Parallel Python usage examples

Â

Download all examples Example #1: sum_primes.pyÂ Â

```
  #!/usr/bin/python
#Â File:Â sum_primes.py
#Â Author:Â VItaliiÂ Vanovschi
#Â Desc:Â ThisÂ programÂ demonstratesÂ parallelÂ computationsÂ withÂ ppÂ module
#Â ItÂ calculatesÂ theÂ sumÂ ofÂ primeÂ numbersÂ belowÂ aÂ givenÂ integerÂ inÂ parallel
#Â ParallelÂ PythonÂ Software:Â http://www.parallelpython.com

importÂ math,Â sys,Â time
importÂ pp

defÂ isprime(n):
Â Â Â Â """ReturnsÂ TrueÂ ifÂ nÂ isÂ primeÂ andÂ FalseÂ otherwise"""
Â Â Â Â ifÂ notÂ isinstance(n,Â int):
Â Â Â Â Â Â Â Â raiseÂ TypeError("argumentÂ passedÂ toÂ is_primeÂ isÂ notÂ ofÂ 'int'Â type")
Â Â Â Â ifÂ nÂ <Â 2:
Â Â Â Â Â Â Â Â returnÂ False
Â Â Â Â ifÂ nÂ ==Â 2:
Â Â Â Â Â Â Â Â returnÂ True
Â Â Â Â maxÂ =Â int(math.ceil(math.sqrt(n)))
Â Â Â Â iÂ =Â 2
Â Â Â Â whileÂ iÂ <=Â max:
Â Â Â Â Â Â Â Â ifÂ nÂ %Â iÂ ==Â 0:
Â Â Â Â Â Â Â Â Â Â Â Â returnÂ False
Â Â Â Â Â Â Â Â iÂ +=Â 1
Â Â Â Â returnÂ True

defÂ sum_primes(n):
Â Â Â Â """CalculatesÂ sumÂ ofÂ allÂ primesÂ belowÂ givenÂ integerÂ n"""
Â Â Â Â returnÂ sum([xÂ forÂ xÂ inÂ xrange(2,n)Â ifÂ isprime(x)])

printÂ """Usage:Â pythonÂ sum_primes.pyÂ [ncpus]
Â Â Â Â [ncpus]Â -Â theÂ numberÂ ofÂ workersÂ toÂ runÂ inÂ parallel,Â
Â Â Â Â ifÂ omittedÂ itÂ willÂ beÂ setÂ toÂ theÂ numberÂ ofÂ processorsÂ inÂ theÂ system
"""

#Â tupleÂ ofÂ allÂ parallelÂ pythonÂ serversÂ toÂ connectÂ with
ppserversÂ =Â ()
#ppserversÂ =Â ("10.0.0.1",)

ifÂ len(sys.argv)Â >Â 1:
Â Â Â Â ncpusÂ =Â int(sys.argv[1])
Â Â Â Â #Â CreatesÂ jobserverÂ withÂ ncpusÂ workers
Â Â Â Â job_serverÂ =Â pp.Server(ncpus,Â ppservers=ppservers)
else:
Â Â Â Â #Â CreatesÂ jobserverÂ withÂ automaticallyÂ detectedÂ numberÂ ofÂ workers
Â Â Â Â job_serverÂ =Â pp.Server(ppservers=ppservers)

printÂ "StartingÂ ppÂ with",Â job_server.get_ncpus(),Â "workers"

#Â SubmitÂ aÂ jobÂ ofÂ calulatingÂ sum_primes(100)Â forÂ execution.Â
#Â sum_primesÂ -Â theÂ function
#Â (100,)Â -Â tupleÂ withÂ argumentsÂ forÂ sum_primes
```

```
#Â (isprime,)Â -Â tupleÂ withÂ functionsÂ onÂ whichÂ functionÂ sum_primesÂ depends
#Â ("math",)Â -Â tupleÂ withÂ moduleÂ namesÂ whichÂ mustÂ beÂ importedÂ beforeÂ sum_primesÂ execution
#Â ExecutionÂ startsÂ asÂ soonÂ asÂ oneÂ ofÂ theÂ workersÂ willÂ becomeÂ available
job1Â =Â job_server.submit(sum_primes,Â (100,),Â (isprime,),Â ("math",))

#Â RetrievesÂ theÂ resultÂ calculatedÂ byÂ job1
#Â TheÂ valueÂ ofÂ job1()Â isÂ theÂ sameÂ asÂ sum_primes(100)
#Â IfÂ theÂ jobÂ hasÂ notÂ beenÂ finishedÂ yet,Â executionÂ willÂ waitÂ hereÂ untilÂ resultÂ isÂ available
resultÂ =Â job1()

printÂ "SumÂ ofÂ primesÂ belowÂ 100Â is",Â result

start_timeÂ =Â time.time()

#Â TheÂ followingÂ submitsÂ 8Â jobsÂ andÂ thenÂ retrievesÂ theÂ results
inputsÂ =Â (100000,Â 100100,Â 100200,Â 100300,Â 100400,Â 100500,Â 100600,Â 100700)
jobsÂ =Â [(input,Â job_server.submit(sum_primes,(input,),Â (isprime,),Â ("math",)))Â forÂ inputÂ inÂ inputs]
forÂ input,Â jobÂ inÂ jobs:
Â Â Â Â printÂ "SumÂ ofÂ primesÂ below",Â input,Â "is",Â job()

printÂ "TimeÂ elapsed:Â ",Â time.time()Â -Â start_time,Â "s"
job_server.print_stats()

#Â ParallelÂ PythonÂ Software:Â http://www.parallelpython.com



Â Example #2: reverse_md5.pyÂ    #!/usr/bin/python
#Â File:Â reverse_md5.py
#Â Author:Â VitaliiÂ Vanovschi
#Â Desc:Â ThisÂ programÂ demonstratesÂ parallelÂ computationsÂ withÂ ppÂ module
#Â ItÂ triesÂ toÂ reverseÂ anÂ md5Â hashÂ inÂ parallel
#Â ParallelÂ PythonÂ Software:Â http://www.parallelpython.com

importÂ math,Â sys,Â md5,Â time
importÂ pp

defÂ md5test(hash,Â start,Â end):
Â Â Â Â """CalculatesÂ md5Â ofÂ theÂ integerssÂ betweenÂ 'start'Â andÂ 'end'Â andÂ comparesÂ itÂ withÂ 'hash'"""
Â Â Â Â forÂ xÂ inÂ xrange(start,Â end):
Â Â Â Â Â Â Â Â ifÂ md5.new(str(x)).hexdigest()Â ==Â hash:
Â Â Â Â Â Â Â Â Â Â Â Â returnÂ x

printÂ """Usage:Â pythonÂ reverse_md5.pyÂ [ncpus]
Â Â Â Â [ncpus]Â -Â theÂ numberÂ ofÂ workersÂ toÂ runÂ inÂ parallel,Â
Â Â Â Â ifÂ omittedÂ itÂ willÂ beÂ setÂ toÂ theÂ numberÂ ofÂ processorsÂ inÂ theÂ system
"""

#Â tupleÂ ofÂ allÂ parallelÂ pythonÂ serversÂ toÂ connectÂ with
ppserversÂ =Â ()
#ppserversÂ =Â ("10.0.0.1",)

ifÂ len(sys.argv)Â >Â 1:
Â Â Â Â ncpusÂ =Â int(sys.argv[1])
Â Â Â Â #Â CreatesÂ jobserverÂ withÂ ncpusÂ workers
Â Â Â Â job_serverÂ =Â pp.Server(ncpus,Â ppservers=ppservers)
else:
Â Â Â Â #Â CreatesÂ jobserverÂ withÂ automaticallyÂ detectedÂ numberÂ ofÂ workers
Â Â Â Â job_serverÂ =Â pp.Server(ppservers=ppservers)

printÂ "StartingÂ ppÂ with",Â job_server.get_ncpus(),Â "workers"

#CalculatesÂ md5Â hashÂ fromÂ theÂ givenÂ number
hashÂ =Â md5.new("1829182").hexdigest()
printÂ "hashÂ =",Â hash
```

```
#NowÂ weÂ willÂ tryÂ toÂ findÂ theÂ numberÂ withÂ thisÂ hashÂ value

start_timeÂ =Â time.time()
startÂ =Â 1
endÂ =Â 2000000

#Â SinceÂ jobsÂ areÂ notÂ equalÂ inÂ theÂ executionÂ time,Â divisionÂ ofÂ theÂ problemÂ
#Â intoÂ aÂ 100Â ofÂ smallÂ subproblemsÂ leadsÂ toÂ aÂ betterÂ loadÂ balancing
partsÂ =Â 128

stepÂ =Â (endÂ -Â start)Â /Â partsÂ +Â 1
jobsÂ =Â []

forÂ indexÂ inÂ xrange(parts):
Â Â Â Â startiÂ =Â start+index*step
Â Â Â Â endiÂ =Â min(start+(index+1)*step,Â end)
Â Â Â Â #Â SubmitÂ aÂ jobÂ whichÂ willÂ testÂ ifÂ aÂ numberÂ inÂ theÂ rangeÂ starti-endiÂ hasÂ givenÂ md5Â hash
Â Â Â Â #Â md5testÂ -Â theÂ function
Â Â Â Â #Â (hash,Â starti,Â endi)Â -Â tupleÂ withÂ argumentsÂ forÂ md5test
Â Â Â Â #Â ()Â -Â tupleÂ withÂ functionsÂ onÂ whichÂ functionÂ md5testÂ depends
Â Â Â Â #Â ("md5",)Â -Â tupleÂ withÂ moduleÂ namesÂ whichÂ mustÂ beÂ importedÂ beforeÂ md5testÂ execution
Â Â Â Â jobs.append(job_server.submit(md5test,Â (hash,Â starti,Â endi),Â (),Â ("md5",)))

#Â RetrieveÂ resultsÂ ofÂ allÂ submitedÂ jobs
forÂ jobÂ inÂ jobs:
Â Â Â Â resultÂ =Â job()
Â Â Â Â ifÂ result:
Â Â Â Â Â Â Â Â break

#Â PrintÂ theÂ results
ifÂ result:
Â Â Â Â printÂ "ReverseÂ md5Â for",Â hash,Â "is",Â result
else:
Â Â Â Â printÂ "ReverseÂ md5Â for",Â hash,Â "hasÂ notÂ beenÂ found"

printÂ "TimeÂ elapsed:Â ",Â time.time()Â -Â start_time,Â "s"
job_server.print_stats()

#Â ParallelÂ PythonÂ Software:Â http://www.parallelpython.com
   Example #3: dynamic_ncpus.pyÂ Â

   #!/usr/bin/python
#Â File:Â dynamic_ncpus.py
#Â Author:Â VitaliiÃ Vanovschi
#Â Desc:Â ThisÂ programÂ demonstratesÂ parallelÂ computationsÂ withÂ ppÂ moduleÂ
#Â andÂ dynamicÂ cpuÂ allocationÂ feature.
#Â ProgramÂ calculatesÂ theÂ partialÂ sumÂ 1-1/2+1/3-1/4+1/5-1/6+...Â (inÂ theÂ limitÂ itÂ isÂ ln(2))
#Â ParallelÂ PythonÂ Software:Â http://www.parallelpython.com

importÂ math,Â sys,Â md5,Â time
importÂ pp

defÂ part_sum(start,Â end):
Â Â Â Â """CalculatesÂ partialÂ sum"""
Â Â Â Â sumÂ =Â 0
Â Â Â Â forÂ xÂ inÂ xrange(start,Â end):
Â Â Â Â Â Â Â Â ifÂ xÂ %Â 2Â ==Â 0:
Â Â Â Â Â Â Â Â Â Â Â Â sumÂ -=Â 1.0Â /Â xÂ
Â Â Â Â Â Â Â Â else:
Â Â Â Â Â Â Â Â Â Â Â Â sumÂ +=Â 1.0Â /Â xÂ
Â Â Â Â returnÂ sum

printÂ """Usage:Â pythonÂ dynamic_ncpus.py"""
printÂ
```

```
startÂ =Â 1
endÂ =Â 20000000

#Â DivideÂ theÂ taskÂ intoÂ 64Â subtasks
partsÃ =Â 64
stepÂ =Â (endÂ -Â start)Â /Â partsÂ +Â 1

#Â CreateÂ jobserver
job_serverÂ =Â pp.Server()

#Â ExecuteÂ theÂ sameÂ taskÂ withÂ differentÂ amountÂ ofÂ activeÂ workersÂ andÂ measureÂ theÂ time
forÂ ncpusÂ inÂ (1,Â 2,Â 4,Â 8,Â 16,Â 1):
Â Â Â Â job_server.set_ncpus(ncpus)
Â Â Â Â jobsÂ =Â []
Â Â Â Â start_timeÂ =Â time.time()
Â Â Â Â printÃ "StartingÂ ",Â job_server.get_ncpus(),Â "Â workers"
Â Â Â Â forÂ indexÂ inÂ xrange(parts):
Â Â Â Â Â Â Â Â startiÂ =Â start+index*step
Â Â Â Â Â Â Â Â endiÂ =Â min(start+(index+1)*step,Â end)
Â Â Â Â Â Â Â Â #Â SubmitÃ aÂ jobÂ whichÂ willÂ calculateÂ partialÂ sumÂ
Â Â Â Â Â Â Â Â #Â part_sumÂ -Â theÂ function
Â Â Â Â Â Â Â Â #Â (starti,Â endi)Â -Â tupleÂ withÂ argumentsÂ forÂ part_sum
Â Â Â Â Â Â Â Â #Â ()Â -Â tupleÂ withÂ functionsÂ onÂ whichÂ functionÂ part_sumÂ depends
Â Â Â Â Â Â Â Â #Â ()Â -Â tupleÂ withÂ moduleÂ namesÂ whichÂ mustÂ beÂ importedÂ beforeÂ part_sumÂ execution
Â Â Â Â Â Â Â Â jobs.append(job_server.submit(part_sum,Â (starti,Â endi)))
Â Â Â 
Â Â Â Â #Â RetrieveÂ allÂ theÂ resultsÂ andÂ calculateÂ theirÂ sum
Â Â Â Â part_sum1Â =Â sum([job()Â forÂ jobÂ inÂ jobs])
Â Â Â Â #Â PrintÃ theÂ partialÂ sum
Â Â Â Â printÃ "PartialÂ sumÂ is",Â part_sum1,Â "|Â diffÂ =",Â math.log(2)Â -Â part_sum1

Â Â Â Â printÃ "TimeÂ elapsed:Â ",Â time.time()Â -Â start_time,Â "s"
Â Â Â Â print
job_server.print_stats()

#Â ParallelÂ PythonÂ Software:Â http://www.parallelpython.com
```

Â Example #4: callback.pyÂ Â

```
 #!/usr/bin/python
#Â File:Â callback.py
#Â Author:Â VitaliiÃ Vanovschi
#Â Desc:Â ThisÃ programÂ demonstratesÃ parallelÃ computationsÃ withÃ ppÃ moduleÃ
#Â usingÃ callbacksÃ (availableÃ sinceÃ ppÃ 1.3).
#Â ProgramÂ calculatesÃ theÃ partialÃ sumÂ 1-1/2+1/3-1/4+1/5-1/6+...Â (inÂ theÂ limitÃ itÃ isÃ ln(2))
#Â ParallelÂ PythonÂ Software:Â http://www.parallelpython.com

importÃ math,Â time,Â thread,Â sys
importÃ pp

#classÃ forÃ callbacks
classÃ Sum:
Â Â Â Â defÃ __init__(self):
Â Â Â Â Â Â Â Â self.valueÂ =Â 0.0
Â Â Â Â Â Â Â Â self.lockÂ =Â thread.allocate_lock()
Â Â Â Â Â Â Â Â self.countÂ =Â 0

Â Â Â Â #theÂ callbackÂ function
Â Â Â Â defÃ add(self,Â value):
Â Â Â Â Â Â Â Â #Â weÃ mustÃ useÃ lockÃ hereÃ becauseÃ +=Â isÃ notÃ atomic
Â Â Â Â Â Â Â Â self.countÃ +=Â 1
Â Â Â Â Â Â Â Â self.lock.acquire()
Â Â Â Â Â Â Â Â self.valueÃ +=Â value
Â Â Â Â Â Â Â Â self.lock.release()
```

```
defÂ part_sum(start,Â end):
Â Â Â Â """CalculatesÂ partialÂ sum"""
Â Â Â Â sumÂ =Â 0
Â Â Â Â forÂ xÂ inÂ xrange(start,Â end):
Â Â Â Â Â Â Â Â ifÂ xÂ %Â 2Â ==Â 0:
Â Â Â Â Â Â Â Â Â Â Â Â sumÂ -=Â 1.0Â /Â xÂ
Â Â Â Â Â Â Â Â else:
Â Â Â Â Â Â Â Â Â Â Â Â sumÂ +=Â 1.0Â /Â xÂ
Â Â Â Â returnÂ sum

printÂ """Usage:Â pythonÂ callback.pyÂ [ncpus]
Â Â Â Â [ncpus]Â -Â theÂ numberÂ ofÂ workersÂ toÂ runÂ inÂ parallel,Â
Â Â Â Â ifÂ omittedÂ itÂ willÂ beÂ setÂ toÂ theÂ numberÂ ofÂ processorsÂ inÂ theÂ system
"""

startÂ =Â 1
endÂ =Â 20000000

#Â DivideÂ theÂ taskÂ intoÂ 128Â subtasks
partsÂ =Â 128
stepÂ =Â (endÂ -Â start)Â /Â partsÂ +Â 1

#Â tupleÂ ofÂ allÂ parallelÂ pythonÂ serversÂ toÂ connectÂ with
ppserversÂ =Â ()
#ppserversÂ =Â ("localhost",)

ifÂ len(sys.argv)Â >Â 1:
Â Â Â Â ncpusÂ =Â int(sys.argv[1])
Â Â Â Â #Â CreatesÂ jobserverÂ withÂ ncpusÂ workers
Â Â Â Â job_serverÂ =Â pp.Server(ncpus,Â ppservers=ppservers)
else:
Â Â Â Â #Â CreatesÂ jobserverÂ withÂ automaticallyÂ detectedÂ numberÂ ofÂ workers
Â Â Â Â job_serverÂ =Â pp.Server(ppservers=ppservers)

printÂ "StartingÂ ppÂ with",Â job_server.get_ncpus(),Â "workers"

#Â CreateÂ an instanceÂ ofÂ callbackÂ class
sumÂ =Â Sum()

#Â ExecuteÂ theÂ sameÂ taskÂ withÂ differentÂ amountÂ ofÂ activeÂ workersÂ andÂ measureÂ theÂ time
start_timeÂ =Â time.time()
forÂ indexÂ inÂ xrange(parts):
Â Â Â Â startiÂ =Â start+index*step
Â Â Â Â endiÂ =Â min(start+(index+1)*step,Â end)
Â Â Â Â #Â SubmitÂ aÂ jobÂ whichÂ willÂ calculateÂ partialÂ sumÂ
Â Â Â Â #Â part_sumÂ -Â theÂ function
Â Â Â Â #Â (starti,Â endi)Â -Â tupleÂ withÂ argumentsÂ forÂ part_sum
Â Â Â Â #Â callback=sum.addÂ -Â callbackÂ function
Â Â Â Â job_server.submit(part_sum,Â (starti,Â endi),Â callback=sum.add)
Â Â
#waitÂ forÂ jobsÂ inÂ allÂ groupsÂ toÂ finishÂ
job_server.wait()
Â Â Â Â 
#Â PrintÂ theÂ partialÂ sum
printÂ "PartialÂ sumÂ is",Â sum.value,Â "|Â diffÂ =",Â math.log(2)Â -Â sum.value

job_server.print_stats()

#Â ParallelÂ PythonÂ Software:Â http://www.parallelpython.com

Â Example #5: auto_diff.pyÂ Â  #!/usr/bin/python
#Â File:Â auto_diff.py
#Â Author:Â VitaliiÂ Vanovschi
#Â Desc:Â ThisÂ programÂ demonstratesÂ parallelÂ computationsÂ withÂ ppÂ moduleÂ
#Â usingÂ classÂ methodsÂ asÂ parallelÂ functionsÂ (availableÂ sinceÂ ppÂ 1.4).
```

```
# Program calculates the partial sums of f(x) = x-x**2/2+x**3/3-x**4/4+...
# and first derivatives f'(x) using automatic differentiation technique.
# In the limit f(x) = ln(x+1) and f'(x) = 1/(x+1).
# Parallel Python Software: http://www.parallelpython.com

import math, sys
import pp

# Partial implemenmtation of automatic differentiation class
class AD:
    def __init__(self, x, dx=0.0):
        self.x = float(x)
        self.dx = float(dx)

    def __pow__(self, val):
        if isinstance(val, int):
            p = self.x**val
            return AD(self.x**val, val*self.x**(val-1)*self.dx)
        else:
            raise TypeError("Second argumnet must be an integer")

    def __add__(self, val):
        if isinstance(val, AD):
            return AD(self.x+val.x, self.dx+val.dx)
        else:
            return AD(self.x+val, self.dx)
    def __radd__(self, val):
        return self+val

    def __mul__(self, val):
        if isinstance(val, AD):
            return AD(self.x*val.x, self.x*val.dx+val.x*self.dx)
        else:
            return AD(self.x*val, val*self.dx)

    def __rmul__(self, val):
        return self*val
    def __div__(self, val):
        if isinstance(val, AD):
            return self*AD(1/val.x, -val.dx/val.x**2)
        else:
            return self*(1/float(val))

    def __rdiv__(self, val):
        return AD(val)/self
    def __sub__(self, val):
        if isinstance(val, AD):
            return AD(self.x-val.x, self.dx-val.dx)
        else:
            return AD(self.x-val, self.dx)

    def __repr__(self):
        return str((self.x, self.dx))

# This class contains methods which will be executed in parallel
class PartialSum:
    def __init__(self, n):
        self.n = n

    #truncated natural logarithm
    def t_log(self, x):
        return self.partial_sum(x-1)
```

```
Â Â Â Â #partialÂ sumÂ forÂ truncatedÂ naturalÂ logarithm
Â Â Â Â defÂ partial_sum(self,Â x):
Â Â Â Â Â Â Â Â returnÂ sum([float(i%2Â andÂ 1Â orÂ -1)*x**i/iÂ forÂ iÂ inÂ xrange(1,self.n)])

printÂ """Usage:Â pythonÂ auto_diff.pyÂ [ncpus]
Â Â Â Â [ncpus]Â -Â theÂ numberÂ ofÂ workersÂ toÂ runÂ inÂ parallel,Â
Â Â Â Â ifÂ omittedÂ itÂ willÂ beÂ setÂ toÂ theÂ numberÂ ofÂ processorsÂ inÂ theÂ system
"""

#Â tupleÂ ofÂ allÂ parallelÂ pythonÂ serversÂ toÂ connectÂ with
ppserversÂ =Â ()
#ppserversÂ =Â ("10.0.0.1",)

ifÂ len(sys.argv)Â >Â 1:
Â Â Â Â ncpusÂ =Â int(sys.argv[1])
Â Â Â Â #Â CreatesÂ jobserverÂ withÂ ncpusÂ workers
Â Â Â Â job_serverÂ =Â pp.Server(ncpus,Â ppservers=ppservers)
else:
Â Â Â Â #Â CreatesÂ jobserverÂ withÂ automaticallyÂ detectedÂ numberÂ ofÂ workers
Â Â Â Â job_serverÂ =Â pp.Server(ppservers=ppservers)

printÂ "StartingÂ ppÂ with",Â job_server.get_ncpus(),Â "workers"

procÂ =Â PartialSum(20000)

resultsÂ =Â []
forÂ iÂ inÂ range(32):
Â Â Â Â #Â CreatesÂ anÂ objectÂ withÂ xÂ =Â float(i)/32+1Â andÂ dxÂ =Â 1.0
Â Â Â Â ad_xÂ =Â AD(float(i)/32+1,Â 1.0)
Â Â Â Â #Â SubmitsÂ aÂ jobÂ ofÂ calulatingÂ proc.t_log(x).Â
Â Â Â Â fÂ =Â job_server.submit(proc.t_log,Â (ad_x,))
Â Â Â Â results.append((ad_x.x,Â f))

forÂ x,Â fÂ inÂ results:
Â Â Â Â #Â RetrievesÂ theÂ resultÂ ofÂ theÂ calculation
Â Â Â Â valÂ =Â f()
Â Â Â Â printÂ "t_log(%lf)Â =Â %lf,Â t_log'(%lf)Â =Â %lf"Â %Â (x,Â val.x,Â x,Â val.dx)

#Â PrintÂ executionÂ statistics
job_server.print_stats()

#Â ParallelÂ PythonÂ Software:Â http://www.parallelpython.com
```