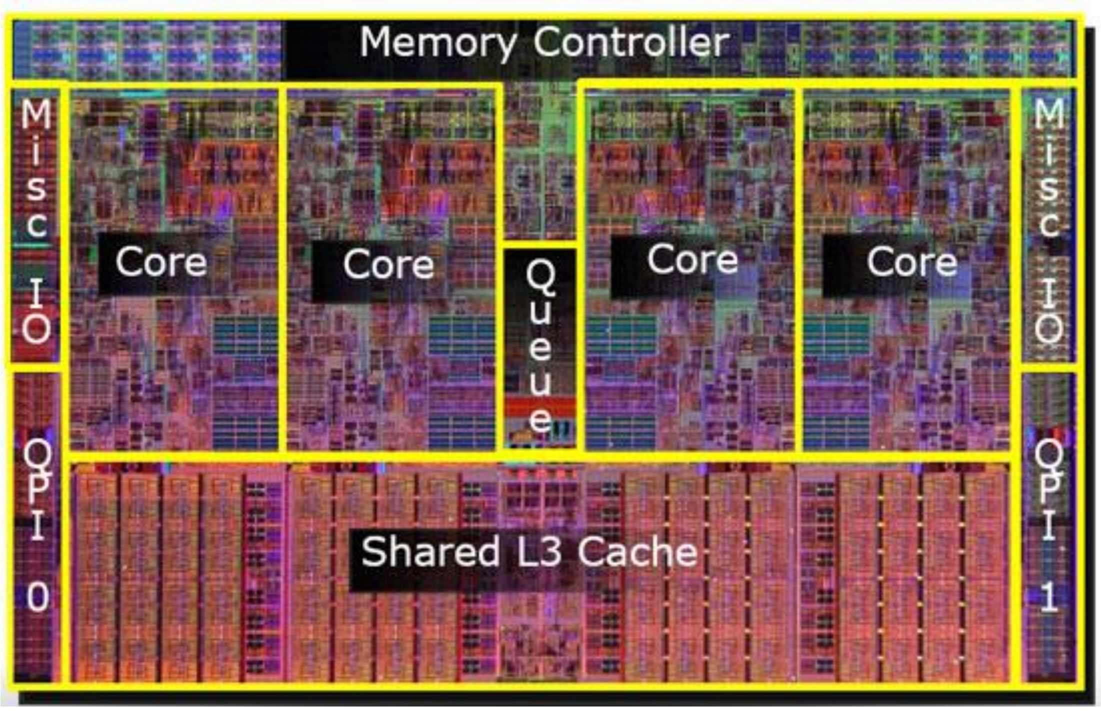
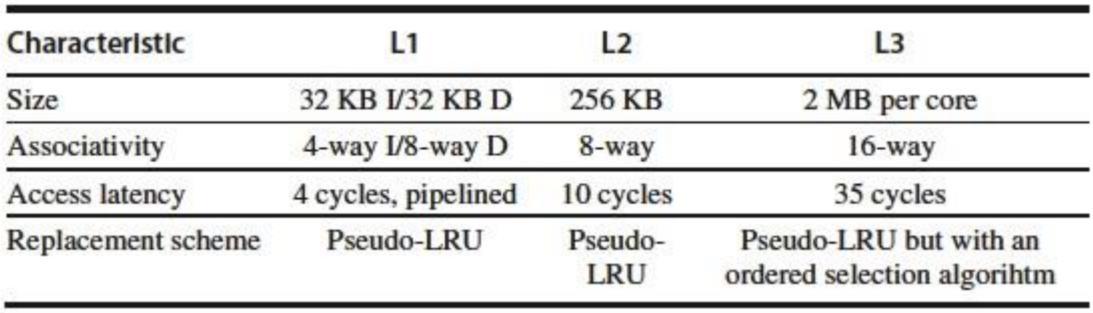
1 CPU Cache

Cache is local memory to store some data that is used often. In a program, some variable is read and updated more often than other. It is reasonable to make a small size of memory near the processor core. The image below show silicon layout of an Intel core i7 that has level 3 cache. The L3 cache in the model allow all cores to read and write. So it is labelled as “shared”. The L1 and L2 are not shown in the image, however they are built very near to the cores. Each core has local L1 and L2 caches.



1.1 Cache level

The memory size of L1 is usually small and quickest to be accessed. As you can see from the table L1 is smallest and fastest while the L3 large and slow.



Characteristic	L1	L2	L3
Size	32 KB I/32 KB D	256 KB	2 MB per core
Associativity	4-way I/8-way D	8-way	16-way
Access latency	4 cycles, pipelined	10 cycles	35 cycles
Replacement scheme	Pseudo-LRU	Pseudo-LRU	Pseudo-LRU but with an ordered selection algorihtm

1.2 Cache blocks

A cache block is a collection of memory in cache (size is normally 64 byte). Cache memory is designed to be transferred as a block instead of by individual. Analogy is like a van that can carry many passengers in a single trip and it is designed to carry about 10 people. If the passenger less than capacity the transfer rate and utility will drop.

1.3 Cache hit and cache miss

The idea of caching memory is to temporarily map some fraction of large memory to reside very close to the cores. However the cache size is very small and a caching algorithm sometimes makes a wrong prediction to not store some fragments of data that is being used. Therefore the request for core to higher level of cache will be execute and this process take longer access latency. The problem is called “cached miss”. In a small program with a small number of local variables, most of variable can be fit inside local cache and this is called “cache hit”.

2 IPython.parallel

IPython is an interactive Python and also provide functionality to program a parallel computing task. The task can be divided and distributed to multiple machines in a cluster.

Example 3 patterns to access variables inside the engine

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

  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])
```

3 Start Cluster

1. install IPython please check http://wasit7.blogspot.com/
2. it is better to make a snapshot at this point
3. create a IPython.parallel profile

```
            $ipython profile create --parallel -profile=[profile_name]
```

4. find profile/ipcontroller_config.py and edit

```
            c = get_config()
            c.HubFactory.ip='*'
```

5. start a controller with --ip=[contoller_ip]

```
            $ipcontroller --ip=10.0.1.74
```

6. copy ipcontroller-engine.json from controller (10.0.1.74) to all engines (10.0.1.90 and the rest) by executing a script

```
$scp /home/ubuntu/.config/ipython/profile_default/security/ipcontroller-
   engine.json
   ubuntu@10.0.1.90:/home/ubuntu/.config/ipython/profile_default/security
   /
```

7. start engine by

```
             $ipcluster engines
```

or

```
             $ipcluster engines -n=8
```

A video of setting the cluster can be found from

https://www.youtube.com/watch?v=7uZpEoLWhb4

4 Compute Pi again with IPython parallel

There are two pieces of code main.py and N_in.py. The main.py is executed on the controller machine and the N_in.py is executed by the code `dview.run('N_in.py')` . And during the process an instruction can be injected to the engine by using `dview.execute('N_total=%d'%(gN_total))` . The results from engines can be collected by `gN_in = dview.gather('N_in')` .

Exanple:

main.py

```
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)
gN_total=4000000

#begin parallel

dview.execute('N_total=%d'%(gN_total))
dview.run('N_in.py')
gN_in = dview.gather('N_in')
#end parallel

import numpy as np
pi=float(np.sum(gN_in))*4.0/float(len(c.ids)*gN_total)
print pi
```

N_in.py

```
import numpy as np
x=np.random.rand(N_total)
y=np.random.rand(N_total)
d_sq=x*x+y*y
N_in=np.sum(d_sq<1.0)
```