Serial vs parallel

python

Meelis Utt

Let's import necessary python packages and define example function f and it's analytical solutions.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import time

def f(x):
    return(x**2 + x**4+ np.sin(x) + np.cos(x) +x**25)

analytical = 613/390 + np.sin(1) - np.cos(1)
```

Let's define simple serial implementation of Monte-Carlo method.

```
def MC(n):
    pyunif = np.random.uniform(0,1,n)
    EX = np.mean(f(pyunif))
    error = np.abs(EX-analytical)
    return(EX,error)
```

Let's run the serial implementation and save results.

```
N = np.array(1000000 * np.array([1,2.5,5,7.5,10], dtype=float),dtype=int)
EX = []
error = []
type = []
walltime = []
for n in N:
  start = time.time()
  ex,err = MC(int(n))
  end = time.time()
  EX.append(ex);error.append(err);type.append("MCser");walltime.append(end-start)
```

Now let's define parallel implementation of Monte-Carlo method using Message Passing Interface (MPI).

```
cat MCpar.py
```

```
##
## from mpi4py import MPI
## from mpi4py.MPI import ANY_SOURCE
##
## import numpy as np
## import time
## import sys
##
```

```
## comm = MPI.COMM_WORLD
## rank = comm.Get_rank()
## size = comm.Get_size()
##
## n = int(sys.argv[1])
##
## def f(x):
##     return(x**2 + x**4+ np.sin(x) + np.cos(x) +x**25)
##
## analytical = 613/390 + np.sin(1) - np.cos(1)
##
##
## def MCpar(n):
##     if n%size!=0:
##         print("Nr of iterations is not divisible by nr of process")
##         exit()
##     ni = int(n//size)
##     pyunif = np.random.uniform(0,1,ni)
##     # TODO: explain multiplying with size
##     EXi = np.mean(f(pyunif))/size
##     if rank == 0:
##         EX = np.empty(1)
##     else:
##         EX = None
##     comm.Reduce(EXi,EX,op=MPI.SUM,root=0)
##     if rank == 0:
##         EX = EX[0]
##         error = np.abs(EX-analytical)
##         return(EX,error)
##     else:
##         return (None,None)
##
## start = time.time()
## ex,error = MCpar(n)
## end = time.time()
## if rank == 0:
##     print(n,ex,error,"MCpar",str(end-start),str(size),sep=",")
```

Let's run the parallel code, with the same n, but with different number of processes p.

```
# MCpar results
rm -f resultsPar.csv
for p in 1 2 4 5 10 25 50 100
do
  for n in 1000000 2500000 5000000 7500000 10000000
  do
    mpirun -n $p --hostfile hostfile python MCpar.py $n >> resultsPar.csv
  done
done
```

Let's vizualise the walltimes and relative speedups.
Walltimes:

```
# dict = {"n": N,"EX": EX,"error":error,"type":type,"walltime":walltime}
dict = {"n": N,"EX": EX,"error":error,"type":type,"walltime":walltime}
```

```
dfSer = pd.DataFrame(dict)
print(dfSer)
```

```
##           n        EX     error   type  walltime
## 0   1000000  1.872246  0.000717  MCser  0.108390
## 1   2500000  1.873481  0.000518  MCser  0.261949
## 2   5000000  1.872904  0.000060  MCser  0.536503
## 3   7500000  1.872983  0.000019  MCser  0.818761
## 4  10000000  1.873064  0.000101  MCser  1.103836
```

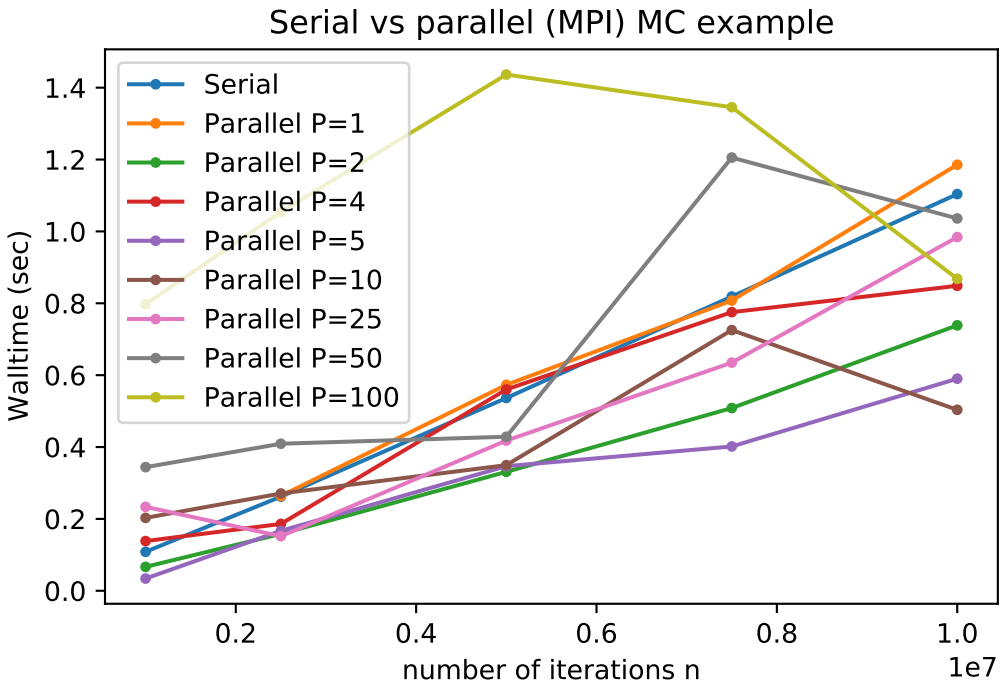
```
dfPar = pd.read_csv("resultsPar.csv",header=0,
  names=["n","EX","error","type","walltime","processes"])
print(dfPar)
```

```
##            n        EX     error   type  walltime  processes
## 0    2500000  1.873048  0.000084  MCpar  0.263915          1
## 1    5000000  1.872730  0.000233  MCpar  0.573036          1
## 2    7500000  1.872934  0.000030  MCpar  0.807565          1
## 3   10000000  1.873076  0.000112  MCpar  1.185522          1
## 4    1000000  1.872735  0.000228  MCpar  0.066495          2
## 5    2500000  1.873079  0.000116  MCpar  0.158256          2
## 6    5000000  1.872994  0.000031  MCpar  0.330961          2
## 7    7500000  1.872633  0.000331  MCpar  0.508478          2
## 8   10000000  1.873373  0.000410  MCpar  0.738672          2
## 9    1000000  1.873313  0.000349  MCpar  0.138310          4
## 10   2500000  1.872542  0.000421  MCpar  0.186077          4
## 11   5000000  1.873090  0.000127  MCpar  0.560181          4
## 12   7500000  1.872887  0.000077  MCpar  0.775292          4
## 13  10000000  1.872982  0.000019  MCpar  0.848675          4
## 14   1000000  1.871846  0.001117  MCpar  0.034008          5
## 15   2500000  1.873217  0.000254  MCpar  0.166746          5
## 16   5000000  1.873313  0.000350  MCpar  0.346693          5
## 17   7500000  1.872602  0.000362  MCpar  0.401604          5
## 18  10000000  1.872706  0.000258  MCpar  0.590169          5
## 19   1000000  1.871768  0.001195  MCpar  0.202893         10
## 20   2500000  1.872659  0.000304  MCpar  0.270713         10
## 21   5000000  1.873108  0.000145  MCpar  0.349660         10
## 22   7500000  1.873072  0.000108  MCpar  0.725423         10
## 23  10000000  1.872747  0.000216  MCpar  0.503518         10
## 24   1000000  1.874342  0.001379  MCpar  0.233594         25
## 25   2500000  1.873269  0.000305  MCpar  0.151836         25
## 26   5000000  1.873565  0.000601  MCpar  0.417937         25
## 27   7500000  1.872821  0.000142  MCpar  0.634833         25
## 28  10000000  1.872966  0.000003  MCpar  0.984255         25
## 29   1000000  1.872894  0.000069  MCpar  0.344006         50
## 30   2500000  1.872533  0.000431  MCpar  0.409171         50
## 31   5000000  1.872268  0.000696  MCpar  0.428620         50
## 32   7500000  1.873299  0.000335  MCpar  1.205514         50
## 33  10000000  1.872645  0.000318  MCpar  1.036092         50
## 34   1000000  1.871896  0.001068  MCpar  0.797653        100
## 35   2500000  1.872478  0.000485  MCpar  1.054086        100
## 36   5000000  1.873082  0.000119  MCpar  1.436539        100
## 37   7500000  1.873343  0.000379  MCpar  1.345594        100
## 38  10000000  1.872812  0.000152  MCpar  0.868207        100
```

```
ax = plt.gca()
dfSer.plot(kind='line',x='n',y='walltime',style='.-',rot=0,label='Serial')

dfPar.groupby('processes').plot(kind='line',x='n',y='walltime',
  style='.-',rot=0,ax=plt.gca(),label="Parallel "+str('processes'))

# plt.legend(bbox_to_anchor=(1.25,0.75), bbox_transform=ax.transData)
```

```
## processes
## 1      AxesSubplot(0.125,0.11;0.775x0.77)
## 2      AxesSubplot(0.125,0.11;0.775x0.77)
## 4      AxesSubplot(0.125,0.11;0.775x0.77)
## 5      AxesSubplot(0.125,0.11;0.775x0.77)
## 10     AxesSubplot(0.125,0.11;0.775x0.77)
## 25     AxesSubplot(0.125,0.11;0.775x0.77)
## 50     AxesSubplot(0.125,0.11;0.775x0.77)
## 100    AxesSubplot(0.125,0.11;0.775x0.77)
## dtype: object
```

```
plt.legend(['Serial','Parallel P=1','Parallel P=2','Parallel P=4',
  'Parallel P=5','Parallel P=10','Parallel P=25',
  'Parallel P=50','Parallel P=100'],loc='upper left')

plt.xlabel('number of iterations n')
plt.ylabel('Walltime (sec)')
plt.title('Serial vs parallel (MPI) MC example')
plt.show()
```



Relative speedups:

```
dfRel = dfPar.merge(dfSer,left_on="n",right_on="n",suffixes=(".par",".ser"))
dfRel["relative.speedup"] = dfRel["walltime.ser"]/dfRel["walltime.par"]
```

```
dfRel.groupby('processes').plot(kind='line',x='n',y='relative.speedup',
  style='.-',rot=0,ax=plt.gca(),label="Relative speedup "+str('processes'))
```

```
## processes
## 1      AxesSubplot(0.125,0.11;0.775x0.77)
## 2      AxesSubplot(0.125,0.11;0.775x0.77)
## 4      AxesSubplot(0.125,0.11;0.775x0.77)
## 5      AxesSubplot(0.125,0.11;0.775x0.77)
## 10     AxesSubplot(0.125,0.11;0.775x0.77)
## 25     AxesSubplot(0.125,0.11;0.775x0.77)
## 50     AxesSubplot(0.125,0.11;0.775x0.77)
## 100    AxesSubplot(0.125,0.11;0.775x0.77)
## dtype: object
```

```
plt.legend(['Parallel P=1','Parallel P=2','Parallel P=4',
  'Parallel P=5','Parallel P=10','Parallel P=25',
  'Parallel P=50','Parallel P=100'],loc='upper left')

plt.xlabel('number of iterations n')
plt.ylabel('Relative speedup')
plt.title('Serial vs parallel (MPI) MC example')
plt.show()
```