

Grado Universitario en Ingeniería Informática

2020-2021

Trabajo Fin de Grado

“Estudio de calidad de software en benchmarks científicos”

Alberto García García

Tutor

José Daniel García Sánchez

Leganés, 2021



Esta obra se encuentra sujeta a la licencia Creative Commons **Reconocimiento**
– No Comercial – Sin Obra Derivada

ABSTRACT

Despite the undeniable benefits it provides, software quality continues to be one of the most underrated code properties. This is especially characteristic in scientific applications, where the lack of quality is justified by the need for high performance.

In this project, in order to verify the veracity of this statement, it has been decided to use the PARSEC benchmark, a set of applications for the evaluation of multiprocessors, as a scientific software model. During the study, the code quality of the benchmark applications has been iteratively improved and, in parallel, the most common errors in the C++ language, one of the most widely used programming languages in scientific applications, have been collected and analyzed.

After iteratively improving the software quality, the performance of each of the modified benchmark applications has been evaluated in order to assess the impact of these modifications. As a result, it has been obtained that the different quality improvements applied in the PARSEC benchmark have not negatively affected performance in any case, and yet some of them, such as the improvements in dynamic memory management, have had a slight positive impact.

In conclusion, this study has determined that the quality of the code is not incompatible with the performance of the program, and that, in some cases, it can even enhance it.

Keywords: Quality · Performance · Scientific software · Analysis · C++ · Benchmark · PARSEC

RESUMEN

A pesar de las innegables ventajas que aporta, la calidad del software continúa siendo una de las propiedades del código más infravaloradas. Esto es especialmente característico en las aplicaciones de carácter científico, donde la falta de calidad se justifica con la necesidad de alto rendimiento.

Para comprobar la veracidad de esta afirmación, en este proyecto se ha decidido emplear como modelo de software científico el *benchmark* PARSEC, un conjunto de aplicaciones destinadas a la evaluación de multiprocesadores. Durante el estudio se ha mejorado iterativamente la calidad del código de las aplicaciones del *benchmark* y, paralelamente, se han recogido y analizado los errores más comunes en el lenguaje C++, uno de los lenguajes de programación más utilizados en las aplicaciones científicas.

Tras mejorar iterativamente la calidad del software, se ha evaluado el rendimiento de cada una de las aplicaciones modificadas del *benchmark* con el objetivo de comprobar el impacto de estas modificaciones. Como resultado se ha obtenido que las numerosas mejoras de calidad aplicadas en el *benchmark* PARSEC no han afectado negativamente al rendimiento en ningún caso, y sin embargo, algunas han llegado a tener un ligero impacto positivo, como es el caso de las mejoras en el manejo dinámico de la memoria.

En conclusión, este estudio ha determinado que la calidad del código no es incompatible con el rendimiento del programa, y que, en ocasiones, puede hasta favorecerlo.

Palabras clave: Calidad · Rendimiento · Software científico · Análisis · C++ · *Benchmark* ·

Dedicado a
mi tutor, José Daniel, por guiarme con paciencia en este proyecto;
a mis padres y hermanos, por apoyarme y aguantarme en los momentos
más difíciles;
a mis amigos de toda la vida, por haber estado siempre a mi lado;
a mis amigos de la carrera, por haberme hecho disfrutar tanto de estos
cuatro años;
y, en especial, a mis abuelos, Domi y Víctor, por esa infinita generosidad
que nunca os podré compensar.

ÍNDICE GENERAL

1.	INTRODUCCIÓN	1
1.1.	Motivación	1
1.2.	Objetivos	2
1.3.	Estructura del documento.....	3
1.4.	Acrónimos	4
2.	ESTADO DEL ARTE	5
2.1.	La calidad del software	5
2.1.1.	Estudios precedentes.....	5
2.1.2.	Dimensiones de la calidad del software.....	6
2.2.	Historia de C++	9
2.2.1.	Antecedentes (1977-1979).....	10
2.2.2.	C con Clases (1979-1982).....	10
2.2.3.	La evolución de C con Clases a C++ (1982-1991).....	11
2.2.4.	Estandarización (1991-1997).....	13
2.2.5.	Periodo de mantenimiento (1997-2003)	14
2.2.6.	Tiempos difíciles (2003-2011).....	14
2.2.7.	C++ en la década de 2010 (2011-2020).....	15
2.2.8.	C++ en la actualidad (2020-¿?).....	16
2.3.	Guías y convenciones de C++.....	17
2.3.1.	<i>C++ Core Guidelines</i>	17
2.3.2.	<i>CERT Secure Coding Guidelines</i>	17
2.3.3.	<i>High Integrity C++ Coding Standard</i>	17
2.4.	<i>Benchmarks</i>	18
2.4.1.	<i>Benchmark PARSEC</i>	18
3.	ESTUDIO DE LA CALIDAD	25
3.1.	Modernización del código fuente.....	25
3.1.1.	Detección y solución de los errores de compilación.....	25
3.1.2.	Recopilación, análisis y solución de los avisos de compilación.....	29
3.2.	Análisis dinámico del código fuente	41
3.3.	Análisis estático del código fuente.....	42
4.	EVALUACIÓN DE LOS RESULTADOS	57
4.1.	Análisis de rendimiento.....	57

4.1.1.	Entorno, metodología de análisis y repositorio	57
4.1.2.	Versión modernizada comparada con la versión base	59
4.1.3.	Versión «dinámica» comparada con la versión modernizada.....	63
4.1.4.	Versión «estática» comparada con la versión «dinámica»	65
4.2.	Análisis de la complejidad ciclométrica	67
5.	MARCO REGULADOR.....	69
5.1.	Análisis legislativo	69
5.1.1.	PARSEC	69
5.1.2.	Otras herramientas	71
5.2.	Estándares técnicos	71
5.2.1.	Estándar ISO/IEC 14882	71
5.2.2.	Estándar ISO/IEC 25010	72
6.	ENTORNO SOCIOECONÓMICO	73
6.1.	Planificación.....	73
6.2.	Presupuesto.....	77
6.2.1.	Recursos humanos	77
6.2.2.	Recursos materiales	78
6.2.3.	Costes indirectos	79
6.2.4.	Coste total	79
6.3.	Impacto socioeconómico.....	79
7.	CONCLUSIONES	81
7.1.	Objetivos cumplidos.....	81
7.2.	Futuras líneas de trabajo.....	82
8.	ENGLISH SUMMARY	83
8.1.	Introduction	83
8.2.	State of the Art	84
8.2.1.	Literature Review	84
8.2.2.	Standard Quality Dimensions	84
8.2.3.	The C++ History	85
8.2.4.	Benchmarks	85
8.3.	Quality Study.....	86
8.3.1.	Source Code Modernization	86
8.3.2.	Dynamic Analysis of the Source Code	89
8.3.3.	Static Analysis of the Source Code.....	90

8.4.	Conclusions and future work	93
9.	BIBLIOGRAFÍA	95
A.	ANEXO A: FALLOS DE CALIDAD	A-1
A.1.	Avisos de compilación	A-1
A.1.1	Blackscholes	A-1
A.1.2	Bodytrack	A-1
A.1.3	Canneal	A-3
A.1.4	Dedup	A-4
A.1.5	Facesim	A-4
A.1.6	Ferret	A-4
A.1.7	Fluidanimate	A-6
A.1.8	Freqmine	A-6
A.1.9	Netdedup	A-7
A.1.10	Netferret	A-7
A.1.11	Netstreamcluster	A-9
A.1.12	Raytrace	A-9
A.1.13	Streamcluster	A-10
A.1.14	Swaptions	A-10
A.1.15	Vips	A-10
A.1.16	x264	A-10
A.2.	Fugas de memoria	A-18
A.2.1	Freqmine	A-18
A.2.2	Netstreamcluster	A-19
A.2.3	Streamcluster	A-21
A.2.4	Swaptions	A-22
B.	ANEXO B: EXPERIMENTACIÓN	B-1
B.1.	Blackscholes	B-1
B.2.	Bodytrack	B-3
B.3.	Canneal	B-6
B.4.	Fluidanimate	B-8
B.5.	Freqmine	B-11
B.6.	Netstreamcluster	B-16
B.7.	Streamcluster	B-21
B.8.	Swaptions	B-26

ÍNDICE DE FIGURAS

Fig. 2.1. Ecuación diferencial parcial Black-Scholes [31]	18
Fig. 2.2. <i>Output</i> de la aplicación Bodytrack [31]	19
Fig. 2.3. Dos fotogramas del <i>output</i> de la aplicación Facesim [31]	20
Fig. 2.4. Algoritmo del motor de búsqueda Ferret [31]	20
Fig. 2.5. Captura de pantalla de un videojuego con efectos de partículas (izq.) y sin ellas (der.) [31]	21
Fig. 2.6. Demostración de la aplicación Raytrace sobre un videojuego [31]	21
Fig. 2.7. <i>Input</i> de la aplicación Vips en PARSEC [31]	22
Fig. 2.8. Fotograma de entrada de la aplicación x264 en PARSEC [31]	23
Fig. 3.1. Excepción dinámica (archivo Thread.h)	26
Fig. 3.2. Error en archivos <i>.pod</i> (archivo smime.pod)	26
Fig. 3.3. Solución al error en archivos <i>.pod</i> (archivo smime.pod)	27
Fig. 3.4. Declaraciones con la palabra clave <i>extern</i> (archivo decoder.c)	27
Fig. 3.5. Comparación con conversión implícita (archivo FILE_UTILITIES.cpp)	27
Fig. 3.6. Conversión explícita (archivo FILE_UTILITIES.cpp)	28
Fig. 3.7. Declaración de la variable <code>__mbstate_t</code> (archivo <code>bsd__types.h</code>)	28
Fig. 3.8. Constante <i>HUGE</i> (archivo LSH_query.c)	28
Fig. 3.9. Opciones <code>-fno-pie</code> y <code>-no-pie</code> (archivo <code>gcc-pthreads.bldconf</code>)	29
Fig. 3.10. Fragmento de código del aviso 01 (main.cpp, Bodytrack)	31
Fig. 3.11. Fragmento de código del aviso 02 (streamcluster.cpp, Streamcluster)	31
Fig. 3.12. Fragmento de código del aviso 03 (FlexImage.cpp, Bodytrack)	32
Fig. 3.13. Fragmento de código del aviso 04 (FlexIO.cpp, Bodytrack)	33
Fig. 3.14. Fragmento de código del aviso 05 (Mutex.cpp, Bodytrack)	34
Fig. 3.15. Fragmento de código del aviso 06 (AsyncIO.h, Bodytrack)	35

Fig. 3.16. Fragmento de código del aviso 07 (ParticleFilterPthread.h, Bodytrack).....	36
Fig. 3.17. Fragmento de código del aviso 08 (Thread.cpp, Bodytrack).....	37
Fig. 3.18. Fragmento de código del aviso 09 (MersenneTwister.h, Canneal).....	38
Fig. 3.19. Fragmento de código del aviso 10 (main.cpp, Canneal).....	39
Fig. 3.20. Fragmento de código del aviso 11 (client.cpp, Netstreamcluster).....	40
Fig. 3.21. Fragmento de código del aviso 12 (client.cpp, Netstreamcluster).....	40
Fig. 3.22. Conversión con posible reducción de memoria (streamcluster.cpp)	44
Fig. 3.23. Uso de la función <i>strcpy</i> (streamcluster.cpp).....	44
Fig. 3.24. Uso de variables de tipo <i>string</i> (streamcluster.cpp).....	44
Fig. 3.25. Uso de funciones <i>vararg</i> (streamcluster.cpp)	45
Fig. 3.26. Uso de funciones de la librería {fmt} (streamcluster.cpp)	45
Fig. 3.27. Declaración de variables sin inicialización (streamcluster.cpp)	46
Fig. 3.28. Uso de aritmética de punteros (streamcluster.cpp)	46
Fig. 3.29. Uso de vectores (streamcluster.cpp)	46
Fig. 3.30. Manejo manual de la memoria (streamcluster.cpp)	47
Fig. 3.31. Uso de punteros inteligentes (streamcluster.cpp)	47
Fig. 3.32. Uso de valores numéricos sin declarar (streamcluster.cpp).....	47
Fig. 3.33. Uso de <i>constexpr</i> (streamcluster.cpp)	48
Fig. 3.34. Clase incumpliendo la «regla del cinco» (streamcluster.cpp).....	48
Fig. 3.35. Clase cumpliendo la «regla del cinco» (streamcluster.cpp).....	49
Fig. 3.36. Uso de <i>override</i> (streamcluster.cpp)	49
Fig. 3.37. Uso de <i>explicit</i> (streamcluster.cpp).....	50
Fig. 3.38. Cabeceras de C (streamcluster.cpp)	50
Fig. 3.39. Cabeceras de C++ (streamcluster.cpp)	50
Fig. 3.40. Uso de <i>assert</i> (parsec_barrier.cpp)	51
Fig. 3.41. Uso de <i>static_assert</i> (parsec_barrier.cpp).....	51

Fig. 3.42. Declaraciones condicionales sin llaves (<code>parsec_barrier.cpp</code>).....	51
Fig. 3.43. Declaraciones condicionales con llaves (<code>parsec_barrier.cpp</code>).....	51
Fig. 3.44. Declaración de un iterador (<code>streamcluster.cpp</code>)	52
Fig. 3.45. Declaración con <i>auto</i> (<code>streamcluster.cpp</code>).....	52
Fig. 3.46. Función especial con cuerpo trivial usando llaves (<code>streamcluster.cpp</code>)	52
Fig. 3.47. Función especial con cuerpo trivial usando «= default» (<code>streamcluster.cpp</code>)	52
Fig. 3.48. Falso positivo de un parámetro sin uso (<code>streamcluster.cpp</code>)	53
Fig. 3.49. Uso de <i>maybe_unused</i> (<code>streamcluster.cpp</code>).....	53
Fig. 3.50. Uso de <i>NULL</i> (<code>streamcluster.cpp</code>).....	54
Fig. 3.51. Uso de <i>nullptr</i> (<code>streamcluster.cpp</code>).....	54
Fig. 3.52. Declaración de alias por medio de <i>typedef</i> (<code>parsec_barrier.hpp</code>)	54
Fig. 3.53. Declaración de alias por medio de <i>using</i> (<code>parsec_barrier.hpp</code>)	54
Fig. 3.54. Uso de un valor entero como <i>bool</i> (<code>streamcluster.cpp</code>)	54
Fig. 3.55. Uso de un literal <i>bool</i> (<code>streamcluster.cpp</code>)	54
Fig. 3.56. Parámetro de tipo puntero susceptible de ser declarado constante (<code>streamcluster.cpp</code>)	55
Fig. 3.57. Parámetro de tipo puntero constante (<code>streamcluster.cpp</code>)	55
Fig. 3.58. Conversión implícita de <i>int</i> a <i>bool</i> (<code>parsec_barrier.cpp</code>).....	55
Fig. 3.59. Condición booleana sin conversiones implícitas (<code>parsec_barrier.cpp</code>).....	55
Fig. 4.1. <i>Output</i> de una aplicación del <i>benchmark</i> PARSEC.....	57
Fig. 4.2. Fórmula del <i>speedup</i>	58
Fig. 4.3. Gráfica del tiempo de ejecución en la aplicación Streamcluster (versión base frente a versión modernizada)	62
Fig. 4.4. Gráfica del tiempo de ejecución en la aplicación Streamcluster (versión modernizada frente a versión «dinámica»).....	65

Fig. 4.5. Gráfica del tiempo de ejecución en la aplicación Streamcluster (versión «dinámica» frente a versión «estática»)	66
Fig. 5.1. Modelo ISO/IEC 25010 [66]	72
Fig. 6.1. Diagrama de Gantt: planificación estimada	75
Fig. 6.2. Diagrama de Gantt: planificación real	76
Fig. 6.3. Fórmula del coste amortizado	78
Fig. 8.1. Dynamic exception declaration (Thread.h file)	87
Fig. 8.2. Implicit conversion (FILE_UTILITIES.cpp file)	87
Fig. 8.3. Explicit cast (FILE_UTILITIES.cpp file)	87
Fig. 8.4. Formatting function error (streamcluster.cpp, Streamcluster)	87
Fig. 8.5. Streamcluster execution times (first iteration)	88
Fig. 8.6. Streamcluster execution times (second iteration)	90
Fig. 8.7. Manual memory management (streamcluster.cpp)	91
Fig. 8.8. Smart pointers solution (streamcluster.cpp)	91
Fig. 8.9. Streamcluster execution times (third iteration)	92

ÍNDICE DE TABLAS

Tabla 3.1. Aplicaciones de PARSEC con avisos de compilación	30
Tabla 4.1. Rendimiento de la versión modernizada frente a la versión base en la aplicación Blackscholes.....	59
Tabla 4.2. Rendimiento de la versión modernizada frente a la versión base en la aplicación Bodytrack	60
Tabla 4.3. Rendimiento de la versión modernizada frente a la versión base en la aplicación Canneal.....	60
Tabla 4.4. Rendimiento de la versión modernizada frente a la versión base en la aplicación Fluidanimate.....	60
Tabla 4.5. Rendimiento de la versión modernizada frente a la versión base en la aplicación Freqmine	61
Tabla 4.6. Rendimiento de la versión modernizada frente a la versión base en la aplicación Netstreamcluster.....	61
Tabla 4.7. Rendimiento de la versión modernizada frente a la versión base en la aplicación Streamcluster.....	61
Tabla 4.8. Rendimiento de la versión modernizada frente a la versión base en la aplicación Swaptions	62
Tabla 4.9. Rendimiento de la versión «dinámica» frente a la versión modernizada en la aplicación Freqmine	63
Tabla 4.10. Rendimiento de la versión «dinámica» frente a la versión modernizada en la aplicación Netstreamcluster.....	64
Tabla 4.11. Rendimiento de la versión «dinámica» frente a la versión modernizada en la aplicación Streamcluster.....	64
Tabla 4.12. Rendimiento de la versión «estática» frente a la versión «dinámica» en la aplicación Streamcluster.....	66
Tabla 4.13. Análisis de la complejidad ciclomática	68
Tabla 6.1. Costes de los recursos humanos	78

Tabla 6.2. Costes del hardware.....	78
Tabla 6.4. Costes indirectos.....	79
Tabla 6.5. Coste total.....	79
Table 8.1. Performance of baseline version vs modernized version in Streamcluster application	88
Table 8.2. Performance of modernized version vs “dynamic” version in Streamcluster application	90
Table 8.3. Performance of “dynamic” version vs “static” version in Streamcluster application	92
Tabla B.1. Tiempos de ejecución en la aplicación Blackscholes (1 hilo)	B–1
Tabla B.2. Tiempos de ejecución en la aplicación Blackscholes (2 hilos).....	B–1
Tabla B.3. Tiempos de ejecución en la aplicación Blackscholes (4 hilos).....	B–2
Tabla B.4. Tiempos de ejecución en la aplicación Blackscholes (8 hilos).....	B–2
Tabla B.5. Tiempos de ejecución en la aplicación Blackscholes (16 hilos).....	B–3
Tabla B.6. Tiempos de ejecución en la aplicación Bodytrack (1 hilo).....	B–3
Tabla B.7. Tiempos de ejecución en la aplicación Bodytrack (2 hilos)	B–4
Tabla B.8. Tiempos de ejecución en la aplicación Bodytrack (4 hilos)	B–4
Tabla B.9. Tiempos de ejecución en la aplicación Bodytrack (8 hilos)	B–5
Tabla B.10. Tiempos de ejecución en la aplicación Bodytrack (16 hilos)	B–5
Tabla B.11. Tiempos de ejecución en la aplicación Canneal (1 hilo)	B–6
Tabla B.12. Tiempos de ejecución en la aplicación Canneal (2 hilos).....	B–6
Tabla B.13. Tiempos de ejecución en la aplicación Canneal (4 hilos).....	B–7
Tabla B.14. Tiempos de ejecución en la aplicación Canneal (8 hilos).....	B–7
Tabla B.15. Tiempos de ejecución en la aplicación Canneal (16 hilos).....	B–8
Tabla B.16. Tiempos de ejecución en la aplicación Fluidanimate (1 hilo)	B–8
Tabla B.17. Tiempos de ejecución en la aplicación Fluidanimate (2 hilos).....	B–9

Tabla B.18. Tiempos de ejecución en la aplicación Fluidanimate (4 hilos).....	B-9
Tabla B.19. Tiempos de ejecución en la aplicación Fluidanimate (8 hilos).....	B-10
Tabla B.20. Tiempos de ejecución en la aplicación Fluidanimate (16 hilos).....	B-10
Tabla B.21. Tiempos de ejecución en la aplicación Freqmine (1 hilo) (parte 1)	B-11
Tabla B.22. Tiempos de ejecución en la aplicación Freqmine (1 hilo) (parte 2)	B-11
Tabla B.23. Tiempos de ejecución en la aplicación Freqmine (2 hilos) (parte 1).....	B-12
Tabla B.24. Tiempos de ejecución en la aplicación Freqmine (2 hilos) (parte 2).....	B-12
Tabla B.25. Tiempos de ejecución en la aplicación Freqmine (4 hilos) (parte 1).....	B-13
Tabla B.26. Tiempos de ejecución en la aplicación Freqmine (4 hilos) (parte 2).....	B-13
Tabla B.27. Tiempos de ejecución en la aplicación Freqmine (8 hilos) (parte 1).....	B-14
Tabla B.28. Tiempos de ejecución en la aplicación Freqmine (8 hilos) (parte 2).....	B-14
Tabla B.29. Tiempos de ejecución en la aplicación Freqmine (16 hilos) (parte 1)...	B-15
Tabla B.30. Tiempos de ejecución en la aplicación Freqmine (16 hilos) (parte 2)...	B-15
Tabla B.31. Tiempos de ejecución en la aplicación Netstreamcluster (1 hilo) (parte 1)	B-16
Tabla B.32. Tiempos de ejecución en la aplicación Netstreamcluster (1 hilo) (parte 2)	B-16
Tabla B.33. Tiempos de ejecución en la aplicación Netstreamcluster (2 hilos) (parte 1)	B-17
Tabla B.34. Tiempos de ejecución en la aplicación Netstreamcluster (2 hilos) (parte 2)	B-17
Tabla B.35. Tiempos de ejecución en la aplicación Netstreamcluster (4 hilos) (parte 1)	B-18
Tabla B.36. Tiempos de ejecución en la aplicación Netstreamcluster (4 hilos) (parte 2)	B-18
Tabla B.37. Tiempos de ejecución en la aplicación Netstreamcluster (8 hilos) (parte 1)	B-19

Tabla B.38. Tiempos de ejecución en la aplicación Netstreamcluster (8 hilos) (parte 2)	B-19
Tabla B.39. Tiempos de ejecución en la aplicación Netstreamcluster (16 hilos) (parte 1)	B-20
Tabla B.40. Tiempos de ejecución en la aplicación Netstreamcluster (16 hilos) (parte 2)	B-20
Tabla B.41. Tiempos de ejecución en la aplicación Streamcluster (1 hilo) (parte 1)	B-21
Tabla B.42. Tiempos de ejecución en la aplicación Streamcluster (1 hilo) (parte 2)	B-21
Tabla B.43. Tiempos de ejecución en la aplicación Streamcluster (2 hilos) (parte 1) ..	B-22
Tabla B.44. Tiempos de ejecución en la aplicación Streamcluster (2 hilos) (parte 2) ..	B-22
Tabla B.45. Tiempos de ejecución en la aplicación Streamcluster (4 hilos) (parte 1) ..	B-23
Tabla B.46. Tiempos de ejecución en la aplicación Streamcluster (4 hilos) (parte 2) ..	B-23
Tabla B.47. Tiempos de ejecución en la aplicación Streamcluster (8 hilos) (parte 1) ..	B-24
Tabla B.48. Tiempos de ejecución en la aplicación Streamcluster (8 hilos) (parte 2) ..	B-24
Tabla B.49. Tiempos de ejecución en la aplicación Streamcluster (16 hilos) (parte 1)	B-25
Tabla B.50. Tiempos de ejecución en la aplicación Streamcluster (16 hilos) (parte 2)	B-25
Tabla B.51. Tiempos de ejecución en la aplicación Swaptions (1 hilo).....	B-26
Tabla B.52. Tiempos de ejecución en la aplicación Swaptions (2 hilos)	B-26
Tabla B.53. Tiempos de ejecución en la aplicación Swaptions (4 hilos)	B-27
Tabla B.54. Tiempos de ejecución en la aplicación Swaptions (8 hilos)	B-27

Tabla B.55. Tiempos de ejecución en la aplicación Swaptions (16 hilos)	B-28
---	------

1. INTRODUCCIÓN

La calidad del código es una de las propiedades más infravaloradas en el desarrollo de software. A pesar de haber logrado cierto nivel de importancia en los últimos años gracias a los beneficios económicos que reporta en la producción de software comercial, esta característica suele quedar relegada a un segundo plano, a la sombra de otras cualidades más perceptibles.

Esta realidad es especialmente notable en el ámbito científico, donde se suele recurrir a la necesidad de rendimiento como excusa para justificar la ausencia de calidad en el código, como si ambas propiedades fuesen excluyentes. La veracidad de esta afirmación será comprobada a través de este estudio.

En este primer capítulo se presentan los motivos que han llevado a la elaboración de este estudio de calidad del software en el ámbito científico, así como los objetivos principales que se pretenden lograr con este proyecto. También incluye una sección donde se define la estructura de este documento, así como una lista de los acrónimos empleados.

1.1. Motivación

A pesar de ser una cualidad frecuentemente ignorada, la calidad del software es una de las propiedades que más beneficios genera. No sólo evita posibles defectos en etapas maduras del desarrollo, donde el coste de solucionar un error es considerablemente más alto, sino que también facilita el mantenimiento del código, su legibilidad y su portabilidad. En consecuencia, la calidad del código es posiblemente una de las mejores inversiones en el desarrollo de software.

En el ámbito del software comercial, estas razones han sido suficientes para que la calidad con la que se escriben los programas haya comenzado a recibir la atención que merece. Sin embargo, dentro del entorno científico, cualquier argumento es eludido con el pretexto del rendimiento. Aun cuando no hay estudios ni pruebas que demuestren que la calidad del código pueda tener un impacto negativo sobre el rendimiento de un programa, esta creencia está bastante extendida en la comunidad científica y, en consecuencia, el software científico suele presentar unos niveles de calidad ciertamente dudosos,

provocando que los programas sean difíciles de mantener y actualizar a lo largo de los años.

Dentro de este ámbito científico uno de los lenguajes de programación más utilizados para el desarrollo de los programas es C++. Comparado con otros lenguajes distinguidos como Java o Python, C++ es el que mejor rendimiento proporciona, una propiedad muy valorada en el software científico, tal y como se ha comentado anteriormente. C++ es un lenguaje maduro, altamente extendido y flexible, concebido para ser utilizado en cualquier dominio de aplicación y proporcionar unos niveles de rendimiento superiores al resto de lenguajes de alto nivel.

El problema de C++ es que, debido a su carácter generalista, permite algunas técnicas de cierto riesgo, como el manejo manual de la memoria, el uso de antiguas funciones de C o las operaciones con punteros. A pesar de que estas técnicas no son intrínsecamente peligrosas, su uso incorrecto puede provocar serios defectos en el desarrollo y mantenimiento de los programas. Así pues, es fundamental conocer a fondo el lenguaje para poder aprovechar de forma segura todas las posibilidades que proporciona.

1.2. Objetivos

El objetivo principal de este estudio es evaluar el impacto que tiene la calidad del código sobre el rendimiento del programa, concretamente en el ámbito del software científico. Para ello, se ha escogido como modelo de estudio el *benchmark* PARSEC, un conjunto de aplicaciones destinadas a la evaluación de arquitecturas paralelas.

Durante este estudio de calidad se procederá al análisis de las aplicaciones que conforman el *benchmark*, se estudiarán los fallos de calidad encontrados y se modernizará el código de forma iterativa, comprobando en cada iteración las consecuencias que las modificaciones realizadas hayan podido tener sobre el rendimiento.

Paralelamente al análisis del impacto que las mejoras de calidad puedan provocar sobre el rendimiento de las aplicaciones, se procederá a estudiar cuáles son los errores más frecuentes del lenguaje C++ cometidos en aplicaciones de carácter científico. El objetivo de este análisis es proponer soluciones y alternativas a estas prácticas defectuosas y poder explotar las capacidades del lenguaje de una forma fiable, garantizando la máxima calidad posible.

1.3. Estructura del documento

El presente documento se divide en ocho capítulos diferentes:

- **Introducción:** presenta las motivaciones y objetivos principales del proyecto, así como la estructura de la memoria. También se incluye la lista de acrónimos empleados en este documento.
- **Estado del arte:** describe la literatura precedente sobre la calidad del software y define las dimensiones estándar que componen dicha calidad, utilizadas en este proyecto. También contiene un breve resumen de la historia de C++, pilar fundamental de este estudio, así como las guías y convenciones del lenguaje de mayor relevancia para el proyecto. Por último se expone la definición de *benchmark* y se introduce el *benchmark* PARSEC, elegido como modelo para este estudio de calidad.
- **Estudio de la calidad:** recoge las diferentes iteraciones realizadas para mejorar la calidad del software del *benchmark* PARSEC. La primera iteración consiste en la modernización básica del código, solventado los avisos de compilación que generan las diferentes aplicaciones. En la segunda y tercera iteración se lleva a cabo el análisis dinámico y estático del código, respectivamente.
- **Evaluación de los resultados:** compara el rendimiento de las aplicaciones del *benchmark* antes y después de las modificaciones realizadas en las diferentes iteraciones del estudio. También incorpora un análisis del impacto de dichas modificaciones en la complejidad ciclomática del código.
- **Marco regulador:** expone la legislación vigente que afecta al proyecto, así como los estándares técnicos aplicados durante la realización del mismo.
- **Entorno socioeconómico:** incluye la planificación del proyecto, el presupuesto estimado para su realización y el potencial impacto socioeconómico del mismo.
- **Conclusiones:** expone los objetivos logrados durante el estudio y plantea las posibles líneas de trabajo futuras que nacen de este proyecto.
- **English Summary:** resume en inglés y de forma concisa todo el estudio.

1.4. Acrónimos

3D	<i>3 dimensiones</i>
AVC	<i>Advanced Video Coding</i>
CERN	<i>Conseil Européen pour la Recherche Nucléaire</i>
DDR4	<i>Double Data Rate Fourth Generation</i>
FP-Growth	<i>Frequent Pattern Growth</i>
GCC	<i>GNU Compiler Collection</i>
HJM	<i>Heath-Jarrow-Morton</i>
I/O	<i>Input/Output</i>
IEC	<i>International Electrotechnical Commission</i>
ISO	<i>International Organization for Standardization</i>
LLVM	<i>Low Level Virtual Machine</i>
PARSEC	<i>Princeton Application Repository for Shared-Memory Computers</i>
RMS	<i>Recognition, Mining and Synthesis</i>
SSL	<i>Secure Socket Layer</i>
STL	<i>Standard Template Library</i>
TCP/IP	<i>Transmission Control Protocol/Internet Protocol</i>
VIPS	<i>VASARI Image Processing System</i>

2. ESTADO DEL ARTE

En este segundo capítulo se expone el estado del arte, estableciendo las nociones que suponen la base de este trabajo. En primer lugar, se realiza una breve revisión de la literatura que precede a este estudio y se definen las dimensiones que se utilizarán para analizar la calidad del software. Posteriormente se desarrolla un resumen conciso de la historia de C++, lenguaje de programación sobre el que está cimentado este proyecto, y se exponen las guías y convenciones del lenguaje seguidas en este proyecto. Por último, se ilustra la definición de *benchmark* y se comentan las distintas aplicaciones que componen PARSEC, *benchmark* elegido para ser analizado en este estudio sobre la calidad del software científico.

2.1. La calidad del software

2.1.1. Estudios precedentes

Desde los últimos años la calidad del software se ha convertido en un tema frecuente dentro de la literatura científica. No obstante, muchas de estas publicaciones, véase [1, 2, 3, 4], se quedan en el nivel más alto y tienden a centrarse más en cómo lograr dicha calidad o en su impacto económico (costes, retribuciones, etc.), sin detenerse a evaluar el efecto que pudiera tener en el rendimiento o a estudiar las razones detrás de los errores de calidad.

Si se reduce el dominio de búsqueda a las publicaciones que sí evalúan este impacto, puede apreciarse que la mayoría de artículos, como ocurre en [5], estudian la calidad de software comercial, de nuevo motivados por el aspecto económico. Encontrar estudios enfocados en el software científico, es decir, aquel empleado en los programas y sistemas destinados a resolver problemas de investigación e ingeniería, es relativamente más difícil, pero no imposible.

En 2005 Raghavan, Irwin, McInnes y Norris publicaron un interesante estudio destinado a evaluar las compensaciones entre calidad, rendimiento y consumo en la computación científica [6]. En este estudio Raghavan et al., quienes definen la calidad del software como su precisión, fiabilidad y escalabilidad, defienden que una aplicación de alto rendimiento, como las utilizadas en el ámbito científico, puede cumplir con unos

requisitos determinados de calidad y rendimiento a la vez que disminuye su consumo mediante técnicas como el escalado dinámico del voltaje. A pesar de que el estudio citado se centre más en la optimización del consumo de energía, la publicación sugiere que dicha optimización es necesaria para que la calidad y el rendimiento sean características compatibles en la computación de alto rendimiento, como si, en condiciones normales, una inevitablemente mermase a la otra.

2.1.2. Dimensiones de la calidad del software

La Real Academia Española define la calidad como «la propiedad o propiedades inherentes a algo, que permiten juzgar su valor» [7]. A pesar de que esta definición puede aclarar más o menos el término y proporcionar una idea coherente sobre qué es la calidad, en el ámbito del desarrollo de software puede ser difícil fijar cuáles son estas propiedades.

Por lo general, se tiende a analizar la calidad de un programa como si se tratase de una cualidad atómica e indivisible del software o, como mucho, estudiando algunas de las características del código, como su legibilidad o su modularidad. Con el objetivo de evitar estas divergencias y normalizar el concepto de calidad del software, así como su estudio y evaluación, la Organización Internacional de Normalización (ISO), junto a la Comisión Electrotécnica Internacional (IEC), definió la familia ISO/IEC 25000 [8], un conjunto de estándares conocido como SQuaRE (*System and software Quality Requirements and Evaluation*).

Dentro de esta familia, encontramos el estándar ISO/IEC 25010 [9], el cual establece una serie de dimensiones que componen la calidad del software, las cuales pueden dividirse a su vez en propiedades más concretas:

- **Adecuación funcional:** grado en el que el software proporciona funciones que cumplen las necesidades declaradas e implícitas cuando dicho software se utiliza en las condiciones definidas. Esta propiedad se divide en:
 - Integridad funcional: grado en el que las funciones cubren todas las tareas definidas, así como los objetivos del usuario.
 - Corrección funcional: grado en el que las funciones proporcionan los resultados correctos con el grado de precisión necesario.
 - Idoneidad funcional: grado en el que las funciones facilitan el cumplimiento de las tareas y objetivos definidos.

- **Eficiencia de rendimiento:** rendimiento del software respecto al número de recursos utilizados en las condiciones definidas. Esta propiedad se divide en:
 - Comportamiento temporal: grado en el que los tiempos de respuesta y de procesamiento del software (al realizar sus funciones) cumplen con los requisitos definidos.
 - Utilización de recursos: grado en el que las proporciones y tipos de recursos empleados por el software (al realizar sus funciones) cumplen con los requisitos definidos.
 - Capacidad: grado en el que los límites máximos de los parámetros del software (como el número de usuarios concurrentes o el tamaño de los elementos guardados) cumplen con los requisitos definidos.
- **Compatibilidad:** grado en el que el software puede intercambiar información con otros productos, sistemas o componentes (y cumplir con sus funciones establecidas) mientras comparten el mismo entorno. Esta propiedad se divide en:
 - Coexistencia: grado en el que el software puede desarrollar sus funciones eficientemente mientras comparte entorno y recursos con otros productos, sin provocar en ellos ningún impacto negativo.
 - Interoperabilidad: grado en el que el software puede intercambiar información con otro producto, sistema o componente y utilizar dicha información intercambiada.
- **Usabilidad:** grado en el que el software puede ser utilizado por usuarios concretos para cumplir los objetivos definidos con eficiencia, eficacia y satisfacción en un contexto de uso determinado. Esta propiedad se divide en:
 - Apreciabilidad de la idoneidad: grado en el que los usuarios pueden reconocer si el software se adecua a sus necesidades.
 - Capacidad de aprendizaje: grado en el que los usuarios pueden aprender a utilizar el software con eficiencia, eficacia, satisfacción y sin riesgo, en un entorno de uso determinado.
 - Operabilidad: grado en el que el software dispone de atributos que facilitan su operación y control.
 - Protección contra errores del usuario: grado en el que el software protege al usuario de cometer fallos mientras lo utiliza.

- Estética de la interfaz de usuario: grado en el que la interfaz proporciona una interacción satisfactoria y agradable para el usuario.
- Accesibilidad: grado en el que el software puede ser utilizado por personas con la más amplia gama de características y capacidades en un contexto de uso determinado.
- **Fiabilidad:** grado en el que el software cumple con sus funciones bajo unas condiciones determinadas y durante un periodo definido de tiempo. Esta propiedad se divide en:
 - Madurez: grado en el que el software cumple con las necesidades de fiabilidad durante su uso estándar.
 - Disponibilidad: grado en el que el software es accesible y operativo cuando se requiere su utilización.
 - Tolerancia al fallo: grado en el que el software funciona como se espera a pesar de la presencia de fallos (en el hardware o en el propio software).
 - Capacidad de recuperación: grado en el que el software, en caso de fallo o interrupción, es capaz de recuperar los datos afectados y restablecer el estado deseado.
- **Seguridad:** grado en el que el software protege la información y los datos de forma que los usuarios y el resto de componentes o sistemas que interactúan con él tengan el grado de acceso correspondiente con su nivel de autorización.
 - Confidencialidad: grado en el que el software asegura que los datos solo son accesibles para aquellos autorizados a su acceso.
 - Integridad: grado en el que el software previene los datos de modificaciones no autorizadas.
 - Capacidad de no repudio: grado en el que las acciones pueden ser demostradas de haber tenido lugar de tal forma que no puedan ser negadas posteriormente.
 - Responsabilidad: grado en el que las acciones de una entidad pueden ser rastreadas únicamente hasta esa entidad.
 - Autenticidad: grado en el que puede demostrarse que la entidad de un sujeto concuerda con la reclamada.

- **Mantenibilidad:** grado de eficacia y eficiencia con el que el software puede ser modificado (incluyendo correcciones y actualizaciones) por los sujetos encargados de su mantenimiento.
 - Modularidad: grado en el que el software está formado por componentes discretos, de tal forma que la sustitución de uno de estos componentes tenga un impacto mínimo en los demás.
 - Reusabilidad: grado en el que el software puede ser utilizado en más de un sistema o para construir nuevos programas.
 - Capacidad de análisis: grado de eficacia y eficiencia con el que es posible evaluar el impacto de un cambio en uno o varios componentes del software, así como diagnosticar deficiencias, motivos de fallo o componentes que requieren una sustitución.
 - Capacidad de modificación: grado en el que el software puede ser modificado sin degradarlo.
 - Capacidad de evaluación: grado de eficacia y eficiencia con el que se puede establecer un criterio de evaluación para el software, así como la realización de evaluaciones con dichos criterios.
- **Portabilidad:**
 - Adaptabilidad: grado de eficacia y eficiencia con el que el software puede ser trasladado desde un entorno operacional a otro.
 - Capacidad de instalación: grado de eficacia y eficiencia con el que el software puede ser instalado y desinstalado de un entorno determinado.
 - Capacidad de reemplazo: grado en el que el software puede ser reemplazado por otro con los mismos propósitos en el mismo entorno.

2.2. Historia de C++

Desde sus primeros años hasta la actualidad, C++ ha sido uno de los lenguajes de programación más dominantes en el mundo entero [10]. En este epígrafe se pretende hacer un resumen de la extensa historia del lenguaje, basado en las propias publicaciones de su autor, Bjarne Stroustrup [11, 12, 13].

2.2.1. Antecedentes (1977-1979)

Tal y como cuenta su fundador, C++ nació buscando la combinación entre las facilidades para programar del lenguaje Simula y la eficiencia de C. El deseo de Bjarne de encontrar dicha armonía surgió alrededor de 1977 mientras realizaba su tesis doctoral en el Laboratorio de Computación de la Universidad de Cambridge, en Inglaterra. En dicha tesis, Bjarne estaba estudiando diferentes alternativas para la organización del software en sistemas distribuidos y, para experimentar la ejecución del código en dichos sistemas, decidió programar un simulador, el cual terminó siendo considerablemente complejo.

Bjarne escribió la primera versión de este simulador utilizando Simula, descubriendo así las comodidades que dicho lenguaje proporcionaba a la hora de programar: jerarquía de clases, código más legible, facilidad para expresar concurrencia, etc. No obstante, todas estas ventajas en el desarrollo del código quedaron en nada al comprobar Bjarne la pobre implementación de Simula: los tiempos de compilación eran considerablemente elevados y el rendimiento del simulador durante la ejecución era tan bajo que impedía su uso para la experimentación y obtención de datos reales.

En consecuencia, Bjarne se vio obligado a reescribir su simulador en otro lenguaje, BCPL. BCPL era de tan bajo nivel que hacía que la programación y la depuración del código fuese una tarea ardua y tediosa. Sin embargo, a diferencia de Simula, BCPL sí contaba con una buena implementación, por lo que el simulador resultante demostraba un muy buen rendimiento y permitió a su autor utilizarlo para los experimentos de su tesis doctoral.

Tras esta experiencia, Bjarne se decidió a no volver a enfrentarse a un problema sin la herramienta adecuada. Para él, esta «herramienta adecuada» debía contar con las comodidades y la organización de Simula, el rendimiento de lenguajes como BCPL, C o Fortran y una alta portabilidad que permitiese la ejecución de los programas en diferentes máquinas. Esta idea continuaría madurando durante un par de años hasta que en 1979 cobraría gran importancia.

2.2.2. C con Clases (1979-1982)

En 1979, durante el desarrollo de un nuevo proyecto sobre la distribución del kernel de UNIX entre ordenadores conectados mediante una red local, Bjarne se enfrentaba a dos problemas importantes: (1) el análisis del tráfico en la red local resultado de la

distribución del kernel y (2) la modulación del propio kernel. Estos problemas eran del tipo que Bjarne había decidido no volver a enfrentar sin las herramientas adecuadas, lo que le llevó a comenzar el desarrollo del lenguaje de programación que había imaginado desde su tesis en Cambridge: «C con Clases».

C con Clases, predecesor de C++, combinaba las clases de Simula con el lenguaje de C. Este nuevo lenguaje de programación no estaba destinado a ser utilizado en aplicaciones específicas, sino a mejorar la organización de los programas en general. Es por ello por lo que C con Clases podía verse como un lenguaje de programación de carácter general más que una extensión de C desarrollada para un área especializada. Esta dicotomía entre especialización y generalización se presentaría en numerosas ocasiones a lo largo de la evolución del lenguaje, y Bjarne siempre optó por la generalización.

Pero las capacidades de esta primeriza versión de C++, las cuales serían recogidas en un informe publicado en la revista *Software: Practice and Experience* [14], no se reducían únicamente a una mejor organización del código; también disponía de las ventajas computacionales de C. Esto era un punto fundamental para Bjarne, siendo una de sus principales preocupaciones que el nuevo lenguaje perdiese el rendimiento de C o alguna de las capacidades de bajo nivel de dicho lenguaje, como las operaciones con bits o la elección del tamaño de los enteros.

A pesar de que Bjarne había logrado un gran resultado con C con Clases, el propio autor era consciente de que el lenguaje tenía aún algunas carencias y limitaciones. Principalmente, C con Clases podía resolver con elegancia pequeños problemas, pero era difícil de escalar para programas de mayor tamaño. Esto llevaría a Bjarne a evolucionar aún más su lenguaje.

2.2.3. La evolución de C con Clases a C++ (1982-1991)

En 1982, a pesar del éxito que C con Clases había alcanzado, Bjarne era consciente de que el lenguaje había llegado a su límite: era suficiente para satisfacer a su autor y un notable grupo de usuarios, pero insuficiente para que las organizaciones pagasen para su soporte y desarrollo. En este punto, Bjarne pensó en dos opciones: (1) abandonar por completo el proyecto de C con Clases, o bien (2) dar un paso más y evolucionar C con Clases a un lenguaje con un público mayor que incluyese organizaciones dispuestas a pagar por su soporte. Bjarne acabaría optando por la segunda.

El nuevo lenguaje, sucesor de C con Clases, fue denominado en un principio C84 para evitar que los usuarios lo confundiesen con C. Sin embargo, este nombre nunca convenció a su autor, que veía C84 como un nombre impuesto por una institución. En consecuencia, Bjarne continuó explorando nuevas opciones y pocos meses después terminó decantándose por el ahora mundialmente conocido C++. El sufijo «++» gustó mucho al autor del lenguaje, en primer lugar, porque le resultaba más atractivo que añadir un simple número; y, en segundo lugar, por el concepto que denotaba dicho sufijo: en programación, el operador ++ hace referencia al siguiente elemento, al sucesor. El nombre de C++ aparecería por primera vez en [15].

C++ incorporaba importantes novedades respecto a C con Clases. Los ejemplos más destacados serían las funciones virtuales, la sobrecarga de operadores, las referencias, las constantes, nuevos operadores para el manejo dinámico de la memoria o una comprobación de tipos más cómoda para el programador. Todas estas capacidades fueron incluidas en la primera versión de C++, la versión 1.0, publicada en 1985 y recogida en la primera edición de «*The C++ Programming Language*» [16]. En este libro Bjarne incluyó también a todas las personas que habían contribuido en el desarrollo de la primera versión, como Tom Cargill, Jim Coplien o Stu Feldman, entre otros.

C++ 1.1 y C++ 1.2 fueron publicadas poco tiempo después, en junio de 1986 y enero de 1987, respectivamente. Estas nuevas versiones solventaban pequeños fallos de la versión anterior e incorporaban algunas novedades como los punteros de memoria a los miembros de una clase o los miembros protegidos. La versión de C++ 1.3 también estaba en camino y se esperaba que fuese publicada a principios de 1988, pero algunos errores e indecisiones provocaron que dicha versión nunca llegase al público y se optase por un cambio mayor.

En 1989, tras la supresión de la versión 1.3, C++ 2.0 fue publicado. La documentación de la versión incluía una publicación de Bjarne [17] donde se resumían las nuevas características del lenguaje, así como agradecimientos a las personas que habían continuado en ese proceso de evolución. El objetivo principal de esta versión siempre fue aumentar la estabilidad del lenguaje e integrar de una mejor forma las capacidades que habían sido añadidas en versiones anteriores, lo que hizo C++ 2.0, a pesar de ser una versión considerablemente mejor que la 1.2, no cumplierse con todas las expectativas en el aspecto de la innovación.

A pesar de que no todas las ideas de Bjarne pudieron ser añadidas en esta versión, C++ 2.0 incorporaba la herencia múltiple (característica clave de la versión), funciones miembro estáticas y constantes, clases abstractas o mejoras en el manejo de memoria dinámico por parte del usuario. Estas novedades contribuían directamente al crecimiento que estaba viviendo el lenguaje, el cual alcanzó en 1991 un número estimado de usuarios cercano a los 400.000, y que veía como esta cifra se duplicaba aproximadamente cada siete meses y medio.

2.2.4. Estandarización (1991-1997)

En 1991 C++ se preparaba para convertirse en un lenguaje dominante e internacional. Coincidiendo con la salida de C++ 3.0, Bjarne publicó ese mismo año la segunda versión de «*The C++ Programming Language*» [18]. El lenguaje contaba ya con cinco compañías desarrollando compiladores (AT&T, GNU, Zortech, Borland y Oregon), a las cuales se sumarían otras tres un año más tarde (Windows, IBM y DEC). La idea de estandarizar el lenguaje, que ya había surgido unos años atrás, cobró más fuerza que nunca auspiciada por la Organización Internacional de Normalización, la ISO.

El nuevo comité de estandarización sería el que dirigiese la evolución del lenguaje, y desde entonces se convertiría en el punto central de la comunidad de C++. Dicho comité estaba formado principalmente por representantes de empresas y de naciones, pero contaba también con personalidades individuales interesadas en el lenguaje. El comité, que se reunía tres veces al año para decidir los cambios de C++, usaba un sistema de votaciones totalmente democrático que buscaba el consenso, aunque en algunas decisiones más controvertidas se podían desarrollar momentos de cierta tensión.

Hasta la votación final en 1997 del primer estándar del lenguaje, C++98 [19], el comité de estandarización aprobó importantes cambios en el lenguaje, como la incorporación del tipo booleano *bool* o la aceptación de los *namespaces*. Sin embargo, la novedad más destacada de todas fue la incorporación de la librería estándar, denominada STL (*Standard Template Library*) por su propio autor, Alex Stepanov [20].

La visión de Alex Stepanov sobre los contenedores y sus iteradores y algoritmos cumplía con todos los criterios y propiedades que Bjarne consideraba fundamentales para dichos contenedores (estos criterios pueden ser encontrados en la tercera edición de «*The C++ Programming Language*» [21]). No obstante, la STL era una idea considerablemente

revolucionaria y, por el contrario, el comité de estandarización tendía a ser conservador, lo que dificultaría la entrada de la STL en el estándar ISO. Finalmente, tras una reducción considerablemente en el tamaño de la STL, la librería acabaría siendo incorporada al estándar en la votación del comité en octubre de 1994 en Waterloo. Dicha librería tendría un impacto inmenso en el desarrollo de C++ y en la programación en general.

2.2.5. Periodo de mantenimiento (1997-2003)

Tras la aprobación del estándar C++98, el comité decidió entrar en un etapa de «mantenimiento», principalmente porque algunos miembros de ese comité llevaban casi una década trabajando en lo mismo y preferían probar cosas nuevas. Además, la comunidad de usuarios y desarrolladores iba bastante por detrás en cuanto a entender e implementar las novedades del lenguaje.

El único objetivo de este periodo de mantenimiento era encontrar estabilidad. Pequeñas correcciones de errores, clarificaciones en el texto del estándar y resolución de contradicciones fueron las tareas principales del comité durante estos años. La innovación técnica se evitaba. No obstante, C++ sí vería un importante cambio respecto a la comunicación con su comunidad.

En 2003, todos los cambios menores realizados desde 1997 fueron recogidos en una versión revisada del estándar, el «Technical Corrigendum 1», dando lugar a C++03 [22]. Gracias a los esfuerzos del comité nacional inglés, se consiguió que la editorial Wiley publicase una versión impresa del estándar revisado [23]. Antes de esta publicación, la única forma que había de conseguir el estándar era pagar cientos de dólares por una copia. A partir de ese momento, cualquier usuario podía adquirirlo en una librería, algo que no había ocurrido con ningún otro estándar ISO.

Este periodo de mantenimiento supuso un respiro tanto para el comité como para el lenguaje. Los usuarios y desarrolladores habían tenido tiempo para adaptarse a las nuevas características de C++, y los miembros del comité estaban preparados para plantear e implementar novedosas ideas.

2.2.6. Tiempos difíciles (2003-2011)

Desde su revisión en 2003, el estándar de C++ continuaría con su proceso de evolución. Bjarne Stroustrup, al igual que muchos miembros del comité ISO C++, confiaba en que

el próximo estándar estuviese listo para 2008 o 2009, repleto de nuevas características. Sin embargo, el nuevo estándar acabaría retrasándose hasta 2011, y los años entre medias no fueron sencillos.

En 2006, por primera vez en toda su historia, C++ experimentó una disminución del número de usuarios (Bjarne estimó dicha disminución en un 4%). C++ ya no era un lenguaje novedoso, y parecía estar empezando a quedarse atrás respecto a otros como Java o C#. Estos lenguajes disponían de herramientas como el recolector de basura que hacían más útil al programador novato y menos necesario al programador avanzado. Además, las revistas de software y tecnología se habían comenzado a sumar a la moda de que el lenguaje estaba muriendo, empeorando la situación.

Incluso en las universidades (donde C++ había sido adoptado para la enseñanza de la programación), el lenguaje estaba siendo sustituido por Java. El propio Bjarne, quien alrededor de 2005 había aceptado enseñar a programar a universitarios novatos, descubrió la disminución en la calidad que la enseñanza de C++ había sufrido. Fue tal su decepción con los libros que decían enseñar C++, que en 2008 acabaría publicando su propio libro destinado a este fin [24].

A pesar de todos estos inconvenientes y de que el lenguaje había dejado de ser elegido como primera opción en los proyectos más novedosos, C++ estaba altamente extendido por una gran variedad de sectores (telecomunicación, videojuegos, animación, finanzas, etc.). Además, la tendencia de la tecnología, que estaba empezando a encontrarse con las limitaciones del hardware, favorecía a un lenguaje como C++ capaz de exprimir al máximo dicho hardware, a la vez que exponía las debilidades de lenguajes más ineficientes, antes desapercibidas. Estos puntos, sumados al desarrollo de los dispositivos portátiles, donde la eficiencia energética era clave, permitieron al lenguaje recuperarse y llegar con fuerza a 2011, año en el que se publicaría el siguiente estándar.

2.2.7. C++ en la década de 2010 (2011-2020)

La publicación en 2011 de C++11 [25] fue un gran evento para el lenguaje: los desarrolladores pronto se adaptaron a las nuevas implementaciones, la comunidad crecía a gran velocidad con nuevos usuarios y el comité había recuperado el entusiasmo. Para Bjarne y para muchos otros miembros de la comunidad, C++11 no parecía una nueva versión del estándar, sino un lenguaje nuevo por completo.

Unos años más tarde, en 2014, siguiendo la premisa de alternar publicaciones mayores del estándar con revisiones menores, el comité ISO C++ publicaría C++14 [26], una nueva revisión del estándar destinada a corregir los fallos de C++11. El estándar de 2014 cumplió su función de completar al de 2011, incorporó algunas características destinadas a necesidades muy específicas y permitió al comité prepararse para la revisión del estándar de 2017.

El estándar de 2017 debía ser una publicación mayor, es decir, C++17 [27] debía contar con características nuevas capaces de cambiar el rumbo de la programación en C++. Sin embargo, esta revisión del estándar, si bien es cierto que incluía una considerable cantidad de pequeñas novedades, no supuso un gran cambio en el lenguaje, puesto que las mayores novedades pensadas para esta revisión no estaban completas y tuvieron que esperar hasta 2020.

En febrero de 2020, a pesar del tamaño del comité ISO y los diferentes puntos de vista de sus miembros, el nuevo estándar fue votado con unanimidad sin un solo voto en contra. La versión ISO oficial de C++20 [28], la cual contenía las grandes novedades que no pudieron ser incluidas en C++17, sería publicada a finales de ese mismo año. De esta forma, C++20 se convertiría en el estándar actual del lenguaje.

2.2.8. C++ en la actualidad (2020-¿?)

En la actualidad, C++ suele pasar desapercibido para el usuario final. Sin embargo, debajo de la superficie, el lenguaje se encuentra presente prácticamente en cualquier campo: telecomunicaciones (AT&T), finanzas (Morgan Stanley), dispositivos móviles (Huawei), videojuegos (Unreal Engine), microelectrónica (AMD), animación (Disney), bases de datos (Oracle), buscadores web (Google), aplicaciones web (Amazon), inteligencia artificial (Tensor Flow), investigación aeroespacial (Space X), física práctica (CERN), medicina (Siemens), desarrollo de software (Java Virtual Machine), automoción (Tesla), etc. Esta (interminable) lista manifiesta la amplia y profunda extensión de C++ en la sociedad actual.

No obstante, el futuro de C++ es, como poco, incierto. Una vez más, el lenguaje está amenazado de ser reemplazado por otros más novedosos (como Julia [29]) y en los foros cada vez se habla más de este hecho. Es responsabilidad del comité ISO, el cual ya prepara

un nuevo estándar para 2023, así como de toda la comunidad de C++, que el lenguaje mantenga su posición dominante y no acabe siendo relegado.

2.3. Guías y convenciones de C++

El lenguaje de programación C++ es un lenguaje flexible, diseñado para cubrir el dominio de aplicaciones más amplio posible. Como consecuencia de esta extensa capacidad, diferentes instituciones han definido una serie de guías y convenciones propias que complementan el estándar del lenguaje. En esta sección se pretende mencionar brevemente aquellas que han sido de mayor relevancia para este proyecto.

2.3.1. C++ *Core Guidelines*

El conjunto de normas *C++ Core Guidelines* [30], definido por el autor del lenguaje, Bjarne Stroustrup, y uno de los expertos más prominentes en C++, Herb Sutter, tiene por objetivo la correcta aplicación del estándar ISO C++. Esta guía se centra principalmente en los asuntos de alto nivel, como el manejo de la memoria o el uso de la concurrencia, más que los detalles de bajo nivel, como las reglas para nombrar funciones y variables (*naming conventions*) o los estilos de sangrías (*indentation styles*).

2.3.2. *CERT Secure Coding Guidelines*

Las *CERT Secure Coding Guidelines* son las guías para el desarrollo de código en diferentes lenguajes que proporciona el *Software Engineering Institute* (SEI) de la *Carnegie Mellon University*. La guía CERT para C++ [31] incluye una serie de recomendaciones y reglas que aseguran un código protegido frente a vulnerabilidades de seguridad.

2.3.3. *High Integrity C++ Coding Standard*

El estándar *High Integrity C++* [32], también conocido como *HIC++*, es una guía de código desarrollada por la compañía Perforce. Esta guía incluye reglas y ejemplos sobre buenas prácticas que permiten un desarrollo de código de alta calidad.

2.4. Benchmarks

Un *benchmark* es un punto de referencia, un valor para comparar. En el ámbito de la ingeniería informática, los *benchmarks* son aplicaciones cuyo propósito es la evaluación de diferentes características, como el rendimiento o el consumo de recursos, tanto del software como del hardware. Un ejemplo de *benchmark* destinado a este fin es el conjunto de aplicaciones PARSEC, el cual será analizado en este estudio.

2.4.1. Benchmark PARSEC

El *benchmark* PARSEC (Repositorio de Aplicaciones de Princeton para Ordenadores con Memoria Compartida) es la solución que la Universidad de Princeton desarrolló para afrontar la evaluación de los multiprocesadores actuales con cargas más representativas de los programas emergentes [33].

A pesar de que el proyecto naciese a finales de 2005, la primera versión pública del *benchmark* PARSEC no aparecería hasta enero de 2008, contando con 12 programas multihilo. Los investigadores pronto empezarían a utilizar el *benchmark* en sus estudios, ganando este una gran importancia dentro de la comunidad. Desde entonces, nuevas versiones del *benchmark* han sido publicadas, con mejores cargas de trabajo y nuevas aplicaciones, siendo la última la 3.0, versión con la que se ha trabajado en este proyecto.

Tal y como se detalla en [34], el núcleo de PARSEC 3.0 son 13 aplicaciones multihilo destinadas a evaluar el rendimiento de ordenadores multinúcleo desde diferentes ángulos, intentando que el *benchmark* sea lo más representativo posible:

- **Blackscholes:** *benchmark* desarrollado por Intel RMS (Reconocimiento, Minado y Síntesis) que calcula los precios de opciones europeas en el ámbito financiero a través de la ecuación diferencial parcial Black-Scholes [35].

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS \frac{\partial V}{\partial S} - rV = 0$$

Fig. 2.1. Ecuación diferencial parcial Black-Scholes [34]

- **Bodytrack:** aplicación de visión por ordenador que utiliza la emergente inteligencia artificial para el seguimiento de un cuerpo humano 3D en una

secuencia de imágenes [36]. También está incluido en el conjunto de aplicaciones de Intel RMS.

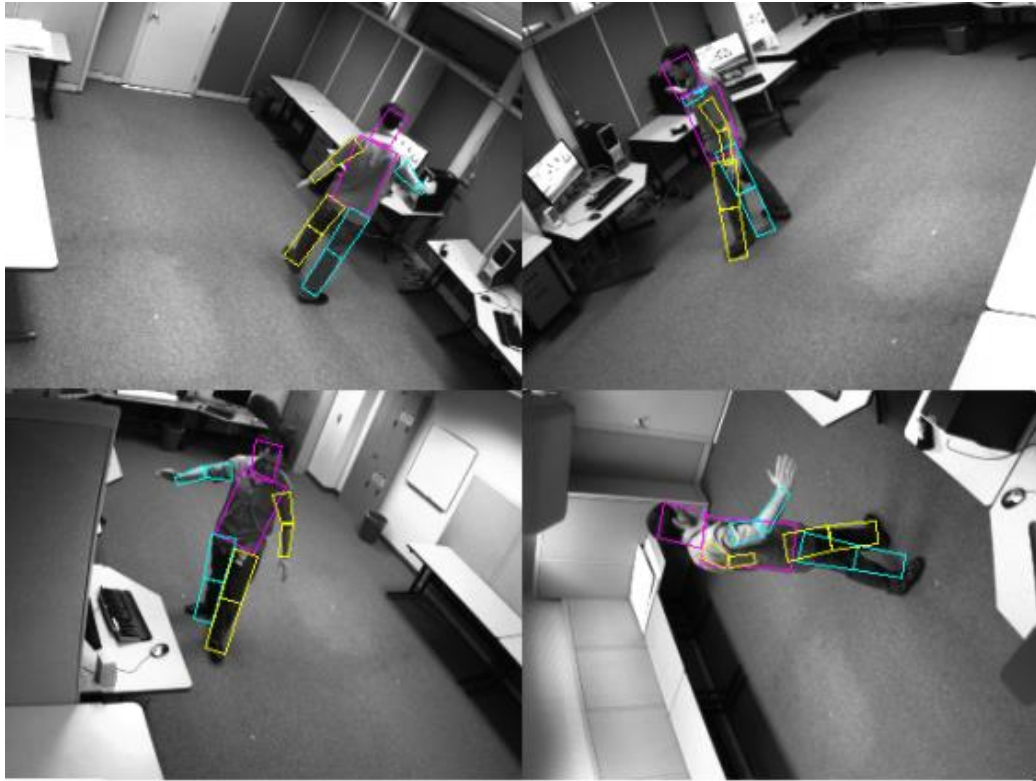


Fig. 2.2. Output de la aplicación Bodytrack [34]

- **Canneal:** kernel desarrollado por la Universidad de Princeton que utiliza el algoritmo de recocido simulado con caché-trascendente (del inglés, *cache-aware simulated annealing*) para la minimización del coste de enrutamiento en el diseño de chips [37].
- **Dedup:** kernel desarrollado también por la Universidad de Princeton destinado a la compresión de datos utilizando un modelo que combina la compresión local con la compresión global, denominado deduplicación (del inglés, *deduplication*) [38].
- **Facesim:** aplicación incluida en el conjunto de Intel RMS desarrollada inicialmente por la Universidad de Stanford. A partir del modelado de una cara humana y de una secuencia de activaciones musculares, calcula una animación realista de dicha cara [39].



Fig. 2.3. Dos fotogramas del *output* de la aplicación Facesim [34]

- **Ferret:** aplicación basada en el conjunto de herramientas Ferret, desarrollado por la Universidad de Princeton. Ferret es un motor de búsqueda de similitud en contenido en elementos multimedia (imágenes, audio, vídeo, etc.) [40], aunque para PARSEC dicha herramienta ha sido configurada para la búsqueda de imágenes.

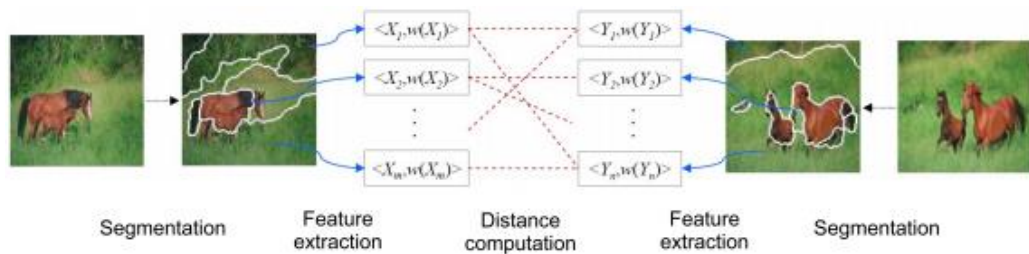


Fig. 2.4. Algoritmo del motor de búsqueda Ferret [34]

- **Fluidanimate:** aplicación de Intel RMS que emplea una extensión del método SPH (Hidrodinámica de Partículas Suavizadas) para simular un fluido incompresible con fines de animación interactiva [41], como los efectos de partículas en videojuegos.

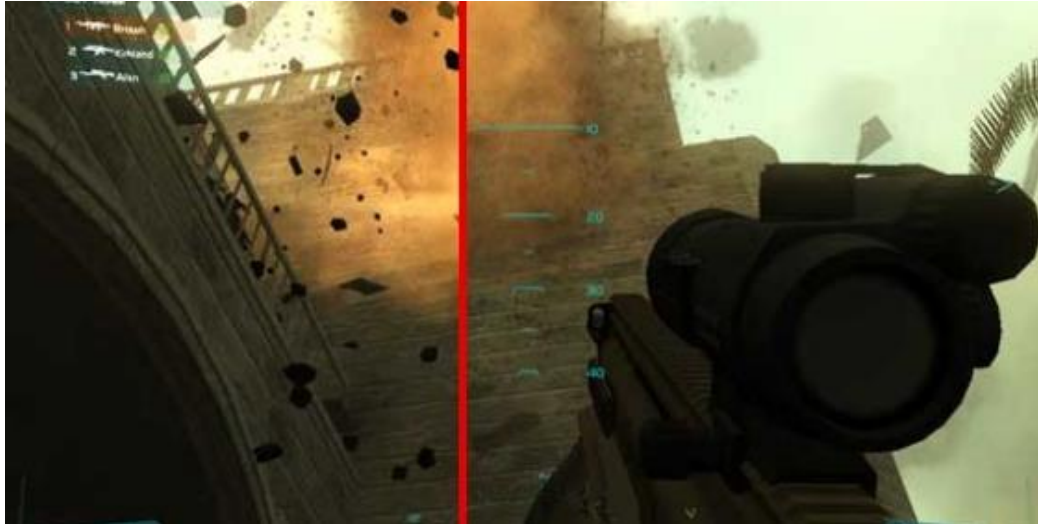


Fig. 2.5. Captura de pantalla de un videojuego con efectos de partículas (izq.) y sin ellas (der.) [34]

- **Freqmine:** aplicación de Intel RMS originalmente desarrollada por la Universidad Concordia que utiliza una versión basada en arreglos del algoritmo FP-Crecimiento (Crecimiento de Patrón Frecuente, del inglés *Frequent Pattern Growth* o *FP-Growth*) para el FIMI (Minado de Conjuntos de Elementos Frecuentes, del inglés *Frequent Itemset Mining*) [42].
- **Raytrace:** aplicación incluida en Intel RMS dedicada a la renderización de una escena 3D animada utilizando la técnica de trazado de rayos (del inglés, *ray tracing*) [43], altamente utilizada en el sector de los videojuegos. En PARSEC esta técnica está más enfocada en la velocidad que en el realismo.



Fig. 2.6. Demostración de la aplicación Raytrace sobre un videojuego [34]

- **Streamcluster:** kernel destinado a resolver el problema del agrupamiento en línea (del inglés, *online clustering*). Este problema consiste en, para una secuencia de puntos de entrada, encontrar un número determinado de medias de tal forma que cada punto es asignado a su media más cercana [44].
- **Swaptions:** aplicación de Intel RMS que hace uso del *framework* HJM (Heath-Jarrow-Morton) y de la simulación MC (Monte Carlo) para establecer el precio de una serie de cambios financieros [45].
- **Vips:** aplicación que tiene su fundamento en el Sistema de Procesado de Imagen VASARI (del inglés, VASARI Image Processing System), también conocido como VIPS [46]. Este sistema desarrollado a través de proyectos financiados por la Unión Europea utiliza una serie de transformaciones que se aplican de forma paralela en subregiones de la imagen, posibilitando el procesado multihilo de dicha imagen.



Fig. 2.7. *Input* de la aplicación Vips en PARSEC [34]

- **X264:** aplicación basada en la codificación de vídeo H.264/AVC (Codificación de Vídeo Avanzada) [47]. Esta codificación de vídeo proporciona mayor calidad con una tasa de bits menor que otros estándares, a expensas de tiempos de codificación y decodificación superiores.



Fig. 2.8. Fotograma de entrada de la aplicación x264 en PARSEC [34]

Además, esta versión de PARSEC incluye también tres nuevas aplicaciones de red cliente-servidor, dirigidas a evaluar la comunicación a través de una pila TCP/IP a nivel usuario en multiprocesadores [48]. Basadas en tres de las aplicaciones base del *benchmark* PARSEC, las aplicaciones de red con las que cuenta este *benchmark* son: Netdedup, Netferret y Netstreamcluster.

Por último, PARSEC 3.0 incluye dos *benchmarks* adicionales, destinados a complementarlo: SPLASH-2 [49], un *benchmark* dirigido a la evaluación de multiprocesadores focalizado en el área de la computación de altas prestaciones, y SPLASH-2x [50], una novedosa versión de SPLASH-2 con entradas escaladas a diferentes tamaños.

3. ESTUDIO DE LA CALIDAD

En este tercer capítulo se procede a analizar y mejorar de forma iterativa la calidad del código fuente de las aplicaciones que componen el *benchmark* PARSEC versión 3.0. En primer lugar, se llevará a cabo una modernización del código, solventado primero los posibles errores que impidan su compilación y, a continuación, detectando, analizando y solucionando los avisos generados al compilar dichas aplicaciones. Posteriormente se realizará un análisis dinámico y un análisis estático del código con el objetivo de detectar y solventar el mayor número posible de fallos de calidad.

Para llevar a cabo este estudio se ha decidido compilar el código con el estándar C++17 ya que, a pesar de que C++20 ya ha sido publicado en el momento en el que se realiza este estudio, se considera que esta penúltima versión del estándar del lenguaje es más estable y compatible. Como compilador se va a emplear la versión 10.2 de GCC (*GNU Compiler Collection*). Por último, a pesar de que SPLASH-2 y SPLASH-2X forman parte de PARSEC 3.0, se ha decidido excluir ambos *benchmarks* de este estudio debido a que están escritos mayoritariamente en C y no en C++. Otras aplicaciones de PARSEC 3.0 serán también apartadas a lo largo del estudio por este y otros motivos. A pesar de estas modificaciones, se considera que la amplitud y la profundidad de este proyecto son adecuadas para cumplir con los objetivos del mismo.

3.1. Modernización del código fuente

3.1.1. Detección y solución de los errores de compilación

La primera fase en esta modernización del código fuente consiste en conseguir compilar las aplicaciones incluidas en PARSEC. Para ello es necesario solventar los diferentes errores generados durante la compilación de cada una de estas aplicaciones.

El primer error que aparece al compilar el código se debe a la presencia de excepciones dinámicas en la aplicación Bodytrack, no permitidas en C++17. La Fig. 3.1 ilustra una declaración de este tipo. Este error se repite en los archivos Thread.h (línea 62), Thread.cpp (línea 51), Mutex.h (líneas 85, 86, 89, 91 y 93), Mutex.cpp (líneas 27, 63, 92, 128, 164), Condition.h (líneas 65, 66, 92, 128 y 164), Condition.cpp (líneas 28, 77, 114, 158 y 202), Barrier.h (líneas 63, 64 y 67), Barrier.cpp (líneas 28, 69 y 100), RWLock.h

(69, 70, 73, 75, 77, 79 y 81) y RWLock.cpp (líneas 22, 55, 80, 112, 144, 175, 206). Para solventar este error, basta con eliminar todas las declaraciones donde se lanzan excepciones dinámicas.

```
60. public:
61.     Thread(Runnable &) throw(ThreadCreationException);
62.
63.     //Wait until Thread object has finished
64.     void Join();
65. };
66.
```

Fig. 3.1. Excepción dinámica (archivo Thread.h)

El segundo error que impide la compilación de PARSEC se encuentra en una de las librerías que incluye el *benchmark*, en concreto la librería SSL. En el código fuente de esta librería hay una extensa serie de archivos con la extensión *.pod*, los cuales contienen numerosas expresiones del tipo «=*item n*» que deben ser reemplazadas por «=*item C<n>*». La Fig. 3.2 y la Fig. 3.3 muestran un ejemplo de este error y su solución, respectivamente. Este error se repite en los archivos *smime.pod* (líneas 268, 272, 276, 280, 285, 289), *SSL_COMP_add_compression_method.pod* (líneas 56, 60), *SSL_CTX_add_session.pod* (líneas 55, 61), *SSL_CTX_load_verify_locations.pod* (líneas 103, 109), *SSL_CTX_set_client_CA_list.pod* (líneas 69, 73), *SSL_CTX_set_session_id_context.pod* (líneas 67, 73), *SSL_CTX_set_ssl_version.pod* (líneas 45, 49), *SSL_accept.pod* (líneas 47, 52), *SSL_clear.pod* (líneas 53, 58), *SSL_connect.pod* (líneas 44, 49), *SSL_do_handshake.pod* (líneas 48, 53), *SSL_read.pod* (línea 89), *SSL_session_reused.pod* (líneas 30, 34), *SSL_set_fd.pod* (líneas 38, 42), *SSL_set_session.pod* (líneas 40, 44), *SSL_shutdown.pod* (líneas 95, 100, 107) y *SSL_write.pod* (línea 82).

```
264. =head1 EXIT CODES
265.
266. =over 4
267.
268. =item 0
269.
270. the operation was completely successfully.
271.
272. =item 1
273.
274. an error occurred parsing the command options.
275.
276. =item 2
277.
```

Fig. 3.2. Error en archivos *.pod* (archivo *smime.pod*)


```

264. =head1 EXIT CODES
265.
266. =over 4
267.
268. =item C<0>
269.
270. the operation was completely successfully.
271.
272. =item C<1>
273.
274. an error occurred parsing the command options.
275.
276. =item C<2>
277.

```

Fig. 3.3. Solución al error en archivos *.pod* (archivo smime.pod)

El tercer error aparece al compilar el kernel Dedup y su variante, Netdedup, debido a una serie de definiciones múltiples que el compilador no es capaz de diferenciar durante el tiempo de enlace. Para solventar este error es necesario añadir la palabra clave *extern* delante de las declaraciones de las variables afectadas, como puede verse en la Fig. 3.4. Dichas variables afectadas se encuentran en los archivos decoder.c (líneas 39 y 42) y encoder.c (líneas 65 y 68) de ambas aplicaciones.

```

38.
39. //The configuration block defined in main
40. extern config_t * conf;
41.
42. //Hash table data structure & utility functions
43. extern struct hashtable *cache;
44.

```

Fig. 3.4. Declaraciones con la palabra clave *extern* (archivo decoder.c)

El cuarto error de compilación se encuentra en la aplicación Facesim, debido a una serie de comparaciones no permitidas. Este tipo de comparaciones requieren una conversión explícita, ya que el compilador no acepta conversiones implícitas. Un ejemplo de este error y su solución pueden verse en la Fig. 3.5 y en la Fig. 3.6, respectivamente. Dicho error se repite en los archivos FILE_UTILITIES.h (líneas 73 y 83) y FILE_UTILITIES.cpp (línea 101).

```

98.
99. bool Directory_Exists (const std::string& dirname)
100. {
101.     return std::ifstream (dirname.c_str()) != 0;
102. }
103.

```

Fig. 3.5. Comparación con conversión implícita (archivo FILE_UTILITIES.cpp)

```

98.
99.  bool Directory_Exists (const std::string& dirname)
100. {
101.     return static_cast<bool>(std::ifstream (dirname.c_str()));
102. }
103.

```

Fig. 3.6. Conversión explícita (archivo FILE_UTILITIES.cpp)

El quinto error que imposibilita la compilación del *benchmark* PARSEC se debe a un conflicto entre la declaración de la variable `__mbstate_t` en las librerías del sistema y la declaración de la misma variable en la librería Utpcpip de PARSEC. Para solucionar este error, se debe comentar la declaración de la variable en el archivo `bsd__types.h`, mostrada en la Fig. 3.7, entre las líneas 100 y 106.

```

95.  /*
96.   * mbstate_t is an opaque object to keep conversion state during multibyte
97.   * stream conversions.
98.   */
99.  #ifndef __mbstate_t_defined
100. # define __mbstate_t_defined 1
101. typedef union {
102.     char      __mbstate8[128];
103.     __int64_t  __mbstatel; /* for alignment */
104. } __mbstate_t;
105. #endif

```

Fig. 3.7. Declaración de la variable `__mbstate_t` (archivo `bsd__types.h`)

El sexto error de compilación se debe a una constante (*HUGE*) presente en los archivos de las aplicaciones Ferret y Netferret. Dicha constante fue reemplazada por *DBL_MAX*, pero en determinadas líneas del código fuente la constante no ha sido actualizada, como puede verse en Fig. 3.8. Para solucionarlo se debe reemplazar la constante *HUGE* por la constante *DBL_MAX* en los archivos `ferret-pthreads.c` (línea 390), `ferret-serial.c` (línea 223), `ferret-tbb.cpp` (línea 407), `LSH_query.c` (líneas 200, 246, 361) y `LSH_query_batch.c` (líneas 142, 323, 325) de la aplicación Ferret y en los archivos `ferret-pthreads.c` (línea 477), `LSH_query.c` (líneas 200, 246, 361) y `LSH_query_batch.c` (líneas 142, 323, 325) de su variante, Netferret.

```

320. for (i = 0; i < N; i++)
321. {
322.     int j;
323.     TOPK_INIT(topk[i], dist, K, HUGE);
324.     for (j = 0; j < T; j++)
325.         TOPK_INIT(ptopk[i][j], dist, K, HUGE);
326. }
327.

```

Fig. 3.8. Constante *HUGE* (archivo `LSH_query.c`)

El último error de compilación encontrado consiste en un problema de soporte con *PIE* en x64, la última aplicación de PARSEC. Para solventarlo es necesario añadir las opciones *-fno-pie* y *-no-pie* en la línea 38 del archivo `gcc-pthreads.bldconf` del paquete x264, tal y como se muestra en la Fig. 3.9.

```
37.  
38. # Arguments to pass to the configure script, if it exists  
39. build_conf="--enable-pthread --extra-asflags=\"${ASFLAGS}\" --extra-  
cflags=\"${CFLAGS} -fno-pie -no-pie\" --extra-ldflags=\"${LDFLAGS} -fno-pie -no-  
pie ${LIBS}\"  
40.
```

Fig. 3.9. Opciones *-fno-pie* y *-no-pie* (archivo `gcc-pthreads.bldconf`)

Después de lograr la compilación sin errores de todas las aplicaciones, se ha procedido a probar la ejecución de cada una de ellas, con el objetivo de comprobar que ninguna de las modificaciones anteriores haya afectado al funcionamiento del código. Todas las aplicaciones demuestran un correcto funcionamiento a excepción de Raytrace, cuya ejecución no termina. Se ha probado la compilación de dicha aplicación con versiones más antiguas tanto de GCC como del estándar C++ (en concreto, GCC 5.0, 4.9 y 4.8 y C++11), pero sin obtener resultados. En consecuencia, se ha decidido eliminar la aplicación Raytrace de este estudio de calidad.

3.1.2. Recopilación, análisis y solución de los avisos de compilación

Una vez conseguida una versión base del código fuente capaz de compilar sin errores y funcional, se procede a estudiar y solventar los avisos que el compilador genera al procesar el código.

La recopilación completa de todos los avisos puede consultarse en el Anexo A.1. Tras esta recopilación, siguiendo la premisa empleada desde el inicio de este estudio, se ha decidido eliminar los paquetes que contenían fallos en archivos escritos en C, incluyendo las aplicaciones Ferret, Netdedup, Netferret, Vips y x264. Las aplicaciones Dedup y Facesim también han sido apartadas de este estudio, ya que no presentan ningún aviso de compilación. La relación entre las aplicaciones que permanecen en el estudio (Blackscholes, Bodytrack, Canneal, Fluidanimate, Freqmine, Netstreamcluster, Streamcluster y Swaptions) y los avisos de compilación presentes en dichas aplicaciones puede verse en la Tabla 3.1.

Tabla 3.1. Aplicaciones de PARSEC con avisos de compilación

Avisos de compilación	Blackscholes	Bodytrack	Canneal	Fluidanimate	Frequine	Netstreamcluster	Streamcluster	Swaptions
Sufijo inválido en literal; C++17 requiere un espacio entre literal y cadena macro [-Wliteral-suffix]	X	X	X	X	X	X	X	X
El formato '%{FORMATO}' espera un argumento del tipo '{TIPO1}', pero el argumento {n} tiene tipo '{TIPO2}' [-Wformat=]	X					X	X	
Se sugieren llaves explícitas para evitar ambiguos 'else' [-Wdangling-else]		X					X	
Ignorando el valor de retorno de '{FUNCIÓN}', declarado con el atributo warn_unused_result [-Wunused-result]		X						
Throw siempre llamará a terminate() [-Wterminate]		X						
'{Variable}' se inicializará después [-Wreorder]		X						
'{Variable}' no se declaró en este ámbito, y no se encontraron declaraciones mediante la búsqueda dependiente de argumentos en el punto de instanciación [-fpermissive]		X						
Capturando el tipo polimórfico '{TIPO}' por valor [-Wcatch-value=]		X						
ISO C++17 no permite el especificador de clase de almacenamiento 'register' [-Wregister]			X		X			X
No hay declaración de retorno en la función que devuelve non-void [-Wreturn-type]			X					
Bandera ' ' utilizada con el formato gnu_printf '%c' [-Wformat=]						X		
El formato '%c' espera un argumento 'int' coincidente [-Wformat=]						X		

A continuación, se presenta el análisis de los avisos de compilación detectados en las aplicaciones mantenidas en el estudio. En dicho análisis se expondrá, para cada uno de los avisos recopilados, el mensaje (traducido) del aviso, un fragmento de código que ejemplifique el fallo que provoca el aviso, las aplicaciones que presentan el fallo (incluyendo el número de veces que se repite en cada aplicación), la descripción del aviso, su impacto en la calidad del código y su posible solución:

- **Aviso 01:**

- Mensaje del aviso: «Sufijo inválido en literal; C++17 requiere un espacio entre literal y cadena macro [-Wliteral-suffix]».

- Fragmento de código:

```
376. #define __PARSEC_XSTRING(x) __PARSEC_STRING(x)
377.     cout << "PARSEC Benchmark Suite Version " __PARSEC_XSTRING(PARSEC_
    C_VERSION) << endl << flush;
378. #else
```

Fig. 3.10. Fragmento de código del aviso 01 (main.cpp, Bodytrack)

- Aplicaciones que lo presentan: Blackscholes (1), Bodytrack (1), Canneal (1), Fluidanimate (1), Freqmine (1), Netstreamcluster (1), Streamcluster (1), Swaptions (1).
- Descripción: C++ permite la concatenación de un *string* literal y una cadena macro. No obstante, desde la introducción de los literales definidos por usuario en C++11, es necesario incluir un espacio entre el *string* literal y la cadena macro para diferenciar este tipo de concatenación de un literal definido por el usuario.
- Impacto en la calidad: este fallo afecta principalmente a la portabilidad del programa, ya que, como se ha explicado en el punto anterior, este tipo de declaraciones no son válidas en las versiones más recientes del estándar del lenguaje. También afecta a la mantenibilidad del código, ya que puede dificultar su legibilidad y, por tanto, su capacidad de ser analizado, modificado y evaluado. El resto de propiedades de la calidad no se ven afectadas por este fallo.
- Solución: incluir un espacio entre la cadena literal y la cadena macro.

- **Aviso 02:**

- Mensaje del aviso: «El formato '%{FORMATO}' espera un argumento del tipo '{TIPO1}', pero el argumento {n} tiene tipo '{TIPO2}' [-Wformat=]».
- Fragmento de código:

```
1825.     fprintf(fp, "%u\n", centerIDs[i]);
```

Fig. 3.11. Fragmento de código del aviso 02 (streamcluster.cpp, Streamcluster)

- Aplicaciones que lo presentan: Blackscholes (1), Netstreamcluster (1), Streamcluster (2).
- Descripción: si el formato indicado no coincide con el tipo del argumento, el código puede presentar un comportamiento indefinido. En el fragmento

de código de ejemplo, el formato `%u` espera recibir un *unsigned int*, pero la variable `centerIDs[i]` es del tipo *long unsigned int*. En consecuencia, dependiendo de la arquitectura donde se ejecute el programa, el argumento puede contener valores que desborden la capacidad de tipo esperado por el formato, resultando en comportamiento indefinido.

- Impacto en la calidad: en primer lugar, este fallo repercute en la fiabilidad del código, debido al comportamiento indefinido que puede generarse como resultado de dicho fallo. La confusión entre los formatos requeridos y el tipo de los argumentos también afecta a la legibilidad del código y, por tanto, a su mantenimiento. El resto de propiedades de la calidad no se ven afectadas por este fallo.
- Solución: cambiar el formato requerido para que coincida con el tipo de los argumentos que recibe la función.

- **Aviso 03:**

- Mensaje del aviso: «Se sugieren llaves explícitas para evitar 'else' ambiguos [-Wdangling-else]».
- Fragmento de código:

```
201. if ( !ValidRect(img.Size(), x, y, width, height) )
202.     if(width == 0 || height == 0)
203.     { //point sub-image to top left of image with no size
204.         x = 0; y = 0;
205.         width = 0; height = 0;
206.         mStatus = FlexStsInvalidSubImage; //Error reported
207.     }
208.     else
209.         mStatus = FlexStsSubImageTruncated; //warning of truncated
210.         sub-image reported
```

Fig. 3.12. Fragmento de código del aviso 03 (FlexImage.cpp, Bodytrack)

- Aplicaciones que lo presentan: Bodytrack (2).
- Descripción: la falta de llaves explícitas en un condicional que contiene a otra declaración del mismo tipo puede confundir sobre a qué cláusula *if* corresponde la cláusula *else*.
- Impacto en la calidad: este fallo repercute considerablemente en la mantenibilidad del código ya que perjudica la legibilidad del mismo, dificultando su análisis y posibles modificaciones. El resto de propiedades de la calidad no se ven afectadas por este fallo.

- Solución: añadir llaves explícitas en todos los condicionales.
- **Aviso 04:**
- Mensaje del aviso: «Ignorando el valor de retorno de '{FUNCIÓN}', declarado con el atributo warn_unused_result [-Wunused-result]».
 - Fragmento de código:

```
112. fread(&bmfh, BFHSIZE, 1, in);           //read BMP header
113.     ConvertBmfh(&bmfh);
114. if(bmfh.bfType != 19778)                //check for valid BMP file
115.     return(false);
116. fread(&bmih, BIHSIZE, 1, in);           //read info header
117.     ConvertBmih(&bmih);
```

Fig. 3.13. Fragmento de código del aviso 04 (FlexIO.cpp, Bodytrack)

- Aplicaciones que lo presentan: Bodytrack (13).
 - Descripción: el valor de retorno de las funciones I/O debe ser comprobado para evaluar si la función se ha ejecutado correctamente. Ignorar este valor de retorno imposibilita la capacidad de manejar un posible fallo en dicha función y, en el peor de los casos, puede provocar la terminación del programa.
 - Impacto en la calidad: este fallo afecta de forma notable a la fiabilidad del programa, ya que este puede sufrir una terminación repentina puesto que no se comprueba el resultado de las operaciones *input/output*. Además, un error no manejado en este tipo de operaciones podría provocar la corrupción de los datos o archivos con los que opera el programa, amenazando la integridad de los mismos. El resto de propiedades de la calidad no se ven afectadas por este fallo.
 - Solución: comprobar el valor de retorno de las funciones I/O y manejar los posibles fallos de dichas funciones.
- **Aviso 05:**
- Mensaje del aviso: «Throw siempre llamará a terminate() [-Wterminate]».

- Fragmento de código:

```

63. Mutex::~Mutex() throw(MutexException) {
64. #if defined(HAVE_LIBPTHREAD)
65.     int rv;
66.
67.     rv = pthread_mutex_destroy(&m);
68.
69.     switch(rv) {
70.         case 0:
71.             break;
72.         case EBUSY:
73.         case EINVAL:
74.         {
75.             MutexDestroyException e;
76.             throw e;
77.             break;
78.         }
79.         default:
80.         {
81.             MutexUnknownException e;
82.             throw e;
83.             break;
84.         }
85.     }
86. #else //default: winthreads
87.     DeleteCriticalSection(&m);
88. #endif //HAVE_LIBPTHREAD
89. }

```

Fig. 3.14. Fragmento de código del aviso 05 (Mutex.cpp, Bodytrack)

- Aplicaciones que lo presentan: Bodytrack (8).
- Descripción: la declaración de una cláusula *throw* dentro del destructor de un objeto podría llevar a la terminación del programa. Esto ocurrirá durante el proceso conocido como *stack unwinding*, el cual tiene lugar cuando una función lanza una excepción y dicha función no cuenta con una declaración para manejarla. En consecuencia, el programa buscará de forma lineal dicha declaración en la pila de llamadas a funciones. Durante este proceso se eliminan todos los objetos creados desde la primera cláusula *try* por medio de sus correspondientes destructores. Si alguno de estos destructores lanza una nueva excepción, como el programa está tratando la primera excepción lanzada, la ejecución se detendrá.
- Impacto en la calidad: el impacto de este fallo en la fiabilidad del programa es crítico, ya que podría provocar su terminación de forma repentina. Además, también afecta a su usabilidad, disminuyendo la protección del programa frente a los errores de usuario, debido a que si el usuario provoca una excepción durante la ejecución del programa y se llama a uno de estos

destructores que lanzan excepciones, el programa no será capaz de manejar ambas excepciones. El resto de propiedades de la calidad no se ven afectadas por este fallo.

- Solución: la solución de este fallo no es trivial. No es suficiente la eliminación de las declaraciones que lanzan las excepciones dentro del destructor, ya que esto podría variar por completo el comportamiento del programa. Se necesitaría un cambio integral en el diseño y la estructura de la aplicación como, por ejemplo, recurrir al uso de las herramientas de concurrencia (hilos, bloqueos, etc.) que proporciona C++ en vez de utilizar librerías propias, ya que es en estas librerías de la aplicación Bodytrack donde se encuentran este tipo de fallos. No obstante, este cambio requeriría una importante inversión de tiempo y esfuerzo, por lo que se ha decidido dejar como materia de futuros trabajos.

- **Aviso 06:**

- Mensaje del aviso: «' {VARIABLE}' se inicializará después [-Wreorder]».
- Fragmento de código:

```
32. //Asynchronous Image loading object
33. class AsyncImageLoader : public threads::Runnable {
34.
35. protected:
36.     std::deque<ImageSet > mImageBuffer;    //image buffer
37.     std::deque<BinaryImageSet > mFGBuffer; //foreground image buffer
38.     unsigned int mNumCameras;    //number of cameras (images) per frame
39.     unsigned int mBufferSize;    //max number of images in the buffer
40.     std::string mPath;           //dataset path
41.
42.     threads::Mutex mDataLock;    //synchronization objects
43.     threads::Mutex mLock1, mLock2;
44.     threads::Condition mCondFull;
45.     threads::Condition mCondEmpty;
46.
47.     bool mFailed;                //image load failed flag
48.     unsigned int mCurrentFrame;  //current frame to be loaded
49.     unsigned int mNumFrames;     //total number of frames
50.
51. //load a given set of image and foreground files
52. void LoadSet(std::vector<std::string> &FGFiles, BinaryImageSet
    &FGImages, std::vector<std::string> &ImageFiles, ImageSet &images);
53.
54. public:
55.
56. AsyncImageLoader() : mCondFull(mLock1), mCondEmpty(mLock2),
    mCurrentFrame(0), mNumCameras(0), mBufferSize(16), mFailed(false)
57. {
58.     mImageBuffer.resize(0);
59. }
```

Fig. 3.15. Fragmento de código del aviso 06 (AsyncIO.h, Bodytrack)

- Aplicaciones que lo presentan: Bodytrack (4).
 - Descripción: la inicialización de los miembros de una clase no sigue el orden establecido en el código sino el orden definido por el estándar del lenguaje, el cual sigue el orden en el que dichos miembros han sido declarados en la clase. Es por ello por lo que el compilador avisa de que la inicialización de ciertas variables no coincide con lo definido en el código. Esto puede provocar comportamiento indefinido si un miembro A de la clase se inicializa con otro miembro B que, aunque en el código se inicialice aparentemente antes que A, en realidad se está inicializando más tarde. En consecuencia, el programa estará intentando inicializar un miembro con una variable que aún no ha sido inicializada, provocando comportamientos indefinidos.
 - Impacto en la calidad: ya que en ninguna de las ocasiones en las que se repite este fallo se inicializa un miembro de la clase a partir de otro, no existe la posibilidad de que se desencadene un comportamiento indefinido. No obstante, este fallo sí que afecta a la mantenibilidad del código, debido a que dificulta su legibilidad, modificación y evaluación al inicializar los miembros en un orden diferente al de declaración de los mismos. El resto de propiedades de la calidad no se ven afectadas por este fallo.
 - Solución: inicializar los miembros de la clase siguiendo el mismo orden que en su declaración.
- **Aviso 07:**
- Mensaje del aviso: «'{FUNCIÓN}' no se declaró en este ámbito, y no se encontraron declaraciones mediante la búsqueda dependiente de argumentos en el punto de instanciación [-fpermissive]».
 - Fragmento de código:

```

133. for(i = ticket; i < mNParticles && i < ticket + WORKUNIT_SIZE_NEWPARTIC
    LES; i++) {
134.     //add new particle for each entry in each bin distributed randomly
135.     mNewParticles[i] = mParticles[mIndex[i]];
136.     AddGaussianNoise(mNewParticles[i], mModel->StdDevs()
        [annealing_parameter], mRnd[i]);
137. }
138.

```

Fig. 3.16. Fragmento de código del aviso 07 (ParticleFilterPthread.h, Bodytrack)

- Aplicaciones que lo presentan: Bodytrack (1).
- Descripción: durante la compilación en C++ de una clase con plantilla, la búsqueda de los nombres de variables y funciones se divide en dos fases que permiten distinguir las funciones y variables que no dependen de la plantilla de las que sí, las cuales deben esperar al momento al que se instancia la clase para conocer el tipo de la plantilla. El problema es que la definición de una función miembro de la clase no será visible para el compilador hasta la instanciación de la clase. En consecuencia, si dicha función miembro no depende de la plantilla y en el código no se indica específicamente que la definición de la función está en la clase, el compilador buscará dicha definición en un ámbito mayor y no podrá encontrarla. Esto debería producir un error, pero sólo genera un aviso porque la compilación se está llevando a cabo con la opción *-fpermissive*, que indica al compilador que espere al momento de instanciación de la clase para buscar la definición de todas aquellas funciones para las que no ha encontrado una definición aún.
- Impacto en la calidad: este fallo afecta principalmente a la mantenibilidad del código, ya que, al igual que dificulta la búsqueda de la definición de la función para el compilador, también lo hace para otros desarrolladores que intenten modificar o evaluar el código. El resto de propiedades de la calidad no se ven afectadas por este fallo.
- Solución: añadir la declaración *this->* delante de la función para indicar al compilador que dicha función es un miembro de la clase y que debe buscar su definición tras la instanciación de dicha clase.

- **Aviso 08:**

- Mensaje del aviso: «Capturando el tipo polimórfico '{TIPO}' por valor [-Wcatch-value=]».
- Fragmento de código:

```

1. try {
2.     _tobj = &dynamic_cast<Stoppable &>(_tobj);
3. } catch(std::bad_cast e) {
4.     isStoppable = false;
5. }
6.

```

Fig. 3.17. Fragmento de código del aviso 08 (Thread.cpp, Bodytrack)

- Aplicaciones que lo presentan: Bodytrack (1).
 - Descripción: al capturar por valor una excepción, sólo la clase base indicada y sus propiedades serán capturadas, por lo que, aunque la excepción lanzada en la cláusula *try* sea otro tipo de excepción derivado, sólo se podrán utilizar las propiedades de la clase base indicada. Este proceso se conoce como *slicing*, y puede llegar a provocar corrupción de memoria y otros errores.
 - Impacto en la calidad: este fallo afecta a la fiabilidad del código debido a que al capturar por valor una excepción es posible la pérdida de algunas de las propiedades de la excepción realmente capturada. Esta pérdida de ciertas propiedades podría afectar también a la integridad funcional del programa. El resto de propiedades de la calidad no se ven afectadas por este fallo.
 - Solución: capturar por referencia usando *const&*. En el fragmento de código de ejemplo quedaría como *catch(std::bad_cast const&)*.
- **Aviso 09:**
- Mensaje del aviso: «ISO C++17 no permite el especificador de clase de almacenamiento 'register' [-Wregister]».
 - Fragmento de código:

```

232. register int i = 1;
233. register uint32 j = 0;
234. register int k = ( N > seedLength ? N : seedLength );
235.

```

Fig. 3.18. Fragmento de código del aviso 09 (MersenneTwister.h, Canneal)

- Aplicaciones que lo presentan: Canneal (18), Freqmine (2), Swaptions (2).
- Descripción: el especificador *register* se utilizaba como pista para el compilador para indicar que guardase en un registro del procesador la variable a la que precedía. Sin embargo, este especificador quedó obsoleto y, a partir de C++17, *register* se guardó como una palabra clave sin uso y reservada, a la espera de ser utilizada por el estándar más adelante.
- Impacto en la calidad: este fallo afecta principalmente a la portabilidad del código, ya que no podrá ser compilado en máquinas y sistemas que utilicen las versiones más recientes (ni futuras) del estándar ISO C++. El resto de propiedades de la calidad no se ven afectadas por este fallo.

- Solución: eliminar del código la declaración *register*, puesto que no proporciona ninguna función.

- **Aviso 10:**

- Mensaje del aviso: «No hay declaración de retorno en la función que devuelve non-void [-Wreturn-type]».
- Fragmento de código:

```
137. void* entry_pt(void* data)
138. {
139.     annealer_thread* ptr = static_cast<annealer_thread*>(data);
140.     ptr->Run();
141. }
142.
```

Fig. 3.19. Fragmento de código del aviso 10 (main.cpp, Canneal)

- Aplicaciones que lo presentan: Canneal (1).
- Descripción: cuando una función indica que devolverá algo distinto a *void* (como en el fragmento de ejemplo, *void**, que es un puntero vacío) pero no contiene una declaración de *return*, C++ puede producir comportamientos indefinidos.
- Impacto en la calidad: este fallo afecta a la fiabilidad del código en consecuencia del comportamiento indefinido explicado en el punto anterior. El resto de propiedades de la calidad no se ven afectadas por este fallo.
- Solución: en general, se debe incorporar una declaración de *return* que devuelva el tipo indicado por la función. En este caso, como se trata de un puntero vacío, se podría devolver un *nullptr*.

- **Aviso 11:**

- Mensaje del aviso: «El formato '%c' espera un argumento 'int' coincidente [-Wformat=] ».

- Fragmento de código:

```
192. if(n % chunksize != 0){
193.     printf("[Client] ensure that n \\% chunksize == 0\\n");
194.     exit(1);
195. }
196.
```

Fig. 3.20. Fragmento de código del aviso 11 (client.cpp, Netstreamcluster)

- Aplicaciones que lo presentan: Netstreamcluster (1).
- Descripción: el fallo que provoca este aviso se debe a una incorrecta implementación para imprimir un carácter escapado como «%». En consecuencia, el compilador no detecta que ese símbolo es un carácter escapado y lo trata como si fuera un comando de formato («%c») para recibir un *char*. Al no recibir la función ningún argumento que se corresponda con ese formato, el compilador lanza el aviso.
- Impacto en la calidad: este fallo afecta en primer lugar a la integridad funcional del código, pues este mensaje de error no se va a imprimir tal y como se había definido. También afecta claramente a la legibilidad del código y, por lo tanto, a su mantenimiento. El resto de propiedades de la calidad no se ven afectadas por este fallo.
- Solución: para escapar el carácter se debe escribir «%%» en vez de «/%».

- **Aviso 12:**

- Mensaje del aviso: «Bandera ' ' utilizada con el formato gnu_printf '%c' [-Wformat=]».
- Fragmento de código:

```
197. if(n % chunksize != 0){
198.     printf("[Client] ensure that n \\% chunksize == 0\\n");
199.     exit(1);
200. }
201.
```

Fig. 3.21. Fragmento de código del aviso 12 (client.cpp, Netstreamcluster)

- Aplicaciones que lo presentan: Netstreamcluster (1).
- Descripción: este aviso ocurre por la misma razón que el aviso anterior. El símbolo no se detecta como un carácter escapado para la impresión y el compilador entiende que se trata de un formato «%c» en el cual se ha introducido de por medio un espacio blanco inválido.

- Impacto en la calidad: como en el aviso anterior, este fallo afecta a la integridad funcional porque no se va a imprimir el mensaje definido para esa situación. También disminuye la mantenibilidad del código al dificultar su legibilidad. El resto de propiedades de la calidad no se ven afectadas por este fallo.
- Solución: para escapar el carácter se debe escribir «%%» en vez de «/%».

3.2. Análisis dinámico del código fuente

El análisis dinámico es una técnica de evaluación del software que permite examinar el comportamiento de un programa durante su ejecución [51]. De esta forma, se pueden cuantificar diferentes métricas que no pueden ser analizadas estáticamente, como las fugas de memoria del programa, las llamadas al sistema, los comportamientos indefinidos, los recursos que consume dicho programa o las condiciones de carrera en ejecuciones multihilo.

Si bien es cierto que existen numerosas herramientas destinadas a este fin, en este proyecto se ha decidido recurrir a los *sanitizers* que proporciona el compilador GCC. Los *sanitizers* de GCC son una opción del compilador que activa el análisis del código en tiempo de ejecución, permitiendo la evaluación de diferentes parámetros mientras el código se ejecuta. La lista completa de opciones de análisis dinámico que presenta GCC puede consultarse en [52].

Para este estudio de calidad se ha decidido utilizar las opciones «-fsanitize=undefined» y «-fsanitize=address», con el objetivo de buscar comportamientos indefinidos y errores en el manejo de la memoria que amenazan la calidad del código y suponen un riesgo para su fiabilidad. Estas opciones cubren un gran porcentaje del análisis dinámico de GCC puesto que incluyen la mayor parte de las opciones disponibles (por ejemplo, el *sanitizer address* incluye también al *sanitizer leak*, utilizado para detectar fugas de memoria). También se planteó el uso de «-fsanitize=thread» para detectar posibles condiciones de carrera entre los hilos, pero dicha opción quedó descartada debido a la complejidad que suponía solventar este tipo de comportamientos, ya que se requeriría cambiar por completo la arquitectura de los programas del *benchmark*.

Tras analizar el código dinámicamente con el *sanitizer undefined*, ninguna de las aplicaciones mantenidas en el estudio presentó comportamientos indefinidos. Sin embargo, tras compilar el código con el *sanitizer address*, cuatro aplicaciones diferentes han mostrado errores en el manejo de la memoria:

- **Freqmine**: presenta 8 fugas de memoria, de las cuales únicamente 5 han podido ser solucionadas.
- **Netstreamcluster**: presenta 12 fugas de memoria, de las cuales únicamente 9 han podido ser solucionadas.
- **Streamcluster**: presenta 8 fugas de memoria, las cuales han podido ser resueltas en su totalidad.
- **Swaptions**: presenta 3 fugas de memoria, las cuales no han podido ser solucionadas.

En total, el *sanitizer address* ha detectado 31 fugas de memoria, de las cuales han podido ser resueltas 22, casi un 71% (la recopilación completa puede consultarse en el Anexo A.2). Para solventar las fugas se ha buscado en las aplicaciones las estructuras que estaban reservando memoria sin liberarla, causando dichas fugas, y se ha tratado de liberar dicha memoria por medio de los operadores *free* y *delete*. Algunas de las fugas que no han podido ser solucionadas, como las detectadas en la aplicación Swaptions, son potencialmente falsos positivos, ya que los desarrolladores han implementado sus propias funciones para liberar la memoria pero el *sanitizer* de GCC no detecta dichas funciones. El resto de fugas que no han podido ser resueltas y que no son falsos positivos requerían modificaciones en el código altamente complejas, por lo que se ha decidido no tratarlas.

3.3. Análisis estático del código fuente

Debido a que el análisis dinámico puede resultar insuficiente si no se realiza con un considerable número de entradas diferentes, se ha decidido llevar a cabo también un análisis estático. El análisis estático consiste en la evaluación en profundidad del código fuente a través de una herramienta automatizada sin llegar a ejecutarlo, de tal forma que se detecten posibles errores en el manejo de la memoria, accesos inválidos en los arreglos o simples defectos de legibilidad, entre otros [53].

De entre las numerosas aplicaciones que existen para realizar este tipo de análisis, se ha decidido recurrir a la herramienta de análisis estático que proporciona el compilador Clang para el lenguaje C++, conocida como Clang-Tidy. Esta herramienta proporciona un elevado número de opciones de comprobación (*checks*), las cuales son agrupadas en diferentes categorías. La lista completa de *checks*, así como sus categorías, puede consultarse en [54].

Para este estudio se ha decidido utilizar un gran porcentaje de los *checks* disponibles, a excepción de las categorías de Android, Boost, Cert, Fuchsia, LLVM, MPI y Zircon, ya que se ha considerado que estas categorías y sus convenciones de código no eran relevantes para el análisis del *benchmark* PARSEC. En consecuencia, se ha optado por las categorías de Abseil (17 *checks*), Bugprone (48 *checks*), Clang-Analyzer (55 *checks*), CppCoreGuidelines (24 *checks*), Google (22 *checks*), Hicpp (30 *checks*), Misc (13 *checks*), Modernize (30 *checks*), Openmp (2 *checks*), Performance (14 *checks*), Portability (1 *checks*) y Readability (39 *checks*).

Dado el gran número de *checks* mantenidos para el estudio y la complejidad que supone analizar y solucionar los defectos que detectan dichos *checks*, se ha decidido analizar una única aplicación del *benchmark* PARSEC. Dicha aplicación elegida ha sido Streamcluster, ya que ha presentado avisos de compilación desde el inicio del estudio y errores de manejo de memoria en el análisis dinámico. De esta forma, se podrá comparar posteriormente el impacto de las soluciones aplicadas en cada iteración de este estudio de calidad.

A continuación se presentan ordenados por categorías los *checks* de Clang-Tidy que han generado avisos tras analizar con ellos el código fuente, así como el enfoque empleado para solucionarlos:

- **Bugprone:**

- Bugprone-narrowing-conversions: este *check* avisa de una posible reducción en la memoria en una conversión de tipos entre variables. Por ejemplo, al asignar una variable de tipo *long* a una de tipo *int*, tal y como se muestra en la Fig. 3.22. Esta clase de fallos se ha solucionado utilizando el tipo adecuado para cada variable.

```

708. //my block
709. long bsize = points->num/nproc;
710. long k1 = bsize * pid;
711. long k2 = k1 + bsize;
712. if( pid == nproc-1 ) k2 = points->num;

    /* ... */

725. /* create center at first point, send it to itself */
726. for( int k = k1; k < k2; k++ ) {
727.     float distance = dist(points->p[k],points->p[0],points->dim);
728.     points->p[k].cost = distance * points->p[k].weight;
729.     points->p[k].assign=0;
730. }

```

Fig. 3.22. Conversión con posible reducción de memoria (streamcluster.cpp)

- Clang-analyzer:

- Clang-analyzer-security.insecureAPI.strcpy: este *check* avisa de que la función *strcpy*, mostrada en la Fig. 3.23, no es segura ya que no comprueba el tamaño de los *buffers* que recibe como argumento y genera comportamientos indefinidos si el buffer de destino es más pequeño que el buffer de origen. Para solucionar este fallo se ha recurrido a utilizar los *strings* de C++, tal y como ilustra la Fig. 3.24, los cuales permiten el uso de forma segura del operador de asignación «=».

```

1999. kmin = atoi(argv[1]);
2000. kmax = atoi(argv[2]);
2001. dim = atoi(argv[3]);
2002. n = atoi(argv[4]);
2003. chunksize = atoi(argv[5]);
2004. clustersize = atoi(argv[6]);
2005. strcpy(infilename, argv[7]);
2006. strcpy(outfilename, argv[8]);
2007. nproc = atoi(argv[9]);

```

Fig. 3.23. Uso de la función *strcpy* (streamcluster.cpp)

```

1999. kmin = atoi(argv[1]);
2000. kmax = atoi(argv[2]);
2001. dim = atoi(argv[3]);
2002. n = atoi(argv[4]);
2003. chunksize = atoi(argv[5]);
2004. clustersize = atoi(argv[6]);
2005. string infilename = argv[7];
2006. string outfilename = argv[8];
2007. nproc = atoi(argv[9]);

```

Fig. 3.24. Uso de variables de tipo *string* (streamcluster.cpp)

- **Cppcoreguidelines:**

- Cppcoreguidelines-pro-type-vararg: este *check* avisa de que las funciones *vararg* de C, es decir, aquellas funciones de C que toman un número variable de argumentos (como la función *fprintf* que se muestra en la Fig. 3.25), son inseguras ya que asumen que se leerá un argumento del tipo correcto y no imponen seguridad de tipos. Para solucionar este error se planteó inicialmente utilizar las funciones de la librería *Iostream* de C++, las cuales sí proporcionan seguridad de tipos. El problema de estas funciones es que su rendimiento es considerablemente peor comparadas con las *vararg* de C. En consecuencia, se ha decidido utilizar las funciones de formateo de *{fmt}*, una librería de código abierto que proporciona una alternativa segura y eficiente a las funciones *vararg* de C y a los *iostreams* de C++ [55]. En la Fig. 3.26 puede verse un ejemplo de estas funciones.

```
1823. for( int i = 0; i < centers->num; i++ ) {
1824.     if( is_a_median[i] ) {
1825.         fprintf(fp, "%u\n", centerIDs[i]);
1826.         fprintf(fp, "%lf\n", centers->p[i].weight);
1827.         for( int k = 0; k < centers->dim; k++ ) {
1828.             fprintf(fp, "%lf ", centers->p[i].coord[k]);
1829.         }
1830.         fprintf(fp, "\n\n");
1831.     }
1832. }
```

Fig. 3.25. Uso de funciones *vararg* (streamcluster.cpp)

```
1823. for( int i = 0; i < centers->num; i++ ) {
1824.     if( is_a_median[i] != 0 ) {
1825.         fmt::print(fp.get(), "{}\n", centerIDs[i]);
1826.         fmt::print(fp.get(), "%{}\n", centers->p[i].weight);
1827.         for( int k = 0; k < centers->dim; k++ ) {
1828.             fmt::print(fp.get(), "{} ", centers->p[i].coord[k]);
1829.         }
1830.         fmt::print(fp.get(), "\n\n");
1831.     }
1832. }
```

Fig. 3.26. Uso de funciones de la librería *{fmt}* (streamcluster.cpp)

- Cppcoreguidelines-init-variables: este *check* comprueba que las variables no se utilicen sin inicializar, como ocurre en la Fig. 3.27. Para solucionarlo se han inicializado todas las variables a la vez que eran declaradas.

```

610. /* comparator for floating point numbers */
611. static int floatcomp(const void *i, const void *j)
612. {
613.     float a, b;
614.     a = *(float *)(i);
615.     b = *(float *)(j);
616.     if (a > b) return (1);
617.     if (a < b) return (-1);
618.     return(0);
619. }

```

Fig. 3.27. Declaración de variables sin inicialización (streamcluster.cpp)

- Cppcoreguidelines-pro-bounds-pointer-arithmetic: este *check* genera un aviso cuando el código presenta operaciones aritméticas con punteros de memoria, como la que puede observarse en la Fig. 3.28, ya que dichas operaciones pueden resultar en punteros inválidos. Para solventar este fallo se han sustituido los punteros de memoria que provocaban los avisos por vectores de C++ y por iteradores, tal y como se aprecia en la Fig. 3.29, los cuales permiten acceder de forma segura a los datos.

```

634. /* shuffle an array of integers */
635. void intshuffle(int *intarray, int length)
636. {
637.     long i, j;
638.     int temp;
639.     for (i=0; i<length; i++) {
640.         j=(lrand48()%(length - i))+i;
641.         temp = intarray[i];
642.         intarray[i]=intarray[j];
643.         intarray[j]=temp;
644.     }
645. }

```

Fig. 3.28. Uso de aritmética de punteros (streamcluster.cpp)

```

634. /* shuffle an array of integers */
635. void intshuffle(vector<long> intarray, long length)
636. {
637.     for (long i=0; i<length; i++) {
638.         long j=(lrand48()%(length - i))+i;
639.         long temp = intarray[i];
640.         intarray[i]=intarray[j];
641.         intarray[j]=temp;
642.     }
643. }

```

Fig. 3.29. Uso de vectores (streamcluster.cpp)

- Cppcoreguidelines-no-malloc: este *check* comprueba que no se maneje manualmente la memoria mediante operadores como *malloc*, *free*, *new* o *delete*, tal y como ocurre en la Fig. 3.30. Para solucionarlo se han

sustituido los punteros de memoria que generaban los avisos por vectores de C++. También se ha recurrido a punteros inteligentes (*smart pointers*) que simplifican el manejo de la memoria, como puede apreciarse en la Fig. 3.31, donde no es necesario llamar a la función *delete* para destruir el puntero y liberar la memoria.

```
2017. PStream* stream;
2018. if( n > 0 ) {
2019.     stream = new SimStream(n);
2020. }
2021. else {
2022.     stream = new FileStream(infilename);
2023. }

    /* ... */

2036. delete stream;
```

Fig. 3.30. Manejo manual de la memoria (streamcluster.cpp)

```
2017. unique_ptr<PStream> stream;
2018. if( n > 0 ) {
2019.     stream = make_unique <SimStream> (n);
2020. }
2021. else {
2022.     stream = make_unique <FileStream> (infilename);
2023. }
```

Fig. 3.31. Uso de punteros inteligentes (streamcluster.cpp)

- Cppcoreguidelines-avoid-magic-numbers: este *check* detecta si se utilizan números directamente en el código, tal y como ocurre en la Fig. 3.32, en vez de incorporarlos a través de variables o constantes. Para solucionar este fallo se han reemplazado los valores numéricos no declarados por expresiones constantes (*constexpr*), como las definidas en la Fig. 3.33.

```
1594. if (k < kmin) {
1595.     /* facilities too expensive */
1596.     /* decrease facility cost and reduce the cost accordingly */
1597.     hiz = z; z = (hiz+loz)/2.0;
1598.     cost += (z-hiz)*k;
1599. }
1600.
1601. /* if k is good, return the result */
1602. /* if we're stuck, just give up and return what we have */
1603. if (((k <= kmax)&&(k >= kmin))||((loz >= (0.999)*hiz)) ){
1604.     break;
1605. }
```

Fig. 3.32. Uso de valores numéricos sin declarar (streamcluster.cpp)

```

1594. constexpr double divisor = 2.0;
1595. constexpr double almost_one = 0.999;
1596. if (k < kmin) {
1597.     /* facilities too expensive */
1598.     /* decrease facility cost and reduce the cost accordingly */
1599.     hiz = z; z = (hiz+loz)/ divisor ;
1600.     cost += (z-hiz)*k;
1601. }
1602.
1603. /* if k is good, return the result */
1604. /* if we're stuck, just give up and return what we have */
1605. if (((k <= kmax)&&(k >= kmin))||((loz >= (almost_one)*hiz)) ){
1606.     break;
1607. }

```

Fig. 3.33. Uso de *constexpr* (streamcluster.cpp)

- Cppcoreguidelines-special-member-functions: este *check* comprueba si las clases definidas siguen la «regla del cinco». Según esta regla, cuando una clase define un destructor (como ocurre en la Fig. 3.34), un constructor de copia o un constructor de movimiento, posiblemente necesite las tres funciones especiales, así como un operador de asignación de copia y un operador de asignación de movimiento [56]. Para solventar este fallo se han añadido las funciones y los operadores que faltaban en las clases, como ilustra la Fig. 3.35.

```

1757. //synthetic stream
1758. class SimStream : public PStream {
1759. public:
1760.     SimStream(long n_ ) {
1761.         n = n_;
1762.     }
1763.     size_t read( float* dest, int dim, int num ) {
1764.         size_t count = 0;
1765.         for( int i = 0; i < num && n > 0; i++ ) {
1766.             for( int k = 0; k < dim; k++ ) {
1767.                 dest[i*dim + k] = lrand48()/(float)INT_MAX;
1768.             }
1769.             n--;
1770.             count++;
1771.         }
1772.         return count;
1773.     }
1774.     int ferror() {
1775.         return 0;
1776.     }
1777.     int feof() {
1778.         return n <= 0;
1779.     }
1780.     ~SimStream() {
1781.     }
1782. private:
1783.     long n;
1784. };

```

Fig. 3.34. Clase incumpliendo la «regla del cinco» (streamcluster.cpp)

```

1757. //synthetic stream
1758. class SimStream : public PStream {
1759. public:
1760.     SimStream(long n_ ) explicit {
1761.         n = n_;
1762.     }
1763.     size_t read( float* dest, int dim, int num ) override{
1764.         size_t count = 0;
1765.         for( int i = 0; i < num && n > 0; i++ ) {
1766.             for( int k = 0; k < dim; k++ ) {
1767.                 dest[i*dim + k] = (float)lrand48()/(float)INT_MAX;
1768.             }
1769.             n--;
1770.             count++;
1771.         }
1772.         return count;
1773.     }
1774.     int ferror() override{
1775.         return 0;
1776.     }
1777.     int feof() override{
1778.         return static_cast<int> (n <= 0);
1779.     }
1780.     ~SimStream() = default;
1781.     SimStream(const SimStream& other) = default;
1782.     SimStream& operator=(const SimStream& other) = default;
1783.     SimStream(SimStream&& other) = default;
1784.     SimStream& operator=(SimStream&& other) = default;
1785.
1786. private:
1787.     long n;
1788. };

```

Fig. 3.35. Clase cumpliendo la «regla del cinco» (streamcluster.cpp)

- Cppcoreguidelines-explicit-virtual-functions: este *check* comprueba que se utiliza la declaración *override* (o *final*) para indicar al compilador que la función definida es una modificación de la función virtual declarada en la clase base. Para solucionar este defecto se ha incluido la palabra clave *override* en la declaración de la función de la clase que hereda dicha función, como muestra la Fig. 3.36.

```

1748. class PStream {
1749. public:
1750.     virtual size_t read( float* dest, int dim, int num ) = 0;

    / ... /

1758. //synthetic stream
1759. class SimStream : public PStream {
1760. public:
1761.     SimStream(long n_ ) explicit {
1762.         n = n_;
1763.     }
1764.     size_t read( float* dest, int dim, int num ) override{

    / ... /

```

Fig. 3.36. Uso de *override* (streamcluster.cpp)

- **Google:**

- Google-explicit-constructor: este *check* sugiere añadir la palabra clave *explicit* en constructores que toman un único argumento como parámetro para evitar conversiones implícitas. Sin la palabra clave *explicit*, el compilador tiene la capacidad de realizar una conversión implícita en este tipo de constructores, lo cual es un comportamiento peligroso ya que esta conversión puede pasar desapercibida para el programador y provocar errores difíciles de detectar. Para evitar este comportamiento se ha añadido *explicit* a los constructores que tomaban un único argumento como parámetro, tal y como se muestra en la Fig. 3.37.

```
1757. //synthetic stream
1758. class SimStream : public PStream {
1759. public:
1760.     SimStream(long n_ ) explicit {
1761.         n = n_;
1762.     }
```

Fig. 3.37. Uso de *explicit* (streamcluster.cpp)

- **Hicpp:**

- Hicpp-deprecated-headers: este *check* comprueba que no se utilicen las cabeceras de C (como ocurre en la Fig. 3.38), pues algunas de estas librerías están obsoletas en C++ o no tienen ningún efecto. En consecuencia, se ha modernizado el código con las cabeceras de C++, tal y como se presenta en la Fig. 3.39.

```
9.  #include <stdio.h>
10. #include <iostream>
11. #include <fstream>
12. #include <stdlib.h>
13. #include <string.h>
14. #include <assert.h>
15. #include <math.h>
```

Fig. 3.38. Cabeceras de C (streamcluster.cpp)

```
9.  #include <cstdio>
10. #include <iostream>
11. #include <fstream>
12. #include <cstdlib>
13. #include <cstring>
14. #include <cassert>
15. #include <cmath>
```

Fig. 3.39. Cabeceras de C++ (streamcluster.cpp)

- Hicpp-static-assert: este *check* sugiere reemplazar *assert* por *static_assert* siempre que sea posible, tal y como se ilustra en la Fig. 3.40 y en la Fig. 3.41. La función *static_assert* evalúa la condición en tiempo de compilación, siendo más segura y eficiente que la función *assert*, que evalúa la condición en tiempo de ejecución.

```
62. //check assumptions used in header
63. assert(PARSEC_BARRIER_SERIAL_THREAD != 0);
```

Fig. 3.40. Uso de *assert* (parsec_barrier.cpp)

```
62. //check assumptions used in header
63. static_assert(PARSEC_BARRIER_SERIAL_THREAD != 0);
```

Fig. 3.41. Uso de *static_assert* (parsec_barrier.cpp)

- Hicpp-braces-around-statements: este *check* comprueba que los cuerpos de las declaraciones condicionales y de los bucles estén definidos entre llaves, con el objetivo de incrementar la legibilidad del código. Para cumplir con dicha convención se han añadido llaves a todas las declaraciones condicionales y bucles que no presentaban este símbolo. La Fig. 3.42 y la Fig. 3.43 ejemplifican la diferencia en la legibilidad entre el ausencia y la presencia de llaves en este tipo de declaraciones.

```
65. //check arguments
66. if(barrier==NULL) return EINVAL;
67. if(count<=0) return EINVAL;
68. //only private barriers (the default) are currently supported
69. if(attr!=NULL && *attr==PARSEC_PROCESS_PRIVATE) NOT_IMPLEMENTED();
```

Fig. 3.42. Declaraciones condicionales sin llaves (parsec_barrier.cpp)

```
65. //check arguments
66. if(barrier==nullptr){
67.     return EINVAL;
68. }
69. if(count<=0){
70.     return EINVAL;
71. }
72. //only private barriers (the default) are currently supported
73. if(attr!=nullptr && *attr==PARSEC_PROCESS_PRIVATE){
74.     NOT_IMPLEMENTED();
75. }
```

Fig. 3.43. Declaraciones condicionales con llaves (parsec_barrier.cpp)

- Hicpp-use-auto: este *check* sugiere el uso del especificador de tipos *auto* para deducir el tipo de una variable en ciertas inicializaciones donde dicho

uso mejore la legibilidad del código. Un claro ejemplo son los iteradores, cuya declaración suele ser larga y legiblemente compleja, como puede verse en la Fig. 3.44. Para solucionar este defecto se ha sustituido el tipo de las variables por el especificador *auto* en aquellas declaraciones que el *check* proponía, como muestra la Fig. 3.45.

```
1018. //my *lower* fields
1019. vector<double>::iterator lower = work_mem.begin() + pid*stride;
1020. //global *lower* fields
1021. vector<double>::iterator gl_lower = work_mem.begin() + nproc*stride;
```

Fig. 3.44. Declaración de un iterador (streamcluster.cpp)

```
1018. //my *lower* fields
1019. auto lower = work_mem.begin() + pid*stride;
1020. //global *lower* fields
1021. auto gl_lower = work_mem.begin() + nproc*stride;
```

Fig. 3.45. Declaración con *auto* (streamcluster.cpp)

- Hicpp-use-equals-default: este *check* comprueba que se utilice la expresión «= default» para definir cuerpos *default* de funciones especiales, como constructores y destructores, en vez del uso de llaves vacías. Esta declaración explícita de que la función es trivial permite mejores opciones de optimización para el compilador. Para solucionar este error se ha añadido la expresión «= default» en aquellas funciones especiales con llaves vacías. La Fig. 3.46 y la Fig. 3.47 muestran respectivamente un ejemplo del fallo y su solución.

```
1748. class PStream {
1749. public:
1750.     virtual size_t read( float* dest, int dim, int num ) = 0;
1751.     virtual int ferror() = 0;
1752.     virtual int feof() = 0;
1753.     virtual ~PStream() {
1754.     }
```

Fig. 3.46. Función especial con cuerpo trivial usando llaves (streamcluster.cpp)

```
1748. class PStream {
1749. public:
1750.     virtual size_t read( float* dest, int dim, int num ) = 0;
1751.     virtual int ferror() = 0;
1752.     virtual int feof() = 0;
1753.     virtual ~PStream() = default;
```

Fig. 3.47. Función especial con cuerpo trivial usando «= default» (streamcluster.cpp)

- **Misc:**

- Misc-unused-parameters: este *check* avisa de posibles parámetros en una función que nunca llegan a utilizarse dentro de dicha función. En este caso, se trata de un falso positivo, ya que dependiendo de ciertas declaraciones el parámetro marcado como *unused* se utilizará o no, como puede verse en la Fig. 3.48. Para solventar este problema se ha utilizado la declaración *maybe_unused*, la cual suprime el aviso. El uso de esta declaración puede apreciarse en la Fig. 3.49.

```
675.double pspeedy(Points *points, double z, long *kcenter, int pid,  
    pthread_barrier_t* barrier)  
676.{  
677.#ifdef ENABLE_THREADS  
678.    pthread_barrier_wait(barrier);  
679.#endif  
680.    //my block  
681.    long bsize = points->num/nproc;  
682.    long k1 = bsize * pid;  
683.    long k2 = k1 + bsize;  
684.    if( pid == nproc-1 ){  
685.        k2 = points->num;  
686.    }
```

Fig. 3.48. Falso positivo de un parámetro sin uso (streamcluster.cpp)

```
675.double pspeedy(Points *points, double z, long *kcenter, int pid,  
    [[maybe_unused]] pthread_barrier_t* barrier)  
676.{  
677.#ifdef ENABLE_THREADS  
678.    pthread_barrier_wait(barrier);  
679.#endif  
680.    //my block  
681.    long bsize = points->num/nproc;  
682.    long k1 = bsize * pid;  
683.    long k2 = k1 + bsize;  
684.    if( pid == nproc-1 ){  
685.        k2 = points->num;  
686.    }
```

Fig. 3.49. Uso de *maybe_unused* (streamcluster.cpp)

- **Modernize:**

- Modernize-use-nullptr: este *check* recomienda el uso de *nullptr* en lugar de constantes de puntero nulo como *NULL*. La razón detrás de esta sugerencia es que *NULL* se evalúa como un entero (ya que se define como una constante igual a 0), mientras que *nullptr* se evalúa como un puntero, presentando un comportamiento más acorde a su definición y su uso. Para

solucionar este defecto se han sustituido todas las declaraciones de *NULL* en el código por *nullptr*, tal y como muestran la Fig. 3.50 y la Fig. 3.51.

```
1790. if( fp == NULL ) {  
1791.     fprintf(stderr, "error opening file %s\n.", filename);  
1792.     exit(1);  
1793. }
```

Fig. 3.50. Uso de *NULL* (streamcluster.cpp)

```
1790. if( fp == nullptr ) {  
1791.     fmt::print(stderr, "error opening {}\n", outfile);  
1792.     exit(1);  
1793. }
```

Fig. 3.51. Uso de *nullptr* (streamcluster.cpp)

- Modernize-use-using: este *check* sugiere uso de *using* para la declaración de alias de tipos en lugar de *typedef*, ya que el primero permite declarar alias con plantillas. Para cumplir con esta recomendación se ha reemplazado las declaraciones de *typedef* por *using*, tal y como se ilustra en la Fig. 3.52 y la Fig. 3.53.

```
67. typedef int parsec_barrierattr_t;
```

Fig. 3.52. Declaración de alias por medio de *typedef* (parsec_barrier.hpp)

```
67. using parsec_barrierattr_t = int;
```

Fig. 3.53. Declaración de alias por medio de *using* (parsec_barrier.hpp)

- Modernize-use-bool-literals: este *check* detecta aquellos valores enteros que son utilizados como booleanos, como ocurre en la Fig. 3.54, práctica que dificulta la legibilidad del código. Para solucionar este fallo se han sustituido los valores enteros por su correspondiente valor *bool*, como muestra la Fig. 3.55.

```
742. while(1) {
```

Fig. 3.54. Uso de un valor entero como *bool* (streamcluster.cpp)

```
742. while(true) {
```

Fig. 3.55. Uso de un literal *bool* (streamcluster.cpp)

- **Readability:**

- Readability-non-const-parameter: este *check* detecta en las funciones aquellos parámetros de tipo puntero que podrían declararse como punteros a tipos constantes por medio de la palabra clave *const*. Usada correctamente, esta palabra clave hace más segura la función, evitando posibles modificaciones inintencionadas de los datos y produciendo avisos adicionales en caso de que los datos no estén inicializados. Para cumplir con esta práctica, se ha incorporado la declaración *const* a aquellos parámetros indicados por el *check*, como muestran la Fig. 3.56 y la Fig. 3.57.

```
97. //Barrier attribute initialization & destruction
98. int parsec_barrierattr_destroy(parsec_barrierattr_t *attr) {
99.     if(attr==NULL) return EINVAL;
100.    //simply do nothing
101.    return 0;
102. }
```

Fig. 3.56. Parámetro de tipo puntero susceptible de ser declarado constante (streamcluster.cpp)

```
103. //Barrier attribute initialization & destruction
104. int parsec_barrierattr_destroy(const parsec_barrierattr_t *attr) {
105.     if(attr==NULL) return EINVAL;
106.     //simply do nothing
107.     return 0;
108. }
```

Fig. 3.57. Parámetro de tipo puntero constante (streamcluster.cpp)

- Readability-implicit-bool-conversion: este *check* comprueba la existencia de conversiones implícitas entre booleano y otros tipos, como enteros, las cuales disminuyen la legibilidad del código. Para evitar este tipo de conversiones implícitas se ha utilizado el tipo requerido en cada declaración que presentaba este tipo de comportamientos, tal y como ejemplifican la Fig. 3.58 y la Fig. 3.59.

```
139. while(!barrier->is_arrival_phase) {
```

Fig. 3.58. Conversión implícita de *int* a *bool* (parsec_barrier.cpp)

```
139. while(barrier->is_arrival_phase == 0) {
```

Fig. 3.59. Condición booleana sin conversiones implícitas (parsec_barrier.cpp)

4. EVALUACIÓN DE LOS RESULTADOS

En este cuarto capítulo se presenta el análisis de rendimiento de las aplicaciones del *benchmark* PARSEC, comparando el tiempo de ejecución de las versiones iniciales del código fuente con el de las versiones modernizadas tras las diferentes iteraciones del estudio de calidad. También se expone el análisis de la complejidad ciclomática entre las diferentes versiones.

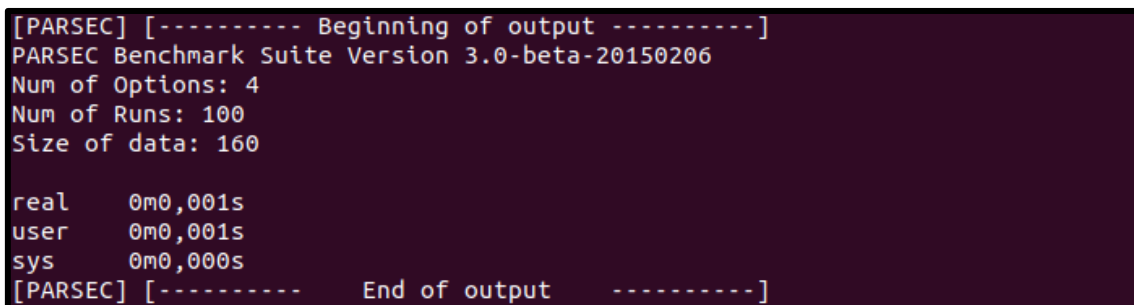
4.1. Análisis de rendimiento

4.1.1. Entorno, metodología de análisis y repositorio

El análisis de rendimiento entre las diferentes versiones de las aplicaciones que componen el *benchmark* PARSEC se ha realizado en una máquina compuesta por un procesador Ryzen 7 5800x de 8 núcleos (16 hilos) con una frecuencia de 3.8 GHz y 16 GB de RAM DDR4 con una frecuencia de 3600 MHz. El sistema operativo utilizado es Linux, bajo la distribución de Ubuntu 20.04.

La metodología seguida para realizar este análisis se basa en el tiempo de ejecución de las aplicaciones del *benchmark*. Dicho tiempo de ejecución se obtiene del *output* de las propias aplicaciones (Fig. 4.1), las cuales hacen uso del comando *time* para mostrar tres medidas diferentes:

- **Real**: tiempo real transcurrido desde el inicio del programa hasta su finalización.
- **User**: tiempo del procesador transcurrido en modo usuario (fuera del kernel).
- **Sys**: tiempo del procesador transcurrido en modo kernel.



```
[PARSEC] [----- Beginning of output -----]
PARSEC Benchmark Suite Version 3.0-beta-20150206
Num of Options: 4
Num of Runs: 100
Size of data: 160

real    0m0,001s
user    0m0,001s
sys     0m0,000s
[PARSEC] [----- End of output -----]
```

Fig. 4.1. *Output* de una aplicación del *benchmark* PARSEC

En ejecuciones secuenciales, la suma de los valores de *user* y *sys* debería ser muy similar a *real*. Sin embargo, en ejecuciones multihilo, *user* y *sys* muestran el total del tiempo transcurrido por cada hilo, por lo que su suma superará al valor de *real*. En consecuencia, para evitar malentendidos y simplificar el análisis, se ha decidido utilizar únicamente la medida de *real* para la evaluación del rendimiento. No obstante, las medidas de *user* y *sys* de cada ejecución se han mantenido para comprobar que ninguna modificación en el código produjese un comportamiento extraño (como un bloqueo en el modo kernel) y pueden ser consultadas en el Anexo B, donde se recoge la experimentación completa.

Para evaluar cada programa del *benchmark* se han realizado 10 ejecuciones en diferentes modos de ejecución: secuencial, 2 hilos, 4 hilos, 8 hilos y 16 hilos. La razón principal por la que se han definido los diferentes modos de ejecución es la naturaleza del *benchmark* PARSEC, cuyo objetivo es la evaluación de las arquitecturas multihilo. Además, esta metodología permite comprobar si la variación en el número de hilos afecta al impacto que las mejoras de calidad puedan tener sobre el rendimiento del programa.

Una vez obtenidos los tiempos de ejecución de cada modo para cada aplicación, se ha calculado la media aritmética de dichos tiempos de ejecución. También se ha calculado su desviación típica, con el objetivo de asegurar que los valores obtenidos fueran significativos y no fuese necesario realizar más ejecuciones.

Finalmente, para comparar las diferentes versiones del código, se ha calculado el *speedup* (aceleración) usando las medias de tiempo de ejecución de cada versión. El *speedup* (S) se define como el cociente entre el tiempo de ejecución antiguo (T_a) y el tiempo de ejecución mejorado (T_m). La Fig. 4.2 ilustra la fórmula para calcular esta medida. Un *speedup* mayor que 1 ($S > 1$) implica una mejora en el rendimiento, mientras que un *speedup* menor que 1 ($S < 1$) implica una disminución. Si el *speedup* es igual a 1 ($S = 1$), se puede afirmar que el rendimiento no ha variado.

$$S = \frac{T_a}{T_m}$$

Fig. 4.2. Fórmula del *speedup*

Se debe mencionar que las aplicaciones del *benchmark* PARSEC cuentan con diferentes entradas. Entre estas entradas encontramos el *input test*, diseñado con el objetivo de

depurar el código y probar las funcionalidades básicas del programa; los *inputs simdev*, *simsmall*, *simmedium* y *simlarge*, diseñados para la evaluación de simuladores; y el *input native*, diseñado para el estudio de la ejecución nativa en multiprocesadores. Dado que no tendría sentido utilizar las entradas de *test* ni las de simulación en este estudio de calidad, el *input native* es la entrada elegida para realizar el análisis del rendimiento. Para todas las aplicaciones del *benchmark* PARSEC, la entrada *native* es la de mayor tamaño y, por lo tanto, la más realista.

Para reproducir este análisis, el código modernizado de las aplicaciones del *benchmark* PARSEC ha sido subido a un repositorio de GitHub. Dicho repositorio puede consultarse en el siguiente enlace: <https://github.com/albertogg99/PARSEC>.

4.1.2. Versión modernizada comparada con la versión base

En esta primera sección del análisis se compara el rendimiento de la versión base, es decir, la versión inicial del código que producía avisos de compilación; con el rendimiento de la versión modernizada, es decir, la versión modificada del código que solventaba dichos avisos generados por el compilador y que fueron analizados en el apartado 3.1.2 de este documento.

A continuación se expone, para cada aplicación del *benchmark* mantenida en la primera iteración del estudio, una tabla que ilustra, para cada modo de ejecución (secuencial, 2 hilos, 4 hilos, 8 hilos y 16 hilos), el tiempo de ejecución medio de cada versión, su desviación típica muestral y el *speedup* entre ambas versiones.

Tabla 4.1. Rendimiento de la versión modernizada frente a la versión base en la aplicación Blackscholes

Blackscholes				
Hilos	Versión	Media (s)	Desviación típica (s)	Speedup
1	Base	33,0804	0,0807	0,9985
	Modernizada	33,1288	0,1854	
2	Base	20,0675	0,0600	0,9993
	Modernizada	20,0825	0,0587	
4	Base	13,6946	0,0747	1,0001
	Modernizada	13,6931	0,0615	
8	Base	10,6383	0,0733	0,9982
	Modernizada	10,6571	0,0879	
16	Base	9,607	0,0736	1,0009
	Modernizada	9,5981	0,0955	

Tabla 4.2. Rendimiento de la versión modernizada frente a la versión base en la aplicación Bodytrack

Bodytrack				
Hilos	Versión	Media (s)	Desviación típica (s)	<i>Speedup</i>
1	Base	47,161	0,1647	0,9972
	Modernizada	47,2953	0,2918	
2	Base	25,9088	0,1070	0,9931
	Modernizada	26,09	0,1198	
4	Base	14,8849	0,1481	0,9996
	Modernizada	14,8909	0,1484	
8	Base	10,3566	0,1051	1,0015
	Modernizada	10,3413	0,1195	
16	Base	8,2592	0,3231	0,9998
	Modernizada	8,2611	0,2496	

Tabla 4.3. Rendimiento de la versión modernizada frente a la versión base en la aplicación Canneal

Canneal				
Hilos	Versión	Media (s)	Desviación típica (s)	<i>Speedup</i>
1	Base	73,3666	0,3371	1,0121
	Modernizada	72,4911	0,2641	
2	Base	44,593	0,1503	1,0113
	Modernizada	44,095	0,2448	
4	Base	30,6972	0,1325	1,0055
	Modernizada	30,5296	0,1684	
8	Base	24,4985	0,1399	1,0050
	Modernizada	24,3759	0,0955	
16	Base	22,4258	0,1019	1,0014
	Modernizada	22,3938	0,0890	

Tabla 4.4. Rendimiento de la versión modernizada frente a la versión base en la aplicación Fluidanimate

Fluidanimate				
Hilos	Versión	Media (s)	Desviación típica (s)	<i>Speedup</i>
1	Base	155,8691	0,1710	0,9997
	Modernizada	155,9142	0,2926	
2	Base	83,0472	0,2646	1,0009
	Modernizada	82,9684	0,2408	
4	Base	45,3338	0,0965	0,9981
	Modernizada	45,4219	0,1705	
8	Base	31,0106	0,2815	0,9985
	Modernizada	31,0587	0,1633	
16	Base	23,0082	0,0714	0,9936
	Modernizada	23,1571	0,1225	

Tabla 4.5. Rendimiento de la versión modernizada frente a la versión base en la aplicación Freqmine

Freqmine				
Hilos	Versión	Media (s)	Desviación típica (s)	Speedup
1	Base	207,917	0,5593	0,9951
	Modernizada	208,9393	1,2051	
2	Base	104,9523	0,7734	0,9947
	Modernizada	105,5119	0,7623	
4	Base	54,5789	0,1931	1,0003
	Modernizada	54,5644	0,2224	
8	Base	29,5934	0,6232	1,0000
	Modernizada	29,5946	0,4680	
16	Base	24,2245	0,2645	1,0013
	Modernizada	24,1934	0,2106	

Tabla 4.6. Rendimiento de la versión modernizada frente a la versión base en la aplicación Netstreamcluster

Netstreamcluster				
Hilos	Versión	Media (s)	Desviación típica (s)	Speedup
1	Base	236,681	2,4303	0,9941
	Modernizada	238,0967	1,7374	
2	Base	124,0665	0,3972	0,9980
	Modernizada	124,3121	0,3906	
4	Base	69,5519	0,1360	0,9992
	Modernizada	69,6067	0,1341	
8	Base	52,5676	0,2320	1,0064
	Modernizada	52,2313	0,2519	
16	Base	51,5118	0,5340	0,9983
	Modernizada	51,5977	0,9037	

Tabla 4.7. Rendimiento de la versión modernizada frente a la versión base en la aplicación Streamcluster

Streamcluster				
Hilos	Versión	Media (s)	Desviación típica (s)	Speedup
1	Base	237,9539	1,489451536	0,999108186
	Modernizada	238,1663	1,371354971	
2	Base	124,2905	0,455708789	1,001761882
	Modernizada	124,0719	0,289302014	
4	Base	69,4926	0,071941643	1,004586897
	Modernizada	69,1753	0,191278767	
8	Base	52,1121	0,10189041	1,000460758
	Modernizada	52,0881	0,184412852	
16	Base	53,1751	0,524018755	1,004124124
	Modernizada	52,9567	0,505705679	

Tabla 4.8. Rendimiento de la versión modernizada frente a la versión base en la aplicación Swaptions

Swaptions				
Hilos	Versión	Media (s)	Desviación típica (s)	<i>Speedup</i>
1	Base	79,9741	0,564066279	1,007207645
	Modernizada	79,4018	0,353751576	
2	Base	40,9464	0,18633554	1,007591436
	Modernizada	40,6379	0,435136364	
4	Base	20,9581	0,239412452	1,010944856
	Modernizada	20,7312	0,224854916	
8	Base	11,677	0,35962913	1,004723759
	Modernizada	11,6221	0,327700897	
16	Base	9,8895	0,248074474	0,991746726
	Modernizada	9,9718	0,288413207	

La razón por la que las tablas han sido expuestas de forma seguida, sin ser comentadas individualmente, es que todas las aplicaciones presentan el mismo comportamiento: el rendimiento de la versión modernizada no varía respecto al rendimiento de la versión base, sin importar el número de hilos utilizado. Esto puede apreciarse en los valores de *speedup* de cada tabla, todos extremadamente cercanos a la unidad.

Para ilustrar de una forma más visual este comportamiento, se ha decidido representar gráficamente los tiempos de ejecución de una de las aplicaciones del *benchmark* PARSEC (Fig. 4.3). Se ha elegido la aplicación Streamcluster porque es la única aplicación mantenida en las tres iteraciones de este estudio de la calidad.

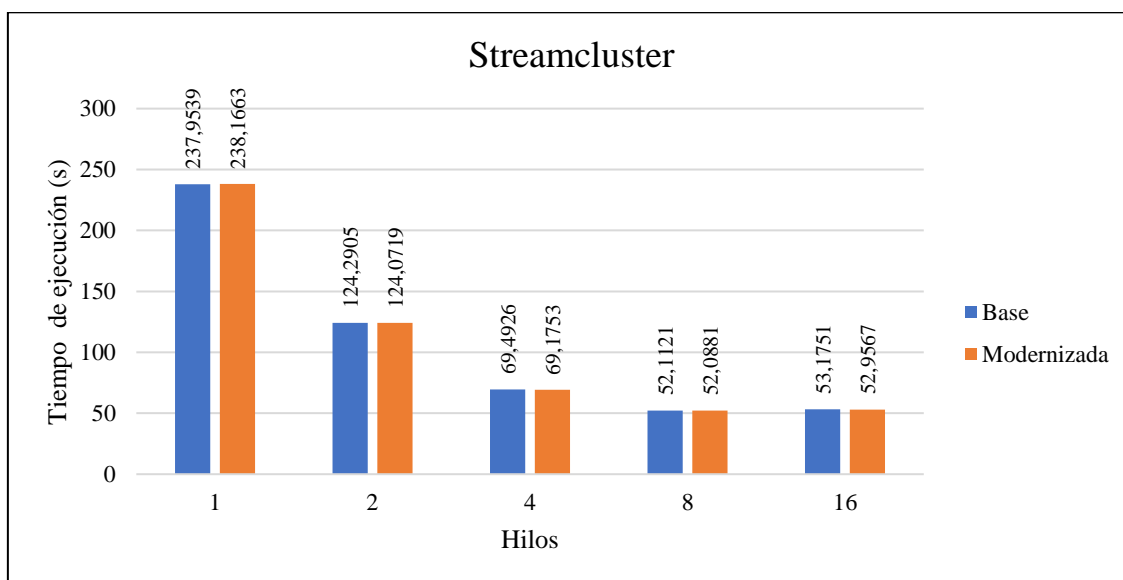


Fig. 4.3. Gráfica del tiempo de ejecución en la aplicación Streamcluster (versión base frente a versión modernizada)

Tanto la gráfica como las tablas expuestas anteriormente confirman que el tiempo de ejecución entre la versión base y la versión modernizada del código no ha sufrido ninguna variación, pudiéndose concluir que las mejoras de calidad aplicadas en la primera iteración de este estudio no han tenido ningún impacto, ni positivo ni negativo, en el rendimiento de las aplicaciones.

4.1.3. Versión «dinámica» comparada con la versión modernizada

En esta segunda sección del análisis del rendimiento se compara la versión modernizada del código con la versión «dinámica», es decir, aquella que fue analizada dinámicamente por medio de los *sanitizers* que proporciona el compilador GCC.

Se recuerda que la versión analizada dinámicamente incluye a las aplicaciones Freqmine, Netstreamcluster, Streamcluster y Swaptions, ya que fueron las únicas que presentaron fugas de memoria tras el análisis dinámico. No obstante, dado que las fugas detectadas en la aplicación Swaptions (posiblemente falsos positivos) no pudieron ser resueltas, no se ha incluido dicha aplicación en esta sección del análisis ya que su código no experimentó ninguna modificación respecto a la versión anterior. El resto de aplicaciones que no generaron ningún aviso durante el análisis dinámico tampoco han sido incluidas en este análisis.

A continuación se presentan las tablas que comparan el tiempo medio de ejecución, en función del número de hilos, de cada aplicación modificada durante la segunda iteración de este estudio de calidad:

Tabla 4.9. Rendimiento de la versión «dinámica» frente a la versión modernizada en la aplicación Freqmine

Freqmine				
Hilos	Versión	Media (s)	Desviación típica (s)	Speedup
1	Modernizada	208,9393	1,2051	1,0046
	Dinámica	207,9729	0,6998	
2	Modernizada	105,5119	0,7623	1,0036
	Dinámica	105,1318	0,5128	
4	Modernizada	54,5644	0,2224	1,0047
	Dinámica	54,3101	0,3606	
8	Modernizada	29,5946	0,4680	0,9956
	Dinámica	29,7252	0,6659	
16	Modernizada	24,1934	0,2106	1,0004
	Dinámica	24,1847	0,0588	

Tabla 4.10. Rendimiento de la versión «dinámica» frente a la versión modernizada en la aplicación Netstreamcluster

Netstreamcluster				
Hilos	Versión	Media (s)	Desviación típica (s)	Speedup
1	Modernizada	238,0967	1,7374	1,0006
	Dinámica	237,9546	0,9837	
2	Modernizada	124,3121	0,3906	1,0005
	Dinámica	124,2527	0,3163	
4	Modernizada	69,6067	0,1341	0,9985
	Dinámica	69,7127	0,2015	
8	Modernizada	52,2313	0,2519	1,0048
	Dinámica	51,9807	0,2938	
16	Modernizada	51,5977	0,9037	1,0013
	Dinámica	51,5308	0,4450	

Tabla 4.11. Rendimiento de la versión «dinámica» frente a la versión modernizada en la aplicación Streamcluster

Streamcluster				
Hilos	Versión	Media (s)	Desviación típica (s)	Speedup
1	Modernizada	238,1663	1,3714	1,0002
	Dinámica	238,1231	0,8592	
2	Modernizada	124,0719	0,2893	1,0011
	Dinámica	123,9325	0,5418	
4	Modernizada	69,1753	0,1913	0,9945
	Dinámica	69,56	0,1049	
8	Modernizada	52,0881	0,1844	1,0065
	Dinámica	51,7504	0,4546	
16	Modernizada	52,9567	0,5057	1,0014
	Dinámica	52,8835	0,1879	

Al igual que en la sección anterior, los valores mostrados en las tablas demuestran que el rendimiento no ha variado a pesar de las mejoras realizadas tras el análisis dinámico del código. Este comportamiento puede apreciarse en los tiempos medios de ejecución, los cuales son prácticamente idénticos para ambas versiones de cada aplicación sin importar el número de hilos. En consecuencia, los valores de la columna del *speedup* son prácticamente iguales a la unidad, reiterando que el rendimiento entre las versiones no ha sufrido ningún incremento ni disminución y que el impacto en dicho rendimiento de las mejoras de calidad de la segunda iteración es nulo.

De nuevo se expone gráficamente, a través de la Fig. 4.4, los tiempos medios de ejecución de la aplicación Streamcluster para las dos versiones del código, con el objetivo de ilustrar

de una forma más visual la nula variación que experimenta el rendimiento tras las mejoras de calidad de la segunda iteración.

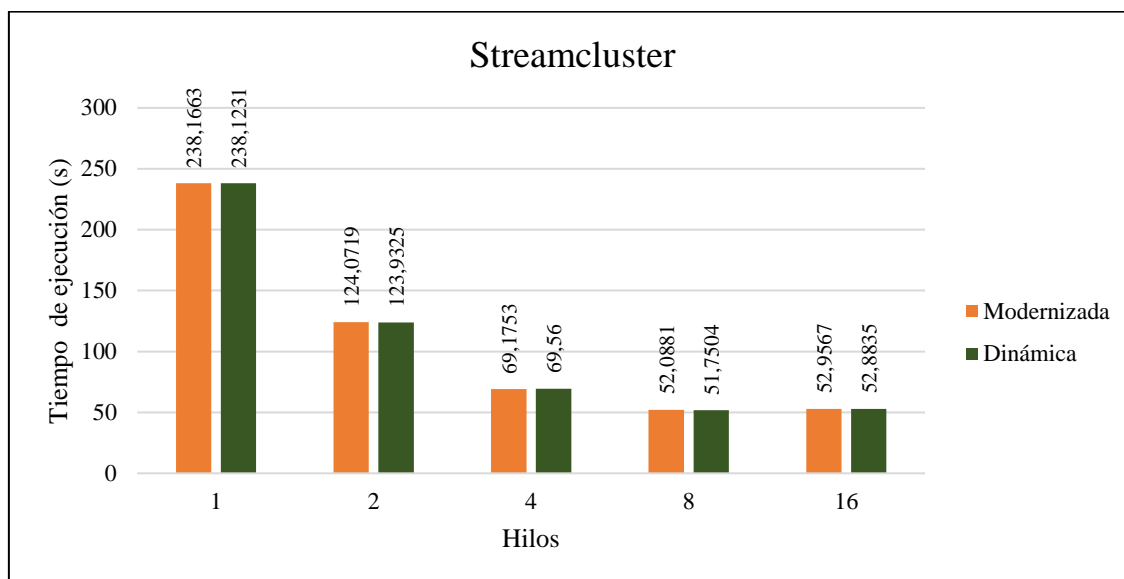


Fig. 4.4. Gráfica del tiempo de ejecución en la aplicación Streamcluster (versión modernizada frente a versión «dinámica»)

4.1.4. Versión «estática» comparada con la versión «dinámica»

En esta última sección del análisis del rendimiento se compara la versión «dinámica» del código con la versión «estática», es decir, aquella que fue analizada estáticamente por medio de los diferentes *checks* de la herramienta Clang-Tidy en la última iteración del estudio de calidad.

Se recuerda que en esta última iteración se trabajó a modo representativo con una única aplicación, Streamcluster, debido al elevado número de opciones que presentaba el análisis estático y la complejidad que suponía realizarlo sobre el resto de las aplicaciones del estudio. En consecuencia, en este apartado sólo se compara el rendimiento de dicha aplicación.

A continuación, la Tabla 4.12 refleja los tiempos de ejecución medios de cada versión del código, así como el *speedup* que relaciona ambos valores, en función del número de hilos.

Tabla 4.12. Rendimiento de la versión «estática» frente a la versión «dinámica» en la aplicación Streamcluster

Streamcluster				
Hilos	Versión	Media (s)	Desviación típica (s)	Speedup
1	Dinámica	238,1231	0,8592	1,0633
	Estática	223,9469	0,4620	
2	Dinámica	123,9325	0,5418	1,0577
	Estática	117,1706	0,3524	
4	Dinámica	69,56	0,1049	1,0624
	Estática	65,4744	0,1334	
8	Dinámica	51,7504	0,4546	1,0579
	Estática	48,9185	0,1443	
16	Dinámica	52,8835	0,1879	1,0559
	Estática	50,0857	0,4677	

Si observamos los tiempos medios de ejecución, se puede observar que, a diferencia de las dos iteraciones anteriores, las mejoras de calidad introducidas tras el análisis estático sí han tenido un impacto positivo en el rendimiento. Esto también puede apreciarse en los valores de *speedup*, todos cercanos a 1,06, los cuales denotan que el rendimiento ha mejorado en torno a un 6%, mejora que no había ocurrido en las iteraciones anteriores.

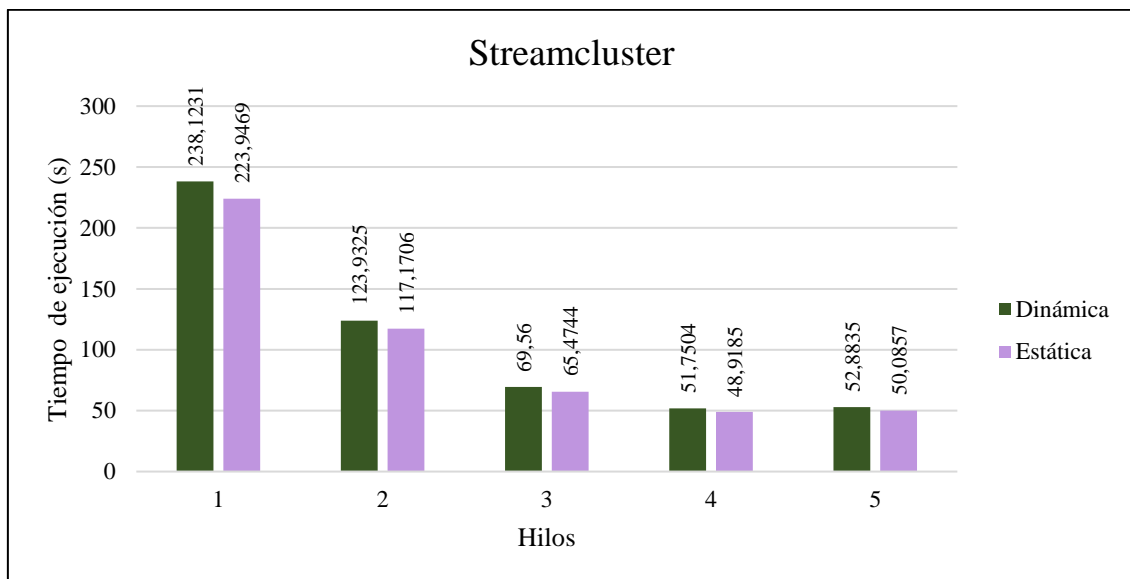


Fig. 4.5. Gráfica del tiempo de ejecución en la aplicación Streamcluster (versión «dinámica» frente a versión «estática»)

La leve pero significativa diferencia entre los tiempos de ejecución de la segunda y de la tercera iteración puede apreciarse de una forma más visual en la Fig. 4.5. Esta diferencia viene motivada principalmente por la sustitución del complejo sistema de manejo manual

de la memoria que presentaba la aplicación Streamcluster, utilizando funciones y librerías de C, por el uso de vectores, iteradores y punteros inteligentes, estructuras modernas que han permitido una notable mejora en la calidad del código a la vez que han tenido un pequeño pero positivo impacto en el rendimiento.

4.2. Análisis de la complejidad ciclomática

De forma complementaria al análisis del rendimiento, se ha decidido realizar un análisis de la complejidad ciclomática. La complejidad ciclomática, introducida por Thomas McCabe, es una métrica altamente extendida en el desarrollo de software, la cual permite medir el número de caminos independientes en una función, permitiendo predecir la complejidad lógica de un programa de una forma simple e intuitiva y detectando aquellos fragmentos de código cuya calidad es más deficiente [57].

Para realizar el análisis de la complejidad ciclomática se ha recurrido a Lizard [58], una herramienta *open software* que proporciona diferentes métricas del código, incluyendo la complejidad ciclomática. Esta herramienta se ha utilizado para estudiar la complejidad ciclomática del código fuente de las aplicaciones del *benchmark* PARSEC antes y después de las diversas mejoras de calidad que han sido aplicadas a lo largo de las diferentes iteraciones de este estudio, con el objetivo de comprobar que dichas mejoras no hayan aumentado la complejidad de las aplicaciones y que, en todo caso, la hayan disminuido.

Los resultados del análisis se recogen en la Tabla 4.13. En esta tabla se presentan, para cada aplicación y separados por la versión del código (antes y después del estudio de calidad), tres valores diferentes:

- Complejidad ciclomática máxima: valor de la complejidad ciclomática de la función con mayor número de caminos independientes.
- Complejidad ciclomática total: suma de los valores de complejidad ciclomática de todas las funciones de la aplicación.
- Complejidad ciclomática media: valor de la complejidad ciclomática total dividido por el número de funciones de la aplicación.

Tabla 4.13. Análisis de la complejidad ciclomática

Aplicación	Versión	Complejidad máxima	Complejidad total	Complejidad media
Blackscholes	Antes del estudio	8	42	3
	Después del estudio	8	42	3
Bodytrack	Antes del estudio	29	1202	2,2
	Después del estudio	29	1188	2,2
Canneal	Antes del estudio	13	204	1,7
	Después del estudio	13	204	1,7
Fluidanimate	Antes del estudio	65	1034	6,2
	Después del estudio	65	1034	6,2
Freqmine	Antes del estudio	37	355	5,4
	Después del estudio	37	355	5,4
Netstreamcluster	Antes del estudio	40	353	4,7
	Después del estudio	40	353	4,7
Streamcluster	Antes del estudio	40	369	5,1
	Después del estudio	39	366	5
Swaptions	Antes del estudio	32	189	5,6
	Después del estudio	32	189	5,6

Como puede apreciarse en la tabla, las mejoras de calidad aplicadas a lo largo de las diferentes iteraciones del estudio no han provocado prácticamente ninguna variación en la complejidad ciclomática del código fuente. Sólo la aplicación Streamcluster, modificada en las tres iteraciones del estudio, presenta una ínfima disminución en las tres métricas de complejidad. La aplicación Bodytrack, la cual experimentó un mayor número de cambios que el resto de aplicaciones del *benchmark* durante la primera iteración del estudio, también muestra una leve disminución en la complejidad ciclomática total, mientras el resto de sus métricas se mantienen invariables para ambas versiones.

5. MARCO REGULADOR

En este quinto capítulo se analiza la legislación vigente respecto a las tecnologías y herramientas utilizadas en el estudio de calidad, así como los estándares técnicos aplicables al proyecto.

5.1. Análisis legislativo

Las diversas tecnologías utilizadas a lo largo de las diferentes iteraciones de este estudio de calidad son de código abierto (*open source*), permitiendo la reproducibilidad del proyecto. A continuación se expone la revisión de las licencias del *benchmark* utilizado como modelo en este estudio, así como las licencias del resto de herramientas empleadas en el proyecto.

5.1.1. PARSEC

El *benchmark* PARSEC, distribuido como código abierto, requiere la citación de su publicación resumen [34] a la hora de publicar resultados obtenidos tras la utilización del *benchmark*. Además, trabajando con PARSEC se deben diferenciar dos tipos de licencia diferentes:

- Licencia del *framework* PARSEC.
- Licencias propias de algunas aplicaciones y librerías del *benchmark*, tanto para el código fuente como para las entradas que se emplean en dichas aplicaciones.

En primer lugar, el *framework* PARSEC está sujeto al Copyright © 2006-2009 Princeton University, todos los derechos reservados. Su licencia de tipo BSD (*Berkley Software Distribution*) permite su redistribución y su uso, tanto en su forma fuente como binaria, con o sin modificaciones, siempre que las siguientes condiciones se cumplan:

- La redistribución del código fuente debe mantener el aviso de copyright anterior, esta lista de condiciones y la posterior exención de responsabilidad.
- La redistribución en forma binaria del código debe mantener el aviso de copyright anterior, esta lista de condiciones y la posterior exención de responsabilidad.

- Ni el nombre de la Universidad de Princeton ni los nombres de sus contribuidores pueden ser utilizados para respaldar o promocionar productos derivados de este software sin un previo permiso específico por escrito.

Exención de responsabilidad:

«THIS SOFTWARE IS PROVIDED BY PRINCETON UNIVERSITY ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL PRINCETON UNIVERSITY BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE».

En segundo lugar, se exponen las licencias del código fuente y de las entradas de las aplicaciones y librerías del *benchmark* que presentan una licencia propia. Se incluye la lista completa, ya que todas estas aplicaciones y librerías han sido empleadas durante el proyecto:

- Aplicaciones con licencia propia:
 - Bodytrack emplea la licencia *Apache License Version 2.0* [59] tanto para el código fuente como para sus entradas.
 - Canneal, Facesim y Freqmine emplean una licencia de tipo BSD para el código fuente y sus entradas.
 - Ferret emplea la licencia *GNU GPL Version 3* [60] para el código fuente mientras que sus entradas son de dominio público.
 - Vips emplea la licencia *GNU GPL Version 2.1* [60] para el código fuente. Sus entradas presentan las siguientes licencias: *Creative Commons Attribution-Share Alike 2.5 Generic* [61], *GNU FDL Version 1.2* [62] y *NASA Usage Guidelines* [63].
 - x264 emplea la licencia *GNU GPL Version 2* [60] para el código fuente y la licencia *Creative Commons Public License* [64] para sus entradas.

- Librerías con licencia propia:
 - GLib, GSL, TBBLib y Yasm emplean la licencia *GNU GPL Version 2* [60].
 - Imagick emplea la licencia *Image Magick License* [65].
 - Libxml2, SSL y Zlib emplean una licencia de tipo BSD.
 - Mesa emplea diferentes licencias que pueden consultarse en [66].

5.1.2. Otras herramientas

A lo largo del proyecto se han utilizado diversas tecnologías y herramientas sujetas a diferentes licencias:

- **GCC:** el compilador GNU Compiler Collection, utilizado a lo largo del proyecto tanto para la compilación de las aplicaciones como para el análisis dinámico de las mismas, está sujeto a la licencia *GNU GPL Version 3* [60].
- **Clang-Tidy:** la herramienta Clang-Tidy, empleada durante el análisis estático de la tercera iteración del estudio, se incluye en el conjunto Extra Clang Tools del grupo LLVM Project, bajo la licencia *Apache License Version 2.0* [59].
- **FMT:** la librería FMT, empleada en la tercera iteración del estudio, está sujeta a una licencia de tipo MIT, la cual puede ser consultada en [67].
- **Lizard:** la herramienta Lizard, utilizada en el análisis de la complejidad ciclométrica, presenta una licencia de tipo MIT, la cual puede ser consultada en [68].

5.2. Estándares técnicos

En esta sección se exponen los estándares técnicos aplicados durante el estudio de calidad. Estos estándares, regulados por la Organización Internacional de Normalización (ISO) y la Comisión Electrotécnica Internacional (IEC), afectan tanto al lenguaje de programación utilizado, C++, como a la metodología empleada para el análisis de calidad desarrollado a lo largo de este estudio.

5.2.1. Estándar ISO/IEC 14882

El estándar ISO/IEC 14882 expone el conjunto de requisitos y normas que regulan las implementaciones que hacen uso del lenguaje de programación C++, así como la

definición del mismo lenguaje. Su primera versión fue publicada en el año 1998 y, desde entonces, el estándar ha experimentado numerosas revisiones, siendo la versión del año 2020 la más reciente.

En este proyecto en concreto, para la modernización del código llevada a cabo durante las diferentes iteraciones del estudio, se han seguido las normas definidas en la versión del estándar de 2017 [27], también conocida como C++17.

5.2.2. Estándar ISO/IEC 25010

El estándar ISO/IEC 25010, incluido dentro del conjunto de normas ISO/IEC 25000, también conocido como SQuaRE (*System and software Quality Requirements and Evaluation*), define el modelo de evaluación de la calidad del software.

Tal y como se explicó en el segundo capítulo de esta memoria, este modelo de evaluación divide la calidad en ocho características diferentes, las cuales a su vez se dividen en diferentes propiedades.



Fig. 5.1. Modelo ISO/IEC 25010 [69]

El modelo de evaluación de la última revisión del estándar ISO/IEC 25010, llevada a cabo en el año 2011 [9], es el que se ha empleado para analizar la calidad del software de las aplicaciones que componen el *benchmark* PARSEC.

6. ENTORNO SOCIOECONÓMICO

En este sexto capítulo se detalla la planificación por tareas del proyecto y el presupuesto estimado del mismo. También se explica el potencial impacto económico y social de este estudio de calidad.

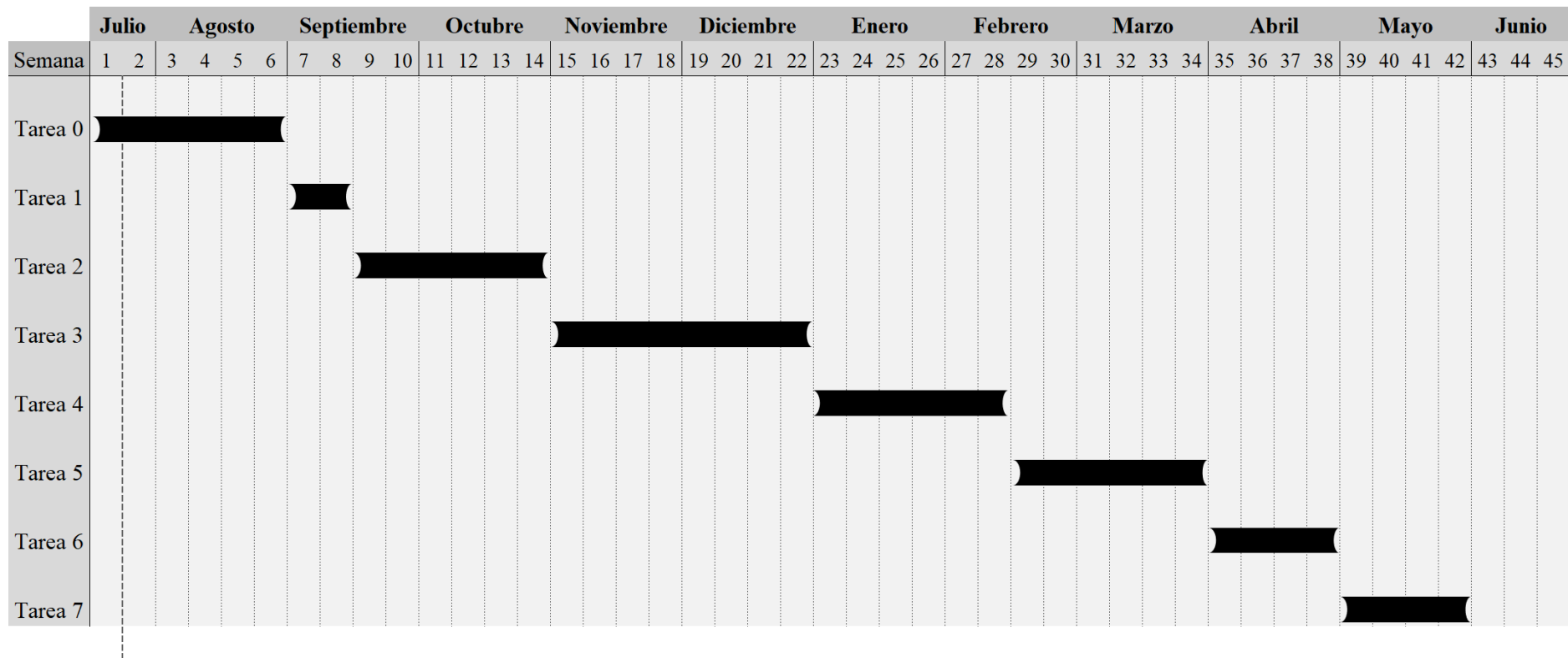
6.1. Planificación

Desde su comienzo, este proyecto fue dividido en diferentes tareas con el objetivo de simplificar su planificación y facilitar su desarrollo. Se estimó un tiempo determinado para cada una de las tareas, y dichos tiempos quedaron recogidos en un diagrama de Gantt inicial (Fig. 6.1). Posteriormente, al final del proyecto, se ha desarrollado un nuevo diagrama exponiendo la duración real de cada tarea (Fig. 6.2).

A continuación se definen las diferentes tareas en las que se dividió el proyecto:

- **Tarea 0:** trabajo previo. Durante esta tarea se debían adquirir los niveles de conocimiento necesarios sobre el lenguaje de programación C++. Su duración se estimó en 6 semanas y se dividía en las siguientes subtareas:
 - Lectura de libros y artículos sobre el lenguaje C++.
 - Ejercicios prácticos.
- **Tarea 1:** preparación del entorno. Esta tarea consistía en la instalación del sistema operativo y del compilador que iba utilizarse durante el proyecto. Su duración se estimó en 2 semanas y se dividía en las siguientes subtareas:
 - Instalación del sistema operativo Ubuntu 20.04.
 - Actualización del compilador GCC a la versión 10.2.
- **Tarea 2:** instalación del *benchmark* PARSEC. Esta tarea consistía en la descarga e instalación del *benchmark* elegido como modelo para el estudio de calidad. Durante esta tarea se debían resolver los diferentes errores que impidiesen la compilación de las diferentes aplicaciones que componen el *benchmark*. Su duración se estimó en 6 semanas y se dividía en las siguientes subtareas:
 - Descarga del *benchmark* PARSEC
 - Compilación de las aplicaciones del *benchmark* PARSEC.
 - Resolución de los errores que eviten la compilación.

- **Tarea 3:** modernización del código. Esta tarea era la primera iteración del estudio de calidad, donde se analizarían los avisos generados al compilar las aplicaciones del *benchmark*. Su duración se estimó en 8 semanas y se dividía en las siguientes subtareas:
 - Recopilación de los avisos de compilación.
 - Análisis de los avisos de compilación.
 - Resolución de los avisos de compilación.
- **Tarea 4:** análisis dinámico. Esta tarea era la segunda iteración del estudio de calidad, donde se corregirían los errores detectados tras el análisis dinámico de las aplicaciones del *benchmark*. Su duración se estimó en 6 semanas y se dividía en las siguientes subtareas:
 - Elección de la herramienta para llevar a cabo el análisis dinámico.
 - Realización del análisis dinámico del código
 - Solución de los defectos detectados por medio del análisis dinámico.
- **Tarea 5:** análisis estático. Esta tarea era la tercera y última iteración del estudio de calidad, donde se estudiarían y resolverían los fallos de calidad detectados tras el análisis estático del *benchmark*. Su duración se estimó en 6 semanas y se dividía en las siguientes subtareas:
 - Elección de la herramienta para llevar a cabo el análisis estático.
 - Realización del análisis estático del código
 - Solución de los defectos detectados por medio del análisis estático.
- **Tarea 6:** evaluación de los resultados. Durante esta tarea debía realizarse el análisis del rendimiento de las aplicaciones para estudiar el impacto de las mejoras de la calidad, así como el análisis de la complejidad ciclomática. Su duración se estimó en 4 semanas y se dividía en las siguientes subtareas:
 - Preparación de la experimentación.
 - Recopilación de los tiempos de ejecución.
 - Elección de la herramienta para llevar a cabo el análisis de la complejidad ciclomática.
 - Realización del análisis de la complejidad ciclomática.
- **Tarea 7:** documentación. Esta tarea consistía en la redacción de la memoria del proyecto. Su duración se estimó en 4 semanas.



Día en el que se estimó la planificación

Fig. 6.1. Diagrama de Gantt: planificación estimada

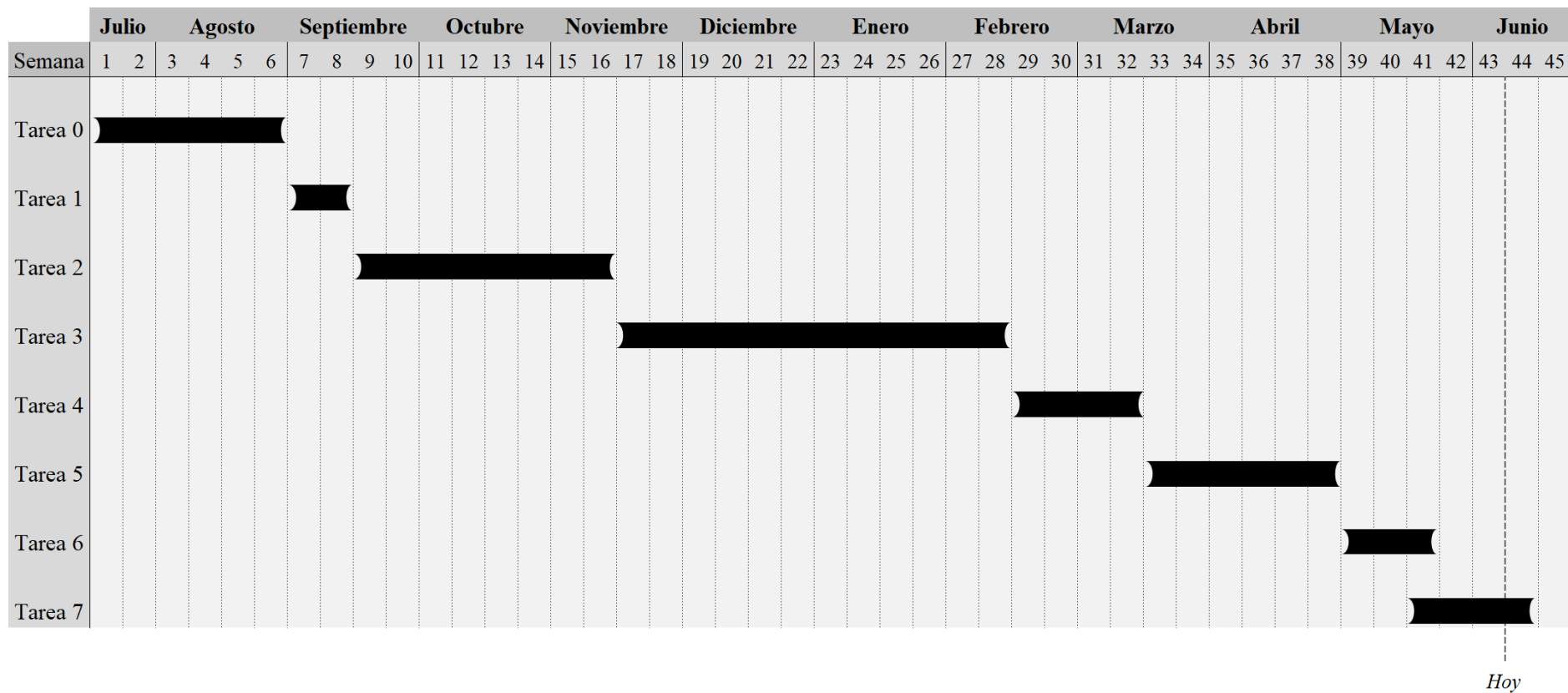


Fig. 6.2. Diagrama de Gannt: planificación real

Tal y como puede apreciarse en el primer diagrama de Gantt, se estimaba haber terminado el proyecto antes del mes de junio de 2021. No obstante, algunos obstáculos impidieron cumplir con la planificación inicial.

La duración de la tarea 2 (instalación del benchmark PARSEC) y, en especial, de la tarea 3 (modernización del código) se extendieron más de lo previsto. Esta situación estuvo motivada principalmente por la dificultad para compaginar el desarrollo del proyecto con el resto de labores académicas, subestimadas a la hora de realizar el primer diagrama de Gantt. Entre los meses de octubre y enero, las prácticas curriculares y los exámenes finales ralentizaron considerablemente el estudio, lo que provocó esta descoordinación respecto a la planificación inicial.

No obstante, la tarea 4 (análisis dinámico) duró dos semanas menos de lo previsto, y la tarea 6 (evaluación de los resultados) también pudo completarse en menos tiempo del estimado. Gracias a estos avances se logró terminar el proyecto con tan solo dos semanas de retraso respecto a la planificación inicial.

6.2. Presupuesto

Para la estimación del presupuesto se han tenido en cuenta los gastos en recursos humanos, los gastos en recursos materiales y los costes indirectos del proyecto. La suma de estos gastos da lugar al coste total del mismo.

6.2.1. Recursos humanos

El coste estimado de los recursos humanos hace referencia al gasto económico que supondría la contratación del personal necesario para llevar a cabo el proyecto: un ingeniero técnico y un jefe de proyecto. Para estimar la dedicación del ingeniero técnico se ha considerado el número de semanas totales del proyecto (44 semanas, tal y como puede verse en la sección de la planificación) y la media de horas semanales dedicadas (aproximadamente 17,5 horas), lo que suma un total de 770 horas. La dedicación del jefe de proyecto se ha estimado en un 5% de la dedicación del ingeniero técnico. El coste por hora se ha estimado a partir de las tablas presupuestarias de la Universidad Carlos III de Madrid en el año 2020, las cuales pueden ser consultadas en [70]. La Tabla 6.1 recoge las cifras mencionadas.

Tabla 6.1. Costes de los recursos humanos

Nombre y apellidos	Categoría	Dedicación (horas)	€/hora	Coste (€)
Alberto García García	Ingeniero técnico	770	15	11550
José Daniel García Sánchez	Jefe de proyecto	38,5	50	1925
Total:				13475

6.2.2. Recursos materiales

El coste estimado de los recursos materiales hace referencia al gasto económico de los elementos materiales empleados durante la realización del proyecto, tales como el software y el hardware.

Respecto al software, todas las tecnologías y herramientas utilizadas en el estudio (PARSEC, GCC, Clang-Tidy, etc.) son de código abierto, tal y como se explicó en el quinto capítulo de este documento. Además, si bien es cierto que se ha utilizado Microsoft Word para la redacción de la memoria, esta herramienta cuenta con una versión gratuita para estudiantes. En consecuencia, el software empleado no implica ningún coste para el presupuesto del proyecto.

Respecto al hardware, se debe considerar tanto el coste del ordenador de sobremesa como el portátil personal utilizado para llevar a cabo este proyecto (se estima una vida útil de 5 años para ambos dispositivos, siguiendo el periodo de amortización estándar de la Universidad Carlos III de Madrid). Dado que ambas máquinas fueron adquiridas para el proyecto, se considera que su porcentaje de uso dedicado al proyecto es del 100%. La Tabla 6.2 expone los costes en hardware del proyecto.

Tabla 6.2. Costes del hardware

Concepto	Coste (€)	Vida útil (meses)	Uso (meses)	Coste amortizado (€)
Ordenador de sobremesa	2166	60	4	144,40
Portátil personal	699	60	11	128,15
Total:				272,55

Para el cálculo de la amortización se ha utilizado la siguiente fórmula:

$$\text{coste amortizado} = \frac{\text{coste}}{\text{vida útil}} * \text{uso}$$

Fig. 6.3. Fórmula del coste amortizado

6.2.3. Costes indirectos

Los costes indirectos estimados hacen referencia a los gastos en luz, agua, internet y consumibles del proyecto. Siguiendo el ejemplo de uno de los proyectos de investigación del Departamento de Informática de la Universidad Carlos III de Madrid [71], los costes indirectos se estiman en un 20% de los costes directos (costes de los recursos humanos y de los recursos materiales). La Tabla 6.3 recoge las cifras mencionadas.

Tabla 6.3. Costes indirectos

Costes directos (€)	Tasa de costes indirectos (%)	Costes indirectos (€)
13747,55	20	2749,51

6.2.4. Coste total

A continuación, la Tabla 6.4 desglosa los costes totales del proyecto:

Tabla 6.4. Coste total

Concepto	Coste (€)
Recursos humanos	13475
Software	0
Hardware	272,55
Costes indirectos	2749,51
Total (sin IVA)	16497,06
IVA (21%)	3464,38
Total	19961,44

6.3. Impacto socioeconómico

La investigación llevada a cabo en este proyecto puede tener un importante impacto en el desarrollo de software, especialmente en el ámbito científico en el que estaba orientado el estudio. La demostración experimental de que calidad y rendimiento no son propiedades excluyentes puede contribuir al desarrollo de programas y aplicaciones más seguras y portables, con códigos más fáciles de mantener y actualizar, lo que implica importantes beneficios.

En primer lugar, la calidad del software puede reportar notables ventajas económicas, las cuales ya son conocidas en el entorno del software comercial. El coste que supone asegurar la calidad del código desde el principio del desarrollo es considerablemente

menor que el gasto de solventar un defecto del producto de software detectado durante su uso o aplicación.

Además, la calidad del código incrementa notablemente su legibilidad y, por tanto, facilita su mantenimiento y actualización. La simplificación de estos procesos no solo permite abaratar su coste, sino que evita que muchos de los programas y aplicaciones que estaban condenados a terminar obsoletos puedan continuar su ciclo de vida.

Por otro lado, se considera que este proyecto puede tener también cierto impacto en la educación. La calidad del software es una propiedad generalmente subestimada y es necesario cambiar esta visión desde la raíz. Las clases y cursos de programación, ya sean dados en colegios, institutos, academias o universidades, deben insistir en la importancia de esta propiedad. Se considera que este proyecto es un buen aliciente para impulsar la difusión de las ventajas de la calidad del software en las instituciones y centros educativos.

Este proyecto de investigación no ha dado lugar a una aplicación comercial ni generará ingresos económicos. No obstante, se espera de él que cambie la visión de la comunidad sobre la importancia de la calidad en el desarrollo de software.

7. CONCLUSIONES

En este séptimo capítulo se revisan los objetivos cumplidos a lo largo del proyecto. También se exponen las potenciales líneas de trabajo futuras que nacen de este estudio de la calidad.

7.1. Objetivos cumplidos

El principal objetivo de este proyecto era estudiar el impacto de la calidad del software sobre el rendimiento del programa. El interés por llevar a cabo esta investigación nacía de la necesidad de comprobar si la irregular calidad de muchas aplicaciones de software científico, como las que conforman el *benchmark* PARSEC, estaba justificada por la necesidad de un alto rendimiento.

A lo largo del estudio se han llevado a cabo tres iteraciones a través de las cuales se han realizado centenares de mejoras de calidad en diferentes aplicaciones del *benchmark* PARSEC. Tras evaluar el rendimiento entre el código base y el código modificado después de cada iteración, en ninguna de las aplicaciones se ha observado una disminución del rendimiento y, de hecho, tras las mejoras aplicadas en la tercera iteración, sí se podía apreciar un ligero aumento del mismo.

Sin embargo, este estudio no sólo ha permitido comprobar que una alta calidad en el código no implica un peor rendimiento del software. También se han podido detectar los errores más habituales cometidos en el lenguaje C++. Si bien es cierto que los errores solventados en cada iteración eran diferentes, muchos de ellos compartían el mismo origen.

En la primera iteración, muchos de los avisos de compilación observados se debían a errores en el uso de funciones de formateo con un número variable de argumentos (como *printf*), propias del lenguaje C. Más tarde, en la tercera iteración, las convenciones empleadas durante el análisis estático también desaconsejaban el uso de este tipo de funciones. Claramente el uso de estas funciones es propenso a errores por su incapacidad para proporcionar seguridad de tipos.

Otra de las prácticas que más defectos provocaba en el código era el manejo manual de la memoria. Esto es especialmente notable en la segunda iteración, donde se detectaron

decenas de fugas de memoria; y en la tercera, donde diferentes *checks* recomendaban encarecidamente la sustitución de las funciones de manejo manual de la memoria y la aritmética de punteros por contenedores de C++, iteradores y punteros inteligentes. El uso de estas estructuras de C++ no solo simplifica la labor del desarrollador, el cual no tiene que preocuparse por las arduas tareas de liberar o reservar memoria, sino que además, tal y como se ha comprobado en este estudio, puede implicar un rendimiento ligeramente superior.

7.2. Futuras líneas de trabajo

La investigación llevada a cabo en este proyecto ha demostrado que la calidad y el rendimiento no son cualidades incompatibles. A partir de este estudio, se abren diferentes líneas de trabajo que podrían aumentar la profundidad y amplitud de la investigación. Se proponen como trabajos futuros:

- Modernizar las aplicaciones del *benchmark* PARSEC que estaban programadas en C, las cuales han sido apartadas de este estudio, y reescribirlas en C++. Este estudio es de gran interés ya que podría permitir evaluar las diferencias de rendimiento entre ambos lenguajes.
- Analizar estáticamente el *benchmark* completo por medio de la herramienta Clang-Tidy. Debido a las limitaciones del proyecto, solamente se pudo analizar una de las aplicaciones del *benchmark* con esta herramienta y dicha aplicación presentó un rendimiento ligeramente superior. Llevar a cabo un análisis del *benchmark* completo con dicha herramienta podría demostrar los beneficios de la misma.
- Reescribir por completo la aplicación Bodytrack, la cual utiliza un sistema propio de concurrencia. En la primera iteración, el sistema de concurrencia propio de Bodytrack dificultó su modernización, por lo que sería interesante comprobar el efecto de sustituir ese complejo sistema de concurrencia por el sistema de hilos de C++ y analizar las diferencias en rendimiento.

8. ENGLISH SUMMARY

8.1. Introduction

Code quality is one of the most underrated properties in software development. Despite having achieved a certain degree of importance in recent years, thanks to the economic benefits reported in commercial software production, this characteristic is often outshined by other more perceptible ones.

This situation is especially remarkable in the scientific field, where the lack of software quality is usually justified with the need of performance, as if both properties were not compatible. The aim of this study is to verify this claim.

The interest behind assessing the impact of quality in scientific software performance is because, in spite of being ignored, code quality is one of the best inversions in software development. It not only avoids potential defects in mature stages of the development, but it also makes the code easier to maintain and to keep updated.

Nonetheless, testing if quality and performance are compatible properties is not the only goal of this project. Parallel to this evaluation, an in-depth analysis of the most common code errors in C++ scientific applications will be carried out.

Among other well-known programming languages like Java or Python, C++ is the favorite one in the scientific community thanks to its incredible performance. This programming language is flexible and able to adapt to any application, always providing better performance than any other high-level language.

However, C++ has an important downside. This programming language allows certain risky behaviors like manually managing dynamic memory, performing bit operations or using some old and unsafe C functions. Although these practices might not be inherently dangerous, their incorrect use may produce severe defects in the development of the programs. Therefore, it is essential to study the language thoroughly in order to be able to safely exploit all its capabilities.

In order to achieve both objectives, the PARSEC benchmark suite, a benchmark for multiprocessor computers evaluation, has been selected as model of scientific software.

During this project, the code of this benchmark will be iteratively analyzed to detect potential quality errors, solve them and assess the impact of these modifications.

8.2. State of the Art

In this section it will be reviewed the literature preceding this study and the standard dimensions of software quality which will be used in this project to analyze the code defects of the PARSEC benchmark. It also includes a summary of the C++ programming language history and a brief introduction to benchmarks.

8.2.1. Literature Review

During the last two decades software quality has become a frequent matter in scientific literature. However, most of these publications, see [1, 2, 3, 4], study the economic benefits of software quality and how to achieve them, without assessing the effects of this software quality in the applications performance.

If the search domain is filtered in order to find publications that actually evaluate the impact of software quality, it can be seen that most articles, as in [5], are focused only on commercial software, again motivated by the economic advantages. Finding studies that target scientific software, that is, the one used in programs and systems designed to solve research and engineering problems, is relatively more difficult, but not impossible.

In 2005 Raghavan, Irwin, McInnes and Norris published an interesting paper where the balance between quality, performance and power consumption in scientific computation was evaluated [6]. In this study Raghavan et al. defend that high-performance applications, like the ones used in scientific environments, can achieve certain quality and performance requirements at the same time its power consumption is decreased through voltage dynamic scaling techniques. Although this study focuses on power consumption optimization, the publication suggests that this optimization is required for quality and performance to be compatible characteristics in high-performance computing, as if, under normal conditions, one inevitably detracts from the other.

8.2.2. Standard Quality Dimensions

The definition of quality may be quite abstract in software development. Generally, the quality of a program is analyzed as if it were an atomic property of the software, or, at

most, studying just some characteristics of the code, such as its readability or its modularity.

In order to avoid these divergences and to normalize the concept of software quality, the International Organization for Standardization (ISO), together with the International Electrotechnical Commission (IEC), defined the ISO/IEC 25000 [8], a set of standards also known as SQuaRE (System and software Quality Requirements and Evaluation). Within this set it can be found the ISO/IEC 25010 standard [9], which establishes the dimensions that compose the software quality: functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability and portability.

8.2.3. The C++ History

Just as Bjarne Stroustrup (the C++ author) tells in his publications [11, 12, 13], C++ was born aiming the combination of performance (as C language provides) and programming easiness (as Simula language provides). This idea was conceived during the realization of his PhD thesis in Cambridge University, while developing a complex simulator, where Bjarne had to choose between using a comfortable high-level programming language with poor performance or a complex low-level one but with really good performance.

From that moment, Bjarne started thinking about developing a new programming language. First known as C with Classes, and later C++, the programming language developed by Bjarne was designed to be used in as many domains as possible rather than in specific applications.

The C++ programming language was a complete success which quickly become international. This led to the standardization of the language under the auspices of the International Organization for Standardization, in 1998. From that moment, the C++ standard has experienced several revisions, being the last one in 2020.

Nowadays, even though it may not be noticed by the final user, C++ can be found in any field: telecommunications, finance, videogames, animation, web applications, artificial intelligence, etc. This endless list shows the deep and wide extension of C++.

8.2.4. Benchmarks

A benchmark is a reference point, a value to compare. In the field of computer science, benchmarks are applications designed to evaluate different characteristics of both

software and hardware, such as performance or resources consumption. One example of benchmark is the PARSEC benchmark suite, used as model in this software quality study.

The PARSEC (Princeton Application Repository for Shared-Memory Computers) benchmark suite is the solution that Princeton University developed to face the evaluation of current multiprocessor with more representative loads of the emerging programs [30].

In its last version, PARSEC 3.0, the core of the benchmark is composed of 13 multithreaded applications: Blackscholes, Bodytrack, Canneal, Dedup, Facesim, Ferret, Fluidanimate, Freqmine, Raytrace, Streamcluster, Swaptions, Vips and x264. It also contains three network applications (Netdedup, Netferret and Netstreamcluster) and two additional benchmarks, SPLASH-2.0 and SPLASH-2x, focused on high-performance computing.

8.3. Quality Study

The study of the code quality in the PARSEC benchmark applications has been done in an iterative way. This section aims to explain the analysis carried on each of the three iterations performed, as well as the results obtained in each iteration.

It must be said that the compilation was done using the C++17 standard and the GCC compiler (version 10.2). Furthermore, even though SPLASH-2 and SPLASH-2X are part of the PARSEC 3.0 suite, both benchmarks have been excluded from the study since they are programmed in C instead of C++. Other applications will also be set aside through the study for this reason. Despite these modifications, the breadth and the depth of this project are considered adequate to meet its objectives.

8.3.1. Source Code Modernization

The goal of the first iteration of the study was to compile the benchmark applications without warnings from the compiler. Therefore, the first step was to solve every compilation error that was preventing the compilation.

Different errors were found during this first analysis, like dynamic exception declarations (Fig. 8.1) in the Bodytrack application, which had to be removed, or certain implicit conversions not accepted by the compiler (Fig. 8.2) in the Facesim application, which were substituted by explicit casts (Fig. 8.3).

```

60. public:
61.     Thread(Runnable &) throw(ThreadCreationException);
62.
63.     //Wait until Thread object has finished
64.     void Join();
65. };
66.

```

Fig. 8.1. Dynamic exception declaration (Thread.h file)

```

98.
99. bool Directory_Exists (const std::string& dirname)
100. {
101.     return std::ifstream (dirname.c_str()) != 0;
102. }
103.

```

Fig. 8.2. Implicit conversion (FILE_UTILITIES.cpp file)

```

104.
105. bool Directory_Exists (const std::string& dirname)
106. {
107.     return static_cast<bool>(std::ifstream (dirname.c_str()));
108. }
109.

```

Fig. 8.3. Explicit cast (FILE_UTILITIES.cpp file)

After obtaining a base version of the source code capable of compiling without errors, the warnings generated during the compilation were gathered in order to be analyzed and solved (all the compilation warnings can be consulted in the Annex A.1). After gathering all these compilation warnings, the applications presenting warnings in C-written files were removed from the study. The applications kept are Blackscholes, Bodytrack, Canneal, Fluidanimate, Freqmine, Netstreamcluster, Streamcluster and Swaptions.

Most of the warnings generated during the compilation of these applications were due to errors in formatting functions with variable number of arguments, as *printf*. For example, the argument of the function might be waiting to receive a variable of certain type, yet it receives another type. See Fig. 8.4, where %u expects an unsigned int but the type of the argument received is long unsigned int. Depending on the architecture where the program is executed, the value of the argument received may overflow the capacity of the awaited type, producing unexpected behavior, decreasing the reliability of the application.

```

1824.
1825. fprintf(fp, "%u\n", centerIDs[i]);
1826.

```

Fig. 8.4. Formatting function error (streamcluster.cpp, Streamcluster)

In total, 12 different warnings (produced by around 65 code defects) were solved during this iteration. Nevertheless, the impact of these quality improvements over the performance was null. The application Streamcluster, which is the only application kept in every iteration of the study, has been chosen to illustrate this situation. Table 8.1 shows the performance comparison between the baseline source code of the application and the modernized one.

Table 8.1. Performance of baseline version vs modernized version in Streamcluster application

Streamcluster				
Threads	Version	Mean (s)	Standard Deviation (s)	Speedup
1	Baseline	237,9539	1,489451536	0,999108186
	Modernized	238,1663	1,371354971	
2	Baseline	124,2905	0,455708789	1,001761882
	Modernized	124,0719	0,289302014	
4	Baseline	69,4926	0,071941643	1,004586897
	Modernized	69,1753	0,191278767	
8	Baseline	52,1121	0,10189041	1,000460758
	Modernized	52,0881	0,184412852	
16	Baseline	53,1751	0,524018755	1,004124124
	Modernized	52,9567	0,505705679	

The last column, speedup, is the quotient between the old execution time and the new one. Values really close to the unit, as the ones displayed in the table, imply that both execution times are almost identical. Fig. 8.5. shows this in a more graphical way.

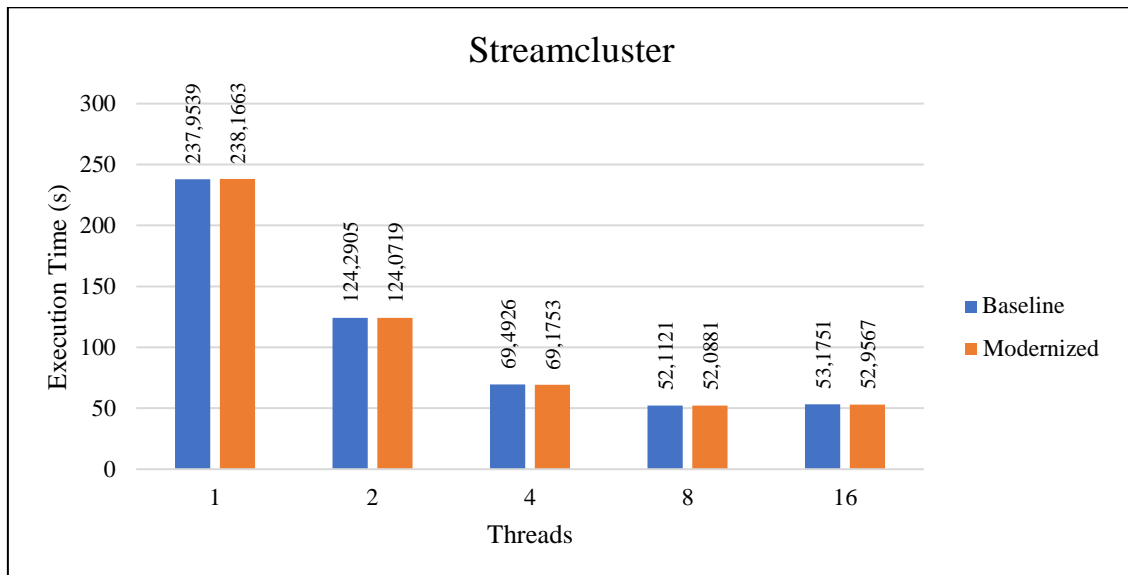


Fig. 8.5. Streamcluster execution times (first iteration)

8.3.2. Dynamic Analysis of the Source Code

During the second iteration of the study, a dynamic analysis of the source code was carried on. This type of analysis allows to assess the application behavior while its execution. In this way, it is possible to evaluate different metrics which cannot be statically analyzed, such as memory leaks or race conditions in multithreaded executions.

In order to perform the dynamic analysis of the benchmark applications, sanitizers provided by the GCC compiler were used. GCC sanitizers are a compilation option which enables code analysis during runtime, allowing the evaluation of different parameters while the code is being executed.

There are different GCC sanitizers. The full list can be seen in [52]. For this study, the options “-fsanitize=undefined” and “-fsanitize=address” were chosen, since these two sanitizers cover most of the available options. The undefined option detects potential undefined behavior in the code while the address option looks for memory leaks and other errors in memory management.

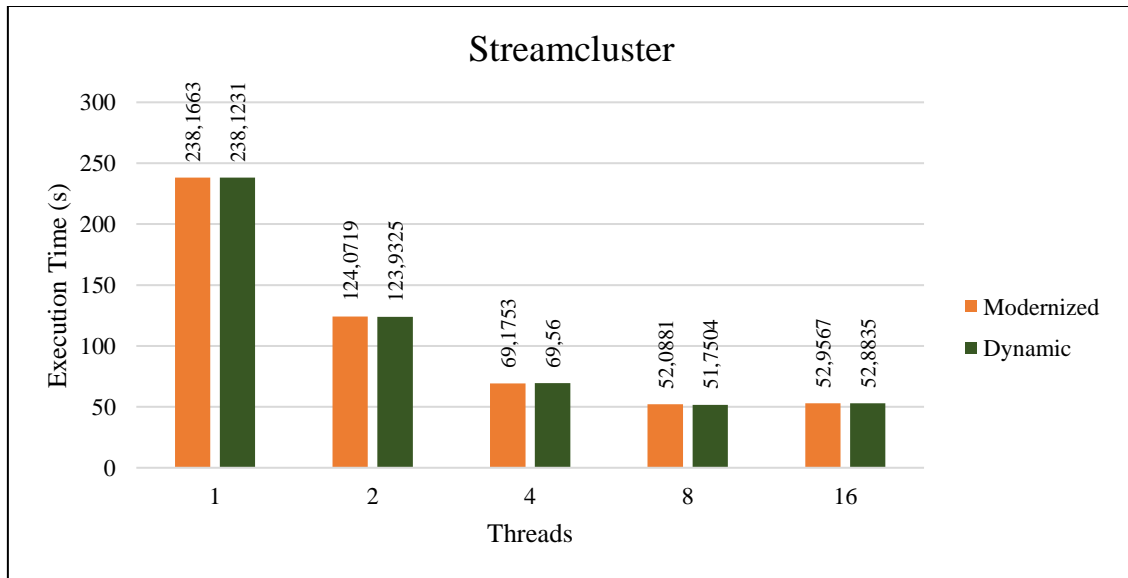
After performing the dynamic analysis, the undefined sanitizer did not find any potential undefined behavior in the code. Nonetheless, the address sanitizer did find several memory leaks in four different applications: Freqmine, Netstreamcluster, Streamcluster and Swaptions. In total, 31 memory leaks were detected by the sanitizer, of which only 22 could be solved (almost 71% of the total leaks) by freeing the memory that was being reserved in the code but not relieved. The rest of the leaks could not be solved because they would need really complex changes in the applications architecture or because they were false positives.

Despite solving the different memory leaks detected during the runtime analysis, once again the impact of these improvements over the applications performance was non-existent. The execution times of this “dynamic” version of the code were practically the same as those of the modernized version from the first iteration. The Streamcluster application has been selected to illustrate this situation once more, as it can be seen in Table 8.2.

Table 8.2. Performance of modernized version vs “dynamic” version in Streamcluster application

Streamcluster				
Threads	Version	Mean (s)	Standard Deviation (s)	Speedup
1	Modernized	238,1663	1,3714	1,0002
	Dynamic	238,1231	0,8592	
2	Modernized	124,0719	0,2893	1,0011
	Dynamic	123,9325	0,5418	
4	Modernized	69,1753	0,1913	0,9945
	Dynamic	69,56	0,1049	
8	Modernized	52,0881	0,1844	1,0065
	Dynamic	51,7504	0,4546	
16	Modernized	52,9567	0,5057	1,0014
	Dynamic	52,8835	0,1879	

As it can be seen in the table, the execution times of both versions are almost identical. Therefore, the speedup values are extremely close to the unit. This can also be appreciated in a more visual way in Fig. 8.6.

**Fig. 8.6.** Streamcluster execution times (second iteration)

8.3.3. Static Analysis of the Source Code

In order to discover more quality defects in the code, a static analysis of the PARSEC benchmark has been performed during the third iteration. The static analysis consists in the evaluation of the source code through an automatic tool without executing it, which allows to detect potential errors in memory management, invalid array accesses or simple legibility flaws.

Between the different tools available to carry on the static analysis of the code, Clang-Tidy was the chosen one for this project. This tool, provided by the Clang compiler, displays a considerable number of different checks, which are gathered in categories. The full list of categories, as well as the different checks they contain, can be consulted in [54].

For this study, the following categories were selected: Abseil (17 checks), Bugprone (48 checks), Clang-Analyzer (55 checks), CppCoreGuidelines (24 checks), Google (22 checks), Hicpp (30 checks), Misc (13 checks), Modernize (30 checks), Openmp (2 checks), Performance (14 checks), Portability (1 checks) and Readability (39 checks). Due to the high number of checks kept for the project and the complexity it supposes to analyze and solve the errors detected by these checks, it was decided to analyze only one application of the benchmark, the Streamcluster application.

The static analysis of the application presented many different quality warnings. Nevertheless, the most remarkable ones were related to manual memory management. After substituting the old C functions for memory management (Fig. 8.7), as well as the pointer arithmetic operations, with the use of C++ vectors, iterators and smart pointers (Fig. 8.8), the code was not only easier to read, to maintain and to update, but also showed a slightly better performance.

```
2024. PStream* stream;
2025. if( n > 0 ) {
2026.     stream = new SimStream(n);
2027. }
2028. else {
2029.     stream = new FileStream(infile);
2030. }

    /* ... */

2037. delete stream;
```

Fig. 8.7. Manual memory management (streamcluster.cpp)

```
2024. unique_ptr<PStream> stream;
2025. if( n > 0 ) {
2026.     stream = make_unique<SimStream>(n);
2027. }
2028. else {
2029.     stream = make_unique<FileStream>(infile);
2030. }
```

Fig. 8.8. Smart pointers solution (streamcluster.cpp)

The small but positive impact of all these improvements can be appreciated in Table 8.3.

Table 8.3. Performance of “dynamic” version vs “static” version in Streamcluster application

Streamcluster				
Threads	Version	Mean (s)	Standard Deviation (s)	Speedup
1	Dynamic	238,1231	0,8592	1,0633
	Static	223,9469	0,4620	
2	Dynamic	123,9325	0,5418	1,0577
	Static	117,1706	0,3524	
4	Dynamic	69,56	0,1049	1,0624
	Static	65,4744	0,1334	
8	Dynamic	51,7504	0,4546	1,0579
	Static	48,9185	0,1443	
16	Dynamic	52,8835	0,1879	1,0559
	Static	50,0857	0,4677	

The mean execution times of the “static” version are lower than the ones from the previous version. In consequence, the speedup values of this third iteration are close to 1.06, the best results in performance of all the study. For the first time in the project, the quality improvements present an impact over the application performance. This slight performance improvement is graphically displayed in Fig. 8.9.

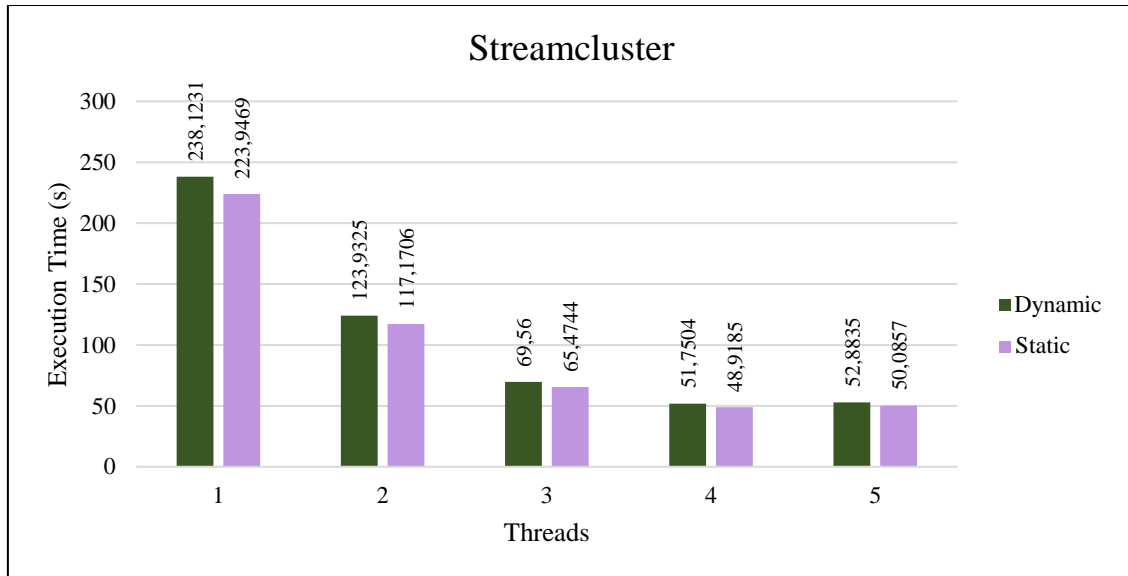


Fig. 8.9. Streamcluster execution times (third iteration)

8.4. Conclusions and future work

The main goal of this project was to study the impact of software quality in the program performance, in order to check if the lack of quality that many scientific software applications present, like the ones of the PARSEC benchmark suite, was justified by the need of high performance.

Along the study, three iterations have been carried out through which hundreds of quality improvements have been made in different applications of the PARSEC benchmark. After evaluating the performance between the base code and the modified code in each iteration, none of the applications have presented a decrease in its performance. In fact, after the improvements applied in the third iteration, a slight increase could be seen in the program performance.

However, this study has not only allowed to verify that higher quality code does not imply a worse performance of the software. It has also been possible to detect the most common mistakes in the C++ programming language. Although it is true that the errors found in each iteration were different, many of them shared the same origin, like the manual memory management.

Furthermore, the goals achieved in this project have also opened new lines of work which could increase the depth and the breadth of this investigation. The following ideas are proposed:

- To modernize and rewrite in C++ the PARSEC applications which were programmed in C, in order to assess the performance differences between both programming languages.
- To perform a static analysis of the whole PARSEC benchmark, in order to check if the benefits obtained in the Streamcluster application can also be found by statically analyzing the rest of the applications.
- To rewrite the Bodytrack application which uses its own concurrency system. This concurrency system made it difficult to modernize the application in the first iteration of the study, so it would be interesting to change that complex concurrency system by the C++ thread system.

9. BIBLIOGRAFÍA

- [1] O. Alshathry, H. Janicke, H. Zedan y A. Alhussein, «Quantitative Quality Assurance Approach,» de *2009 International Conference on New Trends in Information and Service Science*, 2009.
- [2] S. Ibarra y M. Muñoz, «Support tool for software quality assurance in software development,» de *2018 7th International Conference On Software Process Improvement (CIMPS)*, 2018.
- [3] M. Menzies y J. Hihn, «Evidence-based cost estimation better-quality for software,» *IEEE Software*, vol. 23, pp. 64-66, 2006.
- [4] S. Slaughter, D. Harter y M. Krishnan, «Evaluating the Cost of Software Quality,» *Commun. ACM*, vol. 41, pp. 67-73, 8 1998.
- [5] U. B. Corrêa, L. F. G. Millani, A. C. S. Beck y L. Carro, «Quality Impact on Software Performance,» de *2013 III Brazilian Symposium on Computing Systems Engineering*, 2013.
- [6] R. Raghavan, M. J. Irwin, L. C. McInnes y B. Norris, «Adaptive software for scientific computing: co-managing quality-performance-power tradeoffs,» de *19th IEEE International Parallel and Distributed Processing Symposium*, 2005.
- [7] Real Academia Española, «Diccionario de la lengua española, 23.^a ed.,» [En línea]. Available: <https://dle.rae.es>. [Último acceso: 3 de Mayo de 2021].
- [8] «Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Guide to SQuaRE,» ISO/IEC 25000:2014, International Organization for Standardization, Ginebra, Suiza, 2014.
- [9] «Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models,» ISO/IEC 25010:2011, International Organization for Standardization, Ginebra, Suiza, 2011.

- [10] «TIOBE Index: Very Long Term History,» [En línea]. Available: <https://www.tiobe.com/tiobe-index/>. [Último acceso: 30 de Abril de 2021].
- [11] B. Stroustrup, «A history of C++: 1979–1991,» *Sigplan Notices - SIGPLAN*, vol. 28, pp. 271-297, 3 1993.
- [12] B. Stroustrup, «Evolving a Language in and for the Real World: C++ 1991-2006,» de *Proceedings of the Third ACM SIGPLAN Conference on History of Programming Languages*, New York, NY, USA, 2007.
- [13] B. Stroustrup, «Thriving in a Crowded and Changing World: C++ 2006–2020,» *Proc. ACM Program. Lang.*, vol. 4, 6 2020.
- [14] B. Stroustrup, «Adding classes to the C language: An exercise in language evolution,» *Software: Practice and Experience*, vol. 13, pp. 139-161, 1983.
- [15] B. Stroustrup, «The UNIX System: Data Abstraction in C,» *AT&T Bell Laboratories Technical Journal*, vol. 63, pp. 1701-1732, 1984.
- [16] B. Stroustrup, *The C++ Programming Language*, USA: Addison-Wesley, 1986.
- [17] B. Stroustrup, «The Evolution of C++: 1985 to 1989,» *Comput. Syst.*, vol. 2, pp. 191-250, 1989.
- [18] B. Stroustrup, *The C++ programming language*, 2nd ed., Reading, Mass., USA: Addison-Wesley, 1991.
- [19] «Programming languages — C++,» ISO/IEC 14882:1998, International Organization for Standardization, Ginebra, Suiza, 1998.
- [20] A. Stepanov y M. Lee, «The Standard Template Library,» Hewlett-Packard Labs HPL-95-11(R.1), October, 1995.
- [21] B. Stroustrup, *The C++ programming language*, 3rd ed., Reading, Mass., USA: Addison-Wesley, 1997.

- [22] «Programming languages — C++,» ISO/IEC 14882:2003, International Organization for Standardization, Ginebra, Suiza, 2003.
- [23] The British Standard Institution (BSI), The C++ Standard: Incorporating Technical Corrigendum No. 1, John Wiley & Sons Inc, 2003.
- [24] B. Stroustrup, Programming: Principles and Practice using C++, 1 ed., Addison-Wesley Professional, 2008.
- [25] «Programming languages — C++,» ISO/IEC 14882:2011, International Organization for Standardization, Ginebra, Suiza, 2011.
- [26] «Programming languages — C++,» ISO/IEC 14882:2014, International Organization for Standardization, Ginebra, Suiza, 2014.
- [27] «Programming languages — C++,» ISO/IEC 14882:2017, International Organization for Standardization, Ginebra, Suiza, 2017.
- [28] «Programming languages — C++,» ISO/IEC 14882:2020, International Organization for Standardization, Ginebra, Suiza, 2020.
- [29] «The Julia Language,» [En línea]. Available: <https://docs.julialang.org/en/v1/>. [Último acceso: 2 de Mayo de 2021].
- [30] B. Stroustrup y H. Sutter, «C++ Core Guidelines,» [En línea]. Available: <https://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines>. [Último acceso: 22 de junio de 2021].
- [31] «SEI CERT C++ Coding Standard,» [En línea]. Available: <https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682>. [Último acceso: 22 de junio de 2021].

- [32] «High Integrity C++ Coding Standard,» [En línea]. Available: <https://www.perforce.com/resources/qac/high-integrity-cpp-coding-standard>. [Último acceso: 22 de junio de 2021].
- [33] C. Bienia, S. Kumar, J. P. Singh y K. Li, «The PARSEC Benchmark Suite: Characterization and Architectural Implications,» de *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, New York, NY, USA, 2008.
- [34] C. Bienia, «Benchmarking Modern Multiprocessors,» Princeton University, USA, 2011.
- [35] F. Black y M. Scholes, «The Pricing of Options and Corporate Liabilities,» *Journal of Political Economy*, vol. 81, p. 637–654, 1973.
- [36] J. Deutscher y I. Reid, «Articulated Body Motion Capture by Stochastic Search,» *International Journal of Computer Vision*, vol. 61, p. 185–205, 2005.
- [37] P. Banerjee, *Parallel Algorithms for VLSI Computer-Aided Design*, USA: Prentice-Hall, Inc., 1994.
- [38] Q. He, Z. Li y X. Zhang, «Data deduplication techniques,» de *2010 International Conference on Future Information Technology and Management Engineering*, 2010.
- [39] E. Sifakis, I. Neverov y R. Fedkiw, «Automatic Determination of Facial Muscle Activations from Sparse Motion Capture Marker Data,» *ACM Trans. Graph.*, vol. 24, p. 417–425, 7 2005.
- [40] Q. Lv, W. Josephson, Z. Wang, M. Charikar y K. Li, «Ferret: A Toolkit for Content-Based Similarity Search of Feature-Rich Data,» *SIGOPS Oper. Syst. Rev.*, vol. 40, p. 317–330, 4 2006.

- [41] M. Müller, D. Charypar y M. Gross, «Particle-Based Fluid Simulation for Interactive Applications,» de *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Goslar, 2003.
- [42] G. Grahne y J. Zhu, «Efficiently Using Prefix-trees in Mining Frequent Itemsets,» 2003.
- [43] T. Whitted, «An Improved Illumination Model for Shaded Display,» *Commun. ACM*, vol. 23, p. 343–349, 6 1980.
- [44] S. Guha, N. Mishra, R. Motwani y L. O'Callaghan, «Clustering data streams,» *IEEE FOCS Conference*, pp. 359-366, 1 2000.
- [45] D. Heath, R. Jarrow y A. Morton, «Bond Pricing and the Term Structure of Interest Rates: A New Methodology for Contingent Claims Valuation,» *Econometrica*, vol. 60, p. 77–105, 1992.
- [46] K. Martinez y J. Cupitt, «VIPS - a highly tuned image processing software architecture,» de *IEEE International Conference on Image Processing 2005*, 2005.
- [47] T. Wiegand, G. J. Sullivan, G. Bjontegaard y A. Luthra, «Overview of the H.264/AVC video coding standard,» *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, pp. 560-576, 2003.
- [48] «up-TCP/IP Stack Library,» [En línea]. Available: <https://parsec.cs.princeton.edu/parsec3-doc.htm#uptcpip>. [Último acceso: 28 de abril de 2020].
- [49] S. Woo, M. Ohara, E. Torrie, J. Singh y A. Gupta, «The SPLASH-2 programs: characterization and methodological considerations,» *Proceedings of the 22nd annual international symposium on Computer architecture*, pp. 24-36, 1995.
- [50] PARSEC Group, «A Memo on Exploration of SPLASH-2 Input Sets,» Princeton University, 2011.

- [51] T. Ball, «The Concept of Dynamic Analysis,» de *Software Engineering — ESEC/FSE '99*, Berlin, 1999.
- [52] «Program Instrumentation Options,» [En línea]. Available: <https://gcc.gnu.org/onlinedocs/gcc/Instrumentation-Options.html>. [Último acceso: 25 de mayo de 2020].
- [53] W. Wei, M. Yunxiu, H. Lilong y B. He, «From source code analysis to static software testing,» de *2014 IEEE Workshop on Advanced Research and Technology in Industry Applications (WARTIA)*, 2014.
- [54] «Clang-Tidy Extra Clang Tools 10 documentation,» [En línea]. Available: <https://releases.llvm.org/10.0.0/tools/clang/tools/extra/docs/clang-tidy/index.html>. [Último acceso: 4 de junio de 2021].
- [55] «{fmt},» [En línea]. Available: <https://github.com/fmtlib/fmt>. [Último acceso: 4 de junio de 2021].
- [56] «Rule of three/five/zero,» [En línea]. Available: https://en.cppreference.com/w/cpp/language/rule_of_three. [Último acceso: 5 de junio de 2021].
- [57] C. Ebert, J. Cain, G. Antoniol, S. Counsell y P. Laplante, «Cyclomatic Complexity,» *IEEE Software*, vol. 33, pp. 27-29, 2016.
- [58] T. Yin, «Lizard,» [En línea]. Available: <https://github.com/terryyin/lizard>. [Último acceso: 8 de junio de 2021].
- [59] «APACHE LICENSE, VERSION 2.0,» [En línea]. Available: <https://www.apache.org/licenses/LICENSE-2.0.html>. [Último acceso: 11 de junio de 2021].
- [60] «GNU General Public License,» [En línea]. Available: <http://www.gnu.org/licenses/gpl-3.0.html>. [Último acceso: 11 de junio de 2021].

- [61] «Attribution-ShareAlike 2.5 Generic (CC BY-SA 2.5),» [En línea]. Available: <https://creativecommons.org/licenses/by-sa/2.5/>. [Último acceso: 11 de junio de 2021].
- [62] «GNU Free Documentation License,» [En línea]. Available: <http://www.gnu.org/licenses/fdl-1.3.html>. [Último acceso: 11 de junio de 2021].
- [63] «NASA Media Usage Guidelines,» [En línea]. Available: <https://www.nasa.gov/multimedia/guidelines/index.html>. [Último acceso: 11 de junio de 2021].
- [64] «Creative Commons Public License,» [En línea]. Available: <https://creativecommons.org/>. [Último acceso: 11 de junio de 2021].
- [65] «Image Magick License,» [En línea]. Available: <http://www.imagemagick.org/script/license.php>. [Último acceso: 11 de junio de 2021].
- [66] «Mesa License and Copyright,» [En línea]. Available: <https://docs.mesa3d.org/license.html>. [Último acceso: 11 de junio de 2021].
- [67] T. Yin, «Lizard License,» [En línea]. Available: <https://github.com/terryyin/lizard/blob/master/LICENSE.txt>. [Último acceso: 11 de junio de 2021].
- [68] V. Zverovich, «FMT License,» [En línea]. Available: <https://github.com/fmtlib/fmt/blob/master/LICENSE.rst>. [Último acceso: 11 de junio de 2021].
- [69] «ISO 25000,» [En línea]. Available: <https://iso25000.com/index.php/normas-iso-25000/iso-25010>. [Último acceso: 11 de junio de 2021].

- [70] «INFORMACIÓN ECONÓMICA | UC3M,» [En línea]. Available: https://www.uc3m.es/ss/Satellite/UC3MInstitucional/es/TextoMixta/1371208931714/Informacion_economica. [Último acceso: 13 de junio de 2021].
- [71] F. J. Aliseda Casado, «PRESUPUESTO DE PROYECTO,» [En línea]. Available: https://e-archivo.uc3m.es/bitstream/handle/10016/11822/Presupuesto_PFC_FJ_Aliseda_Casado.pdf. [Último acceso: 12 de junio de 2021].

A. ANEXO A: FALLOS DE CALIDAD

A.1. Avisos de compilación

En esta primera sección del Anexo A se recoge la recopilación completa de los avisos resultantes de la compilación del benchmark PARSEC. Esta recopilación incorpora todas las aplicaciones del benchmark, incluyendo también aquellas que fueron eliminadas del estudio de calidad. Se recuerda que esta compilación se ha llevado a cabo usando el estándar ISO C++17, así como la versión 10.2 de GCC.

A.1.1 Blackscholes

1. blackscholes.m4.cpp:408:16: warning: invalid suffix on literal; C++11 requires a space between literal and string macro [-Wliteral-suffix]
2. blackscholes.m4.cpp:507:81: warning: format '%d' expects argument of type 'int', but argument 2 has type 'long unsigned int' [-Wformat=]

A.1.2 Bodytrack

1. FlexImage.cpp:168:5: warning: suggest explicit braces to avoid ambiguous 'else' [-Wdangling-else]
2. FlexImage.cpp:201:5: warning: suggest explicit braces to avoid ambiguous 'else' [-Wdangling-else]
3. FlexIO.cpp:112:7: warning: ignoring return value of 'size_t fread(void*, size_t, size_t, FILE*)', declared with attribute warn_unused_result [-Wunused-result]
4. FlexIO.cpp:116:7: warning: ignoring return value of 'size_t fread(void*, size_t, size_t, FILE*)', declared with attribute warn_unused_result [-Wunused-result]
5. FlexIO.cpp:123:8: warning: ignoring return value of 'size_t fread(void*, size_t, size_t, FILE*)', declared with attribute warn_unused_result [-Wunused-result]
6. FlexIO.cpp:134:8: warning: ignoring return value of 'size_t fread(void*, size_t, size_t, FILE*)', declared with attribute warn_unused_result [-Wunused-result]
7. FlexIO.cpp:137:9: warning: ignoring return value of 'size_t fread(void*, size_t, size_t, FILE*)', declared with attribute warn_unused_result [-Wunused-result]
8. FlexIO.cpp:153:7: warning: ignoring return value of 'size_t fread(void*, size_t, size_t, FILE*)', declared with attribute warn_unused_result [-Wunused-result]

9. FlexIO.cpp:157:7: warning: ignoring return value of 'size_t fread(void*, size_t, size_t, FILE*)', declared with attribute warn_unused_result [-Wunused-result]
10. FlexIO.cpp:165:8: warning: ignoring return value of 'size_t fread(void*, size_t, size_t, FILE*)', declared with attribute warn_unused_result [-Wunused-result]
11. FlexIO.cpp:176:8: warning: ignoring return value of 'size_t fread(void*, size_t, size_t, FILE*)', declared with attribute warn_unused_result [-Wunused-result]
12. FlexIO.cpp:179:9: warning: ignoring return value of 'size_t fread(void*, size_t, size_t, FILE*)', declared with attribute warn_unused_result [-Wunused-result]
13. FlexIO.cpp:195:7: warning: ignoring return value of 'size_t fread(void*, size_t, size_t, FILE*)', declared with attribute warn_unused_result [-Wunused-result]
14. FlexIO.cpp:199:7: warning: ignoring return value of 'size_t fread(void*, size_t, size_t, FILE*)', declared with attribute warn_unused_result [-Wunused-result]
15. FlexIO.cpp:230:7: warning: ignoring return value of 'size_t fread(void*, size_t, size_t, FILE*)', declared with attribute warn_unused_result [-Wunused-result]
16. FlexIO.cpp:234:7: warning: ignoring return value of 'size_t fread(void*, size_t, size_t, FILE*)', declared with attribute warn_unused_result [-Wunused-result]
17. Mutex.cpp:76:13: warning: throw will always call terminate() [-Wterminate]
18. Mutex.cpp:82:13: warning: throw will always call terminate() [-Wterminate]
19. Condition.cpp:91:13: warning: throw will always call terminate() [-Wterminate]
20. Condition.cpp:91:13: warning: throw will always call terminate() [-Wterminate]
21. Barrier.cpp:82:13: warning: throw will always call terminate() [-Wterminate]
22. Barrier.cpp:88:13: warning: throw will always call terminate() [-Wterminate]
23. RWLock.cpp:67:13: warning: throw will always call terminate() [-Wterminate]
24. RWLock.cpp:73:13: warning: throw will always call terminate() [-Wterminate]
25. main.cpp:377:17: warning: invalid suffix on literal; C++11 requires a space between literal and string macro [-Wliteral-suffix]
26. AsyncIO.h:48:15: warning: 'AsyncImageLoader::mCurrentFrame' will be initialized after [-Wreorder]
27. ParticleFilterPthread.h:136:21: warning: 'AddGaussianNoise' was not declared in this scope, and no declarations were found by argument-dependent lookup at the point of instantiation [-fpermissive]
28. AsyncIO.h:48:15: warning: 'AsyncImageLoader::mCurrentFrame' will be initialized after [-Wreorder]

29. TrackingModelPthread.h:65:7: warning: 'TrackingModelPthread::IOthread-Started' will be initialized after [-Wreorder]
30. AsyncIO.h:48:15: warning: 'AsyncImageLoader::mCurrentFrame' will be initialized after [-Wreorder]
31. Thread.cpp:74:25warning: catching polymorphic type 'class std::bad_cast' by value [-Wcatch-value=]

A.1.3 Canneal

1. MersenneTwister.h:187:18: warning: ISO C++17 does not allow 'register' storage class specifier [-Wregister]
2. MersenneTwister.h:232:15: warning: ISO C++17 does not allow 'register' storage class specifier [-Wregister]
3. MersenneTwister.h:233:18: warning: ISO C++17 does not allow 'register' storage class specifier [-Wregister]
4. MersenneTwister.h:234:15: warning: ISO C++17 does not allow 'register' storage class specifier [-Wregister]
5. MersenneTwister.h:269:20: warning: ISO C++17 does not allow 'register' storage class specifier [-Wregister]
6. MersenneTwister.h:270:16: warning: ISO C++1z does not allow 'register' storage class specifier [-Wregister]
7. MersenneTwister.h:271:17: warning: ISO C++17 does not allow 'register' storage class specifier [-Wregister]
8. MersenneTwister.h:289:19: warning: ISO C++17 does not allow 'register' storage class specifier [-Wregister]
9. MersenneTwister.h:290:19: warning: ISO C++17 does not allow 'register' storage class specifier [-Wregister]
10. MersenneTwister.h:291:15: warning: ISO C++17 does not allow 'register' storage class specifier [-Wregister]
11. MersenneTwister.h:305:19: warning: ISO C++17 does not allow 'register' storage class specifier [-Wregister]
12. MersenneTwister.h:306:15: warning: ISO C++17 does not allow 'register' storage class specifier [-Wregister]

13. MersenneTwister.h:345:19: warning: ISO C++17 does not allow ‘register’ storage class specifier [-Wregister]
14. MersenneTwister.h:346:25: warning: ISO C++17 does not allow ‘register’ storage class specifier [-Wregister]
15. MersenneTwister.h:356:19: warning: ISO C++17 does not allow ‘register’ storage class specifier [-Wregister]
16. MersenneTwister.h:357:15: warning: ISO C++17 does not allow ‘register’ storage class specifier [-Wregister]
17. MersenneTwister.h:366:33: warning: ISO C++17 does not allow ‘register’ storage class specifier [-Wregister]
18. MersenneTwister.h:367:15: warning: ISO C++17 does not allow ‘register’ storage class specifier [-Wregister]
19. main.cpp:60:17: warning: invalid suffix on literal; C++11 requires a space between literal and string macro [-Wliteral-suffix]
20. main.cpp:141:1: warning: no return statement in function returning non-void [-Wreturn-type]

A.1.4 Dedup

1. No presenta ningún aviso de compilación.

A.1.5 Facesim

1. No presenta ningún aviso de compilación.

A.1.6 Ferret

1. env.c:25:2: warning: #warning to be elaborated. [-Wcpp]
2. env.c:246:2: warning: #warning add file switch [-Wcpp]
3. assign.c:78:8: warning: argument 1 range [18446744071562067968, 18446744073709551615] exceeds maximum object size 92233720368547758-07 [-Walloc-size-larger-than=]
4. assign.c:79:8: warning: argument 1 range [18446744071562067968, 18446744073709551615] exceeds maximum object size 92233720368547758-07 [-Walloc-size-larger-than=]

5. string_fortified.h:71:10: warning: ‘__builtin_memset’: specified size between 18446744071562067968 and 18446744073709551615 exceeds maximum object size 9223372036854775807 [-Wstringop-overflow=]
6. emd.c:118:12: warning: assignment from incompatible pointer type [-Wincompatible-pointer-types]
7. dist.c:110:10: warning: initialization from incompatible pointer type [-Wincompatible-pointer-types]
8. dist.c:130:10: warning: initialization from incompatible pointer type [-Wincompatible-pointer-types]
9. dist.c:145:10: warning: initialization from incompatible pointer type [-Wincompatible-pointer-types]
10. dist.c:160:10: warning: initialization from incompatible pointer type [-Wincompatible-pointer-types]
11. dist.c:175:10: warning: initialization from incompatible pointer type [-Wincompatible-pointer-types]
12. dist.c:195:10: warning: initialization from incompatible pointer type [-Wincompatible-pointer-types]
13. dataset.c:158:8: warning: assignment from incompatible pointer type [-Wincompatible-pointer-types]
14. dataset.c:159:16: warning: cast to pointer from integer of different size [-Wint-to-pointer-cast]
15. cass_array.h:155:13: warning: initialization from incompatible pointer type [-Wincompatible-pointer-types]
16. util.c:223:20: warning: passing argument 1 of ‘bitmap_free_vec’ from incompatible pointer type [-Wincompatible-pointer-types]
17. import.c:34:2: warning: #warning to be extended to allow external id [-Wcpp]
18. bench.c:39:3: warning: ignoring return value of ‘fscanf’, declared with attribute warn_unused_result [-Wunused-result]
19. bench.c:48:4: warning: ignoring return value of ‘fscanf’, declared with attribute warn_unused_result [-Wunused-result]
20. bench.c:49:4: warning: ignoring return value of ‘fscanf’, declared with attribute warn_unused_result [-Wunused-result]

21. `cass_matrix.h:54:11`: warning: assignment from incompatible pointer type [-Wincompatible-pointer-types]
22. `ferret-pthreads.c:234:27`: warning: passing argument 2 of ‘`dequeue`’ from incompatible pointer type [-Wincompatible-pointer-types]
23. `ferret-pthreads.c:245:17`: warning: passing argument 1 of ‘`image_segment`’ from incompatible pointer type [-Wincompatible-pointer-types]
24. `ferret-pthreads.c:265:30`: warning: passing argument 2 of ‘`dequeue`’ from incompatible pointer type [-Wincompatible-pointer-types]
25. `ferret-pthreads.c:293:30`: warning: passing argument 2 of ‘`dequeue`’ from incompatible pointer type [-Wincompatible-pointer-types]
26. `ferret-pthreads.c:335:27`: warning: passing argument 2 of ‘`dequeue`’ from incompatible pointer type [-Wincompatible-pointer-types]
27. `ferret-pthreads.c:380:27`: warning: passing argument 2 of ‘`dequeue`’ from incompatible pointer type [-Wincompatible-pointer-types]
28. `ferret-pthreads.c:468:28`: warning: passing argument 2 of ‘`cass_env_open`’ discards ‘`const`’ qualifier from pointer target type [-Wdiscarded-qualifiers]
29. `ferret-pthreads.c:513:21`: warning: assignment discards ‘`const`’ qualifier from pointer target type [-Wdiscarded-qualifiers]

A.1.7 Fluidanimate

1. `pthread.cpp:1204:22`: warning: invalid suffix on literal; C++11 requires a space between literal and string macro [-Wliteral-suffix]

A.1.8 Freqmine

1. `fpmax.cpp:114:16`: warning: invalid suffix on literal; C++11 requires a space between literal and string macro [-Wliteral-suffix]
2. `buffer.cpp:104:24`: warning: ISO C++17 does not allow ‘`register`’ storage class specifier [-Wregister]
3. `buffer.cpp:105:17`: warning: ISO C++17 does not allow ‘`register`’ storage class specifier [-Wregister]

A.1.9 Netdedup

1. encoder.c:237:17: warning: implicit declaration of function ‘htons’ [-Wimplicit-function-declaration]

A.1.10 Netferret

1. env.c:25:2: warning: #warning to be elaborated. [-Wcpp]
2. env.c:246:2: warning: #warning add file switch [-Wcpp]
3. assign.c:78:8: warning: argument 1 range [18446744071562067968, 18446744073709551615] exceeds maximum object size 9223372036854775807 [-Walloc-size-larger-than=]
4. assign.c:79:8: warning: argument 1 range [18446744071562067968, 18446744073709551615] exceeds maximum object size 9223372036854775807 [-Walloc-size-larger-than=]
5. string_fortified.h:71:10: warning: ‘__builtin_memset’: specified size between 18446744071562067968 and 18446744073709551615 exceeds maximum object size 9223372036854775807 [-Wstringop-overflow=]
6. emd.c:118:12: warning: assignment from incompatible pointer type [-Wincompatible-pointer-types]
7. dist.c:110:10: warning: initialization from incompatible pointer type [-Wincompatible-pointer-types]
8. dist.c:130:10: warning: initialization from incompatible pointer type [-Wincompatible-pointer-types]
9. dist.c:145:10: warning: initialization from incompatible pointer type [-Wincompatible-pointer-types]
10. dist.c:160:10: warning: initialization from incompatible pointer type [-Wincompatible-pointer-types]
11. dist.c:175:10: warning: initialization from incompatible pointer type [-Wincompatible-pointer-types]
12. dist.c:195:10: warning: initialization from incompatible pointer type [-Wincompatible-pointer-types]
13. dataset.c:158:8: warning: assignment from incompatible pointer type [-Wincompatible-pointer-types]

14. dataset.c:159:16: warning: cast to pointer from integer of different size [-Wint-to-pointer-cast]
15. cass_array.h:155:13: warning: initialization from incompatible pointer type [-Wincompatible-pointer-types]
16. util.c:223:20: warning: passing argument 1 of 'bitmap_free_vec' from incompatible pointer type [-Wincompatible-pointer-types]
17. import.c:34:2: warning: #warning to be extended to allow external id [-Wcpp]
18. bench.c:39:3: warning: ignoring return value of 'fscanf', declared with attribute warn_unused_result [-Wunused-result]
19. bench.c:48:4: warning: ignoring return value of 'fscanf', declared with attribute warn_unused_result [-Wunused-result]
20. bench.c:49:4: warning: ignoring return value of 'fscanf', declared with attribute warn_unused_result [-Wunused-result]
21. local.c:101:28: warning: format '%u' expects argument of type 'unsigned int', but argument 3 has type 'size_t {aka long unsigned int}' [-Wformat=]
22. cass_matrix.h:54:11: warning: assignment from incompatible pointer type [-Wincompatible-pointer-types]
23. ferret-pthreads.c:242:27: warning: passing argument 2 of 'dequeue' from incompatible pointer type [-Wincompatible-pointer-types]
24. ferret-pthreads.c:311:27: warning: passing argument 2 of 'dequeue' from incompatible pointer type [-Wincompatible-pointer-types]
25. ferret-pthreads.c:322:17: warning: passing argument 1 of 'image_segment' from incompatible pointer type [-Wincompatible-pointer-types]
26. ferret-pthreads.c:342:30: warning: passing argument 2 of 'dequeue' from incompatible pointer type [-Wincompatible-pointer-types]
27. ferret-pthreads.c:370:30: warning: passing argument 2 of 'dequeue' from incompatible pointer type [-Wincompatible-pointer-types]
28. ferret-pthreads.c:412:27: warning: passing argument 2 of 'dequeue' from incompatible pointer type [-Wincompatible-pointer-types]
29. ferret-pthreads.c:467:27: warning: passing argument 2 of 'dequeue' from incompatible pointer type [-Wincompatible-pointer-types]
30. ferret-pthreads.c:566:28: warning: passing argument 2 of 'cass_env_open' discards 'const' qualifier from pointer target type [-Wdiscarded-qualifiers]

31. image.c:479:30: warning: passing argument 2 of ‘net_jpeg_stdio_src’ discards ‘const’ qualifier from pointer target type [-Wdiscarded-qualifiers]
32. client.c:240:31: warning: passing argument 2 of ‘dequeue’ from incompatible pointer type [-Wincompatible-pointer-types]
33. client.c:405:21: warning: assignment discards ‘const’ qualifier from pointer target type [-Wdiscarded-qualifiers]

A.1.11 Netstreamcluster

1. streamcluster.cpp:2278:24: warning: invalid suffix on literal; C++11 requires a space between literal and string macro [-Wliteral-suffix]
2. client.cpp:193:58: warning: ‘ ’ flag used with ‘%c’ gnu_printf format [-Wformat=]
3. client.cpp:193:58: warning: format ‘%c’ expects a matching ‘int’ argument [-Wformat=]

A.1.12 Raytrace

1. RTThread.cxx:136:74: warning: cast to pointer from integer of different size [-Wint-to-pointer-cast]
2. render.cxx:728:10: warning: deleting ‘LRTContext {aka void*}’ is undefined [-Wdelete-incomplete]
3. render.cxx:740:10: warning: deleting ‘LRTCera {aka void*}’ is undefined [-Wdelete-incomplete]
4. rtview.cxx:341:11: warning: invalid suffix on literal; C++11 requires a space between literal and string macro [-Wliteral-suffix]
5. ImagePPM.hxx:24:15: warning: ignoring return value of ‘int fscanf(FILE*, const char*, ...)’, declared with attribute warn_unused_result [-Wunused-result]
6. ImagePPM.hxx:30:22: warning: ignoring return value of ‘char* fgets(char*, int, FILE*)’, declared with attribute warn_unused_result [-Wunused-result]
7. ImagePPM.hxx:35:15: warning: ignoring return value of ‘int fscanf(FILE*, const char*, ...)’, declared with attribute warn_unused_result [-Wunused-result]
8. ImagePPM.hxx:51:27: warning: ignoring return value of ‘int fscanf(FILE*, const char*, ...)’, declared with attribute warn_unused_result [-Wunused-result]

A.1.13 Streamcluster

1. streamcluster.cpp:1970:24: warning: invalid suffix on literal; C++11 requires a space between literal and string macro [-Wliteral-suffix]
2. streamcluster.cpp:1825:39: warning: format ‘%u’ expects argument of type ‘unsigned int’, but argument 3 has type ‘long int’ [-Wformat=]
3. streamcluster.cpp:1890:46: warning: format ‘%d’ expects argument of type ‘int’, but argument 3 has type ‘size_t {aka long unsigned int}’ [-Wformat=]

A.1.14 Swaptions

1. icdf.cpp:11:18: warning: ISO C++1z does not allow ‘register’ storage class specifier [-Wregister]
2. icdf.cpp:11:21: warning: ISO C++1z does not allow ‘register’ storage class specifier [-Wregister]
3. HJM_Securities.cpp:143:16: warning: invalid suffix on literal; C++11 requires a space between literal and string macro [-Wliteral-suffix]

A.1.15 Vips

1. string_fortified.h:106:10: warning: ‘__builtin_strncpy’ specified bound depends on the length of the source argument [-Wstringop-overflow=]

A.1.16 x264

1. muxers.c:141:10: warning: variable ‘interlaced’ set but not used [-Wunused-but-set-variable]
2. macroblock.c:278:9: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
3. macroblock.c:287:17: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
4. macroblock.c:293:5: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
5. macroblock.c:294:5: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
6. macroblock.c:373:23: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]

7. macroblock.c:374:23: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
8. macroblock.c:374:23: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
9. macroblock.c:408:5: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
10. macroblock.c:409:5: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
11. macroblock.c:410:5: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
12. macroblock.c:411:5: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
13. macroblock.c:412:5: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
14. macroblock.c:413:5: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
15. macroblock.c:414:5: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
16. macroblock.c:415:5: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
17. macroblock.c:433:9: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
18. macroblock.c:1160:17: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
19. macroblock.c:1166:17: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
20. macroblock.c:1178:17: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
21. macroblock.c:1179:17: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
22. macroblock.c:1184:17: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]

23. macroblock.c:1185:17: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
24. macroblock.c:1195:17: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
25. macroblock.c:1201:17: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
26. macroblock.c:1214:17: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
27. macroblock.c:1215:17: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
28. macroblock.c:1216:17: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
29. macroblock.c:1217:17: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
30. macroblock.c:1225:21: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
31. macroblock.c:1235:21: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
32. macroblock.c:1236:21: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
33. macroblock.c:1241:21: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
34. macroblock.c:1242:21: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
35. macroblock.c:1249:21: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
36. macroblock.c:1250:21: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
37. macroblock.c:1251:21: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
38. macroblock.c:1252:21: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]

- 39. macroblock.c:1258:25: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
- 40. macroblock.c:1392:17: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
- 41. macroblock.c:1393:17: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
- 42. macroblock.c:1403:21: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
- 43. macroblock.c:1404:21: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
- 44. macroblock.c:1435:17: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
- 45. macroblock.c:1436:17: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
- 46. macroblock.c:1441:21: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
- 47. macroblock.c:1442:21: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
- 48. macroblock.c:893:31: warning: array subscript is below array bounds [-Warray-bounds]
- 49. macroblock.c:894:31: warning: array subscript is below array bounds [-Warray-bounds]
- 50. cpu.c:129:28: warning: variable ‘stepping’ set but not used [-Wunused-but-set-variable]
- 51. cabac.c:745:22: warning: ‘x264_cabac_probability’ defined but not used [-Wunused-const-variable=]
- 52. set.c:158:100: warning: iteration 8 invokes undefined behavior [-Waggressive-loop-optimizations]
- 53. set.c:147:100: warning: iteration 4 invokes undefined behavior [-Waggressive-loop-optimizations]
- 54. set.c:138:49: warning: iteration 8 invokes undefined behavior [-Waggressive-loop-optimizations]

- 55. set.c:132:49: warning: iteration 4 invokes undefined behavior [-Waggressive-loop-optimizations]
- 56. cabac.c:689:9: warning: variable 'i_coeff_sign' set but not used [-Wunused-but-set-variable]
- 57. analyse.c:930:22: warning: variable 'p_src_by' set but not used [-Wunused-but-set-variable]
- 58. analyse.c:855:9: warning: variable 'i_pred_mode' set but not used [-Wunused-but-set-variable]
- 59. analyse.c:1091:9: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
- 60. analyse.c:1092:9: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
- 61. analyse.c:1102:9: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
- 62. analyse.c:1102:9: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
- 63. analyse.c:1105:13: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
- 64. analyse.c:1197:5: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
- 65. analyse.c:1215:9: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
- 66. analyse.c:1217:9: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
- 67. analyse.c:1263:13: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
- 68. analyse.c:1264:13: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
- 69. analyse.c:1265:13: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
- 70. analyse.c:1277:9: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]

71. analyse.c:1312:13: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
72. analyse.c:1313:13: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
73. analyse.c:1314:13: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
74. analyse.c:1326:9: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
75. analyse.c:1395:9: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
76. analyse.c:1435:9: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
77. analyse.c:1472:9: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
78. analyse.c:1547:9: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
79. analyse.c:1574:9: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
80. analyse.c:1613:13: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
81. analyse.c:1616:13: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
82. analyse.c:1617:13: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
83. analyse.c:1620:13: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
84. analyse.c:1621:13: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
85. analyse.c:1624:13: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
86. analyse.c:1625:13: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]

87. analyse.c:1626:13: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
88. analyse.c:1627:13: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
89. analyse.c:1677:9: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
90. analyse.c:1682:5: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
91. analyse.c:1686:5: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
92. analyse.c:1726:13: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
93. analyse.c:1786:13: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
94. analyse.c:1787:13: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
95. analyse.c:1855:13: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
96. analyse.c:1856:13: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
97. analyse.c:2770:21: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
98. analyse.c:2776:21: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
99. analyse.c:2777:21: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
100. analyse.c:2783:21: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
101. analyse.c:2784:21: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
102. analyse.c:2806:13: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]

103. analyse.c:2832:21: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
104. analyse.c:2844:21: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
105. analyse.c:2848:21: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
106. analyse.c:2851:21: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
107. slicetype.c:156:17: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
108. slicetype.c:159:17: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
109. slicetype.c:161:21: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
110. slicetype.c:163:21: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
111. slicetype.c:170:13: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
112. slicetype.c:172:13: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
113. slicetype.c:177:13: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
114. slicetype.c:183:5: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
115. slicetype.c:183:5: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
116. slicetype.c:65:9: warning: variable 'i_cost_bak' set but not used [-Wunused-but-set-variable]
117. macroblock.c:698:13: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]
118. macroblock.c:699:13: warning: dereferencing type-punned pointer will break strict-aliasing rules [-Wstrict-aliasing]

119. encoder.c:1734:53: warning: iteration 3 invokes undefined behavior [-Waggressive-loop-optimizations]

A.2. Fugas de memoria

En esta segunda sección del Anexo A se recoge la recopilación completa de las fugas de memoria detectadas por medio del análisis dinámico en las aplicaciones mantenidas en el estudio de calidad del *benchmark* PARSEC, indicando cuáles han sido resueltas.

A.2.1 Freqmine

1. (Resuelta) Direct leak of 4 byte(s) in 1 object(s) allocated from:
#0 0x7f283ca9c087 in operator new[](unsigned long) (/lib/x86_64-linux-gnu/libasan.so.6+0xb2087)
#1 0x55d7033f0002 in void first_transform_FPTree_into_FPArray<unsigned short>(FP_tree*, unsigned short) (fp_tree.cpp:187)
2. (Resuelta) Direct leak of 16 byte(s) in 1 object(s) allocated from:
#0 0x7f283ca9c087 in operator new[](unsigned long) (/lib/x86_64-linux-gnu/libasan.so.6+0xb2087)
#1 0x55d7033f06b1 in void first_transform_FPTree_into_FPArray<unsigned short>(FP_tree*, unsigned short) (fp_tree.cpp:194)
3. (Resuelta) Indirect leak of 6632 byte(s) in 1 object(s) allocated from:
#0 0x7f283ca9c087 in operator new[](unsigned long) (/lib/x86_64-linux-gnu/libasan.so.6+0xb2087)
#1 0x55d7033f06f7 in void first_transform_FPTree_into_FPArray<unsigned short>(FP_tree*, unsigned short) (fp_tree.cpp:195)
4. (Resuelta) Direct leak of 24 byte(s) in 1 object(s) allocated from:
#0 0x7f283ca9bf17 in operator new(unsigned long) (/lib/x86_64-linux-gnu/libasan.so.6+0xb1f17)
#1 0x55d7033e05a9 in FP_tree::scan1_DB(Data*) (fp_tree.cpp:850)
5. (Resuelta) Direct leak of 16 byte(s) in 1 object(s) allocated from:
#0 0x7f283ca9bf17 in operator new(unsigned long) (/lib/x86_64-linux-gnu/libasan.so.6+0xb1f17)
#1 0x55d7033d3897 in main (fpmax.cpp:131)

6. Direct leak of 400000 byte(s) in 1 object(s) allocated from:
 - #0 0x7f283ca9c087 in operator new[](unsigned long) (/lib/x86_64-linux-gnu/libasan.so.6+0xb2087)
 - #1 0x55d7033d49d5 in MapFileNode::int(int, int) (data.cpp:130)
 - #2 0x55d7033d49d5 in Data::parseDataFile(MapFile*) (data.cpp:34)
7. Direct leak of 177200000 byte(s) in 443 object(s) allocated from:
 - #0 0x7f283ca9c087 in operator new[](unsigned long) (/lib/x86_64-linux-gnu/libasan.so.6+0xb2087)
 - #1 0x55d7033d5c6d in MapFileNode::init(int, int) (data.cpp:130)
 - #2 0x55d7033d5c6d in Data::parseDataFile(MapFile*) (data.cpp:66)
8. Direct leak of 1600000 byte(s) in 4 object(s) allocated from:
 - #0 0x7f283ca9c087 in operator new[](unsigned long) (/lib/x86_64-linux-gnu/libasan.so.6+0xb2087)
 - #1 0x55d7033d5b30 in MapFileNode::init(int, int) (data.cpp:130)
 - #2 0x55d7033d5b30 in Data::parseDataFile(MapFile*) (data.cpp:106)

A.2.2 Netstreamcluster

1. (Resuelta) Direct leak of 102400000 byte(s) in 1 object(s) allocated from:
 - #0 0x7f70db4bf9d1 in malloc (/lib/x86_64-linux-gnu/liblsan.so.0+0xf9d1)
 - #1 0x5606ef967f52 in send_to_server(void*) (client.cpp:97)
2. (Resuelta) Direct leak of 264 byte(s) in 1 object(s) allocated from:
 - #0 0x7effd83d7722 (/lib/x86_64-linux-gnu/liblsan.so.0+0xf722)
 - #1 0x56183676a708 in outcenterIDs(Points*, long*, char*) (streamcluster.cpp:1857)
3. (Resuelta) Direct leak of 120 byte(s) in 1 object(s) allocated from:
 - #0 0x7effd83d79d1 in malloc (/lib/x86_64-linux-gnu/liblsan.so.0+0xf9d1)
 - #1 0x56183676affd in streamCluster(PStream*, long, long, int, long, long, char*) (streamcluster.cpp:2121)
4. (Resuelta) Indirect leak of 32000000 byte(s) in 5 object(s) allocated from:
 - #0 0x7effd83d79d1 in malloc (/lib/x86_64-linux-gnu/liblsan.so.0+0xf9d1)
 - #1 0x56183676b089 in streamCluster(PStream*, long, long, int, long, long, char*) (streamcluster.cpp:2142)

5. (Resuelta) Indirect leak of 2560000 byte(s) in 1 object(s) allocated from:
#0 0x7effd83d79d1 in malloc (/lib/x86_64-linux-gnu/liblsan.so.0+0xf9d1)
#1 0x56183676b4bb in streamCluster(PStream*, long, long, int, long, long, char*) (streamcluster.cpp:2155)
6. (Resuelta) Direct leak of 5120000 byte(s) in 1 object(s) allocated from:
#0 0x7effd83d79d1 in malloc (/lib/x86_64-linux-gnu/liblsan.so.0+0xf9d1)
#1 0x56183676b4cb in streamCluster(PStream*, long, long, int, long, long, char*) (streamcluster.cpp:2156)
7. (Resuelta) Direct leak of 160000 byte(s) in 1 object(s) allocated from:
#0 0x7effd83d79d1 in malloc (/lib/x86_64-linux-gnu/liblsan.so.0+0xf9d1)
#1 0x56183676b4df in streamCluster(PStream*, long, long, int, long, long, char*) (streamcluster.cpp:2165)
8. (Resuelta) Direct leak of 1024 byte(s) in 1 object(s) allocated from:
#0 0x7effd83d8511 in operator new[](unsigned long) (/lib/x86_64-linux-gnu/liblsan.so.0+0x10511)
#1 0x561836763ba1 in main (streamcluster.cpp:2270)
9. (Resuelta) Direct leak of 1024 byte(s) in 1 object(s) allocated from:
#0 0x7effd83d8511 in operator new[](unsigned long) (/lib/x86_64-linux-gnu/liblsan.so.0+0x10511)
#1 0x561836763bae in main (streamcluster.cpp:2271)
10. Indirect leak of 512000000 byte(s) in 5 object(s) allocated from:
#0 0x7effd83d79d1 in malloc (/lib/x86_64-linux-gnu/liblsan.so.0+0xf9d1)
#1 0x56183676b069 in streamCluster(PStream*, long, long, int, long, long, char*) (streamcluster.cpp:2127)
11. Direct leak of 96 byte(s) in 2 object(s) allocated from:
#0 0x7f70db4bf9d1 in malloc (/lib/x86_64-linux-gnu/liblsan.so.0+0xf9d1)
#1 0x5606ef969bcf in host_thread_cond_init (host_serv.c:800)
12. Direct leak of 24 byte(s) in 3 object(s) allocated from:
#0 0x7f70db4bf9d1 in malloc (/lib/x86_64-linux-gnu/liblsan.so.0+0xf9d1)
#1 0x5606ef969301 in host_thread_create (host_serv.c:330)

A.2.3 Streamcluster

1. (Resuelta) Direct leak of 264 byte(s) in 1 object(s) allocated from:
#0 0x7fcad276e722 (/lib/x86_64-linux-gnu/liblsan.so.0+0xf722)
#1 0x5605ff552d98 in outcenterIDs(Points*, long*, char*) (streamcluster.cpp:1818)
2. (Resuelta) Indirect leak of 102400000 byte(s) in 1 object(s) allocated from:
#0 0x7fcad276e9d1 in malloc (/lib/x86_64-linux-gnu/liblsan.so.0+0xf9d1)
#1 0x5605ff553408 in streamCluster(PStream*, long, long, int, long, long, char*) (streamcluster.cpp:1846)
3. (Resuelta) Indirect leak of 2560000 byte(s) in 1 object(s) allocated from:
#0 0x7fcad276e9d1 in malloc (/lib/x86_64-linux-gnu/liblsan.so.0+0xf9d1)
#1 0x5605ff55341d in streamCluster(PStream*, long, long, int, long, long, char*) (streamcluster.cpp:1847)
4. (Resuelta) Direct leak of 5120000 byte(s) in 1 object(s) allocated from:
#0 0x7fcad276e9d1 in malloc (/lib/x86_64-linux-gnu/liblsan.so.0+0xf9d1)
#1 0x5605ff55342d in streamCluster(PStream*, long, long, int, long, long, char*) (streamcluster.cpp:1848)
5. (Resuelta) Direct leak of 6400000 byte(s) in 1 object(s) allocated from:
#0 0x7fcad276e9d1 in malloc (/lib/x86_64-linux-gnu/liblsan.so.0+0xf9d1)
#1 0x5605ff553458 in streamCluster(PStream*, long, long, int, long, long, char*) (streamcluster.cpp:1863)
6. (Resuelta) Direct leak of 160000 byte(s) in 1 object(s) allocated from:
#0 0x7fcad276e9d1 in malloc (/lib/x86_64-linux-gnu/liblsan.so.0+0xf9d1)
#1 0x5605ff553978 in streamCluster(PStream*, long, long, int, long, long, char*) (streamcluster.cpp:1876)
7. (Resuelta) Direct leak of 1024 byte(s) in 1 object(s) allocated from:
#0 0x7fcad276f511 in operator new[](unsigned long) (/lib/x86_64-linux-gnu/liblsan.so.0+0x10511)
#1 0x5605ff54c50e in main (streamcluster.cpp:1962)

8. (Resuelta) Direct leak of 1024 byte(s) in 1 object(s) allocated from:
#0 0x7fcad276f511 in operator new[](unsigned long) (/lib/x86_64-linux-gnu/liblsan.so.0+0x10511)
#1 0x5605ff54c501 in main (streamcluster.cpp:1963)

A.2.4 Swaptions

1. Direct leak of 721496 byte(s) in 896 object(s) allocated from:
#0 0x7f6ccbc459d1 in malloc (/lib/x86_64-linux-gnu/liblsan.so.0+0xf9d1)
#1 0x5651ba8baf37 in dvector(long, long) (nr_routines.c:138)
2. Direct leak of 16416 byte(s) in 257 object(s) allocated from:
#0 0x7f6ccbc459d1 in malloc (/lib/x86_64-linux-gnu/liblsan.so.0+0xf9d1)
#1 0x5651ba8bafad in dmatrix(long, long, long, long) (nr_routines.c:162)
3. Indirect leak of 2015480 byte(s) in 257 object(s) allocated from:
#0 0x7f6ccbc459d1 in malloc (/lib/x86_64-linux-gnu/liblsan.so.0+0xf9d1)
#1 0x5651ba8bafdf in dmatrix(long, long, long, long) (nr_routines.c:168)

B. ANEXO B: EXPERIMENTACIÓN

En este segundo anexo se recoge la experimentación completa del estudio llevada a cabo para evaluar el rendimiento. Se exponen para cada aplicación del *benchmark* PARSEC los tiempos de las diez ejecuciones medidas durante las diferentes iteraciones del estudio.

B.1. Blackscholes

Tabla B.1. Tiempos de ejecución en la aplicación Blackscholes (1 hilo)

	1 hilo					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	33,014	32,817	0,192	33,105	32,877	0,224
Native #2	33,142	32,791	0,324	33,072	32,79	0,276
Native #3	32,931	32,679	0,248	33,15	32,845	0,3
Native #4	33,188	32,886	0,296	33,044	32,771	0,264
Native #5	33,011	32,738	0,268	33,004	32,752	0,248
Native #6	33,109	32,863	0,24	33,618	33,304	0,308
Native #7	33,053	32,794	0,252	32,944	32,716	0,224
Native #8	33,077	32,74	0,332	33,088	32,822	0,26
Native #9	33,1	32,823	0,272	33,071	32,77	0,296
Native #10	33,179	32,913	0,26	33,192	32,855	0,332
Media (\bar{x})	33,0804	32,8044	0,2684	33,1288	32,8502	0,2732
Desviación típica (σ)	0,080694486	0,072234	0,041245	0,185435	0,167159	0,035978

Tabla B.2. Tiempos de ejecución en la aplicación Blackscholes (2 hilos)

	2 hilos					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	20,11	32,832	0,304	19,999	32,78	0,264
Native #2	20,062	32,8	0,264	20,082	32,809	0,312
Native #3	20,209	32,967	0,272	20,076	32,817	0,268
Native #4	20,045	32,718	0,32	20,096	32,889	0,236
Native #5	20,1	32,859	0,268	20,132	32,843	0,292
Native #6	20,04	32,817	0,244	20,047	32,863	0,244
Native #7	20,018	32,838	0,216	19,993	32,862	0,164
Native #8	20,04	32,746	0,312	20,165	33,046	0,252
Native #9	19,998	32,746	0,26	20,08	32,901	0,204
Native #10	20,053	32,796	0,296	20,155	32,851	0,324
Media (\bar{x})	20,0675	32,8119	0,2756	20,0825	32,8661	0,256
Desviación típica (σ)	0,060022681	0,070933	0,032521	0,058677	0,073014	0,048295

Tabla B.3. Tiempos de ejecución en la aplicación Blacksholes (4 hilos)

	4 hilos					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	13,689	33,256	0,28	13,67	33,381	0,196
Native #2	13,651	33,198	0,244	13,756	33,366	0,276
Native #3	13,669	33,102	0,32	13,717	33,361	0,292
Native #4	13,516	33,154	0,208	13,693	33,191	0,268
Native #5	13,745	33,282	0,304	13,66	33,229	0,304
Native #6	13,763	33,38	0,264	13,655	33,177	0,308
Native #7	13,7	33,467	0,224	13,726	33,466	0,24
Native #8	13,711	33,272	0,276	13,811	33,37	0,308
Native #9	13,72	33,343	0,344	13,597	33,169	0,324
Native #10	13,782	33,429	0,3	13,646	33,363	0,252
Media (\bar{x})	13,6946	33,2883	0,2764	13,6931	33,3073	0,2768
Desviación típica (σ)	0,074748764	0,118029	0,042727	0,061504	0,105273	0,039046

Tabla B.4. Tiempos de ejecución en la aplicación Blacksholes (8 hilos)

	8 hilos					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	10,629	34,37	0,216	10,716	34,369	0,272
Native #2	10,609	34,264	0,26	10,649	34,181	0,324
Native #3	10,746	34,35	0,316	10,714	34,384	0,312
Native #4	10,551	34,171	0,252	10,588	34,166	0,32
Native #5	10,761	34,373	0,268	10,514	34,138	0,204
Native #6	10,648	34,312	0,288	10,815	34,346	0,3
Native #7	10,567	34,289	0,204	10,678	34,435	0,26
Native #8	10,554	34,312	0,24	10,708	34,465	0,268
Native #9	10,662	34,28	0,304	10,565	34,227	0,284
Native #10	10,656	34,182	0,336	10,624	34,517	0,308
Media (\bar{x})	10,6383	34,2903	0,2684	10,6571	34,3228	0,2852
Desviación típica (σ)	0,073318256	0,070284	0,042852	0,087916	0,135386	0,036322

Tabla B.5. Tiempos de ejecución en la aplicación Blackscholes (16 hilos)

	16 hilos					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	9,561	47,202	0,324	9,508	47,139	0,296
Native #2	9,595	47,294	0,292	9,803	47,166	0,311
Native #3	9,627	47,318	0,288	9,568	47,262	0,252
Native #4	9,648	47,318	0,288	9,57	47,193	0,324
Native #5	9,587	47,306	0,284	9,519	47,287	0,208
Native #6	9,468	47,19	0,236	9,637	47,29	0,316
Native #7	9,568	47,179	0,352	9,702	47,456	0,24
Native #8	9,695	47,35	0,332	9,513	47,261	0,228
Native #9	9,591	47,272	0,296	9,624	47,324	0,288
Native #10	9,73	47,427	0,272	9,537	47,268	0,24
Media (\bar{x})	9,607	47,2856	0,2964	9,5981	47,2646	0,2703
Desviación típica (σ)	0,073569317	0,077749	0,032793	0,095457	0,089421	0,041425

B.2. Bodytrack

Tabla B.6. Tiempos de ejecución en la aplicación Bodytrack (1 hilo)

	1 hilo					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	47,208	47,282	0,404	46,844	46,977	0,357
Native #2	47,074	47,162	0,381	46,968	47,065	0,396
Native #3	46,995	47,09	0,385	47,38	47,513	0,333
Native #4	47,139	47,18	0,413	46,923	47	0,417
Native #5	47,435	47,565	0,321	47,497	47,634	0,345
Native #6	47,015	47,124	0,381	47,558	47,62	0,381
Native #7	47,293	47,423	0,337	47,634	47,716	0,389
Native #8	47,024	47,068	0,429	47,181	47,255	0,417
Native #9	47,398	47,468	0,385	47,46	47,613	0,297
Native #10	47,029	47,14	0,377	47,508	47,625	0,341
Media (\bar{x})	47,161	47,2502	0,3813	47,2953	47,4018	0,3673
Desviación típica (σ)	0,164653	0,17545	0,032387	0,291818	0,295131	0,039209

Tabla B.7. Tiempos de ejecución en la aplicación Bodytrack (2 hilos)

	2 hilos					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	25,796	49,05	0,505	25,785	48,96	0,55
Native #2	26,051	49,541	0,376	26,013	49,428	0,56
Native #3	26,04	49,554	0,498	26,144	49,768	0,392
Native #4	26,047	49,528	0,538	26,15	49,661	0,481
Native #5	25,793	48,982	0,581	26,193	49,862	0,369
Native #6	25,917	49,255	0,409	26,179	49,778	0,389
Native #7	25,813	48,975	0,489	26,127	49,734	0,373
Native #8	25,949	49,302	0,409	26,131	49,77	0,328
Native #9	25,83	49,018	0,633	26,056	49,55	0,385
Native #10	25,852	49,152	0,406	26,122	49,748	0,348
Media (\bar{x})	25,9088	49,2357	0,4844	26,09	49,6259	0,4175
Desviación típica (σ)	0,10696	0,237099	0,084449	0,119829	0,265834	0,082722

Tabla B.8. Tiempos de ejecución en la aplicación Bodytrack (4 hilos)

	4 hilos					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	14,757	51,319	0,47	14,833	51,614	0,482
Native #2	14,798	51,499	0,426	14,712	51,209	0,488
Native #3	14,89	51,872	0,406	15,086	52,441	0,458
Native #4	14,984	52,38	0,412	14,938	52,021	0,46
Native #5	14,721	51,305	0,404	14,763	51,22	0,508
Native #6	14,711	51,201	0,439	14,698	51,218	0,405
Native #7	15,043	52,394	0,465	15,019	52,355	0,476
Native #8	15,093	52,664	0,349	15,118	52,639	0,502
Native #9	15,058	52,626	0,377	14,861	51,614	0,524
Native #10	14,794	51,43	0,507	14,881	51,781	0,51
Media (\bar{x})	14,8849	51,869	0,4255	14,8909	51,8112	0,4813
Desviación típica (σ)	0,148095	0,590505	0,04666	0,148393	0,533789	0,034448

Tabla B.9. Tiempos de ejecución en la aplicación Bodytrack (8 hilos)

	8 hilos					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	10,223	63,696	0,603	10,352	64,24	0,688
Native #2	10,251	63,756	0,608	10,31	64,199	0,521
Native #3	10,399	64,32	0,623	10,45	64,46	0,677
Native #4	10,326	64,088	0,644	10,292	64,156	0,777
Native #5	10,583	65,567	0,494	10,134	63,098	0,729
Native #6	10,359	64,53	0,833	10,408	64,559	0,594
Native #7	10,314	64,122	0,609	10,448	64,796	0,574
Native #8	10,462	64,918	0,651	10,518	65,34	0,756
Native #9	10,352	64,353	0,679	10,308	63,771	0,608
Native #10	10,297	64,076	0,792	10,193	63,667	0,679
Media (\bar{x})	10,3566	64,3426	0,6536	10,3413	64,2286	0,6603
Desviación típica (σ)	0,105142	0,55904	0,097251	0,119477	0,626204	0,083584

Tabla B.10. Tiempos de ejecución en la aplicación Bodytrack (16 hilos)

	16 hilos					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	8,603	79,716	0,823	8,533	79,831	0,853
Native #2	8,462	79,467	1,14	8,204	79,09	0,972
Native #3	8,218	77,339	0,947	8,489	78,926	0,963
Native #4	8,347	79,156	0,974	7,939	79,144	0,843
Native #5	8,522	79,407	0,927	8,043	78,919	0,947
Native #6	8,396	79,76	0,956	8,539	79,505	1,057
Native #7	8,503	80,027	0,83	7,836	78,678	1,031
Native #8	8,122	78,927	1,029	8,38	79,366	1,001
Native #9	7,724	79,02	0,915	8,312	79,137	0,941
Native #10	7,695	78,78	1,123	8,336	79,728	0,909
Media (\bar{x})	8,2592	79,1599	0,9664	8,2611	79,2324	0,9517
Desviación típica (σ)	0,323098	0,753542	0,106711	0,249604	0,370098	0,069813

B.3. Canneal

Tabla B.11. Tiempos de ejecución en la aplicación Canneal (1 hilo)

	1 hilo					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	73,966	73,753	0,2	72,163	71,917	0,232
Native #2	73,159	72,954	0,18	72,236	71,964	0,26
Native #3	72,999	72,804	0,184	72,298	72,062	0,224
Native #4	73,52	73,333	0,176	72,368	72,157	0,2
Native #5	73,079	73,824	0,244	72,285	72,056	0,216
Native #6	73,152	73,008	0,132	72,826	72,615	0,2
Native #7	73,499	73,295	0,192	72,447	72,253	0,18
Native #8	73,512	74,26	0,24	72,745	72,522	0,212
Native #9	73,007	72,787	0,208	72,882	72,602	0,268
Native #10	73,773	73,578	0,184	72,661	72,486	0,164
Media (\bar{x})	73,3666	73,3596	0,194	72,4911	72,2634	0,2156
Desviación típica (σ)	0,337136	0,489719	0,0323453	0,2641297	0,2706651	0,0324660

Tabla B.12. Tiempos de ejecución en la aplicación Canneal (2 hilos)

	2 hilos					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	44,33	74,914	0,18	43,894	74,125	0,176
Native #2	44,386	75,038	0,212	43,721	73,916	0,18
Native #3	44,664	75,478	0,208	43,977	74,249	0,208
Native #4	44,308	74,909	0,192	43,948	74,106	0,26
Native #5	44,67	75,296	0,192	43,882	74,033	0,196
Native #6	44,464	75,198	0,228	44,076	74,4	0,18
Native #7	44,521	75,174	0,232	44,644	74,999	0,22
Native #8	44,55	75,198	0,212	44,016	74,239	0,224
Native #9	44,249	74,706	0,22	43,92	74,162	0,26
Native #10	44,593	75,336	0,176	44,095	74,326	0,228
Media (\bar{x})	44,4735	75,1247	0,2052	44,0173	74,2555	0,2132
Desviación típica (σ)	0,1503405	0,232315	0,019418	0,2448205	0,2966191	0,03104405

Tabla B.13. Tiempos de ejecución en la aplicación Canneal (4 hilos)

	4 hilos					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	30,525	81,313	0,216	30,845	81,345	0,22
Native #2	30,807	81,969	0,192	30,805	81,383	0,176
Native #3	30,623	81,341	0,216	30,459	80,907	0,208
Native #4	30,769	81,828	0,172	30,327	80,756	0,216
Native #5	30,545	81,105	0,216	30,454	80,859	0,276
Native #6	30,726	81,796	0,212	30,496	80,977	0,231
Native #7	30,907	82,286	0,172	30,531	80,893	0,228
Native #8	30,733	81,865	0,18	30,518	80,944	0,212
Native #9	30,8	81,701	0,2	30,491	81,008	0,261
Native #10	30,537	81,04	0,236	30,37	80,717	0,259
Media (\bar{x})	30,6972	81,6244	0,2012	30,5296	80,9789	0,2287
Desviación típica (σ)	0,1324527	0,40547	0,0216682	0,1684228	0,2222598	0,0296762

Tabla B.14. Tiempos de ejecución en la aplicación Canneal (8 hilos)

	8 hilos					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	24,727	98,973	0,241	24,357	98,023	0,292
Native #2	24,426	98,153	0,22	24,319	97,661	0,233
Native #3	24,589	98,516	0,265	24,232	97,689	0,276
Native #4	24,356	98,145	0,228	24,269	97,759	0,24
Native #5	24,559	98,609	0,312	24,416	97,677	0,257
Native #6	24,569	98,66	0,224	24,396	98,118	0,341
Native #7	24,662	98,375	0,276	24,524	97,807	0,265
Native #8	24,38	97,87	0,313	24,359	97,937	0,281
Native #9	24,33	97,881	0,294	24,363	97,929	0,264
Native #10	24,387	97,963	0,285	24,524	98,034	0,292
Media (\bar{x})	24,4985	98,3145	0,2658	24,3759	97,8634	0,2741
Desviación típica (σ)	0,1399184	0,372357	0,0357764	0,0954875	0,1664413	0,0307262

Tabla B.15. Tiempos de ejecución en la aplicación Canneal (16 hilos)

	16 hilos					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	22,305	144,729	0,342	22,323	143,721	0,322
Native #2	22,227	143,208	0,325	22,571	143,853	0,325
Native #3	22,457	143,596	0,389	22,437	143,646	0,308
Native #4	22,439	143,485	0,359	22,267	143,449	0,362
Native #5	22,467	143,476	0,363	22,399	143,329	0,381
Native #6	22,56	143,788	0,354	22,495	143,615	0,497
Native #7	22,426	143,329	0,31	22,367	143,252	0,409
Native #8	22,517	143,938	0,33	22,323	143,782	0,407
Native #9	22,501	143,588	0,385	22,363	143,557	0,412
Native #10	22,359	143,301	0,281	22,393	143,642	0,406
Media (\bar{x})	22,4258	143,6438	0,3438	22,3938	143,5846	0,3829
Desviación típica (σ)	0,1018733	0,440997	0,0334624	0,0890103	0,1922678	0,0565419

B.4. Fluidanimate

Tabla B.16. Tiempos de ejecución en la aplicación Fluidanimate (1 hilo)

	1 hilo					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	156,242	156,135	0,084	155,659	155,539	0,096
Native #2	155,743	155,64	0,08	155,869	155,737	0,108
Native #3	155,824	155,714	0,088	155,743	155,607	0,112
Native #4	155,607	155,487	0,096	155,908	155,783	0,1
Native #5	155,894	155,76	0,112	156,277	156,152	0,1
Native #6	155,92	155,794	0,104	155,724	155,62	0,08
Native #7	155,759	155,631	0,08	156,512	156,391	0,096
Native #8	155,883	155,77	0,092	155,693	155,576	0,092
Native #9	155,813	155,682	0,092	155,658	155,526	0,104
Native #10	156,006	155,885	0,1	156,099	155,958	0,116
Media (\bar{x})	155,8691	155,7498	0,0928	155,9142	155,7889	0,1004
Desviación típica (σ)	0,170977	0,1732806	0,0104647	0,2925750	0,2913945	0,0104051

Tabla B.17. Tiempos de ejecución en la aplicación Fluidanimate (2 hilos)

	2 hilos					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	82,891	163,873	0,112	82,798	163,666	0,128
Native #2	83,008	164,006	0,104	82,736	163,708	0,12
Native #3	83,015	164,005	0,124	83,214	164,319	0,112
Native #4	83,382	164,778	0,116	82,732	163,512	0,108
Native #5	83,563	164,999	0,088	83,112	163,863	0,112
Native #6	82,751	163,671	0,116	82,887	163,966	0,108
Native #7	82,772	163,668	0,124	83,073	163,947	0,12
Native #8	82,954	163,776	0,108	83,115	164,399	0,132
Native #9	83,229	166,266	0,156	82,648	163,386	0,128
Native #10	82,907	163,746	0,168	83,369	164,427	0,1
Media (\bar{x})	83,0472	164,2788	0,1216	82,9684	163,9193	0,1168
Desviación típica (σ)	0,264570	0,8373299	0,0238662	0,2408472	0,3670997	0,0104647

Tabla B.18. Tiempos de ejecución en la aplicación Fluidanimate (4 hilos)

	4 hilos					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	45,446	177,377	0,156	45,368	176,885	0,136
Native #2	45,257	176,501	0,144	45,56	177,04	0,16
Native #3	45,495	177,643	0,144	45,374	177,163	0,148
Native #4	45,191	176,233	0,12	45,266	176,67	0,12
Native #5	45,352	177,326	0,128	45,427	177,293	0,132
Native #6	45,263	176,54	0,14	45,333	176,454	0,144
Native #7	45,32	176,418	0,168	45,301	176,815	0,128
Native #8	45,41	177,155	0,108	45,831	178,178	0,148
Native #9	45,355	176,66	0,116	45,281	176,566	0,14
Native #10	45,249	176,356	0,144	45,478	177,109	0,128
Media (\bar{x})	45,3338	176,8209	0,1368	45,4219	177,0173	0,1384
Desviación típica (σ)	0,0965203	0,5036600	0,0186476	0,1704519	0,489150	0,0119554

Tabla B.19. Tiempos de ejecución en la aplicación Fluidanimate (8 hilos)

	8 hilos					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	31,124	211,374	0,168	30,986	211,908	0,196
Native #2	31,42	212,287	0,196	30,995	212,44	0,156
Native #3	30,536	210,062	0,156	30,813	211,072	0,184
Native #4	30,964	211,524	0,236	30,989	212,419	0,196
Native #5	30,846	210,732	0,168	30,952	212,029	0,184
Native #6	31,403	211,444	0,192	31,049	212,622	0,156
Native #7	30,715	210,726	0,188	31,43	213,318	0,196
Native #8	31,182	211,116	0,184	31,103	212,705	0,164
Native #9	30,991	211,615	0,196	31,171	212,728	0,184
Native #10	30,925	211,52	0,176	31,099	212,947	0,2
Media (\bar{x})	31,0106	211,24	0,186	31,0587	212,4188	0,1816
Desviación típica (σ)	0,2815481	0,6128967	0,0220907	0,1633367	0,6265905	0,016991

Tabla B.20. Tiempos de ejecución en la aplicación Fluidanimate (16 hilos)

	16 hilos					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	22,91	313,365	0,348	23,159	314,503	0,313
Native #2	23,094	313,083	0,397	23,13	314,664	0,412
Native #3	22,985	312,84	0,376	23,319	315,103	0,389
Native #4	22,987	313,671	0,333	23,188	314,81	0,341
Native #5	23,153	314,258	0,269	23,197	315,531	0,346
Native #6	22,951	313,315	0,337	23,228	315,208	0,386
Native #7	22,981	312,988	0,317	22,913	314,016	0,324
Native #8	22,963	313,814	0,405	23,043	314,01	0,377
Native #9	23,024	314,029	0,345	23,086	315,973	0,437
Native #10	23,034	313,777	0,309	23,308	315,51	0,39
Media (\bar{x})	23,0082	313,514	0,3436	23,1571	314,9328	0,3715
Desviación típica (σ)	0,071437	0,4691711	0,0412558	0,1224531	0,6533559	0,0395509

B.5. Freqmine

Tabla B.21. Tiempos de ejecución en la aplicación Freqmine (1 hilo) (parte 1)

	1 hilo					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	207,628	207,404	0,196	210,724	211,511	0,18
Native #2	208,029	207,797	0,204	209,965	209,71	0,22
Native #3	207,7	207,456	0,216	209	208,7	0,264
Native #4	208,209	207,969	0,212	207,453	207,179	0,24
Native #5	207,61	207,386	0,196	208,048	207,838	0,176
Native #6	207,436	207,221	0,188	207,591	207,266	0,292
Native #7	208,224	207,987	0,208	209,168	208,93	0,204
Native #8	206,967	206,702	0,236	210,454	210,181	0,236
Native #9	208,895	208,624	0,244	209,291	209,036	0,22
Native #10	208,472	208,251	0,192	207,699	207,448	0,216
Media (\bar{x})	207,917	207,6797	0,2092	208,9393	208,7799	0,2248
Desviación típica (σ)	0,559293	0,5566355	0,0185759	1,2051266	1,4101805	0,0355177

Tabla B.22. Tiempos de ejecución en la aplicación Freqmine (1 hilo) (parte 2)

	1 hilo		
	Analizado dinámicamente		
	Real	User	Sys
Native #1	207,807	207,561	0,216
Native #2	207,827	207,583	0,216
Native #3	208,76	208,523	0,208
Native #4	208,549	208,323	0,2
Native #5	208,925	208,68	0,216
Native #6	208,347	208,082	0,236
Native #7	206,54	206,275	0,236
Native #8	207,739	207,438	0,272
Native #9	207,632	207,41	0,196
Native #10	207,603	207,39	0,184
Media (\bar{x})	207,9729	207,7265	0,218
Desviación típica (σ)	0,699811	0,705316	0,0250333

Tabla B.23. Tiempos de ejecución en la aplicación Freqmine (2 hilos) (parte 1)

	2 hilos					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	105,013	208,87	0,292	106,127	211,126	0,24
Native #2	105,969	210,819	0,268	106,037	210,875	0,324
Native #3	104,645	208,145	0,264	104,349	207,564	0,252
Native #4	104,48	207,767	0,332	105,747	210,289	0,296
Native #5	105,864	210,61	0,26	105,525	209,932	0,24
Native #6	104,191	207,281	0,252	104,402	207,627	0,296
Native #7	104,504	207,844	0,292	104,672	208,137	0,284
Native #8	104,13	207,059	0,344	106,06	210,961	0,276
Native #9	106,201	211,256	0,28	105,755	210,372	0,252
Native #10	104,526	207,919	0,284	106,445	211,711	0,272
Media (\bar{x})	104,9523	208,757	0,2868	105,5119	209,8594	0,2732
Desviación típica (σ)	0,773431	1,5597528	0,030231	0,7622594	1,5256831	0,0275874

Tabla B.24. Tiempos de ejecución en la aplicación Freqmine (2 hilos) (parte 2)

	2 hilos		
	Analizado dinámicamente		
	Real	User	Sys
Native #1	105,713	210,141	0,263
Native #2	105,487	209,66	0,275
Native #3	104,802	208,276	0,279
Native #4	104,653	208,016	0,248
Native #5	106,107	210,883	0,303
Native #6	105,367	209,384	0,322
Native #7	105,015	208,759	0,228
Native #8	104,894	208,437	0,304
Native #9	104,686	208,089	0,244
Native #10	104,594	207,827	0,33
Media (\bar{x})	105,1318	208,9472	0,2796
Desviación típica (σ)	0,51277779	1,02610805	0,03449058

Tabla B.25. Tiempos de ejecución en la aplicación Freqmine (4 hilos) (parte 1)

	4 hilos					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	54,782	216,365	0,324	54,788	216,325	0,312
Native #2	54,635	215,741	0,324	54,491	215,11	0,324
Native #3	54,582	215,48	0,344	54,815	216,345	0,312
Native #4	54,856	216,674	0,268	54,858	216,585	0,328
Native #5	54,57	215,468	0,336	54,299	214,332	0,32
Native #6	54,422	214,916	0,304	54,728	215,998	0,312
Native #7	54,586	215,482	0,352	54,223	213,977	0,308
Native #8	54,209	214,007	0,328	54,418	214,718	0,324
Native #9	54,411	214,857	0,332	54,512	215,16	0,328
Native #10	54,736	216,056	0,356	54,512	215,032	0,36
Media (\bar{x})	54,5789	215,5046	0,3268	54,5644	215,3582	0,3228
Desviación típica (σ)	0,193113	0,7802284	0,0255812	0,2223776	0,9068141	0,0149725

Tabla B.26. Tiempos de ejecución en la aplicación Freqmine (4 hilos) (parte 2)

	4 hilos		
	Analizado dinámicamente		
	Real	User	Sys
Native #1	54,53	215,001	0,282
Native #2	54,316	214,151	0,288
Native #3	53,938	212,512	0,331
Native #4	53,731	211,703	0,372
Native #5	54,219	213,713	0,334
Native #6	53,986	213,68	0,39
Native #7	54,915	216,493	0,308
Native #8	54,369	214,303	0,323
Native #9	54,388	214,338	0,305
Native #10	54,709	214,666	0,318
Media (\bar{x})	54,3101	214,056	0,3251
Desviación típica (σ)	0,36057191	1,31374029	0,03417423

Tabla B.27. Tiempos de ejecución en la aplicación Freqmine (8 hilos) (parte 1)

	8 hilos					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	29,328	227,253	0,504	29,244	226,425	0,44
Native #2	31,24	240,929	0,38	29,465	227,32	0,452
Native #3	29,852	229,87	0,42	29,223	226,495	0,396
Native #4	29,759	228,779	0,392	30,773	234,894	0,456
Native #5	29,5	226,516	0,416	29,561	229,573	0,452
Native #6	29,224	226,11	0,424	29,473	227,696	0,464
Native #7	29,27	226,348	0,48	29,944	230,421	0,436
Native #8	29,131	226,058	0,436	29,634	227,081	0,448
Native #9	29,338	226,566	0,452	29,251	227,127	0,4
Native #10	29,292	227,624	0,46	29,378	226,951	0,444
Media (\bar{x})	29,5934	228,6053	0,4364	29,5946	228,3983	0,4388
Desviación típica (σ)	0,6232244	4,5046823	0,038387	0,4679689	2,6306723	0,0229434

Tabla B.28. Tiempos de ejecución en la aplicación Freqmine (8 hilos) (parte 2)

	8 hilos		
	Analizado dinámicamente		
	Real	User	Sys
Native #1	29,081	224,412	0,408
Native #2	29,357	225,057	0,393
Native #3	29,324	224,438	0,428
Native #4	30,575	232,031	0,388
Native #5	30,851	231,868	0,467
Native #6	29,416	225,692	0,392
Native #7	30,548	234,674	0,42
Native #8	29,657	224,981	0,487
Native #9	29,274	226,084	0,48
Native #10	29,169	225,208	0,4
Media (\bar{x})	29,7252	227,4445	0,4263
Desviación típica (σ)	0,66594191	3,84125617	0,03807318

Tabla B.29. Tiempos de ejecución en la aplicación Freqmine (16 hilos) (parte 1)

	16 hilos					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	24,111	321,686	0,54	24,141	320,303	0,604
Native #2	24,962	321,819	0,588	24,128	320,671	0,64
Native #3	24,149	320,605	0,548	24,75	321,683	0,64
Native #4	24,194	320,82	0,6	24,039	319,961	0,74
Native #5	24,124	320,698	0,568	24,262	320,544	0,656
Native #6	24,127	320,696	0,488	24,01	320,275	0,596
Native #7	24,177	320,737	0,6	24,153	320,066	0,708
Native #8	24,244	320,32	0,576	24,238	321,205	0,592
Native #9	24,106	319,836	0,64	24,135	320,958	0,624
Native #10	24,051	319,849	0,58	24,078	319,843	0,576
Media (\bar{x})	24,2245	320,7066	0,5728	24,1934	320,5509	0,6376
Desviación típica (σ)	0,264534	0,6552211	0,0410874	0,2106488	0,5879821	0,0523984

Tabla B.30. Tiempos de ejecución en la aplicación Freqmine (16 hilos) (parte 2)

	16 hilos		
	Analizado dinámicamente		
	Real	User	Sys
Native #1	24,266	321,863	0,672
Native #2	24,143	321,641	0,592
Native #3	24,118	320,566	0,576
Native #4	24,197	320,916	0,612
Native #5	24,255	320,591	0,54
Native #6	24,234	321,266	0,632
Native #7	24,125	320,907	0,712
Native #8	24,136	321,218	0,708
Native #9	24,137	319,102	0,56
Native #10	24,236	320,048	0,544
Media (\bar{x})	24,1847	320,8118	0,6148
Desviación típica (σ)	0,05884452	0,80275827	0,0644167

B.6. Netstreamcluster

Tabla B.31. Tiempos de ejecución en la aplicación Netstreamcluster (1 hilo) (parte 1)

	1 hilo					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	235,156	255,159	0,893	237,621	257,578	1,028
Native #2	235,664	255,63	1,108	237,164	257,317	1,136
Native #3	240,484	260,973	0,832	237,688	257,727	1,279
Native #4	233,105	253,053	1,292	240,839	260,86	0,936
Native #5	238,09	258,112	0,96	239,161	259,666	1,186
Native #6	236,401	256,624	1,215	239,947	260,381	1,219
Native #7	233,714	253,524	1,121	237,01	257,737	1,105
Native #8	237,517	257,442	1,008	239,571	259,971	1,017
Native #9	239,995	260,388	1,178	236,86	257,054	1,313
Native #10	236,684	256,906	0,963	235,106	255,393	1,145
Media (\bar{x})	236,681	256,7811	1,057	238,0967	258,3684	1,1364
Desviación típica (σ)	2,430335	2,6096782	0,1489571	1,7374310	1,7521925	0,1192720

Tabla B.32. Tiempos de ejecución en la aplicación Netstreamcluster (1 hilo) (parte 2)

	1 hilo		
	Analizado dinámicamente		
	Real	User	Sys
Native #1	239,261	252,432	0,691
Native #2	236,329	255,544	0,835
Native #3	238,132	252,422	0,747
Native #4	238,567	251,319	0,983
Native #5	237,879	253,321	0,778
Native #6	238,031	252,321	0,842
Native #7	237,07	253,967	0,831
Native #8	238,97	254,862	0,695
Native #9	236,667	252,413	0,784
Native #10	238,64	255,714	0,996
Media (\bar{x})	237,9546	253,4315	0,8182
Desviación típica (σ)	0,98367533	1,5191029	0,10487961

Tabla B.33. Tiempos de ejecución en la aplicación Netstreamcluster (2 hilos) (parte 1)

	2 hilos					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	123,398	255,227	1,025	124,847	257,05	0,836
Native #2	124,384	257,221	1,126	124,099	256,626	0,863
Native #3	124,217	256,501	1,156	123,726	255,136	1,351
Native #4	123,95	256,553	1,081	124,423	256,446	1,25
Native #5	123,748	255,848	0,878	124,27	256,907	0,973
Native #6	124,346	256,488	0,773	124,338	257,103	1,02
Native #7	123,816	256,792	1,24	123,803	255,875	1,232
Native #8	124,796	257,464	1,146	124,576	257,096	1,072
Native #9	123,841	255,577	1,176	124,153	256,509	0,744
Native #10	124,169	256,754	1,167	124,886	257,731	1,468
Media (\bar{x})	124,0665	256,4425	1,0768	124,3121	256,6479	1,0809
Desviación típica (σ)	0,397179	0,7044596	0,1462910	0,3906241	0,7267572	0,2381910

Tabla B.34. Tiempos de ejecución en la aplicación Netstreamcluster (2 hilos) (parte 2)

	2 hilos		
	Analizado dinámicamente		
	Real	User	Sys
Native #1	124,263	255,224	0,95
Native #2	124,404	256,822	1,009
Native #3	124,023	255,242	0,956
Native #4	123,702	254,882	1,015
Native #5	124,555	255,894	0,65
Native #6	124,295	256,079	1,019
Native #7	123,959	255,084	0,83
Native #8	124,047	256,291	0,799
Native #9	124,65	255,952	0,7
Native #10	124,629	254,466	1,023
Media (\bar{x})	124,2527	255,5936	0,8951
Desviación típica (σ)	0,31626749	0,72628525	0,14039346

Tabla B.35. Tiempos de ejecución en la aplicación Netstreamcluster (4 hilos) (parte 1)

	4 hilos					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	69,694	278,172	1,084	69,666	277,959	1,336
Native #2	69,494	278,137	1,297	69,34	278,105	1,335
Native #3	69,559	277,565	1,141	69,542	278,182	1,147
Native #4	69,286	277,239	1,187	69,667	278,569	1,047
Native #5	69,752	278,598	1,225	69,483	277,192	1,373
Native #6	69,599	277,891	1,263	69,57	277,566	1,325
Native #7	69,572	278,253	1,064	69,658	277,877	0,997
Native #8	69,583	277,8	1,442	69,762	277,521	1,393
Native #9	69,388	277,522	1,082	69,584	277,737	1,029
Native #10	69,592	277,556	0,907	69,795	277,828	1,312
Media (\bar{x})	69,5519	277,8733	1,1692	69,6067	277,8536	1,2294
Desviación típica (σ)	0,135973	0,4157034	0,1484945	0,1340970	0,3858330	0,1563800

Tabla B.36. Tiempos de ejecución en la aplicación Netstreamcluster (4 hilos) (parte 2)

	4 hilos		
	Analizado dinámicamente		
	Real	User	Sys
Native #1	69,441	277,435	1,03
Native #2	70,026	278,766	1,232
Native #3	69,716	278,062	1,199
Native #4	69,616	278,015	0,881
Native #5	69,644	277,754	1,126
Native #6	69,963	277,761	1,134
Native #7	69,9	277,779	1,038
Native #8	69,769	277,885	0,981
Native #9	69,592	277,641	1,125
Native #10	69,46	276,922	1,134
Media (\bar{x})	69,7127	277,802	1,088
Desviación típica (σ)	0,20149058	0,47006832	0,1055483

Tabla B.37. Tiempos de ejecución en la aplicación Netstreamcluster (8 hilos) (parte 1)

	8 hilos					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	52,762	402,975	1,405	52,097	398,23	1,645
Native #2	52,39	402,103	1,486	51,65	396,216	1,516
Native #3	52,825	403,404	1,165	52,197	401,062	1,713
Native #4	52,451	401,882	1,442	52,455	401,559	1,405
Native #5	52,318	399,54	1,453	52,438	401,089	1,792
Native #6	52,835	403,408	1,701	52,3	399,702	1,504
Native #7	52,507	402,507	1,415	52,019	398,516	1,583
Native #8	52,839	403,776	1,81	52,379	399,698	1,467
Native #9	52,532	401,827	1,405	52,387	400,922	1,455
Native #10	52,217	399,358	1,347	52,391	400,67	1,457
Media (\bar{x})	52,5676	402,078	1,4629	52,2313	399,7664	1,5537
Desviación típica (σ)	0,2320211	1,5398743	0,1793243	0,251935	1,6776756	0,1264779

Tabla B.38. Tiempos de ejecución en la aplicación Netstreamcluster (8 hilos) (parte 2)

	8 hilos		
	Analizado dinámicamente		
	Real	User	Sys
Native #1	52,298	402,205	1,612
Native #2	51,742	401,511	1,32
Native #3	51,718	400,247	1,533
Native #4	51,802	399,542	1,407
Native #5	52,081	401,734	1,392
Native #6	51,866	400,64	1,382
Native #7	51,65	399,066	1,265
Native #8	52,558	403,111	1,364
Native #9	52,194	401,818	1,818
Native #10	51,898	401,899	1,799
Media (\bar{x})	51,9807	401,1773	1,4892
Desviación típica (σ)	0,29378603	1,26640015	0,19532014

Tabla B.39. Tiempos de ejecución en la aplicación Netstreamcluster (16 hilos) (parte 1)

	16 hilos					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	51,409	717,826	3,957	50,357	715,559	3,802
Native #2	51,906	718,706	4,344	50,892	714,917	3,64
Native #3	51,781	717,461	3,972	53,432	717,205	4,695
Native #4	50,994	718,478	3,803	51,555	716,487	3,976
Native #5	51,046	718,258	3,807	51,789	716,153	4,408
Native #6	50,973	718,616	3,729	51,509	715,63	4,211
Native #7	50,951	718,313	3,602	52,683	715,877	5,246
Native #8	52,151	718,631	4,86	50,835	713,715	4,381
Native #9	52,422	719,035	4,038	51,693	715,603	4,57
Native #10	51,485	717,451	3,424	51,232	714,615	4,048
Media (\bar{x})	51,5118	718,2775	3,9536	51,5977	715,5761	4,2977
Desviación típica (σ)	0,5339660	0,5367008	0,4055397	0,9037390	0,9850069	0,471642

Tabla B.40. Tiempos de ejecución en la aplicación Netstreamcluster (16 hilos) (parte 2)

	16 hilos		
	Analizado dinámicamente		
	Real	User	Sys
Native #1	51,027	718,694	4,631
Native #2	51,372	717,111	3,581
Native #3	52,141	717,554	3,773
Native #4	51,337	718,287	4,016
Native #5	50,971	716,954	3,899
Native #6	51,997	718,694	3,613
Native #7	51,905	718,047	3,504
Native #8	51,825	719,213	3,716
Native #9	51,742	717,896	3,828
Native #10	50,991	718,264	3,836
Media (\bar{x})	51,5308	718,0714	3,8397
Desviación típica (σ)	0,44499958	0,71741081	0,31827104

B.7. Streamcluster

Tabla B.41. Tiempos de ejecución en la aplicación Streamcluster (1 hilo) (parte 1)

	1 hilo					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	237,648	237,549	0,032	239,193	239,099	0,016
Native #2	241,237	241,155	0,02	239,392	239,288	0,032
Native #3	236,513	236,419	0,02	238,357	238,24	0,036
Native #4	237,837	237,743	0,028	237,651	237,551	0,008
Native #5	237,068	236,979	0,02	237,106	237,004	0,028
Native #6	235,936	235,855	0,012	235,566	235,46	0,024
Native #7	237,933	237,837	0,028	239,117	239,021	0,02
Native #8	239,298	239,208	0,032	239,225	239,127	0,024
Native #9	237,63	237,529	0,02	236,593	236,493	0,024
Native #10	238,439	238,346	0,032	239,463	239,368	0,02
Media (\bar{x})	237,9539	237,862	0,0244	238,1663	238,0651	0,0232
Desviación típica (σ)	1,489451	1,4908406	0,0069153	1,3713549	1,3736498	0,0079554

Tabla B.42. Tiempos de ejecución en la aplicación Streamcluster (1 hilo) (parte 2)

	1 hilo					
	Analizado dinámicamente			Analizado estáticamente		
	Real	User	Sys	Real	User	Sys
Native #1	238,711	238,605	0,024	224,119	223,987	0,032
Native #2	236,817	236,705	0,04	224,324	224,185	0,044
Native #3	238,275	238,176	0,028	224,544	224,4	0,044
Native #4	238,494	238,395	0,028	224,208	224,068	0,04
Native #5	239,409	239,327	0,012	223,424	223,284	0,036
Native #6	238,534	238,451	0,012	223,621	223,483	0,032
Native #7	238,026	238,946	0,008	224,364	224,217	0,048
Native #8	238,788	238,678	0,032	224,033	223,9	0,032
Native #9	237,182	237,095	0,016	223,732	223,596	0,04
Native #10	236,995	236,904	0,02	223,1	222,962	0,032
Media (\bar{x})	238,1231	238,1282	0,022	223,9469	223,8082	0,038
Desviación típica (σ)	0,85922063	0,907063	0,0101980	0,46199	0,46053	0,00603

Tabla B.43. Tiempos de ejecución en la aplicación Streamcluster (2 hilos) (parte 1)

	2 hilos					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	124,573	246,773	0,048	124,006	246,042	0,06
Native #2	123,939	245,902	0,068	124,13	246,193	0,044
Native #3	124,353	246,72	0,064	123,662	245,345	0,044
Native #4	124,017	246,123	0,064	124,661	247,368	0,068
Native #5	123,718	245,391	0,04	124,2	246,681	0,04
Native #6	124,968	247,275	0,036	123,891	245,767	0,056
Native #7	123,564	245,568	0,04	124,075	246,242	0,044
Native #8	124,581	247,085	0,048	124,373	246,19	0,04
Native #9	124,622	246,584	0,068	123,835	245,682	0,044
Native #10	124,57	247,013	0,04	123,886	245,912	0,052
Media (\bar{x})	124,2905	246,4434	0,0516	124,0719	246,1422	0,0492
Desviación típica (σ)	0,455708	0,6584667	0,0129888	0,2893020	0,5636030	0,0094375

Tabla B.44. Tiempos de ejecución en la aplicación Streamcluster (2 hilos) (parte 2)

	2 hilos					
	Analizado dinámicamente			Analizado estáticamente		
	Real	User	Sys	Real	User	Sys
Native #1	124,212	247,268	0,056	116,784	231,873	0,068
Native #2	124,065	247,29	0,068	117,213	232,29	0,06
Native #3	124,642	247,432	0,06	117,279	232,817	0,076
Native #4	124,056	246,833	0,056	117,381	232,327	0,072
Native #5	124,582	247,116	0,064	117,11	232,037	0,084
Native #6	123,726	246,469	0,032	117,697	233,161	0,072
Native #7	123,878	245,848	0,068	117,586	233,311	0,092
Native #8	123,408	245,177	0,076	117,257	232,83	0,048
Native #9	122,799	244,816	0,064	116,645	231,543	0,08
Native #10	123,957	245,825	0,044	116,754	231,696	0,092
Media (\bar{x})	123,9325	246,4074	0,0588	117,1706	232,3885	0,0744
Desviación típica (σ)	0,541838	0,940154	0,012795	0,35243	0,61675	0,01375

Tabla B.45. Tiempos de ejecución en la aplicación Streamcluster (4 hilos) (parte 1)

	4 hilos					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	69,529	271,67	0,14	69,392	271,334	0,176
Native #2	69,38	271,193	0,156	69,405	271,466	0,2
Native #3	69,598	271,864	0,136	69,282	271,488	0,192
Native #4	69,492	271,43	0,208	68,93	271,78	0,156
Native #5	69,527	271,743	0,184	69,355	271,535	0,18
Native #6	69,406	271,388	0,144	69,292	271,12	0,172
Native #7	69,409	271,365	0,16	69,059	271,189	0,16
Native #8	69,52	271,796	0,172	69,109	271,605	0,168
Native #9	69,563	271,928	0,124	69,016	270,997	0,14
Native #10	69,502	271,369	0,148	68,913	270,578	0,192
Media (\bar{x})	69,4926	271,5746	0,1572	69,1753	271,3092	0,1736
Desviación típica (σ)	0,0719416	0,254443	0,0250191	0,1912787	0,3495510	0,0184944

Tabla B.46. Tiempos de ejecución en la aplicación Streamcluster (4 hilos) (parte 2)

	4 hilos					
	Analizado dinámicamente			Analizado estáticamente		
	Real	User	Sys	Real	User	Sys
Native #1	69,598	271,477	0,088	65,36	256,58	0,156
Native #2	69,55	271,338	0,152	65,483	257,163	0,144
Native #3	69,456	270,966	0,136	65,446	256,593	0,248
Native #4	69,692	271,863	0,148	65,427	256,649	0,248
Native #5	69,673	270,795	0,196	65,461	256,88	0,164
Native #6	69,629	271,572	0,204	65,495	256,958	0,148
Native #7	69,577	271,432	0,18	65,575	257,139	0,132
Native #8	69,616	271,497	0,136	65,259	256,159	0,14
Native #9	69,405	271,779	0,192	65,469	256,953	0,156
Native #10	69,404	271,57	0,168	65,769	257,562	0,192
Media (\bar{x})	69,56	271,4289	0,16	65,4744	256,8636	0,1728
Desviación típica (σ)	0,104933	0,33044	0,035377	0,13339	0,389671	0,042825

Tabla B.47. Tiempos de ejecución en la aplicación Streamcluster (8 hilos) (parte 1)

	8 hilos					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	52,219	394,428	0,484	51,848	393,54	0,753
Native #2	52,155	393,819	0,599	52,158	393,837	0,64
Native #3	51,935	393,784	0,526	51,801	393,271	0,685
Native #4	52,126	394,498	0,576	51,884	393,692	0,617
Native #5	52,108	394,113	0,491	52,291	393,786	0,745
Native #6	51,987	393,602	0,674	52,032	393,4	0,677
Native #7	52,213	394,47	0,644	52,201	393,132	0,555
Native #8	52,247	393,631	0,675	52,148	393,032	0,48
Native #9	52,076	393,63	0,536	52,245	393,411	0,568
Native #10	52,055	394,02	0,623	52,273	393,514	0,689
Media (\bar{x})	52,1121	393,9995	0,5828	52,0881	393,4615	0,6409
Desviación típica (σ)	0,101890	0,3614994	0,0715647	0,1844128	0,2675012	0,0869986

Tabla B.48. Tiempos de ejecución en la aplicación Streamcluster (8 hilos) (parte 2)

	8 hilos					
	Analizado dinámicamente			Analizado estáticamente		
	Real	User	Sys	Real	User	Sys
Native #1	51,706	393,962	0,521	48,615	371,564	0,419
Native #2	50,773	393,925	0,557	49,042	374,067	0,56
Native #3	52,278	394,248	0,4	48,892	372,702	0,387
Native #4	52,037	393,851	0,368	48,862	372,55	0,455
Native #5	51,948	394,195	0,499	49,027	373,446	0,575
Native #6	51,223	394,049	0,511	49,081	373,329	0,419
Native #7	52,085	393,634	0,431	48,874	372,47	0,463
Native #8	51,625	393,67	0,52	49,014	373,433	0,439
Native #9	51,767	393,663	0,64	49,001	373,258	0,343
Native #10	52,062	393,921	0,654	48,777	372,659	0,459
Media (\bar{x})	51,7504	393,9118	0,5101	48,9185	336,9478	0,4519
Desviación típica (σ)	0,4546	0,215012	0,093507	0,14428	0,7101	0,071044

Tabla B.49. Tiempos de ejecución en la aplicación Streamcluster (16 hilos) (parte 1)

	16 hilos					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	53,056	741,659	2,749	53,107	742,284	2,396
Native #2	52,698	742,191	1,998	52,966	742,281	2,261
Native #3	52,945	743,451	2,106	53,374	742,588	2,371
Native #4	53,518	741,841	2,576	52,488	742,379	2,37
Native #5	54,455	743,566	2,462	52,07	741,482	2,178
Native #6	52,822	741,729	2,531	52,661	742,578	2,32
Native #7	53,492	742,842	2,623	52,821	741,797	2,142
Native #8	52,877	743,389	2,459	53,208	741,935	2,49
Native #9	52,937	743,267	2,186	52,951	741,432	2,072
Native #10	52,951	743,391	2,534	53,921	742,164	2,458
Media (\bar{x})	53,1751	742,7326	2,4224	52,9567	742,092	2,3058
Desviación típica (σ)	0,524018	0,7902649	0,2435310	0,5057056	0,4165631	0,1388994

Tabla B.50. Tiempos de ejecución en la aplicación Streamcluster (16 hilos) (parte 2)

	16 hilos					
	Analizado dinámicamente			Analizado estáticamente		
	Real	User	Sys	Real	User	Sys
Native #1	52,682	741,45	1,779	49,822	692,851	2,168
Native #2	53,048	742,546	2,883	49,797	693,557	2,438
Native #3	52,959	741,8	2,178	50,072	692,457	2,3
Native #4	52,817	743,087	2,644	50,007	694,276	2,145
Native #5	53,215	742,203	2,456	49,93	693,986	2,164
Native #6	52,719	742,214	2,494	50,883	692,708	2,456
Native #7	52,758	742,097	2,434	50,575	693,805	1,883
Native #8	53,096	743,083	1,977	49,957	693,582	2,076
Native #9	52,679	743,271	2,247	49,867	692,493	2,542
Native #10	52,862	744,099	2,233	49,947	692,767	2,023
Media (\bar{x})	52,8835	742,585	2,3325	50,0857	693,2482	2,2195
Desviación típica (σ)	0,1879499	0,7951575	0,3206837	0,356252	0,666249	0,210259

B.8. Swaptions

Tabla B.51. Tiempos de ejecución en la aplicación Swaptions (1 hilo)

	1 hilo					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	80,306	80,3	0	79,913	79,906	0
Native #2	78,945	78,937	0	79,956	79,95	0
Native #3	80,044	80,038	0	79,668	79,661	0
Native #4	79,592	79,586	0	79,332	79,322	0,004
Native #5	80,376	80,369	0	78,983	78,973	0,004
Native #6	80,707	80,701	0	79,087	79,077	0,004
Native #7	79,675	79,665	0,004	79,565	79,558	0
Native #8	79,566	79,561	0	79,104	79,083	0,004
Native #9	80,723	80,713	0,004	79,291	79,281	0,004
Native #10	79,807	79,796	0,004	79,119	79,109	0,004
Media (\bar{x})	79,9741	79,9666	0,0012	79,4018	79,392	0,0024
Desviación típica (σ)	0,564066	0,564126	0,001932	0,3537515	0,3563790	0,0020655

Tabla B.52. Tiempos de ejecución en la aplicación Swaptions (2 hilos)

	2 hilos					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	40,912	81,783	0,032	40,335	80,642	0,02
Native #2	40,775	81,525	0,016	40,479	80,921	0,028
Native #3	41,429	82,779	0,072	40,356	80,681	0,02
Native #4	40,748	81,452	0,036	41,046	81,028	0,056
Native #5	40,984	81,903	0,056	40,311	80,566	0,048
Native #6	40,927	81,819	0,028	41,586	81,137	0,028
Native #7	40,945	81,844	0,036	41,045	81,055	0,028
Native #8	40,969	81,915	0,016	40,327	80,602	0,044
Native #9	40,902	81,771	0,02	40,442	80,865	0,012
Native #10	40,873	81,722	0,016	40,452	80,876	0,02
Media (\bar{x})	40,9464	81,8513	0,0328	40,6379	80,8373	0,0304
Desviación típica (σ)	0,186335	0,359350	0,018647	0,4351363	0,2039586	0,0142610

Tabla B.53. Tiempos de ejecución en la aplicación Swaptions (4 hilos)

	4 hilos					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	20,816	83,24	0,012	20,692	82,659	0,092
Native #2	20,865	83,432	0,016	20,682	82,702	0,016
Native #3	21,064	84,206	0,04	20,95	83,765	0,024
Native #4	21,468	85,66	0,196	20,583	83,285	0,036
Native #5	20,851	83,375	0,016	20,476	81,883	0,008
Native #6	20,988	83,922	0,016	20,52	82,061	0,008
Native #7	21,215	84,683	0,164	21,163	84,456	0,18
Native #8	20,645	82,546	0,02	20,98	83,774	0,132
Native #9	20,768	83,049	0,012	20,677	82,646	0,048
Native #10	20,901	83,562	0,028	20,589	83,309	0,032
Media (\bar{x})	20,9581	83,7675	0,052	20,7312	83,054	0,0576
Desviación típica (σ)	0,2394124	0,896366	0,068404	0,2248549	0,8104589	0,0583993

Tabla B.54. Tiempos de ejecución en la aplicación Swaptions (8 hilos)

	8 hilos					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	11,513	91,72	0,288	11,681	92,572	0,628
Native #2	11,337	90,473	0,172	11,927	94,265	0,856
Native #3	11,807	93,867	0,456	12,087	94,128	0,484
Native #4	11,196	89,455	0,08	11,377	90,693	0,256
Native #5	11,792	93,627	0,596	12,174	94,311	0,88
Native #6	11,485	91,624	0,192	11,371	90,41	0,428
Native #7	12,007	94,795	0,913	11,528	91,921	0,208
Native #8	12,439	94,938	0,456	11,353	90,54	0,236
Native #9	11,699	92,997	0,488	11,429	91,181	0,204
Native #10	11,495	91,564	0,316	11,294	90,045	0,232
Media (\bar{x})	11,677	92,506	0,3957	11,6221	92,0066	0,4412
Desviación típica (σ)	0,3596291	1,8326957	0,243279	0,327700	1,7059716	0,2650621

Tabla B.55. Tiempos de ejecución en la aplicación Swaptions (16 hilos)

	16 hilos					
	Original			Modernizado		
	Real	User	Sys	Real	User	Sys
Native #1	10,089	151,858	1,986	9,894	150,144	1,537
Native #2	10,199	153,087	1,849	10,139	152,515	1,597
Native #3	9,695	146,818	1,345	10,232	153,732	1,846
Native #4	9,891	149,789	1,353	10,043	151,316	2,017
Native #5	10,028	149,569	1,664	10,034	151,335	1,422
Native #6	10,011	149,846	1,625	9,319	146,857	1,673
Native #7	10,054	152	1,027	10,031	150,194	1,22
Native #8	9,884	148,44	1,363	10,228	153,872	1,174
Native #9	9,365	142,812	0,64	9,642	150,525	1,896
Native #10	9,679	147,099	1,223	10,156	152,268	1,898
Media (\bar{x})	9,8895	149,1318	1,4075	9,9718	151,2758	1,628
Desviación típica (σ)	0,248074	3,0247109	0,3971308	0,2884132	2,0504505	0,2924707