Using MPI to Break the Data Encryption Standard

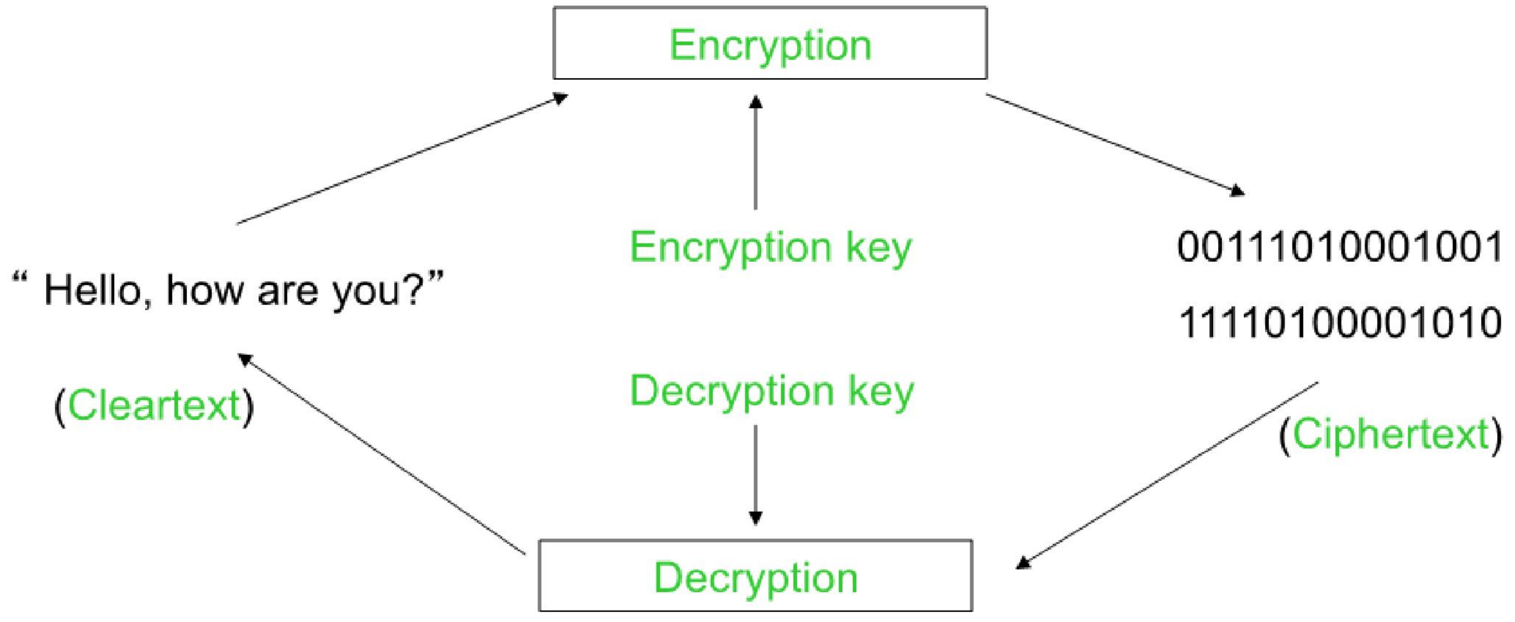
CSE 633: Parallel Algorithms

Group #24 Members:
Shane Anderson
Sergey Cherny
Pratik Deshpande
Aniket Chauhan

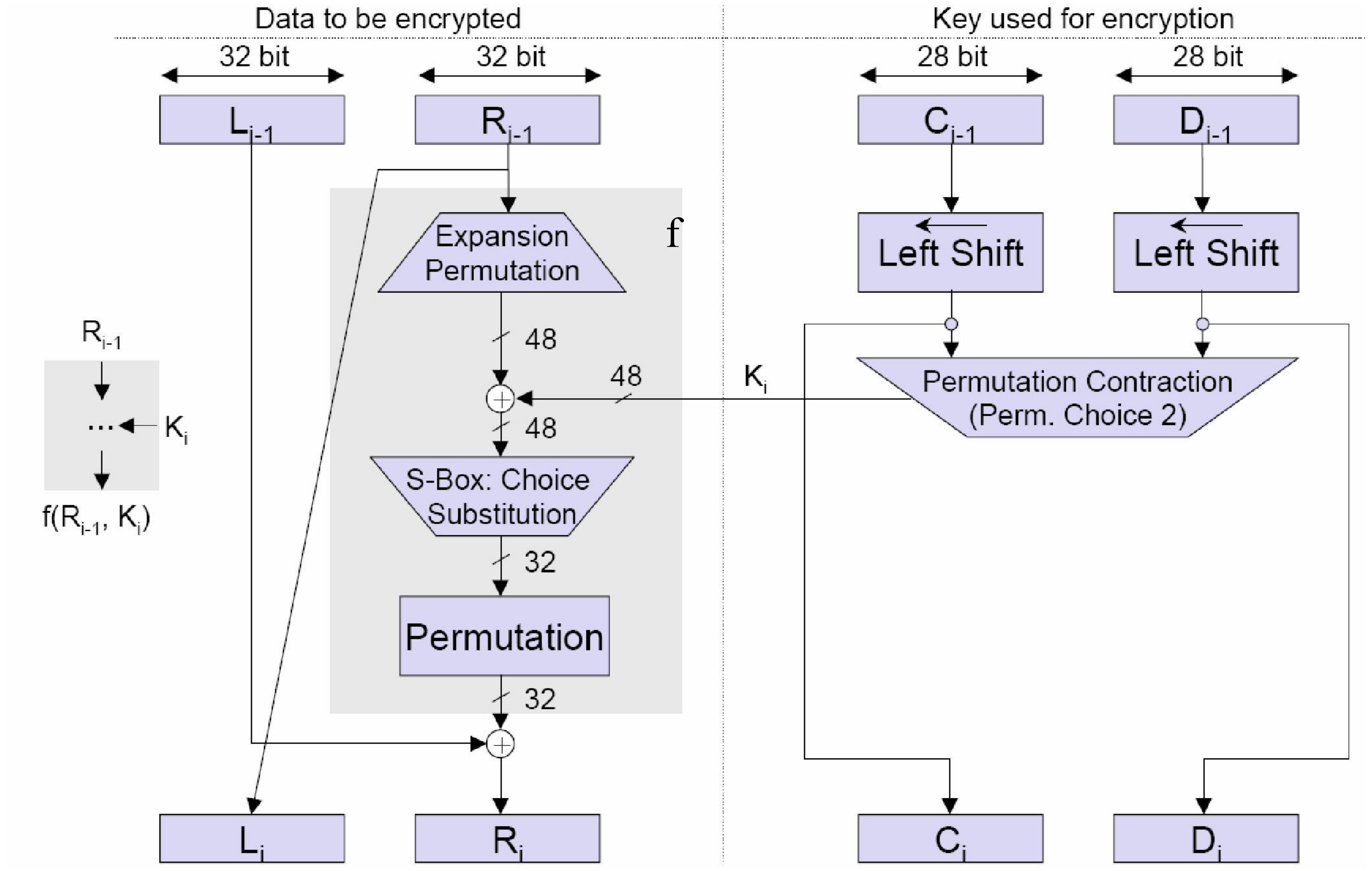
What is DES?

- DES (or Data Encryption Standard) is an algorithm used to encrypt electronic data.
- Encryption takes place as bitwise data transformation over several stages.
- Developed in the late 1970s and implemented by the early 1980s.
- Still used today, many variants are out there such as AES (Advanced Encryption Standard).

DES Overview



Structure of DES: Single iteration



Short-Key Weakness

- Cryptographers built a machine to break DES in 1998.
  - Cost: About $250,000
  - Took about 56 hours to find a key

- DES uses a 64 bit key but uses only 56 of those bits for encryption.
  - When it was first implemented, this was good enough
  - Nowadays, this is too short. More computing power means keys can be obtained by an exhaustive search, “brute force”

Known-Plaintext Attack

- We know a corresponding plaintext with ciphertext
- We run DES encryption on each 56-bit key in the key space and check if it matches the ciphertext.
- You can perform both ways because it is symmetric.
- Our goal is to find the key.
- Each processor will receive a subset of the key space to search.
- This is different from chosen-plaintext attack.

Total KeySpace to Search

- The total keyspace is 2^56
- 2^56 = 7.2057594037927936 x 10^16
- 72 quadrillion... = 72 million billion
- This is reduced by one half since taking the complement of the plaintext and complement of key will result in the same ciphertext. Therefore we can just disregard the complement of a key we already attempted.
- (1/2) x 2^55 = 3.6028797018963968 x 10^16
- 36 quadrillion... = 36 million billion

Breaking DES (Logic)

- Distribute the key space among workers.
- Master is responsible for synchronization of the program.
- Each worker is given the cipher text and corresponding plaintext.
- The worker tries to find the key by encrypting the plain text and comparing it with cipher text using its own key space.
- As soon as it breaks the key it notifies the master and the master notifies all other workers to stop processing.

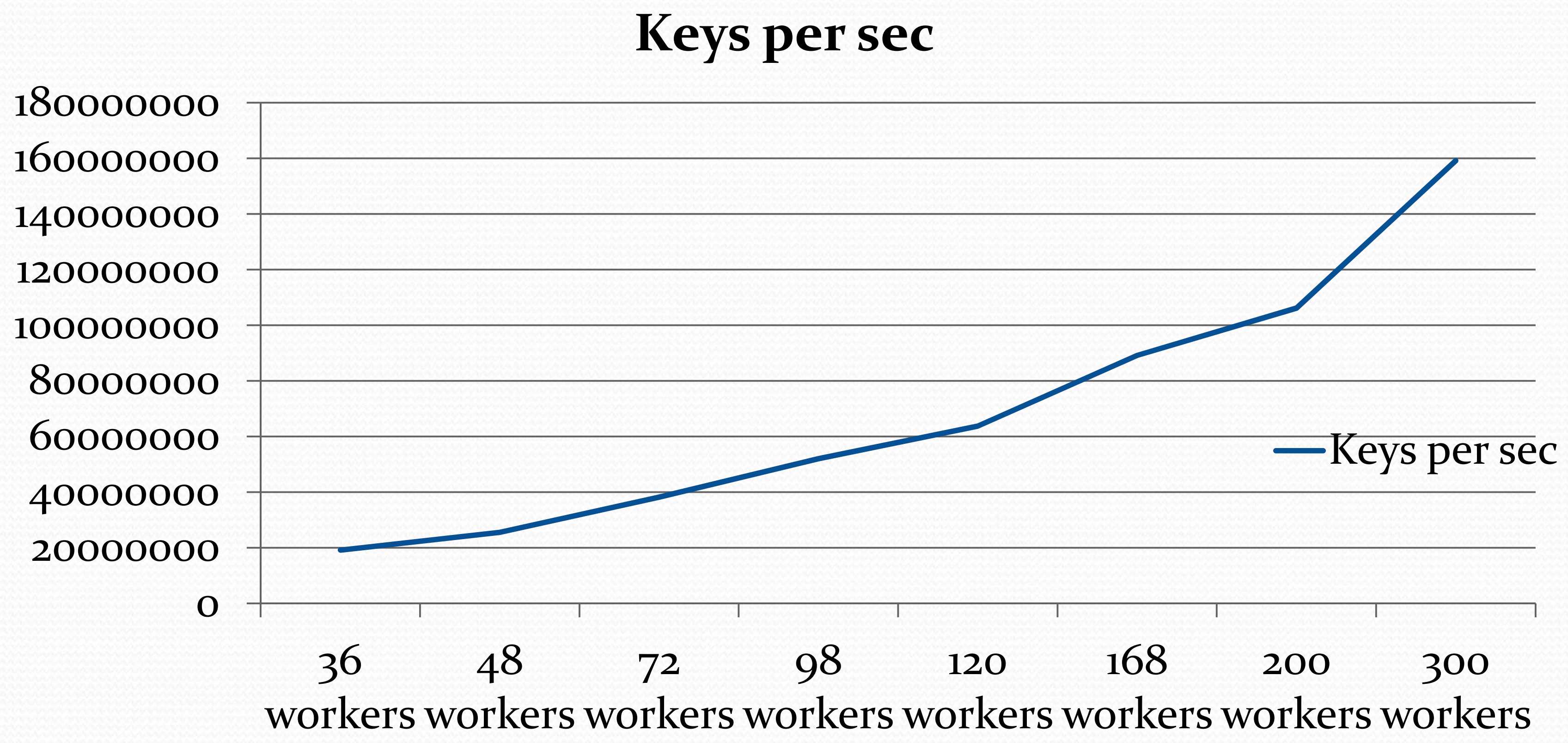
MPI Implementation Results

OBSERVATION TABLE FOR

Keys Searched Per Sec(approximately)

NUMBER OF WORKERS	Keys Checked per sec
36 workers	19097562
48 workers	25461504
72 workers	38192256
98 workers	51983904
120 workers	63653760
168 workers	89115264

Keys Searched Per Sec(approximately)



Importance of Key position

- In this program the running time is extremely affected by the position of the key.
- A particular key will be located at the different positions according to the total number of workers chosen.
- Due to this it is very important to calculate the results based on a specific position of the key for a particular number of workers.

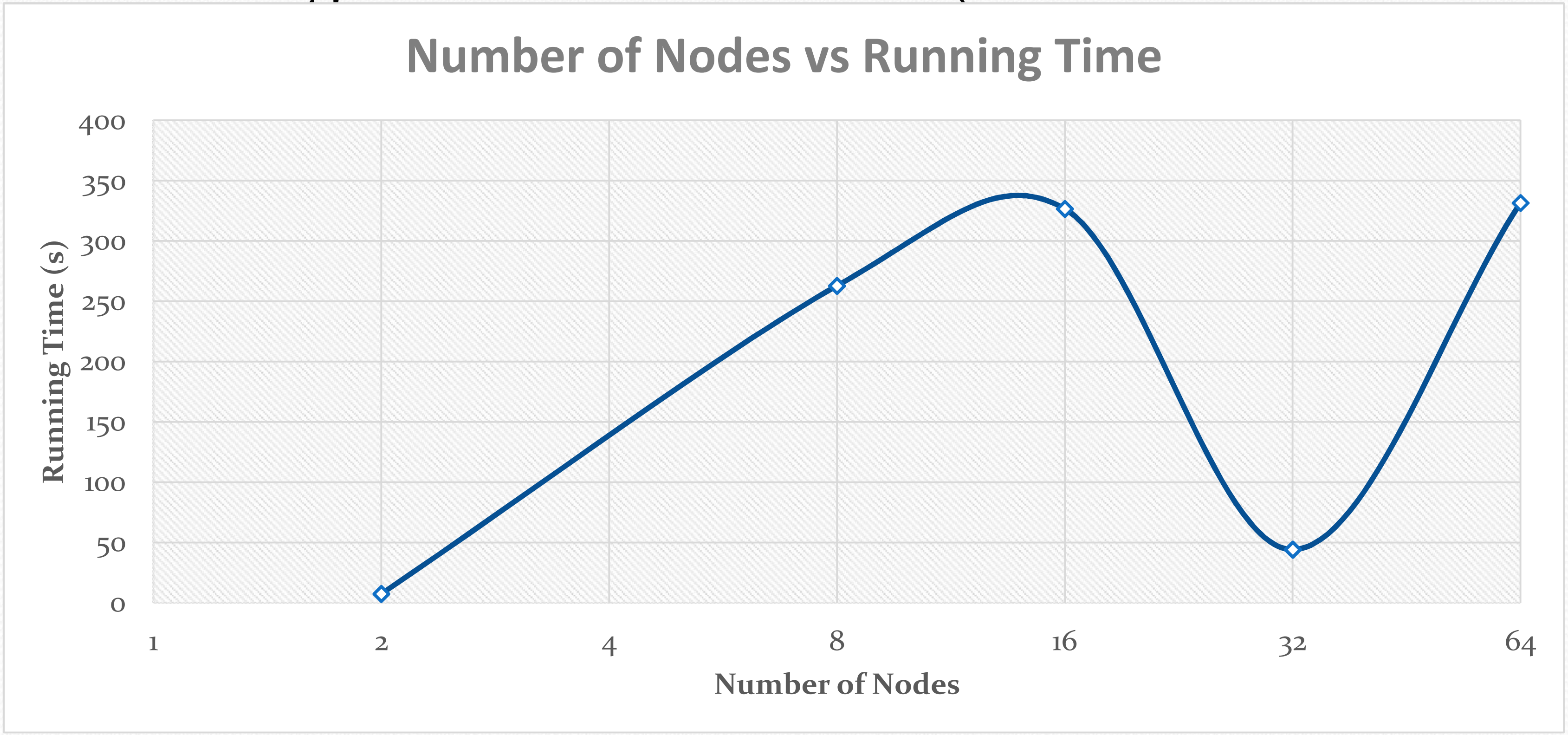
Graphs without taking KEY POSITION into consideration

Implementation

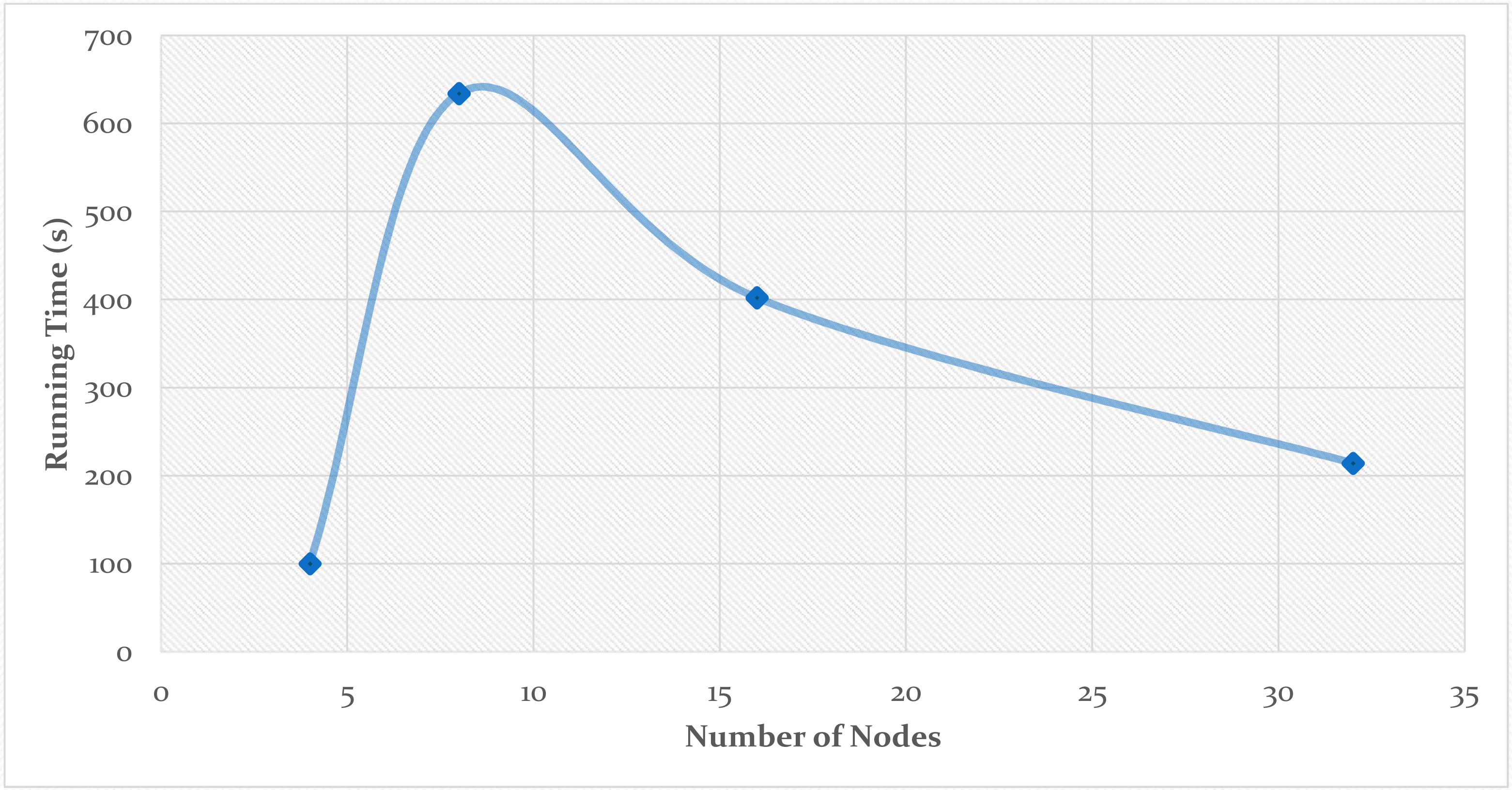
- These graphs are calculated using random keys.
- The running time is in SECONDS.
- As we will see further that we do not get any kind of consistent results due to it.
- These graphs are included just to highlight the importance of key position.

Nodes vs Running Time

- We observed the number of nodes vs running time to be as follows: (1



N Cores/Node

- 

Running Time (s)
700
600
500
400
300
200
100
0
0
5
10
15
20
25
30
35
Number of Nodes

Graphs obtained taking KEY POSITION into consideration

Implementation

- We used 0.001 percent location of key consistently for variable number of Workers.
- This is done due to the long running time constraint.
- To select the key we wrote a small function.
- The function takes input as total key space and total number of workers.
- It also takes a particular location of key as input in percentage form. E.g. Find the key located at 10 percent of each worker when key space is 2^56 and total number of workers are 32.

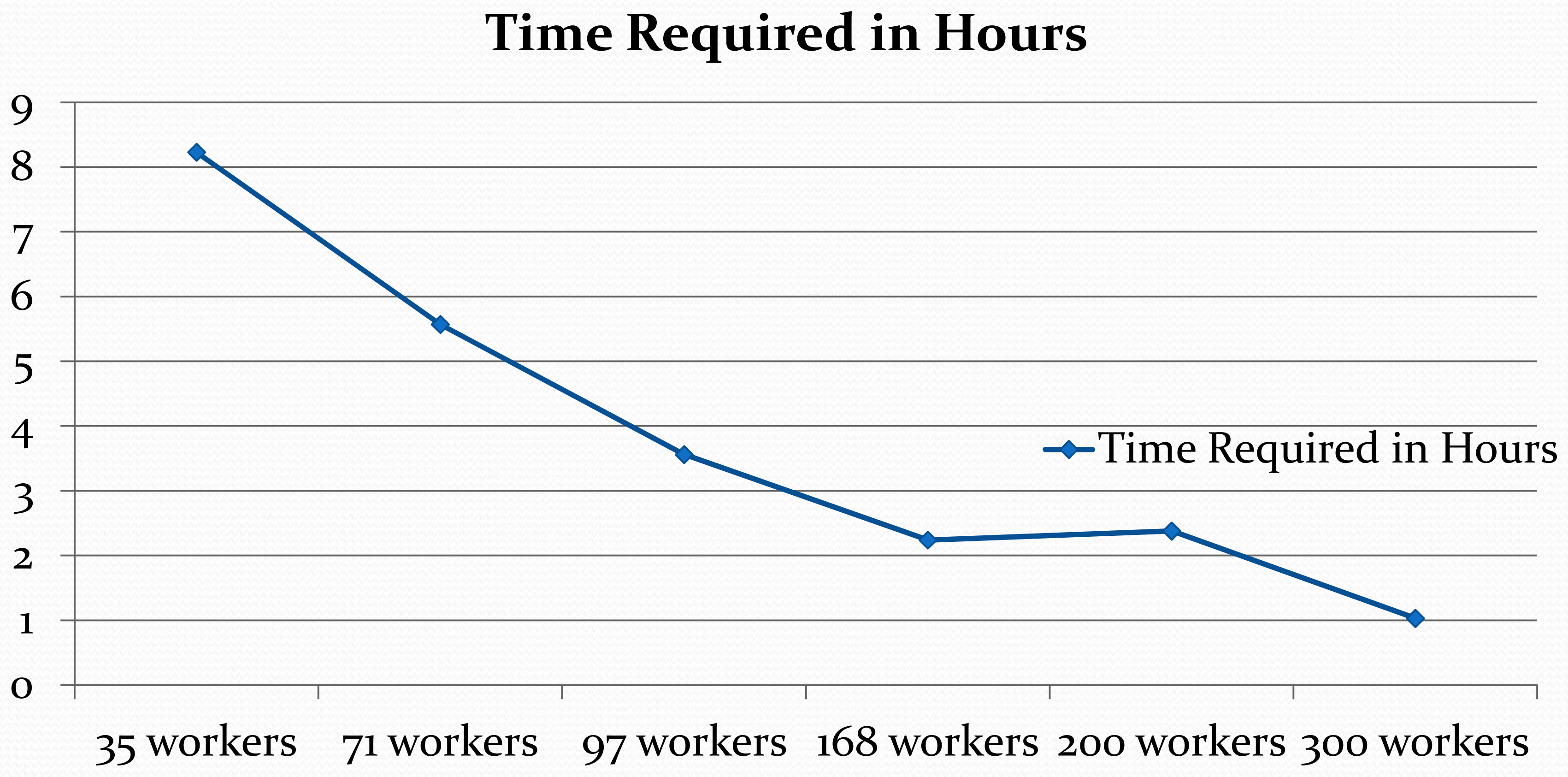
Implementation

- The function provides output as the keys in each worker at that particular user defined location as well as approximate time required to break the key.
- The graphs are calculated using one of the random keys provided by the function.

OBSERVATION TABLE FOR 1 Core Per Node

NUMBER OF WORKERS	TIME REQUIRED IN HOURS TO BREAK THE CIPHER TEXT
36	8.23
72	5.57
98	3.56
168	2.24
200	2.38
300	1.03

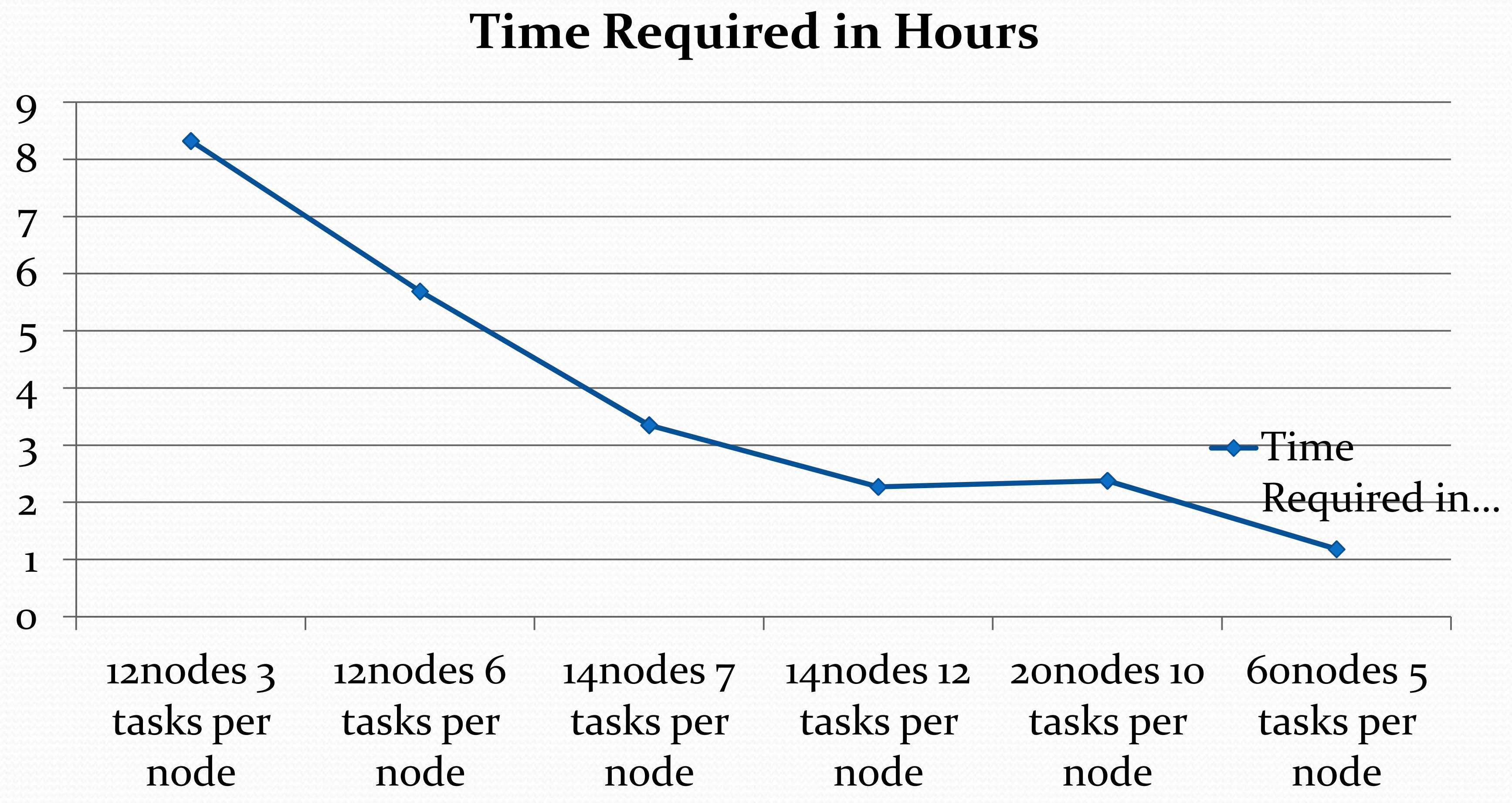
1 Core Per Node



OBSERVATION TABLE FOR N Tasks Per Node

NUMBER OF WORKERS AND THEIR CONFIGURATION	TIME REQUIRED IN HOURS TO BREAK THE CIPHER TEXT
12nodes 3 tasks per node	8.32
12nodes 6 tasks per node	5.69
14nodes 7 tasks per node	3.35
14nodes 12 tasks per node	2.27
20nodes 10 tasks per node	2.38
60nodes 5 tasks per node	1.18

N Tasks Per Node



Things Learned.....

- The most important part of this project was the enormous data size.
- So we learned how to handle as well as compute such a huge data size and make efficient use of computing power provided to us.
- It also painfully taught us to debug long running parallel programs eg. Getting errors after the program runs for a long time.
- One of the important features particular to this project was the impact of key position and the calculation required to find ideal keys prior to computation.
- So in short it did teach some basic techniques of Parallel Programming even though sometimes the lessons were hard.

References & Resources

- Coppersmith, Don. (1994). The data encryption standard (DES) and its strength against attacks at the Wayback Machine (archived June 15, 2007). *IBM Journal of Research and Development*, **38**(3), 243–250.

- Center for Concurrent Research (CCR) Systems

- DES Cracker Machine Project, EFF's "Deep Crack" (1998):
  - http://w2.eff.org/Privacy/Crypto/Crypto_misc/DESCracker/HTML/19980716_eff_des_faq.html

- *The DES Algorithm Illustrated* by J. Orlin Grabbe

Thank you for your attention! ☺