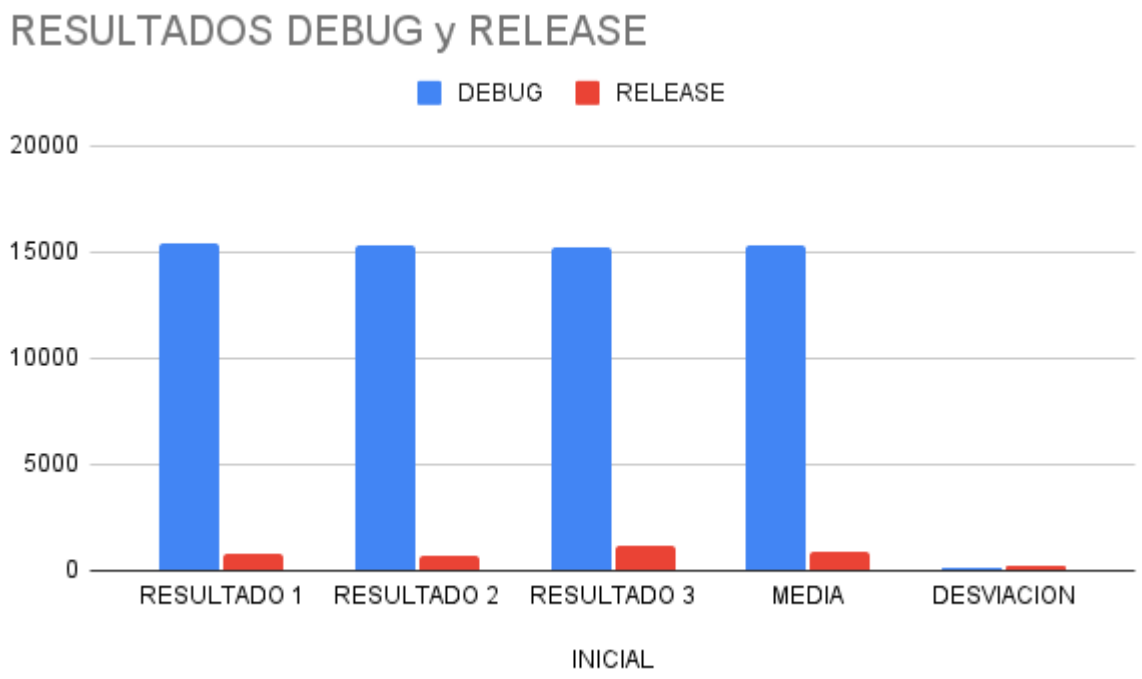
Lab 6: Concurrencia y Modelo de Memoria

El objetivo de este laboratorio consiste en eliminar, mediante distintos métodos, carreras de datos así como ilustrar el concepto de concurrencia.
A la hora de realizar esta práctica, todas las medidas están en microsegundos y se ha ejecutado cada prueba 3 veces distintas para ver la desviación típica de los resultados.

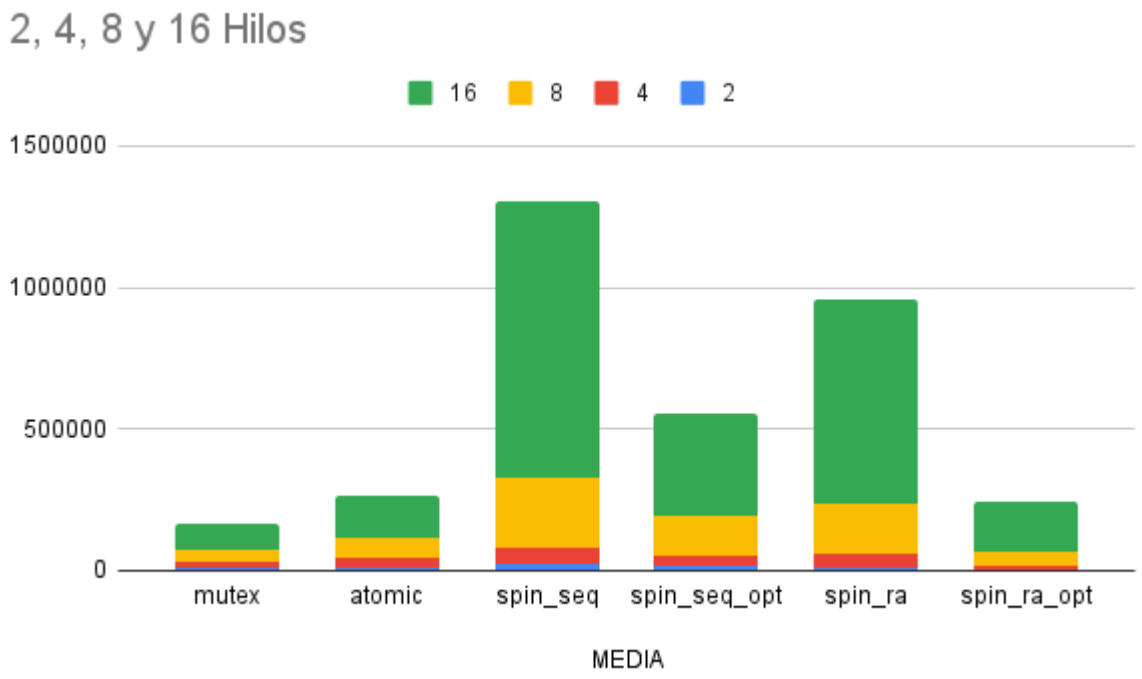
Primero se ha ejecutado el código inicial dado, y se han extraído los resultados de las ejecuciones de las versiones de Release y Debug. Se ha podido observar que los resultados del counter no coinciden con los esperados ya que no hay 100.000 actualizaciones por cada hilo. Esto es debido a que se produce una carrera de datos y no hay ningún mecanismo que regule este problema. A continuación se verán estos resultados con sus correspondientes medias y desviaciones típicas y se mostrará en un gráfico para tener una mejor visión de los mismos.

INICIAL	RESULTADO 1	RESULTADO 2	RESULTADO 3	MEDIA	DESVIACIÓN
DEBUG	15396	15273	15205	15291,33333	96,81081207
RELEASE	796	737	1147	893,3333333	221,6536337



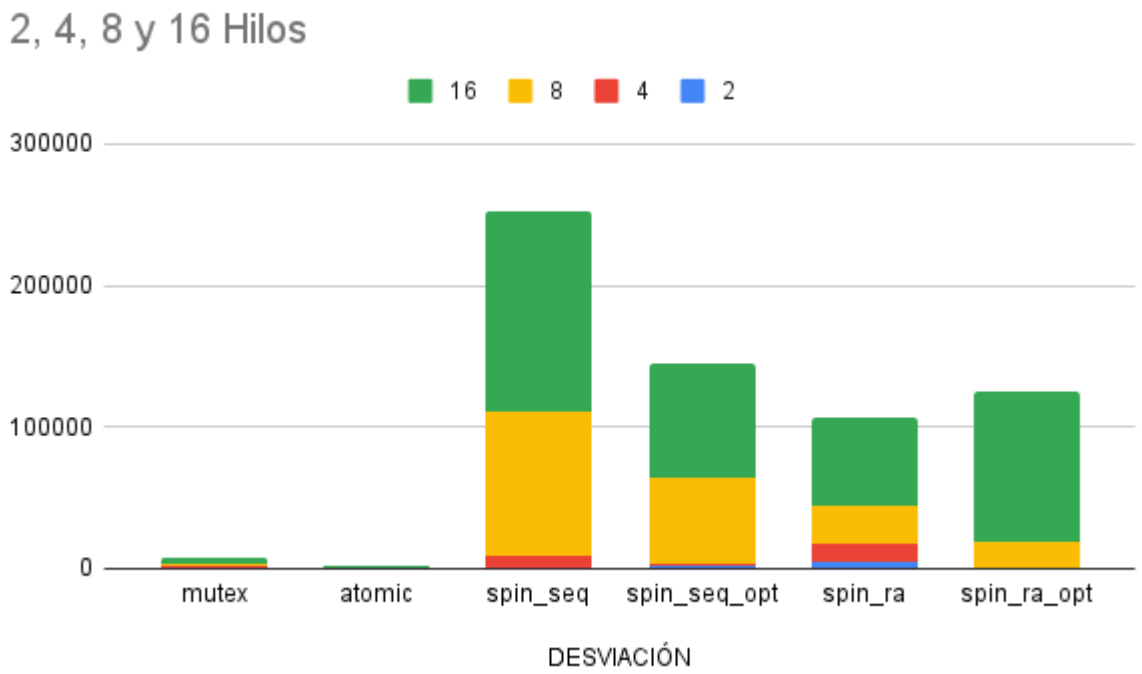
Tras ver los resultados principales, se han hecho pruebas con distintos métodos que eliminan las carreras de datos. Estos métodos son: el uso de mutex (cerrojos), protección con atómicos, protección con spinlock y consistencia secuencial, protección con spinlock y consistencia de liberación/adquisición y finalmente con una optimización de estos dos últimos métodos. A continuación mostraremos dos tablas con los resultados de los tiempos de ejecución. Una tabla contendrá las medias de los resultados de las tres ejecuciones por número de hilos y método y la otra las respectivas desviaciones típicas de estos resultados. A su vez cada tabla dispondrá de un gráfico aparte para tener una mejor visualización.

MEDIA	2	4	8	16
mutex	9380,333333	24564,33333	43048	88721,66667
atomic	11545,66667	35394	72824,66667	147680,6667
spin_seq	21411,66667	62035	241492	975696,3333
spin_seq_opt	14777,33333	39499	143189,3333	355031
spin_ra	12139,66667	47032	180223	720550,6667
spin_ra_opt	5140,333333	14756,33333	48723,66667	171798,6667



DESVIACIÓN	2	4	8	16
mutex	674,3517875	1676,892761	1036,812905	3943,906735

atomic	252,3971738	349,6097825	577,3078324	170,6614583
spin_seq	1265,754452	8531,303124	100446,5232	142108,4252
spin_seq_opt	1745,231312	2338,869171	59684,80404	80483,16912
spin_ra	5227,565431	12529,42964	26588,62907	61827,92121
spin_ra_opt	194,1914863	958,5250823	17768,93025	105912,7409



CONCLUSIONES

Con los resultados obtenidos podemos hacer varias afirmaciones.

Comparando los resultados de las primeras ejecuciones con las que evitan las carreras de datos podemos ver como el tiempo de ejecución es superior en todos los casos. Esto tiene sentido ya que, como va a haber que esperar a las lecturas y escrituras de otros hilos, el tiempo general va a aumentar considerablemente.

Si comparamos las ejecuciones de un mismo método con los distintos hilos podemos ver como en todos los casos tardará más en compilar cuantos más hilos haya. Esto se debe a que al final, por cada hilo que se declare el bucle realizará 100.000 iteraciones más y, al estar accediendo constantemente al mismo dato, lo único que se consigue al paralelizar este bucle es que el programa tarde más en ejecutarse al tener que realizar más operaciones.

Comparando los distintos métodos podemos ver como el más costoso es el de protección con spinlock y consistencia secuencial seguido de protección con

spinlock y consistencia de liberación/adquisición. A su vez podemos ver como las versiones optimizadas de ambas reducen considerablemente el tiempo de ejecución. Por último, los que menos tardan son las ejecuciones que usan cerrojos y atómicos.

En cuanto a las desviaciones típicas, lo que podemos sacar en claro sobre todo es que las ejecuciones en las que el tiempo varía más son las que se usan 8 ó 16 hilos menos en los casos del uso de cerrojos y atómicos. En estos últimos casos los resultados son muy similares entre ellos.