Cilk and Cilk++
Lecture 4

- Quick sort
- Selection

Randomized Parallel QuickSort

Input: An array $A[\ q : r\ ]$ of distinct elements.

Output: Elements of $A[\ q : r\ ]$ sorted in increasing order of value.


```
Par-Randomized-QuickSort ( A[ q : r ] )
    n ← r − q + 1
    if n ≤ 30 then
        sort A[ q : r ] using any sorting algorithm
    else
        select a random element x from A[ q : r ]
        k ← Par-Partition ( A[ q : r ], x )
        spawn Par-Randomized-QuickSort ( A[ q : k − 1 ] )
        Par-Randomized-QuickSort ( A[ k + 1 : r ] )
        sync
```


spawn, sync $\Rightarrow$ cilk_spawn, cilk_sync

(pseudo-code)

Parallel Partition

Input: An array $A[\ q : r\ ]$ of distinct elements, and an element x from $A[\ q : r\ ]$.

Output: Rearrange the elements of $A[\ q : r\ ]$, and return an index $k \in [\ q, r\ ]$, such that all elements in $A[\ q : k - 1\ ]$ are smaller than x, all elements in $A[\ k + 1 : r\ ]$ are larger than x, and $A[\ k\ ] = x$.

```
Par-Partition ( A[ q : r ], x )
    n ← r − q + 1
    if n = 1 then return q
    array B[ 0: n − 1 ], lt[ 0: n − 1 ], gt[ 0: n − 1 ]
    parallel for i ← 0 to n − 1 do
        B[ i ] ← A[ q + i ]
        if B[ i ] < x then lt[ i ] ← 1 else lt[ i ] ← 0
        if B[ i ] > x then gt[ i ] ← 1 else gt[ i ] ← 0
    lt [ 0: n − 1 ] ← Par-Prefix-Sum ( lt[ 0: n − 1 ], + )
    gt[ 0: n − 1 ] ← Par-Prefix-Sum ( gt[ 0: n − 1 ], + )
    k ← q + lt [ n − 1 ],  A[ k ] ← x
    parallel for i ← 0 to n − 1 do
        if B[ i ] < x then A[ q + lt [ i ] − 1 ] ← B[ i ]
        else if B[ i ] > x then A[ k + gt [ i ] ] ← B[ i ]
    return k
```


parallel for $\Rightarrow$ cilk_for

Parallel Partition

A:

9	5	7	11	1	3	8	14	4	21

$x = 8$

Parallel Partition

A:

9	5	7	11	1	3	8	14	4	21

$x = 8$

B:

0	1	2	3	4	5	6	7	8	9
9	5	7	11	1	3	8	14	4	21

***lt*:**

0	1	2	3	4	5	6	7	8	9
0	1	1	0	1	1	0	0	1	0

***gt*:**

0	1	2	3	4	5	6	7	8	9
1	0	0	1	0	0	0	1	0	1

Parallel Partition

$x = 8$

***A*:**	9	5	7	11	1	3	8	14	4	21

	0	1	2	3	4	5	6	7	8	9
***B*:**	9	5	7	11	1	3	8	14	4	21

	0	1	2	3	4	5	6	7	8	9
***lt*:**	0	1	1	0	1	1	0	0	1	0
***lt*:**	0	1	2	2	3	4	4	4	5	5

prefix sum

	0	1	2	3	4	5	6	7	8	9
***gt*:**	1	0	0	1	0	0	0	1	0	1
***gt*:**	1	1	1	2	2	2	2	3	3	4

prefix sum

Parallel Partition

A: $x = 8$

9	5	7	11	1	3	8	14	4	21

B:

0	1	2	3	4	5	6	7	8	9
9	5	7	11	1	3	8	14	4	21

lt:

0	1	2	3	4	5	6	7	8	9
0	1	1	0	1	1	0	0	1	0

lt: *prefix sum*

0	1	2	2	3	4	4	4	5	5

gt:

0	1	2	3	4	5	6	7	8	9
1	0	0	1	0	0	0	1	0	1

gt: *prefix sum*

1	1	1	2	2	2	2	3	3	4

A:

0	1	2	3	4	5	6	7	8	9

Parallel Partition

A:	9	5	7	11	1	3	8	14	4	21

***x* = 8**

	0	1	2	3	4	5	6	7	8	9
B:	9	5	7	11	1	3	8	14	4	21

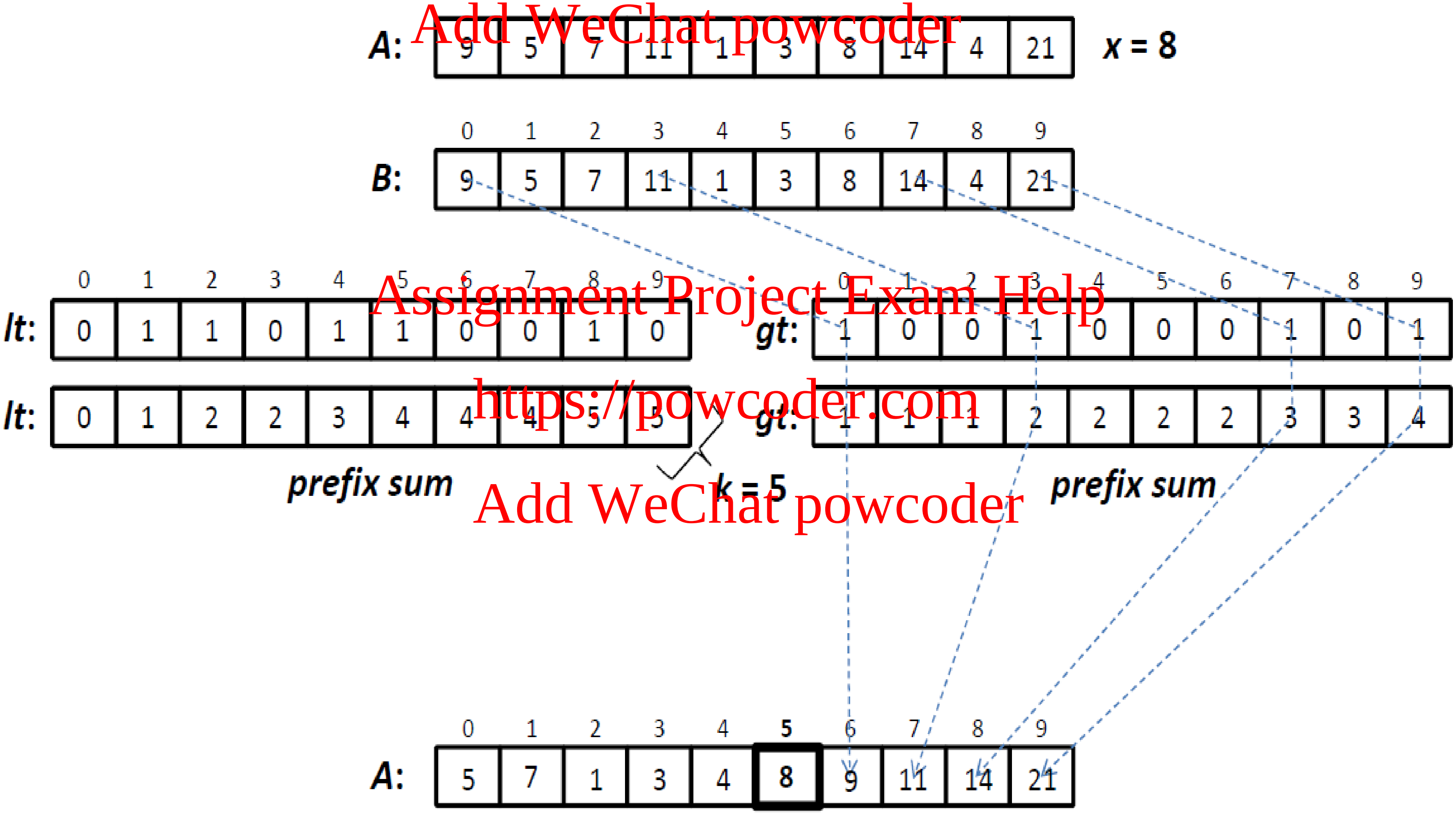
	0	1	2	3	4	5	6	7	8	9
***lt*:**	0	1	1	0	1	1	0	0	1	0
***lt*:**	0	1	2	2	3	4	4	4	5	5

	0	1	2	3	4	5	6	7	8	9
***gt*:**	1	0	0	1	0	0	0	1	0	1
***gt*:**	1	1	1	2	2	2	2	3	3	4

prefix sum ***k* = 5** ***prefix sum***

	0	1	2	3	4	**5**	6	7	8	9
A:	5	7	1	3	4	**8**				

Parallel Partition



Parallel Partition

x = 8

A:	9	5	7	11	1	3	8	14	4	21

	0	1	2	3	4	5	6	7	8	9
B:	9	5	7	11	1	3	8	14	4	21

	0	1	2	3	4	5	6	7	8	9
lt:	0	1	1	0	1	1	0	0	1	0
lt:	0	1	2	2	3	4	4	4	5	5

prefix sum

	0	1	2	3	4	5	6	7	8	9
gt:	1	0	0	1	0	0	0	1	0	1
gt:	1	1	1	2	2	2	2	3	3	4

prefix sum

k = 5

	0	1	2	3	4	**5**	6	7	8	9
A:	5	7	1	3	4	**8**	9	11	14	21

parquicksort.c


```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <cilk/cilk.h>

int median3index (int *A, int x, int y, int z) {
        if (A[x]<=A[y] && A[y]<=A[z]) return y;
        if (A[x]>=A[y] && A[y]>=A[z]) return y;
        if (A[y]<=A[x] && A[x]<=A[z]) return x;
        if (A[y]>=A[x] && A[x]>=A[z]) return x;
        return z;
}
```

parquicksort.c (continued)


```
void ParPrefixSum (int *X, int n) {
        if (n==1) return;
        int m=n/2,i;
        int *Y = malloc (m * sizeof(int));
        cilk_for (i=0; i<m; i++)
                Y[i] = X[2*i] + X[2*i+1];
        ParPrefixSum (Y, m);
        cilk_for (i=0; i<m; i++)
                X[2*i+1] = Y[i];
        cilk_for (i=1; i<(n+1)/2; i++)
                X[2*i] += Y[i-1];
        free(Y);
}
```

parquicksort.c (continued)


```
int ParPartition (int *A, int q, int r, int m) {
    int n = r-q+1, p = A[m], i;
    if (n==1) return q;
    int *B = malloc (n * sizeof(int));
    int *L = malloc (n * sizeof(int));
    int *G = malloc (n * sizeof(int));
    cilk_for (i=0; i<n; i++) {
        B[i] = A[q+i];
        L[i] = B[i]<p;
        G[i] = B[i]>=p && q+i!=m;
    }
```

parquicksort.c (continued)


```
    ParPrefixSum(L, n);
    ParPrefixSum(G, n);
    int k = q+L[n-1];
    A[k] = p;
    cilk_for (i=0; i<n; i++)
        if (B[i]<p) A[q+L[i]-1] = B[i];
        else if (q+i!=m) A[k+G[i]] = B[i];
    free(B);
    free(L);
    free(G);
    return k;
}
```

parquicksort.c (continued)


```
void ParQuickSort (int *A, int q, int r) {
        int n = r-q+1;
        if (n<=1) return;
        if (n==2) {
                if (A[q]>A[r]) { int t=A[q]; A[q]=A[r]; A[r]=t; }
                return;
        }
        int m = median3index (A, q, r, (q+r)/2);         // non-randomized
        int k = ParPartition (A, q, r, m);
        cilk_spawn ParQuickSort (A, q, k-1);
                   ParQuickSort (A, k+1, r);
        cilk_sync;
}
```

parquicksort.c (continued)


```
int main (int argc, char *argv[ ]) {
	int size=atoi (argv[1]);
	int *A = malloc (size * sizeof(int));
	time_t t;
	srand ((unsigned) time(&t));
	for (int k=0; k<size; k++) A[k]=rand( );
	clock_t start=clock( );
	ParQuickSort (A, 0, size-1);
	clock_t finish=clock( );
	double duration=(double)(finish-start)/CLOCKS_PER_SEC;
	printf("ParQuickSort took %lf seconds\n", duration);
	cilk_for (int k=1; k<size; k++)
		if (A[k-1]>A[k])
			printf ("Error in sort\n");
	return 0;
}
```

Compile and run


```
> cilk  parquicksort.c  –o  parquicksort
> ./parquicksort  100000
ParQuickSort took 1.705952 seconds

> ./parquicksort  1000000
ParQuickSort took 9.248920 seconds

> ./parquicksort  10000000
ParQuickSort took 76.821239 seconds
          (about 3 seconds in real time)

> ./quicksort  10000000
QuickSort took 12.692035 seconds
```

Parallel Selection

Input: A subarray $A[\,q : r\,]$ of an array $A[\,1 : n\,]$ of n distinct elements, and a positive integer $k \in [1, r - q + 1]$.

Output: An element x of $A[\,q : r\,]$ such that $rank(x, A[\,q : r\,]) = k$.

That is, x is the kth smallest element in $A(q : r)$.


```
Par-Selection ( A[ q : r ], n, k )
    n′ ← r − q + 1
    if n′ ≤ n / log n then
        sort A[ q : r ] using a parallel sorting algorithm and return A[ q + k - 1 ]
    else
        partition A[ q : r ] into blocks B_i's each containing log n consecutive elements
        parallel for i ← 1 to ⌈ n′ / log n ⌉ do
            M[ i ] ← median of B_i using a sequential selection algorithm
        find the median m of M[ 1 : ⌈ n′ / log n ⌉ ] using a parallel sorting algorithm
        t ← Par-Partition ( A[ q : r ], m )
        if k = t − q + 1 then return A[ t ]
        else if k < t − q + 1 then return Par-Selection ( A[ q : t − 1 ], n, k )
            else return Par-Selection ( A[ t + 1 : r ], n, k − t + q − 1 )
```

parallel for $\Rightarrow$ cilk_for

Pivot = median of medians

One iteration on a randomized set of 100 elements from 0 to 99

	12	15	11	2	9	5	0	7	3	21	44	40	1	18	20	32	19	35	37	39
	13	16	14	8	10	26	6	33	4	27	49	46	52	25	51	34	43	56	72	79
Medians	17	23	24	28	29	30	31	36	42	47	50	55	58	60	63	65	66	67	81	83
	22	45	38	53	61	41	62	82	54	48	59	57	71	78	64	80	70	76	85	87
	96	95	94	86	89	69	68	97	73	92	74	88	99	84	75	90	77	93	98	91

- 20 groups, with 5 elements in each group
- Groups are shown sorted by median, but this isn't necessary
- Red = pivot value = median of medians
- Gray = values < pivot
- White = values > pivot
- All values above/left of pivot must be gray
- All values below/right of pivot must be white

selection.c


```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <cilk/cilk.h>

void ParPrefixSum (int *X, int n) {
    ……                  // same as in parquicksort.c
}
```

selection.c (continued)


```
void ParPartition (int *A, int q, int r, int p, int *t, int *u) {
        int n = r-q+1, i;
        if (n==1) {*t=q; *u=q; return; }
        int *B = malloc (n * sizeof(int));
        int *L = malloc (n * sizeof(int));
        int *E = malloc (n * sizeof(int));
        int *G = malloc (n * sizeof(int));
        cilk_for (i=0; i<n; i++) {
                B[i] = A[q+i];
                L[i] = B[i]<p;
                E[i] = B[i]==p;
                G[i] = B[i]>p;
        }
```

selection.c (continued)


```
    ParPrefixSum(L, n);
    ParPrefixSum(E, n);
    ParPrefixSum(G, n);
    *t = q+L[n-1];
    *u = *t+E[n-1]-1;
    cilk_for (i=0; i<n; i++)
        if (B[i]<p) A[q+L[i]-1] = B[i];
        else if (B[i]==p) A[*t+E[i]-1] = B[i];
        else /* (B[i]>p) */ A[*u+G[i]] = B[i];
    free(B);
    free(L);
    free(E);
    free(G);
}
```

selection.c (continued)


```
int ParSelect (int *A, int q, int r, int k) {
        int n = r-q+1;
        if (n==1) return A[q];
        if (n==2) return ((k==1)==(q<=r)) ? A[q] : A[r];
        int s = round(sqrt(n));
        int groups = (n+s-1)/s, last = n%s;
        int *M = malloc (groups * sizeof(int));
        cilk_for (int i=0; i<groups; i++)
                if (i<n/s) M[i] = ParSelect(A, q+s*i, q+s*i+s-1, (s+1)/2);
                else M[i] = ParSelect(A, q+s*i, r, (last+1)/2);
        int m = ParSelect(M, 0, groups-1, (groups+1)/2);
        int t, u;
        ParPartition (A, q, r, m, &t, &u);
        if (k>=t-q+1 && k<=u-q+1) return A[t];
        else if (k<t-q+1) return ParSelect (A, q, t-1, k);
        else /* (k>u-q+1) */ return ParSelect (A, u+1, r, k-u+q-1);
}
```

selection.c (continued)


```
int main (int argc, char *argv[ ]) {
        int size=atoi (argv[1]);
        int *A = malloc (size * sizeof(int));
        time_t t;
        srand ((unsigned) time(&t));
        for (int k=0; k<size; k++)
                A[k]=rand( );
        clock_t start=clock( );
        int median = ParSelect (A, 0, size-1, (size+1)/2);
        clock_t finish=clock( );
        double duration=(double)(finish-start)/CLOCKS_PER_SEC;
        printf("Median = %d\n", median);
        printf("ParSelect took %lf seconds\n", duration);
        return 0;
}
```

Compile and run

> cilk selection.c –lm –o selection

> ./selection 100000

Median = 1072045003

ParSelect took 5.794388 seconds

> ./selection 1000000

Median = 1074793426

ParSelect took 32.682653 seconds

> ./selection 10000000

Median = 1073516545

ParSelect took 245.058890 seconds

(about 20 seconds in real time)