# Neural Network Optimization Using Parallel Computing

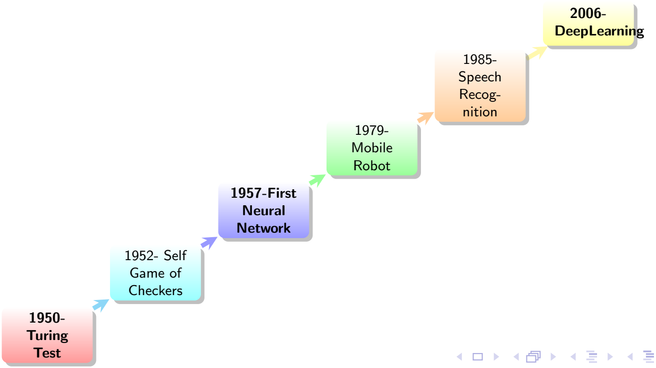
## Project for Master's Degree in HPC

Saumya Bhatnagar

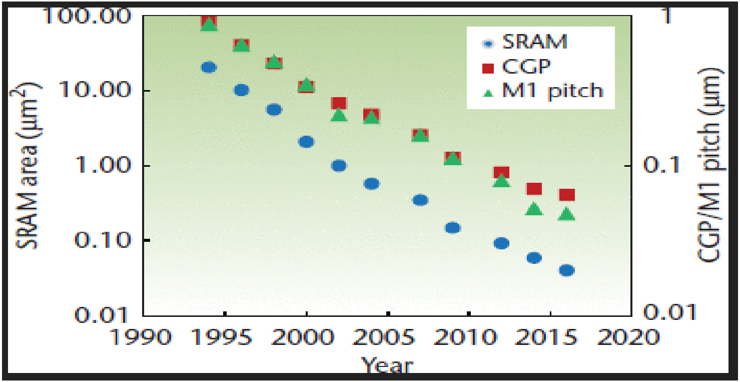
Trinity College Dublin

October 2, 2018

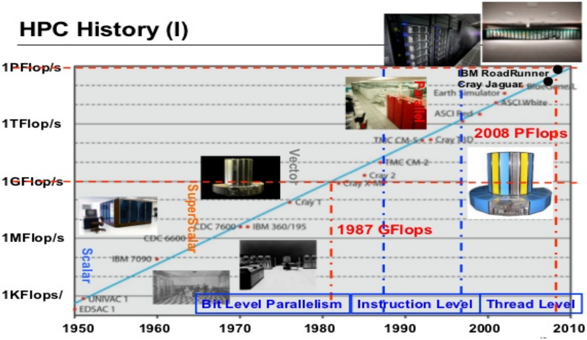
## Fourth Industrial Revolution & Evolution of Machine Learning



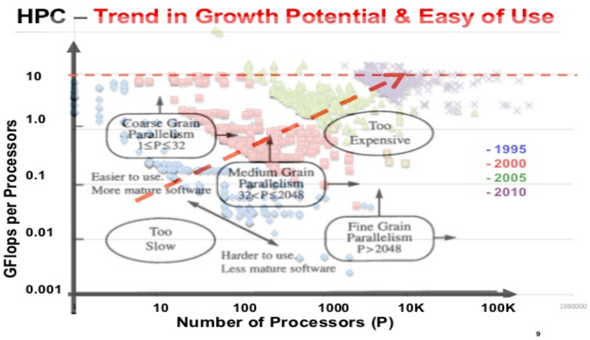
- Fourth Industrial Revolution & Evolution of Machine Learning
- End of Moore's law and Quantum Mechanics(1)



- Fourth Industrial Revolution & Evolution of Machine Learning
- End of Moore's law and Quantum Mechanics(1)
- Evolution of HPC (2)

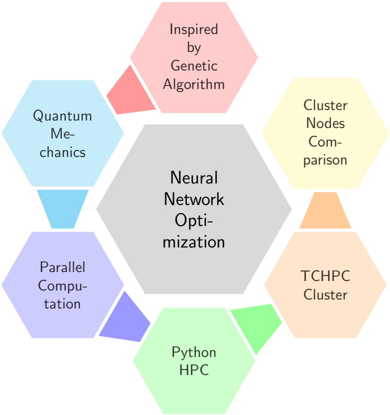


- Fourth Industrial Revolution & Evolution of Machine Learning
- End of Moore's law and Quantum Mechanics(1)
- Evolution of HPC (2)
- Trends of HPC (2)



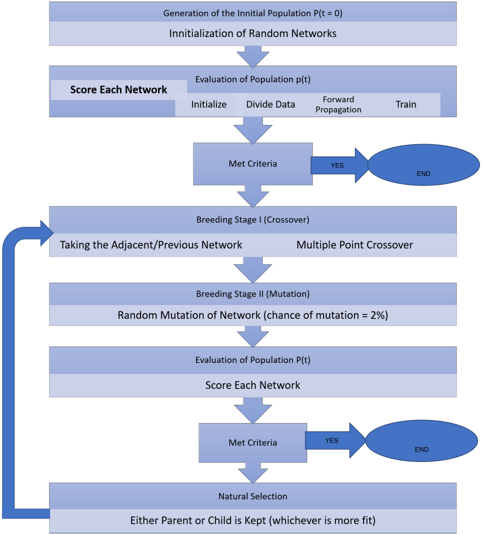
# Table of Contents

1. Compared accuracy of Neural Nets
2. Single Population Fine Grained Algorithms
3. Multiple-population Coarse Grained Algorithms(3)
4. Sequential Time Consumption
5. Mpi4py, Keras, Tensorflow
6. Analyzed TCHPC Clusters
7. Memory Leak



- To access best set of hyper-parameters
- Convolutional Neural Network
- Image Processing
- Machine Learning Benchmark : CIFAR10
- Initiate with randomly selected combination of hyper-parameters
- Sample space kept same with every selection
- MPI parallelism
- Scheduling, Re-Scheduling by resuming the code based on Node availability using Slurm Workload Manager
- Keras has built-in support for multi-GPU data parallelism
- Boyle Cluster: ldd (GNU libc) 2.17

## Optimization Method



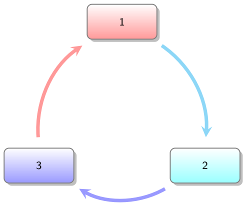

- Inspired by GA (4)
- To crossbreed networks
- Relative accuracy of hyper-parameters set is analyzed

| Dataset | Data-points | Sequential Time (per generation) |
|---|---|---|
| CIFAR10 | 10k | 15 minutes |
| 80M Tiny Images | 80M | 20k minutes |

Number of Processors<32 =>Implemented Course Grained Parallelism (6)

Two Parallel Implementations of GA

1. **Single Population Fine Grained Algorithms** : using 4, 6, 8, 10, 12 processors

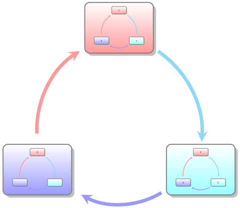


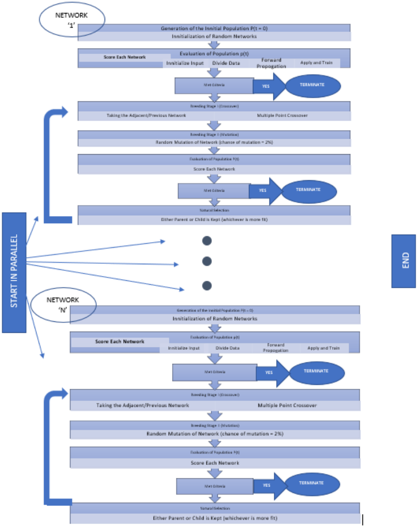
| Dataset | Data-points | Sequential Time (per generation) |
|---|---|---|
| CIFAR10 | 10k | 15 minutes |
| 80M Tiny Images | 80M | 20k minutes |

Number of Processors$<$32 $=>$Implemented Course Grained Parallelism (6)

Two Parallel Implementations of GA

1. **Single Population Fine Grained Algorithms** : using 4, 6, 8, 10, 12 processors
2. **Multiple-population Coarse Grained Algorithms** : with inter-island(every 5 generations) and intra-island breeding
   - population size of 4 in each island, using 12 processors






**Single Population Fine Grained Algorithms**

Send/Recv Model
One Network Per Node
Inter-Network Crossover

**Multiple-population Coarse Grained Algorithms**

MPI Communicators
One Communicator Per Island
Inter-Island & Intra-Island Crossover

## MPI Initialization Testing

```
MPI.Init()
print(MPI.Is_initialized())
print(MPI.Is_finalized())
```

## Non-Blocking Exchange

```
def nonBlockingExchange(self, data):
    reqSend1 = self.comm.isend(data,
            dest=((self.size+self.rank+1)\%self.size),
            tag=self.rank)
    reqRecv2 = self.comm.irecv(source=
            ((self.size+self.rank-1)\%self.size),
            tag=self.rank-1)
    dataPrev = reqRecv2.wait()
    reqSend1.wait()
    return dataPrev
```

## MPI Gather

```
recvdata = self.comm.Gather(data, root = 0)
```

## MPI Broadcast

```
data = self.comm.bcast(data, root = 0)
```

## Get Island

```
self.subGroup = self.rank // groupSize
self.subComm = MPI.Comm.Split(
        self.comm, self.subGroup, self.rank)
```

## Get Island Details

```
self.subSize, self.subRank =
        self.subComm.Get_size(), self.subComm.Get_rank()
```

Qubit in super-position state, increases sample space(5)
Hadamard gate

```
r2=math.sqrt(2.0)
h=np.array([[1/r2, 1/r2],[1/r2,-1/r2]])
```

Rotation Q-gate

```
rot1=float(math.cos(theta));rot2=-float(math.sin(theta))
rot3=float(math.sin(theta));rot4=float(math.cos(theta));
```

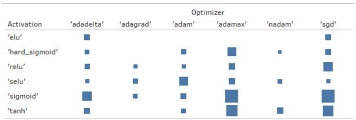
Quantum Population Vector

```
# alpha squared
self.qpv[i,j,0]=np.around(2*pow(AlphaBeta[0],2),2)
# beta squared
self.qpv[i,j,1]=1-self.qpv[i,j,0]
```

Quantum Measure

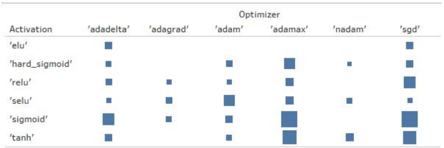
- Search space consists of 5 hyper-parameters: activation functions, optimizers, hidden layers, nodes and dropout (7)
- Six activation functions included are: sigmoid, elu, selu, relu, tanh, hard_sigmoid
- Six Optimizers include sgd, adagrad, adadelta, adam, adamax, nadam.
- Hidden layers range from 1-15.
- Nodes/neurons range from 4-128.
- Dropouts range from 0.1 to 0.5.

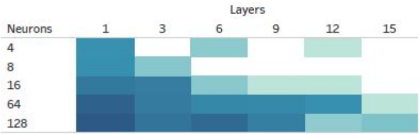
# Samples tested




- Activation functions: hard_sigmoid and elu couldn't come in any set of combinations with the optimizer - adagrad
- Uneven Testing: some combinations have been tested more frequently

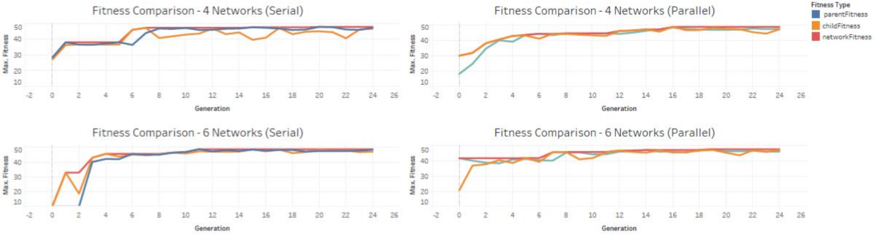
# Samples tested



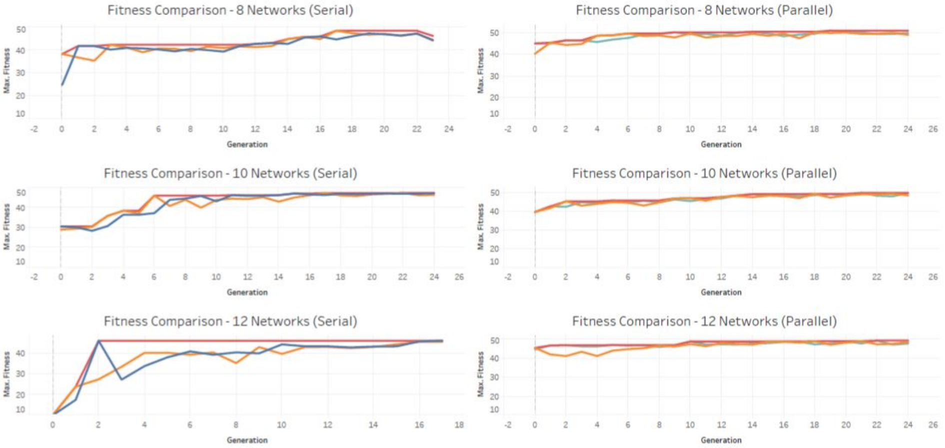

- Activation functions: hard_sigmoid and elu couldn't come in any set of combinations with the optimizer - adagrad
- Uneven Testing: some combinations have been tested more frequently
- Increasing the number of networks, increased the possible combinations
- Higher neurons and lower layers factored in high accuracies

# Fitness Comparison

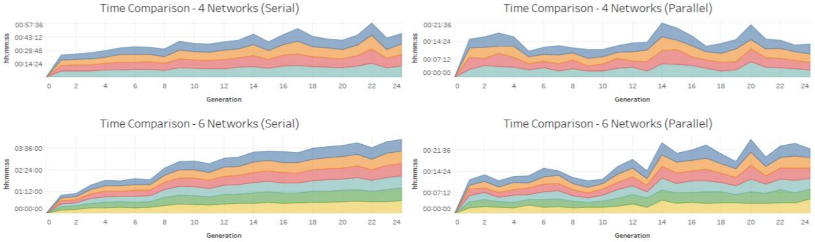



The convergence of parallel implementations is better than that of sequential implementation.

## Sequential Algorithm vs Parallel Algorithm

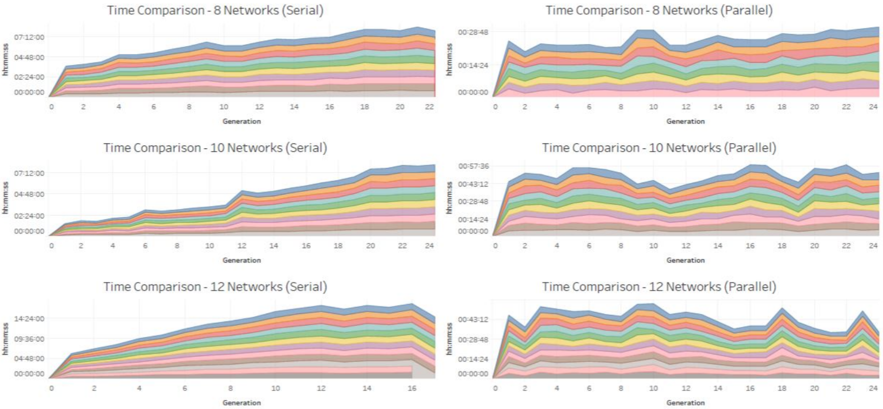


# Time Comparison

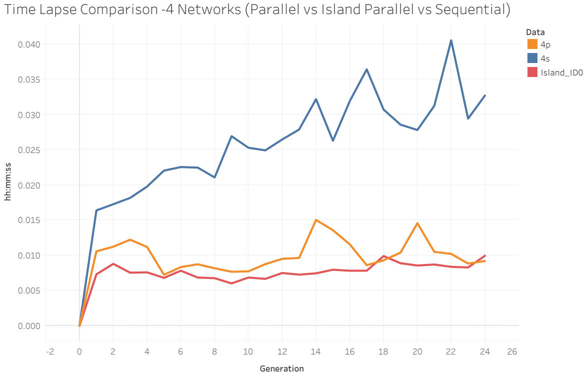



- The time taken by sequential code increases with generations.
- Time taken by Sequential code is in hours while the same can be done in minutes using MPI parallelism

## Sequential Algorithm vs Parallel Algorithm



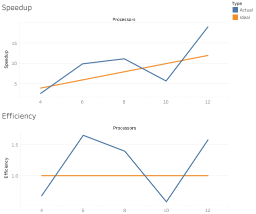
# Multiple-population Coarse Grained Algorithms

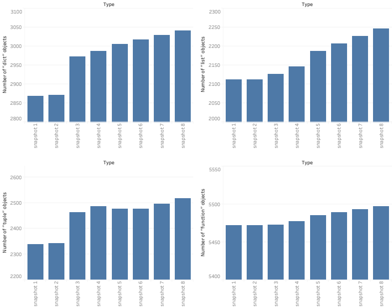



The Island model showed speed-up

# Super-linear behavior

# Super-linear behavior & Memory leak



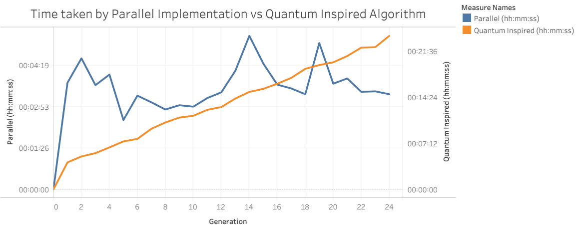

Super-linear speedup

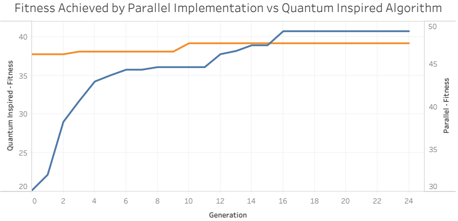
# Memory Leak

| types | Running for 1 generation | | Running for 5 generations | | Times increament in number of objects |
|---|---|---|---|---|---|
| | # objects | total size | # objects | total size | |
| <class 'numpy.ndarray | 3 | 117.23 MB | 3 | 117.23 MB | NA |
| <class 'tuple | 62126 | 5.69 MB | 482777 | 44.19 MB | 7.8 |
| <class 'list | 44141 | 4.30 MB | 250113 | 24.86 MB | 5.7 |
| <class 'dict | 10997 | 2.52 MB | 87719 | 20.02 MB | 8.0 |
| <class 'int | 83692 | 2.23 MB | 630694 | 16.84 MB | 7.5 |
| <class 'tensorflow.core.framework.node_def_pb2.NodeDef | | | 28057 | 2.35 MB | Internally saved from first generation in keras, and then incremented with generations |
| <class 'tensorflow.python.framework.tensor_shape.Dimension | | | 29688 | 1.59 MB | |
| <class 'tensorflow.python.framework.ops.Tensor | | | 29401 | 1.57 MB | |
| <class 'tensorflow.python.framework.tensor_shape.TensorShape | | | 29361 | 1.57 MB | |
| <class 'tensorflow.python.framework.ops.Operation | | | 28057 | 1.50 MB | |
| <class 'str' | 14656 | 1.05 MB | 17258 | 1.24 MB | 1.18 |

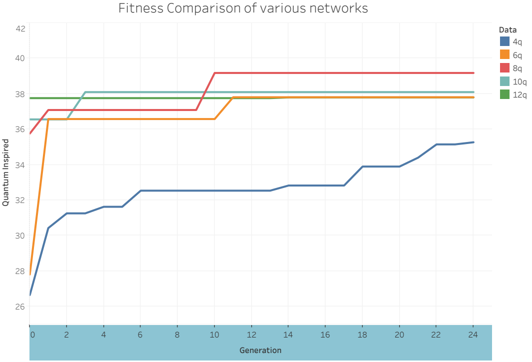
Approx 100MB memory leak in 5 generations

## Quantum Inspired Algorithm



## Quantum Inspired Algorithm




The trend of sum of Quantum Inspired for Generation. Color shows details about Data.

# References I

[1] Thomas N Theis and H-S Philip Wong. The end of moore's law: A new beginning for information technology. *Computing in Science & Engineering*, 19(2):41–50, 2017.

[2] John L Hennessy and David A Patterson. *Computer architecture: a quantitative approach*. Elsevier, 2011.

[3] Erick Cantú-Paz. A survey of parallel genetic algorithms. *Calculateurs paralleles, reseaux et systems repartis*, 10(2):141–171, 1998.

[4] John J Hopfield and David W Tank. "neural" computation of decisions in optimization problems. *Biological cybernetics*, 52(3): 141–152, 1985.

# References II

[5] Ling Wang, Fang Tang, and Hao Wu. Hybrid genetic algorithm based on quantum computing for numerical optimization and parameter estimation. *Applied Mathematics and Computation*, 171(2): 1141–1156, 2005.

[6] Jian-Ming Li, Xiao-Jing Wang, Rong-Sheng He, and Zhong-Xian Chi. An efficient fine-grained parallel genetic algorithm based on gpu-accelerated. In *Network and parallel computing workshops, 2007. NPC workshops. IFIP international conference on*, pages 855–862. IEEE, 2007.

[7] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.