Parallel Optimization for Simulation Data Based on "DASK"

Jiang Wang

MACSS Project Proposal

Apr 4th, 2018

Github: https://github.com/Otamio/DASKopt

Introduction

- Why do we need parallel computing?
- Why choose to parallel optimization based on “DASK”?
- Are there any needs to improve the “DASK” Module?
- Goal: Improve parallel performance for large scale simulation data by balancing and optimizing the “DASK” scheduler

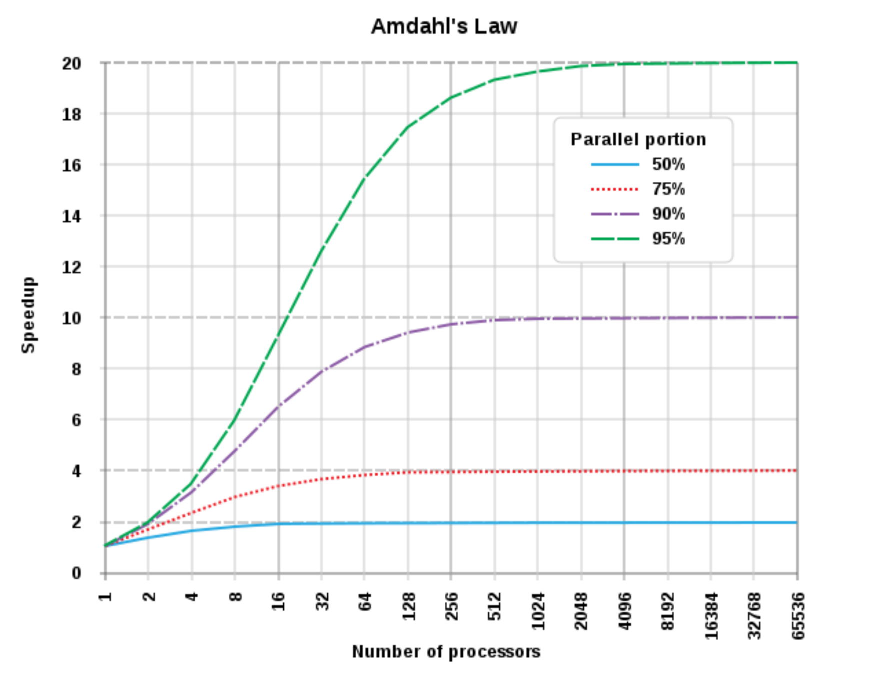
Basic Theory of Parallel Computing

- Amdahl's Law?

$$S_{\text{latency}}(s) = \frac{1}{1 - p + \frac{p}{s}}$$

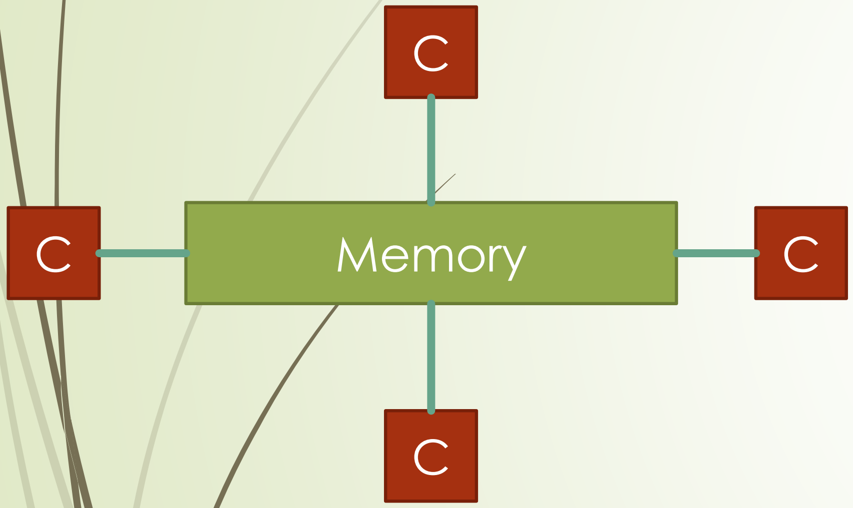
  - p: Execution time of parallelizable part
  - s: Speed up of the parallelized part
  - S: Speed up of the whole part

  - Let p=.8, s=4, we would have S=2.5
  - Let p=0.9, s =16, we would have S=6.4
  - Normally, S would reach a ceiling of 20x

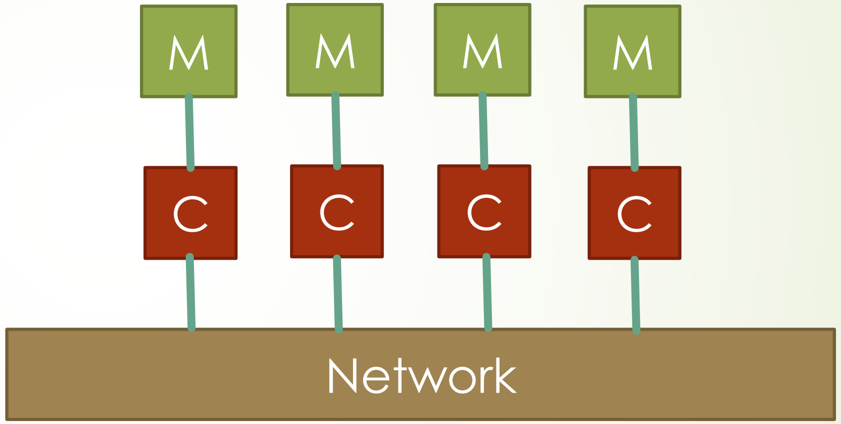


Basic Theory of Parallel Computing

- Memory Management and Communicating Cost



Shared Memory



Distributed Memory

- Modern architecture takes a blend of two
- Transferring data between processes is costly!
- We should prevent unnecessary data transferring, especially when doing cheap operations

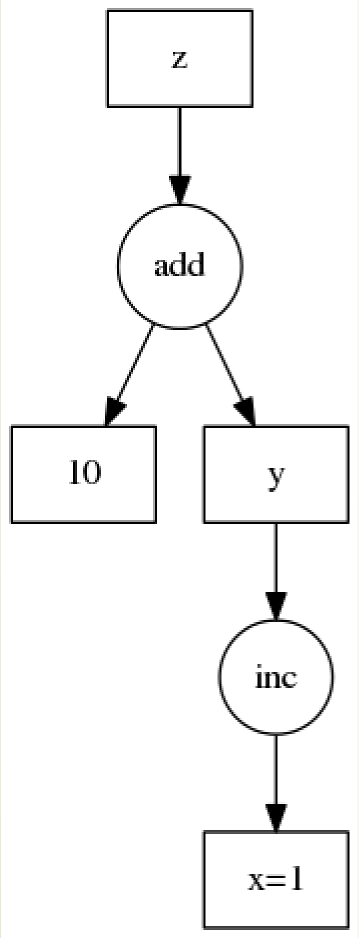
Methods: The "DASK" Scheduler

- Scheduler: The work needs to be serialized and distributed to different processes before they are executed
- DASK Scheduler is a graph
  - E.g.

```python
def inc(i):
    return i + 1

def add(a, b):
    return a + b

x = 1
y = inc(x)
z = add(y, 10)
```

  - Which is encoded as a Python Dictionary:

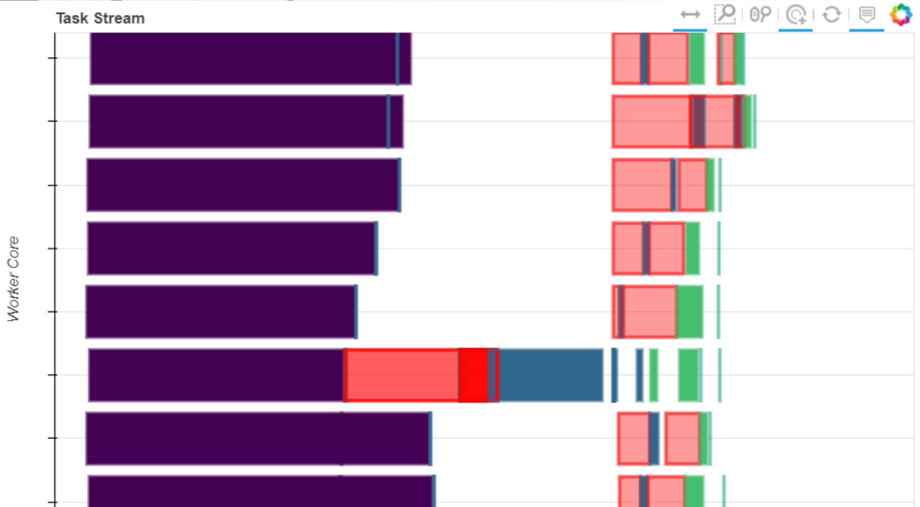
```python
d = {'x': 1,
     'y': (inc, 'x'),
     'z': (add, 'y', 10)}
```



Methods: The "DASK" Scheduler

- Some Issues with the default "DASK" Scheduler
  - Python GIL (Global Interpreter Lock)
    - A "Pain" of CPython when computing with Python data structures
    - Consider the following code:

```
a = da.random.random(size=(10000, 1000), chunks=(1000, 1000))
q, r = da.linalg.qr(a)
a2 = q.dot(r)

out = a2.compute()
```



- We have the following performance analysis:
  - Purple: da.random.random()
  - Red: Transferring Cost
  - Blue: da.linalg.qr()
  - Green: da.array.dot()

Methods: The "DASK" Scheduler

- Data Simulation: Get annual income 40 years later
  - Compare "DASK" and numpy. Which is faster? (Use %time callable)

```
def simulation_dask(init, years):
    y = da.from_array(np.log(init), chunks=(1000000, 1))
    for i in range(years+1):
        n_errors = da.random.normal(0, 0.1, size=(1000000,1), chunks=(1000000, 1))
        y = (1 - 0.2) * (np.log(init) + 0.03 * i) + 0.2 * y + n_errors
    y = da.exp(y)
    return y
```

```
def simulation_numpy(init, years):
    y = np.full((1000000, 1), np.log(init))
    for i in range(years+1):
        n_errors = np.random.normal(0, 0.1, (1000000, 1))
        y = (1 - 0.2) * (np.log(init) + 0.03 * i) + 0.2 * y + n_errors
    y = np.exp(y)
    return y
```

Methods: The "DASK" Scheduler

- Result:

```
%time simulation_numpy(80000, 40).mean()
```

```
Wall time: 3.03 s

264957.69206603663
```

```
%time simulation_dask(80000, 40).mean().compute()
```

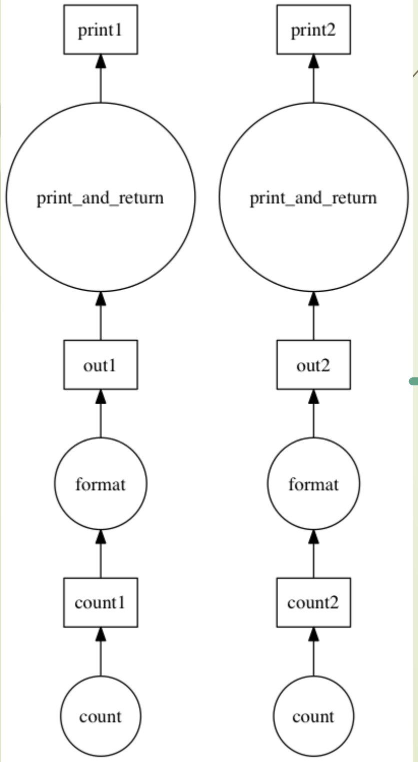
```
Wall time: 5.23 s

264970.07779333438
```

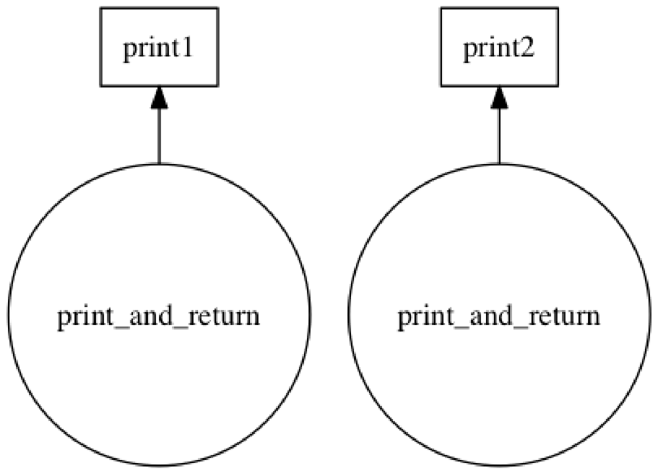
- Reasons?
  - Dependency

```
def simulation_dask(init, years):
    y = da.from_array(np.log(init), chunks=(1000000, 1))
    for i in range(years+1):
        n_errors = da.random.normal(0, 0.1, size=(1000000,1), chunks=(1000000, 1))
        y = (1 - 0.2) * (np.log(init) + 0.03 * i) + 0.2 * y + n_errors
    y = da.exp(y)
    return y
```

Methods: Graph Optimization

- Some possible solutions:
  - Reduce Dependency (Avoid recursion and iteration).
  - Fuse






```
from dask.optimization import fuse
dsk4, dependencies = fuse(dsk3)
results = get(dsk4, outputs)
```

  - Advanced: User defined graph (using dictionaries)

Conclusion

- Research Question: Improve DASK's efficiency for large scale data analysis in simulation research
  - Most emphasis will be placed on optimizing the DASK scheduler
- Schedule:
  - A mini Python project to be completed reducing the data transferring costs between processes
  - A project (mostly Python) to deal with schedulers, dependencies, GIL, and possibly, code rewrite and compilers
- References:
  - http://dask.pydata.org/en/latest/docs.html
- Slides and some example codes will be posted on https://github.com/Otamio/DASKopt
- Thanks for your attention!