Performance Analysis for Parallel Programs
How to estimate the benefit of parallelization?

• What is the ideal speedup on N machines/cores?

• Why cannot we achieve the ideal speedup?
How to estimate the benefit of parallelization?

• What is the ideal speedup on N machines/cores?
  • Nx

• Why cannot we achieve the ideal speedup?
  • Practical reason
    • Thread-related overhead
    • Non-CPU resource limitation
  • Algorithm reason
    • non-parallelizable code component
What affect real parallelization efficiency?

- Amdahl's law
  - https://www.techopedia.com/definition/17035/amdahls-law
- Critical path
  - You can represent a parallel program in a DAG, with an edge representing a task cannot start until another one finish
  - The longest path in your DAG is called critical path
  - The length of the critical path determines the execution time of your program with unlimited number of processors
How to estimate the benefit of parallelization?

• Amdahl's law
• Critical path

• Load imbalance
• Resource competition
• Data sharing cost (false sharing leads to huge performance lost)
• Synchronization overhead (lock, etc.)
• Other parallelization overhead (i.e., data duplication, work duplication, and aggregation)
Software bugs

Memory bugs
Concurrency bugs
Memory bug

Memory Layout
Buffer overflow
Uninitialized read
Memory leak
Memory layout

Text segment: store instruction (code)
Data: global/static initialized variable
BSS: global/static uninitialized variable
Heap: malloc
Stack: local variable
Memory layout (con’t)

```c
int x = 100;       //data segment
int main() {
    int a = 2;       //stack
    static int y;    //BSS segment
    int *ptr = (int *)malloc(2*sizeof(int));
    ptr[0] = 5;      //heap
    ptr[1] = 6;      //heap
    free(ptr);
    return 0;
}
```
Stack buffer overflow

1. Code example:

```c
#include "stdio.h"
#include "string.h"

int main(int argc, void ** argv)
{
    if(argc<2) return 0;
    printf("%s is the argument\n",(char*) argv[1]);

    char p[10];
    strcpy(p, argv[1]);

    printf("%s is the string p\n", p);
    return 0;
}
```
Stack buffer overflow

2. What is a stack buffer overflow bug:
   read/write a buffer in stack beyond the buffer range.

3. Stack buffer overflow:
   - Return address: next instruction after the function return.
   - In the example code, strcpy over writes the return address
   - This overwrite beyonding the array size is a buffer overflow.
   - Consequence:
     - Invalid instruction: program crash
     - Jump to malicious program: hacker attack.
Heap buffer overflow

1. Code example:

```c
#include "stdio.h"
#include "string.h"
#include “stdlib.h”

int main(int argc, void ** argv) {

    if(argc<3) return 0;
    printf("%s and %s are the arguments\n",(char*) argv[1], (char*) argv[2]);

    char* p1 = malloc(10);
    char* p2 = malloc(10);
    strcpy(p1, argv[1]);
    strcpy(p2, argv[2]);

    printf("%s is the string 1\n", p1);
    printf("%s is the string 2\n", p2);
    return 0;
}
```
Heap buffer overflow

2. What is a heap buffer overflow bug:
   - read/write a buffer in heap beyond the buffer range.

3. Impact of heap buffer overflow:
   - Corrupt nearby data
   - Crash the program if overflow into invalid program regions
Concurrency bugs (data races)
public class test2{
    static int x = 0;
    public static void main(String argv[]){
        testThread t1 = new testThread();
        testThread t2 = new testThread();
        t1.start();
        t2.start();
        try{
            t1.join();
            t2.join();
        }catch(Exception e){}
        System.out.print(x+ " ");
    }
}

class testThread extends Thread{
    public void run(){
        if (test2.x == 0)
            test2.x ++;
    }
}
Test 1: exact buy milk

```
X=0

X==0?
Y
X++

Print(X)
2

X==0?
Y
X++

X==0?
N
X=0

X=0
Print(X)
1
```
Test 1: exact buy milk

Data race:
- two operations accessing same memory
- relative order is undetermined
- at least one is write
Test 1: Fixing

Two regions protected by the same lock cannot interleaving with each other.