JAIST
JAPAN
ADVANCED INSTITUTE OF
SCIENCE AND TECHNOLOGY
1990

I116 - Final Assignment

Combination Generation

Student name: **NGUYEN, Tien Minh**

Student ID: **s1810445**

Keywords: ***Combination, Python, Parallel computing***

Table of content

Background

Problems

Definition: A combination is a subset of size **k** from a set of integers in [1, n], where order doesn't matter.

Eg: *With n=3 and k=2, 3 combinations that could be generated: (1, 2), (1, 3), (2, 3)*.

-> When n, k are big, it needs some alternative techniques to make the processing time shorter.

n	**k**	**combinations**
36	8	30.260.340
45	9	886.163.135
55	10	29.248.649.430
66	11	1.074.082.795.968

Background

Parallel Computing

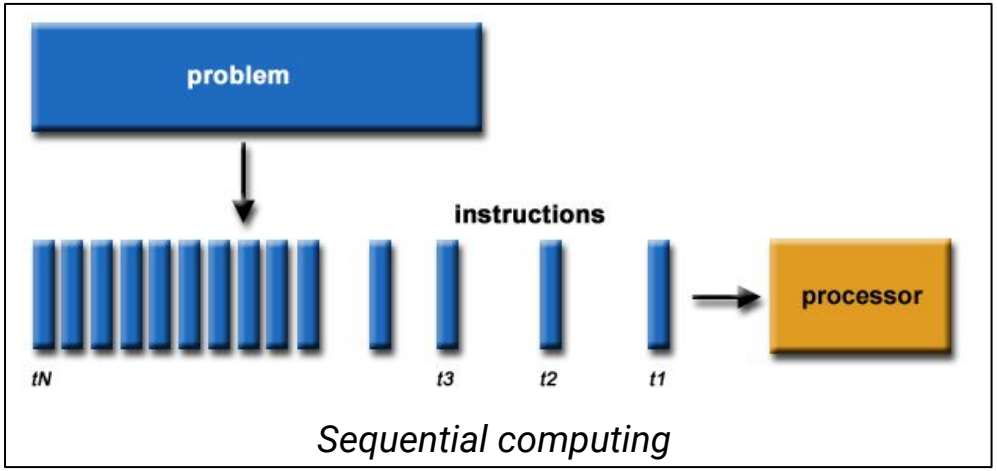
Definition:
Parallel Computing is the simultaneous processing of multiple compute resources to solve a computational problem.

Main type:

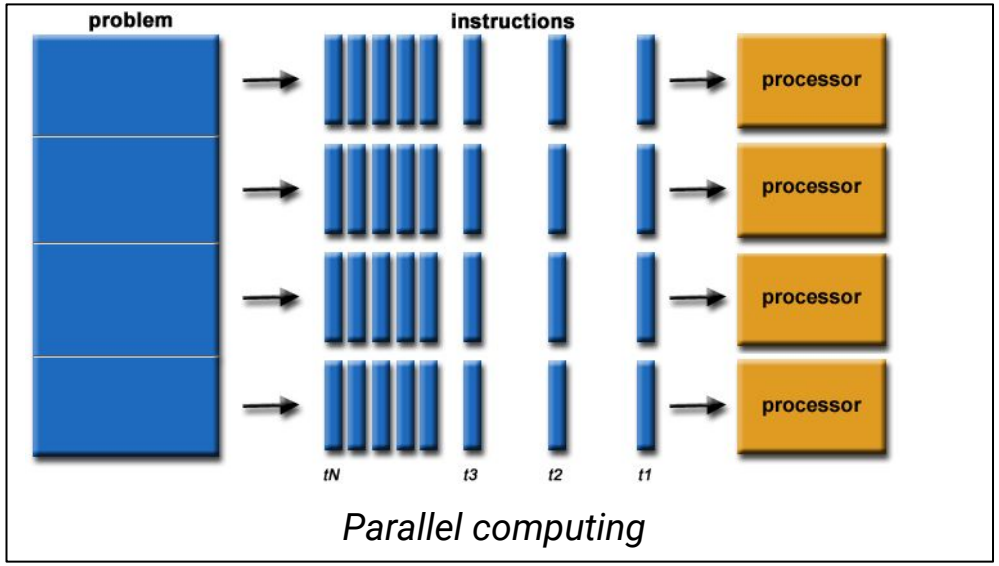
- Data parallelisms
- Task parallelisms

Supported devices:

- Multi-core CPU
- GPU
- FPGA



Sequential computing



Parallel computing

Proposed Algorithms

Sequential Generation (1st method)

Main idea:

- Use an ordered array to generate combinations
- Shift the right-most element that doesn't have maximum value of its position step by step

Advantages:

- Simple
- Fast

Disadvantages:

- Hard to divide into sub-problems, so cannot be parallelized
- Not suitable for searching

Eg: *With n=3 and k=2*

- The initial ordered array is: [1, 2, 3]
- 1st shift: [1, 2]
- 2nd shift: [1, 3]
- 3rd shift: [2, 3]

Proposed Algorithms

By-index Generation (2nd method)

Combinadic of an Integer:

A representation of the number based on combinations that maps to an unique combination.

$$C = C_{k-1}^{n1} + C_{k-2}^{n2} + \cdots + C_{1}^{n(k-1)}$$

Eg: suppose we have to generate combinations of an array [1, 2, 3], so all 2- elements combinations can be generated:

- 0 = 1C2 + 0C1 -> (1, 0) -> (2, 1)
- 1 = 2C2 + 0C1 -> (2, 0) -> (3, 1)
- 2 = 2C2 + 1C1 -> (2, 1) -> (3, 2)

Advantages

- Independent, then could be parallelized

Disadvantages:

- Complex, then requires more computing power

Tools & Experiments

Tools:

- OS: **CentOS 7**
- CPU: **Intel(R) Xeon(R) Gold 6130 CPU @ 2.10GHz (32 cores), Intel(R) Xeon(R) CPU E5-2687W 0 @ 3.10GHz (16 cores)**
- Language: **Python 3.5, Python3.6**
- Libraries: **Multiprocessing**
- Algorithms: **Sequential Generation, By-index Generation**

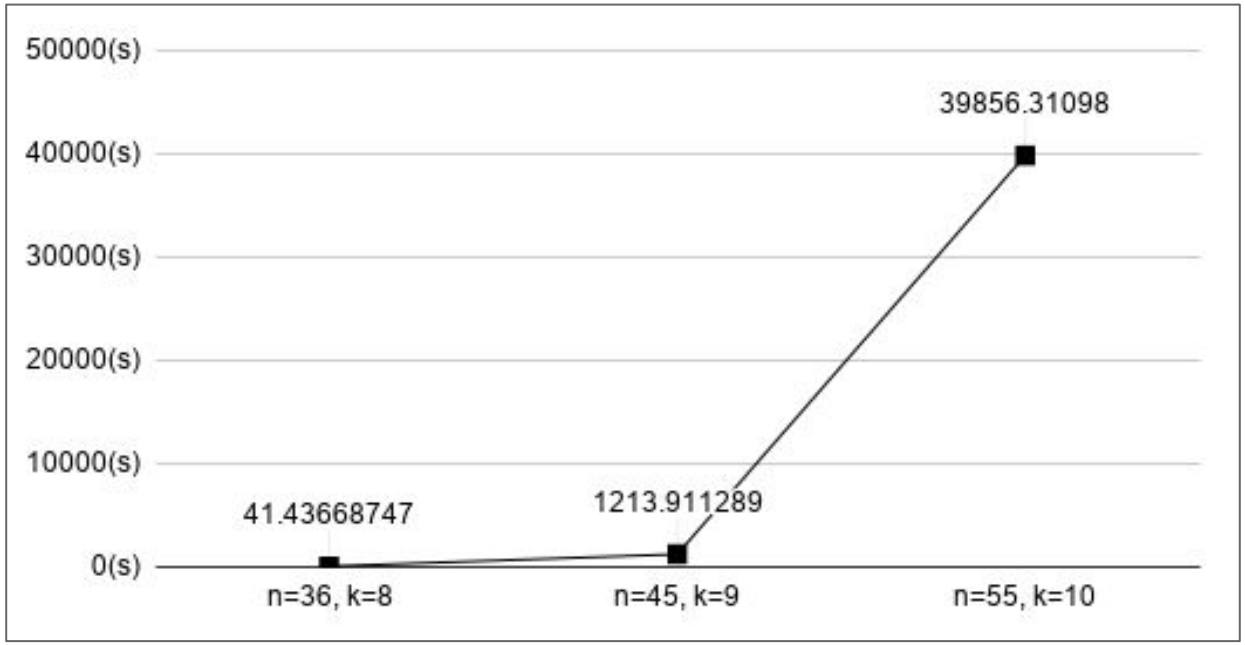
PC Cluster in JAIST

Evaluation:

- To measure the usage, execution time per CPU cores
- To measure the execution time when running an algorithm with 8 cores, 16 cores, 30 cores

Experiments 1

Sequential Generation (1st method)



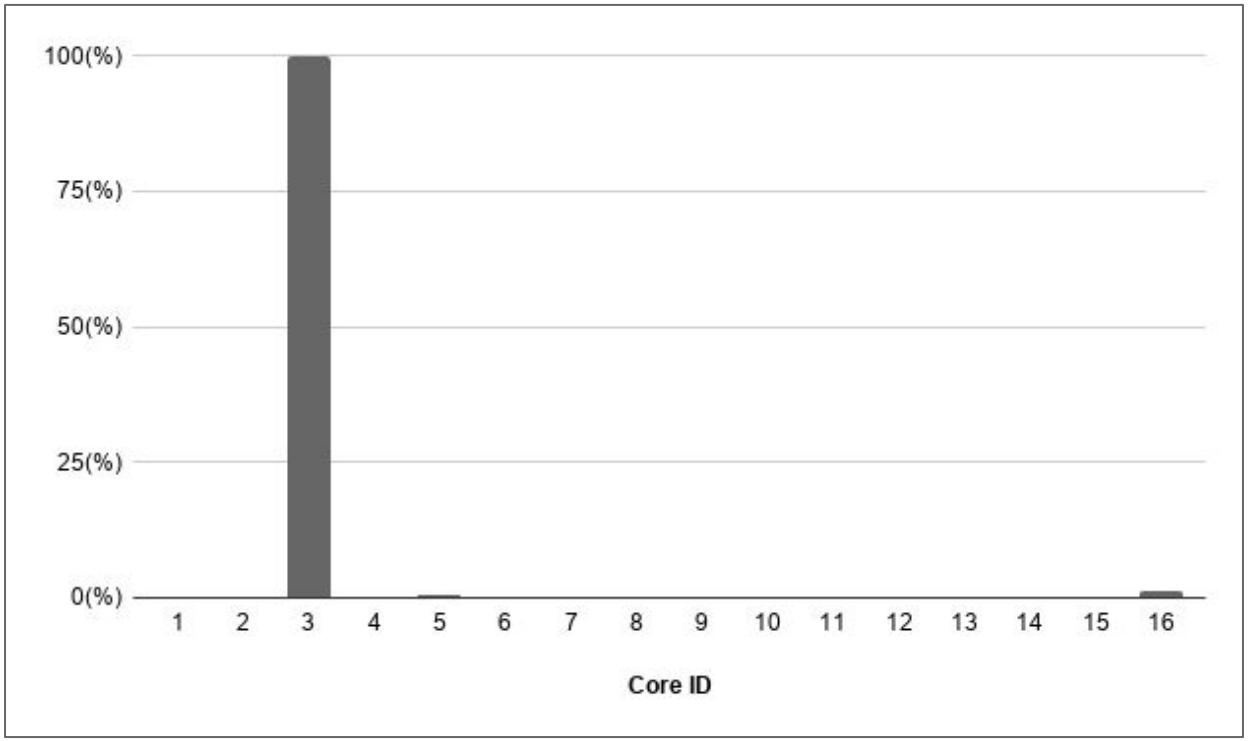
Execution time of running 1st method

Source code can be seen here:
https://github.com/m-inh/i116-final-assignment

Data can be seen here:
https://docs.google.com/spreadsheets/d/1W-LX1mG01MFvtW4Mb-yMZy3pKsY-V6X5fCwKUospUPw/edit#gid=0

Experiments 2

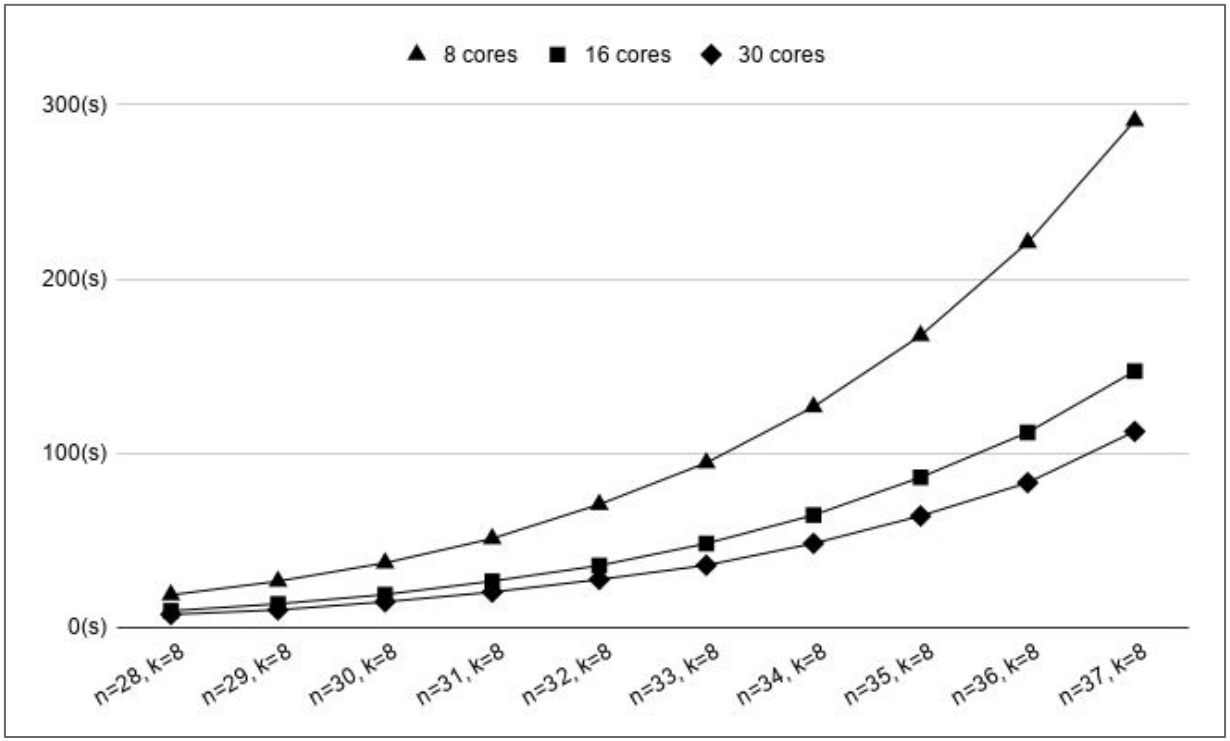
Sequential Generation (1st method)



CPU-cores utilization of running 1st method

Experiments 3

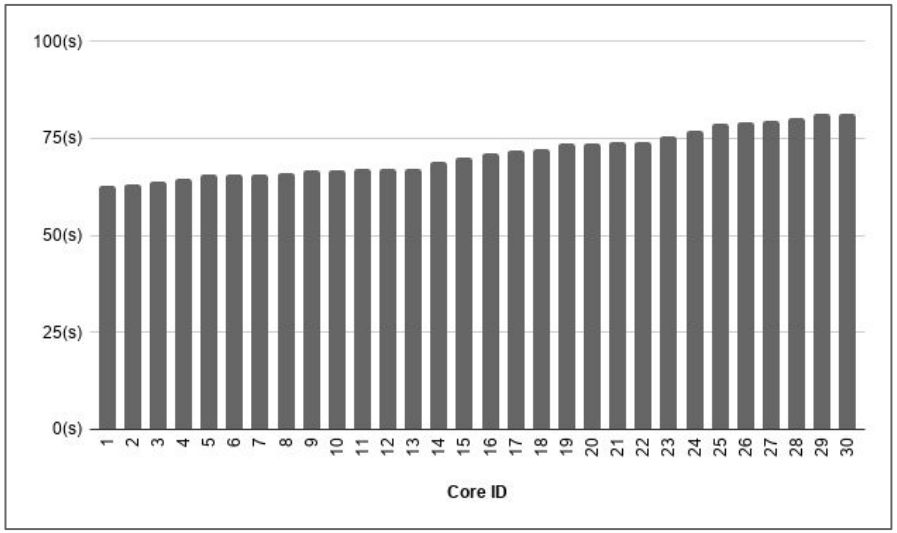
By-index Generation (2nd method)

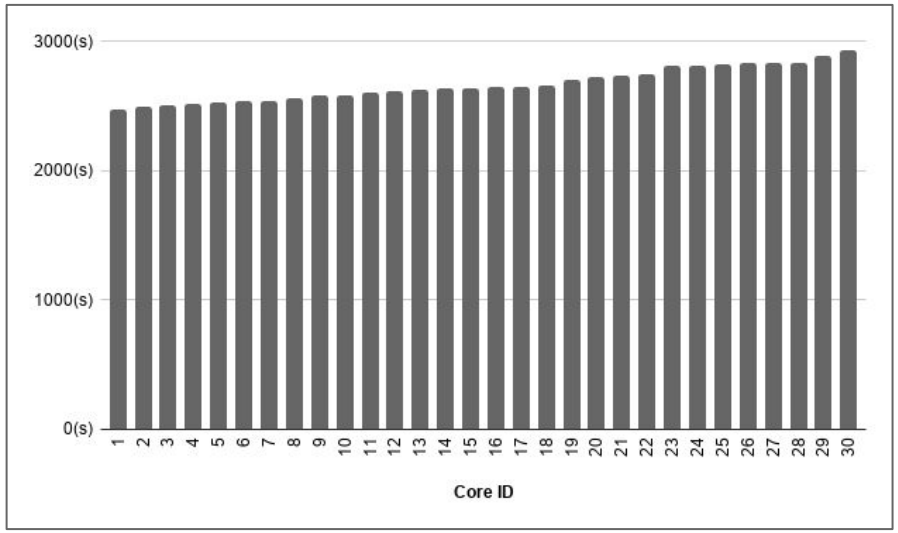


Execution time of running 2nd method

Experiments 4

By-index Generation (2nd method)





Execution time per CPU-core of running 2nd method
Left: n=36, k=8, Right: n=45, k=9

Comments

The reasonable strategy :

- Use 1st method for generating all combinations
- Use 2nd method for searching, or generating a small number of combinations.

Some works can be done to achieve a better performance:

- Port Python code to lower language like C/C++/Rust.
- Find a way to divide a job in 1st method into smaller jobs, so this method can be executed in parallel.
- Distribute jobs of 2nd method to many computers. (use MPI)

Thanks for your kind attention!

Please feel free to comment and ask me some questions.
It helps me improving my presentation in the next times.