NOC Router based PageRank Algorithm ECE6463 Project Final Report

Sijie Liu, Yi Qin

Abstract—
This project is an NOC based implementation of PageRank algorithm, the cornerstone of web search engine, in Verilog language. In the other words, it is an on chip implementation instead of point-to point. Our design consists of three parts: CPU, NOC and a sorting module. The CPU is used to initialize, calculate and update the value of scores once the adjacency matrix is given. The NOC is used to transmit updated score between local CPU and distant CPU. The sorting module is used to sort the converged scores in a descending order. In this report, we assume there are 64 websites in the network and the network is divided into four parts with 16 websites each. Only the top 10 websites' ID and score are required to be displayed.

1 Overall Design and Target

OUR target is to generate the converged score of every website by synchronically updating their scores a certain times, namely, a score will not be updated twice in succession before every other score is updated. Every node uses the score of nodes from the previous iteration instead of the updated value. Since the network was split up into 4 parts, every node should be able to get others' score to do its update through the NOC. It's also required to detect whether the scores are converged. Once a convergence is detected, the top 10 websites can be generated by a bubble sorting module.

2 Architecture

Our design is mainly composed of four CPUs, an NOC module with four routers(one for each CPU), and a sorting module.

2.1 Overall Structure

Every CPU's data output port is connected to the input port of one router in NOC to send its updated scores to other CPUs. The data output port of every router is also connected to the input port of its corresponding CPU to transmit the scores from other routers so its local CPU can do further update. Every router is connected to other three routers to avoid packet transmission across router, which may lead to more latency. Only one hop is enough for every case of packet transmission. The data output ports of the four CPUs are also connected to the input port of the sorting module to sort the converged scores.
The inputs of our overall structure are the clock signal, the reset signal, the adjacency matrix and the weights of every nodes(can be derived from the adjacency matrix). The clock and the reset signal work on every parts above to make the whole design synchronic. The adjacency matrix and the weights are inputs of CPU. The overall outputs are top 10 scores, IDs and a done signal which is asserted when the computation is finished then top10 ID and top10 scores can be read off. They are all from output ports of the sorting module.

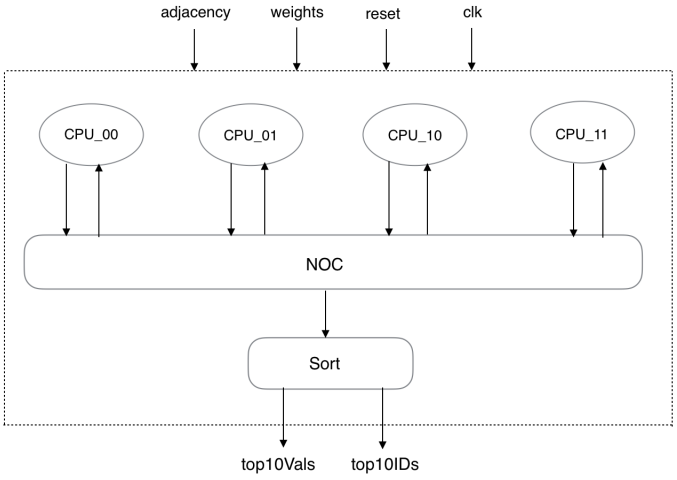

Fig. 1: Packet Format

2.2 CPU Details

The CPU consists of one disassemble part, two convert module, one multi-bit multiplier, one multi-bit adder, one record array and one output port.
The disassemble part detects whether the incoming data is valid, if it is, the packet will be disassembled into score and ID.
The two convert module can convert the input adjacency matrix and node weights from 1D to 2D.
The multiplier is used to compute the ”temp” value based on the node score received from router. Since we decided to use 24 bits to represent score(details in 3.3), a 72 bits (24*3) multiplier can meet the requirement.
The 24-bits adder is able to add the ”temp” value or nothing to an updating score. Whether the ”temp” value will be added is determined by checking the corresponding bit in the adjacency matrix. This adder can be reused 16 times for its 16 local websites every time one score is received. It means the procedure of updating every score is sequential

since the score input is in sequential(details in 3.1). With only one multiplier and one adder, the area cost is significantly reduced.

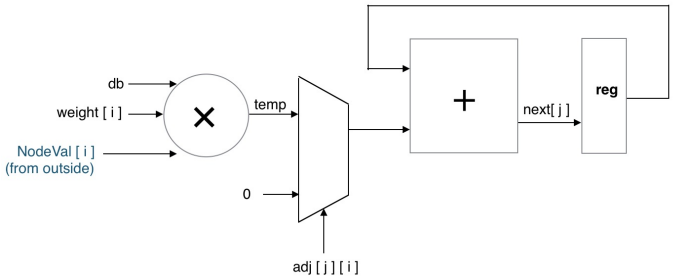

Fig. 2: CPU detail

The record array is designed to record which website's score has been received. Every time here comes a score, the bit with same ID in the array is asserted. Once the array is all asserted, it means all website's scores have passed by the CPU. Thus the CPU can know one iteration is complete. Every score is updated and ready to be sent. Then the array is set to all zero for the next iteration. In this way the CPU could send the old score and compute the new score at the same time, allowing latency to be reduced.
In order to reduce area cost, the data is without tail during computation. When the scores are ready to be sent, they will be added their tail at the output port. The output also has an writenable signal output, it will be set to zero if CPU is not sending score.

2.3 Router Details

There are four ports in our router. One for the local CPU and three for the other routers. Each router is mainly composed of the four FIFOs and one arbiter.
The arbiter is able to grant transmit query fairly by using blind Round-Robin algorithm(avoiding starvation can be achieved using an algorithm similar to Iterative Round-Robin Matching (iRRM), which means the priority pointer will not update when the port with highest priority has a request and it's not granted, so there is an improved version of our design). If it is the local CPU that wants to send data, it will send to all four output ports. If the transmit request comes from another router, it will only be sent to local CPU. Such being a case extra bits for address is not required any more, allowing packet size to be smaller, so does the area cost.
In order to avoid junk traffic flow, there is an writenable signal at the output port from a router to another router. Once the valid scores are transmitted, the writenable is set to zero. The output port and the writenable signal is always set to zero by default unless there is a valid packet to be sent, avoiding the output hold-up problem which may leads to the last valid packet duplicated. Since there is also a similar signal at the CPU's output, all data in travel is valid. Thus the FIFOs are less likely to be full and the latency can be reduced.

2.4 Sorting Module Details

We implement our convergence rule in the sorting module. Since the data output ports of the four CPUs are connected to the input port of the sorting module, it can check if the updated score is converged(details in 3.2). The sorting block will begin to work once the convergence rule is satisfied. It uses bubble sort on the digits representing score in every packet to avoid disassembling the packet. We do not need to finish sorting procedure because only top10 is required and the bubble sort has its property(details in 3.5). So after 10 iterations, the done signal is asserted and the final output can be read off.

3 Discussion of Design Choices

3.1 Parallel Versus Sequential

There are two ways to implement the project, one way is that the CPU can get at most one packet every clock cycle to be used for updating its websites, this one is called sequential mode. The other way is called parallel mode which means CPU can get more than 1 packet every clokc cycle. Since we are not sure whether the parallel or the sequential mode will have less metric. We wrote one parallel version and one sequential version. When comparing their performance, for the sequential mode, the calculation module will be reused in every clock cycle, which cause less area but more slack and latency. On the other hand, in parallel mode, the CPU will process four packet in every clock cycle, so the calculation mode will not be reused but copied 4 same modules to process, which may cause more area but less slack and latency. Finally, after coding and synthesis, we find that the sequential mode has better performance and less metric, so the following analysis except metric and our code submitted are for sequential mode. Details of their performance comparison is in 4.1 and 4.3.

3.2 Score Calculation Principle

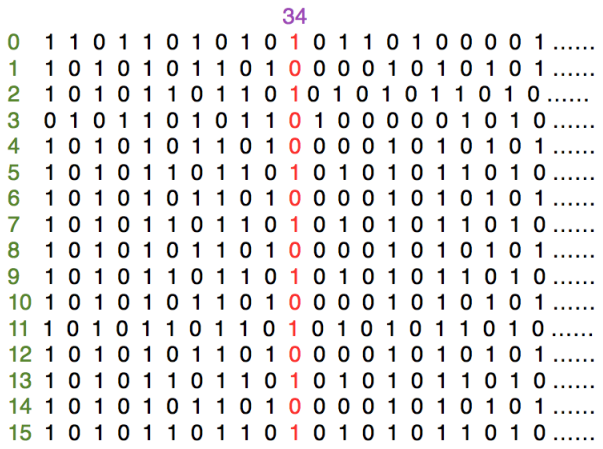

Fig. 3: Computing procedure

For the performance, our goals are higher frequency, less area and less latency. Among the three goals, less area is the most important.

In order to reduce the area cost, we mainly considered the score update procedure in CPU since it is the simplest way to do it. In homework 4, whether the scores are updated parallel or sequential, the updated is based on new value or old value all make a big difference on area cost. When considering about the query-respond architecture and the broadcast architecture, we found that the latency of query-respond architecture will be much better than broadcast theoretically, especially when the network connection is sparse. But this advantage suffers from synchronic problem, namely every CPU needs to wait for the CPU whose websites are with the most connection. Or the iterations will be messed up. So in fact this advantage is not that significant. But for broadcast architecture, even though it's latency is longer, we can make it with less area since there is no query and respond module and the packet can be much smaller.
In our design, the CPU will receive one packet in every clock. When CPU receives a new valid packet, the multiplier will calculate the "temp" and the adder will update all "nodeVal next" for the 16 local websites which have adjacency with the incoming node. So there will be at most 16 times addition for every "nodeVal next" and one multiplication for the "temp" value. In this way, We implement the update procedure sequentially which means dividing the "add" steps in update into separate clock, but for the 16 websites, their update progress are in parallel. If we reuse the adder for the 16 websites every clock, it will save much area because only one adder and one multiplier is required, there is even no need to use register to store the coming packet. Even though reusing the adder may lead to longer clock cycle, it's not that significant compared with the latency of the 72-bit multiplier. After every website packet passed by one CPU, all 16 local websites of the CPU finished their update. In this way, we do not need to wait for a certain packet before we begin to update the 16 local websites' score. CPUs can send data and calculate using received data at the same time. So we can reduce the number of clocks we need. Generally speaking, sending data and calculating at the same time is for less latency consideration, updating the "nodeVal next" sequentially is for less area consideration.

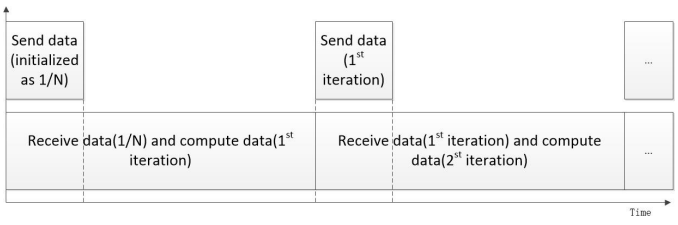

Fig. 4: CPU Activity

3.3 Convergence Rule

Based on the result we got from matlab, We found that in the worst case, 40 times update is required for K from 1 to 4 and p from 0.1 to 0.5. The top10ID will no longer change after 40 iterations even though their scores are still updating.
So we regard the output as converged after 40 iterations. We can also determine the fixed point representation based on this by checking whether the change in score can be distinguished after 40 iterations using a certain bits of number.
The precision test output is showed in Fig.6 and Fig.7. 23 bits value can hardly be distinguished at 40th iteration. So at last we decided to choose 24 bits to represent score.

3.4 The Packet width

This is the packet format of our transmission. We use 24 bit to store the score value of website and 6 bits to store website ID number. The LSB is the valid bit. If the valid bit is 0, the write enable signal of input port which the packet will be sent to will be set to 0.Then the trash packet will be dropped.

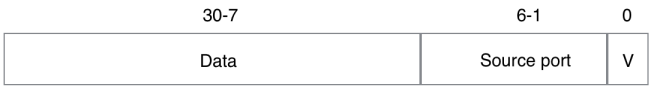

Fig. 5: Packet format

3.4.1 Testing the minimal precision to ensure the top10 right order

We want to check if the difference of score after 40 iterations can be distinguished using a certain bits of number. So we added a "break" when the difference is less than $1 \div 2^n$. When n=23, the difference can't be distinguished right at the 40th iterations. To be on the safe side, we tried n=24, the values shrink at about the 43th iteration. So 24 bits will meet our requirement.

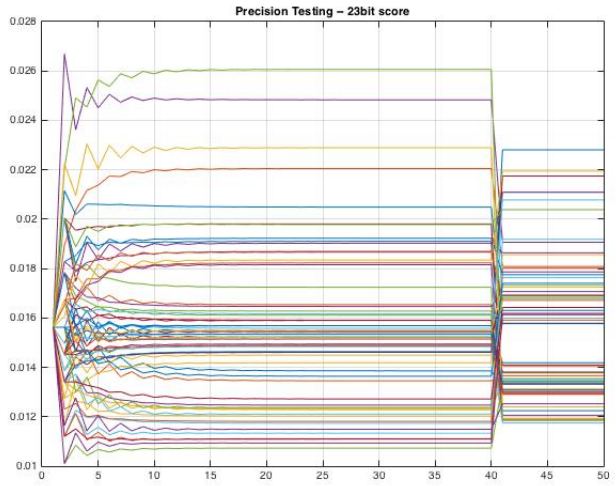

Fig. 6: Testing precision for 23bit

3.5 FIFO depth

Because every CPU would send its 16 scores at most every iteration. So receiving 16 valid packet, the write signal from CPU will be set to 0. Since transmission across router will not happen in our topology of NOC, every input port of router will receive 16 packets in all. Now, a FIFO with depth=16 can meet the requirement even in the worst case.

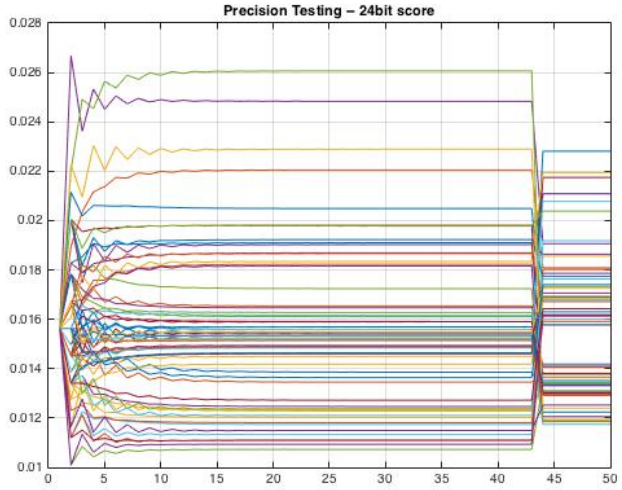

Fig. 7: Testing precision for 24bit

3.6 Sort theory

If there are 64 unsorted numbers, we implement the bubble sort in descending order from the value at bottom. After the 1st exchange iteration, the number on the top will be the biggest one. So the we can get top 10 values only after 10 iteration instead of 64 iterations.

4 Experimental Results and Discussion

4.1 Experimental results

Metric Calculation

Case	Area	CLK (ps)	No.clk	Metric
sequential	93635.205	1310	3235	121.4239
parallel	441338.637	670	1409	276.7857

4.2 Analysis of experimental result

For the parallel and sequential mode synthesis result, we find that although the sequential way has more slack and latency than parallel, its area is significantly small, which impact the result to a large extent. So, we submit the sequential code as our final result.

For the sequential way, in every clock, we only process one packet, on the other hand, for parallel way, we process four packet in every clock cycle. So, in parallel way, the calculation module will not be multiplexed, but copied three times, which will cause higher speed but larger area.

After all the data we have, such as area, slack, latency, we find that if we do not consider the power of area ($area^{1.5}$). We find that the metric result of sequential is 0.3968, the result of parallel is 0.4166. The difference between them is not obvious. So, for later design, we will firstly analyze what is the most important factor in metric and design the trip based on the ratio of factor. If our metric mainly consider the area other than slack or latency, we should pay more attention to sequential. And vice versa.

Top-10 websites ID and score value

From our design		From Matlab	
Top-10 ID	Top-10 score	Top-10 ID	Top-10 score
10	0.024565	10	0.024567
34	0.023542	34	0.023544
55	0.019695	55	0.019696
0	0.019655	0	0.019656
60	0.019384	60	0.019385
46	0.019126	46	0.019127
16	0.019112	16	0.019113
3	0.019087	3	0.019089
14	0.019080	14	0.019081
48	0.019017	48	0.019019

The output top10 ID and top10 score of our design and matlab shows that no reverse occurred and the score is accurate to some extent.

4.3 Result graph

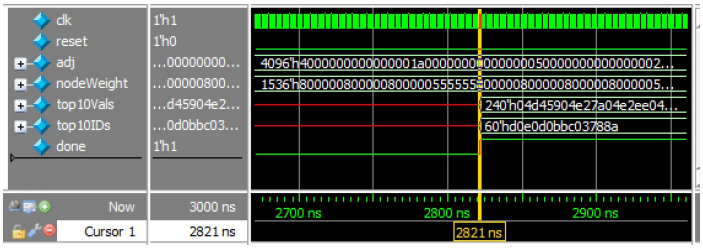

Fig. 8: Parallel Result

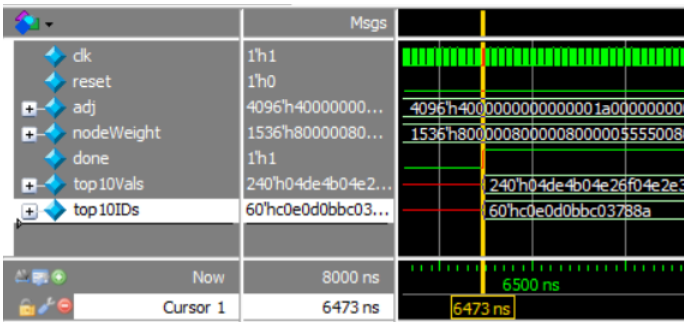

Fig. 9: Sequential Result

5 Further Improvement

Given more time, we would change the Round-Robin algorithm into Iterative Round-Robin Matching(iRRm) to avoid starvation. The converge rule could also be improved based on the variation of K(neighbours) and p(long links).

Acknowledgments

The authors would like to thank Professor Siddharth Garg for his time and energy devoted in teaching course ECE6463 and his generous help and guidance in helping us finishing this project.