

High Productivity in Parallel Programming and Computing with Python

Benefits of the Star-P Software Platform.

The Star-P software from Interactive Supercomputing enables parallel computing with Python on SMP machines or distributed x86/64 clusters for scientific, financial analysis, and engineering applications. This note focuses on one of the benefits of Star-P software platform: simplicity and efficiency of programming parallel applications. A comparison of Star-P/Python code with an equivalent code based on an open-source parallel Python programming environment is provided as an illustration.

The main benefits of using the Star-P software platform come from supporting both “Task Parallel” and “Data Parallel” computing with Python, as well as from the underlying richness of functionality (see Appendix A for a definition of Task and Data parallel).

The example used here follows example #4 published on the open-source site www.parallepython.com. ISC is grateful to the author(s) of this example for providing it in public domain.

The quoted code and the two Star-P/Python examples below all compute, in parallel, the partial sum $1-1/2+1/3-1/4+1/5-1/6+\dots$

By simply counting the lines of code (excluding comment lines) we see that:

- The quoted parallel computing Python code consists of 42 lines of code
- Star-P for Python code consists of 22 lines of code in the mixed task-parallel and data-parallel version
- Star-P for Python code consists of just 9 lines of code for the pure data-parallel version

First, we quote the open-source code example:

Parallel Python - <http://www.parallepython.com/content/view/17/31/> example 4

```
#!/usr/bin/python
# File: callback.py
# Author: Vitalii Vanovschi
# Desc: This program demonstrates parallel computations with pp module
# using callbacks (available since pp 1.3).
# Program calculates the partial sum  $1-1/2+1/3-1/4+1/5-1/6+\dots$  (in the limit it is  $\ln(2)$ )
# Parallel Python Software: http://www.parallepython.com

import math, time, thread, sys
import pp

#class for callbacks
class Sum:
    def __init__(self):
        self.value = 0.0
        self.lock = thread.allocate_lock()
```

```
        self.count = 0

    #the callback function
    def add(self, value):
        # we must use lock here because += is not atomic
        self.count += 1
        self.lock.acquire()
        self.value += value
        self.lock.release()

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 callback.py [ncpus]
       [ncpus] - the number of workers to run in parallel,
       if omitted it will be set to the number of processors in the system
       """

start = 1
end = 20000000

# Divide the task into 128 subtasks
parts = 128
step = (end - start) / parts + 1

# tuple of all parallel python servers to connect with
ppservers = ()
#ppservers = ("localhost",)

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"

# Create anm instance of callback class
sum = Sum()

# Execute the same task with different amount of active workers and measure the time
start_time = time.time()
for index in xrange(parts):
    starti = start+index*step
    endi = min(start+(index+1)*step, end)
    # Submit a job which will calculate partial sum
    # part_sum - the function
    # (starti, endi) - tuple with arguments for part_sum
    # callback=sum.add - callback function
    job_server.submit(part_sum, (starti, endi), callback=sum.add)
```

```
#wait for jobs in all groups to finish
job_server.wait()

# Print the partial sum
print "Partial sum is", sum.value, "| diff =", math.log(2) - sum.value

job_server.print_stats()

# Parallel Python Software: http://www.parallelpython.com
```

Star-P task and data parallel mix

Next, let's look at the Star-P for Python code, a version in which both task-parallel and data-parallel computing is used. Note that the code line

```
y = starp.ppeval(part_sum, startarray, endarray)
```

effects task-parallel computing, while the code line

```
result = starp.sum(y)
```

effects data-parallel computing.

This code example consists of 22 lines of code.

```
import starp, time, math

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

def parallel():
    starp.defaultConnect('myhost', '/starp/server/path', numProcs=8)
    start = 1
    end = 20000000
    step = 100000

    slices = range(start, end, step)
    startarray = starp.array(slices)
    slices.remove(start)
    slices.append(end)
    endarray = starp.array(slices)

    y = starp.ppeval(part_sum, startarray, endarray)
    result = starp.sum(y)

    print "Partial sum is", result, "| diff =", math.log(2) - result
```

Star-P data parallel

Finally, let's look at the Star-P for Python code, a version in which all computation is done in data-parallel mode.

This code example consists of only 9 lines of code.

```
import starp, time, math

starp.defaultConnect('myhost', '/starp/server/path', numProcs=8)

start = 1
end = 20000000

x = starp.array(range(start, end))
y = 2*(x%2)-1
z = y/x
result= starp.sum(z)

print "Partial sum is", result, "| diff =", math.log(2) - result
```

Conclusion

As illustrated above, substantial savings (approximately 50-80%) in code complexity can be achieved for parallel programming in Python using Star-P software platform.

Appendix A

1. “Task Parallel” computations: Task parallelism (sometimes called "coarse-grained" or "embarrassingly parallel") is a powerful method to carry out many independent calculations in parallel, such as Monte Carlo simulations, or "un-rolling" serial FOR loops. For example, in a medical application involving image processing on multiple brain slices, Star-P can distribute the images across several processors, and simultaneously process them.

2. “Data Parallel” computations: Data parallelism (sometimes called "global array syntax") is used for high-level matrix and vector operations on large data sets. This is critical for many of today's toughest computational problems. (For example, a 10 MB data file generated by airborne radar today may swell to a terabyte-sized data set generated by an array of satellites.)