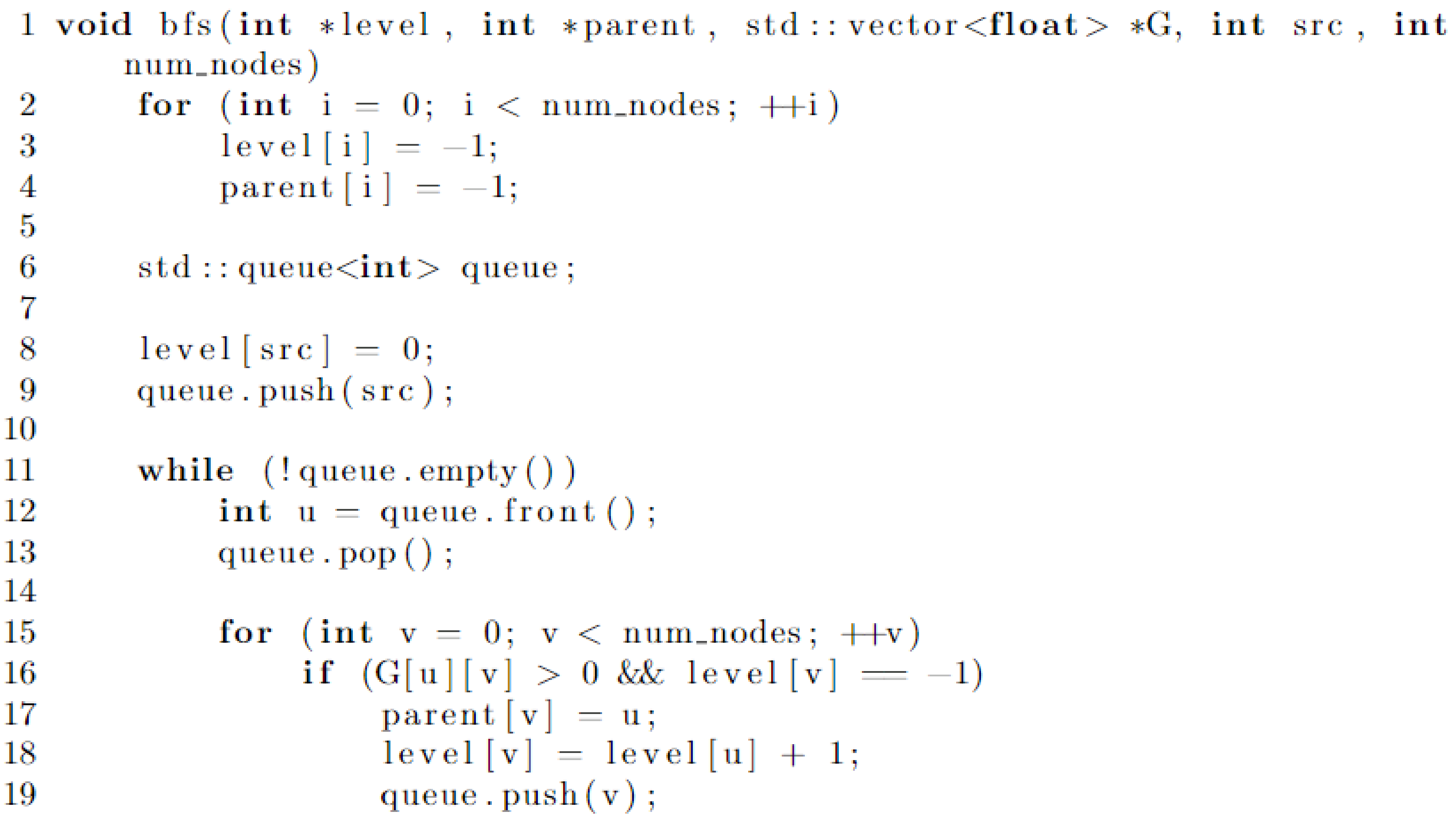
GRAPH MASSIVIZER

WP4 update - Analytical model demonstration

22nd March 2024

This project has received funding from the European Union’s Horizon Research and Innovation Actions under Grant Agreement № 101093202.

Naïve sequential BFS algorithm



```
void bfs(int *level, int *parent, std::vector<float> *G, int src, int num_nodes)
    for (int i = 0; i < num_nodes; ++i)
        level[i] = -1;
        parent[i] = -1;

    std::queue<int> queue;

    level[src] = 0;
    queue.push(src);

    while (!queue.empty())
        int u = queue.front();
        queue.pop();

        for (int v = 0; v < num_nodes; ++v)
            if (G[u][v] > 0 && level[v] == -1)
                parent[v] = u;
                level[v] = level[u] + 1;
                queue.push(v);
```

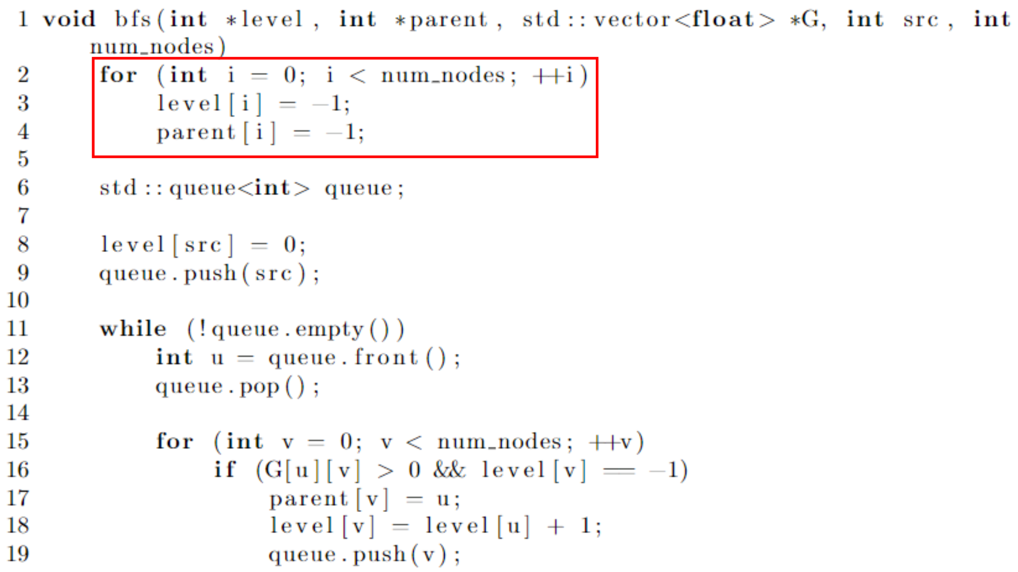
Assumptions

- Graph only has a single connected component
- Write back cache
  - Memory write takes the same time as memory read

Constructing symbolical model

$$T_{BFS} = nT_{init}$$

$$T_{init} = 2T_{mem\_write}$$

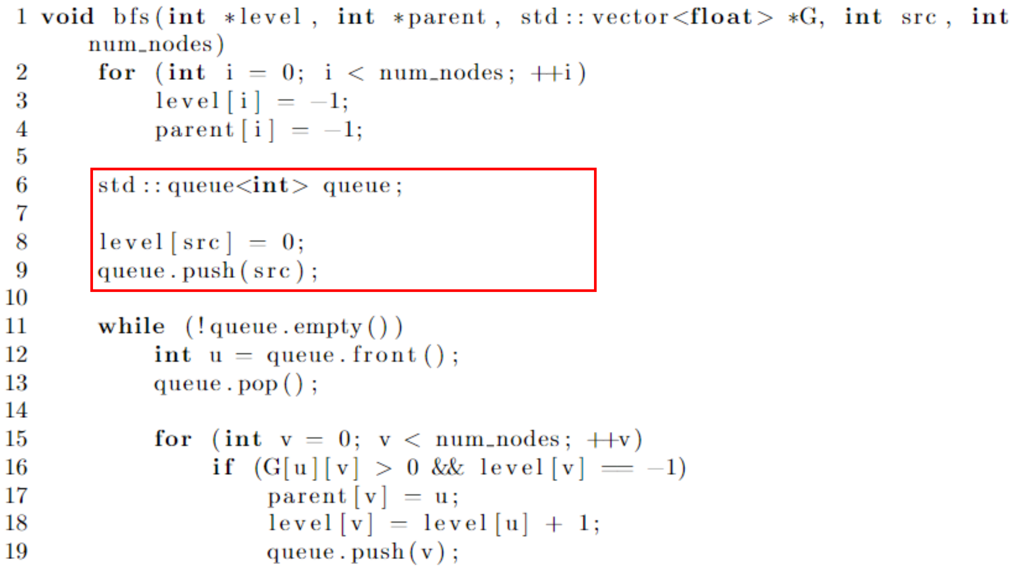


```
void bfs(int *level, int *parent, std::vector<float> *G, int src, int
      num_nodes)
    for (int i = 0; i < num_nodes; ++i)
        level[i] = -1;
        parent[i] = -1;

    std::queue<int> queue;

    level[src] = 0;
    queue.push(src);

    while (!queue.empty())
        int u = queue.front();
        queue.pop();

        for (int v = 0; v < num_nodes; ++v)
            if (G[u][v] > 0 && level[v] == -1)
                parent[v] = u;
                level[v] = level[u] + 1;
                queue.push(v);
```

Constructing symbolical model

$$T_{BFS} = nT_{init}$$

$$T_{init} = 2T_{mem\_write}$$



```
void bfs(int *level, int *parent, std::vector<float> *G, int src, int
    num_nodes)
    for (int i = 0; i < num_nodes; ++i)
        level[i] = -1;
        parent[i] = -1;

    std::queue<int> queue;

    level[src] = 0;
    queue.push(src);

    while (!queue.empty())
        int u = queue.front();
        queue.pop();

        for (int v = 0; v < num_nodes; ++v)
            if (G[u][v] > 0 && level[v] == -1)
                parent[v] = u;
                level[v] = level[u] + 1;
                queue.push(v);
```

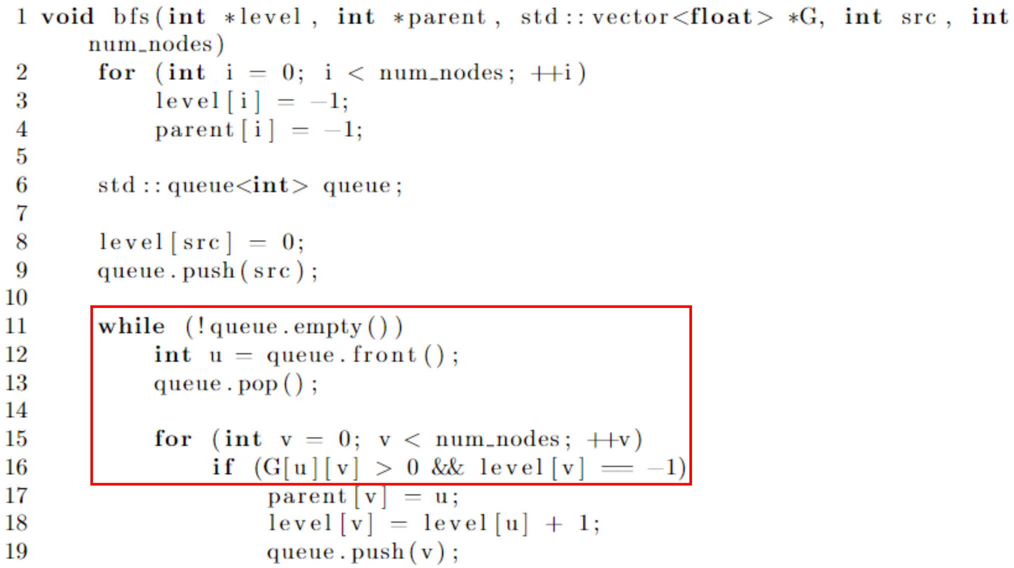
We skip this!

Constructing symbolical model

$$T_{BFS} = nT_{init} + nT_{while\_loop}$$

$$T_{init} = 2T_{mem\_write}$$

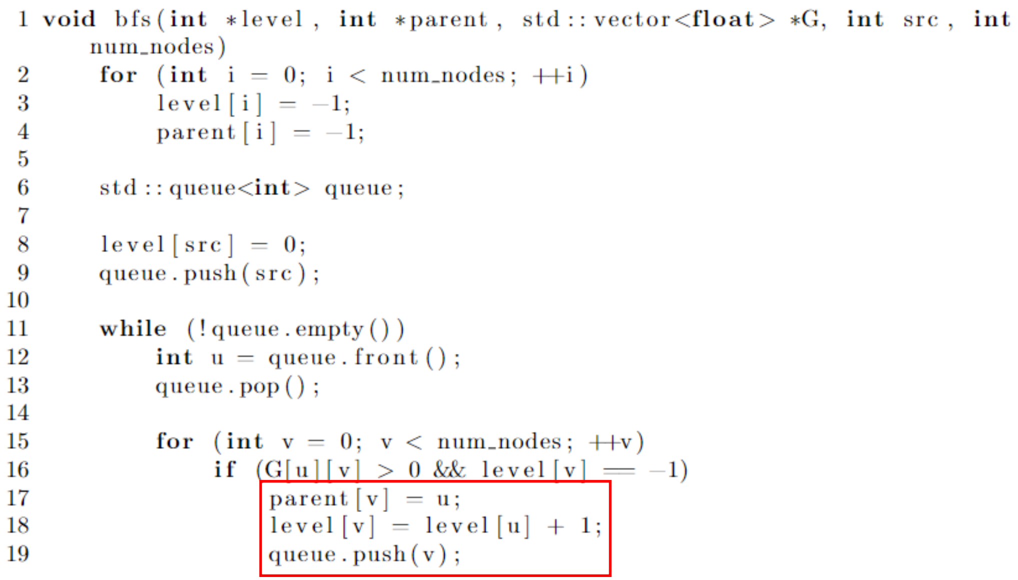
$$T_{while\_loop} = T_{q\_front} + T_{q\_pop} + n\left(T_{add} + T_{G\_mem\_read} + T_{mem\_read}\right)$$



```
void bfs(int *level, int *parent, std::vector<float> *G, int src, int
    num_nodes)
    for (int i = 0; i < num_nodes; ++i)
        level[i] = -1;
        parent[i] = -1;

    std::queue<int> queue;

    level[src] = 0;
    queue.push(src);

    while (!queue.empty())
        int u = queue.front();
        queue.pop();

        for (int v = 0; v < num_nodes; ++v)
            if (G[u][v] > 0 && level[v] == -1)
                parent[v] = u;
                level[v] = level[u] + 1;
                queue.push(v);
```

Constructing symbolical model

$$
\begin{aligned}
T_{BFS} &= nT_{init} + nT_{while\_loop} + nT_{visit\_node} \\
T_{init} &= 2T_{mem\_write} \\
T_{while\_loop} &= T_{q\_front} + T_{q\_pop} + n\left(T_{add} + T_{G\_mem\_read} + T_{mem\_read}\right) \\
T_{visit\_node} &= 2T_{DRAM} + T_{L1} + T_{add} + T_{q\_push}
\end{aligned}
$$



```
void bfs(int *level, int *parent, std::vector<float> *G, int src, int
    num_nodes)
  for (int i = 0; i < num_nodes; ++i)
    level[i] = -1;
    parent[i] = -1;

  std::queue<int> queue;

  level[src] = 0;
  queue.push(src);

  while (!queue.empty())
    int u = queue.front();
    queue.pop();

    for (int v = 0; v < num_nodes; ++v)
      if (G[u][v] > 0 && level[v] == -1)
        parent[v] = u;
        level[v] = level[u] + 1;
        queue.push(v);
```

Constructing symbolical model

$$T_{BFS} = nT_{init} + nT_{while\_loop} + nT_{visit\_node}$$

$$T_{init} = 2T_{mem\_write}$$

$$T_{while\_loop} = T_{q\_front} + T_{q\_pop} + n\left(T_{add} + T_{G\_mem\_read} + T_{mem\_read}\right)$$

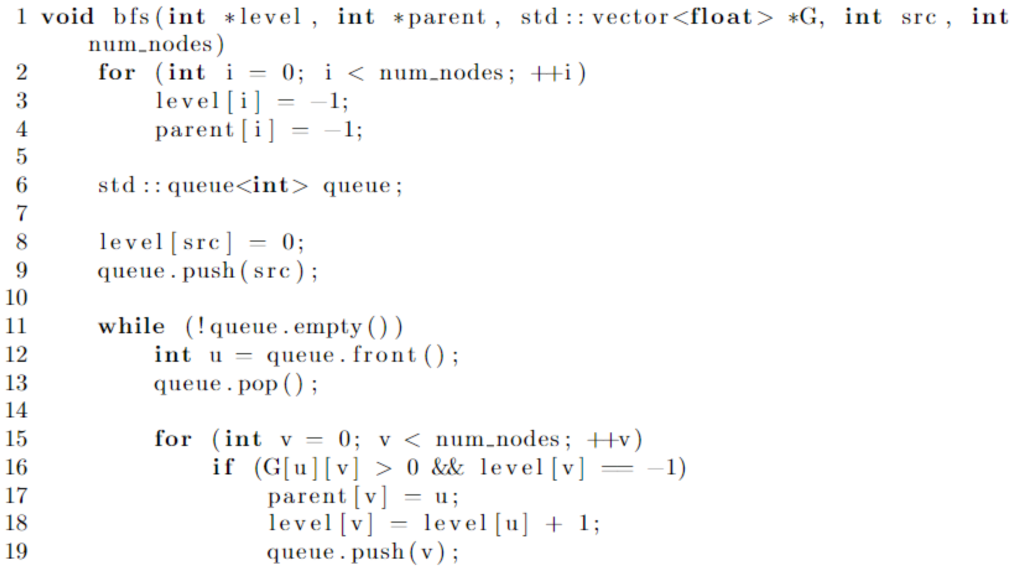
$$T_{visit\_node} = 2T_{DRAM} + T_{L1} + T_{add} + T_{q\_push}$$

$$T_{mem\_write} = T_{mem\_read} = (1 - MR_{L1})T_{L1} + MR_{L1}(1 - MR_{L2})T_{L2} + MR_{L1}MR_{L2}(1 - MR_{L3})T_{L3} + MR_{L1}MR_{L2}MR_{L3}T_{DRAM}$$

$$MR_{L\{1,2,3\}} = \frac{1}{cacheLineSize_{L\{x\}}/sizeof(\texttt{int})}$$

$$T_{G\_mem\_read} = (1 - G\_MR_{L1})T_{L1} + G\_MR_{L1}(1 - G\_MR_{L2})T_{L2} + G\_MR_{L1}G\_MR_{L2}(1 - G\_MR_{L3})T_{L3} + G\_MR_{L1}G\_MR_{L2}G\_MR_{L3}T_{DRAM}$$

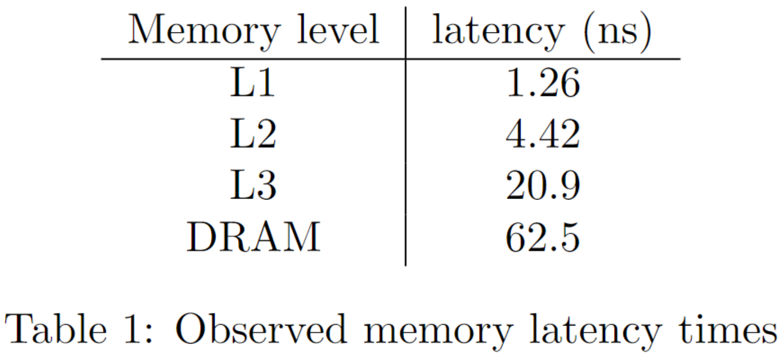
$$G\_MR_{L\{1,2,3\}} = \frac{1}{cacheLineSize_{L\{x\}}/sizeof(\texttt{float})}$$



```
void bfs(int *level, int *parent, std::vector<float> *G, int src, int
    num_nodes)
    for (int i = 0; i < num_nodes; ++i)
        level[i] = -1;
        parent[i] = -1;

    std::queue<int> queue;

    level[src] = 0;
    queue.push(src);

    while (!queue.empty())
        int u = queue.front();
        queue.pop();

        for (int v = 0; v < num_nodes; ++v)
            if (G[u][v] > 0 && level[v] == -1)
                parent[v] = u;
                level[v] = level[u] + 1;
                queue.push(v);
```

Calibrating the model

- Cache line sizes are 64 bytes for L1, L2, and L3
- Memory latency (measured by using LMBench `lat_mem_rd`):



Memory level	latency (ns)
L1	1.26
L2	4.42
L3	20.9
DRAM	62.5

Table 1: Observed memory latency times

Calibrating the model

- Cache line sizes are 64 bytes for L1, L2, and L3
- Memory latency (measured by using LMBench `lat_mem_rd`):

Memory level	latency (ns)
L1	1.26
L2	4.42
L3	20.9
DRAM	62.5

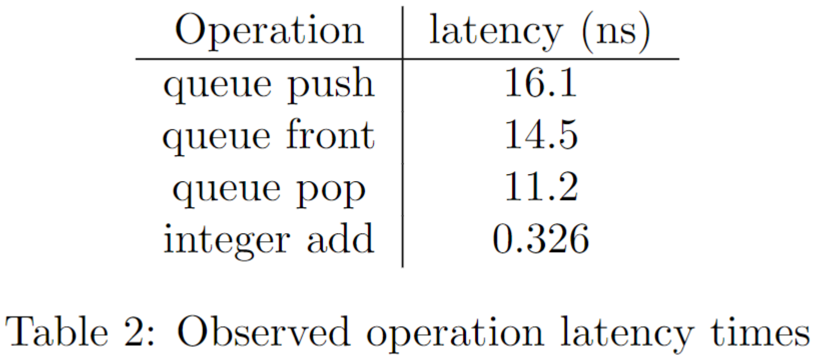
Table 1: Observed memory latency times

$$MR_{L\{1,2,3\}} = G\_MR_{L\{1,2,3\}} = \frac{1}{64/4} = \frac{1}{16}$$

$$\begin{aligned} T_{mem\_write} &= T_{mem\_read} = T_{G\_mem\_read} \\ &= \frac{15}{16}1.26 + \frac{1}{16}\frac{15}{16}4.24 + \frac{1}{16}\frac{1}{16}\frac{15}{16}20.9 + \frac{1}{16}\frac{1}{16}\frac{1}{16}62.5 \\ &= 1.521484375 \end{aligned}$$

Calibrating the model

- Operation latencies (measured by microbenchmarking):



Operation	latency (ns)
queue push	16.1
queue front	14.5
queue pop	11.2
integer add	0.326

Table 2: Observed operation latency times

Calibrating the model

- The final model:

$$
\begin{aligned}
T_{BFS} &= nT_{init} + nT_{while\_loop} + nT_{visit\_node} \\
T_{init} &= 2 \cdot 1.521484375 \\
T_{while\_loop} &= 14.5 + 11.2 + n\left(0.326 + 1.521484375 + 1.521484375\right) \\
T_{visit\_node} &= 2 \cdot 62.5 + 1.26 + 0.326 + 16.1
\end{aligned}
$$

$$
\begin{aligned}
T_{BFS} &= n(2 \cdot 1.521484375 + 14.5 + 11.2 + n\left(0.326 + 1.521484375 + 1.521484375\right) \\
&\quad + 2 \cdot 62.5 + 1.26 + 0.326 + 16.1) \\
&= 171.429n + 3.36897n^2
\end{aligned}
$$

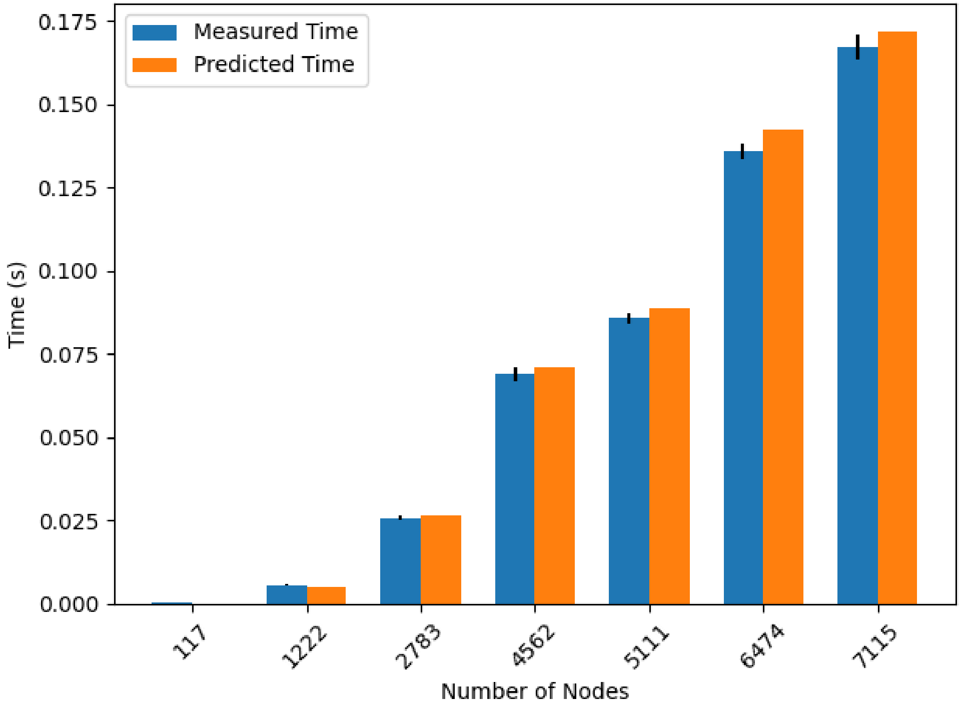
Calibrating the model

- The final model:

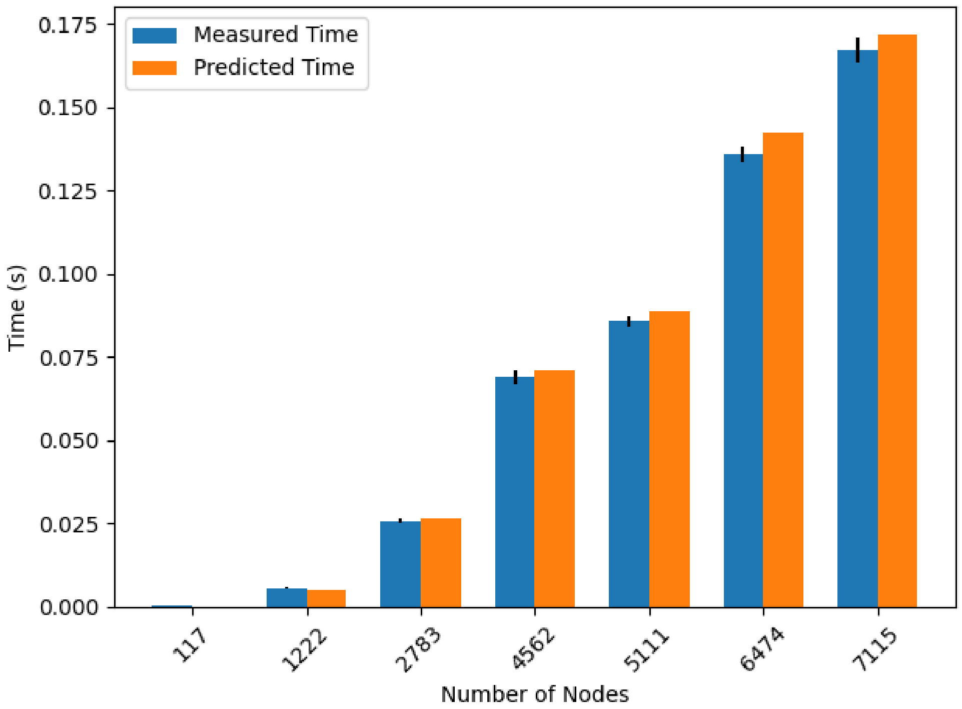
$$\begin{aligned}
T_{BFS} &= nT_{init} + nT_{while\_loop} + nT_{visit\_node} \\
T_{init} &= 2 \cdot 1.521484375 \\
T_{while\_loop} &= 14.5 + 11.2 + n\left(0.326 + 1.521484375 + 1.521484375\right) \\
T_{visit\_node} &= 2 \cdot 62.5 + 1.26 + 0.326 + 16.1
\end{aligned}$$

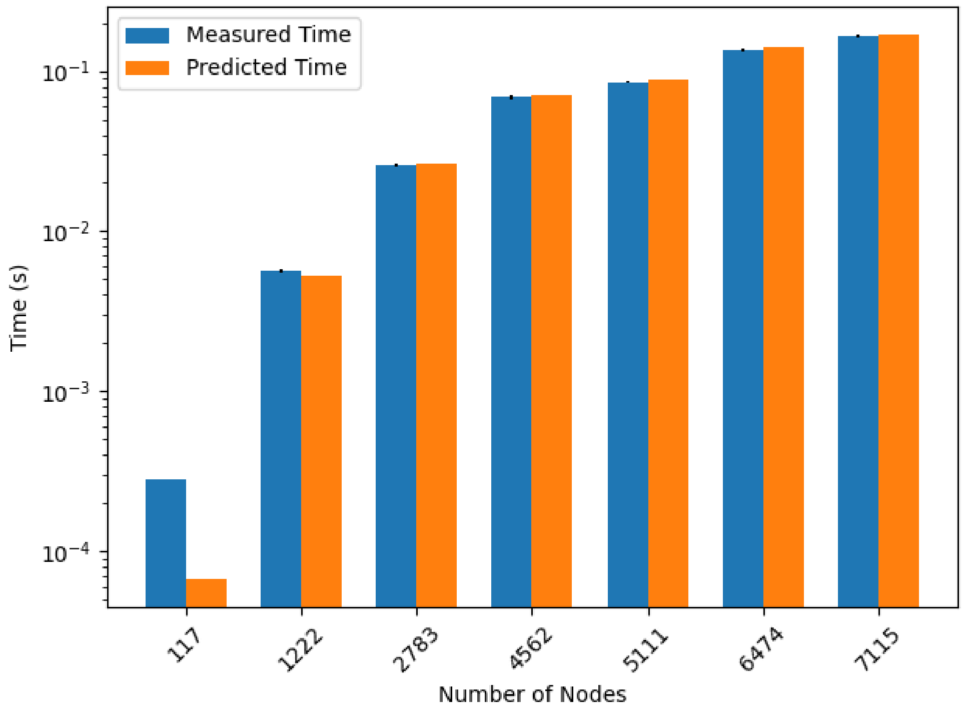
$$\begin{aligned}
T_{BFS} = n(&2 \cdot 1.521484375 + 14.5 + 11.2 + n\left(0.326 + 1.521484375 + 1.521484375\right) \\
&+ 2 \cdot 62.5 + 1.26 + 0.326 + 16.1) \\
&= \boxed{171.429n + 3.36897n^2}
\end{aligned}$$

Accuracy


0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
Time (s)
Measured Time
Predicted Time
117
1222
2783
4562
5111
6474
7115
Number of Nodes

Accuracy


Measured Time
Predicted Time
Time (s)
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
117
1222
2783
4562
5111
6474
7115
Number of Nodes

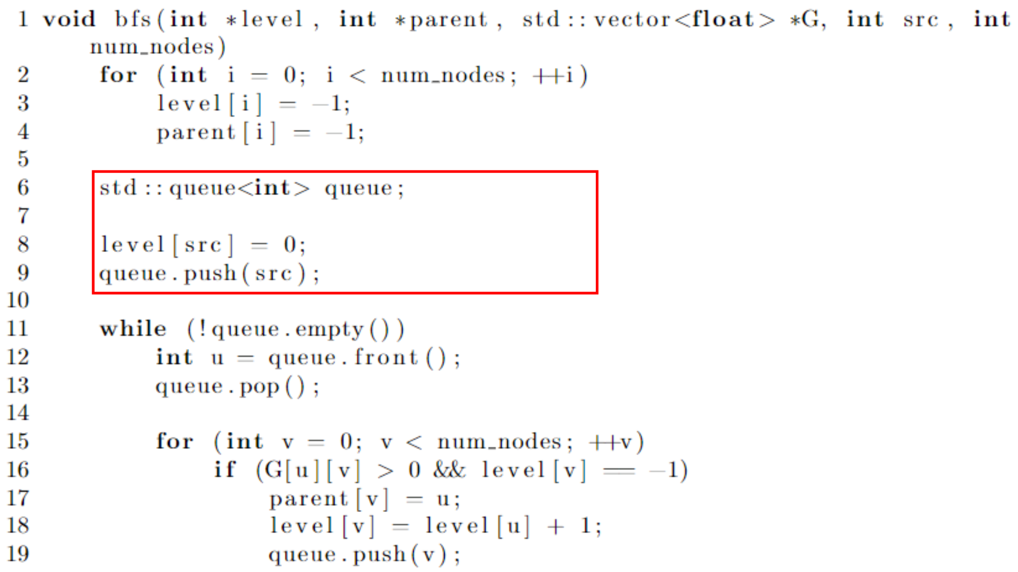

Measured Time
Predicted Time
Time (s)
10^{-4}
10^{-3}
10^{-2}
10^{-1}
117
1222
2783
4562
5111
6474
7115
Number of Nodes

Constructing symbolical model

$$T_{BFS} = nT_{init}$$

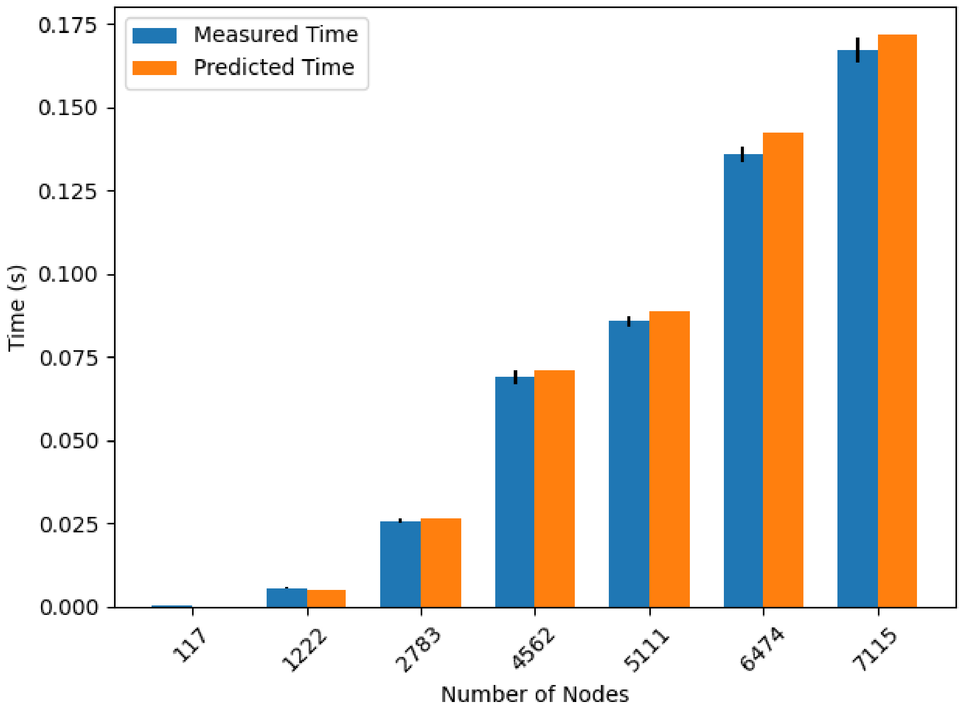
$$T_{init} = 2T_{mem\_write}$$

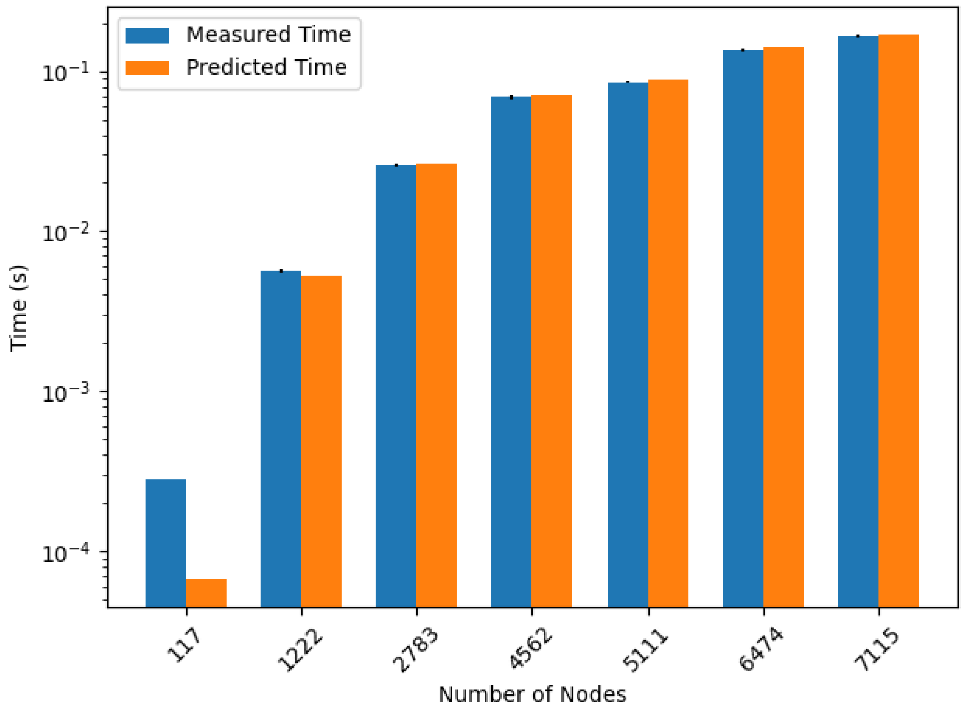
We skip this!



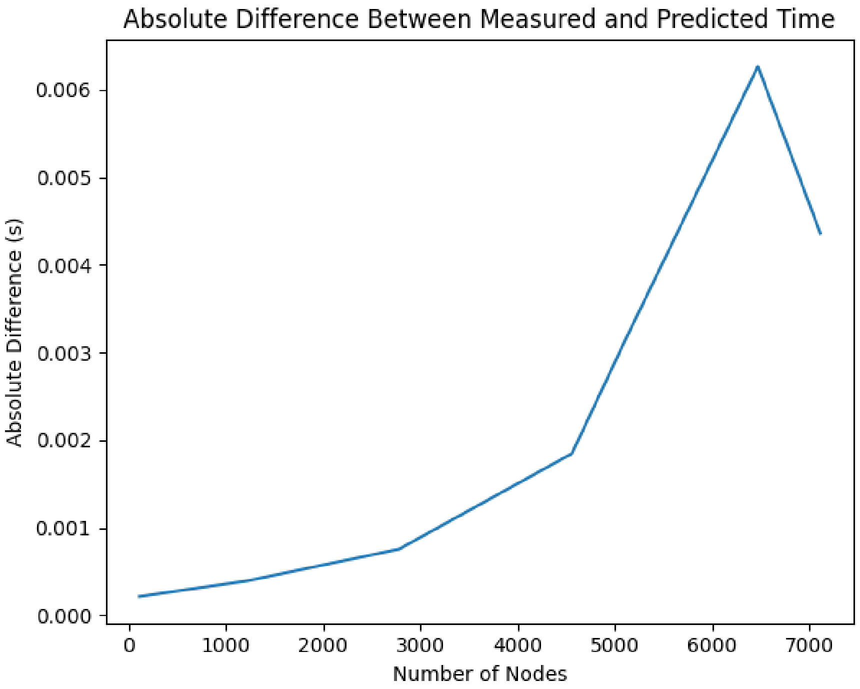
```
void bfs(int *level, int *parent, std::vector<float> *G, int src, int
    num_nodes)
    for (int i = 0; i < num_nodes; ++i)
        level[i] = -1;
        parent[i] = -1;

    std::queue<int> queue;

    level[src] = 0;
    queue.push(src);

    while (!queue.empty())
        int u = queue.front();
        queue.pop();

        for (int v = 0; v < num_nodes; ++v)
            if (G[u][v] > 0 && level[v] == -1)
                parent[v] = u;
                level[v] = level[u] + 1;
                queue.push(v);
```

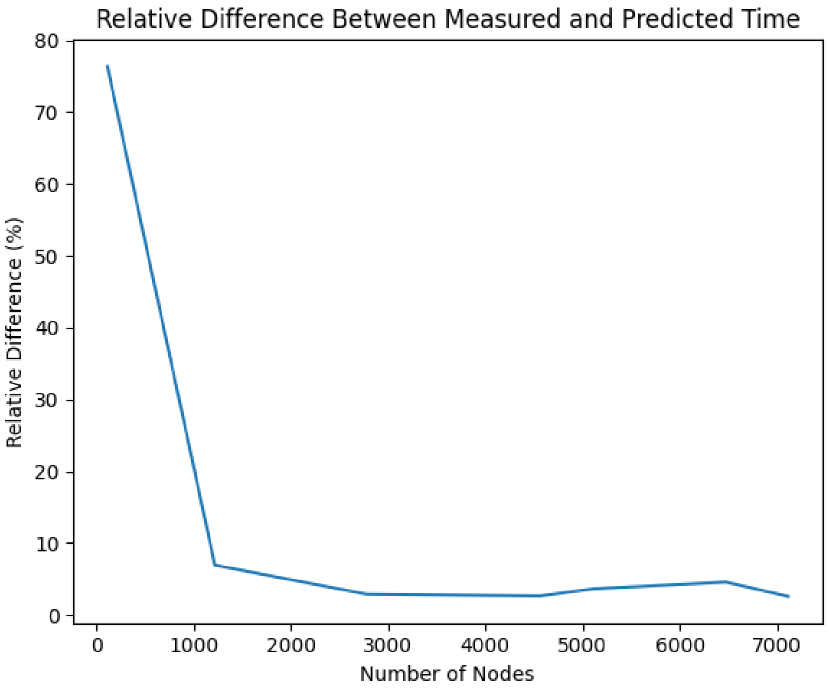
Accuracy


Measured Time
Predicted Time
Time (s)
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
117
1222
2783
4562
5111
6474
7115
Number of Nodes


Measured Time
Predicted Time
Time (s)
10^{-1}
10^{-2}
10^{-3}
10^{-4}
117
1222
2783
4562
5111
6474
7115
Number of Nodes

Accuracy


Absolute Difference Between Measured and Predicted Time
Absolute Difference (s)
0.000
0.001
0.002
0.003
0.004
0.005
0.006
0
1000
2000
3000
4000
5000
6000
7000
Number of Nodes


Relative Difference Between Measured and Predicted Time
Relative Difference (%)
0
10
20
30
40
50
60
70
80
0
1000
2000
3000
4000
5000
6000
7000
Number of Nodes