Parallel Python Best Practices

Version 1.0

By
Ayat Mohammed, Srijith Rajamohan and Nicholas Polys
Advanced Research Computing, Virginia Tech
Feb 10, 2016

Overview:

Parallel Python (**PP)** is a python module which provides mechanism for parallel execution of python code on **SMP** (systems with multiple processors or cores) and **clusters** (computers connected via network).
Currently this module is enabled on Newriver cluster to enable users to launch the ppserver on the computing nodes. In this document we report the results of two showcases we implemented and tested using standard example and a home-made script.

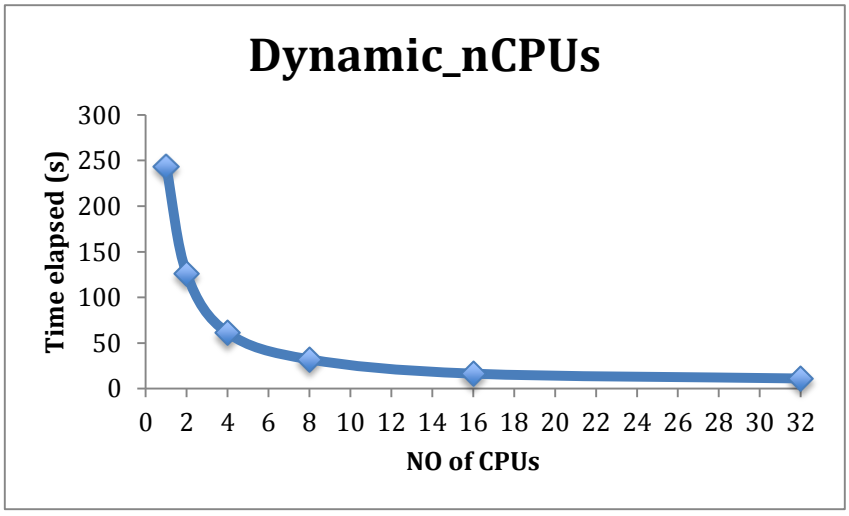
First Showcase:

```
#!/usr/bin/python
# File: dynamic_ncpus.py
# Author: Vitalii Vanovschi
# Desc: This program demonstrates parallel computations with pp module
# and dynamic cpu allocation feature.
# Program calculates the partial sum 1-1/2+1/3-1/4+1/5-1/6+... (in the limit
it is ln(2))
# Parallel Python Software: http://www.parallelpython.com
# Edited and modified by Ayat Mohammed on Feb 10th 2016

import math, sys, md5, time
import pp

def part_sum(start, end):
    """Calculates partial sum"""
    sum = 0
    for x in xrange(start, end):
        if x % 2 == 0:
           sum -= 1.0 / x
        else:
           sum += 1.0 / x
    return sum

print """Usage: python dynamic_ncpus.py"""
print

start = 1
end = 2000000000

# Divide the task into 64 subtasks
parts = 64
step = (end - start) / parts + 1

# Create jobserver

with open(sys.argv[1]) as f:
    content = f.readlines()
print content[0]
print content[1]

# tuple of all parallel python servers to connect with
```

```
ppservers = (content[0],content[1],)

# Execute the same task with different amount of active workers and measure
the time
for ncpus in (1, 2, 4, 8, 16, 32):

    # set the ncpus and the pp servers for the job server
    job_server = pp.Server(ncpus, ppservers=ppservers)
    # list of all jops that will be submitted to the server
    jobs = []
    start_time = time.time()
    print "Starting ", job_server.get_ncpus(), " workers"
    for index in xrange(parts):
        starti = start+index*step
        endi = min(start+(index+1)*step, end)
        jobs.append(job_server.submit(part_sum, (starti, endi)))

    # Retrieve all the results and calculate their sum
    part_sum1 = sum([job() for job in jobs])
    # Print the partial sum
    print "Partial sum is", part_sum1, "| diff =", math.log(2) - part_sum1

    print "Time elapsed: ", time.time() - start_time, "s"
    print
job_server.print_stats()
```

Usage: python dynamic_ncpus.py
Nodes: nr119, nr118
Job Count: 64

NO. Workers	Time elapsed (s)
1	243.344517946
2	125.837057114
4	61.0762679577
8	31.9891979694
16	16.3198401928
32	11.029312849



Second Showcase:

```
# File: pts_vtk_al.py
# Author: Ayat Mohammed
# Desc: This program demonstrates parallel file conversion with pp
  module and dynamic cpu allocation feature.
# the Program convert from pts file type to vtk file type using:
# pandas:http://pandas.pydata.org,
# evtk: https://bitbucket.org/pauloh/pyevtk and
# Parallel Python Software: http://www.parallelpython.com
# Edited and modified by Ayat Mohammed on FEb 10th 2016

import numpy as np
import sys, time
import pp
import pandas as pd
from evtk.hl import pointsToVTK

reading_time = time.time()
fixed_df = pd.read_csv('pts file path', sep=' ', header=None)
x_array = np.asarray(fixed_df[:][0].values, dtype = np.dtype(float))
y_array = np.asarray(fixed_df[:][1].values, dtype = np.dtype(float))
z_array = np.asarray(fixed_df[:][2].values, dtype = np.dtype(float))
color_array = np.asarray(fixed_df[:][3].values, dtype = np.dtype(float))

print "reading Time: ", time.time() - reading_time, "s"
# Fuction used in converting to a vtk file
def pts_vtk(filename,x_array,y_array,z_array,color_array):
    from evtk.hl import pointsToVTK
    pointsToVTK(filename,x_array,y_array,z_array, data = {"C" : color_array})
    return True

start = 1
end = len(x_array)
print "the arrays length is:", end
# Divide the task into 5 subtasks
parts = 5
step = (end - start) / parts + 1
print "the step is:", step
with open(sys.argv[2]) as f:
    content = f.readlines()
print content[0]
print content[1]
# tuple of all parallel python servers to connect with

ppservers = (content[0],content[1],)

if len(sys.argv) > 1:
    ncpus = int(sys.argv[1])
    # Creates jobserver with ncpus workers
    job_server = pp.Server(ncpus, ppservers=ppservers)
else:
    # Creates jobserver with automatically detected number of workers
    job_server = pp.Server(ppservers=ppservers)
```

```
print "Starting pp with", job_server.get_ncpus(), "workers"
start_time = time.time()
# Execution starts as soon as one of the workers will become available
jobs = []
filename = "output_file_path/output_file_name_"

for index in xrange(parts):
    starti = start+index*step
    endi = min(start+(index+1)*step, end)
    print "new_indices:", starti, endi
    newfilename=filename+str(index)
    jobs.append(job_server.submit(pts_vtk,
(newfilename,x_array[starti:endi],y_array[starti:endi],z_array[starti:endi],c
olor_array[starti:endi])))

print "Time elapsed: ", time.time() - start_time, "s"
for job in jobs:
    print "done!", job()
job_server.print_stats()
```

Usage: pts_vtk_al.py [np][nodes_log_file]
Nodes: nr114, nr113
Data file reading time: 8.20501804352 s
The arrays length is: 17852062
Job Count: 5
The step: 3570413
Indices_1: 1 3570414
Indices_2: 3570414 7140827
Indices_3: 7140827 10711240
Indices_4: 10711240 14281653
Indices_5: 14281653 17852062

NO. Workers	Time elapsed (s)
1	40.2060461044
2	23.5049991608
4	15.9269149303
8	8.40742206573
16	8.38217902184
32	8.40001988411
48	8.26138520241