DATA STRUCTURES AND ALGORITHMS 2 PRESENTATION

BY GREG BALBIRNIE

DATABASE SEARCHING AND SORTING

- The Program randomly generates a list of one thousand names and birthdays.
- The program searches for all birthdays in a give month and sorts the names.

PARALLELISATION

- Parallel name generation
- Parallel list search
- Parallel sorting

```
void InitPeople(vector<person> &people, string forenames[], string surnames[], mutex &people_mutex)
{
    thread init_threads[10];

    //create 100 threads
    for (int i = 0; i < 10; i++)
    {
        init_threads[i] = thread(ThreadFunction, ref(people), forenames, surnames, ref(people_mutex), i);
    }
    for (int i = 0; i < 10; i++)
    {
        init_threads[i].join();
    }
}

void ThreadFunction(vector<person> &people, string forenames[], string surnames[], mutex &people_mutex, int rando_seed)
{
    //create 100 people
    for (int k = 0; k < 100; k++)
    {
        //generate a random seed for the randomiser
        int seed = rando_seed + ((rando_seed + 1) * 10) + (k * 100);
        people_mutex.lock();
        people.push_back(person(forenames, surnames, seed));
        people_mutex.unlock();
    }
}
```

```
//define threads
for (int i = 0; i < people.size() / 100; i++)
{
    //search for everyone with specific birth month
    searchThreads.push_back(thread(SearchPeople, people, ref(selected), (i * 1000), ((i * 1000) + 999), ref(no_of_people), ref(count_mutex), chosen_month));
}

//rejoin threads
for (int i = 0; i < searchThreads.size(); i++)
{
    searchThreads[i].join();
}
```

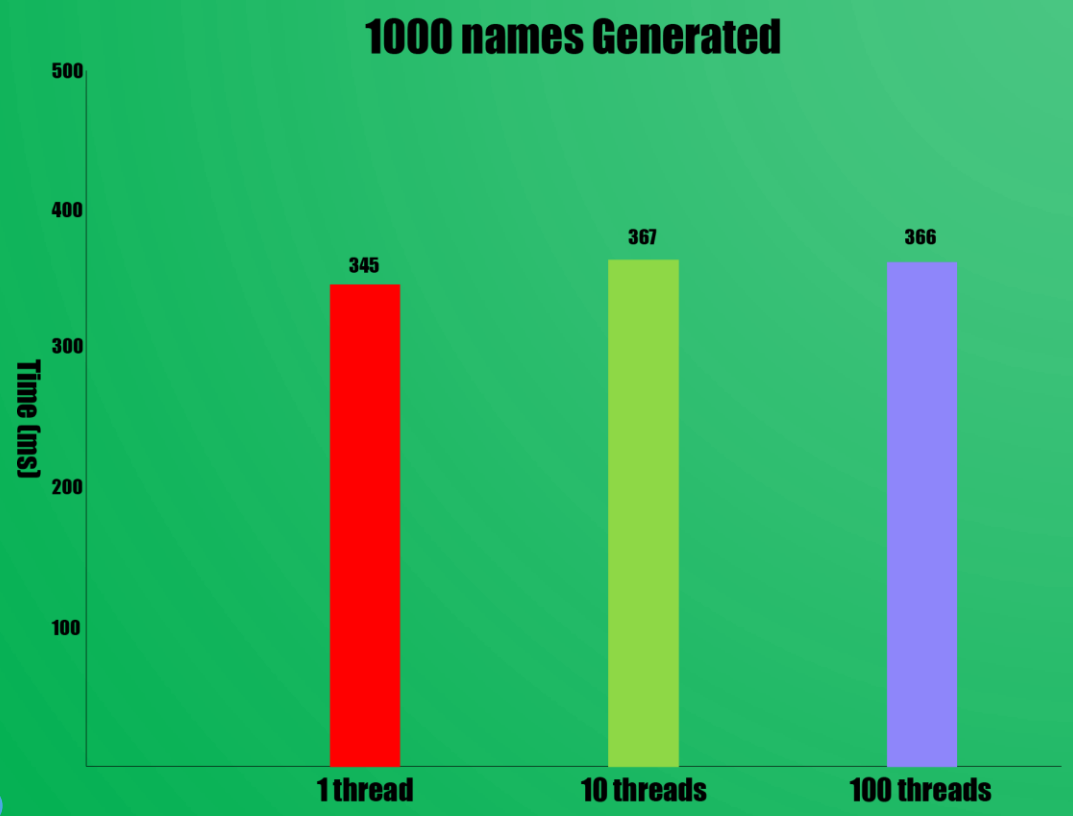
```
void SearchPeople(vector<person> people, vector<person> &selected, int min, int max, int &no_of_people, mutex &count_mutex, int chosen_month)
{
    //search for specific days
    //maybe use condition variables

    for (int i = min; i < max; i++)
    {
        //if (people[i].getDOBMonth() == chosen_month)
        if (people[i].getDOBMonth() == 5)
        {
            //add people born in september to new vector
            AddtoSelected(selected, people[i], no_of_people, count_mutex);
        }
    }
}

void AddtoSelected(vector<person> &selected, person newMemeber, int &no_of_people, mutex &count_mutex)
{
    selected.push_back(newMemeber);
    count_mutex.lock();
    no_of_people += 1;
    count_mutex.unlock();
}
```

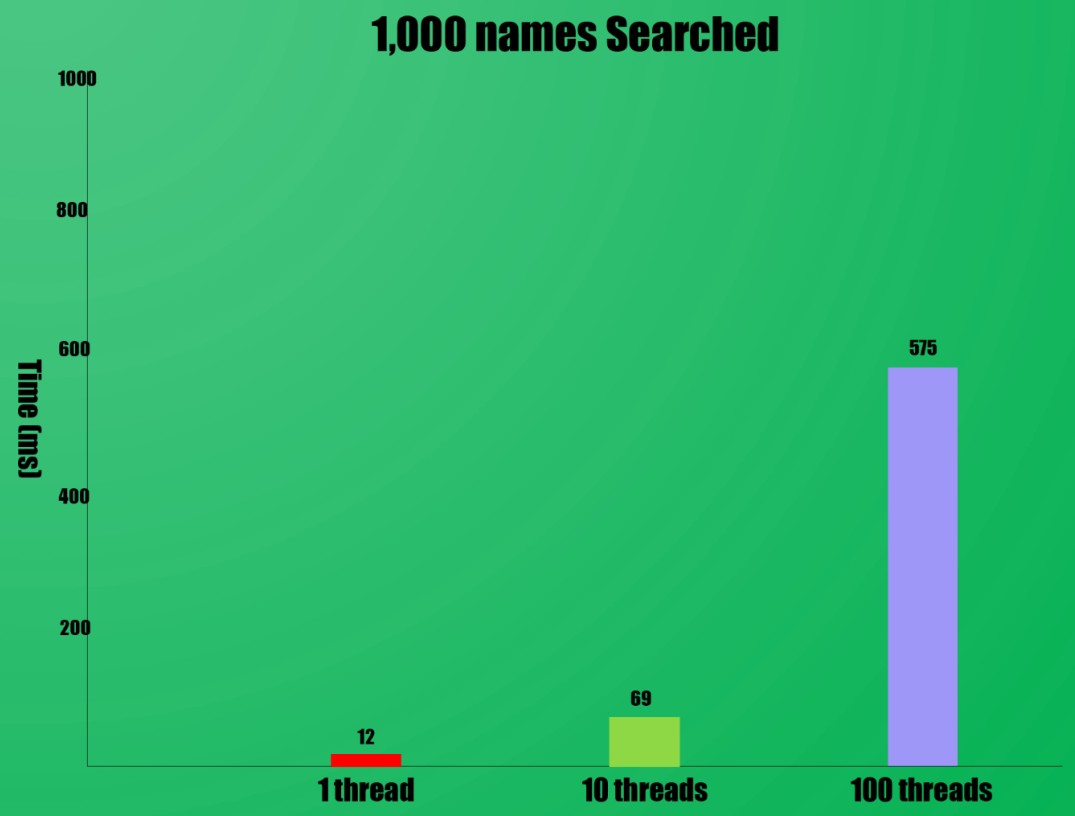
```
//vector of threads
vector<thread> sortThreads;
bool swapped;
int startPoint = 0;

//do until sorted
do
{
    swapped = false;
    for (int i = startPoint; i < selected.size(); i += 2)
    {
        //sort the two that are assigned
        if ((i + 1) <= (selected.size() - 1))
        {
            sortThreads.push_back(thread(SortPeople, ref(selected), forenames, surnames, i, i + 1, ref(swapped)));
        }
    }

    for (int i = 0; i < sortThreads.size(); i++)
    {
        sortThreads[i].join();
    }

    sortThreads.clear();

    if (startPoint == 0)
    {
        startPoint = 1;
    }
    else if (startPoint == 1)
    {
        startPoint = 0;
    }

} while (swapped == true);
```

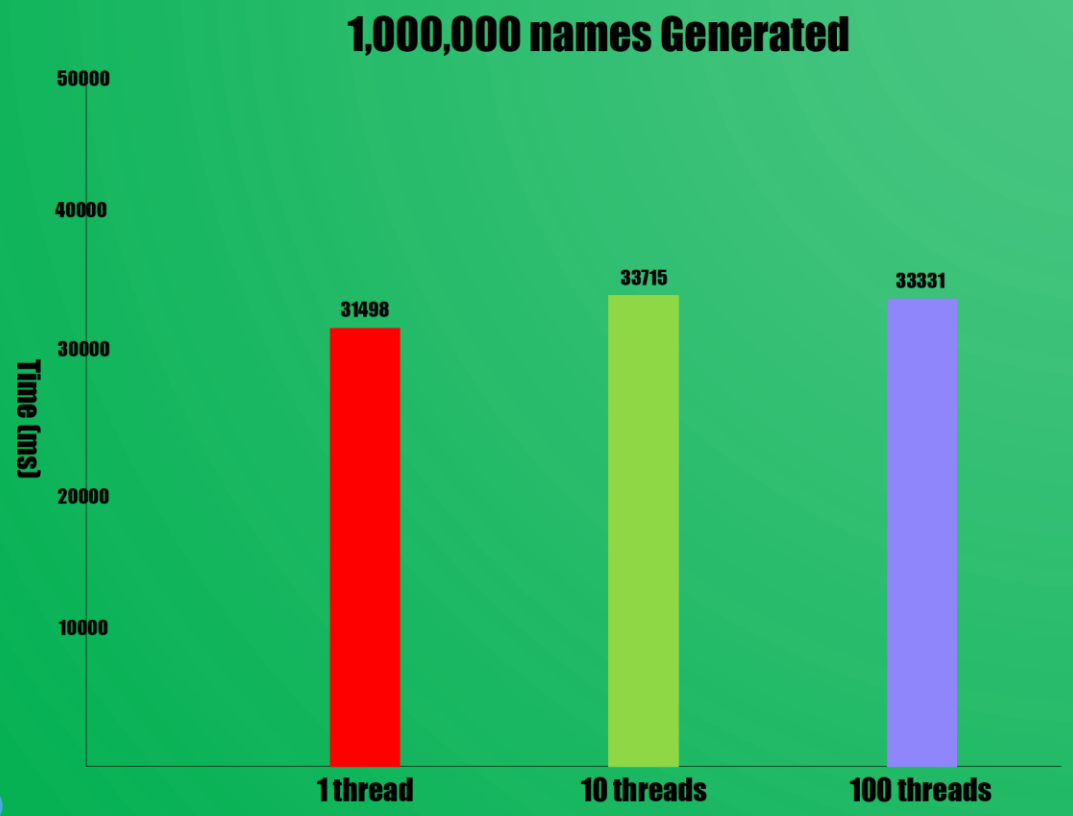
```
void SortPeople(vector<person> &selected, string forenames[], string surnames[], int idx1, int idx2, bool &swapped)
{
    person temp(forenames, surnames, 0);    //blank person

    //check and swap
    if (selected[idx1].getSurname() > selected[idx2].getSurname())
    {
        temp = selected[idx1];
        selected[idx1] = selected[idx2];
        selected[idx2] = temp;

        swapped = true;
    }
}
```

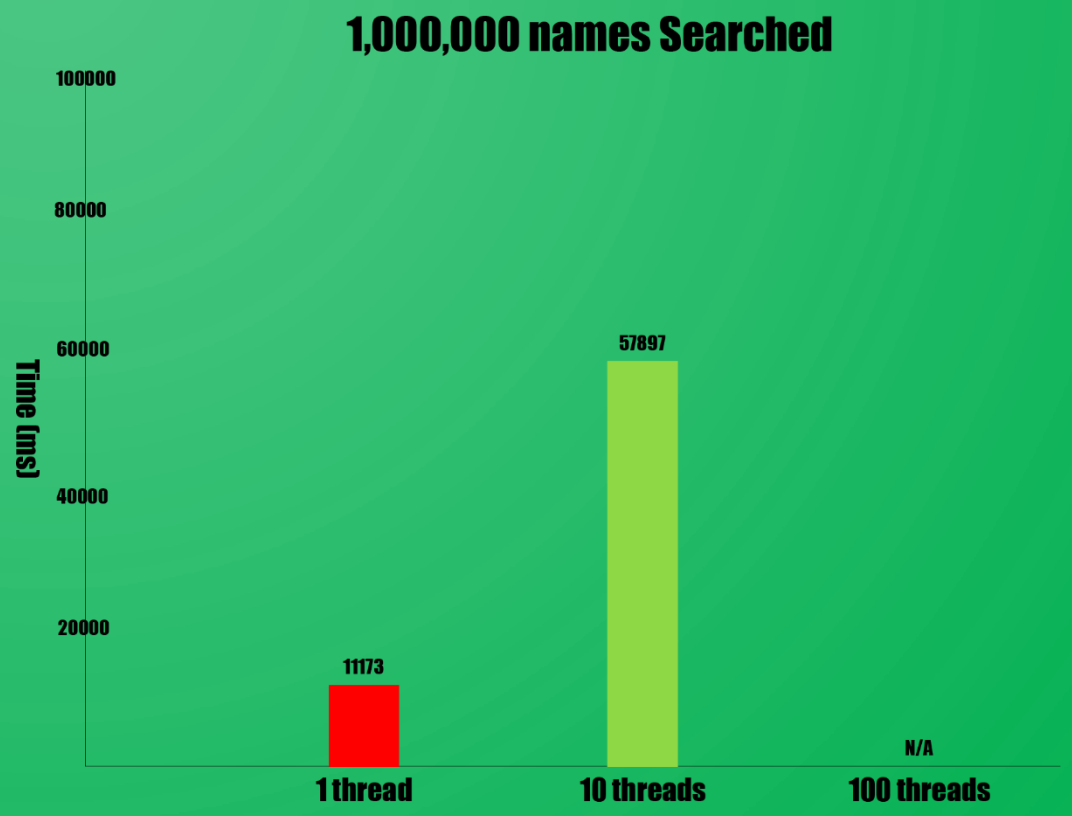
REJECTED IDEAS

- Planned to use a barrier
- Planned to use a channel to pass into vector


1000 names Generated
Time (ms)
500
400
300
200
100
345
367
366
1 thread
10 threads
100 threads


1,000 names Searched
Time (ms)
1000
800
600
400
200
12
69
575
1 thread
10 threads
100 threads


1,000,000 names Generated
Time (ms)
50000
40000
30000
20000
10000
31498
33715
33331
1 thread
10 threads
100 threads


1,000,000 names Searched
Time (ms)
100000
80000
60000
40000
20000
11173
57897
N/A
1 thread
10 threads
100 threads

EVALUATION

- Faster without threading
- Creating threads is slower than the code itself