THE UNIVERSITY OF SYDNEY

SOFT3410 Assignment 2

Due: 11:59pm Wednesday, 25 November 2020

This assignment is worth 20% of your final assessment

Part 1 - Task description

The objective is to write a program that will model the static heat distribution of a room with a fireplace using a stencil pattern. Although a room is 3 dimensional, we will be simulating the room with 2 dimensions. The room is 10 feet wide and 10 feet long with a fireplace along one wall as depicted in Figure 1. The fireplace is 4 feet wide and is centered along one wall (it takes up 40% of the wall, with 30% of the walls on either side). The fireplace emits $100°$C of heat (although in reality a fire is much hotter). The walls are considered to be $20°$C. The boundary values (the fireplace and the walls) are considered to be fixed temperatures. The room temperature is initialized to $20°$C.

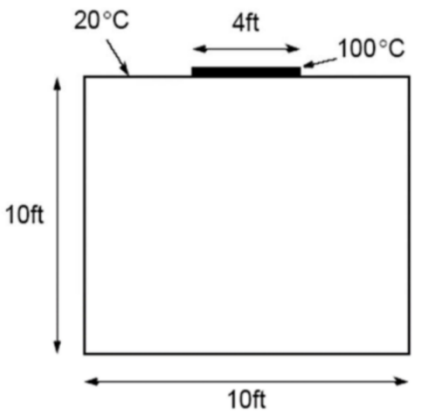


Figure 1: 10x10 Room with a Fireplace

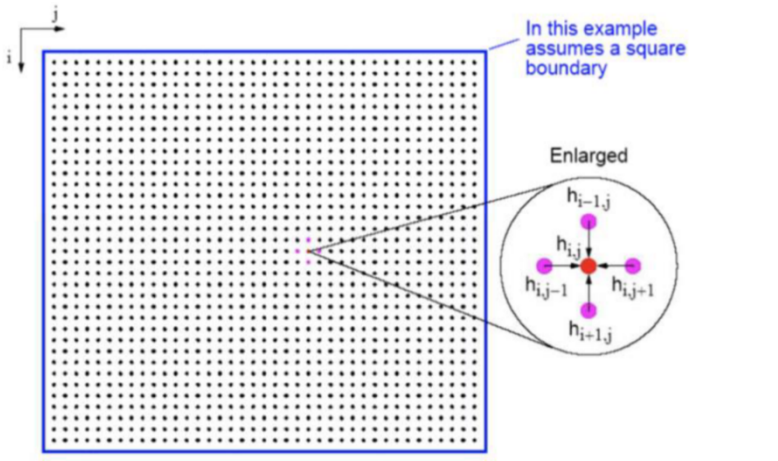


Figure 2: Determining Heat Distribution by a Finite Difference Method

We can find the temperature distribution by dividing the area into a fine mesh of points $h_{i,j}$. The temperature at an inside point can be taken to be the average of the temperatures of the four neighboring points, as illustrated in *Figure 2*.

For this calculation, it is convenient to describe the edges by points adjacent to the interior points. The interior points of $h_{i,j}$ are where $0 < i, 0 < j < n$ [(n - 1) x (n - 1) interior points]. The edge points are when i = 0, i = n, j = 0, or j = n, and have fixed values corresponding to the fixed temperatures of the edges. Hence, the full range of $h_{i,j}$ is 0 <= i <= n, 0 <= j <= n, and there are (n + 1) x (n + 1) points. We can compute the temperature of each point by iterating the equation:

$$h_{i,j} = (h_{i-1,j} + h_{i+1,j} + h_{i,j-1} + h_{i,j+1})/4, (0 < i < n, 0 < j < n)$$

for a fixed number of iterations or until the difference between iterations of a point is less than some very small, prescribed amount. Suppose the temperature of each point is held in an array h[i][j] and the boundary points h[0][x], h[x][0], h[n][x], and h[x][n] (0 <= x <= n) have been initialized to the edge temperatures. The calculation as sequential code could be:

```
for (iteration = 0; iteration < T; iteration++)
{
    for (i = 1; i < n; i++) {
        for (j = 1; j < n; j++) {
            g[i][j] = 0.25 * (h[i - 1][j] + h[i + 1][j]
                    + h[i][j - 1] + h[i][j + 1]);
        }
    }
    /* update points */
    for (i = 1; i < n; i++) {
        for (j = 1; j < n; j++) {
            h[i][j] = g[i][j];
        }
    }
}
```

Using a fixed number of iterations. Notice that a second array g[][] is used to hold the newly computed values of the points from the old values. The array h[][] is updated with the new values held in g[][]. This is known as Jacobi iteration. Multiplying by 0.25 is done for computing the new value of the point rather than dividing by 4 because multiplication is usually more efficient than division.

Also, the above code can be improved to avoid actually copying the array into another array by using a 3-dimensional array of size N x N x 2, say A[i][j][x] and every iteration, alternate between x = 0 to x =1.

Q1 - 4 Points

Re-write the given sequential code above to avoid the unnecessary points updating (h[i][j] = g[i][j]) using a 3-dimensional array introduced above; and complete the C program to model the heat distribution of the room illustrated in Figure 1. Have N x N points and T iterations, where N and T are defined constants in the program. Test your C program on your computer with N = 100 and T= 5000 (Note that we also use same N = 100 and T = 5000 in Q2, Q3, and Q4); and in the report, print out the values of every 10 points from the final heat distribution matrix in a row-by-row fashion, i.e., forming the output as a 10*10 matrix.

Q2 - 2 Points

Parallelize the sequential code given above (with unnecessary points updating) and your new code from Q1 using OpenMP. Analyze the max absolute error of heat distribution matrix between your parallelized version of codes and the corresponding sequential version of codes given above.

Q3 - 2 Point

Profile the execution time for the four programs you got (2D sequential, 2D parallel, 3D sequential, and 3D parallel) using the default scheduling policy on 1, 2, 4, and 8 threads. Plot all your results in a chart.

Q4 - 2 Points

Try default, static, and dynamic scheduling policies on your parallel heat distribution 3D program with 8 threads and different chunk sizes (1, 4, 8). Report these results in a chart and discuss if any of these scheduling policies will cause false sharing and why. Note that it is possible that they don't present significant performance differences. You have to summarize your findings.

Marking

Before you attempt to write optimised code, you should first complete a working unoptimised version. Once you have the basic algorithm correct you can then begin to implement various optimisations.

10 **marks** You will need to submit a report that attempts to answer the above questions and submit your code to a USYD Github repository.

For your submission, your code submission must be uploaded onto USYD Github under the repo name `soft3410a2p1`. We will only consider the last submission before 11:59pm, 25th November, 2020.

You must also submit your report to canvas portal by 11:59pm, 25th November, 2020.

Part 2 - Task description

In this assignment we will implement the basic PageRank algorithm in the C programming language using a variety of parallel programming techniques to ensure peak performance is achieved.

The PageRank algorithm was developed in 1996 by Larry Page and Sergey Brin when they were graduate students at Stanford University. Google and other search engines compare words in search phrases to words in web pages and use ranking algorithms to determine the most relevant results.

PageRank assigns a score to a set of web pages that indicates their importance. The underlying idea behind PageRank is to model a user who is clicking on web pages and following links from one page to another. In this framework, important pages are those which have incoming links from many other pages, or have incoming links from other pages with a high PageRank score, or both.

The PageRank algorithm

PageRank is an iterative algorithm that is repeated until a stopping criteria is met. The last iteration gives us the result of the search, which is a score per web page. A high score indicates a very relevant web page whereas a low score indicates a not so relevant web page for a search. Sorting the web pages by their scores in descending order gives us the order for the result list of a search query.

For describing the PageRank algorithm we introduce the following symbols:

- S is the set of all web pages that we are computing the PageRank scores for
- $N = |S|$ is the total number of web pages
- $\mathbf{P} = [P_1^t, P_2^t, ..., P_N^t]$ is the vector of PageRank scores
- d is a dampening factor for the probability that the user continues clicking on web pages
- ϵ is the convergence threshold
- $\text{IN}(p)$ is the set of all pages in S which link to page p
- $\text{OUT}(p)$ is the set of all pages in S which page p links to

For the PageRank vector, we use the notation $\mathbf{P}_p^{(t)}$ to represent the PageRank score for page p at iteration t. We initialise the scores for all pages to an initial value of $\frac{1}{N}$ so that the sum equals 1.

$$\mathbf{P}_u^{(0)} = \frac{1}{N} \tag{1}$$

During each iteration of the algorithm, the value of $\mathbf{P}$ is updated as follows:

$$\mathbf{P}_u^{(t+1)} = \frac{1-d}{N} + d \sum_{v \in \text{IN}(u)} \frac{\mathbf{P}_v^{(t)}}{|\text{OUT}(v)|} \tag{2}$$

The algorithm continues to iterate until the convergence threshold is reached; that is, the algorithm terminates when PageRank scores stop varying between iterations. The PageRank scores have converged when the following condition is met:

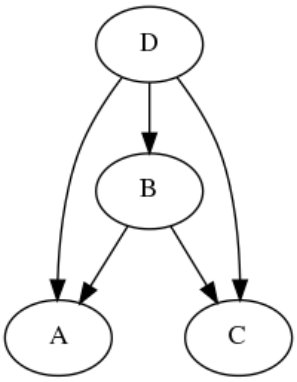
$$||\mathbf{P}^{(t+1)} - \mathbf{P}^{(t)}|| \leq \epsilon \tag{3}$$

The vector norm is defined to be the standard Euclidean vector norm; that is,
for some vector $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$:

$$||\mathbf{x}|| = \sqrt{\sum_i x_i^2} \tag{4}$$

Example

Our example has four web pages: $S = \{A, B, C, D\}$. In this example, $d = 0.85$ and $\epsilon = 0.005$.
The referencing structure of the web pages A, B, C, and D is given in the graph below.



$$\begin{aligned}
\text{IN}(A) &= \{B, D\} & \text{OUT}(A) &= \emptyset \\
\text{IN}(B) &= \{D\} & \text{OUT}(B) &= \{A, C\} \\
\text{IN}(C) &= \{B, D\} & \text{OUT}(C) &= \emptyset \\
\text{IN}(D) &= \emptyset & \text{OUT}(D) &= \{A, B, C\}
\end{aligned}$$

Each node represents a web page.
Edges in the graph indicate that the source of the edge is linking to the destination of the edge.

Initialise $\mathbf{P}^{(0)} = \frac{1}{N}$. Then perform the first iteration for each page.

$$\begin{aligned}
\mathbf{P}_A^{(1)} &= \frac{1-0.85}{4} + 0.85\left(\frac{\mathbf{P}_B^{(0)}}{|\{A,C\}|} + \frac{\mathbf{P}_D^{(0)}}{|\{A,B,C\}|}\right) = \frac{0.15}{4} + 0.85\left(\frac{0.25}{2} + \frac{0.25}{3}\right) \approx 0.214 \\
\mathbf{P}_B^{(1)} &= \frac{1-0.85}{4} + 0.85\left(\frac{\mathbf{P}_D^{(0)}}{|\{A,B,C\}|}\right) = \frac{0.15}{4} + 0.85\left(\frac{0.25}{3}\right) \approx 0.108 \\
\mathbf{P}_C^{(1)} &= \frac{1-0.85}{4} + 0.85\left(\frac{\mathbf{P}_B^{(0)}}{|\{A,C\}|} + \frac{\mathbf{P}_D^{(0)}}{|\{A,B,C\}|}\right) = \frac{0.15}{4} + 0.85\left(\frac{0.25}{2} + \frac{0.25}{3}\right) \approx 0.214 \\
\mathbf{P}_D^{(1)} &= \frac{1-0.85}{4} + 0.85\,(0) = \frac{0.15}{4} + 0 \approx 0.038
\end{aligned}$$

The initialisation and first iteration result in the following values for $\mathbf{P}$:

t	A	B	C	D
0	0.250	0.250	0.250	0.250
1	0.214	0.108	0.214	0.038

Next, we check whether or not the algorithm has converged.

$$\begin{aligned}
||\mathbf{P}^{(1)} - \mathbf{P}^{(0)}|| &= ||\{-0.036, -0.142, -0.036, -0.215\}|| \\
&= \sqrt{0.0677} \\
&\approx 0.260
\end{aligned}$$

Since $0.260 \not\leq 0.005$ (ϵ), we perform another iteration.

$$\begin{aligned}
\mathbf{P}_A^{(2)} &= \frac{1-0.85}{4} + 0.85\left(\frac{\mathbf{P}_B^{(1)}}{|\{A,C\}|} + \frac{\mathbf{P}_D^{(1)}}{|\{A,B,C\}|}\right) = \frac{0.15}{4} + 0.85\left(\frac{0.108}{2} + \frac{0.038}{3}\right) \approx 0.094 \\
\mathbf{P}_B^{(2)} &= \frac{1-0.85}{4} + 0.85\left(\frac{\mathbf{P}_D^{(1)}}{|\{A,B,C\}|}\right) = \frac{0.15}{4} + 0.85\left(\frac{0.038}{3}\right) \approx 0.048 \\
\mathbf{P}_C^{(2)} &= \frac{1-0.85}{4} + 0.85\left(\frac{\mathbf{P}_B^{(1)}}{|\{A,C\}|} + \frac{\mathbf{P}_D^{(1)}}{|\{A,B,C\}|}\right) = \frac{0.15}{4} + 0.85\left(\frac{0.108}{2} + \frac{0.038}{3}\right) \approx 0.094 \\
\mathbf{P}_D^{(2)} &= \frac{1-0.85}{4} + 0.85\,(0) = \frac{0.15}{4} + 0 \approx 0.038
\end{aligned}$$

Which leaves us with the following three iterations of $\mathbf{P}$:

t	A	B	C	D
0	0.250	0.250	0.250	0.250
1	0.214	0.108	0.214	0.038
2	0.094	0.048	0.094	0.038

Again, we check whether or not the algorithm has converged:

$$||\mathbf{P}^{(2)} - \mathbf{P}^{(1)}|| \approx 0.180 \not\leq \epsilon$$

Since convergence has not been reached, we perform another iteration.

$$\begin{aligned}
\mathbf{P}_A^{(3)} &= \frac{1-0.85}{4} + 0.85\left(\frac{\mathbf{P}_B^{(2)}}{|\{A,C\}|} + \frac{\mathbf{P}_D^{(2)}}{|\{A,B,C\}|}\right) = \frac{0.15}{4} + 0.85\left(\frac{0.048}{2} + \frac{0.038}{3}\right) \approx 0.069 \\
\mathbf{P}_B^{(3)} &= \frac{1-0.85}{4} + 0.85\left(\frac{\mathbf{P}_D^{(2)}}{|\{A,B,C\}|}\right) = \frac{0.15}{4} + 0.85\left(\frac{0.038}{3}\right) \approx 0.048 \\
\mathbf{P}_C^{(3)} &= \frac{1-0.85}{4} + 0.85\left(\frac{\mathbf{P}_B^{(2)}}{|\{A,C\}|} + \frac{\mathbf{P}_D^{(2)}}{|\{A,B,C\}|}\right) = \frac{0.15}{4} + 0.85\left(\frac{0.048}{2} + \frac{0.038}{3}\right) \approx 0.069 \\
\mathbf{P}_D^{(3)} &= \frac{1-0.85}{4} + 0.85\,(0) = \frac{0.15}{4} + 0 \approx 0.038
\end{aligned}$$

Again, we check whether or not the algorithm has converged:

$$||\mathbf{P}^{(3)} - \mathbf{P}^{(2)}|| \approx 0.040 \not\leq \epsilon$$

Since convergence has not been reached, we perform another iteration.

$$\begin{aligned}
\mathbf{P}_A^{(4)} &= \frac{1-0.85}{4} + 0.85\left(\frac{\mathbf{P}_B^{(3)}}{|\{A,C\}|} + \frac{\mathbf{P}_D^{(3)}}{|\{A,B,C\}|}\right) = \frac{0.15}{4} + 0.85\left(\frac{0.048}{2} + \frac{0.038}{3}\right) \approx 0.069 \\
\mathbf{P}_B^{(4)} &= \frac{1-0.85}{4} + 0.85\left(\frac{\mathbf{P}_D^{(3)}}{|\{A,B,C\}|}\right) = \frac{0.15}{4} + 0.85\left(\frac{0.038}{3}\right) \approx 0.048 \\
\mathbf{P}_C^{(4)} &= \frac{1-0.85}{4} + 0.85\left(\frac{\mathbf{P}_B^{(3)}}{|\{A,C\}|} + \frac{\mathbf{P}_D^{(3)}}{|\{A,B,C\}|}\right) = \frac{0.15}{4} + 0.85\left(\frac{0.048}{2} + \frac{0.038}{3}\right) \approx 0.069 \\
\mathbf{P}_D^{(4)} &= \frac{1-0.85}{4} + 0.85\,(0) = \frac{0.15}{4} + 0 \approx 0.038
\end{aligned}$$

Again, we check whether or not the algorithm has converged:

$$||\mathbf{P}^{(4)} - \mathbf{P}^{(3)}|| = 0 \leq \epsilon$$

We have now reached convergance, so the algorithm terminates and the values of $\mathbf{P}^{(4)}$ are the final PageRank scores for each of the pages. If this were a query, the resulting ranks would be A, C, B, D where we arbitrarily rank page A before page C since they have the same score.

Implementation details

In the header file **pagerank.h** we have provided the function **`read_input`** that will process the input file for you. The **`read_input`** function will output **error** if the input is malformed in any way and will free any memory that has been previously allocated. We have also provided various structs in the header file that may be useful, including a struct for holding pages and a struct that is a linked list of pages. The function **`read_input`** returns the input data in these structs, with the following form:

```
/* singly linked list to store all pages */
struct list {
  node* head; /* pointer to the head of the list */
  node* tail; /* pointer to the tail of the list */
  int length; /* length of the entire list */
};

/* struct to hold a linked list of pages */
struct node {
  page* page; /* pointer to page data structure */
  node* next; /* pointer to next page in list */
};

/* struct to hold a page */
struct page {
  char name[21]; /* page name */
  int index;     /* index of the page */
  int noutlinks; /* number of outlinks from this page */
  list* inlinks; /* linked list of pages with inlinks to this page */
};
```

Warning: Do not add other files or modify the included header file or main function.
We will replace these files with our own copy when marking your code.

Program input and output

```
<number of cores>
<dampening factors>
<number of pages>
<name of web page>
...
<number of edges>
<source page> <destination page>
...
```

The first line specifies the number of cores that are available on the machine. You will need to take this into account when you are optimising the performance of your program when using parallelism.

This is followed by the the dampening factor to be used in the algorithm and the total number of web pages. The names of the web pages follow, one per line, where each name may be a maximum of 20 characters long. After declaring the names of the web pages, the number of edges in the graph is given, followed by each edge in the graph specified by its source and destination page.

The **`read_input`** function outputs **error** and will terminate if there is invalid input, or there are any memory allocation errors, or a page name exceeds the maximum length, or the dampening factor is not in the range $0 \leq d \leq 1$, a page is declared twice, or an edge is defined to a nonexistent page.

```
2
0.85
4
A
B
C
D
5
D A
D B
D C
B A
B C
```

This example has four web pages with names A, B, C, and D
There are five edges: $D \to A$, $D \to B$, $D \to C$, $B \to A$, and $B \to C$

The output is the list of scores per web page

```
A 0.0686
B 0.0481
C 0.0686
D 0.0375
```

The scores are ordered in the same order they were defined in the input.

For printing the score, use the format string **`"%s %.4lf\n"`**
For all test cases set $\epsilon = 0.005$, use the constant defined as **`EPSILON`**

Marking

Before you attempt to write optimised code, you should first complete a working unoptimised version. Once you have the basic algorithm correct you can then begin to implement various optimisations.

7 **marks** are assigned based on your code submission and the *correctness* of your program. This component will use our own test cases that cover every aspect of the specification. To pass test cases your solution must produce the correct output within a reasonable time limit. With the given scaffold, do not modify sections that have been outlined.

8 **marks** are assigned based on the *report* of your solution and comparison to your single threaded implementation. Observe, document and explain the performance behaviours through your own benchmark suite.

For your submission, your code submission must be uploaded onto USYD Github under the repo name `soft3410a2p2`. We will only consider the last submission before 11:59pm, 25th November, 2020.

You must also submit your report to canvas portal by 11:59pm, 25th November, 2020.

Note: Part 2 contains 5 bonus points.

Academic declaration

By submitting this assignment you declare the following:

I declare that I have read and understood the University of Sydney Student Plagiarism: Coursework Policy and Procedure, and except where specifically acknowledged, the work contained in this assignment/project is my own work, and has not been copied from other sources or been previously submitted for award or assessment.

I understand that understand that failure to comply with the Student Plagiarism: Coursework Policy and Procedure can lead to severe penalties as outlined under Chapter 8 of the University of Sydney By-Law 1999 (as amended). These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgement from other sources, including published works, the internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.

I realise that I may be asked to identify those portions of the work contributed by me and required to demonstrate my knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.

I acknowledge that the School of Information Technologies, in assessing this assignment, may reproduce it entirely, may provide a copy to another member of faculty, and/or communicate a copy of this assignment to a plagiarism checking service or in-house computer program, and that a copy of the assignment may be maintained by the service or the School of IT for the purpose of future plagiarism checking.