**A1gorithm Broadcast \_EREW**

**Processor P1**

**y (in P1'S private memory) ← x**

**L[1] ← y**

**for i = 0 to log p - 1 do**

**for all Pj where 2i + 1<= j <=2i+1 do in parallel**

**y (in Pj ‘ S private memory) ← L [ j – 2i]**

**L[j] ← y**

**endfor**

**endfor**

**Tracing**

**For processor P1:**

We follow the EREW model by reading exclusively information x from memory and write exclusively it in temporary buffer y. Then reading contents of buffer y and writing it into first location of the array L [1]

Using a for loop for i= 0 to log p -1 (= (log2 8) -1= (log2 23)-1= (3 log2 2) -1 = (3x1) - 1= 3 -1 = 2)

i.e. i=0, 1, 2 do

Take i=0

**for all Pj where 2i + 1<= j <=2i+1**

**2i + 1<= j <=2i+1**

**20 + 1<= j <=20+1**

**1+ 1<= j <=21**

**2<= j <=2**

**i.e. j= 2**

**y (in Pj ‘ S private memory) ← L [ j – 2i]**

**L[j] ← y**

**y (in P2 ‘ S private memory) ← L [ 2 – 20]**

**L[2] ← y**

**y (in P2 ‘ S private memory) ← L [ 2 – 1]**

**L[2] ← y**

**y (in P2 ‘ S private memory) ← L [ 1]**

**L[2] ← y**

**Now we exhausted j, so we can take the next i to be i=1**

**for all Pj where 2i + 1<= j <=2i+1**

**2i + 1<= j <=2i+1**

**21 + 1<= j <=21+1**

**2+ 1<= j <=22**

**3<= j <=4**

**i.e. j= 3,4**

**Take j=3 ( where i=1)**

**y (in Pj ‘ S private memory) ← L [ j – 2i]**

**L[j] ← y**

**y (in P2 ‘ S private memory) ← L [ 3 – 21]**

**L[3] ← y**

**y (in P2 ‘ S private memory) ← L [ 3 – 2]**

**L[3] ← y**

**y (in P2 ‘ S private memory) ← L [ 1]**

**L[3] ← y**

**Take j=4 ( where i=1)**

**y (in Pj ‘ S private memory) ← L [ j – 2i]**

**L[j] ← y**

**y (in P2 ‘ S private memory) ← L [ 4 – 21]**

**L[4] ← y**

**y (in P2 ‘ S private memory) ← L [ 4 – 2]**

**L[4] ← y**

**y (in P2 ‘ S private memory) ← L [ 2]**

**L[4] ← y**

**Now we exhausted j, so we can take the next i to be i=2**

**for all Pj where 2i + 1<= j <=2i+1**

**22+ 1<= j <=22+1**

**4+ 1<= j <=23**

**5<= j <=8**

**i.e. j= 5,6,7,8**

**Take j=5 ( where i=2)**

**y (in Pj ‘ S private memory) ← L [ j – 2i]**

**L[j] ← y**

**y (in P2 ‘ S private memory) ← L [ 5 – 22]**

**L[5] ← y**

**y (in P2 ‘ S private memory) ← L [ 5 – 4]**

**L[5] ← y**

**y (in P2 ‘ S private memory) ← L [ 1]**

**L[5] ← y**

**Take j=6 (where i=2)**

**y (in Pj ‘ S private memory) ← L [ j – 2i]**

**L[j] ← y**

**y (in P2 ‘ S private memory) ← L [ 6 – 22]**

**L[6] ← y**

**y (in P2 ‘ S private memory) ← L [ 6-4 ]**

**L[6] ← y**

**y (in P2 ‘ S private memory) ← L [ 2]**

**L[6] ← y**

**Take j=7 ( where i=2)**

**y (in Pj ‘ S private memory) ← L [ j – 2i]**

**L[j] ← y**

**y (in P2 ‘ S private memory) ← L [ 7 – 22]**

**L[7] ← y**

**y (in P2 ‘ S private memory) ← L [ 7 – 4]**

**L[7] ← y**

**y (in P2 ‘ S private memory) ← L [ 3]**

**L[7] ← y**

**Take j=8 (where i=2)**

**y (in Pj ‘ S private memory) ← L [ j – 2i]**

**L[j] ← y**

**y (in P2 ‘ S private memory) ← L [ 8 – 22]**

**L[6] ← y**

**y (in P2 ‘ S private memory) ← L [ 8-4 ]**

**L[8] ← y**

**y (in P2 ‘ S private memory) ← L [ 4]**

**L[8] ← y**

**Now we exhausted j**

**End for of j**

**Also, we exhausted i**

**End for of i**

**Algorithm Assessments**

**Time Complexity:**

Since the number of processors having read *x* doubles in each iteration, the procedure terminates in *O* (log p) time.

e.g. number of processors p = 8, so we need log2 8 steps = (log2 8) steps = (log2 23) steps= (3 log2 2) steps = (3x1) steps = 3 steps

This is done by the fist for loop: **for i = 0 to log p - 1**

**Space Complexity:**

The array *L* is the price paid in terms of memory, its size is 8 ( = p) locations. So we need Space Complexity in the order O *(p).*