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

```
#!/bin/bash
#SBATCH --job-name=array-example
##SBATCH --array=1-5 # Submits a job array with index values between 1 and 5
#SBATCH --array=1,3,5,7 # Submits a job array with index values of 1,3,5,7
##SBATCH --array=1-7:2 # Submits a job array with index values between 1 and 7 with steps of 2 (1,3,5,7)
##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
#SBATCH --partition=short-cpu
#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
#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
#SBATCH --ntasks=1
#SBATCH --time=00:05:00

module load python39

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

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <mpi.h>
#define ARRAY_SIZE 10
int binarySearch(int arr[], int key, int begin, int end) {
        int mid_point = (begin + end) / 2;
        if(arr[mid_point] == key) return mid_point;
        else if(abs(begin - end) == 1) return -1;
        else if(key > arr[mid_point]) return binarySearch(arr, key, mid_point + 1, end);
        else return binarySearch(arr, key, begin, mid_point - 1);
        return -1;
}
void insertionSort(int arr[], int n) {
        int i, j, key;
        for(i = 1; i < n; ++i) {
                key = arr[i];
                j = i - 1;
                while(j >= 0 && key < arr[j]) {
                        arr[j + 1] = arr[j];
                        j = j  - 1;
                }
                arr[j + 1] = key;
        }
}
int main(int argc, char** argv) {
        static int arr[ARRAY_SIZE];
        time_t t;
        int i;
        size_t n = sizeof(arr)/sizeof(arr[0]);
        MPI_Status status;
        MPI_Init(&argc, &argv);
        int pid;
        MPI_Comm_rank(MPI_COMM_WORLD, &pid);
        int number_of_processes;
        MPI_Comm_size(MPI_COMM_WORLD, &number_of_processes);
        if (pid == 0) {
                srand((unsigned) time(&t));
                for( i = 0 ; i < n ; ++i ) arr[i] = rand() % 50;
                int index, i;
                int elms;
```

```
                srand((unsigned) time(&t));
                int key = rand() % 50;
                elms = ARRAY_SIZE / number_of_processes;
                if (number_of_processes > 1) {
                        for (i = 1; i < number_of_processes - 1; i++) {
                                index = i * elms;
                                MPI_Send(&key, 1, MPI_INT, i, 0, MPI_COMM_WORLD);
                                MPI_Send(&elms, 1, MPI_INT, i, 0, MPI_COMM_WORLD);
                                MPI_Send(&arr[index], elms, MPI_INT, i, 0,
                                 ↪  MPI_COMM_WORLD);
                        }
                        index = i * elms;
                        int elements_left = ARRAY_SIZE - index;
                        MPI_Send(&key, 1, MPI_INT, i, 0, MPI_COMM_WORLD);
                        MPI_Send(&elements_left, 1, MPI_INT, i, 0, MPI_COMM_WORLD);
                        MPI_Send(&arr[index], elements_left, MPI_INT, i, 0,
                         ↪  MPI_COMM_WORLD);
                }
                insertionSort(arr, elms);
                index = binarySearch(arr, key, 0, elms);
                if (index == - 1)
                        for (i = 1; i < number_of_processes; i++) {
                                MPI_Recv(&index, 1, MPI_INT, MPI_ANY_SOURCE, 0,
                                 ↪  MPI_COMM_WORLD, &status);
                                if (index == -1) printf("the key %d is not in the
                                 ↪  array\n", key);
                                else printf("the key %d is found in the array\n", key);
                        }
                else printf("the key %d is found in the array\n", key);
        } else {
                int key = 0;
                MPI_Recv(&key, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
                int recv = 0;
                MPI_Recv(&recv, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
                int buffer[recv];
                size_t n = sizeof(buffer)/sizeof(buffer[0]);
                MPI_Recv(&buffer, recv, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
                insertionSort(buffer, n);
                int index = binarySearch(buffer, key, 0, n);
                MPI_Send(&index, 1, MPI_INT, 0, 0, MPI_COMM_WORLD);
        }
        MPI_Finalize();
        return 0;
}
```

mpi-bin.sh

```
#!/bin/bash

#SBATCH --job-name=MpiBinarySearch
#SBATCH --output=%A/mpi-bin.out
#SBATCH --error=%A/mpi-bin.err
#SBATCH --ntasks=10
#SBATCH --time=00:05:00

module load openmpi4/gcc
module load gcc

srun mpicc mpi-binary-search.c -o mpi-binary-search

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

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <time.h>
#define ARRAY_SIZE 10
int binarySearch(int arr[], int key, int begin, int end) {
        int mid_point = (begin + end) /2;
        if (arr[mid_point] == key) return mid_point;
        else if (abs(begin - end) == 1) return -1;
        else if (key > arr[mid_point]) return binarySearch(arr, key, mid_point + 1, end);
        else return binarySearch(arr, key, begin, mid_point - 1);
        return -1;
}
void insertionSort(int arr[], int n) {
        int i, j, key;
```

```
        for (i = 1; i < n; ++i) {
                key = arr[i];
                j = i - 1;
                while (j >= 0 && key < arr[j]) {
                        arr[j+1] = arr[j];
                        j = j - 1;
                }
                arr[j+1] = key;
        }
}
int main(int argc, char** argv) {
        static int arr[ARRAY_SIZE];
        time_t t;
        int i;
        size_t n = sizeof(arr)/sizeof(arr[0]);
        srand((unsigned) time(&t));
        for (i = 0; i < n; ++i) arr[i] = rand() % 50;
        int key = rand() % 50;
        int num_threads = omp_get_max_threads();
        int elms = ARRAY_SIZE / num_threads;
        int found_index = -1;
        printf("num_threads=%d\n", num_threads);
        #pragma omp parallel
        {
                int tid = omp_get_thread_num();
                int index = tid * elms;
                int elements_left = (tid == num_threads - 1) ? ARRAY_SIZE - index : elms;
                int local[elements_left];
                for (int i = 0; i < elements_left; i++) local[i] = arr[index + i];
                insertionSort(local, elements_left);
                int local_index = binarySearch(local, key, 0, elements_left);
                if (local_index != -1) {
                        int global_index = index + local_index;
                        #pragma omp critical
                        {
                                found_index = global_index;
                        }
                }
        }
        if (found_index == -1) printf("The key %d is not in the array.\n", key);
        else printf("The key %d is found at index %d.\n", key, found_index);
        return 0;
}
```

openmp-bin.sh

```
#!/bin/bash
#SBATCH --job-name=omp-bin-search
#SBATCH --output=%A/omp-bin.out
#SBATCH --error=%A/omp-bin.err
#SBATCH --time=00:05:00
#SBATCH --cpus-per-task=4

module load openmpi/gcc/64

srun gcc -fopenmp openmp-binary-search.c -o openmp-binary-search

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK

srun --mpi=pmi2 ./openmp-binary-search
```