LABORATORIO 01

Francisco Castillo 21562

GITHUB

El repositorio puede ser visto acá: https://github.com/FranzCastillo/OpenMP-101/tree/lab-01

EJERCICIO 01

VERSIÓN SECUENCIAL

```
int n = atoi(argv[1]);

double factor = 1.0;
double sum = 0.0;
for (int k=0; k<n; k++) {
    sum += factor/(2*k+1);
    factor = -factor;
}

printf("Pi is approximately %.16f\n", 4.0*sum);
```

```
root@0d78a84e2b30:/src# ./piSeriesSeq 2000
Pi is approximately 3.1410926536210413
```

VERSIÓN PARALELA

```
double factor = 1.0;
double sum = 0.0;
#pragma omp parallel for num_threads(n_threads) reduction(+:sum)
for (int k=0; k<n; k++) {
    sum += factor/(2*k+1);
    factor = -factor;
}

printf("Pi is approximately %.16f\n", 4.0*sum);
```

```
root@0d78a84e2b30:/src# ./piSeriesNaive 2 1000
Pi is approximately 3.1385926573397755
root@0d78a84e2b30:/src# ./piSeriesNaive 2 10000
Pi is approximately 3.1412926535935410
root@0d78a84e2b30:/src# ./piSeriesNaive 4 150000
Pi is approximately 3.1414856161779099
root@0d78a84e2b30:/src# ./piSeriesNaive 10 1000000
Pi is approximately 3.1272061345819804
root@0d78a84e2b30:/src# ./piSeriesNaive 2 1000000
Pi is approximately 3.1411316286866620
```

A. ¿QUÉ SUCEDE CON EL VALOR PRECISO DE PI?

El valor se aproxima major mientras n sea mayor. Mientras más hils se agregaban, iba perdiendo precision decimal a pesar de tener más iteraciones.

B. IDENTIFIQUE EL TIPO DE DEPENDENCIA EN LA VARIABLE *FACTOR*

El tipo de dependencia es de datos. Esto es debido al valor de factor debe ser ”invertido” en base a la iteración previa, por lo que necesita el valor anterior.

C. ESCRIBA EN SUS PROPIAS PALABRAS LA RAZÓN POR LA CUAL *FACTOR = – FACTOR*

Esto se debe a la manera en la que la aproximación se realiza, particularmente en la parte de $(-1)^k$. El tener esta línea permite actualizar el valor de factor para la siguiente iteración.

D. ELIMINE LA DEPENDECIA Y DESCRIBA QUE SUCEDE CON EL RESULATDO

```
double factor = 0.0;
double sum = 0.0;
#pragma omp parallel for num_threads(n_threads) reduction(+:sum)
for (int k=0; k<n; k++) {
    factor = (k%2 == 0) ? 1.0 : -1.0;
    sum += factor/(2*k+1);
}

printf("Pi is approximately %.16f\n", 4.0*sum);
```

```
root@0d78a84e2b30:/src# ./piSeriesNaiveMod 2 10000000
Pi is approximately 3.1413153596733832
```

```
root@0d78a84e2b30:/src# ./piSeriesNaiveMod 2 100000000
Pi is approximately 3.1415936294052109
```

```
root@0d78a84e2b30:/src# ./piSeriesNaiveMod 2 1000000000
Pi is approximately 3.1217591881645390
```

```
root@0d78a84e2b30:/src# ./piSeriesNaiveMod 5 1000000000
Pi is approximately 3.1422343543895344
```

```
root@0d78a84e2b30:/src# ./piSeriesNaiveMod 10 100000000
Pi is approximately 3.1400830133157376
```

Es más preciso que su version anterior y más rápido. Se mantiene la relación descrita anteriormente (directamente proporcional la cantidad de iteraciones con la rpecisión del resultado real); aunque pareciera haber una proporción iteraciones/hilos que permite la major aproximación.

E. EJECUTE CON 1 THREAD Y DESCRIBA LA RAZÓN POR LA CUÁL EL RESULTADO ES DIFERENTE

```
root@0d78a84e2b30:/src# ./piSeriesNaiveMod 1 10000000
Pi is approximately 3.1415925535897915
root@0d78a84e2b30:/src# ./piSeriesNaiveMod 1 100000000
Pi is approximately 3.1415926435893260
root@0d78a84e2b30:/src# ./piSeriesNaiveMod 1 1000000000
Pi is approximately 3.1415926525880504
root@0d78a84e2b30:/src# ./piSeriesNaiveMod 1 10000000000
Pi is approximately 3.1415926530810587
root@0d78a84e2b30:/src# ./piSeriesNaiveMod 1 100000000000
Pi is approximately 3.1415926527987912
```

El problema proviene de la precisión de punto flotante, resulta que las operaciones de punto flotante no son asociativas y el orden de las operaciones puede afectar el resultado (causado por el orden en el que los hilos lo ejecutan).

F. CAMBIE EL *SCOPE* Y REGISTRE LOS RESULTADOS

```
    double sum = 0.0;
    double factor = 0.0;
#pragma omp parallel for num_threads(n_threads) reduction(+:sum) private(factor)
    for (int k = 0; k < n; k++) {
        factor = (k % 2 == 0) ? 1.0 : -1.0;
        sum += factor / (2 * k + 1);
    }
```

```
root@0d78a84e2b30:/src# ./piSeriesNaiveScope 2 10000000
Pi is approximately 3.1415925535897795
root@0d78a84e2b30:/src# ./piSeriesNaiveScope 2 100000000
Pi is approximately 3.1415926435902506
root@0d78a84e2b30:/src# ./piSeriesNaiveScope 2 1000000000
Pi is approximately 3.1415926525892579
root@0d78a84e2b30:/src# ./piSeriesNaiveScope 5 1000000000
Pi is approximately 3.1415926525894080
root@0d78a84e2b30:/src# ./piSeriesNaiveScope 10 1000000000
Pi is approximately 3.1415926525893263
```

G. CON LA ÚLTIMA VERSIÓN, REALICE LOS CALCULOS ESPECÍFICADOS

```
    double start_time = omp_get_wtime();  // Record start time

#pragma omp parallel for num_threads(n_threads) reduction(+:sum) private(factor)
    for (int k = 0; k < n; k++) {
        factor = (k % 2 == 0) ? 1.0 : -1.0;
        sum += factor / (2 * k + 1);
    }

    double end_time = omp_get_wtime();  // Record end time


    printf("Pi is approximately %.16f\n", 4.0 * sum);
    printf("Execution time: %f seconds\n", end_time - start_time);  // Print elapsed time
```

```
root@0d78a84e2b30:/src# nproc
12
```

```
root@8fd656b3ffea:/src# ./piTimed 12 10000000000
Pi is approximately 3.1415926530827529
Execution time: 0.671203 seconds
root@8fd656b3ffea:/src# ./piTimed 12 10000000000
Pi is approximately 3.1415926530827529
Execution time: 0.686830 seconds
root@8fd656b3ffea:/src# ./piTimed 24 10000000000
Pi is approximately 3.1415926530836242
Execution time: 0.675324 seconds
root@8fd656b3ffea:/src# ./piTimed 24 10000000000
Pi is approximately 3.1415926530836242
Execution time: 0.678361 seconds
root@8fd656b3ffea:/src# ./piTimed 24 10000000000
Pi is approximately 3.1415926530836242
Execution time: 0.656439 seconds
root@8fd656b3ffea:/src# ./piTimed 24 10000000000
Pi is approximately 3.1415926530836238
Execution time: 0.698085 seconds
root@8fd656b3ffea:/src# ./piTimed 24 10000000000
Pi is approximately 3.1415926530836242
Execution time: 0.655793 seconds
root@8fd656b3ffea:/src# ./piTimed 12 100000000000
Pi is approximately 3.1415926527999081
Execution time: 0.521918 seconds
root@8fd656b3ffea:/src# ./piTimed 12 100000000000
Pi is approximately 3.1415926527999081
Execution time: 0.569952 seconds
root@8fd656b3ffea:/src# ./piTimed 12 100000000000
Pi is approximately 3.1415926527999081
Execution time: 0.559710 seconds
root@8fd656b3ffea:/src# ./piTimed 12 100000000000
Pi is approximately 3.1415926527999081
Execution time: 0.575841 seconds
root@8fd656b3ffea:/src# ./piTimed 12 100000000000
Pi is approximately 3.1415926527999085
Execution time: 0.621209 seconds
```

RESULTADOS

	N	Threads	Tiempo de Ejecución	Speed Up	Eficiencia
Tiempo Secuencial	1.00E+10	1	3.261990	N/A	N/A
	1.00E+10	1	3.393095	N/A	N/A
	1.00E+10	1	3.379451	N/A	N/A
	1.00E+10	1	3.362280	N/A	N/A
	1.00E+10	1	3.483705	N/A	N/A
Tiempo Paralelo (Threads = Cores)	1.00E+10	12	0.618175	2.016480668	0.168040056
	1.00E+10	12	0.719318	2.440714309	0.203392859
	1.00E+10	12	0.671539	2.269433145	0.189119429
	1.00E+10	12	0.671203	2.256772423	0.188064369
	1.00E+10	12	0.68683	2.392713105	0.199392759

Tiempo Paralelo (Threads = 2*Cores)	1.00E+10	24	0.675324	2.202900135	0.183575011
	1.00E+10	24	0.678361	2.301743317	0.191811943
	1.00E+10	24	0.656439	2.218403435	0.184866953
	1.00E+10	24	0.698085	2.347157234	0.195596436
	1.00E+10	24	0.655793	2.284589353	0.190382446
Tiempo Paralelo (n=n*10 y threads = cores)	1.00E+11	12	0.521918	1.702491297	0.141874275
	1.00E+11	12	0.569952	1.933901281	0.16115844
	1.00E+11	12	0.55971	1.891512519	0.157626043
	1.00E+11	12	0.575841	1.936138677	0.16134489
	1.00E+11	12	0.621209	2.164108899	0.180342408

H. PRUEBE DIFERENTES POLÍTICAS DE PLANIFICACIÓN Y *BLOCK_SIZE* Y CALCULE LA DIFERENCIA DE SPEEDUP PARA CADA MECANISMOS DE SCHEDULING

```
    int n = 1000000000;  // 1e9
    int block_sizes[] = {16, 64, 128};
    const char* schedules[] = {"static", "dynamic", "guided", "auto"};
    omp_sched_t schedule_types[] = {omp_sched_static, omp_sched_dynamic, omp_sched_guided, omp_sched_auto};

    for (int i = 0; i < 4; i++) {  // Iterate over schedules
        printf("Schedule: %s\n", schedules[i]);
        for (int j = 0; j < 3; j++) {  // Iterate over block sizes
            printf("\tBlock size: %d\n", block_sizes[j]);

            for (int repeat = 0; repeat < 5; repeat++) {  // Repeat 5 times
                double sum = 0.0;
                double factor = 0.0;
                double start_time = omp_get_wtime();  // Record start time

                // Set the schedule and block size
                omp_set_schedule(schedule_types[i], block_sizes[j]);

#pragma omp parallel for num_threads(n_threads) reduction(+:sum) private(factor) schedule(runtime)
                for (int k = 0; k < n; k++) {
                    factor = (k % 2 == 0) ? 1.0 : -1.0;
                    sum += factor / (2 * k + 1);
                }

                double end_time = omp_get_wtime();  // Record end time
                printf("\t\t[%d] T=%.8f → Pi=%.16f\n", repeat, end_time - start_time, 4.0 * sum);
            }
        }
    }
```

Schedule	Block Size	N	Threads	Tiempo de Ejecución	Speed Up	Eficiencia
Static	16	1.00E+09	1	2.39270783	N/A	N/A
				2.37650663	N/A	N/A
				2.43762058	N/A	N/A
				2.53083513	N/A	N/A
				2.38727044	N/A	N/A
			12	0.61019285	3.921232	0.326769
				0.63258663	3.756808	0.313067
				0.62477877	3.901574	0.325131
				0.6559828	3.858082	0.321507
				0.60099332	3.972208	0.331017
	64		1	2.3383757	N/A	N/A
				2.35165304	N/A	N/A
				2.40867004	N/A	N/A
				2.43529282	N/A	N/A
				2.34160756	N/A	N/A
			12	0.49062012	4.766163	0.39718
				0.48152506	4.88376	0.40698
				0.48501066	4.966221	0.413852
				0.48672171	5.00346	0.416955
				0.48790604	4.7993	0.399942
	128		1	2.39226692	N/A	N/A
				2.46295052	N/A	N/A
				2.39892065	N/A	N/A
				2.38896108	N/A	N/A
				2.36791791	N/A	N/A
			12	0.46864968	5.104595	0.425383
				0.45965062	5.35831	0.446526
				0.45585321	5.262485	0.43854
				0.47324922	5.047998	0.420666
				0.4552731	5.201093	0.433424

Schedule	Block Size	N	Threads	Tiempo de Ejecución	Speed Up	Eficiencia
Dynamic	16	1.00E+09	1	3.23671474	N/A	N/A
				3.27626422	N/A	N/A
				3.23636665	N/A	N/A
				3.18709991	N/A	N/A
				3.2737242	N/A	N/A
			12	1.65799476	1.952186	0.162682
				1.64705555	1.989164	0.165764
				1.62220815	1.995038	0.166253
				1.63041279	1.954781	0.162898
				1.62951476	2.009018	0.167418
	64		1	2.58982567	N/A	N/A
				2.56115134	N/A	N/A
				2.62586771	N/A	N/A
				2.63800465	N/A	N/A
				2.55432166	N/A	N/A
			12	0.58494624	4.427459	0.368955
				0.58817683	4.35439	0.362866
				0.62244183	4.218656	0.351555
				0.6147323	4.291306	0.357609
				0.58058671	4.399552	0.366629
	128		1	2.56083489	N/A	N/A
				2.5840658	N/A	N/A
				2.53258045	N/A	N/A
				2.55413811	N/A	N/A
				2.59848777	N/A	N/A
			12	0.52998684	4.831884	0.402657
				0.49173942	5.254949	0.437912
				0.4978059	5.087486	0.423957
				0.51576582	4.952128	0.412677
				0.52778777	4.923357	0.41028

Schedule	Block Size	N	Threads	Tiempo de Ejecución	Speed Up	Eficiencia
Guided	16	1.00E+09	1	2.54369662	N/A	N/A
				2.55431838	N/A	N/A
				2.49290652	N/A	N/A
				2.3458325	N/A	N/A
				2.47002442	N/A	N/A
			12	0.47069141	5.40417	0.450348
				0.45722272	5.586595	0.46555
				0.43499097	5.730939	0.477578
				0.4292327	5.465177	0.455431
				0.4463692	5.533591	0.461133
	64		1	2.4968283	N/A	N/A
				2.60063765	N/A	N/A
				2.4637474	N/A	N/A
				2.55156291	N/A	N/A
				2.60982774	N/A	N/A
			12	0.51682355	4.831104	0.402592
				0.47990128	5.41911	0.451592
				0.46175675	5.335596	0.444633
				0.47073221	5.420413	0.451701
				0.46275288	5.639787	0.469982
	128		1	2.64890512	N/A	N/A
				2.54843129	N/A	N/A
				2.56657331	N/A	N/A
				2.54292942	N/A	N/A
				2.56168285	N/A	N/A
			12	0.43002311	6.159913	0.513326
				0.43698946	5.831791	0.485983
				0.43063137	5.960024	0.496669
				0.42889982	5.928959	0.49408
				0.45018133	5.690336	0.474195

Schedule	Block Size	N	Threads	Tiempo de Ejecución	Speed Up	Eficiencia
Auto	16	1.00E+09	1	2.54194225	N/A	N/A
				2.4291021	N/A	N/A
				2.43068756	N/A	N/A
				2.36674394	N/A	N/A
				2.47615967	N/A	N/A
			12	0.48899645	5.198284	0.43319
				0.50907243	4.771624	0.397635
				0.4698738	5.173065	0.431089
				0.5603919	4.223373	0.351948
				0.53617201	4.618219	0.384852
	64		1	2.454398	N/A	N/A
				2.37601627	N/A	N/A
				2.43182839	N/A	N/A
				2.44543303	N/A	N/A
				2.46431415	N/A	N/A
			12	0.54561628	4.498396	0.374866
				0.54675511	4.345668	0.362139
				0.68255511	3.562831	0.296903
				0.55829954	4.380145	0.365012
				0.59077508	4.171324	0.34761
	128		1	2.41046733	N/A	N/A
				2.37171913	N/A	N/A
				2.32215819	N/A	N/A
				2.37471723	N/A	N/A
				2.41609225	N/A	N/A
			12	0.55241708	4.363492	0.363624
				0.49041797	4.836118	0.40301
				0.44848546	5.177778	0.431482
				0.44490326	5.337604	0.4448
				0.44496226	5.429881	0.45249

¿CON QUÉ POLÍTICA DE PLANIFICACIÓN OBTUVO MEJORES RESULTADOS?

La que tuvo un mejor desempeño es la de Guided con un tamaño de bloque de 128. Este tuvo un speed up de casi 6 unidades

EJERCICIO 2

A. IMPLEMENTE EL PROGRAMA DESCRITO POR LA ECUACIÓN ANTERIOR. DESCRIBA QUE SUCEDE CON EL RESULTADO Y HAGA UNA COMPARACIÓN CON SU MEJOR VERSIÓN DEL INCISO H

```
    double sum_even = 0.0;
    double sum_odd = 0.0;
#pragma omp parallel for num_threads(n_threads) reduction(+:sum_even)
    for (int i = 0; i < n; i += 2) {
        sum_even += 1.0 / (2 * i + 1);
    }

#pragma omp parallel for num_threads(n_threads) reduction(+:sum_odd)
    for (int i = 1; i < n; i += 2) {
        sum_odd += 1.0 / (2 * i + 1);
    }
```

		N	Threads	Tiempo de Ejecución	Speed Up	Eficiencia
Guided, block size = 128	Tiempo Secuencial	1.00E+09	1	2.338835	N/A	N/A
		1.00E+09	1	2.314425	N/A	N/A
		1.00E+09	1	2.376991	N/A	N/A
		1.00E+09	1	2.321799	N/A	N/A
		1.00E+09	1	2.321786	N/A	N/A
	Tiempo Paralelo	1.00E+09	12	0.409918	5.705616733	0.475468061
		1.00E+09	12	0.431604	5.362380794	0.446865066
		1.00E+09	12	0.410193	5.794811223	0.482900935
		1.00E+09	12	0.412598	5.627266734	0.468938895
		1.00E+09	12	0.404583	5.738713688	0.478226141
Diferentes Sumatorias	Tiempo Secuencial	1.00E+09	1	2.316840	N/A	N/A
		1.00E+09	1	2.393725	N/A	N/A
		1.00E+09	1	2.364465	N/A	N/A
		1.00E+09	1	2.298619	N/A	N/A
		1.00E+09	1	2.322087	N/A	N/A
	Tiempo Paralelo	1.00E+09	12	0.36197432	6.400565626	0.533380469
		1.00E+09	12	0.35433842	6.755477123	0.562956427
		1.00E+09	12	0.36546941	6.469665628	0.539138802
		1.00E+09	12	0.36390997	6.316448269	0.526370689
		1.00E+09	12	0.3911345	5.936799771	0.494733314

Es evidente la superioridad de Speed Up, viendo que es mejor por una unidad la versión que separa las sumatorias.

B. COMPILE LA MEJOR VERSIÓN, PERO CON LA OPTIMIZACIÓN -O2. ¿QUÉ PUDIERON OBSERVAR?

```
root@a75a78c20891:/src# gcc -fopenmp -O2 piSeriesAlt.c -o piSeriesAltO2
```

		N	Threads	Tiempo de Ejecución	Speed Up	Eficiencia
Diferentes Sumatorias	Tiempo Secuencial	1.00E+09	1	2.316840	N/A	N/A
		1.00E+09	1	2.393725	N/A	N/A
		1.00E+09	1	2.364465	N/A	N/A
		1.00E+09	1	2.298619	N/A	N/A
		1.00E+09	1	2.322087	N/A	N/A
	Tiempo Paralelo	1.00E+09	12	0.36197432	6.400565626	0.533380469
		1.00E+09	12	0.35433842	6.755477123	0.562956427
		1.00E+09	12	0.36546941	6.469665628	0.539138802
		1.00E+09	12	0.36390997	6.316448269	0.526370689
		1.00E+09	12	0.3911345	5.936799771	0.494733314
-O2	Tiempo Secuencial	1.00E+09	1	1.006590	N/A	N/A
		1.00E+09	1	1.029893	N/A	N/A
		1.00E+09	1	1.014427	N/A	N/A
		1.00E+09	1	1.033424	N/A	N/A
		1.00E+09	1	1.023649	N/A	N/A
	Tiempo Paralelo	1.00E+09	12	0.20258642	4.968696767	0.414058064
		1.00E+09	12	0.20900092	4.927695199	0.410641267
		1.00E+09	12	0.21717305	4.671052923	0.38925441
		1.00E+09	12	0.20958193	4.930883259	0.410906938
		1.00E+09	12	0.21786336	4.698581074	0.391548423

Podemos ver que el Speed Up y la Eficiencia es menor a comparación que sin la bandera de optimización, es evidente la reducción de tiempo de ejecución tanto en modo secuencial como en paralelo. En nuestro caso, esta optimización nos es muy útil, pues reordena el código, optimiza ciclos, propaga valores constantes, entre otros; todas estas acciones son de beneficio para el código ejecutado.