SPLASH-4: UNA BENCHMARK SUITE MODERNA CON CONSTRUCTOS LOCK-FREE

Eduardo José Gómez-Hernández[1] Juan M. Cebrian[1]
Stefanos Kaxiras[2] Alberto Ros[1]

[1]Computer Engineering Department
University of Murcia, Spain

[2]Department of Information Technology
Uppsala University, Sweden

VISTA PREVIA...

- Los benchmark suites son la piedra angular para medir el rendimiento de un sistema
- Los benchmark suites pueden distorsionar el rendimiento
- Mantener las mismas cargas de trabajo, pero siguiendo el ritmo de los cambios micro-arquitecturales es un gran reto

VISTA PREVIA...

- Los benchmark suites son la piedra angular para medir el rendimiento de un sistema
- Los benchmark suites pueden distorsionar el rendimiento
- Mantener las mismas cargas de trabajo, pero siguiendo el ritmo de los cambios micro-arquitecturales es un gran reto
- Presentamos Splash-4, una versión revisada de Splash-3 (la cual es una actualización sobre Splash-2), que introduce técnicas de programación modernas para mejorar la escalabilidad en el hardware actual

MOTIVACIÓN

[1]Woo, Steven Cameron, et al, "The SPLASH-2 programs: Characterization and methodological considerations." ACM SIGARCH computer architecture news 23, 1995

[2]Venetis, Ioannis E., et al, "The Modified SPLASH-2", https://www.capsl.udel.edu//splash/ 2007

[3]Sakalis, Christos, et al, "Splash-3: A Properly Synchronized Benchmark Suite for Contemporary Research", ISPASS 2016

[4]Gómez-Hernández, Eduardo José et al, "Splash-4: Improving Scalability with Lock-Free Constructs", ISPASS 2021

MOTIVACIÓN

La primera gran benchmark suite paralela. Sigue siendo usada (+5k citas)[1]

[1] Woo, Steven Cameron, et al, "The SPLASH-2 programs: Characterization and methodological considerations." ACM SIGARCH computer architecture news 23, 1995

[2] Venetis, Ioannis E., et al, "The Modified SPLASH-2", https://www.capsl.udel.edu//splash/ 2007

[3] Sakalis, Christos, et al, "Splash-3: A Properly Synchronized Benchmark Suite for Contemporary Research", ISPASS 2016

[4] Gómez-Hernández, Eduardo José et al, "Splash-4: Improving Scalability with Lock-Free Constructs", ISPASS 2021

MOTIVACIÓN

La primera gran benchmark suite paralela. Sigue siendo usada (+5k citas)[1]

Una pequeña actualización que corrige errores y cambia el estilo de programación[2]

[1] Woo, Steven Cameron, et al, "The SPLASH-2 programs: Characterization and methodological considerations." ACM SIGARCH computer architecture news 23, 1995

[2] Venetis, Ioannis E., et al, "The Modified SPLASH-2", https://www.capsl.udel.edu//splash/ 2007

[3] Sakalis, Christos, et al, "Splash-3: A Properly Synchronized Benchmark Suite for Contemporary Research", ISPASS 2016

[4] Gómez-Hernández, Eduardo José et al, "Splash-4: Improving Scalability with Lock-Free Constructs", ISPASS 2021

MOTIVACIÓN

La primera gran benchmark suite paralela. Sigue siendo usada (+5k citas)[1]

Una pequeña actualización que corrige errores y cambia el estilo de programación[2]

La primera gran actualización que elimina las condiciones de carrera y varios errores de rendimiento[3]

[1] Woo, Steven Cameron, et al, "The SPLASH-2 programs: Characterization and methodological considerations." ACM SIGARCH computer architecture news 23, 1995

[2] Venetis, Ioannis E., et al, "The Modified SPLASH-2", https://www.capsl.udel.edu//splash/ 2007

[3] Sakalis, Christos, et al, "Splash-3: A Properly Synchronized Benchmark Suite for Contemporary Research", ISPASS 2016

[4] Gómez-Hernández, Eduardo José et al, "Splash-4: Improving Scalability with Lock-Free Constructs", ISPASS 2021

MOTIVACIÓN

La primera gran benchmark suite paralela. Sigue siendo usada (+5k citas)[1]

Una pequeña actualización que corrige errores y cambia el estilo de programación[2]

La primera gran actualización que elimina las condiciones de carrera y varios errores de rendimiento[3]

La versión actual, con uso intensivo de lockfree y operaciones atómicas[4]

[1] Woo, Steven Cameron, et al, "The SPLASH-2 programs: Characterization and methodological considerations." ACM SIGARCH computer architecture news 23, 1995

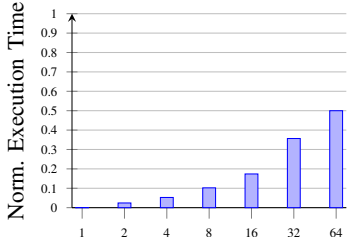
[2] Venetis, Ioannis E., et al, "The Modified SPLASH-2", https://www.capsl.udel.edu//splash/ 2007

[3] Sakalis, Christos, et al, "Splash-3: A Properly Synchronized Benchmark Suite for Contemporary Research", ISPASS 2016

[4] Gómez-Hernández, Eduardo José et al, "Splash-4: Improving Scalability with Lock-Free Constructs", ISPASS 2021

MOTIVATION

- Splash-2 y Splash-3 fueron creados usando técnicas de programación obsoletas
- En trabajos anteriores, se muestra que los tamaños por defecto de las entradas, limitan la escalabilidad de algunas aplicaciones.
  - La cantidad de computo a realizar entre los puntos de sincronización no es substancial respecto al tiempo requerido para sincronizar
  - Usar entradas más grandes incrementa el tiempo de ejecución, y esto es un problema cuando se usa con simuladores



VOLREND (ADAPTIVE.C.IN:182/199)

Splash-3

```
1 /* Lock */
2 ALOCK(Global->QLock,local_node);
3 work = Global->Queue[local_node
      ][0];
4 Global->Queue[local_node][0] +=
      1;
5 AULOCK(Global->QLock,local_node);
```

Splash-4

```
1 /* Lock-free */
2 work = FETCH_ADD(Global->Queue[
      local_node][0], 1);
```

27 secciones críticas (18%) han sido convertidos a atómicos de C11

ALOCK/AULOCK → mutex lock/unlock

FETCH_ADD → atomic_fetch_and_add - Sequential Consistency

OCEAN (MULTI.C.IN:90)
ATOMIC MAX

Splash-3

```
1 /* Lock */
2 LOCK(locks->error_lock)
3 if (local_err > multi->err_multi)
      {
4   multi->err_multi = local_err;
5 }
6 UNLOCK(locks->error_lock)
```

Splash-4

```
1 /* Lock-free */
2 double expected = LOAD(multi->
      err_multi);
3 do {
4   if (local_err <= expected)
      break;
5 } while (!CAExch(multi->err_multi
      , expected, local_err));
```

18 secciones críticas (12%)han sido convertidas a constructos-CAS

LOCK/UNLOCK → mutex lock/unlock
LOAD → Atomic Load - Sequential Consistency

WATER (POTENG.C.IN 159/253)

FETCH_AND_ADD_DOUBLE

```
1 double oldValue = LOAD(ptr); double newValue;
2 do {
3  newValue = oldValue + addition;
4 } while (!CAExch(ptr, oldValue, newValue));
```

LOCK/UNLOCK → mutex lock/unlock
LOAD → Atomic Load - Sequential Consistency

WATER (POTENG.C.IN 159/253)

FETCH_AND_ADD_DOUBLE

```
1 double oldValue = LOAD(ptr); double newValue;
2 do {
3  newValue = oldValue + addition;
4 } while (!CAExch(ptr, oldValue, newValue));
```

Splash-3

```
1 /* Lock */
2 LOCK(gl->PotengSumLock);
3 *POTA = *POTA + LPOTA;
4 *POTR = *POTR + LPOTR;
5 *PTRF = *PTRF + LPTRF;
6 UNLOCK(gl->PotengSumLock);
```

Splash-4

```
1 /* Lock-free */
2 FETCH_AND_ADD_DOUBLE(POTA,LPOTA);
3 FETCH_AND_ADD_DOUBLE(POTR,LPOTR);
4 FETCH_AND_ADD_DOUBLE(PTRF,LPTRF);
```

5 secciones críticas adicionales (3%) usando 15 constructos-CAS

LOCK/UNLOCK → mutex lock/unlock
LOAD → Atomic Load - Sequential Consistency

SPLASH-4: BARRERA CENTRALIZADA DE INVERSIÓN DE SENTIDO

- La ejecución entre barrera es pequeño
- Para minimizar el *overhead*, imeplementamos la barrera de inversión de sentido
- Este construct está optimizado para esperas cortas, pero sin más hilos que cores disponibles

---

LOAD → Atomic Load - Sequential Consistency
STORE → Atomic Store - Sequential Consistency

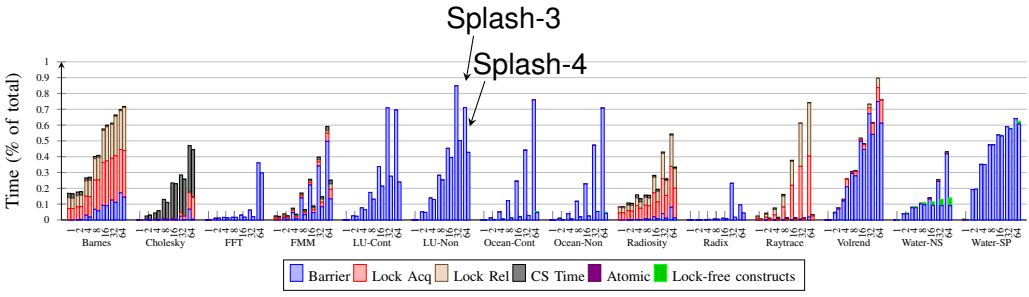
SPLASH-4: BARRERA CENTRALIZADA DE INVERSIÓN DE SENTIDO

- La ejecución entre barrera es pequeño
- Para minimizar el *overhead*, imeplementamos la barrera de inversión de sentido
- Este construct está optimizado para esperas cortas, pero sin más hilos que cores disponibles

```
1 local_sense = !local_sense;
2 if (atomic_fetch_sub(&(count), 1)
      == 1) {
3     count = threads;
4   STORE(sense, local_sense);
5 } else {
6   do {} while (LOAD(sense) !=
      local_sense);
7 }
```

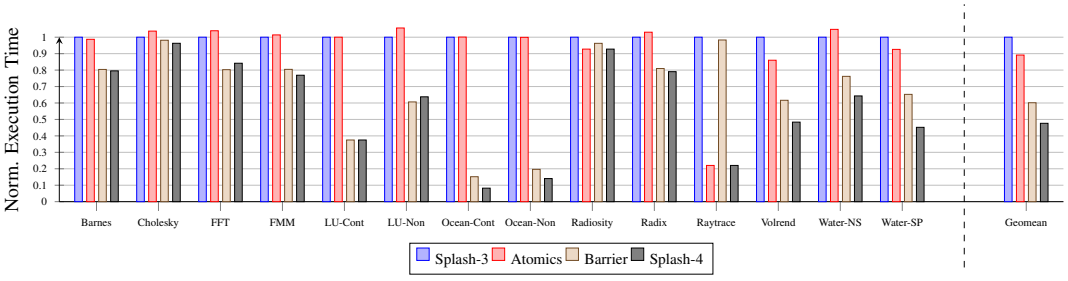
LOAD → Atomic Load - Sequential Consistency
STORE → Atomic Store - Sequential Consistency

EVALUACIÓN: DESGLOSE DEL SOBRECOSTE EN EL SIMULADOR



La mayoría del tiempo de ejecución está dominado por la sincronización en las barreras

Evaluación: Resumen en máquina real (64 threads)



- Con constructos lock-free, el tiempo de ejecución se reduce un 11%
- Con las nuevas barreras, el tiempo de ejecución se reduce un 40%
- La mejora combinada es del 52%

Conclusión

- Hemos presentado The Splash-4 Benchmark Suite
- Hemos realizado un análisos detallado en comparación con Splash-3

Conclusión

- Hemos presentado The Splash-4 Benchmark Suite
- Hemos realizado un análisos detallado en comparación con Splash-3
- Nuestro estudio muestra:
  - Una mejora significativa de escalabilidad
  - Una reducción significativa del tiempo de ejecución en máquina real y en el simulador

Conclusión

- Hemos presentado The Splash-4 Benchmark Suite
- Hemos realizado un análisos detallado en comparación con Splash-3
- Nuestro estudio muestra:
  - Una mejora significativa de escalabilidad
  - Una reducción significativa del tiempo de ejecución en máquina real y en el simulador
- En resumen:
  - The Splash benchmark suite sigue siendo la piedra angular en la investigación de arquitectura de computadores
  - Tiene que ser actualizada para aprovechar las características del hardware actual

SPLASH-4: UNA BENCHMARK SUITE MODERNA CON CONSTRUCTOS LOCK-FREE

Eduardo José Gómez-Hernández[1] Juan M. Cebrian[1]
Stefanos Kaxiras[2] Alberto Ros[1]

eduardojose.gomez@um.es

Thank you for your attention!

ECHO, ERC Consolidator Grant (No 819134)
Spanish Ministerio de Economía, Industria y Competitividad – Agencia Estatal de Investigación (ERC2018-092826)

VARIABLES INDEPENDIENTES

- Durante la posible ventana de ejecución de la sección critica, ninguna variable puede estar relacionada de ninguna manera, incluso sin dar información una de otra

```
1 LOCK(lock);
2  ptr = NULL;
3 some_random_boolean = false;
4 UNLOCK(lock);
```

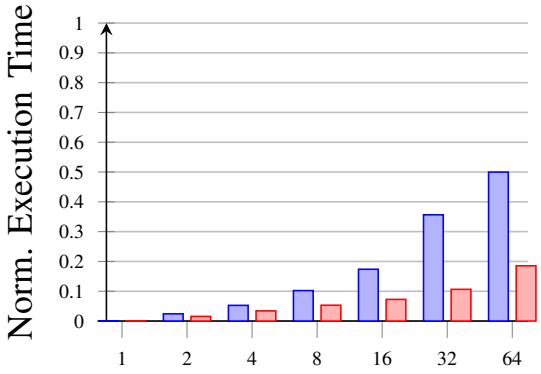
VARIABLES INDEPENDIENTES

- Durante la posible ventana de ejecución de la sección critica, ninguna variable puede estar relacionada de ninguna manera, incluso sin dar información una de otra

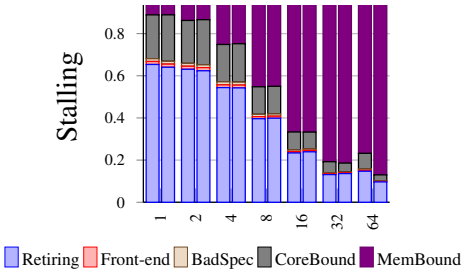
```
1 LOCK(lock);
2  ptr = NULL;
3 some_random_boolean = false;
4 UNLOCK(lock);
```

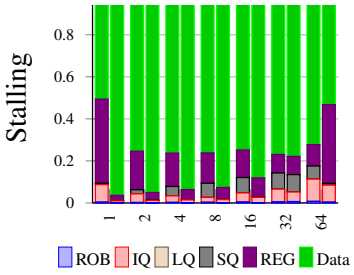
```
1 LOCK(lock);
2  if (some_random_boolean) {
3      local = *ptr;
4  }
5 UNLOCK(lock);
```

Overhead Summary



EVALUACIÓN: TOP-DOWN (OCEAN-CONT)





SPLASH-3

- Splash-3 expone las condiciones de carrera y problemas de rendimiento, y los corrige
- En el análisis hecho con GEMS (en un processador con 64-cores en-orden), las aplicaciones llegaban a un speedup entre 16× a 47×
- En nuestro entorno de simulación (gem5-20 Intel's IceLake-like 64 cores fuera de orden): la escalabilidad termina entre 16 y 32 cores, con un speedup medio de 2.3×
- En nuestra máquina física (64-Core AMD EPYC 7702P), la escalabilidad acaba entre 4 y 16 cores, con un speedup medio de 4.7×

SPLASH-4

- La sincronización está implementada como una combinación de locks software, variables condicionales, y barreras
- El objetivo principal es reemplazar las operaciones de sincronización de alto coste, con alternativas más livianas
- Esto se traduce un incremento del rendimiento que además utiliza caracterisitcas del procesador que antes no estaban siendo medidas

SPLASH-4: LOCKFREE Y ATÓMICOS

- Las ISAs modernas proporcionan un conjunto básico de operaciones atómicas que proporcionan atomicidad y sincronización
- Este conjunto consiste en lecturas y escrituras atómicas, operaciones read-modify-write (RMW) y operaciones de comparación e intercambio
- Comúnmente, estas operaciones solo están disponibles para tipos de datos enteros (integer)

SPLASH-4: LOCKFREE Y ATÓMICOS
CONSTRUCTO WHILE&CAEXCH

- La operación de comparación-e-intercambio (CAS Compare-and-Swap) y su variable Compare-and-Exchange (CAExch) son agnósticas al tipo
- Pueden ser usadas para implementar operaciones RMW para tipos más complejos

```
1 CAExch(ptr, oldValue, newValue);
```

- If $oldValue == (*ptr)$ then $(*ptr) = newValue$
- Else $oldValue = (*ptr)$

LOAD → Atomic Load

- La operación de comparación-e-intercambio (CAS Compare-and-Swap) y su variable Compare-and-Exchange (CAExch) son agnósticas al tipo
- Pueden ser usadas para implementar operaciones RMW para tipos más complejos

```
1 CAExch(ptr, oldValue, newValue);
```

- If $oldValue == (*ptr)$ then $(*ptr) = newValue$
- Else $oldValue = (*ptr)$

```
1 var oldValue = LOAD(ptr);
2 var newValue;
3 do {
4   newValue = new;
5 } while (!CAExch(ptr, oldValue, newValue));
```

LOAD → Atomic Load

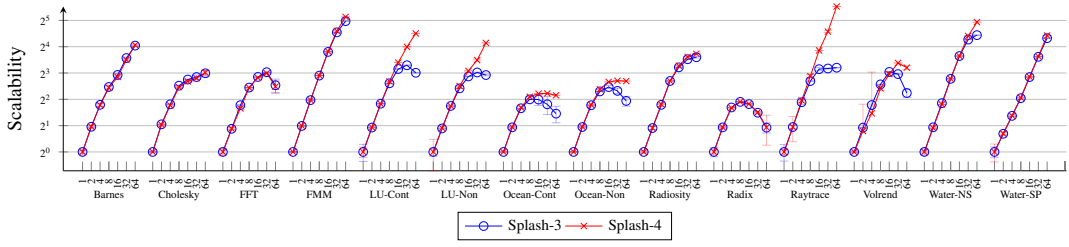
Splash-4: Particionado de Secciones Críticas

- Splash-4 usa constructos lock-free para manipular una única dirección de memoria y por tanto reemplaza secciones críticas que modifican una única dirección de memoria
- Partir secciones críticas que modifican más de una dirección nos permitiría usar constructos lock-free en más casos
- Desafortunadamente, partir secciones críticas, por lo general, no es posible
- Varias secciones críticas en Splash-3, actualizan (atómicamente) variables independientes y se encuentran agrupadas en secciones críticas más grandes **sin razón aparente**

Evaluación: Metodología

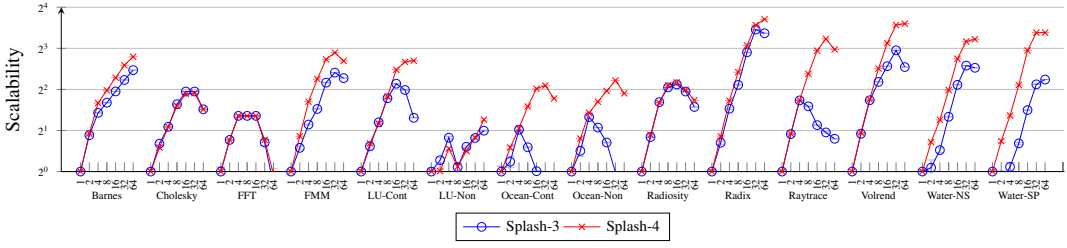
- Las mediciones se han realizado en la región de interés (ROI)
- Hemos usado las entradas recomendadas de Splash-3 para todas las ejecuciones
- El hardware usado es un AMD Epyc 7702P CPU
- La máquina simulada (en gem5-20) intenta imitar un procesador Intel IceLake
  - Las estadísticas se reinician después de un periodo de warm-up (como sugieren los desarrolladores originales de Splash-2)

EVALUACIÓN: ESCALABILIDAD EN EL SIMULADOR



- En general, la mejora de escalabilidad pasa de 16 a 32

EVALUACIÓN: ESCALABILIDAD EN LA MÁQUINA REAL



- En general, la escalabilidad mejora de 4 a 16

SPLASH-4: RESUMEN

- Un total de 50 secciones críticas (33%) han sido modificadas.
  - 27 secciones críticas (18%) han sido convertidos a atómicos de C11
  - 18 secciones críticas (12%)han sido convertidas a constructos-CAS (sin particionar)
  - Particionar las secciones críticas nos ha permitido transformar a constructos-CAS 5 secciones críticas adicionales (3%) usando 15 constructos-CAS