Marcus Alexander Karmi September

Advantages of Parallel Compared with Serial Computation, for Route-Planning Problems and How Increased Data Size May Effect This

Computer Science Tripos – Part II

Clare College

January 7, 2022

Declaration

I, Marcus Alexander Karmi September of Clare College, being a candidate for Part II of the Computer Science Tripos, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose. I am content for my dissertation to be made available to the students and staff of the University.

Signed Marcus Alexander Karmi September

Date XX. XX. YYYY.

Proforma

Candidate number:	**2244E**
Project Title:	**Advantages of Parallel Compared with Serial Computation, for Route-Planning Problems and How Increased Data Size May Effect This**
Examination:	**Computer Science Tripos – Part II**
Year:	**2022**
Word Count:	**????**
Final line count:	**??**
Project Originator:	**Dr Jagdish Modi**
Supervisor:	**Dr Jagdish Modi**

Original Aims of the Project

TODO

Worked completed

TODO

Special Difficulties

None.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Some text

Chapter 2

Preparation

2.1 Some notes about the predecessor matrix

Let $\odot$ represent the (min, +)-matrix product.

Let A be the adjacency matrix for a weighted directed acyclic graph, where all weights are positive (cannot be zero).

Definition 1. *A matrix P of size $n \times n$ is a predecessor matrix if for every pair i, j, we have $d(i, j) = d(i, P(i, j)) + w(P(i, j), j)$, where $d(i, j)$ is the distance from i to j.*

This definition gives us a procedure to retrieve a shortest path between any pairs of nodes i and j, assuming that the two nodes are connected and that the path is *simple*, in other words does not revisit any nodes. The procedure works as follows: Start with the pair (i, j) and keep replacing j with the predecessor $P(i, j)$ until $i = j$, keeping track of the nodes j that are visited.

When computing $A^{(k)}$, we want the product to give us the weight of all paths of length $\leq 2^k$, not just paths of length equal to 2^k. To achieve this, it is necessary to set $A[i, i] := 0$ for all $i = 1 \dots n$. This allows paths of length $< 2^k$ to be represented by one or more 0-weight cyclic edges. For instance, we could have a 3-length path be represented as $a \to b \to c \to c$ in $A^{(2)}$. However, this invalidates one of the assumptions behind the procedure to reconstruct paths from the predecessor matrix P: The paths must be *simple*. We can mitigate this problem by adding another constraint to the predecessor matrix P: If there is a simple path $i \to^* j$ of length $\leq 2^k$ that consists of 1 or more edges, we have $P(i, j) \neq j$. I will get to a proof of why this causes our procedure of reconstructing paths to always terminate and give a simple path in a moment, if we construct P using algorithm 1.

Definition 2. *A witness for $(A \odot B)[i, j]$ is the element $W[i, j]$ of the witness matrix. This witness is the index k that gives $A[i, k] + B[k, j] = (A \odot B)[i, j]$ in other words, it is the argmin of $A[i, k] + B[k, j]$ in the $\odot$-product.*

With this definition, we can give our algorithm for computing the transitive closure of a graph G captured by adjacency matrix A (see algorithm 1).

We now argue for why the returned predecessor matrix is correct, where *correct* means that it will cause

Algorithm 1: Transitive closure of graph with adjacency matrix A

Data: A
Result: T, P

1 $A^{(0)} \leftarrow A$
2 For all i, $A^{(0)}[i,i] \leftarrow 0$
3 For all i and j, $P^{(0)} \leftarrow j$ if $A^{(0)}[i,j] = \infty$, i otherwise
4 **for** $k = 1, \ldots, \log n$ **do**
5 $A^{(k)} \leftarrow (A^{(k-1)})^2$
6 $W^{(k)} \leftarrow$ witness matrix for the product above
7 **for** *all pairs i and j* **do**
8 **if** $W^{(k)}[i,j] = j$ **then**
9 $P^{(k)}[i,j] \leftarrow P^{(k-1)}[i,j]$
10 **else**
11 $P^{(k)}[i,j] \leftarrow P^{(k-1)}[W^{(k)}[i,j],\ j]$
12 **end**
13 **end**
14 **end**
15 $T \leftarrow A^{(\log n)}$
16 $P \leftarrow P^{(\log n)}$
17 **return** (T, P)

the above-described procedure for reconstructing paths to halt at $i = j$ for any pair $i \neq j$ if there is a simple path of non-zero length from i to j. We do so by induction on the k.

After iteration k, we claim $P^{(k)}$ is a correct predecessor matrix for all paths $i \rightarrow^* j$ of length $\leq 2^k$. The base case $k = 0$ follows from line 3 in our algorithm. When $k > 0$, assume there is a path of length $\leq 2^k$ from i to j. If we let $w = W^{(k)}[i,j]$, we get by induction that there are paths $i \rightsquigarrow w$ and $w \rightsquigarrow j$ (which are simple if $i \neq w$ and $w \neq j$, respectively) which are each of length $\leq 2^{k-1}$. Consider the case $w = j$: This triggers the if-condition on line 8, so we use the same predecessor as previously, which must be correct by induction. Now consider $w \neq j$: The predecessor of path $i \rightsquigarrow j$ is the same as predecessor of path $w \rightsquigarrow j$, which we assign on line 11.

I have written the above code for ease of reading and to make the proof simpler. When implementing this, it is possible to perform both the matrix multiplication, the witness matrix computation and compute the predecessor matrix, all in one go, allowing everything to be computed in parallel. This can be done as follows:

Algorithm 2: Computation done by $PE(i, j)$

```
   Data: i, j, n, A, W, P
 1 a ← ∞
 2 W[i, j] ← j
 3 for w = 1, ..., n do
   |  /* Prior to this, PE(i, j) has received P[w, j], A[w, j] and A[i, w] from the
   |     appropriate processing elements.                                          */
 4 |  if A[i, w] + A[w, j] < a then
 5 |  |  a ← A[i, w] + A[w, j]
 6 |  |  W[i, j] ← w // The matrix W is not needed here, but given here for
   |  |     completeness
 7 |  |  if w = j then
 8 |  |  |  P[i, j] ← P[i, j]
 9 |  |  else
10 |  |  |  P[i, j] ← P[w, j]
11 |  |  end
12 |  end
13 end
```

Chapter 3

Implementation

Even more text

Chapter 4

Evaluation

text

Chapter 5

Conclusions

text