPI_Send(&index, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
82 }
83 MPI_Finalize();
84 return 0;
85 }

```

## mpi-bin.sh

```
1  #!/bin/bash
2
3  #SBATCH --job-name=MpiBinarySearch
4  #SBATCH --output=%A/mpi-bin.out
5  #SBATCH --error=%A/mpi-bin.err
6  #SBATCH --ntasks=10
7  #SBATCH --time=00:05:00
8
9  module load openmpi4/gcc
10 module load gcc
11
12 srun mpicc mpi-binary-search.c -o mpi-binary-search
13
14 srun --mpi=pmix
   ↪ --export=ALL,OMPI_MCA_btl_openib_allow_ib=true,OMPI_MCA_btl=openib,self,sm
   ↪ ./mpi-binary-search
```

## 3 OpenMP

### 3.1 Use Cases

Thread based parallelization where the different threads share memory.

### 3.2 Examples

#### openmp-binary-search.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <omp.h>
4  #include <time.h>
5  #define ARRAY_SIZE 10
6  int binarySearch(int arr[], int key, int begin, int end) {
7      int mid_point = (begin + end) / 2;
8      if (arr[mid_point] == key) return mid_point;
9      else if (abs(begin - end) == 1) return -1;
10     else if (key > arr[mid_point]) return binarySearch(arr, key, mid_point + 1, end);
11     else return binarySearch(arr, key, begin, mid_point - 1);
12     return -1;
13 }
14 void insertionSort(int arr[], int n) {
15     int i, j, key;
```

```

16     for (i = 1; i < n; ++i) {
17         key = arr[i];
18         j = i - 1;
19         while (j >= 0 && key < arr[j]) {
20             arr[j+1] = arr[j];
21             j = j - 1;
22         }
23         arr[j+1] = key;
24     }
25 }
26 int main(int argc, char** argv) {
27     static int arr[ARRAY_SIZE];
28     time_t t;
29     int i;
30     size_t n = sizeof(arr)/sizeof(arr[0]);
31     srand((unsigned) time(&t));
32     for (i = 0; i < n; ++i) arr[i] = rand() % 50;
33     int key = rand() % 50;
34     int num_threads = omp_get_max_threads();
35     int elms = ARRAY_SIZE / num_threads;
36     int found_index = -1;
37     printf("num_threads=%d\n", num_threads);
38     #pragma omp parallel
39     {
40         int tid = omp_get_thread_num();
41         int index = tid * elms;
42         int elements_left = (tid == num_threads - 1) ? ARRAY_SIZE - index : elms;
43         int local[elements_left];
44         for (int i = 0; i < elements_left; i++) local[i] = arr[index + i];
45         insertionSort(local, elements_left);
46         int local_index = binarySearch(local, key, 0, elements_left);
47         if (local_index != -1) {
48             int global_index = index + local_index;
49             #pragma omp critical
50             {
51                 found_index = global_index;
52             }
53         }
54     }
55     if (found_index == -1) printf("The key %d is not in the array.\n", key);
56     else printf("The key %d is found at index %d.\n", key, found_index);
57     return 0;
58 }

```

## openmp-bin.sh

```
1  #!/bin/bash
2  #SBATCH --job-name=omp-bin-search
3  #SBATCH --output=%A/omp-bin.out
4  #SBATCH --error=%A/omp-bin.err
5  #SBATCH --time=00:05:00
6  #SBATCH --cpus-per-task=4
7
8  module load openmpi/gcc/64
9
10 srun gcc -fopenmp openmp-binary-search.c -o openmp-binary-search
11
12 export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
13
14 srun --mpi=pmi2 ./openmp-binary-search
```