AUTOMATING HPC SIMULATIONS ON RESCALE CLOUD PLATFORM USING REST API

Karthik Reddy Lyathakula

PhD Candidate, North Carolina State University

GitHub: https://github.com/karthikncsu/Rescale-Tutorial

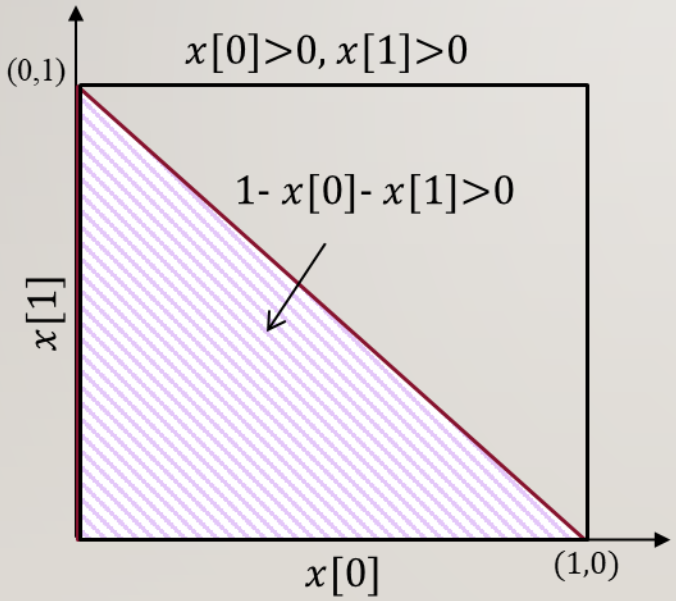
CONTENTS

- Problem: Sampling from High Dimensional Space
- Sampling Methods
- Parallelization of Sampling
  - Sequential Monte Carlo (SMC) Method
- Benchmarking Speed-up of SMC code
  - Simulation on Rescale Cloud Platform
  - Automating multi jobs using REST API

SAMPLING FROM HIGH DIMENSIONAL SPACE

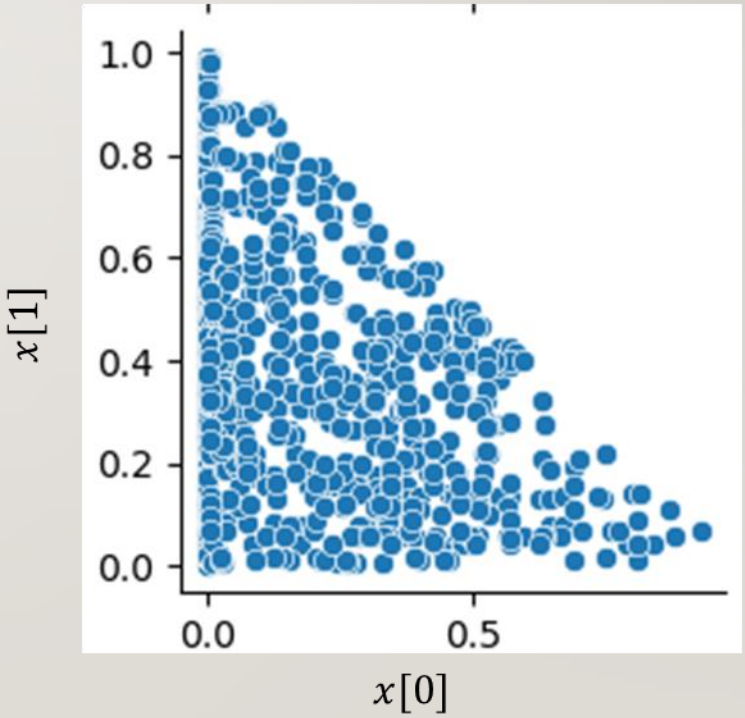
n-dimensional space with *K* non-linear constraints

$$C_k(\boldsymbol{x}) > 0, k = 1, \cdots, K$$
$$\boldsymbol{x} = (x_1, \cdots, x_n)$$



- Used in many applications
  - Uncertainty quantification
  - Inverse problems

Generated Samples

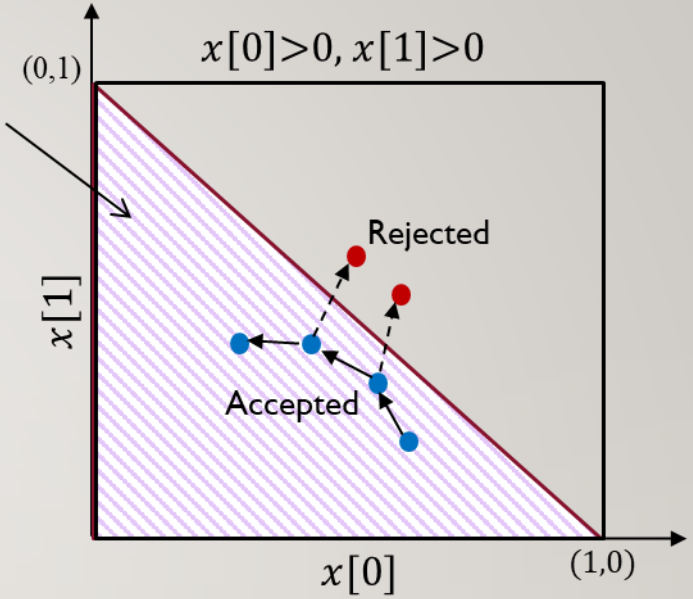


SAMPLING METHODS

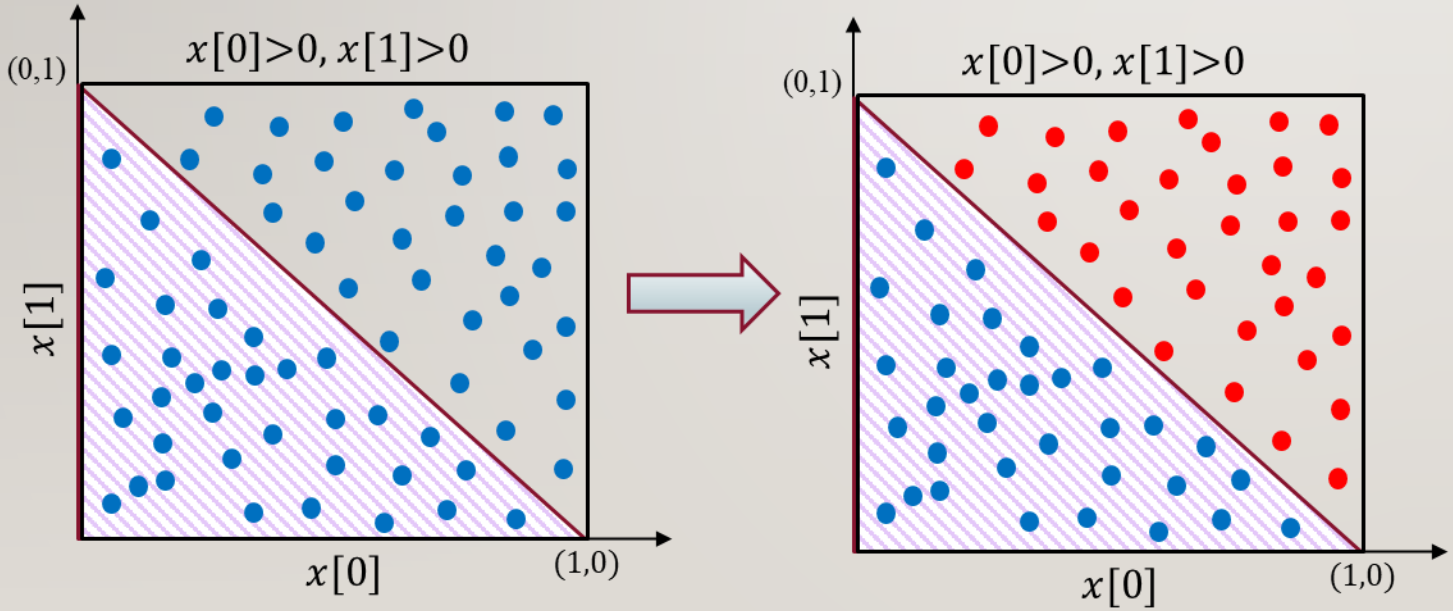
- Metropolis-Hastings Random Walk
- Adaptive Metropolis
- Gibbs Sampler
- Sequential Monte Carlo (SMC)

MCMC methods

Generates samples sequentially



Can be parallelized



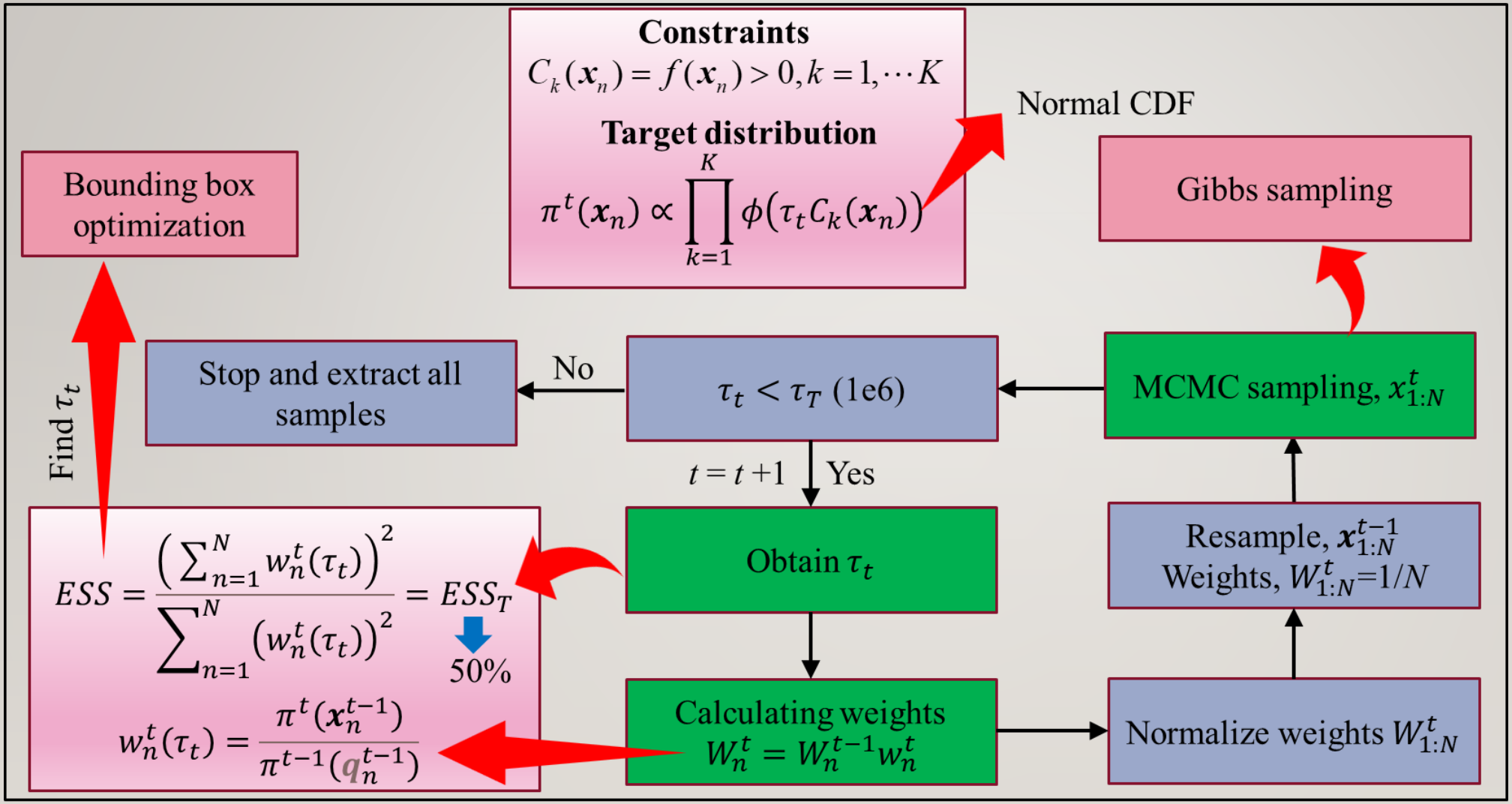
MCMC Methods: Fails to generate samples

SMC Methods: Can generate samples

High Dimensional Space

```
1.0 - x[0] - x[1] - x[2] - x[3] >= 0.0
x[3] / (x[0] + x[1]) - 0.1 >= 0.0
0.15 - x[3] / (x[0] + x[1]) >= 0.0
x[2] - 0.5 >= 0.0
x[0] + x[1]) / (x[0] + x[1] + x[3]) - 0.8 >= 0.0
```

SEQUENTIAL MONTE CARLO (SMC)



Parallelized

Serial

- **Coded in Python**

GitHub: https://github.com/karthikncsu/Sampling-from-high-dimensional-space

BENCHMARKING SMC CODE

- Benchmarking parallelization of SMC code
  - Calculating speedup (S_p) and efficiency (E_p)

$$S_p = \frac{T_s}{T_p} \qquad E_p = \frac{T_s}{nT_p}$$

T_s : Sequential simulation time
T_p : Parallelized simulation time

Simulation on Rescale cloud platform

- Multiple SMC simulations
  - Using 1,2 4, 8, 18 processors
- Automated simulations
  - Submit multi jobs using REST API
  - Download SMC log files once completed
  - Plot speedup and efficiency

AUTOMATING MULTI JOBS USING REST API

- Python is used for REST API
- Implemented using python class: BatchJobSubmit
- Steps for automating simulation
  - Step 1 : Upload input file
  - Step 2 : Setup and Submit job
  - Step 3 : Monitor status of Jobs
  - Step 4 : Download SMC log files
  - Step 5 : Postprocessing

AUTOMATING MULTI JOBS USING REST API

```python
def file_upload(self,input_file,oncloud=False):
    """
    Function for uploading input files
    """
    if oncloud:
        self.file_ids.append(file_id)
        return True,file_id
    else:
        #Uploading a file to teh rescale cloud
        file_upload = requests.post(
            'https://platform.rescale.com/api/v2/files/contents/',
            data=None,
            files={'file': open(input_file,'rb')},
            headers={'Authorization': self.myapi_token}
        )

        status=self.print_status(file_upload,"File upload status")
        if status:
            upload_details=json.loads(file_upload.text)
            file_id=upload_details["id"]
            print("File ID:",file_id)
            self.file_ids.append(file_id)
            return status
        else:
            return status
```

Step 1: Upload input file

- Place entire code into single folder
- Create a zip file
- File id is created for successfully uploaded input files

AUTOMATING MULTI JOBS USING REST API

```python
def setup_submit_job(self,data):
    """
    Function for setting up and submitting the job
    """

    job_setup=requests.post(
        'https://platform.rescale.com/api/v2/jobs/',
        json=data,
        headers={'Content-Type': 'application/json',
                 'Authorization': self.myapi_token}
    )

    status=self.print_status(job_setup,"Job setup status")
    if status:
        job_details=json.loads(job_setup.text)
        job_id=job_details["id"]
        print("Job ID:",job_id)

        job_submit=requests.post(
            'https://platform.rescale.com/api/v2/jobs/%s/submit/'%job_id,
            headers={'Content-Type': 'application/json',
                     'Authorization': self.myapi_token}
        )

        status=self.print_status(job_submit,"Job submiting status")

    if status:
        self.job_ids.append(job_id)
    else:
        self.job_ids.append(None)
    return status
```

Step 2 : Setup and Submit job

- Select
  - Number of processors
  - Core type
  - Input file id
  - Commands to be executed
  - Other details……

```python
def jobdata(file_id,numproc):

    command="cd Sampling-from-high-dimensional-space\n"
    command=command+"cd code\n"
    command=command+'conda install -y mpi4py \nmpirun -np '+str(numproc)+' python sampler.py alloy.txt alloy.out 20000 SMC'

    data={
        'name': 'Intern assignment sampling '+str(numproc),
        "projectId":'gxzXS',
        'jobanalyses': [
          {
            'useMpi': True,
            'command': command,
            'analysis': {
              'code': 'anaconda',
              'version': '5.3.1'
            },
            'hardware': {
              'coresPerSlot': numproc,
              'slots': 1,
              'coreType': 'hpc-3'
            },
            'inputFiles': [
              {
                'id': file_id
              }
            ]
          }
        ]
    }
```

AUTOMATING MULTI JOBS USING REST API

Step 2 : Setup and Submit job

- Multiple job submissions using loop

```
input_file='Sampling-from-high-dimensional-space.zip'


jobsobj=BatchJobSubmit()
status_input=jobsobj.file_upload(input_file)
if status_input:
    for numproc in [1,2,4,8,18,36]:
        data=jobdata(jobsobj.file_ids[0],numproc=numproc)
        status_submit=jobsobj.setup_submit_job(data)

jobsobj.status_job()
```

AUTOMATING MULTI JOBS USING REST API

```python
def status_ind_job(self):
  """
  Function for status of all submitted jobs
  """

  job_statuses=[]
  for job_id in self.job_ids:
    if job_id is not None:

      job_status=requests.get(
        'https://platform.rescale.com/api/v2/jobs/%s/statuses/'%job_id,
        headers={'Content-Type': 'application/json',
                 'Authorization': self.myapi_token})

      output_json=json.loads(job_status.text)
      status=output_json["results"][0]["status"]
      job_statuses.append(status)

      statusReason=output_json["results"][0]["statusReason"]
      print("Job Id:",job_id,", Status:",job_statuses[-1],", StatusReason:",statusReason)

    else:
      job_statuses.append("Completed")

  return job_statuses

def status_job(self):
  """
  Output for status of all the jobs
  """
  while True:
    time.sleep(15)

    print("---------",datetime.now(),"----------")
    if len(self.job_ids)==0:
      break
    job_statuses=self.status_ind_job()
    if job_statuses.count('Completed')==len(job_statuses):
      break
```

Step 3 : Monitor status of Jobs

```
TERMINAL
Job Id: jeWkhb , Status: Completed , StatusReason: Completed successfully
Job Id: ZTVkhb , Status: Completed , StatusReason: A run failed
--------- 2021-03-29 13:21:09.618741 ----------
Job Id: deWkhb , Status: Executing , StatusReason: None
Job Id: JrczW , Status: Executing , StatusReason: None
Job Id: XTVkhb , Status: Executing , StatusReason: None
Job Id: VBczW , Status: Completed , StatusReason: Completed successfully
Job Id: jeWkhb , Status: Completed , StatusReason: Completed successfully
Job Id: ZTVkhb , Status: Completed , StatusReason: A run failed
--------- 2021-03-29 13:21:25.426903 ----------
Job Id: deWkhb , Status: Executing , StatusReason: None
Job Id: JrczW , Status: Executing , StatusReason: None
Job Id: XTVkhb , Status: Executing , StatusReason: None
Job Id: VBczW , Status: Completed , StatusReason: Completed successfully
Job Id: jeWkhb , Status: Completed , StatusReason: Completed successfully
Job Id: ZTVkhb , Status: Completed , StatusReason: A run failed
--------- 2021-03-29 13:21:41.318903 ----------
Job Id: deWkhb , Status: Executing , StatusReason: None
Job Id: JrczW , Status: Executing , StatusReason: None
Job Id: XTVkhb , Status: Executing , StatusReason: None
Job Id: VBczW , Status: Completed , StatusReason: Completed successfully
Job Id: jeWkhb , Status: Completed , StatusReason: Completed successfully
Job Id: ZTVkhb , Status: Completed , StatusReason: A run failed
```

- Job with 36 processors failed due to technical issue with Rescale
  - mpi4py module cannot be installed using 36 processors

AUTOMATING MULTI JOBS USING REST API

```
def download_file(self,filename,job_id):
  """
  Download files from job numbers
  """

  #Finding all the url to download the files
  url='https://platform.rescale.com/api/v2/jobs/%s/files/'%job_id
  headers={'Authorization': self.myapi_token}

  file_download=requests.get(url, headers=headers)
  status=self.print_status(file_download,"File download status")
  file_found=False

  if status:
    job_details=json.loads(file_download.text)

    while True:
      #Get all the details of the files in the current page
      for arg in job_details["results"]:
        if arg["name"]==filename:
          file_found=True
          downloadUrl=arg["downloadUrl"]

          filename_save="./downloads/out_"+job_id+"_"+filename
          fout=open(filename_save,"wb")
          with closing(requests.get(downloadUrl, headers=headers, stream=True)) as r:
            with open(filename, 'r') as f:
              for chunk in r.iter_content(chunk_size=100):
                  fout.write(chunk)
          fout.close()
          fout=open(filename_save,"r")
          data=fout.readlines()
          fout.close()
          return True,data

      #Checking for details of the files in the next page
      if job_details["next"]==None:
        break
      url=job_details["next"]

      file_download=requests.get(url, headers=headers)
      job_details=json.loads(file_download.text)

    return False,None
```

Step 4 : Download SMC log files

```
all_proc_data=[]
for job_id in jobsobj.job_ids:
    status,data=jobsobj.download_file("comp_time.log",job_id)
    if status:
        all_proc_data.append(data[0].split())
```

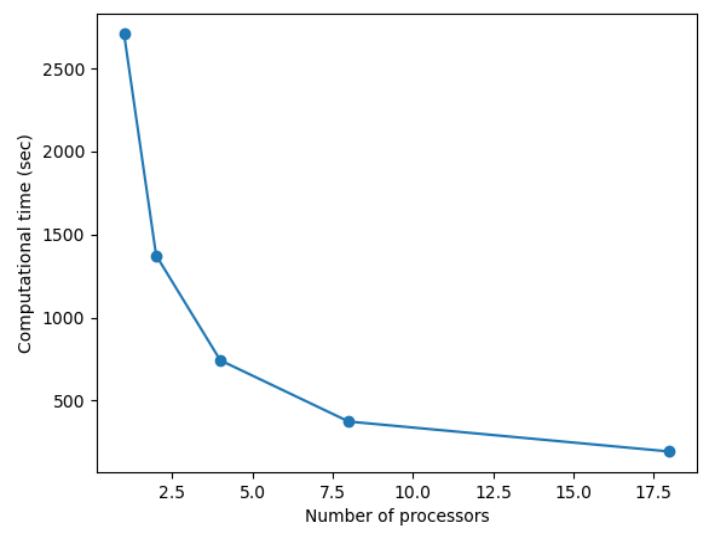
- Computational time is saved in comp_time.log by SMC on Rescale cloud

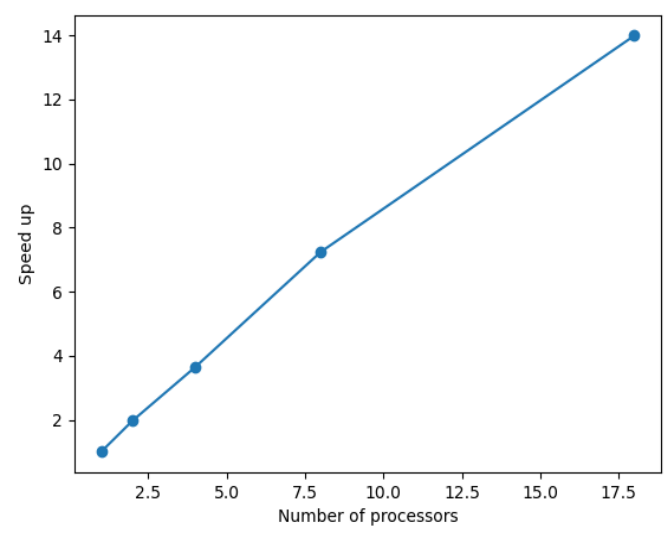
AUTOMATING MULTI JOBS USING REST API

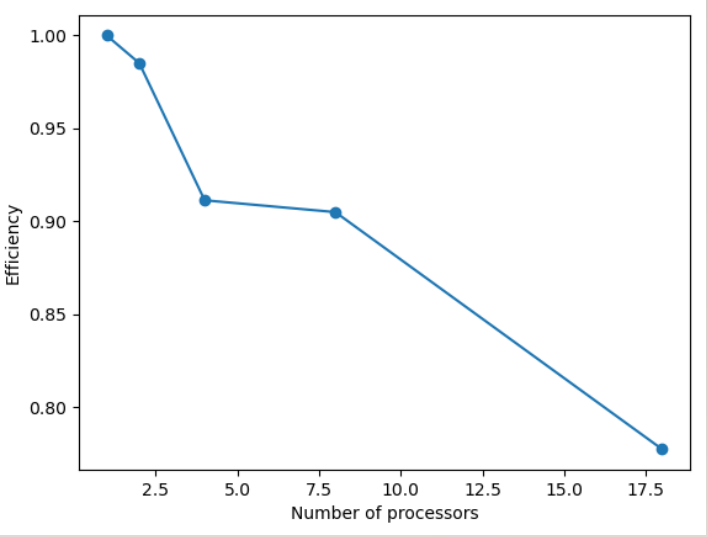
```
print(all_proc_data)
if len(all_proc_data)>0:
    all_proc_data=np.asarray(all_proc_data).astype(np.float64)

    fig=plt.figure()
    plt.plot(all_proc_data[:,0],all_proc_data[0,1]/all_proc_data[:,1],"o-")
    plt.xlabel("Number of processors")
    plt.ylabel("Speed up")
    fig.savefig("speedup.png")
    plt.close()

    fig=plt.figure()
    plt.plot(all_proc_data[:,0],all_proc_data[0,1]/(all_proc_data[:,1]*all_proc_data[:,0]),"o-")
    plt.xlabel("Number of processors")
    plt.ylabel("Efficiency")
    fig.savefig("efficiency.png")
    plt.close()

    fig=plt.figure()
    plt.plot(all_proc_data[:,0],all_proc_data[:,1],"o-")
    plt.xlabel("Number of processors")
    plt.ylabel("Computational time (sec)")
    fig.savefig("comptime.png")
    plt.close()
```

Step 5 : Postprocessing

- Computational time decreased with number of processors
- Linear speed up is achieved







Thank You