# Parallel Random Access Machines

Morgan Powell

*If there is something I missed like: Reference or “” let me know.*

***Index page:***

Parallel Random Access Machine (PRAM)

Imperfect models which will only tangentially relate to a genuine parallel machine's performance, much as Turing Machines, will only tangentially relate to a real computer's performance.

Goal: Make it feasible to reason about and classify parallel algorithms, as well as derive complexity results (optimality, minimal complexity, and so on). One way to look at it is that it allows you to determine the "maximum parallelism" in an algorithm or a problem and create new algorithms.

Memory size is endless, as is the number of processors. However, nothing prohibits you from "folding" this back to something more practical.

There is no direct communication between processors; they interact through memory; they can work in an asynchronous fashion; each processor accesses any memory address in one cycle, and all processors typically perform the same algorithm in a synchronous fashion. Some processors can remain idle (for example, even-numbered processors may not work while odd-numbered processors do, and vice versa).

PRAM would need to have an Error control coding (ECC) which been helped improve the memory reliability for PRAM for being feasible for modern-day, but the main issue is having the ECC for Pram is costly in the area and decoding latency. Pram suffers the stand-alone of dealing which would need help to be sufficient for long run use. (Chengen, Yang)

Issues in implementing PRAM on a feasible parallel computational model are as discussed below;

1) Brent's Theorem

Brent's theorem connects the time and works complexity of parallel algorithms defined in the WT formalism to its running time on a -processor PRAM.

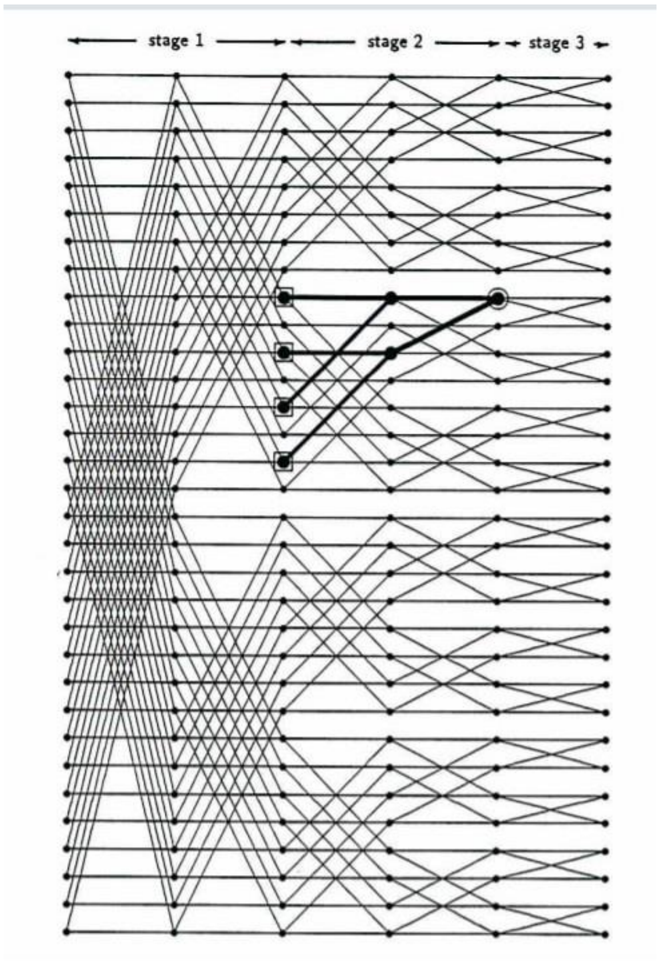
Theorem No. 1 (Brent 1974) A WT algorithm with work and step complexity can be emulated in no more than parallel steps on a -processor PRAM.

Proof: For each time step, get the number of the operation performed in the step. On a -processor PRAM, we mimic each step of the WT algorithm.

Parallel steps are achieved by scheduling activities on the processor in groups of operations at a time. If the last groups are not divided evenly, they may not have operations. In such a situation, we distribute the leftover operations amongst the processors with the smallest index. The time to simulate the step of the WT algorithm will be reduced as a result of this simulation technique.

The Butterfly Practical PRAM

Here a simple PRAM diagram format but does not show the Memory Unit Access on the comparison to "sample2" diagram which displays more structure format including [ Pp] and [Mp] to show an infinite amount of processor and memory from the Diagram version 2. The



Another issue with PRAM algorithm implementation is the fast Fourier transform problem. To explain more on that, the 'butterfly' algorithm.

The (n)input or (n=power of 2) for the butterfly is a directed graph with the n(log n+1) nodes which is viewed as n amount of rows of log n. For the butterfly, graph should be an equal amount to have an efficient butterfly graph.

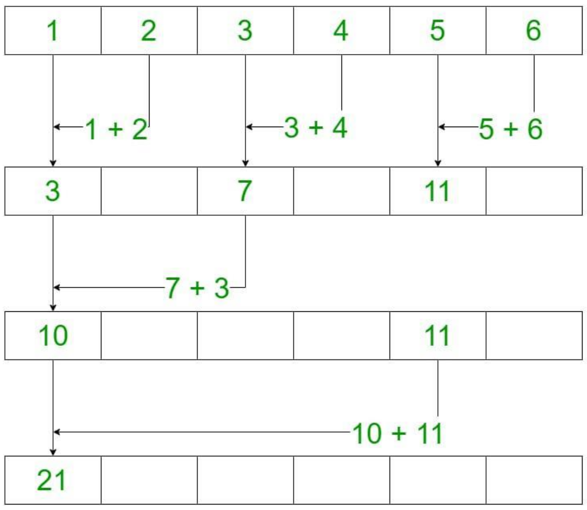
2) Creating effective parallel algorithms

PRAM methods have temporal complexities whereby both the number of processors and the size of the issues are variables.

Let be the optimum (or best known) sequential time complexities for the issue given a PRAM technique with running time.

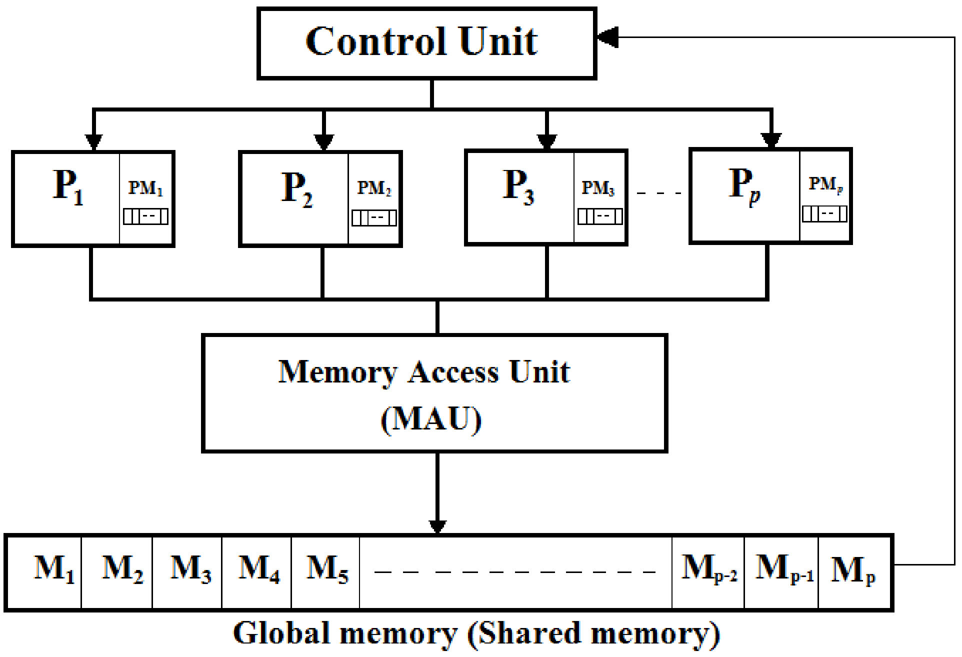
In conclusion, when comparing multiple work-efficient parallel algorithms for an issue, the ones with the lower step complexities are more scalable in the sense that they maintain optimum speedup over a wider variety of processors.

Example illustrating the PRAM model: Supposing someone wishes to add an array with N numbers. They, in general, iterate through the array using N steps to get the array's sum. Hence, if N is the array's size and for every step, we can assume the taken time to be 1 second. So, N seconds are taken in completing the iteration. That same operation can be more efficiently done using a CRCW model of a PRAM. Assuming an array of size N has N / 2 parallel processors, the time required to run is 4, which is less than N = 6 seconds in the figure below.



Here is a simple PRAM diagram format but does not show the Memory Unit Access on the comparison to “sample2” diagram which displays more structure format including [ Pp] and [Mp] to show an infinite amount of processor and memory from the Diagram version 2. The

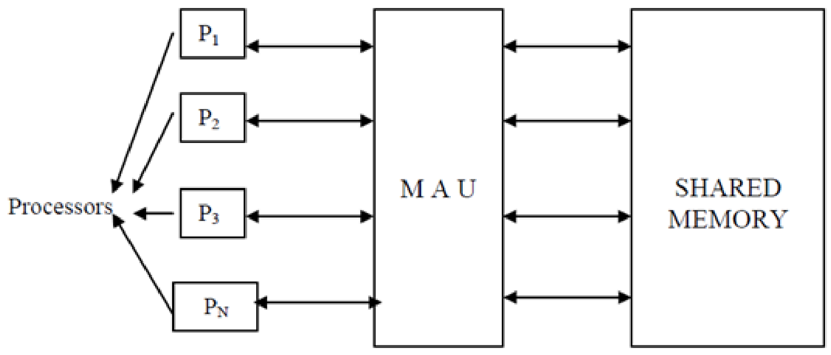
Finding more in-depth information for practical PRAMs online was hard to find as during my research there was a book about Practical PRAMs but it was not open to the public so it was impossible to get the information. As it was not open to the public for me to buy or get.


Control Unit
$P_1$ $PM_1$
$P_2$ $PM_2$
$P_3$ $PM_3$
$P_p$ $PM_p$
Memory Access Unit (MAU)
$M_1$ $M_2$ $M_3$ $M_4$ $M_5$ $M_{p-2}$ $M_{p-1}$ $M_p$
Global memory (Shared memory)

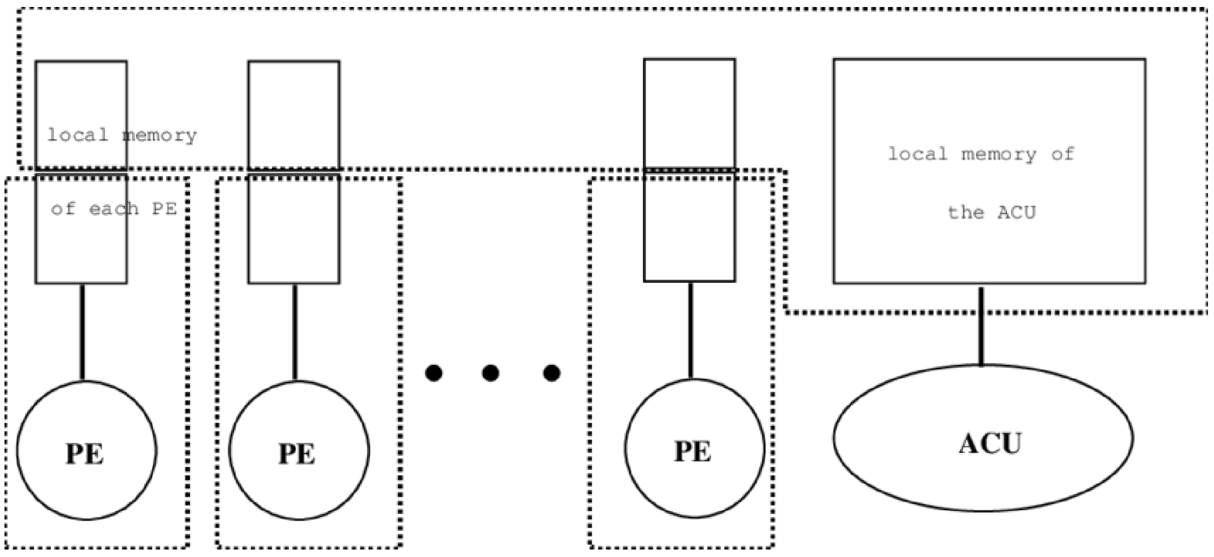
2nd Graph PRAM version.

This is a Global memory PRAM version and displays multiple processors which are attached to a block of memory/MAU. The PRAM model can be feasible and can grab a

It makes the complexity and the correctness structure to analyse the PRAM algorithms easier and more feasible than most of the Practical models of CRCW PRAM.



This Graph is a not feasible for any PRAM display as it does not present the individual (M1.. Mp-1,**M**p).



3rd PRAM MODEL DESIGN

This PRAM MODEL is not feasible and does not present more depth to the other PRAM Graph I have researched and for display for large scale does not include further details.

**<u>Types of PRAMs</u>**

**Block Pram –**
This Block Pram has a (+n) amount of collection of *p* Processors which displays their own local individual memory. The processors can execute all the same programs as once but also depends/may have to wait when until it's get given orders to execute a program/instruction. Its unique feature has the structure of the share0memory model

The Block PRAM structure is formatted in a user-defined array, which consists of a block of shared memory locations. The Block PRAM can be more feasible compared to EREW/CREW PRAM where it can execute all of the programs at the same time and also wait until receiving instructions. (Andrew,Chin)

It has a defined array of consisting blocks of the shared memory location that has a consecutive logical address.

What makes Block PRAM be so well because of it is a shared memory model of parallel computation but also the complexity theory. It makes sure it has no read or writes conflicts are allowed but in an arbitrary order which is sequential.

**(SLC) PRAM –**
**Single Level Cell PRAM**
The Single level cell won't be feasible compared to other PRAM as it consists of two parts, one it has the 'SET' stat corresponding to the low resistance crystalline phone or get

presented as state ‘1’,’2’) it has the Reset stat whole goal is to correspond to the high resistance amorphous phase/section which is also present as State ‘0’.

The temperature is set high but not too high but set to a sufficient level which provides a current to be noticed by a sense amplifier but it has to be low enough to avoid the write disturbance. (Chengen Yan, 2012)

**MLC (Multi-Level Cell) PRAM**. this helps improve storage density. However, it has the disadvantage of having lower reliability which might make some people lose interest.  Then again, it has the advantage of being less costly, which makes it very appealing in the market compared to other Prams, (Y.N. Hwang)
The MLC is described as having the resistance being changed mathematically by implementing a controlled pulse. Which the temperature change temperature and sense margin to write and justify within the operation cell. (Y.N Hawng)

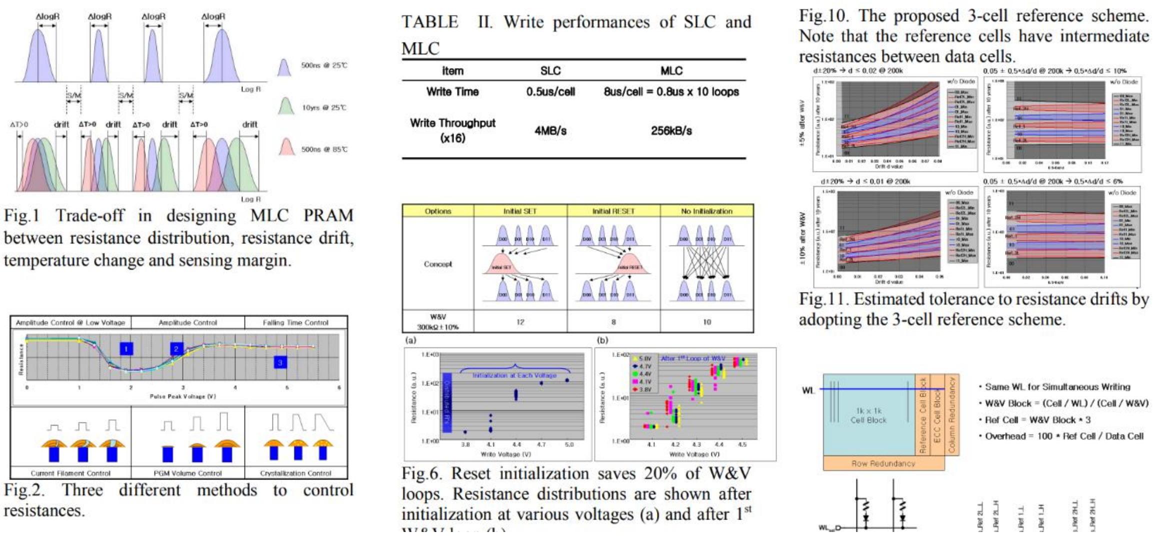


Fig.1 Trade-off in designing MLC PRAM between resistance distribution, resistance drift, temperature change and sensing margin.

Fig.2. Three different methods to control resistances.

TABLE II. Write performances of SLC and MLC

| item | SLC | MLC |
|---|---|---|
| Write Time | 0.5us/cell | 8us/cell = 0.8us x 10 loops |
| Write Throughput (x16) | 4MB/s | 256kB/s |

Fig.6. Reset initialization saves 20% of W&V loops. Resistance distributions are shown after initialization at various voltages (a) and after 1st

Fig.10. The proposed 3-cell reference scheme. Note that the reference cells have intermediate resistances between data cells.

Fig.11. Estimated tolerance to resistance drifts by adopting the 3-cell reference scheme.

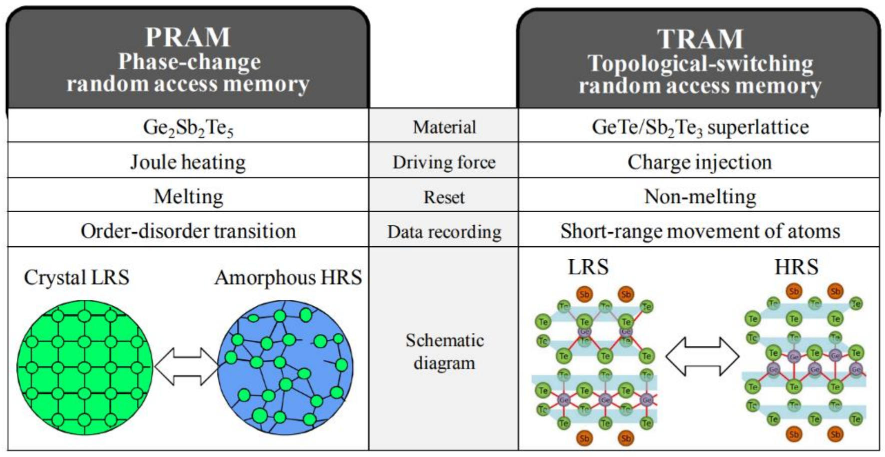
- Diagram of MLC PRAM - (Y.N. Hwang)

This Show the MLC and SLC PRAM performance from the diagram above.  This is the understanding of the control resistances, speed, temperature for each individual cell.

**DRAM** PRAM- is a more powerful version than PRAM for the time on my research. Which give it attraction to having a high performance and better active power compared to PRAM. DRAM energy consumption can be reduced by 23.5-94.7% for reducing the DRAM refreshes. Which is very interesting and helpful compare to PRAM.
 While Pram is researched and to be known as the next generation for main memory component. DRAM works as a cache which is due to having low latency and providing a smaller power consumption for the read and writes during its run time than PRAM which makes it more needed/feasible to include to have to help with PRAM. If there is a memory access request to DRAM and give a miss, then PRAM help fetch the data and send it to

DRAM to take in the data. Which I found very interesting  from research of effective Hierarchical PRAM - (N. Lu, I. -S. Choi- effective Hierarchical PRAM)

**TRAM - Topological0switching RAM(TRAM)**



| PRAM Phase-change random access memory | | TRAM Topological-switching random access memory |
|---|---|---|
| $Ge_2Sb_2Te_5$ | Material | $GeTe/Sb_2Te_3$ superlattice |
| Joule heating | Driving force | Charge injection |
| Melting | Reset | Non-melting |
| Order-disorder transition | Data recording | Short-range movement of atoms |
| Crystal LRS Amorphous HRS | Schematic diagram | LRS HRS |

Fig. 1 Comparison of PRAM and TRAM.

*Diagram from N.Takaura  - Fabrication of topological switching RAM(TRAM)*

TRAM has non-melting compared to Pram which melts. So PRAM would need a higher voltage point to increase the level of the resistance compared to TRAM is does not need a high vantage point and get to a high level of resistance to 1E+7 before PRAM.

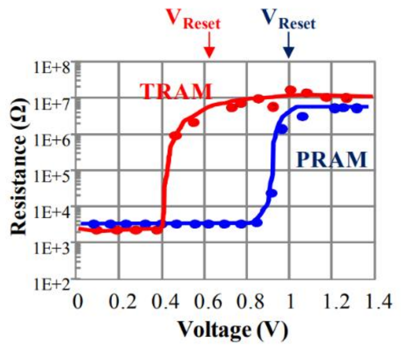


Fig. 6 Reset behaviors of TRAM and PRAM 1R TEGs. $V_{reset}$ represents reset voltage.

*Diagram from N.Takaura - Fabrication of topological switching RAM(TRAM)*
The diagrams show the experiment of TRAM performance vs PRAM. This is one of the advantages of TRAM has over PRAM with better fabrication.

**In conclusion,**
PRAM is not efficient, but it would need help from DRAM, MLC to help the performance and PRAM would need to change the fabrications from TRAM to even save cost and increase the efficiency even more.

**Question 2:**

2A)

So the algorithm searches an array of integers in parallel to find within the index It will keep looping and trying to find the lowest index number within the array. Within the array will be 7. [ 21],[ 87], [212], [109], [41], [21] Then find the lowest.

```
Function findMajorityElement(Argument array, Argument n)

For i > 0 and less than n

    Majority count ++

        For j>0 and less than n

        If (array[i] == array[j])

          Majority count ++

If (majority count > n/2)
```

Return array[i]

Pseudo code

**Question 2B)**

arr[] = {21, 87, 212, 109, 41, 21}

n=7

loop will iterate and

i = 0 n = 7

if majoritycount[21] == 87 not true

then count[21]++;

ittr 2

i = 1 n = 7   1<7 true

if count[87] == 212 not true

then count[87]++;

ittr 3

i = 2 n = 7   2<7 true

if count[212] == 1 not true

then count[212]++;

ittr 4

i = 3 n = 7 3<7 true

if count[109] == 41 not true

then count[109]++

ittr 5

i = 4 n = 7 4<7 true

if count[41] == 21 not true

then count[81]++;

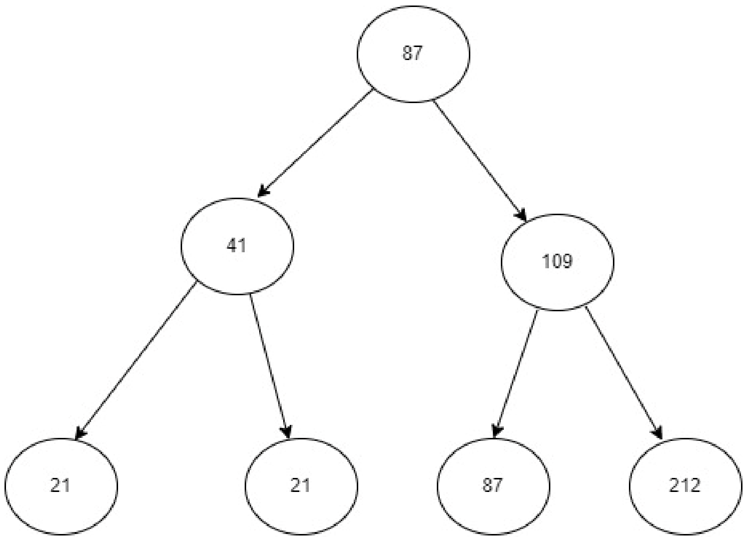
ittr 6

i = 5 n = 7 5<7 true

if count[21] == 21 not true

then majoritycount++;

return 21 as an output



***Here is my rushed binary tree, which I know is wrong.***

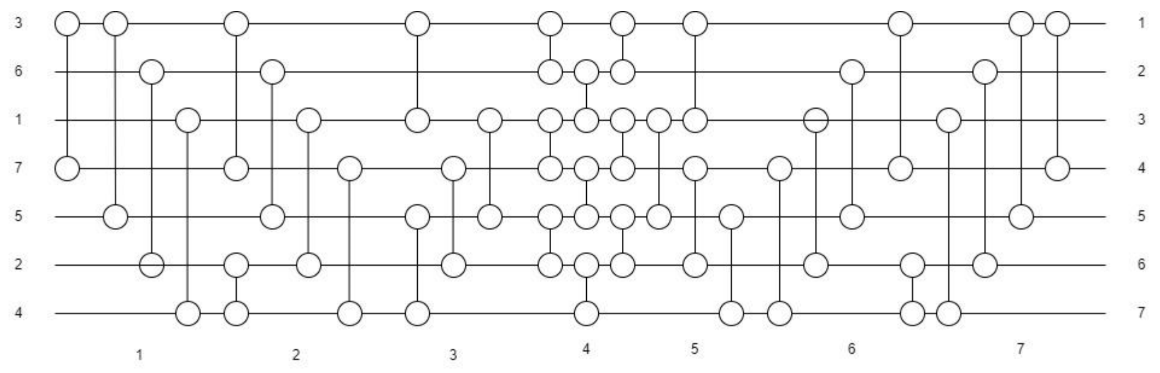
This is a rushed design for the diagram binary tree.

***Time Complexity***

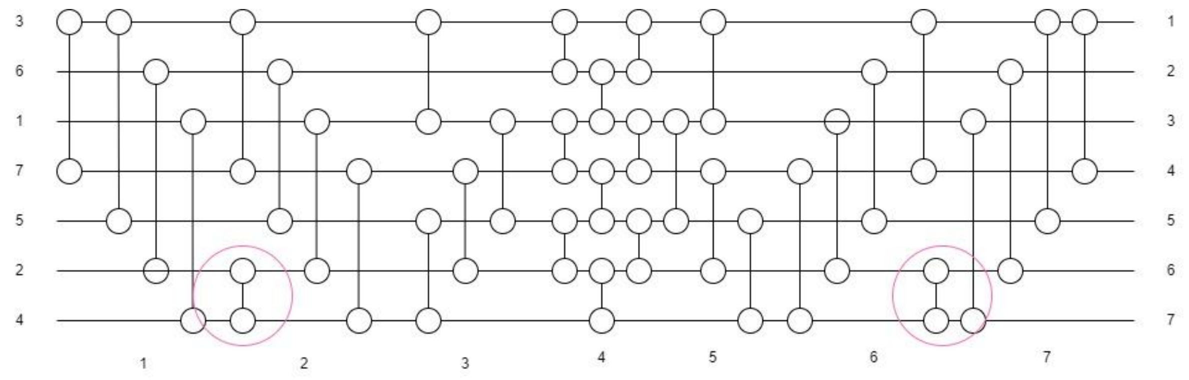
A fundamental methodology is to check whether or not each component is the component of the largest part. We can use two fixed circles where the outer circle emphasises the cluster and the inner circle counts the events of each component. Assuming we have found a component with more remarkable recurrence than n/2, this is our largest component.

So as the time is divided by 2 then the complexity of function will be O (logn).

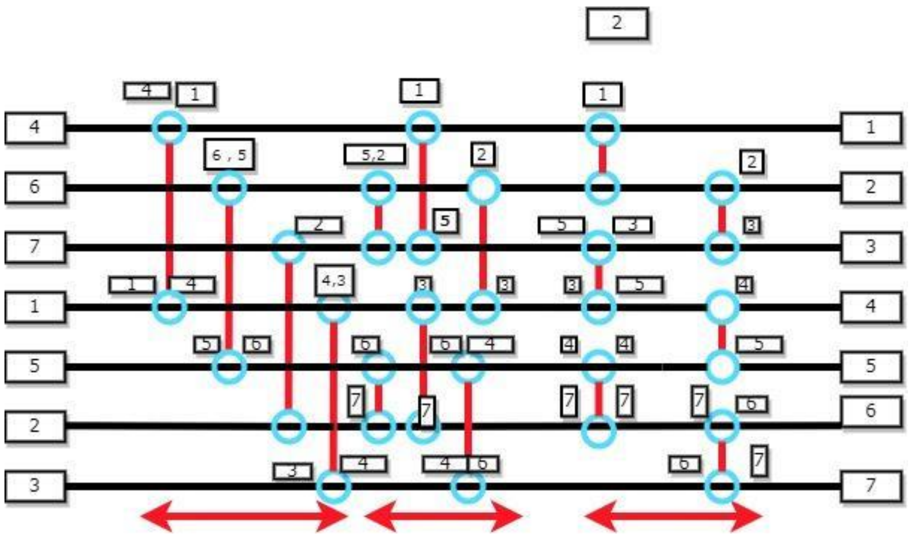
**Question 3a)**



Here is my new and improved layout. I was only able to do it in 7 parts but I tried to find a way to do it with 6 but I couldn't figure out a feasible way. So I went 7 but during the process of design the sorting network O(logn) for 7 wires.If this was into a large scale graph then this would also be feasible for any odd numbers of nodes In the millions for real world usage.



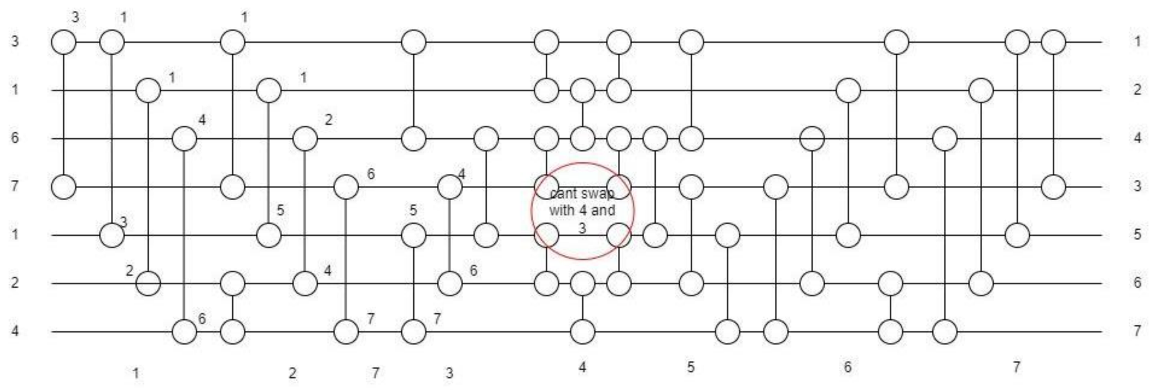
The cycle that is highlighted in pink. I would say excessive comparators but I think one is needed for a smooth sorting list.



Here is one of my old designs. Which is a good example of a non-efficient sorting network. This can only sort a few inputs and sort it out in the final output. But this is not a good design graph compared to the new one, which I listed above as It lacks the sorting of most of the numbers.

3(b)

Here is a short example of when the comparators on the 4th and 5th lines are not connecting which does affect the sorting. This is a small example but if there are more complex formats with included numbers on the graph and if there was the same number or similar number from 5th wire to 1st wire being number 1'.



So in conclusion, the sorting network won't work as intended for having the lowest number at the top and largest at the bottom. If it does not matter where the number goes and only format some of the numbers then it would be fine. For this structure format for the sorting list which will not function as asked from question 3a.

**References:**

Bardwin Gibbons, P., 2022. The Asynchronous Pram. [Online] Www1.icsi.berkeley.edu. Available at: <https://www1.icsi.berkeley.edu/pubs/techreports/tr-89-62.pdf> [Accessed 29 February 2022].

Homes.cs.washington.edu. 2022. [Online] Available at: <https://homes.cs.washington.edu/~arvind/cs424/readings/pram.pdf> [Accessed 2 March 2022].

Mitp-content-server.mit.edu. 2022. [online] Available at: <http://mitp-content-server.mit.edu:18180/books/content/sectbyfn?collid=books_pres_0&fn=Chapter%2027.pdf&id=8030> [Accessed 28 February 2022].

Graph version 2 [online] Available at:
<https://er.yuvayana.org/pram-model-of-computation-features-constraint-pram/>

Control unit graph

3nd PRAM [online] Available at:
<http://www.expertsmind.com/questions/parallel-random-access-machines-30138496.aspx>

4rd graph [online] Available at

<https://www.researchgate.net/figure/Mapping-from-the-MasPar-architecture-to-the-PRAM-model-PRAM-algorithm-on-the-MasPar_fig5_265435004>

2022. Brent’s Theorem PRAM. [Online] Available at: <https://er.yuvayana.org/brents-principle-state-and-proof-with-example/#:~:text=Theorem%3A%20%28Brent%E2%80%99s%20Theorem%29%3A%20A%20PRAM%20algorithm%20involving%20t,%7B%20%28m%20%E2%80%93%20t%29%20%2F%20p%7D%5D%20time%20steps.> [Accessed 2 March 2022].

Andrew Chin.,1993. Permutations on the Block PRAM.
<Permutations on the Block PRAM - ScienceDirect>, (1993), pp 69-73

PRACTICAL PRAMs [online] Available at <https://www.ida.liu.se/~chrke55/fork95/norfolk2x2.pdf>

Gibbons, P., 1989. The asynchronous PRAM. Berkeley, Calif.: International Computer Science Institute.

Gibbons, P., Matias, Y. and Ramachandran, V., 2018. The queue-read queue-write asynchronous PRAM model. Theoretical Computer Science, 196(1-2), pp.3-29.

Harris, T., 2014. A survey of PRAM simulation techniques. Edinburgh: University of Edinburgh. Department of Computer Science.

Karp, R. and Ramachandran, V., 2017. A survey of parallel algorithms for shared-memory machines. Berkeley, CA: Computer Soc. Div. (EECS).

Keller, J., Kessler, C. and Träff, J., 2001. Practical PRAM programming. New York: J. Wiley.

KhurramPasha, M., Feroze, M. and Ahmad Pasha, K., 2015. A Parallel Random Access Machine (PRAM) Model for English Language Recognizer (PRAM-ELR). International Journal of Computer Applications, 118(6), pp.12-18.

Sanz, J., 2018. Opportunities and constraints of parallel computing. New York: Springer-Verlag.

Y. N. Hwang et al., "MLC PRAM with SLC write-speed and robust read scheme," 2010 Symposium on VLSI Technology, 2010, pp. 201-202, doi: 10.1109/VLSIT.2010.5556227.

C. Yang, Y. Emre, Y. Cao and C. Chakrabarti, "Multi-Tiered Approach to Improving the Reliability of Multi-Level Cell PRAM," 2012 IEEE Workshop on Signal Processing Systems, 2012, pp. 114-119, doi: 10.1109/SiPS.2012.46.

DRAM -
N. Lu, I. -S. Choi, S. -H. Ko and S. -D. Kim, "An Effective Hierarchical PRAM-SLC-MLC Hybrid Solid State Disk," 2012 IEEE/ACIS 11th International Conference on Computer and Information Science, 2012, pp. 113-118, doi: 10.1109/ICIS.2012.30.
<IEEE Xplore Full-Text PDF: >

C. Yang, M. Mao, Y. Cao and C. Chakrabarti, "Cost-Effective Design Solutions for Enhancing PRAM Reliability and Performance," in IEEE Transactions on Multi-Scale

Computing Systems, vol. 3, no. 1, pp. 1-11, 1 Jan.-March 2017, doi: 10.1109/TMSCS.2016.2536026.
<Cost-Effective Design Solutions for Enhancing PRAM Reliability and Performance | IEEE Journals & Magazine | IEEE Xplore >

H. Park, S. Yoo and S. Lee, "Power management of hybrid DRAM/PRAM-based main memory," 2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC), 2011, pp. 59-64.

TRAM - *N.Takaura*
N. Takaura, T. Ohyanagi, M. Tai, T. Morikawa, M. Kinoshita and K. Akita, "Fabrication of topological-switching RAM (TRAM)," 2014 14th Annual Non-Volatile Memory Technology Symposium (NVMTS), 2014, pp. 1-4, doi: 10.1109/NVMTS.2014.7060835.