1 Flynn's taxonomy

a. Single Instruction, Single Data stream (SISD)
b. Single Instruction, Multiple Data streams (SIMD)
c. Multiple Instruction, Single Data stream (MISD)
d. Multiple Instruction, Multiple Data streams (MIMD)

2 Speedup

The speedup ratio is an accelerating ration when apply parallelism to the problem. It is defined by

$$S = \frac{T_s}{T_p}$$

, where T_s the processing time of the program is implemented using normal serial programming and T_p is processing time of parallel version.

2.1 Linear speedup

In theory, when a number of processing core increase, we should expect to see T_p reducing linearly respect the number of cores. The linear speedup estimates

$$T_p = \frac{T_s}{N}$$

, where N in a number of processing cores.

However, in practical we cannot make a perfect parallelization. The process usually has an initialization process and data transferring in serial sub-processes. The primarily sub-process required in the process is also called "parallel overhead". This overhead process typically necessarily implemented in serial program. Therefore the processing time of the parallel program can be written as

$$T_p = \frac{T_s}{N} + T_o$$

, where T_o is the processing time of the overhead processes.

2.2 Amdahl's Law

In the 1960s, Gene Amdahl studied parallelism and found that in a program some parts of the program are able to parallelize but there must be some serial program remaining in the parallelized version. For example, optimized parallel program can have 90% processed in parallel and 10% must be left in serial program. So we can estimate the processing time of the optimized program running in parallel mode as

$$T_p = \frac{P.T_s}{N} + (1-P).T_s$$

$$\frac{T_p}{T_s} = P/N + (1-P)$$

$$S = \frac{1}{P/N + (1-P)}$$

3

4 IPython.parallel

There are so many ways to program a computer system to perform parallel task. One of many useful tool is IPython.parallel. Unlike OpenMP that is designed for single machine that has many processing cores. IPython.parallel is able to handle programming in a large computer cluster. Here is the example code to test code in a cluster or multicore machine. For more information about cluster setting, please check the detail in my blog (wasit7.blogspot.com).

```
####test ipython.parallel
# to start ipcluster use
# $ipcluster start -n 4

##create clients
from IPython import parallel
c = parallel.Client(packer='pickle')
c.block = True
print(c.ids)

##create direct view
dview = c.direct_view()
dview.block = True
print(dview)

dview.execute('import numpy as np')

##using execute
dview.execute('a=np.random.randint(0,5,size=(1,2))')
print("a:\n{}".format(dview.gather('a')))

##using run
f=open('iptest.py','w')
f.write('import numpy as np\nb=np.zeros(shape=(1,4),dtype=np.int32)')
f.close()
dview.run('iptest.py')
print("b:\n{}".format(dview.gather('b')))
print type(dview.gather('b')[0,0])

##using dictionary
dview['c']=np.ones(shape=(2,3),dtype=np.int8)
print("a:\n{}".format(dview.gather('c')))
print type(dview.gather('c')[0,0])
```