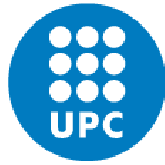


UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Reorganització del runtime Nanos++

TREBALL FINAL DE GRAU

Grau en Enginyeria Informàtica
menció en Enginyeria de Computadors

Autor
Jaume Bosch Pons

Director
Dr. Carlos Alvarez Martinez

Codirector
Dr. Daniel Jimenez Gonzalez

Juny 2015

Abstract

Due to the end of Dennard's Law, the current large computation systems started to integrate thousands of processors because the upgrade of computation capacity. The importance of parallel programing is growing year after year and with it the parallel programing models relevance. OmpSs is one of these models based on specification of tasks that can run concurrently. The execution order correctness is defined by dependencies between different tasks. In these models, the dependencies management that is based on locks ends up limiting application parallelism when many processing units are used. This is because as more elements requesting resource access, more contention and more time is needed to get the exclusivity. In OmpSs runtime, this problem appears on dependencies graph resource. This project aims to reorganize this system with the objective of reducing contention, increasing parallelism and achieving better use of computing resources.

Resum

Degut a la fi de la Llei de Dennard, els grans sistemes de computació actuals han evolucionat cap a la integració de grans quantitats de processadors per tal d'incrementar la seva capacitat de càlcul. Per aquest motiu, la importància de la programació paral·lela ha augmentat en els últims anys i amb ella la dels models de programació paral·lels. OmpSs n'és un d'ells que es basa en l'especificació de tasques que poden executar-se al mateix temps, a més de les dependències entre elles per determinar-ne l'odre correcte d'execució. En aquests models, la gestió d'aquestes dependències en entorns amb moltes unitats de procés provoca que el 'runtime' acabi limitant el paral·lelisme de les aplicacions. Això és deu a la contenció que augmenta amb el nombre d'elements que volen accedir al mateix temps a un mateix recurs, en aquest cas el graf amb les dependències entre les tasques. Aquest projecte proposa una reorganització d'aquest sistema amb l'objectiu de reduir aquesta contenció, augmentar el paral·lelisme i obtenir un millor aprofitament dels recursos.

Resumen

Debido al fin de la Ley de Dennard, los grandes sistemas de computación actuales han evolucionado hacia una integración masiva de procesadores para incrementar su capacidad de cómputo. Por este motivo, la importancia de la programación paralela ha aumentado en los últimos años y con ella la de los modelos de programación paralelos. OmpSs es uno de ellos que tiene su base en la especificación de tareas que pueden ejecutarse al mismo tiempo, además de las dependencias entre ellas para determinar el orden de ejecución correcto. En estos modelos, la gestión de estas dependencias en entornos con muchas unidades de procesamiento hace que el 'runtime' limite el paralelismo de las aplicaciones. Esto se debe a la contención que se ve incrementada con el número de elementos que quieren acceder al mismo tiempo a un mismo recurso, en este caso el grafo con las dependencias entre las tareas. Este proyecto propone una reorganización de este sistema con el objetivo de reducir la citada contención, aumentando el paralelismo y obteniendo un mejor aprovechamiento de les recursos.

Agraiments

Gràcies a en Dani Jimenez i a en Carlos Alvarez per donar-me la possibilitat de portar a terme aquest projecte i completar amb ell i ells una etapa. Gràcies també per tots els consells i recomanacions.

Gràcies a tots els companys de l'InLab, als del BSC i a tots els amics que m'han sofert en aquests mesos. Gràcies a tots pel vostre suport durant aquest període.

Gràcies a tota la meva família, i en especial als meus pares, pel suport en aquests quatre anys i per confiar amb mi. Gràcies per donar-me la possibilitat de cursar aquest grau i realitzar aquest treball, sense vosaltres no hauria estat possible.

Gràcies.

Índex

1 Introducció **1**

1.1 Motivació . . . 1

1.1.1 Problemàtica . . . 3

1.2 Actors implicats . . . 4

1.3 Estat de l'art . . . 5

1.4 Marc legal . . . 5

1.5 Objectiu . . . 6

1.6 Abast . . . 6

1.7 Obstacles . . . 7

1.7.1 Codi existent . . . 7

1.7.2 Introducció d'errors . . . 7

1.7.3 Correcció d'errors i/o problemes . . . 8

1.8 Metodologia i rigor . . . 8

1.8.1 Mètode de treball . . . 8

1.8.2 Eines de treball . . . 8

1.8.3 Eines de seguiment . . . 9

1.8.4 Mètodes de validació . . . 9

2 Planificació del projecte **10**

2.1 Planificació temporal . . . 10

2.1.1 Descripció de les tasques . . . 10

2.1.2 Dependències i seqüència de les tasques . . . 13

2.1.3 Valoració d'alternatives i pla d'acció . . . 16

2.2 Llistat de recursos . . . 16

2.3 Planificació econòmica . . . 18

2.3.1 Pressupost . . . 18

2.3.2 Control de costos . . . 20

2.4 Sostenibilitat . . . 20

2.5 Revisió de la planificació . . . 22

2.5.1 Canvis respecte a la planificació inicial . . . 22
2.5.2 Implicacions . . . 23

3 Model de programació Base: OmpSs **25**
3.1 Característiques generals . . . 25
3.2 Mercurium . . . 28
3.3 Nanos++ . . . 29

4 Disseny d'un model centralitzat **32**
4.1 Model operacional . . . 32
4.1.1 Sincronisme . . . 33
4.1.2 Gestió de cues . . . 35
4.2 Garanties . . . 36

5 Implementació **37**
5.1 Identificació de les regions de codi implicades . . . 37
5.1.1 Creació de tasques . . . 38
5.1.2 Finalització de tasques . . . 38
5.1.3 Resum de regions o peticions . . . 40
5.2 Creació del gestor . . . 40
5.2.1 Valoració d'alternatives . . . 40
5.2.2 Estructures de dades . . . 42
5.2.3 Peticions . . . 43
5.2.4 Thread extra . . . 46
5.2.5 Altres modificacions . . . 48
5.3 Optimització . . . 49
5.3.1 Regions d'exclusió mútua . . . 49
5.3.2 Balanceig de càrrega . . . 49
5.3.3 Fusió de peticions . . . 51

6 Avaluació del rendiment **52**
6.1 Entorns d'execució . . . 52
6.1.1 Arvei . . . 53
6.1.2 Knights . . . 54
6.2 Aplicacions sintètiques . . . 56
6.2.1 Test 1 . . . 56
6.2.2 Test 2 . . . 57
6.3 Aplicacions reals . . . 58
6.3.1 Cholesky . . . 58

6.3.2 Matrix Multiply 58
6.3.3 Sparse LU 59
6.4 Anàlisi de les diferents versions 59
6.4.1 Eliminació de les regions d'exclusió mútua 59
6.4.2 Fusió de peticions 61
6.4.3 Nanos original vs Nanos centralitzat 61
6.5 Anàlisi del rendiment a les aplicacions 63
6.5.1 Test 1 63
6.5.2 Test 2 66
6.5.3 Cholesky 71
6.5.4 Matrix Multiply 74
6.5.5 Sparse LU 78

7 Conclusions **81**
7.1 Resultats 81
7.2 Limitacions 82
7.3 Treball futur 82

Bibliografia **84**

A Temps d'execució a Arvei **86**

B Temps d'execució als MIC **106**

Índex de figures

2.1 Diagrama de Gantt amb el seqüenciament de les tasques 15
2.2 Diagrama de Gantt amb el seqüenciament final de les tasques 24

3.1 Fragment d'exemple de l'ús de la clàusula 'task' [1] 26
3.2 Fragment d'exemple de l'ús de la clàusula 'task' a declaracions [1] 26
3.3 Fragment d'exemple de l'ús de la clàusula 'task' amb dependències [1] 28
3.4 Graf de dependències [1] 28
3.5 Proces de compilació de Mercurium [2] 29
3.6 Estructura del 'runtime' Nanos++ [2] 31

4.1 Representació gràfica del model original d'administració de les tasques 33
4.2 Representació gràfica del model centralitzat d'administració de les tasques . . . 33
4.3 Diagrama de flux d'un 'worker' síncron 34
4.4 Diagrama de flux d'un 'worker' asíncron 34
4.5 Diagrama d'execució original i amb el model proposat 35
4.6 Diagrama de flux del bucle gestor dependències 36

5.1 Declaració del tipus DASTWork 42
5.2 Declaració del tipus DASTWorkType 43
5.3 Codi de la funció System::submitWithDependencies 44
5.4 Codi de la funció Scheduler::finishWork 44
5.5 Codi de la funció WorkDescriptor::done 45
5.6 Codi del bucle del DAST 47
5.7 Fragment de codi de la funció Scheduler::wakeUp 48
5.8 Traces d'execució d'una aplicació amb un mal balanceig del DAST 50
5.9 Traces d'execució d'una aplicació amb el balanceig del DAST corregit 51

6.1 Representació de l'estructura de dos processadors NUMA 54
6.2 Extensió d'una CPU Intel Xenon amb un coprocessador MIC [3] 55
6.3 Pseudocodi del Test 1 57
6.4 Pseudocodi del Test 2 58

6.5 Speedup 'Test 1' a Arvei. PARALLEL_CREATIONS: 2. TASK_SIZE: 25000 (esq.) i 50000 (dta.) . . . 64
6.6 Speedup 'Test 1' a Arvei. PARALLEL_CREATIONS: 2. TASK_SIZE: 100000 (esq.) i 200000 (dta.) . . . 64
6.7 Speedup 'Test 1' a Arvei. PARALLEL_CREATIONS: 3 (esq.) i 4 (dta.). TASK_SIZE: 100000 . . . 65
6.8 Speedup 'Test 1' a Knights. TASK_SIZE: 25000. Workers per core: 1 (esq.) i 4 (dta.) . . . 65
6.9 Speedup 'Test 1' a Knights. TASK_SIZE: 100000. Workers per core: 1 (esq.) i 4 (dta.) . . . 66
6.10 Speedup 'Test 2' a Arvei. PRALLEL_CREATIONS: 2. TASK_SIZE: 25000 (esq.) i 50000 (dta.) . . . 67
6.11 Speedup 'Test 2' a Arvei. PRALLEL_CREATIONS: 2. TASK_SIZE: 100000 (esq.) i 200000 (dta.) . . . 67
6.12 Speedup 'Test 2' a Arvei. PRALLEL_CREATIONS: 3 (esq.) i 4 (dta.). TASK_SIZE: 100000 . . . 68
6.13 Speedup 'Test 2' a Arvei. Comparativa diferents PARALLEL_CREATIONS (esq.: original, dta.: DAST) . . . 69
6.14 Traces del 'Test 2' a Arvei amb el nombre de 'ready tasks' (dalt: original, baix: DAST) . . . 69
6.15 Traces del 'Test 2' a Arvei amb les tasques en execució (dalt: original, baix: DAST) 70
6.16 Speedup 'Test 2' a Knights. TASK_SIZE: 25000. Workers per core: 1 (esq.) i 4 (dta.) . . . 70
6.17 Speedup 'Test 2' a Knights. TASK_SIZE: 100000. Workers per core: 1 (esq.) i 4 (dta.) . . . 71
6.18 Speedup 'Cholesky' a Arvei amb diferents BLOCK_SIZE. MATRIX_SIZE: 2048 . . . 72
6.19 Speedup 'Cholesky' a Arvei amb diferents MATRIX_SIZE i BLOCK_SIZE . . . 72
6.20 Speedup 'Cholesky' a Knights. BLOCK_SIZE: 128. Workers per core: 1 (esq.) i 4 (dta.) . . . 73
6.21 Speedup 'Cholesky' a Knights. BLOCK_SIZE: 256. Workers per core: 1 (esq.) i 4 (dta.) . . . 73
6.22 Speedup 'Cholesky' a Knights. BLOCK_SIZE: 512. Workers per core: 1 (esq.) i 4 (dta.) . . . 74
6.23 Speedup 'Matrix Multiply' a Arvei. BLOCK_SIZE: 64. MATRIX_SIZE: 2048 (esq.) i 4096 (dta.) . . . 75
6.24 Speedup 'Matrix Multiply' a Arvei. BLOCK_SIZE: 128. MATRIX_SIZE: 2048 (esq.) i 4096 (dta.) . . . 75
6.25 Speedup 'Matrix Multiply' a Arvei. BLOCK_SIZE: 256. MATRIX_SIZE: 2048 (esq.) i 4096 (dta.) . . . 76

6.26 Speedup 'Matrix Multiply' a Arvei. Comparativa mapeig DAST. `BLOCK_SIZE`: 64. `MATRIX_SIZE`: 2048 (esq.) i 4096 (dta.) 76
6.27 Speedup 'Matrix Multiply' a Knights. `BLOCK_SIZE`: 64. Workers per core: 1 (esq.) i 4 (dta.) 77
6.28 Speedup 'Matrix Multiply' a Knights. `BLOCK_SIZE`: 128. Workers per core: 1 (esq.) i 4 (dta.) 78
6.29 Speedup 'Matrix Multiply' a Knights. `BLOCK_SIZE`: 256. Workers per core: 1 (esq.) i 4 (dta.) 78
6.30 Speedup 'Sparse LU' a Arvei amb diferents paràmetres 79
6.31 Speedup 'Sparse LU' a Knights. `BSIZE`: 50. Workers per core: 1 (esq.) i 4 (dta.) 79
6.32 Speedup 'Sparse LU' a Knights. `BSIZE`: 75. Workers per core: 1 (esq.) i 4 (dta.) 80

Índex de taules

2.1 Costos directes del projecte 18
2.2 Costos indirectes del projecte 18
2.3 Costos d'amortització del projecte 19
2.4 Costos totals del projecte 20
2.5 Matriu de sostenibilitat del projecte 20

6.1 Temps d'execució de la funció *submitWithDependencies* 60
6.2 Temps d'execució de la funció *dependenciesDone* 60
6.3 Temps d'execució del DAST amb una i tres peticions 61
6.4 Temps de creació i inserció de les peticions dels 'workers' 62
6.5 Temps d'execució del codi de la petició DAST_WORK_NEW 62
6.6 Temps d'execució del codi de la petició DAST_WORK_DONE 62
6.7 Temps d'execució del codi de la petició DAST_WORK_DELETE 63

A.1 Execucions del Test 1 (`PARALLEL_CREATIONS`: 2, `TASK_SIZE`: 25000) 86
A.2 Execucions del Test 1 (`PARALLEL_CREATIONS`: 2, `TASK_SIZE`: 50000) 87
A.3 Execucions del Test 1 (`PARALLEL_CREATIONS`: 2, `TASK_SIZE`: 100000) 87
A.4 Execucions del Test 1 (`PARALLEL_CREATIONS`: 2, `TASK_SIZE`: 200000) 88
A.5 Execucions del Test 1 (`PARALLEL_CREATIONS`: 3, `TASK_SIZE`: 100000) 88
A.6 Execucions del Test 1 (`PARALLEL_CREATIONS`: 4, `TASK_SIZE`: 100000) 89
A.7 Execucions del Test 2 (`PARALLEL_CREATIONS`: 2, `TASK_SIZE`: 25000) 89
A.8 Execucions del Test 2 (`PARALLEL_CREATIONS`: 2, `TASK_SIZE`: 50000) 90
A.9 Execucions del Test 2 (`PARALLEL_CREATIONS`: 2, `TASK_SIZE`: 100000) 90
A.10 Execucions del Test 2 (`PARALLEL_CREATIONS`: 2, `TASK_SIZE`: 200000) 91
A.11 Execucions del Test 2 (`PARALLEL_CREATIONS`: 3, `TASK_SIZE`: 100000) 91
A.12 Execucions del Test 2 (`PARALLEL_CREATIONS`: 4, `TASK_SIZE`: 100000) 92
A.13 Execucions del Test 2 (`PARALLEL_CREATIONS`: 3, `TASK_SIZE`: 25000) 92
A.14 Execucions del Test 2 (`PARALLEL_CREATIONS`: 4, `TASK_SIZE`: 25000) 93
A.15 Execucions del Test 2 (`PARALLEL_CREATIONS`: 5, `TASK_SIZE`: 25000) 93
A.16 Execucions del programa Cholesky (`MATRIX_SIZE`: 2048, `BLOCK_SIZE`: 32) 94

A.17 Execucions del programa Cholesky (MATRIX_SIZE: 2048, BLOCK_SIZE: 64) . . . 94
A.18 Execucions del programa Cholesky (MATRIX_SIZE: 2048, BLOCK_SIZE: 128) . . . 95
A.19 Execucions del programa Cholesky (MATRIX_SIZE: 2048, BLOCK_SIZE: 256) . . . 95
A.20 Execucions del programa Cholesky (MATRIX_SIZE: 2048, BLOCK_SIZE: 512) . . . 96
A.21 Execucions del programa Cholesky (MATRIX_SIZE: 4096, BLOCK_SIZE: 128) . . . 96
A.22 Execucions del programa Cholesky (MATRIX_SIZE: 8192, BLOCK_SIZE: 128) . . . 97
A.23 Execucions del programa Cholesky (MATRIX_SIZE: 4096, BLOCK_SIZE: 256) . . . 97
A.24 Execucions del programa Cholesky (MATRIX_SIZE: 8192, BLOCK_SIZE: 256) . . . 98
A.25 Execucions del programa Matrix Multiply (MATRIX_SIZE: 2048, BLOCK_SIZE: 64) 98
A.26 Execucions del programa Matrix Multiply (MATRIX_SIZE: 4096, BLOCK_SIZE: 64) 99
A.27 Execucions del programa Matrix Multiply (MATRIX_SIZE: 2048, BLOCK_SIZE: 128) 99
A.28 Execucions del programa Matrix Multiply (MATRIX_SIZE: 4096, BLOCK_SIZE: 128) 100
A.29 Execucions del programa Matrix Multiply (MATRIX_SIZE: 2048, BLOCK_SIZE: 256) 100
A.30 Execucions del programa Matrix Multiply (MATRIX_SIZE: 4096, BLOCK_SIZE: 256) 101
A.31 Execucions del programa Matrix Multiply canviant el 'mapeig' del DAST (MATRIX_SIZE: 2048, BLOCK_SIZE: 64) . . . 101
A.32 Execucions del programa Matrix Multiply canviant el 'mapeig' del DAST (MATRIX_SIZE: 4096, BLOCK_SIZE: 64) . . . 102
A.33 Execucions del programa Sparse LU (NB: 16, BSIZE: 50) . . . 102
A.34 Execucions del programa Sparse LU (NB: 32, BSIZE: 50) . . . 103
A.35 Execucions del programa Sparse LU (NB: 64, BSIZE: 50) . . . 103
A.36 Execucions del programa Sparse LU (NB: 16, BSIZE: 75) . . . 104
A.37 Execucions del programa Sparse LU (NB: 32, BSIZE: 75) . . . 104
A.38 Execucions del programa Sparse LU (NB: 64, BSIZE: 75) . . . 105

B.1 Execucions del Test 1 (PARALLEL_CREATIONS: 1, TASK_SIZE: 25000) . . . 107
B.2 Execucions del Test 1 (PARALLEL_CREATIONS: 2, TASK_SIZE: 25000) . . . 108
B.3 Execucions del Test 1 (PARALLEL_CREATIONS: 1, TASK_SIZE: 100000) . . . 109
B.4 Execucions del Test 1 (PARALLEL_CREATIONS: 2, TASK_SIZE: 100000) . . . 110
B.5 Execucions del Test 1 (PARALLEL_CREATIONS: 3, TASK_SIZE: 100000) . . . 111
B.6 Execucions del Test 2 (PARALLEL_CREATIONS: 1, TASK_SIZE: 25000) . . . 112
B.7 Execucions del Test 2 (PARALLEL_CREATIONS: 2, TASK_SIZE: 25000) . . . 113
B.8 Execucions del Test 2 (PARALLEL_CREATIONS: 1, TASK_SIZE: 100000) . . . 114
B.9 Execucions del Test 2 (PARALLEL_CREATIONS: 2, TASK_SIZE: 100000) . . . 115
B.10 Execucions del programa Cholesky (MATRIX_SIZE: 2048, BLOCK_SIZE: 128) . . . 116
B.11 Execucions del programa Cholesky (MATRIX_SIZE: 2048, BLOCK_SIZE: 256) . . . 117
B.12 Execucions del programa Cholesky (MATRIX_SIZE: 2048, BLOCK_SIZE: 512) . . . 118
B.13 Execucions del programa Cholesky (MATRIX_SIZE: 4096, BLOCK_SIZE: 128) . . . 119

B.14 Execucions del programa Cholesky (MATRIX_SIZE: 4096, BLOCK_SIZE: 256) . . . 120
B.15 Execucions del programa Cholesky (MATRIX_SIZE: 4096, BLOCK_SIZE: 512) . . . 121
B.16 Execucions del programa Cholesky (MATRIX_SIZE: 8192, BLOCK_SIZE: 128) . . . 122
B.17 Execucions del programa Cholesky (MATRIX_SIZE: 8192, BLOCK_SIZE: 256) . . . 123
B.18 Execucions del programa Cholesky (MATRIX_SIZE: 8192, BLOCK_SIZE: 512) . . . 124
B.19 Execucions del programa Matrix Multiply (MATRIX_SIZE: 2048, BLOCK_SIZE: 64) 125
B.20 Execucions del programa Matrix Multiply (MATRIX_SIZE: 4096, BLOCK_SIZE: 64) 125
B.21 Execucions del programa Matrix Multiply (MATRIX_SIZE: 2048, BLOCK_SIZE: 128) 126
B.22 Execucions del programa Matrix Multiply (MATRIX_SIZE: 4096, BLOCK_SIZE: 128) 126
B.23 Execucions del programa Matrix Multiply (MATRIX_SIZE: 2048, BLOCK_SIZE: 256) 127
B.24 Execucions del programa Matrix Multiply (MATRIX_SIZE: 4096, BLOCK_SIZE: 256) 127
B.25 Execucions del programa Sparse LU (NB: 16, BSIZE: 50) 128
B.26 Execucions del programa Sparse LU (NB: 32, BSIZE: 50) 128
B.27 Execucions del programa Sparse LU (NB: 64, BSIZE: 50) 129
B.28 Execucions del programa Sparse LU (NB: 16, BSIZE: 75) 129
B.29 Execucions del programa Sparse LU (NB: 32, BSIZE: 75) 130
B.30 Execucions del programa Sparse LU (NB: 64, BSIZE: 75) 130

Capítol 1

Introducció

Aquest capítol explica el context en el qual es planteja aquest projecte, la motivació del projecte, l'objectiu i l'abast del mateix, a més de la metodologia seguida en el seu desenvolupament.

1.1 Motivació

Cada 2 anys, aproximadament, es duplica la densitat de transistors en els circuits integrats, com bé diu la Llei de Moore [4], i fins fa uns anys també es mantenia constant la densitat de potència a mesura que els transistors tornaven petis, permetent incrementar la freqüència de treball dels circuits, com sosté la Llei de Dennard [5]. En els últims anys s'han arribat a unes dimensions que incrementen en excés les fuites de corrent, trencant així aquest augment de freqüència que, en la majoria dels casos, implicava un augment de la capacitat de càlcul.

En canvi, gràcies a la reducció de la grandària dels transistors i la possibilitat de col·locar-ne més en la mateixa superfície, ha sorgit una nova tendència que consisteix a replicar recursos per seguir augmentant la capacitat de càlcul dels processadors. Aquest canvi ha fet que en els últims anys la programació paral·lela hagi adquirit major importància en l'entorn de la computació d'altes prestacions. Per altra banda, aquest paral·lelisme implica que les aplicacions s'han d'adaptar per poder esprémer al màxim la capacitat dels nous processadors.

Treballar directament amb aquests sistemes a baix nivell és complex i requereix coneixements específics de l'arquitectura sobre la qual s'està corrent les aplicacions. Per tal de simplificar aquesta tasca apareixen llibreries com PThreads (POSIX Threads) que abstreuen d'algunes gestions, tot i que segueixen requerint tractar amb entitats com 'threads', mecanismes de sincronització, etc. Per tal de fer-ho més simple, apareixen els models de programació paral·lels (OpenMP [Open Multi-Processing], MPI, OmpSs, etc.). Aquests abstreuen tota aquesta gestió i es limiten a requerir unes indicacions mínimes de l'usuari a alt nivell (directives), evitant així que s'hagin d'implementar funcions per la creació de 'threads' i la seva sincronització. L'objectiu dels models és obtenir el mateix resultat que en una versió seqüencial amb un temps menor [6].

Normalment aquests models són per entorns de memòria compartida, en els quals tots els

elements processadors tenen el mateix espai d'adreçament. Els entorns més comuns de memòria compartida són 'Symmetric Multiprocessors' (SMP). Tot i així alguns models de programació (OmpSs, UPC, X10...) relaxen aquest requeriment i donen una visió de memòria global entre els diferents espais d'adreçament [7].

El model de programació involucrat en aquest projecte és OmpSs, desenvolupat pel Barcelona Supercomputing Center - Centro Nacional de Supercomputación (BSC - CNS). A la plana web del projecte el defineixen de la següent forma:

> *OmpSs is an effort to integrate features from the StarSs programming model developed by BSC into a single programming model. In particular, our objective is to extend OpenMP with new directives to support asynchronous parallelism and heterogeneity (devices like GPUs) [8].*

Una de les directives que estén les possibilitats de OpenMP, i que és molt potent, és la capacitat d'expressar dependències entre les tasques creades i resoldre-les en temps d'execució. Aquestes dependències es poden expressar amb molta precisió, podent definir-les sobre variables i/o regions de memòria, segons les necessitats [9].

Per administrar i satisfer aquestes dependències, el sistema crea i gestiona un graf amb elles en temps d'execució. Per tant, les tasques no tenen per què executar-se immediatament després de la seva creació. La definició correcta del tipus i de la llista de dependències de cada tasca és responsabilitat del programador, ja que OmpSs no comprova aquesta informació, únicament la interpreta [1].

Aquest model de programació segueix la filosofia 'thread-pool'. En aquesta els diversos 'threads' poden crear tasques a executar de forma asíncrona, les quals es gestionen i s'executen pels mateixos quan acaben d'executar-ne una o aprofitant bloquejos deguts a la sincronització entre ells. A diferència d'altres que segueix la filosofia 'fork-join', que consisteix en l'alternança de zones d'execució seqüencial i zones d'execució paral·lela, en les que els diversos 'threads' executen codi en paral·lel i després se sincronitzen per sortir-ne [1].

OmpSs suporta un entorn d'execució heterogeni, és a dir, una combinació de diverses unitats de processament especifiques amb espais de memòria disjunts. En aquest entorn apareix la figura de 'master' que és l'encarregat d'administrar i assignar treball a la resta i és aquest qui conté la major part d'estructures.

Aquest tipus de models de programació tenen dos components claus pel seu correcte funcionament: el compilador i en 'runtime' (llibreries que implementen les funcionalitats i són utilitzades durant l'execució del programa paral·lel).

Compilador del model

La funció del compilador és convertir els fitxers amb totes les indicacions donades pel programador en un altre fitxer paral·lelitzat (de codi o executable) amb les crides corresponents

a la llibreria. Aquest tipus de compilador pot seguir dos models: 'source-to-source' o integral, cadascun dels quals té punts forts i debilitats [10].

OmpSs té el seu propi compilador que s'anomena Mercurium. Aquest és un compilador 'source-to-source' [11], és a dir, genera un nou fitxer en el mateix llenguatge que l'original però amb les directives reemplaçades per estructures de dades i crides a funcions de la llibreria. Aquest nou codi s'acaba compilant amb un compilador estàndard del llenguatge utilitzat per generar el fitxer executable.

Llibreria d'execució del model: 'Runtime'

La funció del 'runtime' és implementar la majoria de funcionalitats suportades pel model de programació: administrar els 'threads' (crear-los, destruir-los, identificar-los, aturar-los, despertar-los), gestionar el treball a efectuar, proporcionar una implementació per sincronitzar els diferents actors, proporcionar una implementació per les funcions intrínseques disponibles, gestionar les variables i estructures internes de control, entre d'altres [10].

La llibreria que implementa aquest sistema i suporta OmpSs s'anomena Nanos++ (o Nanos), la qual també proporciona mòduls específics per suportar OpenMP i Chapel [12]. De forma implícita en utilitzar-la s'estan creant un conjunt de 'threads' i tot un conjunt d'estructures per tal d'anar guardant les tasques que aquests creen i obtenir les que s'han d'executar, a més d'altres estructures amb finalitats diverses.

Aquestes estructures pertanyents al 'runtime' són compartides entre tots els 'threads' creats per l'execució de l'aplicació. Aquest fet implica que, en crear una tasca amb dependències i voler-la inserir o en acabar i voler-la eliminar del graf existent en les estructures del 'runtime', s'ha d'anar amb compte perquè poden existir més fils d'execució fent-ho deixant inconsistent la informació. Aquesta inconsistència pot crear 'deadlocks' o fer que no es respecti l'ordre d'execució seqüencial de les tasques. Les principals finalitats del graf són controlar les dependències i fer que quan una tasca té les seves satisfetes s'executi. La solució utilitzada a Nanos++ per garantir-ne la consistència és crear zones d'exclusió en la llibreria, forçant que si ja hi ha algú treballant-hi, la resta s'hagi d'esperar.

1.1.1 Problemàtica

Utilitzant OmpSs com a model de programació en sistemes SMP per executar alguns programes s'observa un estancament, o un increment, en el temps d'execució en augmentar el nombre de 'threads' utilitzats. El comportament esperat és reduir-lo de forma proporcional sempre que el programa tingui suficient paral·lelisme. Una de les causes d'aquest comportament pot ser l'increment de la contenció en l'accés al graf de dependències entre tasques existent al 'runtime', retardant l'execució de treball definit per l'usuari i limitant el paral·lelisme dels programes. Aquesta contenció ve donada per la solució utilitzada per garantir la consistència del graf. Per tant, un plantejament diferent podria reduir la contenció i alliberar part del paral·lelisme. Un possible plantejament és permetre únicament a un 'thread' accedir a aquestes estructures, eliminant així

la contenció per accedir-hi, centralitzant la gestió de les tasques amb dependències.

1.2 Actors implicats

Els diferents actors relacionats amb aquest projecte sigui de forma directa o indirecta són: el desenvolupador, el director i el codirector, els potencials usuaris i els potencials beneficiaris. Els quals s'expliquen amb major detall en els següents subapartats.

Desenvolupador

El desenvolupador del projecte és en Jaume Bosch Pons. És l'actor principal del mateix, el que hi té més responsabilitats i el més interessat en què tiri endavant, ja que és el seu Treball Final de Grau (TFG). Les seves funcions han estat realitzar les tasques planificades dins del temps previst, aquestes han estat des d'implementar funcionalitats, fins a redactar la memòria i preparar les presentacions necessàries, així com la seva exposició.

Director i codirector

El director i el codirector són en Carlos Álvarez Martínez i Daniel Jiménez González, respectivament. El seu rol en el projecte ha estat el d'assessorar, guiar i supervisar el desenvolupador en el transcurs del projecte amb l'objectiu d'evitar desviacions respecte al planificat i en cas d'aparèixer dificultats aconseguir solucionar-les de la forma menys costosa possible.

Potencials usuaris

Els potencials usuaris del nou model creat també són actors implicats. Gràcies a ells, el treball realitzat té més sentit i una finalitat que és ser-los útil. Aquests usuaris depenien dels resultats obtinguts, ja que si l'escenari resultant era d'empitjorament de la velocitat d'execució no té sentit que algú ho utilitzi, més enllà de possibles proves de concepte derivades o per reutilitzar-ne alguna part. En el cas positiu, els potencials usuaris són tots els usuaris del 'runtime' perquè poden veure reduït el temps d'execució de les seves aplicacions.

Potencials beneficiaris

Els potencials beneficiaris del model són també com a mínim els usuaris, ja que obtenen un rendiment del mateix. Però també es poden considerar, per transitivitat, tots els possibles beneficiaris dels projectes que efectuïn els usuaris amb el 'runtime'. Aquest és un col·lectiu immens perquè com s'ha explicat la programació paral·lela és de cada cop un fet més comú

i gairebé qualsevol aplicació que vulgui ser potent està paral·lelitzada i podria utilitzar aquest sistema per executar-se.

1.3 Estat de l'art

Aquest és un projecte de recerca que pretén comprovar l'efectivitat d'un nou plantejament pel que fa a la gestió de les dependències entre tasques i de les mateixes en el 'runtime' oficial d'OmpSs, Nanos++ (Nanos). És per tant una prova de concepte que sorgeix per tal d'intentar solucionar la problemàtica descrita en la motivació del projecte. No es planteja com a conseqüència d'haver observat la proposta en altres models de programació o implementacions alternatives, situació on s'estaria introduint una idea ja existent en una altre lloc.

Un dels pilars de Nanos++ és ser una eina oberta i disponible per tothom per a fer proves en el sistema, per això el seu codi font és 'open source'. Aquest codi ha estat la base per a portar a terme el projecte. Tot i aquesta filosofia, fins on coneixen les persones implicades en aquest projecte, no existeix cap estudi públic anterior sobre la casuística que tracta aquest projecte. Tanmateix sí que n'existeixen sobre altres aspectes, alguns d'ells són:

- Planificació de tasques de forma eficient [13].
- Suport per a 'memoization' [14].
- Suport per a OpenCL [15].

Els estudis sobre aquest aspecte però en altres models de programació també són inexistents, fins on saben els implicats. Per tant aquest és un projecte pioner sobre un problema que no s'havia tractat prèviament i sobre el qual no existeixen solucions prèvies.

El que sí que existia, previ a l'inici del projecte, era OmpSs i la implementació més recent de Nanos++. Aquest codi ha estat el que s'ha aprofitat com a base de treball i per fer l'adaptació al nou plantejament. S'ha aïllat el codi que accedeix a les estructures amb les dependències entre tasques, tota la resta del 'runtime' no tenia sentit tornar-la a crear. Així doncs s'ha intentat aprofitar al màxim les funcionalitats disponibles de Nanos, modificant únicament les parts estrictament necessàries pel desenvolupament del projecte. El mateix ha succeït amb el compilador d'OmpSs, Mercurium, en el qual no s'ha plantejat cap canvi i per tant s'ha utilitzat el ja existent.

1.4 Marc legal

Els principals aspectes legals que impliquen aquest projecte són el llicenciament del codi desenvolupat i la propietat intel·lectual del mateix, així com la de tota la documentació generada. Aquest apartat descriu els principals punts d'aquests aspectes.

Llicenciament

El punt de partida del projecte ha estat Nanos++, el 'runtime' d'OmpSs, el qual està llicenciat sota GNU versió 2 i per tant el projecte ha de respectar les clàusules d'aquesta llicencia.

El principal punt que permet el desenvolupament del projecte és que qualsevol pot copiar, distribuir i/o modificat el 'software' llicenciat. També estableix que ho fa sense cap tipus de garantia sobre el funcionament del mateix i qualsevol distribució, sigui de la mateixa obra o d'un derivat, ha de fer-se sota unes condicions de reconeixement dels autors i possibilitant l'accés a tot el codi font.

Així doncs el llicenciament del projecte és el mateix (GNU versió 2) i per tant s'estan complint les condicions de qualsevol obra derivada del projecte original.

Propietat intel·lectual

Pel que fa a la propietat intel·lectual del projecte s'aplica la Normativa sobre els drets de Propietat Industrial i Intel·lectual a la UPC. La qual estableix un conjunt de condicions i restriccions sobre les obres dels estudiants dirigides o coordinades pel professorat de la UPC.

La normativa estableix que d'aquestes obres correspondrà a la UPC la titularitat dels drets d'explotació sobre les mateixes i que l'estudiant i els professors seran considerats coautors d'aquestes. També estableix la forma de distribució dels beneficis derivats de l'explotació de drets de la propietat industrial i/o intel·lectual de les invencions o creacions, a més de condicionar-ne la divulgació del desenvolupament i els resultats de la recerca [16].

1.5 Objectiu

L'objectiu principal del projecte és modificar el 'runtime' Nanos++ per tal d'avaluar el rendiment obtingut si es centralitza la creació i destrucció de les tasques amb dependències. Això vol dir, crear una versió que utilitzi un 'thread' extra, el qual s'encarregui de generar, administrar i destruir les tasques que els 'workers' ('threads' que executen el codi de l'usuari) vagin generant i executant.

1.6 Abast

L'abast del projecte és el disseny teòric d'aquest nou model i la implementació d'aquest en el codi font del 'runtime'. Aquesta implementació consisteix a crear un nou 'thread' i donar-li la funcionalitat desitjada, aconseguint així que els 'workers' deleguin la responsabilitat de gestionar les tasques que executen. D'aquesta manera s'aconseguiria reduir la pressió en l'accés a les citades estructures i, idealment, reduir també el temps d'execució dels programes. Una segona part del

projecte consisteix a optimitzar aquesta gestió, per exemple, eliminant les instruccions d'exclusió a les regions que amb aquest canvi únicament executa un 'thread'.

Els canvis plantejats han estat seguits d'un conjunt de proves i execucions de programes per tal de comprovar el correcte funcionament del sistema. Un cop realitzades aquestes comprovacions s'han executat els mateixos programes amb les dues versions de Nanos i comparat els diferents rendiments.

1.7 Obstacles

Durant l'evolució del projecte podien aparèixer diversos obstacles o problemes que complicant-ne l'avanç i l'assoliment dels objectius explicats prèviament. A priori, existeixen tot una sèrie de punts que poden resultar més conflictius, els quals estan explicats en els següents punts.

Els possibles obstacles que, a l'inici del projecte, es preveia podien aparèixer durant l'evolució del mateix dificultant-ne l'avanç i l'assoliment dels objectius fixats són els descrits en els següents punts.

1.7.1 Codi existent

El projecte començava amb un gran volum de codi ja existent, el qual ha estat el punt de partida del projecte.

Codi extern

La utilització d'un codi extern i per tant amb una estructura desconeguda fa més costos l'inici del projecte i pot provocar problemes futurs pel desconeixement d'alguns punts crítics del codi. Per tal de minimitzar aquest obstacle es va dedicar un temps a familiaritzar-se amb el codi.

Codi col·laboratiu

La utilització d'un codi escrit i mantingut per diverses persones fa que diferents parts d'aquest puguin tenir plantejaments diferents que dificultin la comprensió global del sistema. Com en el cas anterior, familiaritzar-se amb el codi i contactar amb els autors en cas de tenir dubtes van ser les millors solucions.

1.7.2 Introducció d'errors

Durant la implementació dels canvis es podien introduir nous errors en la llibreria que esdevinguessin un comportament erroni o simplement la no-operabilitat de la mateixa. La metodologia

escollida pel desenvolupament del projecte pretenia minimitzar aquest obstacle.

1.7.3 Correcció d'errors i/o problemes

Solucionar un problema en un programa paral·lel és més complicat que fer-ho en un de seqüencial. Tenir diversos fils d'execució pot suposar l'existència d'errors que esdevinguin sols alguns cops, segons com s'ordenin les instruccions en aquella execució, dificultant la localització del mateix. Per la localització dels problemes es va decidir utilitzar un 'debugger', eina que usualment simplifica la localització dels errors.

1.8 Metodologia i rigor

La metodologia i el rigor en el desenvolupament del projecte han estat uns aspectes importats per tal d'assolir els objectius del mateix amb uns resultats fiables i de qualitat. A continuació es descriuen breument els principals aspectes de la metodologia de treball aplicada en aquest projecte.

1.8.1 Mètode de treball

El sistema de treball per desenvolupar aquest projecte s'ha basat en l'ús de metodologies àgils, basades en 'sprints', retrospectives, revisions i planificacions a curt termini. Concretament la metodologia ha estat SCRUM, per tant s'han establert uns períodes de treball abans dels que es definia concretament les tasques o punts a tractar en els dies posteriors i al final dels quals es revisava els resultats obtinguts i també els problemes sorgits. El treball ha estat incremental, és a dir cada 'sprint' s'avançava en una part concreta dels objectius globals, definits en la reunió prèvia.

Aquest plantejament ha permès que si en algun moment sorgia un problema que dificultava l'avanç del projecte aquest es considerava en la planificació del pròxim període de treball. Aquesta planificació dinàmica ha intentat minimitzar l'impacte dels problemes en el desenvolupament global de projecte, donant una certa flexibilitat per evitar no assolir els objectius.

1.8.2 Eines de treball

Les eines de treball per realitzar el projecte s'han format en dos grups principals: les eines locals del portàtil del desenvolupador i les remotes dels servidors del DAC així com tot un conjunt de programes i llibreries comunes en les dues parts. Aquestes han estat tant 'hardware' com 'software'. Les primeres no han estat especifiques per aquest projecte, són eines disponibles abans d'aquest projecte. Les segones són totes gratuïtes, ja sigui perquè són 'open source' o bé perquè s'han utilitzat en la seva modalitat gratuïta. A l'apartat 2.2 estan llistats totes els

recursos necessaris per assolir l'objectiu del projecte i en el capítol 3 es descriuen els aspectes claus de l'eina bàsica del projecte, el codi font d'OmpSs.

1.8.3 Eines de seguiment

Les eines utilitzades per fer un seguiment rigorós del projecte han estat:

- Git. S'ha utilitzat aquest sistema de control de versions per tal d'anar guardant el treball fet i permetre tornar a una versió anterior si en algun moment era necessari. A més, aquest també permet tenir una visió global de l'evolució d'un projecte en el 'commit log'.
- Sprint meetings. Les reunions associades a cada 'sprint' són una forma més d'anar fent un seguiment d'un projecte.
- Correu electrònic. La comunicació entre els diferents membres involucrats en el projecte s'ha realitzat per aquest canal perquè permet una ràpida propagació de la informació cap a tothom.

1.8.4 Mètodes de validació

La validació dels resultats s'ha efectuat en dues línies principals:

- Intrínseca al mètode de treball. Una de les bases de la metodologia SCRUM és tenir al final de cada 'sprint' una versió estable per poder fer una demostració del treball fet. Així doncs, existeix una validació constant dels canvis que es van realitzant, intentant prevenir així un dels obstacles comentats que era la introducció d'errors en el 'runtime'.
- Període d'anàlisi. Com es comenta en l'abast del projecte, un cop els canvis realitzats siguin suficientment complets, s'ha procedit a l'execució d'uns programes de prova per comprovar el comportament i comparar el nou 'runtime' amb l'original. Això comporta una important validació del treball realitzat.

Capítol 2

Planificació del projecte

Aquest capítol analitza la planificació temporal (tasques previstes inicialment amb els seus recursos i les seves dependències, així com el pla d'acció front possibles contratemps), llista els recursos necessaris, descriu la planificació econòmica (pressupost del projecte i el pla de control de costos), s'analitza la sostenibilitat del projecte i finalment s'especifiquen els canvis en la planificació inicial.

2.1 Planificació temporal

La planificació especificada en aquest capítol és l'inicial i s'ha d'entendre com a tal i no com a una especificació rígida i inamovible.

2.1.1 Descripció de les tasques

En cadascun dels subapartats següents s'especifiquen les tasques previstes per a la realització del projecte (descripció, duració aproximada i recursos necessaris). La totalitat de les tasques han estat planificades en el període comprés entre el 16/02/2015 i el 21/06/2015, aproximadament, per tant s'han disposat d'unes 19 setmanes per fer-ho. La dedicació mitjana aproximada en una setmana de treball del desenvolupador s'ha considerat d'unes 25 hores i per cada dues setmanes s'ha de considerar una hora de reunió amb el director i codriector.

Fita inicial

Duració: 5 setmanes.

Aquesta tasca agrupa tot el treball i lliuraments relacionats amb l'assignatura GEP (Gestió de Projectes) que realitzen tots els estudiants de la Facultat d'Informàtica de Barcelona (FIB) al matricular el TFG. L'objectiu d'aquesta va ser, mitjançant diverses eines i metodologies, ajudar

a la planificació inicial del projecte. Va constar de set entregues que tracten diversos aspectes de la planificació i gestió, aquestes van ser:

- Abast del projecte.
- Planificació temporal.
- Gestió econòmica i sostenibilitat.
- Presentació preliminar.
- Contextualització i bibliografia.
- Mòdul especific Enginyeria de Computadors.
- Presentació oral i document final.

Els recursos humans involucrats en aquesta tasca han estat: el desenvolupador, el director i el codirector; aquests dos darrers únicament en dues reunions d'una hora. La resta de recursos necessaris per la realització de la tasca han estat: Atenea (plataforma d'aula virtual de la UPC), Racó de la FIB (sistema per l'administració del projecte), Sharelatex (editor en línia de LaTex), una càmera gravadora, Power Point Online (editor de presentacions en línia) i els recursos necessaris per accedir a totes aquestes eines com un ordinador i una connexió a Internet.

Creació de l'entorn de treball i familiarització

Duració: 2 setmanes.

Aquesta tasca va consistir en la creació de l'entorn de treball per poder desenvolupar el projecte en l'equip portàtil del desenvolupador i en la familiarització/estudi del codi font, aquest punt va ser molt important com es comenta en l'apartat de riscos perquè podia complicar l'avanç del projecte.

Per crear l'entorn de treball es va descarregar una versió estable de OmpSs (composta per Mercurium i Nanos++, compilador i 'runtime' respectivament), es van compilar els codis font i instal·lar els binaris. La compilació havia d'incorporar altres llibreries com Extrae per permetre l'execució de programes instrumentats posteriorment. Aquesta seqüència d'operacions era necessari que fos el més simple possible, ja que s'ha executat repetidament durant l'avanç del projecte.

Per familiaritzar-se amb l'estructura del codi i el funcionament del 'runtime', la part que s'ha modificat, es va compilar algun programa de prova amb l'opció pertinent perquè Mercurium no elimines els fitxers intermedis utilitzats en la creació del binari. Aquests són llegibles per un programador i permeten veure totes les crides que s'efectuen al sistema en l'execució del programa. Amb aquest punt de partida, l'objectiu principal va ser analitzar la creació dels 'threads', la creació de les tasques i la gestió de les mateixes, mirant el codi font que es va executant.

Els recursos humans involucrats en aquesta tasca han estat: el desenvolupador, el director i el codirector; aquests dos darrers únicament en una reunió d'una hora. La resta de recursos

necessaris per la realització de la tasca han estat: el codi font d'Ompss i de les llibreries necessàries per a la compilació del mateix, les llibreries i eines que permeten la compilació (make, automake, autotools, gcc, etc.), l'ordinador on s'ha instal·lat aquest sistema i s'ha desenvolupat el projecte, el repositori de Git on s'han anat enviant els canvis efectuats i l'editor de text per visualitzar el codi font (Atom).

Realització dels canvis

Duració: 6 setmanes.

Aquesta tasca va consistir a aplicar tots els canvis necessaris en el 'runtime' per aconseguir el comportament desitjat. Aquests comporten la creació d'un nou 'thread' en el sistema que executi el nou codi de gestió i la delegació de la responsabilitat de crear, administrar i destruir les tasques a executar de la resta de 'threads'. També s'ha de considerar el cost d'anar comprovant que els canvis efectuats no deixaven les llibreries en un estat inestable i l'anàlisi final per comprovar el correcte funcionament.

Els recursos humans involucrats en aquesta tasca han estat: el desenvolupador, el director i el codirector; aquests dos darrers únicament en tres reunions d'una hora. La resta de recursos necessaris per la realització de la tasca han estat: Atom, les llibreries i eines que permeten la compilació del codi font i programes de prova (make, automake, autotools, gcc, etc.), git i l'ordinador on executar-ho.

Optimització de la gestió

Duració: 2 setmanes.

Aquesta tasca va consistir en optimitzar la gestió de les tasques efectuada per nou 'thread'. Una d'aquestes optimitzacions ha estat localitzar i eliminar les instruccions que garanteixen l'exclusió en l'accés a les estructures on es guarda la informació de les tasques. Aquestes tenen un cost i al tenir únicament un 'thread' accedint-hi no són necessàries. També s'ha de considerar el cost d'anar comprovant que els canvis efectuats no deixaven les llibreries en un estat inestable i l'anàlisi final per comprovar el correcte funcionament.

Els recursos humans involucrats en aquesta tasca han estat: el desenvolupador, el director i el codirector; aquests dos darrers únicament en una reunió d'una hora. La resta de recursos necessaris per la realització de la tasca han estat: Atom, les llibreries i eines que permeten la compilació del codi font i programes de prova (make, automake, autotools, gcc, etc.), git i l'ordinador on executar-ho.

Anàlisi dels resultats

Duració: 1 setmanes.

Aquesta tasca va consistir a crear un entorn en el clúster Arvei del DAC per poder executar-hi programes de prova, tant amb la versió original del 'runtime' com amb la nova versió creada. Aquest entorn havia de permetre l'execució mitjançant les cues d'execució exclusiva en els nodes que disposa el clúster. Un cop creat aquest entorn s'havia de corre diversos programes de prova amb ambdues versions per obtenir els diferents temps d'execució i comparar-los. Aquesta tasca tenia com a objectiu també fer una verificació final del correcte funcionament del nou sistema.

Els recursos humans involucrats en aquesta tasca han estat: el desenvolupador, el director i el codirector; aquests dos darrers únicament en una reunió d'una hora. La resta de recursos necessaris per la realització de la tasca han estat: el codi font d'Ompss i de les llibreries necessàries per a la compilació del mateix, les llibreries i eines que permeten la compilació (make, automake, autotools, gcc, etc.), el repositori de Git amb el codi modificat, el clúster, l'ordinador del programador per accedir als servidors del DAC, així com el 'software' que ho permeti (OpenVPN i SSH, 'Secure Shell')

Etapa final

Duració: 3 setmanes.

Aquesta tasca o etapa ha estat destinada a la redacció/realització de la memòria, la presentació i, en general, la documentació associada al projecte.

Els recursos humans involucrats en aquesta tasca han estat: el desenvolupador, el director i el codirector; aquest dos darrers únicament en dues reunions d'una hora. La resta de recursos necessaris per la realització de la tasca han estat: Racó de la FIB, Sharelatex, Power Point Online, Plotly i els recursos necessaris per accedir a totes aquestes eines com un ordinador i una connexió a internet.

2.1.2 Dependències i seqüència de les tasques

Dependències de precedència

Les tasques especificades prèviament estan totes relacionades i algunes d'elles depenen dels resultats d'altres i per tant no podien realitzar-se en paral·lel i/o en odre invers. A continuació s'expliciten les dependències de cadascuna de les tasques i a la figura 2.1 apareix el diagrama de Gantt d'acord amb l'especificat.

- Fita inicial. Aquesta tasca no té cap dependència prèvia. Les seves 7 subtasques sí que en tenen, cadascuna d'elles depèn de l'anterior per ordre de llistat.
- Creació de l'entorn de treball i familiarització. Aquesta tasca no depèn de cap altra però a causa de la seva carrega es va planificar com a posterior a la 'Fita inicial'. Està composta per dues subtasques: 'creació de l'entorn' i 'familiarització', que són independents entre elles però es va planificar primer la primera per si sorgís algun inconvenient en la creació.

- Realització dels canvis. Aquesta tasca depèn de tenir un entorn de treball i per tant de la tasca 'Creació de l'entorn de treball i familiarització'. La subtasca d'anàlisi depèn que els canvis s'hagin implementat, per tant es va definir una primera subtasca d'implementació.
- Optimització de la gestió. Aquesta tasca depèn de 'Realització dels canvis' i té dues subtasques: 'implementació' i 'anàlisi' (la segona és posterior a la primera).
- Anàlisi dels resultats. Aquesta tasca té dues parts: 'creació de l'entorn' i 'execució'. La creació de l'entorn depèn de la tasca 'Creació de l'entorn de treball i familiarització' i l'execució dels programes de prova depèn de 'Optimització de la gestió', així com de la primera subtasca (s'havia de crear l'entorn d'execució).
- Etapa final. Aquesta tasca depèn de 'Anàlisi dels resultats'. Tot i que de forma parcial podia portar-se a terme en paral·lel a la resta de tasques.

Diagrama de Gantt

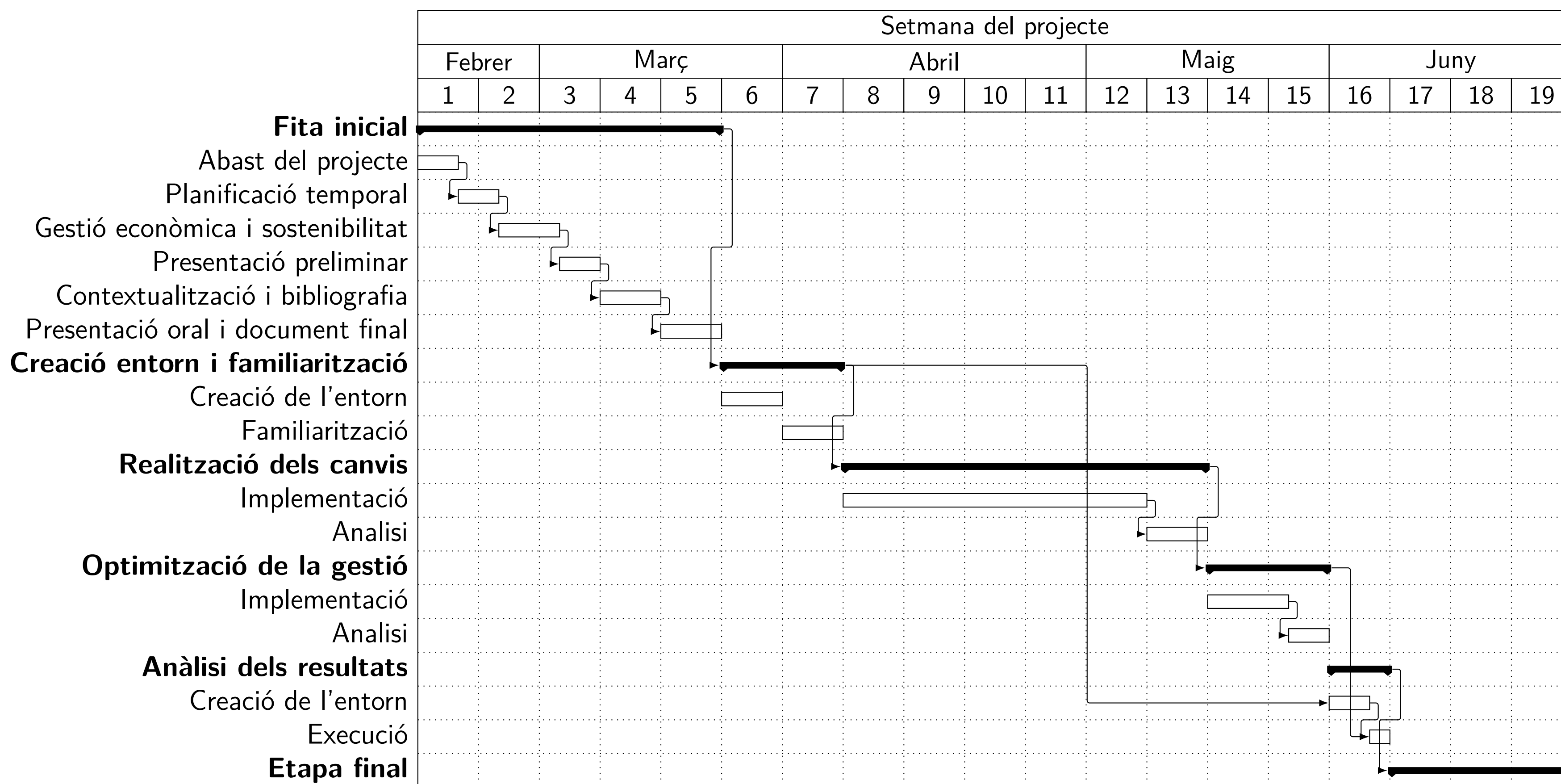


Figura 2.1: Diagrama de Gantt amb el seqüenciament de les tasques

2.1.3 Valoració d'alternatives i pla d'acció

Per garantir la finalització del projecte en el temps establert sense afectar ni a la duració total del mateix, ni als recursos utilitzats, es va plantejar el següent pla d'acció contra les desviacions. En general, les desviacions no haurien d'afectar a l'obtenció de resultats, per aconseguir-ho la metodologia de treball escollida ha estat dinàmica, permetent re-planificar part de les tasques i/o prioritzar parts del projecte. Si la desviació era positiva, s'acabava la tasca abans del previst, s'avançava l'inici de la següent, guanyant així un marge per a possibles futurs endarreriments. Si la desviació era negativa, s'allargava la tasca més del previst, una primera opció era endarrerí l'inici de la posterior, si no segons el problema es prenien unes decisions o unes altres, segons els següents punts:

- Problemes amb els sistemes/llibreries a utilitzar. El pla d'acció consisteix a demanar suport als creadors de les llibreries (OmpSs, Extrae, etc.) o al departament de sistemes del DAC (servidors, cues, etc.) i mentre s'espera resposta avançar alguna altra tasca que no depengui directament de la bloquejada. És a dir, si el problema sorgeix en la tasca 'Creació de l'entorn de treball i familiarització', s'avançarà la part de familiarització; i si és en la 'Anàlisi dels resultats', s'avançaria parcialment l'etapa final.
- Dificultats d'implementació. El pla d'acció dependrà de la previsió de retard. Si és mínima, simplement s'endarrerirà l'inici de la següent tasca. Si és considerable, s'haurà de replanificar el treball a fer. Podent arribar a descartar alguna tasca o subtasca com l'anàlisi de la 'Realització dels canvis' o 'Optimització de la gestió', completament. La millor opció es valoraria i consensuaria en un dels 'sprint meetings'.

2.2 Llistat de recursos

A continuació s'agrupen els diferents recursos utilizats en les tasques planificades prèviament segons la seva naturalesa: recursos humans, 'hardware' o 'software'.

Humans

- Desenvolupador.
- Director.
- Codirector.

Hardware

- Càmera gravadora. Nikon D5100.

- Clúster Arvei del DAC. Nodes amb processador Intel Xeon E5-2630L Dual, 64GB de memòria RAM, 1TB de disc dur [17].
- Orinador portàtil. Model: ASUS U36SD. Components principals: processador i5-2410M, 4GB de memòria RAM, 640GB de disc dur, targeta gràfica NVIDIA GT520M [18].

Software

- Atenea. Plataforma d'aula virtual de la UPC.
- Atom. Editor de text 'open source'. Versió 0.125.
- Autotools. Conjunt d'eines per a la creació d'executables en diferents distribucions UNIX. Versió autoreconf 2.69. Versió automake 1.14.1.
- Codi font OmpSs. 'Release' del codi font de Nanos++ i Mercurium. Versió 14.09 (Versió Mercurium 1.99.4, versió Nanos 0.7.2).
- Extrae. Llibreria per instrumentar codi i generar traces [19]. Versió 3.0.3.
- GCC. Compilador de C i C++ lliure. Versió 4.8.2.
- GDB. 'Debugger' lliure. Versió 7.7.1.
- Git. Eina de control de versions del codi font del projecte. Versió client 1.9.1.
- Linux Mint. Sistema operatiu de l'equip del desenvolupador. Versió 17 amb Cinnamon 2.2.16 64-bit.
- Make. Eina per la generació automatitzada de fitxers executables. Versió 3.81.
- Opera. Navegador web basat en Chromium. Versió 28.
- Paraver. Eina per a la visualització gràfica de traces [20]. Versió 4.5.4.
- Plotly. Plataforma on-line per crear gràfics.
- Power Point Online. Editor de presentacions en línia.
- Racó FIB. Sistema per l'administració institucional del projecte.
- Sharelatex. Editor en línia de LaTex.
- SSH i OpenVPN. Eines per l'accés remot als serveis del DAC.
- Ubuntu. Sistema operatiu d'Arvei. Versió 14.04.2 LTS.
- Vim. Editor de text per terminals. Versió 7.4.52.

2.3 Planificació econòmica

Aquest secció analitza la gestió econòmica del projecte. S'hi detallen els diferents costos i s'analitzen les possibles desviacions del pressupost inicial per la realització del projecte.

2.3.1 Pressupost

Donats tots els recursos necessaris pel correcte desenvolupament del projecte, a continuació s'estimen els costos associats a aquests considerant el temps d'utilizació prèviament especificat per cadascun d'ells.

Identificació i estimació de costos

Cadascuna de les tasques planificades té diversos costos directes, costos indirectes i amortitzacions; derivats de la utilització dels diferents recursos humans, 'hardware' i 'software' per la seva realització. Els únics costos directes atribuïbles a l'execució de les tasques del projecte han estat els dels recursos humans, els quals estan comptabilitats a la taula 2.1.

Descripció	Unitats	Preu unitari	Cost
Director	10 hores	30 €/hora	300'00 €
Codirector	10 hores	30 €/hora	300'00 €
Desenvolupador	475 hores	20 €/hora	9500'00 €
Total			10.100'00 €

Taula 2.1: Costos directes del projecte

Respecte als costos indirectes d'aquest projecte, aquests han estat tals com: aigua, llum, climatització, gas, paper, etc. Aquest conjunt de costos està estipulat que suposa al voltant del 25% dels costos directes atribuïbles al projecte segons la mètrica utilitzada en el DAC, la qual sorgeix d'altes projectes de referència on s'ha calculat. A la taula 2.2 apareix la quantia total d'aquest concepte.

Descripció	Cost
Costos directes	10.100'00 €
Costos indirectes (25%)	2.525'00 €

Taula 2.2: Costos indirectes del projecte

Tots els recursos 'hardware' utilitzats per portar a terme les tasques especificades no han estat exclusius per aquestes i per tant únicament es comptabilitza l'amortització corresponent a

les hores d'ús per fer-ho. Amb els recursos 'software' hi ha el mateix plantejament, a més, tot és 'open source' o 'freemium' i per tant el seu cost d'amortització és zero com es veu a la taula 2.3.

Descripció	Cost base	Vida útil	Utilizació	Cost
Ordinador	880 €	5 anys	475 hores	145'20 €
Arvei	4500 €	3 anys	30 hores	5'14 €
Camara	550 €	5 anys	3 hores	0'87 €
Atom	0 €			0'00 €
Vim	0 €			0'00 €
Opera	0 €			0'00 €
Linux Mint	0 €			0'00 €
Atenea	0 €			0'00 €
Racó FIB	0 €			0'00 €
Sharelatex	0 €			0'00 €
Power Point Online	0 €			0'00 €
Make	0 €			0'00 €
GCC	0 €			0'00 €
Autotools	0 €			0'00 €
PAPI	0 €			0'00 €
Extrae	0 €			0'00 €
Paraver	0 €			0'00 €
GDB	0 €			0'00 €
SSH	0 €			0'00 €
OpenVPN	0 €			0'00 €
Git	0 €			0'00 €
OmpSs	0 €			0'00 €
Total				151'21 €

Taula 2.3: Costos d'amortització del projecte

Cost total

A la taula 2.4 es dóna el pressupost total del projecte. S'ha considerat un marge del 10% del cost total brut per a possibles imprevistos sorgits en les diverses tasques i per contingències. Aquest es considerava suficient perquè el treball a fer estava suficientment especificat i s'havien valorat alternatives per a superar les possibles desviacions en la planificació efectuada. Posteriorment s'aplica el respectiu impost general sobre el consum del 21%.

Descripció	Cost
Base	12.776'21 €
Contingencies i imprevistos	1.277'62 €
Subtotal	14.053'83 €
IVA	2.951'30 €
Total	17.005'14 €

Taula 2.4: Costos totals del projecte

2.3.2 Control de costos

El control dels costos s'ha basat principalment en l'aplicació de totes les metodologies descrites en els apartats anteriors i l'aplicació del pla d'acció per evitar desviacions en la planificació. En cas de no ser suficient el procediment a seguir consistia a: registrar la desviació respecte del pressupostat, aprofitar la reunió de final de 'sprint' pertinent per analitzar el succeït i intentar trobar les possibles causes i, finalment, aplicar els canvis oportuns per tal d'evitar futures desviacions.

Aquest sistema de control no pretenia ser necessari perquè, com s'ha comentat, les tasques a efectuar estaven molt especificades i en el pressupost s'havia reservat una quantitat en concepte d'imprevistos, la qual hauria de ser suficient per a sufragar els costos derivats dels possibles problemes.

2.4 Sostenibilitat

Aquest apartat avalua la sostenibilitat del projecte des de tres punts de vista: econòmic, social i ambiental. En la taula 2.5 apareix la matriu de sostenibilitat amb les puntuacions obtingudes en cada aspecte i en els subapartats posteriors es justifica el perquè d'aquestes puntuacions.

	Econòmica	Social	Ambiental
Planificació	9	8	7
Resultats	10	10	10
Riscs	0	0	0
Valoració Total	56		

Taula 2.5: Matriu de sostenibilitat del projecte

Dimensió econòmica

El projecte consta d'una avaluació de costos materials i humans, en la que s'han considerat les desviacions i per tant els petits ajustos respecte els costos pressupostats. Pel tipus de projecte no

s'ha considerat el cost d'actualitzacions i/o reparacions, el resultat del projecte pretén ser definitiu i funcional. A més el pressupost és realment detallat i contingut, per tant econòmicament parlant és competitiu. L'única forma de reduir costos seria que els recursos humans utilitzats fossin ja experts en el 'runtime' a optimitzar, així s'aconseguiria reduir o eliminar la duració d'algunes de les tasques. Tot i que tenir una mà d'obra més especialitzada, probablement tindria un cost per hora major, minvant així la millora pressupostaria. No es poden minvar costos utilitzant tecnologies ja existents o reduint la duració de les tasques, la qual és proporcional a la seva importància dins del projecte. El motiu, el projecte tracta un tema nou i desconegut. El que sí que s'aprofita és tot el codi del 'runtime' que no es modifica així com el compilador. Aquest projecte projecte està relacionat amb un altre projecte acadèmic d'investigació que pretén crear una versió 'hardware' de Nanos++. El plantejament d'aquest projecte és una part imprescindible per la futura implementació del model aquí proposat en el 'runtime' 'hardware', el qual cerca una reducció en el sobre-cost que suposen els models del programació paral·lela.

Dimensió social

La situació social del lloc on es desenvolupa el projecte és de benestar amb una sistema polític democràtic. La situació del sector acadèmic on es desenvolupa el projecte és complexa econòmicament parlant, a causa de les retallades pressupostàries, però aquest fet no implica una inestabilitat o problemàtica addicional en l'entorn del projecte. Aquest projecte no té perquè contribuir a la millora d'aquestes situacions, però el que no farà serà empitjorar-la; no es tracten temes controvertits ni conflictius. Aquest projecte no és imprescindible, ja que en realitat el sistema existent ja funciona, però una millora en aquest sempre és bona perquè això permet reduir el temps d'execució de les aplicacions i en el temps restant poden executar-se'n d'altres que abans no podien o millorar la precisió de l'actual. Aquesta millora no suposa un benefici directe pels usuaris, però segons l'ús que se'n faci sí que pot suposar una millora per a terceres persones (per exemple, malats). El resultat del projecte no pretén canviar la vida dels usuaris del mateix, de fet el resultat hauria de ser transparent per ells. En cap cas aquest treball suposa un perjudici per algú, ja que simplement s'investiga sobre una nova metodologia de treballar en sistemes multiprocessadors.

Dimensió ambiental

El recurs utilitzat al llarg de tot el projecte que afecta el medi ambient és l'electricitat. La resta de recursos no són adquirits específicament pel projecte i per tant no considerem el seu impacte ambiental de fabricació, reciclatge, etc. únicament el seu consum energètic. Aquest consum ha estat estimat en el pressupost juntament amb una part de consum elèctric per la il·luminació i altres costos indirectes. En cas d'efectuar-se el projecte sense ser un treball final de grau, el consum elèctric seria el mateix, l'únic que deixaria d'utilitzar-se serien algunes plataformes en línia (Atenea, Raco FIB...) el consum de les quals podem considerar independent de la realització d'aquest treball. D'altres projectes es poden reaprofitar els recursos 'software' com OmpSs, Sharelatex, etc.

Aquest projecte no crea un producte tangible i per tant no té un procés de fabricació associat que consumeixi energia, ni tampoc s'ha creat una infraestructura que s'hagi de desmantellar en acabar. Com que no és un producte físic no es requereixen matèries primeres, tampoc fonts d'extracció de les mateixes de forma directa. La contaminació que es pot generar és la derivada a la fabricació de l'electricitat consumida la qual depèn de si les fonts d'energia del Campus Nord són renovables o no. El material manufacturat necessari pel projecte no és específic del mateix, com s'ha comentat, sinó que són recursos ja disponibles.

Si el projecte suposa una millora en el temps d'execució dels programes com es planteja, això ajudaria a reduir la petjada ecologia de l'execució dels programes en utilitzar menys temps i per tant menys energia elèctrica. En cap cas implica un augment de la petjada ecològica, ja que si no s'obtenen resultats, es descarta el model i es continua amb el ja existent. Com s'ha comentat, part d'aquest projecte (codi i model) pot aprofitar-se per ser implementat en un nou Nanos++ 'hardware'.

2.5 Revisió de la planificació

Aquest apartat descriu els diferents problemes que han aparegut en l'avanç del projecte, els canvis que han implicat aquests contratemps en la planificació i les implicacions que han tingut per tal d'aconseguir que el projecte s'acabi en el temps establert.

2.5.1 Canvis respecte a la planificació inicial

En el desenvolupament del projecte han sorgit diversos problemes que han allargat la duració de les tasques 'Realització dels canvis' i la posterior 'Optimització de la gestió' bloquejant-ne l'avanç. Alguns dels problemes sorgits han estat: problemes en la instrumentació del 'runtime', dificultats per instal·lar/compilar els programes de prova, creació de diverses condicions de carrera en l'execució dels programes difícils de solucionar. Per resoldre alguns problemes s'ha requerit suport dels desenvolupadors o de les persones encarregades de mantenir els codis utilitzats, d'acord amb el pla d'acció dissenyat per tal efecte.

El pla d'acció contemplava dues possibilitats per afrontar aquests problemes: retardar l'inici de les tasques posteriors o dedicar el temps de bloqueig a tasques o subtasques posteriors que depenguessin del treball ja realitzat. Per exemple, s'ha avançat la creació de l'entorn per a l'anàlisi de resultats i l'escriptura de la memòria del projecte de forma parcial en els moments de bloqueig de les tasques prèvies. Aquests canvis de treball s'han anat intercalant segons les necessitats en cada moment i d'acord amb el criteri del desenvolupador i les recomanacions del director i codirector.

Un altre canvi que s'ha introduït ha estat la realització de proves de rendiment en un segon entorn. Aquest entorn té molt més nuclis de càlcul disponibles i es va considerar que podia aportar informació extra valuosa pel projecte.

2.5.2 Implicacions

Les implicacions en la planificació d'aquests canvis han estat una reorganització parcial de l'odre de les tasques i la realització de diverses d'elles de forma concurrent com apareix en el diagrama de Gantt revisat (figura 2.2). Amb aquests canvis s'ha garantit l'assoliment dels objectius del projecte sense implicar més recursos humans.

En el cas de recursos materials, sí que s'ha introduït un nou equip el qual té una amortització que en pressupost original no s'havia comptabilitzat. Tot i això, aquest tipus d'increments sí que s'havien contemplat en el pressupost en l'apartat de contingències i imprevistos, essent el cost d'aquesta nova amortització (al voltant d'uns 8 €) molt inferior a la suma estimada per aquest concepte. A continuació es detallen els diferents recursos addicionals no llistats en l'apartat 2.2.

Nous recursos 'hardware'

- Coprocessador MIC del node Knights4 del BSC. Model C0PRQ-7120.

Nous recursos 'software'

- Intel C/C++ compiler. Compilador de C/C++ propietari d'Intel (necessari per compilar a l'arquitectura MIC). Versió 15.0.2.
- SUSE Linux. Sistema operatiu de Knights4. Versió Enterprise Server 11.
- Taskset. Eina definir una mascara de processadors a utilitzar per un proces [21].

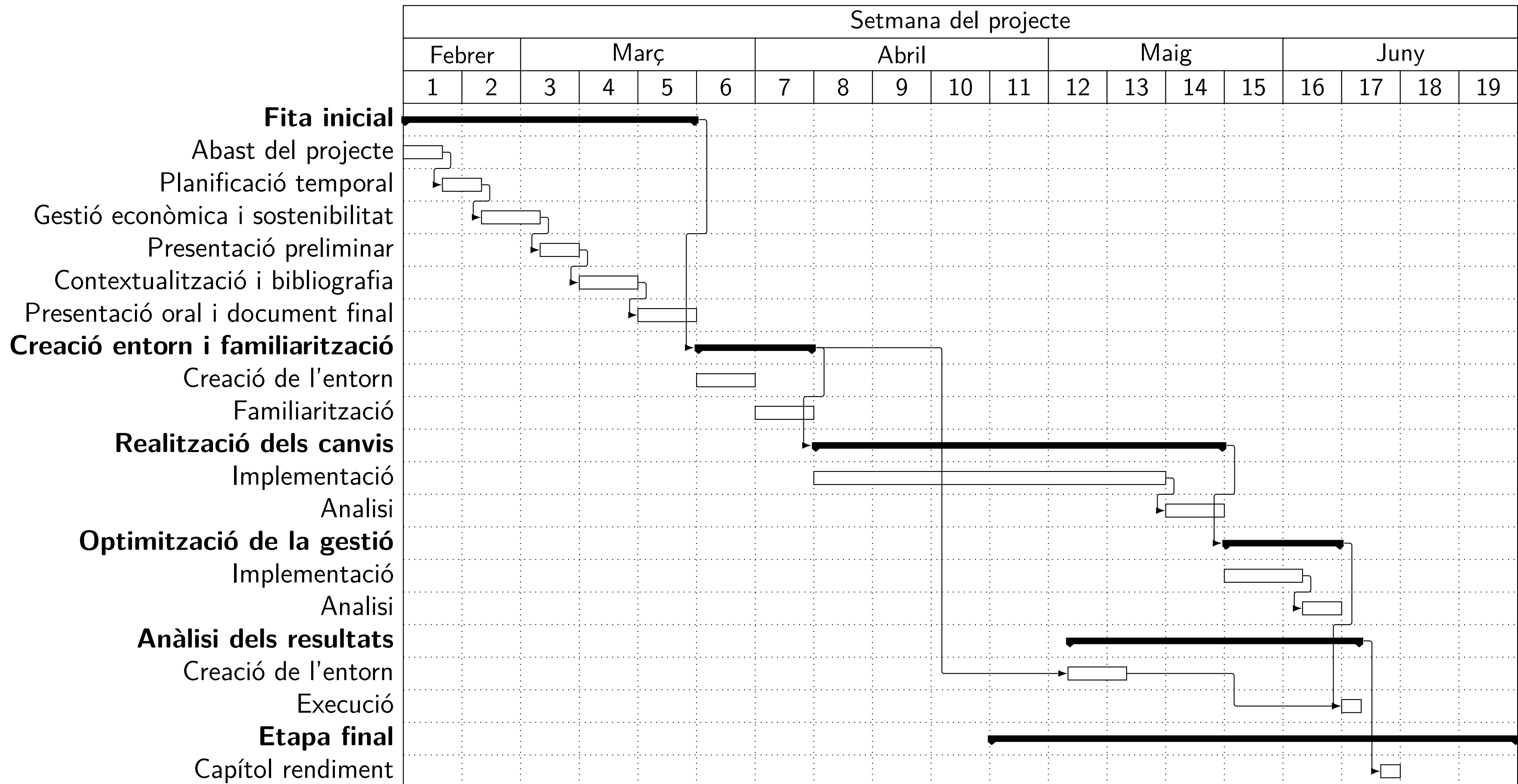


Figura 2.2: Diagrama de Gantt amb el seqüenciament final de les tasques

Capítol 3

Model de programació Base: OmpSs

Aquest capítol explica el model de programació paral·lela OmpSs que és la principal eina per desenvolupar el projecte. En els següents punts es descriu el seu model d'execució, la clàusula 'task', el sistema de dependències, el compilador i els aspectes implicats del 'runtime' en el projecte.

3.1 Característiques generals

L'execució de les aplicacions sempre comença amb la creació d'un equip de 'threads' compost per l'inicial (màster), creat de forma implícita pel sistema operatiu en començar l'execució, i diversos més creats posteriorment ('workers'). El nombre total el defineix l'usuari mitjançant una variable d'entorn o fent una crida a l'API ('Application Programming Interface') per a tal fi. Si no s'ha definit cap valor o aquest és incorrecte el valor que s'utilitza és el nombre de processadors existents en el sistema [1].

El fil de processament inicial comença a executar de forma seqüencial el codi d'usuari de l'aplicació com si aquest fos una tasca més, la qual s'anomena tasca inicial i engloba el programa complet. La resta de fils, els treballadors ('workers'), queden esperant fins que hi ha tasques disponibles per ser executades concurrentment [1].

Els diversos 'threads' executen les tasques implícites o explicites definides amb les directives d'OmpSs. La intenció del model de programació és que els programes paral·lels s'executin correctament com a programes seqüencials si s'ignoren les directives [1].

Quan un flux arriba a una clàusula del tipus 'for', l'espai d'iteració del bucle intern es divideix en diferents parts d'acord a la política de planificació especificada. Per cadascuna d'elles es crea una tasca implícita separada i tots els fluxos dins de l'equip cooperen per executar la totalitat d'aquestes. Al final de cada bucle paral·lelitzat existeix un 'taskwait' implícit, si no s'ha indicat el contrari amb la paraula reservada 'nowait' [1].

Quan un flux arriba a una clàusula del tipus 'task' es crea una nova tasca explicita. L'execució

d'aquesta és assignada a un del 'threads' de l'equip inicial, balancejant la càrrega de treball. Per tant, l'execució de la tasca pot ser immediata o retardada d'acord amb la disponibilitat dels fils o per motius de la política de planificació. A més l'execució d'una tasca pot ser suspesa en un punt de planificació per executar-se'n una altra i la continuació de la mateixa serà efectuada pel mateix 'thread', si està fermada a aquest, o per qualsevol altre, si no hi està [1].

Un altre aspecte d'OmpSs és que l'especificació no garanteix que la lectura i/o escriptura en un mateix fitxer sigui síncrona quan s'executa en paral·lel. En aquest cas, el programador és qui ha de sincronitzar les rutines o crides d'entrada i sortida amb les construccions de sincronització disponibles o altres llibreries. En el cas on cada 'thread' accedeix a un fitxer diferent no és necessari cap tipus de sincronització per part del programador [1].

Clàusula 'task' i dependències

La directiva *#pragma omp task* permet especificar regions de codi que s'executaran de forma asíncrona. Pot ser utilitzada en qualsevol bloc de codi del programa i el codi de la nova tasca serà la sentència posterior [1], com en l'exemple de la figura 3.1.

```
float x = 0.0;
float y = 0.0;
float z = 0.0;

int main() {
#pragma omp task
   do_computation(x);
#pragma omp task
   {
      do_computation(y);
      do_computation(z);
   }
   return 0;
}
```

Figura 3.1: Fragment d'exemple de l'ús de la clàusula 'task' [1]

La clàusula 'task' també permet anotar les declaracions o definicions de funcions, a més de blocs de codi. Quan una funció és anotada, cada crida a la mateixa esdevé la creació d'una nova tasca [1]. El fragment de codi de la figura 3.2 és un exemple d'aquest ús.

```
extern void do_computation(float a);
#pragma omp task
extern void do_computation_task(float a);

float x = 0.0;
int main() {
   do_computation(x); //regular function call
   do_computation_task(x); //this will create a task
   return 0;
}
```

Figura 3.2: Fragment d'exemple de l'ús de la clàusula 'task' a declaracions [1]

A la figura 3.2, la invocació de la funció *do_computation_task* dins del *main* crea una nova tasca. En els dos exemples de codi (figures 3.1 i 3.2), no es pot garantir que les tasques s'hagin executat abans que el *main* acabi la seva execució. Únicament forma part de la tasca l'execució de la funció no l'avaluació dels arguments i les tasques no poden retornar cap valor [1].

Habitualment les tasques requereixen dades per executar-se i acabar generant uns resultats que després poden ser utilitzats per altres tasques o altres parts del mateix programa. La directiva 'task' es pot estendre amb les clàusules 'in' (dades d'entrada), 'out' (dades de sortida) i 'inout' (dades d'entrada i sortida) per especificar les dades necessàries en l'execució de la tasca pertinent, a més de notificar a la resta la disponibilitat de les de sortida. La responsabilitat que aquesta informació sigui correcte recau en el programador [1].

Una tasca creada amb la clàusula 'inout' (format: `inout(llistat posicions memòria)`) per un l-valor [22] és equivalent a una creada amb les clàusules 'in' i 'out' per separat sobre el mateix l-valor [1].

Una tasca creada amb la clàusula 'in' (format: `in(llistat posicions memòria)`) per un l-valor no estarà disponible per ser executada fins que una tasca prèvia amb una clàusula 'out' sobre el mateix l-valor hagi completat la seva execució [1].

Una tasca creada amb la clàusula 'out' (format: `out(llistat posicions memòria)`) per un l-valor no estarà disponible per ser executada fins que les tasques prèvies amb una clàusula 'out' o 'in' sobre el mateix l-valor hagin completat la seva execució [1].

El format del llistat de posicions de memòria, o bé les dependències de la tasca, es defineixen en algun dels següents formats i separades per comes:

- Una variable.
- Una secció de memòria especificada pels extrems superior i inferior (ambdós inclosos): `[inferior : superior]`. Si no s'especifica una posició inferior, s'assumeix un zero. Si no s'especifica una posició superior i la posició de memòria és un vector, s'assumeix l'últim element [1].
- Una secció de memòria especificada per l'extrem inferior i el nombre d'elements: `[inferior ; #elements]` [1].

El 'runtime' d'OmpSs utilitza la informació sobre dependències i l'ordre de creació de cada tasca per analitzar les dependències entre elles. Això genera restriccions en l'ordre d'execució entre les diferents tasques perquè l'aplicació s'executi correctament. Aquestes restriccions d'ordre s'anomenen dependències entre parells de tasques i han de complir dues propietats [1]. Suposant dues tasques T1 i T2, T1 s'executarà forçosament abans que T2 si:

1. T1 s'ha creat abans que T2 i ambdues en la mateixa tasca o fora de qualsevol tasca [1].
2. T2 té almenys una posició de memòria que es solapa amb alguna de T1 i una de les dues és de sortida [1].

Cada cop que una tasca és creada les seves dependències d'entrada i sortida són comprovades contra les dependències de les tasques existents. Si se'n troba alguna, la tasca passa a dependre de la finalització de les altres, ja sigui del tipus RaW ('Read after Write', lectura després d'escriptura), WaW ('Write after Write', escriptura després d'escriptura) o WaR ('Write after Read', escriptura després de lectura). Aquest procediment genera un graf de dependències en temps d'execució. Les tasques són planificades per ser executades quan tots els seus predecessors en el graf han acabat (execució asíncrona de tasques) o bé quan són creades si no tenen predecessors [1].

L'exemple de la figura 3.3 mostra un codi amb tasques dependents i el graf de dependencies del mateix apareix a la figura 3.4.

```
void foo (int *a, int *b) {
  for (int i = 1; i < N; i++) {
#pragma omp task in(a[i-1]) inout(a[i]) out(b[i])
      propagate(&a[i-1], &a[i], &b[i]);

#pragma omp task in(b[i-1]) inout(b[i])
       correct(&b[i-1], &b[i]);
  }
}
```

Figura 3.3: Fragment d'exemple de l'ús de la clàusula 'task' amb dependències [1]

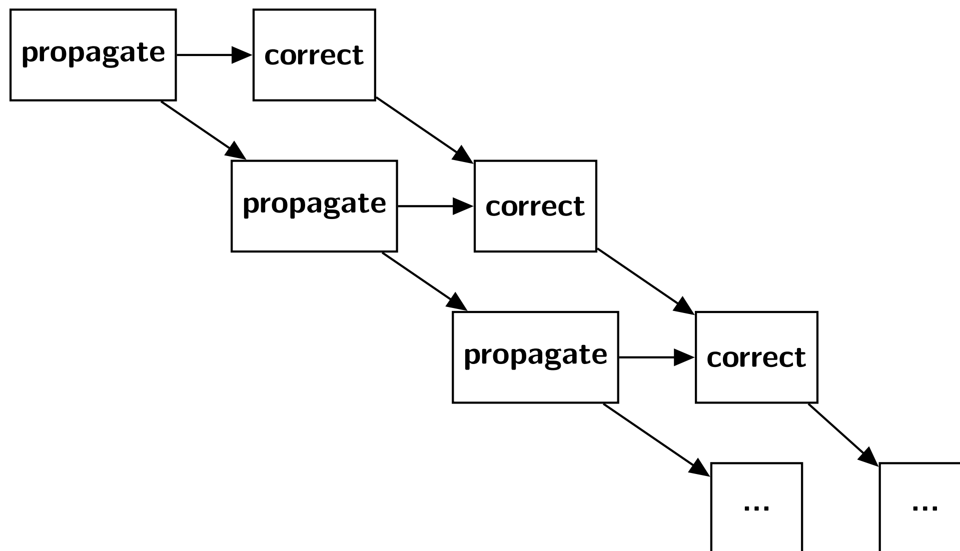


Figura 3.4: Graf de dependències [1]

3.2 Mercurium

Mercurium és un compilador 'source-to-source' pensat pel desenvolupament de prototips i proves ràpides. Els llenguatges suportats actualment són C, C++ i Fortran. Mercurium s'empra principalment en l'entorn de Nanos per implementar OpenMP i OmpSs però gràcies a la

seva extensibilitat s'ha utilitzat per implementar altres models de programació o transformacions de compilació com Cell Superscalar, memòria transaccional 'software', memòria compartida distribuïda i el projecte ACOTES, entre d'altres [11].

Aquesta extensibilitat del compilador s'efectua mitjançant 'plugins' d'arquitectura, els quals representen diferents fases del procés de compilació. Aquests afegits es desenvolupen en C++ i són carregats dinàmicament pel compilador d'acord amb la configuració triada. Les transformacions de codi són implementades en termes del codi font, per tant no és necessari modificar o conèixer la representació sintàctica interna del compilador.

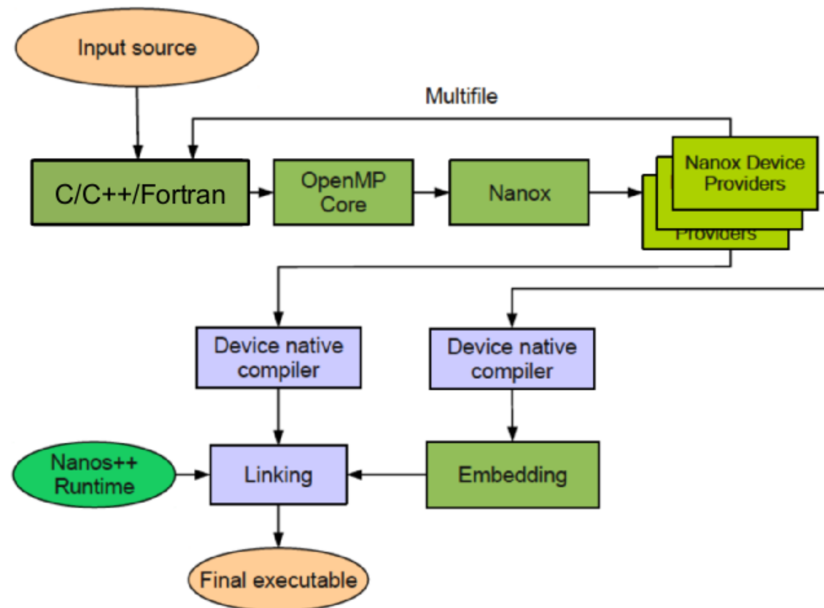


Figura 3.5: Proces de compilació de Mercurium [2]

Mercurium reemplaça les directives en el codi d'usuari per les crides corresponents al 'runtime'. Reestructurant el codi en diferents fitxers, generant 'handlers' específics per cada dispositiu i invocant diferents compiladors per generar el/els executables [2].

A la figura 3.5 apareix l'estructura global del procés de compilació.

3.3 Nanos++

Nanos++, o simplement Nanos, és un 'runtime' dissenyat per fer de suport durant l'execució d'aplicacions en entorns paral·lels i desenvolupat en C++. El seu principal ús és en el model de programació OmpSs, tot i que mitjançant mòduls pot ser estès per suportar-ne d'altres com OpenMP i Chapel [12].

Nanos permet explotar el paral·lelisme de diverses tasques utilitzant sincronitzacions basades en les seves dependències de dades. Aquestes tasques s'implementen com a fils d'execució ('threads') sempre que és possible. També permet mantenir la coherència entre diferents espais

d'adreçament com pot ser el cas de clústers o entorns d'execució heterogenis (SMP + acceleradors, entre d'altres) [12].

El 'runtime' incorpora el suport per a ser instrumentat amb la llibreria Extrae que permet obtenir traces per realitzar anàlisis de rendiment amb l'eina de visualització Paraver. Utilitzant els mòduls d'instrumentació també és possible crear el graf de dependències d'una aplicació. Aquestes eines donen molta informació per entendre millor les característiques de les aplicacions [12].

La modularitat de Nanos permet afegir-li mòduls o seleccionar en cada execució quins es volen utilitzar, ja que el mateix 'runtime' incorpora diferents versions de cadascun d'ells [1]. Els més importants o principals són:

- Polítiques de planificació de tasques ('Scheduling Policies'). Defineixen com les tasques amb les dependències satisfetes són executades. Decideixen l'ordre i el recurs on s'executen [1].
- Barreres de fils d'execució ('Barrier algorithms'). En els punts on s'han de sincronitzar els diferents 'threads' defineixen un sistema per fer-ho (arbre, centralitzat, etc.).
- Suport per dispositius. Donen suport per a l'execució de tasques en diferents tipus de dispositius (targetes gràfiques, acceleradors, FPGA, etc.), la còpia de dades entre ells, entre altres operacions.
- Formats d'instrumentació. Permeten obtenir informació de l'execució de les aplicacions en diferents formats.
- Enfocament de les dependències ('Dependence managers'). Calculen les dependències entre les tasques amb diferents precisions i algorismes.
- Polítiques d'acceleració ('Throttling policies'). Determinen quan les tasques són creades com entitats (i potencialment executades asíncronament) o són executades immediatament (com si el codi no indiques el paral·lelisme) [1].

En el cas de l'enfocament de les dependències, aquest mòdul el que determina és com són calculades, no qui les calcula. És a dir, de tota la regió que l'usuari ha definit com a dependència pot ser que únicament s'empri l'adreça del primer element com a representant de tot el bloc o el primer i darrer element (segons el 'Dependence manager'). En cap cas es determina la utilització d'un model com el proposat en aquest projecte i sempre són tots els 'threads' els que calculen les dependències, accedint a les estructures.

Les tasques que es van creant tenen la seva representació a dins de Nanos a través de la classe Work Descriptor (WD). Aquesta classe té uns atributs i mètodes que representen l'estat de la mateixa, la serva informació bàsica i com pot interactuar amb la resta del 'runtime'. El principal aspecte d'aquesta classe rellevant pel projecte són els atributs relacionats amb dependències entre tasques, que són els següents:

- DependenciesDomain *_depsDomain. Aquest objecte és l'estructura que conté el graf de dependències del Work Descriptor. S'utilitza aquesta representació perquè és la tasca pare la que s'encarrega de coordinar les seves filles, aquestes calculen les seves dependències segons les tasques germanes contingudes en el graf. Aquesta estructura distribuïda és correcte perquè les dependències de les filles sempre són un subconjunt de les del pare i per tant una tasca no necessita coordinar-se amb les seves 'cosines'.
- DOSubmit *_doSubmit. Representa el Work Descriptor en el graf de dependències del seu pare. Serveix per informar a la resta de tasques en l'estructura de la seva finalització.
- Atomic<int>_components. Comptador de tasques filles del Work Descriptor que poden o no estar en el domini de dependències.

Pel desenvolupament d'aquest projecte la implementació d'aquestes classes no és rellevant, únicament ho és l'accés a elles des dels diferents 'threads'.

Els diferents elements de processament ('cores' del processador, coprocessadors, etc.) disponibles a l'inici de cada execució són representats per la classe ProcessingElement (PE). Les instàncies d'aquesta classe vénen determinades per una mascara del sistema operatiu que especifica els identificadors dels elements. Aquesta pot ser canviada entre les execucions i per defecte conté tots els elements de processat disponibles. Els encarregats de la creació dels fils d'execució que poden interactuar amb el 'hardware' són els PE. A més, cada PE pot iniciar-ne diversos, essent responsabilitat del sistema operatiu l'execució concurrent d'aquests en el 'core', o un altre element de processat.

A la figura 3.6 apareix l'estructura global del 'runtime', amb els diferents aspectes comentats.

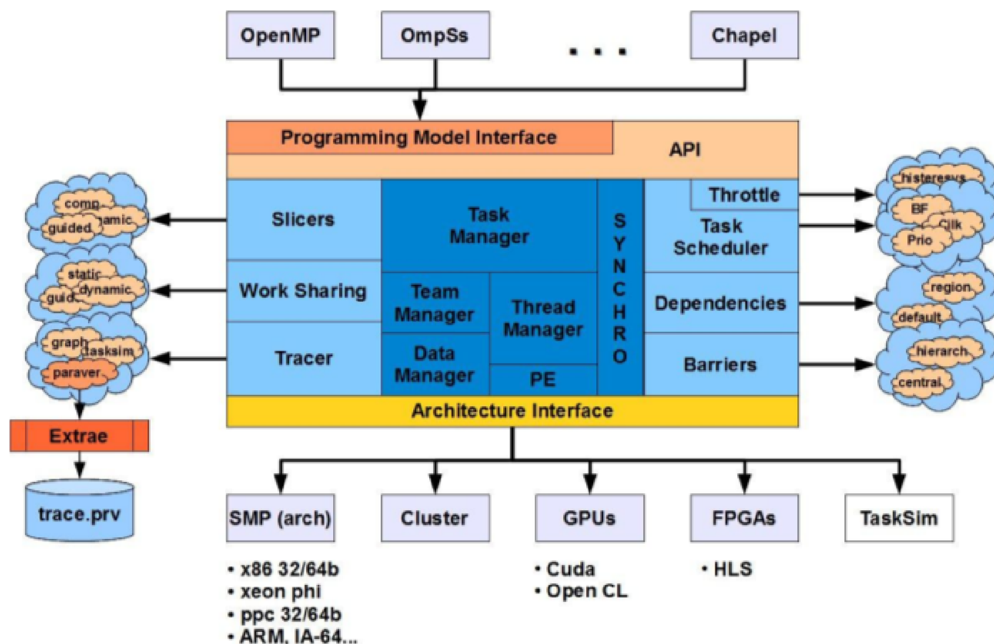


Figura 3.6: Estructura del 'runtime' Nanos++ [2]

Capítol 4

Disseny d'un model centralitzat

Aquest capítol descriu el model operacional plantejat, l'objectiu del qual és la centralització de la gestió de les dependències en un únic flux d'execució del 'runtime'. També descriu els principals aspectes a considerar en el disseny del mateix com són la sincronització, la gestió de les peticions i les garanties del nou model.

4.1 Model operacional

L'objectiu final és que sigui un 'thread' especial extra qui executi certes accions que en el 'runtime' original són realitzades per cadascun dels 'workers'. Les funcionalitats que s'efectuaran en aquest són les que impliquen un accés a les estructures que guarden les dependències entre les diferents tasques creades en l'execució d'un programa. A priori no es coneix quines són aquestes accions, així el sistema ha de ser capaç de suportar els diferents tipus d'accions originals.

El model operacional de la versió original de Nanos++ consisteix en l'accés concurrent a les estructures de dades. És a dir, cadascun dels 'threads' quan necessita accedir al graf de dependències entre tasques ho fa directament després de garantir l'exclusivitat d'ús. Com es veu a la figura 4.1, els diferents 'threads' disponibles envien directament les tasques (cercles grocs), o informació sobre aquestes (com la finalització), a la caixa que representa el graf de dependències. Quan les tasques tenen les dependències satisfetes, passen a una altra estructura que agrupa totes les que estan 'ready' (poden ser executades). D'aquesta segona estructura és d'on els diferents 'threads' extreuen tasques per ser executades quan ho requereixen, com també s'observa a la figura.

En el nou model operacional aquests enviaments d'informació cap al graf de dependències s'efectuen per un únic 'thread', a diferència de l'original. Els diferents 'threads' envien la informació sobre les tasques al nou 'thread' intermediari, en lloc del graf. Com es veu a la figura 4.2, únicament el 'Extra Thread' envia informació de les tasques (cercles grocs) cap a la caixa 'Dependencies Graph', els 'threads' envien missatges cap al fil d'execució del mig. La funcionalitat d'obtenir tasques per part dels 'threads' del 'Ready Task Pool' no es veu afectada, segueix

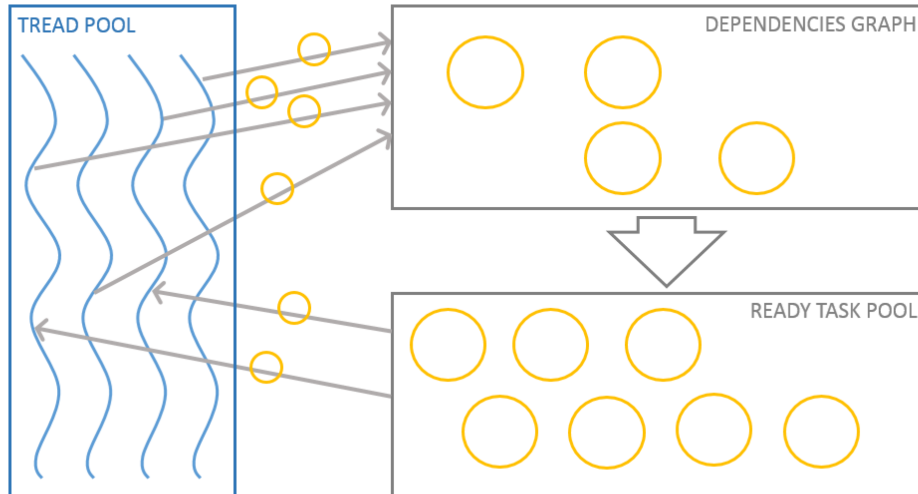


Figura 4.1: Representació gràfica del model original d'administració de les tasques

funcionat igual en ambdues versions.

La comunicació entre qui originalment executava el codi i qui ho realitza en el nou model s'efectuarà mitjançant el pas de missatges en un sistema de cues. Aquests missatges han de contenir tota la informació necessària perquè el 'thread' extra sigui capaç d'efectuar l'acció sol·licitada.

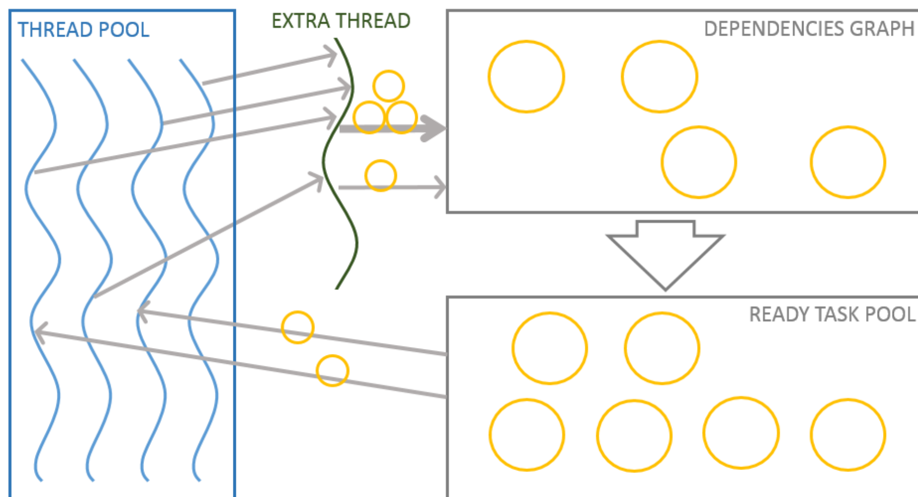


Figura 4.2: Representació gràfica del model centralitzat d'administració de les tasques

4.1.1 Sincronisme

El sistema de cues pot ser síncron, fins que la petició no es satisfà no continua l'execució, o asíncron, s'efectua la petició i immediatament continua l'execució independentment del moment en que serà satisfeta. A continuació s'analitzen els dos plantejaments.

Les peticions d'accions per part dels 'workers' poden requerir que aquests es quedin bloquejats fins que forçosament s'hagi satisfet la petició. En aquest cas la comunicació seria síncrona i s'hauria d'implementar un sistema de 'polling' que bloquegés el 'worker' fins que la petició hagi estat satisfeta. Durant aquest període d'espera podria ser útil i/o necessari indicar al planificador del 'runtime' que el 'worker' està bloquejat per aprofitar-ho i executar alguna tasca. El diagrama de flux de la figura 4.3 representa aquest plantejament.

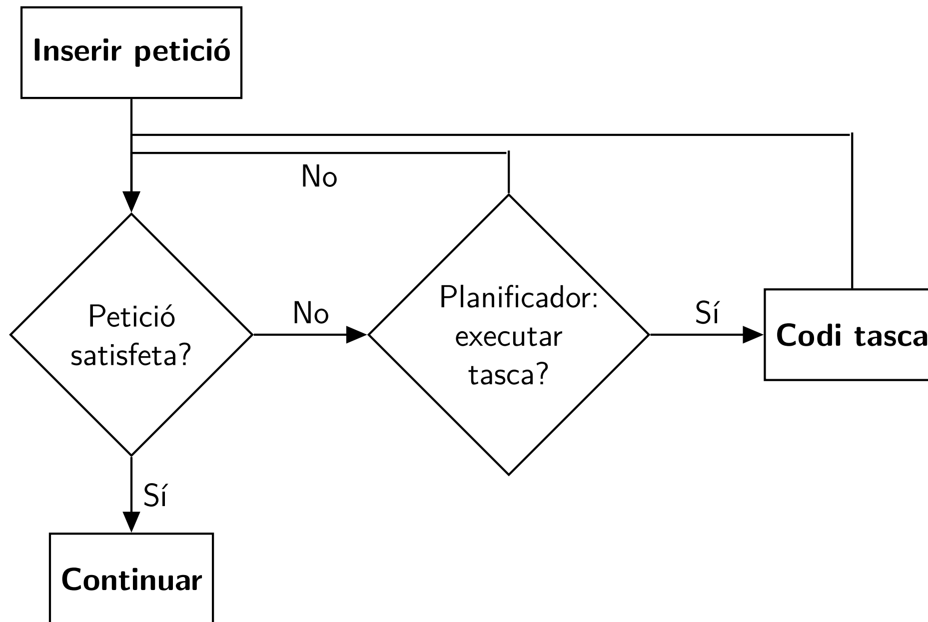


Figura 4.3: Diagrama de flux d'un 'worker' síncron

Aquest plantejament síncron difícilment redueix el temps d'execució global de la petició. Una de les motivacions de les peticions (en lloc de l'execució directa) és que el nou 'thread' les podrà executar més ràpid al no existir contenció per fer algunes accions. El problema és que aquesta reducció habitualment serà menor que el temps de bloqueig del 'worker'.

Per contra, les peticions asíncrones probablement poden implicar una millora. En aquests casos el 'worker' no necessita saber quan s'executa la petició, simplement requereix que en algun moment es faci. El flux d'una petició és molt simple, com es veu a la figura 4.4.

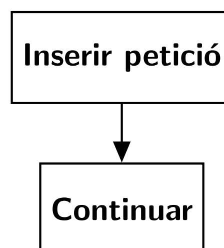


Figura 4.4: Diagrama de flux d'un 'worker' asíncron

Aquest plantejament asíncron fàcilment redueix el temps d'execució en el 'worker', sempre que el cost d'inserció sigui inferior al d'execució. Tot i aquesta reducció, el cost d'execució

únicament s'ha desplaçat a un altre 'thread', el qual hauria de fer-ho més ràpid pel fet de no tenir competència en l'accés.

Independentment del tipus de sincronisme, perquè el nou model sigui millor, amb els mateixos recursos s'ha d'aconseguir que amb un 'worker' menys s'executin el mateix nombre de tasques amb el mateix temps. Així el 'worker' restant passaria a ser el 'thread' extra que únicament executaria el codi de les peticions, per tal d'evitar les regions d'exclusió mútua en ser l'únic que accedirà a les estructures de control de dependències. A la figura 4.5 hi ha una representació d'aquest plantejament on les parts blaves són el les tasques que s'executen pels dieferents 'workers' al llarg del temps i les taronges el codi on s'accedeix a les dependències.

Figura 4.5: Diagrama d'execució original i amb el model proposat

4.1.2 Gestió de cues

El sistema de cues pot estar distribuït tenint una cua independent per a cada 'worker' o un conjunt d'ells, o bé pot ser única i compartida entre tots ells.

En el cas d'una cua única per totes les accions de tots els 'workers' el que ha d'anar fent el 'thread' especial és agafar en ordre les peticions i executar-les. Aquest plantejament té el problema que tots els 'workers' poden accedir al mateix temps al recurs tornant a aparèixer la contenció que s'intenta evitar en el model.

Per tal de minimitzar la contenció, la distribució del sistema de cues amb una cua per cada 'worker' és millor perquè així com a màxim hi ha dos 'threads' que poden competir pel recurs: el propi 'worker' en inserir-hi i en 'thread' especial per treure'n un element. Amb aquestes cues el plantejament del 'thread' que serveix les peticions es basa a anar seleccionant un dels 'workers' comprovar si la seva cua té peticions i, en cas afirmatiu, extreure-la i satisfer-la. La selecció del 'worker' pot seguir una política 'Round-Robin' a escala de petició (després de cada petició es torna a seleccionar-ne un), a escala de cua (mentre la cua no estigui buida s'extreuen peticions d'aquesta) o alguna política segons les necessitats de la implementació. El comportament del gestor descrit prèviament és el que apareix a la figura 4.6.

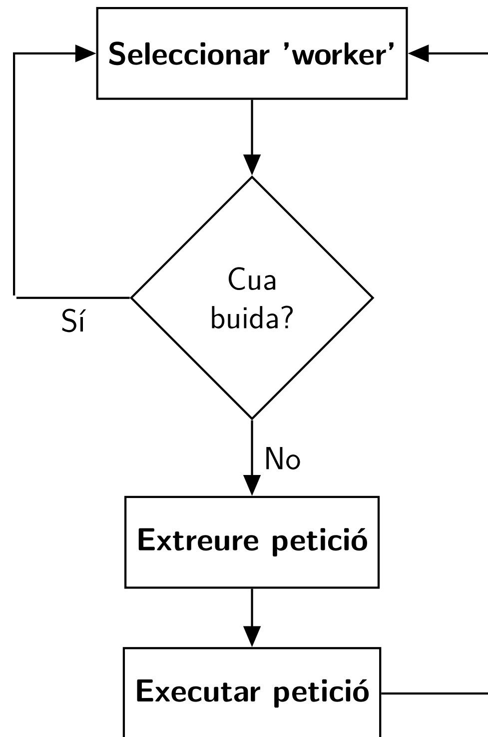


Figura 4.6: Diagrama de flux del bucle gestor dependències

4.2 Garanties

El nou model ha de garantir un seguit d'aspectes que en la versió original hi estan de forma implícita o bé explícita. Aquests estan relacionats bàsicament amb l'ordre d'execució de les peticions.

Ha de garantir l'ordre relatiu a les peticions per part d'un mateix 'thread'. En el model original un 'worker' que executi dues funcions (A i B), sempre ho farà en l'ordre del programa i per tant mai s'executarà B abans que A. Si aquestes dues tasques es converteixen en peticions al sistema centralitzat, s'ha de respectar aquest ordre perquè poden existir dependències entre les dues. En el model plantejat queda garantit si s'utilitza una cua sense prioritats, en la qual el 'worker' pel seu ordre d'execució de programa sempre hi inserirà A i després B i el gestor extraurà en aquest ordre també.

Ha de garantir l'ordre relatiu entre les accions dels diferents 'threads'. En el cas on un 'thread' executa una funció A i un altre una funció B desencadenada després d'A i ambdues passen a ser peticions, s'han d'executar A i després B, és a dir en el mateix ordre d'inserció. En el cas d'una única cua queda garantit en tenir les peticions inserides en l'ordre correcte pels propis 'workers'. En el cas de diverses cues, s'han d'utilitzar peticions síncrones (evitant el desencadenant de B fins després de la satisfacció d'A) o algun altre sistema per sincronitzar ambdues insercions o execucions, mantenint l'ordre original.

Capítol 5

Implementació

Aquest capítol explica els aspectes més rellevants respecte a la implementació del model proposat en l'anterior. Els diferents apartats es corresponen amb les tres tasques principals de desenvolupament del projecte: anàlisi del codi base per identificar les regions implicades, implementació dels canvis i optimització de la gestió.

5.1 Identificació de les regions de codi implicades

Identificar les funcions on s'accedeix a les estructures contenidores del graf de dependències de les tasques era la base per acabar de planejar un model a implementar. Els dos moments on el 'runtime' necessàriament hi ha d'accedir són a la creació d'una tasca per comprovar-ne les dependències i actuar en conseqüència i a la finalització de la mateixa per actualitzar el graf d'acord amb aquest esdeveniment i si escau notificar a les descendents.

El punt inicial per veure quin codi s'executava en crear una tasca va ser desactivar, amb un 'flag' de Mercurium, la destrucció dels fitxers intermedis durant el procés de compilació i creació de l'executable. Això va permetre veure les crides que s'efectuen al 'runtime' cada cop que una tasca amb dependències i sense elles es crea. Amb aquestes dues crides com a referència, el volum de codi font a analitzar era més reduït.

La localització de la funció de finalització d'una tasca va ser menys guiada. El WorkDescriptor associat a cadascuna de les tasques pot ser executat des de diversos punts de planificació del 'runtime'. Tot i això, tenir ja coneixement de l'estructura del codi va facilitar la cerca en general, tot i que algunes parts es van descobrir durant les proves de verificació quan van aparèixer errors en les execucions.

En els pròxims punts s'explica, pels dos moments on s'accedeixen a les estructures: les característiques, implicacions, problemes i altres aspectes a considerar pel correcte funcionament. A més d'un resum amb les regions dde codi a transformar en peticions.

5.1.1 Creació de tasques

La creació d'una tasca en el model de programació OmpSs implica una crida a l'API de Nanos. Aquesta acaba cridant a la funció *submit* o *submitWithDependencies* de la classe System segons si la tasca té dependències o no.

Quan no existeixen dependències, les accions que efectua el sistema són notificar la creació d'una nova tasca a la política de la planificació i executar la funció *submit* del WorkDescriptor associat a la tasca. Aquesta funció es limita a delegar la responsabilitat de què fer amb ella mateixa al planificador. El planificador decideix si executar-la o enviar-la a alguna de les cues d'execució segons les característiques del 'thread', del mateix WorkDescriptor i de l'estat del 'runtime' en aquell punt, entre altres. Sabent que no hi ha cap dependència, en aquesta cadena d'operacions no s'accedeix a les estructures de dependències. Per tant en aquest cas no era necessari efectuar modificacions en el codi, ni fer cap petició, perquè la finalitat del gestor es executar el codi que accedeix a les estructures que guarden les dependències.

En el cas que la tasca sí tingui dependències, les accions que efectua el sistema són les següents: notifica la política de planificació i executa la funció *submitWithDependencies* del WorkDescriptor pare que és el WD que estava executant el 'thread' en aquell moment i que n'ha creat un de nou per representar la tasca. Aquesta funció inicialitza els atributs del fill que el representen dintre les dependències del pare i s'utilitzen per comunicar-se en acabar d'executar-se, inicialitza els possibles accessos commutatius a dades i executa la funció *submitDependableObject* sobre el domini de dependències del pare.

En aquesta part del codi sí que s'accedeix al graf de dependències entre les tasques i per tant almenys l'execució de la funció *submitDependableObject* de la classe DependenciesDomain s'havia de convertir en una petició. Per tal de mantenir l'atomicitat entre la notificació a la política de planificació i la resta de codi, la millor opció era que la petició englobés tot el codi de la classe System. L'execució d'aquesta secció no és necessària perquè el 'thread' que efectua la petició pugui continuar la seva execució, així que aquesta petició podia ser asíncrona. Per tal de fer-ho possible és necessari reservar memòria i copiar-hi les estructures que a la versió original estan a la pila del 'thread'. Així aquestes estan disponibles per ser utilitzades pel gestor més endavant.

Un aspecte important de Nanos pel que fa a la creació de tasques és no tenen per què arribar a crear-se. En funció de l'estat del 'runtime' i de la 'throttle policy' la tasca pot ser executada immediatament, independentment de què el programador hagi especificat condicions amb les clàusules `if` o `final`. Amb l'objectiu de controlar aquest aspecte i poder recrear sempre les mateixes condicions s'ha canviat la política de 'trottle' utilitzant una que sempre decideix crear la tasca ('dummy'), sense considerar cap aspecte.

5.1.2 Finalització de tasques

Sempre que una tasca finalitza la seva execució el planificador del 'runtime' crida a la funció *done* del WorkDescriptor que ha completat la seva execució. Aquí és on es porten a terme les

principals accions relacionades amb aquest esdeveniment.

Aquesta funció s'encarrega de notificar la finalització de l'execució a la resta d'accessos commutatius, a la interfície del model de programació i al WorkDescriptor pare, després espera que els seus fills acabin d'executar-se (aquests han de poder referenciar-lo per alliberar les dependències de dades) i finalment notifica al seu pare que el deixa de referenciar i que per tant, si l'estava esperant, pot acabar. S'observen tres parts diferenciades amb objectius diferents: una primera que notifica l'alliberació de dependències als implicats, una segona que espera els fills i l'última que notifica la finalització real al pare.

La part on s'accedeix a les estructures amb les dependències és la primera de les tres i és la que s'ha de convertir en una petició per seguir el model plantejat. La segona no té sentit que l'executi el 'thread' extra perquè durant aquesta espera el planificador aprofita el 'worker' per executar altres tasques pendents i aquest no és el comportament desitjat. La tercera part és indiferent qui l'executí, però s'ha de fer després de la finalització de la primera perquè aquesta utilitza estructures que en l'ultima de les tres es des-referencien. Per garantir aquest ordre hi havia dues possibles opcions: que la petició fos síncrona o que les pues parts fossin peticions asíncrones (el gestor garanteix l'ordre entre peticions d'un mateix element). Per tal de seguir el mateix criteri que en la creació i donat que a priori el sistema asíncron és més avantatjós, s'han convertit ambdues regions en peticions.

A més de les consideracions prèvies, existeixen dues altres crides o regions que també s'ha de considerar: la funció *getImmediateSuccessor* i el destructor de la classe WorkDescriptor, a més del *delete* o *free* per alliberar la memòria on estan els seus atributs.

La primera de les dues, *getImmediateSuccessor*, accedeix al graf de dependències per extreure'n una de les tasques descendents directament de la que efectua la crida amb les dependències satisfetes. Aquesta funció únicament l'utilitzen algunes polítiques de planificació del 'runtime' per aprofitar la localitat de dades en la notificació *atBeforeExit*. Quan una tasca depèn d'una altra és perquè accedeix a una posició de memòria que la primera està utilitzant, així doncs aquesta estarà, probablement, a la memòria cau del processador la qual és molt més ràpida [23]. El codi d'aquesta funció havia de ser executat també pel gestor i el valor obtingut havia de ser retornat a la política de planificació. S'ha optat per delegar al gestor l'execució de la notificació a la política (*atBeforeExit*) de planificació, convertint en una petició la crida de notificació. Aquesta crida no és important quan s'executi ni qui ho faixi, mentre es porti a terme abans de la finalització de la tasca, que es correspon amb la primera de les tres parts descrites de la funció *done*. Per tant, al ser inserida a la cua abans que les peticions prèvies, queda assegurat l'ordre d'execució entre la notificació a la política de planificació i el codi de finalització.

La segona és quan una tasca acaba la seva execució i ja no serà referenciada pels seus fills, deixant de tenir cap utilitat. En aquest moment, Nanos allibera la memòria que estava ocupant aquesta tasca i crida al destructor del WorkDescriptor. En la versió original aquesta crida s'efectua sempre després d'executar el codi de la funció *done*. Amb el nou plantejament asíncron aquesta alliberació no té per què efectuar-se després de les peticions, el 'worker' podria executar-la abans que el gestor serveixi la petició. Així, per garantir aquest ordre, qui ha d'alliberar la memòria és el gestor de peticions i on hi havia una crida al destructor s'ha d'ubicar una nova petició.

5.1.3 Resum de regions o peticions

D'acord amb les peculiaritats expressades prèviament, a continuació es llisten les diferents regions identificades. L'execució d'aquestes ha d'efectuar-se en un únic 'thread' i la resta de 'workers' han d'efectuar les peticions per tinents.

- Creació d'una tasca, càlcul de les dependències i inserció al graf.
- Notificació a la política de planificació prèvia a la finalització d'una tasca.
- Alliberament de les dependències d'una tasca i notificació a les tasques afectades.
- Notificació al pare de la tasca de finalització definitiva de la mateixa.
- Alliberament de la memòria associada.

5.2 Creació del gestor

La implementació del model i la creació del nou 'thread' centralitzat ha constat de diverses parts, cadascuna de les quals és una part imprescindible pel correcte funcionament del mateix. Una part és la que implica les estructures (sistema de cues) pel pas dels missatges (peticions) entre els actors, una altra part són tots els canvis en el codi original per ubicar-hi les peticions, una part és la creació del 'thread' extra i el seu bucle d'execució, i una part final són les modificacions secundàries en altres parts però igual de necessàries.

Tot i la concreció del model i de les regions a implementar com a peticions, alguns aspectes més tècnics d'implementació s'havien de concretar abans o durant el desenvolupament per mantenir una línia coherent i facilitar en part el procés de desenvolupament. En els pròxims punts s'explica les diferents alternatives valorades i la decisió presa amb la pertinent justificació. Posteriorment es detallen les parts d'implementació llistades prèviament.

5.2.1 Valoració d'alternatives

Alguns aspectes del desenvolupament del projecte poden ser efectuats de diferents formes amb el mateix resultat final. Per cadascun d'aquests s'ha intentat valorar les diferents alternatives i motivar una decisió el millor possible.

Nom del Thread extra: DAST

Una de les primeres de decisions preses, en iniciar el projecte, va ser decidir un nom per a la nova versió centralitzada del 'runtime'. Ja que aquesta tenia un 'thread' amb molta importància, és el que administra les tasques a efectuar per la resta, es va fer un joc de paraules amb l'article alemany 'DAS', que vol dir el, i 'thread' creant el nom de DASThread (DAST). Tot i directament

no implicar molt en la implementació sí que ha repercutit en ella en aspectes com el nom de les classes o facilitant la comunicació en tenir un nom de referència per la versió.

Estructuració del codi font

Un punt a decidir abans de començar a desenvolupar qualsevol projecte és decidir una estructura bàsica a seguir. En aquest cas, per ubicar els nous fitxers en el projecte existent. Després de l'etapa d'anàlisi i d'una reunió amb persones que ja havien treballat en el 'runtime' original existien dues opcions per ubicar les noves classes:

- Core. Aquesta opció suposava integrar directament les noves classes en el nucli del codi.
- Arquitectura SMP. Aquesta opció suposava integrar les noves classes en el codi específic per a l'arquitectura SMP.

De les dues opcions es va escollir integrar-lo en l'arquitectura SMP perquè aquest 'thread' extra seria d'aquest tipus i no de les altres arquitectures. Així doncs el codi quedava més delimitat que si s'integrava directament en el nucli.

Referent a estructura del codi font també es va decidir seguir les guies d'estil establertes prèviament en el desenvolupament del 'runtime' original per no tenir una part totalment diferent de la resta.

Activació del model centralitzat

La centralització de la gestió de dependències s'ha de poder activar/desactivar d'alguna forma, ja que no sempre pot ser interessant que estigui activa. Per fer aquesta elecció existien dues alternatives o dos moments on poder decidir-ho:

- En temps d'execució. Activar/desactivar el DAST amb una opció del 'runtime' en començar l'execució d'un programa. Aquesta opció permet amb una mateixa versió compilada tenir la possibilitat de seleccionar el mode i no haver de tenir dues versions compilades i anar canviant la que s'utilitza. Per contra, aquesta decisió implica un temps extra durant l'execució en molts de punts del sistema.
- En temps de compilació. Activar/desactivar el DAST amb una opció de compilació del 'runtime'. Aquesta opció implica haver d'anar canviant tots els fitxers que formen les llibreries cada cop que es vulgui canviar de versió, a més d'haver de compilar-les dos cops. Per contra, en executar els programes no s'ha de decidir que fer, tenint un 'overhead' menor.

De les dues opcions es va escollir la segona, en temps de compilació, perquè la versió original s'hauria de compilar molts pocs cops i si se cerca reduir el temps d'execució afegir condicionals en el 'runtime' no ajuda.

Estructures de dades

Per desenvolupar el projecte es requereixen diverses estructures com cues, elements de sincronització, etc. Aquestes podien ser implementades de nou pel projecte, es podrien utilitzar les estàndard de C++ o utilitzar les ja existents a la resta del 'runtime' original. L'opció escollida i la més lògica és les que ja s'estan utilitzant perquè estan funcionant i tenen aspectes propis que poden ser necessaris per funcionar correctament (suposant un menor cost per aconseguir que funcioni el nou model).

5.2.2 Estructures de dades

La principal estructura que requereix el model per funcionar és el sistema de cues que fa possible que cadascun dels 'workers' faci peticions. Segons el model proposat aquesta cua havia de ser distribuïda per evitar una alta contenció en l'accés a ella, per tant cada 'worker' té la seva instància.

Per implementar aquesta cua s'ha creat la classe DASTData la qual està formada per un 'lock' i una cua. El 'lock' garantirà l'accés en exclusió mútua a la cua. Això és necessari perquè és possible inserir i extreure de la cua al mateix temps per fils d'execució diferents ('worker' i DAST) i s'ha de garantir l'atomicitat d'aquestes operacions.

Els elements que s'insereixen i extreuen de la cua són d'un tipus especial anomenat DASTWork, la declaració d'aquest tipus s'observa a la figura 5.1. Aquest tipus conté tota la informació necessària per que el DAST serveixi tots els tipus de peticions, les quals s'identifiquen amb el camp *_type* que és del tipus DASTWorkType (figura 5.2) i pren un valor o un altre segons la petició efectuada. En algunes peticions no s'empren tots els camps o aquests serveixen per diferents aspectes, com per exemple el camp *_dependencies* que únicament es fa servir per una de les peticions. No s'han creat diferents tipus al tenir únicament tres punters desaprofitats, en el pitjor des casos, que no suposen un gran desaprofitament de memòria; no s'espera tenir molts d'elements en les cues i així es simplifica la gestió.

```
typedef struct DASTWork_t {
   WD               *_creator;
   WD               *_wd;
   size_t            _number;
   DataAccess       *_dependencies;
   DASTWorkType      _type;
} DASTWork;
```

Figura 5.1: Declaració del tipus DASTWork

La classe DASTData també té definits diversos mètodes per inserir-hi peticions, un per cada tipus de petició amb els paràmetres necessaris per executar-la posteriorment, i un mètode per obtenir un punter a l'element del cap de la cua o NULL en el cas que estigués buida.

Per aconseguir la distribució especificada en el model, cada BaseThread (objecte bàsic que representa un 'worker') té en els seus atributs una instància de l'estructura plantejada. Això

```
typedef enum {
   DAST_WORK_IN,
   DAST_WORK_BEFORE_EXIT,
   DAST_WORK_OUT,
   DAST_WORK_EXIT,
   DAST_WORK_DELETE
} DASTWorkType;
```

Figura 5.2: Declaració del tipus DASTWorkType

permet que cada 'worker' s'invoqui sobre si mateix el mètode corresponent a la petició que vulgui efectuar.

5.2.3 Peticions

La forma d'efectuar les peticions des dels diferents 'workers' fins al DAST consisteix en què cada un d'ells crida un mètode de l'objecte del tipus BaseThread que el representa. Aquest mètode pot accedir a la seva pròpia cua i inserir-hi allà la petició mitjançant la crida a la funció corresponent de DASTData.

Cadascun dels canvis en el codi per efectuar les peticions ha estat introduït dintre d'uns condicionals de compilació que permeten activar o desactivar el model. Podent així compilar una versió del 'runtime' amb les peticions i el model centralitzat o bé una amb l'execució del codi distribuïda en els 'workers'. Per cada petició especificada en l'apartat 5.1 es descriuen les peculiaritats de la seva implementació.

Creació d'una tasca

La funció *submitWithDependencies*, que s'executa quan un programa crea una tasca, rep un punter a una estructura que conté la informació de les dependències. Aquesta informació està a la pila perquè un cop calculades les dependències ja no es necessita més. En el nou model s'ha afegit la reserva de la memòria per guardar una còpia de l'estructura perquè, al ser asíncron, la posició indicada pel punter ja no tindria per què contenir la informació necessària. Després, com s'observa a la figura 5.3 en funció de les opcions de compilació s'hi trobarà una petició sobre el 'thread' que està executant la funció (model del DAST) o el codi que executaria la petició més l'alliberació de la regió de memòria.

Per satisfer posteriorment aquesta petició s'ha de guardar el WorkDescriptor, el WD pare del que s'està creant/inicialitzant, la copia de l'estructura amb les dependències i el numero de dependències que apunta aquesta estructura. Totes aquestes variables són els paràmetres per a la funció que genera la petició al DAST.

```
void System::submitWithDependencies (WD& work, size_t numDataAccesses, DataAccess* dataAccesses)
{
   WD *current = myThread->getCurrentWD();
#if defined(NANOS_ENABLE_DAST)
   DataAccess *dependencies = ( DataAccess * ) malloc( sizeof( DataAccess )*numDataAccesses );
   memcpy( dependencies, dataAccesses, sizeof( DataAccess )*numDataAccesses );
   myThread->submitDASTNewWork( current, &work, numDataAccesses, dependencies );
#else
   SchedulePolicy* policy = getDefaultSchedulePolicy();
   policy->onSystemSubmit( work, SchedulePolicy::SYS_SUBMIT_WITH_DEPENDENCIES );
   current->submitWithDependencies( work, numDataAccesses, dataAccesses );
#endif
}
```

Figura 5.3: Codi de la funció System::submitWithDependencies

Notificació a la política de planificació

La funció *finishWork* del planificador (codi que apareix a la figura 5.4) és des d'on s'efectua la notificació a la política de planificació. Aquesta notificació retorna un punter a un WorkDescriptor, que és el successor que ha d'executar el 'thread' i per tant s'ha d'inserir a la cua de WD pròxims que ha d'executar. En convertir la notificació en una petició al DAST perquè aquest efectuï la notificació, també s'ha delegat la inserció del valor retornat.

```
void Scheduler::finishWork( WD * wd, bool schedule )
{
   updateExitStats (*wd);

   //! \note getting more work to do (only if not going to sleep)
   if ( schedule && !getMyThreadSafe()->isSleeping() ) {
      BaseThread *thread = getMyThreadSafe();
      ThreadTeam *thread_team = thread->getTeam();
      if ( thread_team ) {
#if !defined(NANOS_ENABLE_DAST)
         thread->addNextWD( thread_team->getSchedulePolicy().atBeforeExit( thread, *wd, schedule ) );
#else
         myThread->submitDASTBeforeExitWork( wd, schedule );
#endif
      }
   }

   wd->done( schedule );
   wd->clear();

   (...)
}
```

Figura 5.4: Codi de la funció Scheduler::finishWork

Per satisfer aquesta petició, el gestor necessita la referència al 'thread' a qui ha d'inserir el valor retornat, a més dels paràmetres per efectuar la notificació: 'thread' que ha acabat l'execució, WD a finalitzar i el paràmetre binari 'schedule' que rep la funció *finishWork* . Aquestes variables són les que s'utilitzen com a paràmetre a la funció que crea la nova petició i les guarda a l'estructura DASTWork, llevat del 'thread' perquè aquest està implícit gracies al fet de tenir una cua per cada un d'ells.

Alliberació de les dependències

En la funció *done* que s'executa quan una tasca finalitza la seva execució, s'ha afegit el condicional per o bé compilar la crida a la funció que allibera les dependències de la tasca o bé la crida a la funció que insereix la nova petició. A la figura 5.5 s'observa aquesta condició a l'inici de la funció. Aquesta funció no rep cap paràmetre així que únicament s'ha hagut de relacionar la petició amb el WD sobre el que invocar finalment la funció *done*.

```
void WorkDescriptor::done ()
{
#if !defined(NANOS_ENABLE_DAST)
   dependenciesDone();
#else
   myThread->submitDASTDoneWork( this );
#endif

   // Waiting for children (just to keep structures)
   if ( _components != 0 ) waitCompletion();

   // Notifying parent about current WD finalization
#if !defined(NANOS_ENABLE_DAST)
   notifyParent();
#else
   myThread->submitDASTExitWork( this );
#endif
}
```

Figura 5.5: Codi de la funció WorkDescriptor::done

Alliberació del pare

Com també s'observa en la figura 5.5, un cop tots els fills d'un WorkDescriptor han acabat la seva execució i el pare ha retornat de l'espera (crida a la funció *waitCompletion*), s'ha afegit un condicional de compilació més. Aquest decideix entre o bé notificar el pare directament, o bé, efectuar la petició al DAST perquè realitzi aquesta acció, garantint així l'odre entre la notificació de les dependències i la notificació de la finalització. Com la petició anterior, únicament s'envia el WD sobre el que cridar el mètode *notifyParent*.

Eliminació del WorkDescriptor

S'han efectuat canvis a les crides al destructor de la classe WD amb l'objectiu de no eliminar-los mentre siguin necessaris per a l'execució de les peticions en el DAST i evitar deixar els apuntadors a WD, sobre els quals invocar els mètodes, inutilitzables. Els canvis han estat efectuar la crida a la funció que genera la petició, la qual únicament rep com a paràmetre el WD que s'ha d'eliminar, alliberant la memòria que ocupa. Així com en el cas de l'alliberació del pare queden garantides les dependències entre les diferents parts del codi asíncron.

5.2.4 Thread extra

La creació dels 'threads' en OmpSs és implícita a l'inici de l'execució d'un programa, per tant la creació del nou 'thread' per gestionar les dependències també és implícita durant l'inici del programa. L'objectiu d'aquest, mentre s'executa el programa, és satisfer les peticions dels 'workers', per tant el codi que ha d'executar és un bucle que continuament espera peticions. Aquest acaba quan s'han executat totes les tasques existents, inclús la 'master', i per tant ja no poden haver-hi més peticions a executar.

Creació

La creació del DAST s'efectua en la funció *start* del sistema, després de la creació dels 'worker threads'. Durant la inicialització del sistema és reserva un dels 'threads' que es tenen previst crear. Si l'usuari no ha especificat un nombre, s'utilitza el nombre de processadors existents en el sistema, o si l'usuari ha definit el nombre desitjat, s'incrementa el nombre en un. Per la inicialització del 'thread' extra s'ha creat una funció especial de la classe ProcessingElement. Aquesta és anàloga a la que inicia qualsevol altre 'thread' però en lloc d'enviar-lo a executar el bucle del planificador, fa que executi el bucle del DAST.

Per defecte el DAST es crea en el ProcessingElement següent al del darrer 'worker' creat, sempre que sigui deferent al del fil principal. El fet d'estar en el mateix element i haver d'executar els dos 'threads' concurrentment pot degradar el rendiment de les aplicacions. El PE assignat al DAST és rellevant i afecta al rendiment del 'runtime' com s'explica en el capítol 6.

Bucle d'espera de peticions

El bucle d'execució del DAST segueix el model planejat en el capítol 4 amb alguns afegits per aconseguir el comportament esperat. A cada iteració, es recorre el vector de 'workers' existent en el sistema i per cadascun d'ells efectua un seguit d'accions.

La primera és comprovar que el DAST pertany al mateix ThreadTeam que el 'worker' (línia 11 de la figura 5.6) perquè les diferents polítiques de planificació disponibles a Nanos poden planificar les tasques en funció d'aquest equip, utilitzant com a referència l'equip del 'thread' que està efectuant la notificació de què la tasca és executable. En la majoria d'iteracions, els diferents 'workers' pertanyeran al mateix equip i per tant no s'executarà la funció *enterTeam*. Aquesta té un cost considerable i per tant és millor efectuar el condicional que executar-la en cada iteració incondicionalment.

A continuació i mentre hi hagi elements a la cua d'aquest 'worker', va extraient peticions amb la funció *tryPop* (línia 17 de la figura 5.6) i executant el codi corresponent en funció del tipus de petició. Per reduir la contenció en l'accés a les cues, s'ha implementat una doble comprovació en l'extracció d'elements, optimització coneguda com a 'test and test&set'. En aquesta, únicament s'utilitzen les instruccions atòmiques per comprovar si existeixen elements i extreure'n un (amb garanties d'atomicitat) si prèviament s'ha comprovat que no està buida (comprovació feta sense

garanties d'atomicitat).

A la figura 5.6 hi ha el codi simplificat del bucle amb la funcionalitat descrita prèviament.

```
void DASThread::workLoop () {

   while ( myThread->isRunning() ) {
      memoryFence();

      if ( sys.mustDASTStop() )     break;

      for ( int workerIdx = 0; workerIdx < sys.getNumWorkers(); ++workerIdx ) {
         BaseThread *worker = sys.getWorker( workerIdx );

         if ( worker->getTeamData() != myThread->getTeamData() ) {
            myThread->enterTeam( worker->getTeamData() );
         }

         DASTData *dastData = worker->getDASTData();
         DASTWork* work;
         while ( dastData->tryPop( work ) ) {
            if ( work->_type == DAST_WORK_IN ) {
               SchedulePolicy* policy = sys.getDefaultSchedulePolicy();
               policy->onSystemSubmit( *work->_wd, SchedulePolicy::DAST_SUBMIT );
               ( work->_creator )->submitWithDependencies( *( work->_wd ),
                                                            work->_number,
                                                            work->_dependencies );
               free( ( void * )work->_dependencies );
            } else if ( work->_type == DAST_WORK_OUT ) {
               ( work->_wd )->dependenciesDone();
            } else if ( work->_type == DAST_WORK_DELETE ) {
               ( work->_wd )->~WorkDescriptor();
               delete[] (char *)work._work;
            } else if ( work->_type == DAST_WORK_EXIT ) {
               ( work->_wd )->notifyParent();
            } else if ( work->_type == DAST_WORK_BEFORE_EXIT ) {
               SchedulePolicy* policy = &( worker->getTeam()->getSchedulePolicy() );
               WD* wd = policy->atBeforeExit( worker, *( work->_wd ), work->_number )
               worker->addNextWD( wd );
            } else {
               fatak( "Invalid DAST work type" );
            }
         }
      }
   }
   getMyThreadSafe()->exitTeam();
}
```

Figura 5.6: Codi del bucle del DAST

Finalització

La finalització del bucle s'efectua mitjançant la condició situada a l'inici de cada iteració del mateix, en la figura 5.6 és la línia 6. Aquesta crida comprova si el DAST ha de sortir del bucle d'espera de peticions mitjançant la crida a una funció de la classe System. En el cas dels 'workers' la comprovació amb aquesta finalitat s'efectua sobre la classe BaseThread, però les anotacions com 'static' i 'volatile' en certes funcions derivaven en condicions de carrera o endarreriments en aquest cas. A causa de les implicacions d'algunes modificacions necessàries en algunes funcions, es va crear un nou mecanisme independent i exclusiu pel DAST que garantís la seva finalització

en el moment desitjat.

5.2.5 Altres modificacions

Tot i els canvis descrits prèviament, diversos errors d'execució han aparegut en el transcurs de les proves que s'han anat efectuant. Aquests són aspectes del del 'runtime' que feien que el comportament final del model no fos l'esperat o provocaven acabaments abruptes en l'execució dels programes.

El fet que el DAST no comptabilitzi com a 'worker' va provocar que en activar la instrumentació en un programa de prova i s'intentés afegir informació a la traça des del DAST saltes una excepció i finalitzés bruscament l'execució. Aquest problema era degut al fet que la instrumentació utilitza el nombre de 'workers' com a nombre de 'threads' que poden enviar informació a la traça i en aquest model aquesta relació no es compleix. Per solucionar-ho s'ha afegit una condició de compilació que incrementa en una unitat el valor que rep la instrumentació si el DAST està actiu.

Un altre aspecte és que quan un WorkDescriptor passa a estar disponible per ser executat (després que es compleixin les seves dependències, per exemple) es notifica l'esdeveniment al planificador del sistema que pot decidir que el 'thread' que realitza la notificació executi directament la tasca. En el model plantejat qui efectua aquesta crida és el DAST però no es vol que ell executi la funció, per aquest motiu s'ha canviat aquest comportament del planificador per fer que si és el DAST qui està executant el codi, sempre es delegui la responsabilitat d'executar-lo a un altre 'worker' del sistema. En la figura 5.7 apareix aquest fragment de codi que busca qui executi el WorkDescriptor i en cas de no aconseguir-ne cap llança una excepció.

```
(...)
#if defined(NANOS_ENABLE_DAST)
   if (sys.getDASThread() == myThread) {
      BaseThread * wThread = sys.getInactiveWorker();

      // If no worker found, get one
      if (wThread == NULL) wThread = sys.getWorker(0);
      // If no worker found, throw error
      if (wThread == NULL) fatal("No worker found to execute the wakedUp WD.");

      ThreadTeam *wTeam = wThread->getTeam();
      if ( wTeam ) wTeam->getSchedulePolicy().atWakeUp( wThread, *wd );
      else fatal("Trying to wake up a WD from a thread without team.");

      return;
   }
#endif
(...)
```

Figura 5.7: Fragment de codi de la funció Scheduler::wakeUp

També s'han efectuat altres modificacions a la instrumentació no necessàries per al correcte funcionament del nou model, però que eren interessants per obtenir traces de les execucions amb informació extra respecte les peticions efectuades. Concretament, s'han creat nous codis

d'instrumentació i s'han fet les crides pertinents a l'inici i final de les regions per registrar aquestes. En el cas de la instrumentació amb la llibreria Extrae, això ha permès obtenir uns fitxers amb tota la informació de l'execució i visualitzar-los a Paraver per analitzar l'execució posteriorment.

5.3 Optimització

L'optimització de la gestió de les peticions és un aspecte clau del projecte perquè com més ràpid es gestioni cada petició més se'n poden atendre i les possibles esperes per part dels 'workers' són més breus, desaprofitant menys la seva capacitat de càlcul. Algunes de les millores introduïdes en la gestió ja estaven previstes en la planificació del projecte, com l'eliminació de regions d'exclusió, d'altres s'han efectuat en observar comportaments no desitjats en les aplicacions de prova o aspectes fàcilment millorables.

5.3.1 Regions d'exclusió mútua

Les regions d'exclusió mútua són útils per garantir que un 'thread' efectua un seguit d'accions de forma atòmica sense inferències d'altres i per tant el resultat sempre és correcte. El cost a pagar és la possibilitat de bloquejar la resta de 'threads' que volen entrar en ella fins que no hi hagi ningú en la mateixa.

A la versió original de Nanos, aquests són necessaris en accedir al graf de dependències perquè els diversos 'workers' hi poden treballar al mateix temps. A causa de la utilització de la versió original com a base, aquests seguien existint en el codi de cada gestor de dependències un cop implementat el model centralitat. En aquest únicament un 'thread' és qui pot accedir a les estructures de les dependències i les regions d'exclusió no tenen sentit. Per tant, una millora evident és la seva eliminació perquè, tot i que no és possible que el DAST es bloquegi en una d'elles esperant per accedir a la regió, l'execució de les instruccions corresponents als 'locks' tenen un cost i amb la seva eliminació aquest desapareix.

El procediment per eliminar-les ha estat amb condicionals pel compilador. Si el DAST està activat, les instruccions d'exclusió no es compilen i si s'està compilant una versió del 'runtime' en mode 'debug' en el seu lloc s'hi posiciona una comprovació per assegurar que aquelles parts de Nanos únicament les executa el DAST.

5.3.2 Balanceig de càrrega

El nombre de peticions que el DAST efectuava inicialment per cada 'worker' no estava limitat. L'objectiu era evitar estar accedint constantment a les estructures amb les referències a aquests i evitar possibles canvis del 'thread team' del DAST.

Però això pot fer aparèixer un problema amb les prioritats del treball del DAST. Aquest es donaria en una aplicació on un dels 'workers' crea al principi totes les tasques i la majoria

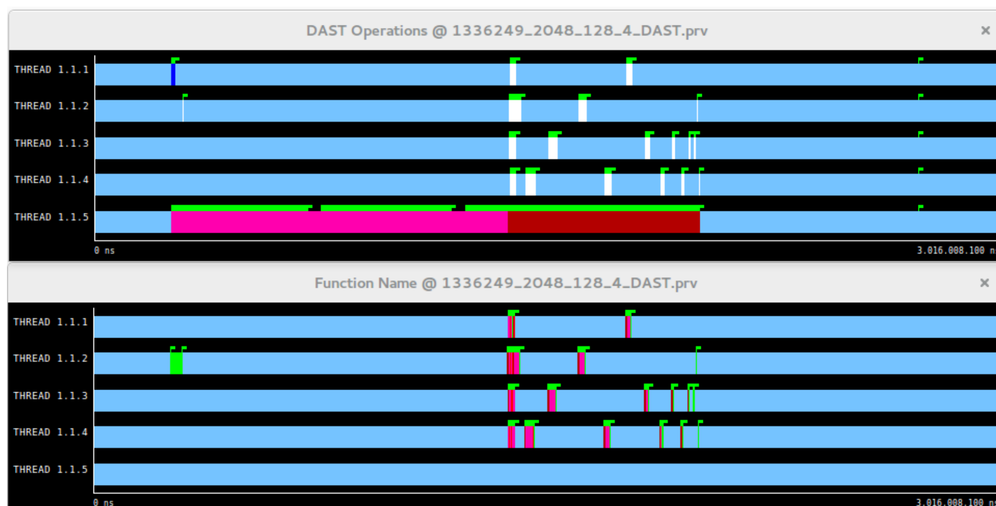


Figura 5.8: Traces d'execució d'una aplicació amb un mal balanceig del DAST

d'aquestes depenen de les primeres poques (directament o indirectament). El problema consisteix en que el DAST prioritzaria les peticions d'un 'worker', el del que crea les tasques, per damunt del treball de la resta, el 'worker' que executes les primeres tasques. La finalització d'aquestes primeres tasques és important perqué obre el paral·lelisme existent a l'aplicació. Al no satisfer aquesta petició, es bloqueja l'execució fins que totes les tasques han estat creades i el DAST canvia el 'worker' amb el que treballar.

A la figura 5.8 es poden veure dues traces (l'eix X representa el temps d'execució) de la mateixa execució d'una aplicació amb aquestes característiques i amb amb 4 'workers' ('threads' 1.1.1, 1.1.2, 1.1.3 i 1.1.4) més el DAST ('thread' 1.1.5). En la traça superior, les parts blau fosc indiquen que s'està creant una petició de creació, les roses que s'està servint una d'aquest tipus, les blanques que s'està creant una petició de finalització o satisfacció de dependències i les marrons que s'està servint aquest tipus. En la traça inferior es representa l'execució de les tasques, cada color representa una de les tasques diferents de l'aplicació. En aquest cas el DAST està creant les tasques durant molt de temps a l'inici, i per tant servint les peticions del 'thread' 1.1.1, mentre té una petició de finalització del 'thread' 1.1.2 que ha executat la tasca verda a l'inici i de la qual depenen la resta. El fet de no servir aquesta petició suposa que cap de les tasques que s'estan creant poden ser executades, repercutint en dos aspectes: els 'workers' estan en estat IDLE sense fer treball útil i el DAST està tardant més a crear les tasques en haver de gestionar un graf de dependències amb moltes més tasques i més complex.

La idea aplicada per evitar aquest problema és tan simple com limitar en nombre de peticions que es satisfan per cada 'worker' en una iteració i anar-les servint en un patró similar a Roud-Robin. Això crea un balanceig de la càrrega del DAST entre tots els 'threads' i es redueix la pressió sobre una mateixa cua.

A la figura 5.9 apareixen les mateixes traces de la mateixa execució amb la correcció aplicada. La primera de les tres traces és amb la mateixa escala temporal que la figura 5.8 i com es pot observar la millora és molt significativa i fa que no es pugui apreciar res. Per aquest motiu les

Figura 5.9: Traces d'execució d'una aplicació amb el balanceig del DAST corregit

dues següents estan a una escala diferent, es corresponen a les dues de la figura 5.8. Com es pot apreciar, el DAST satisfà les peticions molt més ràpid i els 'workers' executen les tasques d'una forma molt més compacta.

5.3.3 Fusió de peticions

En finalitzar una tasca el nombre de peticions a efectuar al DAST és de quatre, les quals gairebé són consecutives entre elles. Després d'analitzar el seu codi i d'analitzar el codi que s'executa entre elles, es va observar que la notificació *atBeforeExit* a la política de planificació es podia retardar fins just abans que el DAST executés el codi de *dependienciesDone* (que correspon a la següent petició, la d'alliberació de les dependències). També es va comprovar que l'alliberació del pare podia ser efectuada just després d'alliberar les dependències, sense esperar que els fills hagin fet la mateixa acció.

Amb aquest anàlisi es van poder agrupar tres peticions en una única: notificació a la política de planificació, alliberació de les dependències i alliberació de pare. Aquest canvi no comporta una reducció de la càrrega del DAST perquè el volum de codi a executar segueix essent el mateix però sí que redueix el nombre d'accessos a la cua de peticions (els quals estan protegits amb 'locks'), a més d'utilitzar menys memòria per peticions. També suposa que una reducció de l'overhead' del codi del 'runtime' en els 'workers', al no haver d'executar res en algunes parts (ni el codi original, ni el codi de generar la petició).

Capítol 6

Avaluació del rendiment

Aquest capítol explica el procés d'avaluació del rendiment pel nou model enfront l'original, descriu els entorns on s'han portat a terme les proves, com s'han fet, les diferents aplicacions utilitzades i per cadascuna d'elles els resultats obtinguts. S'han hagut d'utilitzar diferents programes en l'avaluació perquè aquest projecte modifica el 'runtime' d'un model de programació i per tant el que s'executen són les aplicacions amb el suport de les llibreries (part modificada). Segons el programa que s'executi per sobre del 'runtime' el rendiment d'aquest varia pel volum de tasques, la distribució temporal d'aquestes, les dependències expressades, etc. A més d'aquesta anàlisi comparatiu basat en aplicacions sintètiques i reals, apareix una anàlisi centrada a determinar com les diferents millores/alternatives d'implementació han millorat el 'runtime' amb el model centralitzat, també comparant-lo amb el model original.

6.1 Entorns d'execució

La nova versió del 'runtime' s'ha utilitzat per a l'execució d'aplicacions en diferents entorns. Durant el període de desenvolupament i d'acord amb la metodologia de treball, s'ha executat en l'equip portàtil del desenvolupador amb petites aplicacions per efectuar tests funcionals. Un cop acabat el desenvolupament, s'han efectuat proves de rendiment en un node del clúster Arvei. I finalment s'han portat a terme proves de rendiment en un Intel Many Integrated Core (MIC) del cluster Knights del BSC.

En ambdós entorns el procediment per compilar els diversos 'runtimes' i els programes de prova ha estat similar. S'ha fet una instal·lació d'OmpSs alternativa a la ja existent, s'ha fet pensant amb la possibilitat de canviar la versió de Nanos utilitzada a cada execució simplement canviant el 'soft-link' on Mercurium va a buscar els fitxers amb les llibreries per generar els executables de les aplicacions. Aquest 'link' ha d'apuntar a una altra carpeta amb la instal·lació de la versió pertinent de Nanos. El 'flag' utilitzat en la compilació, com a norma general, han estat el '–ompss' que activa el model de programació desitjat.

6.1.1 Arvei

Arvei és un clúster de computació d'altes prestacions del Departament d'Arquitectura de Computadors de la Universitat Politècnica de Catalunya. Està format per diferents tipus de nodes, la diferència principal entre ells és la quantitat de memòria i el model/quantitat de processadors. L'accés als recursos d'aquest clúster, principalment la capacitat de càlcul, s'efectua mitjançant un sistema de cues, el qual va llançant les diferents execucions dels usuaris segons les seves característiques i la disponibilitat.

Les condicions d'execució que garanteix aquest sistema de cues per defecte no són les més adequades per avaluar el rendiment del projecte. Aquest clúster té configurades diferents cues d'execució les quals garanteixen aspectes com un temps màxim de càlcul o un nombre de nodes per executar l'aplicació. En cap cas garanteix l'exclusivitat d'execució en un node per un treball de les cues, únicament garanteix l'exclusivitat del node entre treballs que sol·licitin l'exclusivitat (aquesta funcionalitat és útil per evitar que dos treballs siguin assignats al mateix node i s'interfereixin entre ells). Aquesta exclusivitat és molt important per obtenir uns resultats de rendiment reals i reproduïbles en aquest projecte. D'existir més processos, el sistema operatiu del node distribueix els recursos entre tots ells canviant el procés que s'està executant periòdicament. Aquesta concurrència entre processos genera una variabilitat en els temps d'execució difícil de controlar i que per tant pot distorsionar els resultats.

Per tenir unes millors garanties en les execucions del sistema, es va sol·licitar la creació d'una cua especial, exclusiva per aquest projecte, que garantís que no existien més treballs en execució en el node. Per aconseguir-ho el que succeeix és que la resta de cues en el clúster no poden enviar treballs a un node concret, únicament pot fer-ho la nova. A més, aquesta no planifica treballs en cap més node. Gràcies a això s'aconseguia garantir que sempre s'executarien les aplicacions en el mateix tipus de node.

El node assignat té dos NUMA (Non-Uniform Memory Access) nodes, amb un 'scoket' cadascun i un processador model Intel Xeon E5-2630L a 2.00 GHz de freqüència màxima en cada 'socket'. Cadascun d'aquests dos processadors té 6 'cores' (nuclis de processament) els quals poden executar de forma paral·lela dos fils d'instruccions compartint recursos com la memòria cau. En total existeixen 12 cores i fins a 24 amb 'hyper-threading'. Tots ells poden accedir als 64 GB de memòria principal disponible en el node. A la figura 6.1 apareix una versió simplificada d'aquesta arquitectura. En aquesta, cadascuna de les caixes de punts discontinus representa un NUMA node, amb un processador (requadre blau fluix), amb diversos 'cores' (requadres blau fosc), amb dos fils d'execució. També hi apareixen les diferents unitats de memòria de la jerarquia (requadres grisos).

Els nodes del clúster tenen dos espais de disc diferenciats: el local i el de xarxa. El primer correspon al disc local instal·lat en cada node i que únicament pot ser accedit des d'ell mateix. El segon és un NAS extern als nodes, accessible des de tots ells i amb servei de còpia de seguretat. El temps d'accés canvia entre un i l'altre, essent molt menor i estable el temps d'accés al disc local. Tot i això, per evitar pèrdues de dades, els fitxers del projecte han estat guardats en el disc de xarxa, evitant també tenir diverses còpies distribuïdes en diferents nodes.

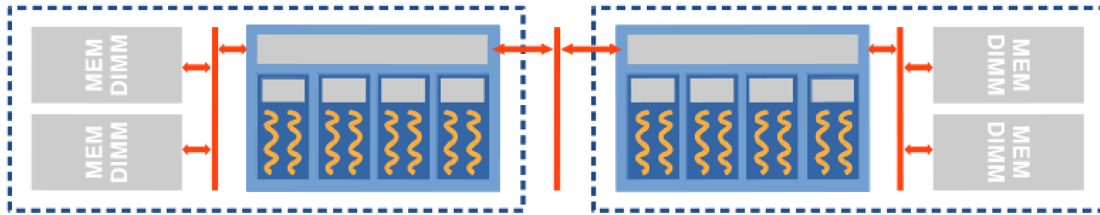


Figura 6.1: Representació de l'estructura de dos processadors NUMA

Metodologia

La metodologia que s'ha seguit en totes les execucions d'aplicacions efectuades en Arvei ha estat la mateixa. L'objectiu d'aquesta és garantir uns resultats fiables i reproduïbles de l'avaluació. Aquesta està descrita en els següents punts:

- Cada execució consisteix en l'execució d'un 'script' bash que s'encarrega de fer totes les accions necessàries com copiar arxius, definir variables d'entorn, obtenir mesures de temps, etc.
- Les execucions s'efectuen mitjançant el sistema de cues per garantir l'exclusivitat del node i evitar inferències d'altres processos, per exemple l'administrador de la connexió ssh.
- A l'inici de les execucions es copien els fitxers necessaris en el disc local del node per evitar la variabilitat dels accessos a un disc de xarxa.
- Cada execució es repeteix 3 cops, com a mínim, i s'agafa la mitja dels tres com a mètrica.

Altres consideracions

Un aspecte comú en totes les execucions en aquest entorn, i apreciable en les gràfiques, és que quan s'utilitzen 24 'workers' el rendiment del 'runtime' DAST cau considerablement per l'associació que s'efectua dels 'threads' a les diferents unitats de càlcul. Aquesta baixada és deguda al fet que el nombre total de 'threads' en el sistema és superior al nombre de fils d'execució disponibles i per tant el thread DAST (creat l'ultim) passa a executar-se de forma concurrent amb un 'worker'. Aquesta concurrència implica que totes les peticions siguin satisfetes amb molt més de retard i els 'workers' desaprofitin la seva capacitat de càlcul perquè depenen de les accions del DAST.

6.1.2 Knights

Els Intel Many Integrated Core són uns coprocessadors que incorporen en un mateix xip molts nuclis computacionals (molts més que els processadors actuals), a més d'altres elements com controladors de memòria. Cadascun dels nuclis MIC és un nucli x86 independent i funcional que pot executar diferents fils d'instruccions x86 ('threads' o processos) [3].

A diferència de la majoria de coprocessadors, els programes originalment dissenyats per a ser executats en una CPU poden córrer també en un xip MIC. Per poder crear fluxos d'execució i/o processos, els MICs executen un petit sistema operatiu basat en Linux que possibilita les principals tasques de gestió. D'acord amb aquestes, el model d'ús és simètric, és a dir, el coprocessador pot ser considerat com un mini super-computador independent [3][24].

La figura 6.2 mostra l'estructura d'un sistema basat en un processador Intel Xenon estès per un coprocessador MIC. Els MICs utilitzats són de la família 'Knights Corner' (KNC) d'Intel, que és el nom de desenvolupament utilitzat per l'empresa en la seva implementació de l'arquitectura MIC [3].

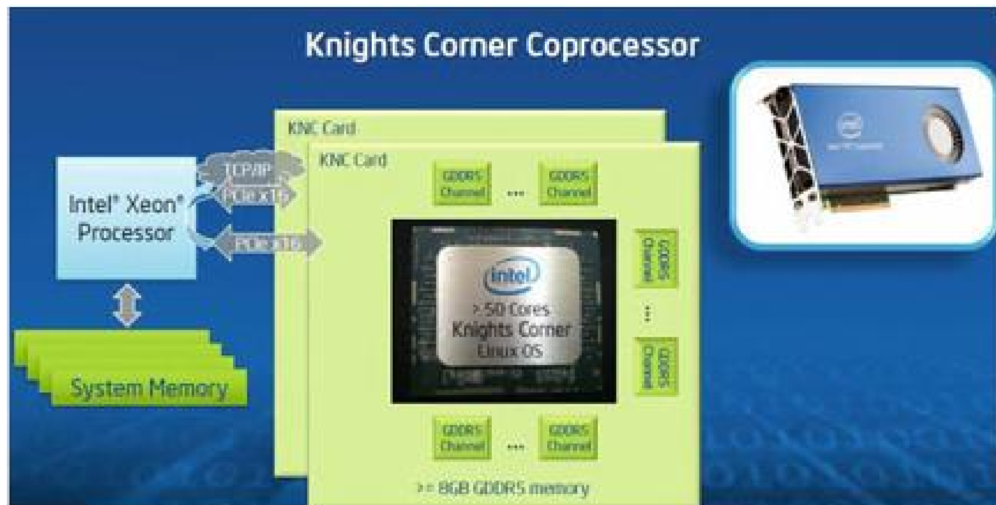


Figura 6.2: Extensió d'una CPU Intel Xenon amb un coprocessador MIC [3]

Per executar alguns dels programes de prova s'ha utilitzat un d'aquests coprocessador, sempre el mateix, model C0PRQ-7120. Aquest disposa de 61 nuclis que poden executar fins a 4 fils d'instruccions cadascun, tenint en total 244 fils d'execució disponibles a 1.238 GHz de freqüència [25]. Durant les proves el màxim que se n'ha utilitzat són 240, deixant lliure el nucli on s'executa el sistema operatiu de la placa. Aquesta està ubicada en el clúster Knights del BSC.

A diferència de les execucions en el node d'Arvei, en aquest entorn no es pot garantir l'exclusivitat del coprocessador. L'accés a aquests dispositius s'efectua per connexió SSH i no amb un sistema de cues. Tot i que a causa de l'ús minoritari d'aquests dispositius no es plantegen tants de problemes. Abans de cada execució i després s'ha comprovat que no hi hagués cap altre usuari connectat al coprocessador que pogués interferir en la mateixa. No s'han detectat més usuaris ni abans ni després de totes les execucions realitzades.

Metodologia

La metodologia que s'ha seguit en totes les execucions d'aplicacions efectuades al clúster Knights ha estat la mateixa. L'objectiu d'aquesta és garantir uns resultats fiables i reproduïbles de l'avaluació. Aquesta està descrita en els següents punts:

- Cada execució consisteix en l'execució d'un 'script' bash que s'encarrega de fer totes les accions necessàries com copiar arxius, definir variables d'entorn, obtenir mesures de temps, etc.
- Abans i després de cada execució es comprova que no hi hagi cap procés en execució, llevat dels del sistema operatiu.
- Cada execució del programa es repeteix 3 cops, com a mínim, i s'agafa la mitja dels tres com a mètrica.
- Es mesura el temps utilitzant 1, 2, 4, 8, 18, 32 i 60 'cores'.
- Es mesura el temps utilitzant 1, 2, 3 i 4 fils d'execució per 'core'. Es a dir, per cada possible valor de nombre de nuclis és repeteixen s'executen els quatre ràtios.
- Definir amb *taskset* una mascara dels processadors per l'execució per aconseguir la distribució dels 'cores' desitjada.

Altres consideracions

Un aspecte comú en totes les gràfiques d'execucions en aquest entorn és que s'ha utilitzat un eix X categoritzat, és a dir, la separació entre punts en aquest eix és arbitrària i no respecta ni una escala lineal ni logarítmica. S'ha utilitzat aquest sistema per facilitar la lectura de les mateixes i considerant que no es perd una part imprescindible de la informació. Aquest eix es correspon amb el nombre de 'cores' (nuclis de processament), cadascun d'ells té des d'un fins a quatre fils d'instruccions en execució (corresponents a un 'worker' de Nanos en aquest projecte) segons la gràfica.

6.2 Aplicacions sintètiques

El primer grup de d'aplicacions de prova utilitzat per avaluar el rendiment de la versió final del nou Nanos++ ha estat sintètic. Aquest esta format per programes creats específicament per comprovar el rendiment de la nova versió en entorns, teòricament, favorables a ell.

6.2.1 Test 1

Aquest programa de prova va ser creat amb l'objectiu de maximitzar la contenció i comparar el rendiment amb diferents volums de tasques i de diferents granularitats. L'estructura del programa s'observa en la figura 6.3. Un aspecte a destacar d'aquest programa és que les tasques més internes poden tenir dependències exclusivament amb la resta de tasques creades en la mateixa regió de codi o tasca, com s'ha explicat en el capítol 3. A més el possible resultat d'una execució no és reproduïble perquè diverses tasques concurrents escriuen en les mateixes posicions de memòria

sense controlar l'ordre, en aquest cas, no és un aspecte important al ser irrellevant pel propòsit del test.

```
ARRAY_SIZE = 1500

for(int p = 0; p < PARALLEL_CREATIONS; ++p) {
  #pragma omp task
  {
    for (int i = 0; i < ARRAY_SIZE; ++i) {
      #pragma omp task in(A[i]) out(B[i]) fistprivate(i)
        B[i] = doSometing(A[i], TASK_SIZE);
    }
    for (int i = 0; i < ARRAY_SIZE; ++i) {
      #pragma omp task in(B[i]) out(C[i]) fistprivate(i)
        C[i] = doSometing(B[i], TASK_SIZE);
    }
    #pragma omp taskwait
  }
}
#pragma omp taskwait
```

Figura 6.3: Pseudocodi del Test 1

Com s'observa a la figura 6.3, el valor de TASK_SIZE repercuteix en la duració de les tasques de nivell més intern, PARALLEL_CREATIONS indica el nombre de creacions paral·leles de tasques de nivell intern i també repercuteix en el nombre de tasques d'aquest tipus que es poden executar concurrentment, el valor ARRAY_SIZE repercuteix en el nombre de tasques de nivell intern creades (valor fixat a 1500 en totes les execucions).

6.2.2 Test 2

Aquest programa segueix la mateixa estructura que el Test 1, la diferència recau en la introducció d'un factor d'aleatorietat en la duració de les tasques evitant una possible col·lisió periòdica dels 'workers' en la finalització de les tasques. La creació de tasques no pot coincidir amb una altra creació, en tot cas amb una finalització. Tot i així també s'ha introduït un temps aleatori entre creació i creació com s'aprecia a la figura 6.4.

D'acord amb la figura 6.4, el valor de TASK_SIZE repercuteix en la duració mitjana de les tasques de nivell més intern, PARALLEL_CREATIONS indica el nombre de creacions paral·leles de tasques de nivell intern i també repercuteix en el nombre de tasques d'aquestes que es poden executar concurrentment, el valor ARRAY_SIZE repercuteix en el nombre de tasques de nivell intern creades (valor fixat a 10000 en totes les execucions) i VAR que repercuteix en la variabilitat de la duració de les tasques i el temps d'espera entre la creació de les mateixes. En totes les execucions que s'han efectuat s'ha utilitzat un factor VAR de 0.9, s'ha escollit aquest valor després d'efectuar unes execucions simples amb diferents valors i valorar la repercussió d'aquest factor en el resultat. Aquesta era insignificant per a valors inferiors i el que s'aconseguia era més aviat limitar el paral·lelisme de l'aplicació, fet que s'ha de considerar en intentar comparar els resultats amb els del Test 1.

```
ARRAY_SIZE = 10000

for(int p = 0; p < PARALLEL_CREATIONS; ++p) {
  #pragma omp task
  {
    int dispersion2 = max((int)((double)(TASK_SIZE)*VAR), 1);
    int dispersion = max(dispersion2/2, 1);
    for (int i = 0; i < ARRAY_SIZE; ++i) {
        int iters = TASK_SIZE - dispersion + rand()%dispersion2;

      #pragma omp task in(A[i]) out(B[i]) fistprivate(i)
        B[i] = doSometing(A[i], iters);

        loseTime(rand()%dispersion);
    }
    for (int i = 0; i < ARRAY_SIZE; ++i) {
        int iters = TASK_SIZE - dispersion + rand()%dispersion2;

      #pragma omp task in(B[i]) out(C[i]) fistprivate(i)
        C[i] = doSometing(B[i], TASK_SIZE);

        loseTime(rand()%dispersion);
    }
    #pragma omp taskwait
  }
}
#pragma omp taskwait
```

Figura 6.4: Pseudocodi del Test 2

6.3 Aplicacions reals

Les diferents aplicacions reals utilitzades per avaluar el rendiment de la versió final del nou Nanos++ han estat seleccionades d'acord amb uns criteris per tal que tinguessin sentit les execucions i tingues sentit la utilització del nou model plantejat. Aquests criteris són tals com: tenir taques amb dependències entre elles, la quantitat de tasques fos considerable i no fos una aplicació recursiva.

6.3.1 Cholesky

Aquest programa de prova calcula la descomposició de Cholesky [26] d'una matriu en paral·lel. La versió utilitzada s'ha extret del repositori d'aplicacions del BSC, 'BSC Application Repository' (BAR), i rep dos paràmetres en el moment d'execució que són: `MATRIX_SIZE` i `BLOCK_SIZE`. El primer paràmetre determina la dimensió de la matriu sobre la qual calcular la factorització i el segon la dimensió de les submatrius en les quals es descompon la matriu per fer el càlcul.

6.3.2 Matrix Multiply

Aquest programa de prova calcula la multiplicació de dues matrius de forma paral·lela. La versió utilitzada s'ha obtingut del repositori d'aplicacions del BSC i rep dos paràmetres en el

moment d'execució que són: MATRIX_SIZE i BLOCK_SIZE. El primer paràmetre determina la dimensió de la matriu sobre la qual calcular la factorització i el segon la dimensió de les submatrius en les quals es descompon la matriu per fer el càlcul.

6.3.3 Sparse LU

Aquest programa de prova calcula la descomposició LU ('Lower Upper') d'una matriu de forma paral·lela. La versió utilitzada s'ha obtingut del repositori d'aplicacions del BSC i efectua el càlcul en funció de dos valors: NB i BSIZE. El primer paràmetre determina el nombre de blocs que compondran la matriu i el segon la dimensió de cada bloc, tenint en total una matriu de dimensions: $\texttt{NB} * \texttt{NB} * \texttt{BSIZE} * \texttt{BSIZE}$.

6.4 Anàlisi de les diferents versions

Les mesures del temps d'execució de regions del codi de Nanos depenen de l'aplicació sota la qual s'executen, el nombre de 'threads' que s'utilitzin en aquella execució, la càrrega del sistema operatiu en aquell moment (la llibreria fa peticions a aquest), etc. Les condicions sota les quals s'han efectuat les següents proves en Arvei són similars a les d'execució dels programes de prova: l'entorn ha estat el mateix i el procediment per fer les mesures s'ha basat en l'ús de la crida a sistema *clock_gettime* que proporciona una precisió propera als nanosegons. En temps d'execució, les diferents mesures s'anaven escrivint a un fitxer agrupades segons el tipus de mesura i, en temps de post-execució, es llegien aquests fitxers i s'extreien les mètriques necessàries. Algunes mesures són de l'ordre de 200-300 nanosegons, valors molt petits i volàtils. Els resultats d'aquesta secció són un interval de confiança ($\alpha = 0.95$) d'acord amb els valors de la població de mostres recollida (mínim 7000).

Cadascuna de les mesures s'ha efectuat amb tres dels programes de prova i per cadascun d'ells amb 1 i 12 'workers'. Com s'observa en els resultats dels sub-apartats posteriors, el nombre de 'workers' influeix de forma significativa en els temps d'execució. La principal motivació d'aquests increments és la contenció creada, en augmentar els 'workers', en els diferents elements atòmics del 'runtime' i en les crides a llibreries o sistema.

6.4.1 Eliminació de les regions d'exclusió mútua

Una de les finalitats del nou 'runtime' era l'eliminació de les regions d'exclusió en les dues funcions principals que accedeixen al graf de dependències: *submitWithDependencies* i *dependenciesDone*. Aquestes ja no són necessàries i la seva execució suposava un cost no útil. Per avaluar el rendiment d'aquest canvi s'han compilat tres versions diferents de Nanos que han servit per mesurar el temps d'execució de les dues funcions: l'original (columnes 'ORIG'), una nova mantenint els 'locks' (columnes 'DAST Locks') i la nova sense 'locks' (columnes 'DAST').

	#workers	ORIG	DAST Locks	DAST
Test 1	1	2792 ± 88 ns	2078 ± 76 ns	2171 ± 83 ns
(1, 25000)	12	6033 ± 255 ns	3538 ± 149 ns	3157 ± 226 ns
Test 2	1	1573 ± 76 ns	2354 ± 126 ns	1780 ± 95 ns
(1, 25000)	12	3838 ± 75 ns	3153 ± 68 ns	2782 ± 63 ns
Matrix Multiply	1	1982 ± 29 ns	2876 ± 72 ns	2597 ± 77 ns
(512, 32)	12	6280 ± 287 ns	4084 ± 142 ns	3387 ± 112 ns

Taula 6.1: Temps d'execució de la funció *submitWithDependencies*

L'execució de la funció *submitWithDependencies* depèn directament de les dependències especificades en cada tasca i de les existents en el graf durant l'execució, per tant, depèn de cada aplicació. Dels temps d'execució d'aquesta funció s'observen diferents aspectes:

- L'increment del temps d'execució entre 1 i 12 'workers' és molt més significatiu per la versió original (2.6x contra 1.5x).
- La versió amb regions d'exclusió redueix al voltant d'un 30% el temps d'execució de la funció i sense les regions la reducció és d'un 40% amb 12 'workers'.
- La implementació d'aquesta millora ha permès reduir el temps d'execució de la funció.

	#workers	Original	DAST Locks	DAST
Test 1	1	4786 ± 209 ns	3689 ± 131 ns	3532 ± 152 ns
(1, 25000)	12	11619 ± 450 ns	2989 ± 309 ns	2647 ± 217 ns
Test 2	1	40802 ± 739 ns	37884 ± 700 ns	33597 ± 594 ns
(1, 25000)	12	3819 ± 151 ns	1627 ± 4 ns	1145 ± 4 ns
Matrix Multiply	1	1705 ± 10 ns	1815 ± 13 ns	1215 ± 10 ns
(512, 32)	12	9583 ± 513 ns	2212 ± 17 ns	1430 ± 15 ns

Taula 6.2: Temps d'execució de la funció *dependenciesDone*

L'execució de la funció *dependenciesDone* també depèn del nombre de dependències especificades a la tasca a més de les tasques existents en el graf. Els principals punts que s'observen en els temps d'execució són:

- Els temps d'execució en els programes Test 1 i Matrix Multiply són molt més estables en la versió del DAST.
- La versió amb regions d'exclusió redueix al voltant d'un 70% el temps d'execució de la funció i sense les regions la reducció és d'un 78% amb 12 'workers'. Amb 1 'worker' les reduccions són de 8% i 25%, respectivament.
- La implementació d'aquesta millora ha permès reduir el temps d'execució de la funció.

6.4.2 Fusió de peticions

Una millora també efectuada en el nou Nanos ha estat agrupar tres peticions independents en una única com s'explica en l'apartat 5.3. Per avaluar la possible millora en el temps d'execució s'han compilat dues versions del 'runtime' (Nanos DAST amb 3 peticions i Nanos DAST amb 1 petició) amb mesures de temps d'inserció de les tres peticions i en la satisfacció de les mateixes. Els temps que apareixen a la taula 6.3 són la suma dels temps, de realització o de satisfacció, de les tres peticions, en un cas, i el temps d'una única, en l'altre cas. D'aquests temps els principals aspectes a destacar són:

- El temps d'inserció per una petició única és inferior al de tres peticions en tots els casos.
- El temps que inverteix en DAST a executar el codi de la petició és del mateix ordre en ambdós casos.
- Aquesta millora ha reduït el cost del nou model en els 'workers' i ha simplificat la gestió per part del DAST.

	#workers	3 peticions		1 petició	
		Workers	DAST	Workers	DAST
Test 1	1	839 ± 4 ns	4024 ± 85 ns	328 ± 8 ns	4044 ± 167 ns
(1, 25000)	12	4179 ± 967 ns	2204 ± 88 ns	3431 ± 1889 ns	2351 ± 282 ns
Test 2	1	945 ± 5 ns	34141 ± 303 ns	294 ± 7 ns	34321 ± 602 ns
(1, 25000)	12	1450 ± 104 ns	1308 ± 11 ns	703 ± 144 ns	1325 ± 5 ns
Matrix Multiply	1	812 ± 4 ns	1972 ± 10 ns	276 ± 3 ns	1776 ± 22 ns
(512, 32)	12	973 ± 15 ns	3289 ± 13 ns	338 ± 14 ns	2978 ± 31 ns

Taula 6.3: Temps d'execució del DAST amb una i tres peticions

6.4.3 Nanos original vs Nanos centralitzat

L'objectiu final del nou model plantejat és millorar el temps d'execució de les aplicacions modificant la forma d'administrar les tasques. Per comparar la possible millora en el temps d'execució del codi de les peticions s'han compilat dues versions de Nanos (l'original i la versió nova final) amb mesures del temps d'execució d'aquestes regions. Les tres regions de codi modificades que han implicat diferents tipus de peticions pel gestor centralitzat i que es comparen a continuació són: càlcul de dependències (DAST_WORK_NEW), notificació a la política de planificació + alliberació de dependències + alliberació del pare (DAST_WORK_DONE) i alliberament de la memòria (DAST_WORK_DELETE). A la taula 6.4 apareix el temps estimat en cada aplicació pels tres tipus de peticions que fan els 'workers' (temps d'efectuar la petició) i a les taules 6.5, 6.6 i 6.5 apareixen els temps estimats d'execució del codi d'aquestes en la versió original (columna

	#workers	WORK_NEW	WORK_DONE	WORK_DELETE
Test 1	1	298 ± 13 ns	313 ± 2 ns	293 ± 10 ns
(1, 25000)	12	338 ± 20 ns	3604 ± 1881 ns	276 ± 6 ns
Test 2	1	300 ± 4 ns	275 ± 4 ns	241 ± 3 ns
(1, 25000)	12	216 ± 4 ns	845 ± 218 ns	380 ± 5 ns
Matrix Multiply	1	169 ± 9 ns	305 ± 5 ns	340 ± 5 ns
(512, 32)	12	312 ± 14 ns	343 ± 8 ns	313 ± 16 ns

Taula 6.4: Temps de creació i inserció de les peticions dels 'workers'

'Original') i la nova (columna 'DAST'), a més del temps acumulat entre la generació i satisfacció de les peticions en el nou model (columna 'DAST + Petició').

Les mètriques de la taula 6.4 són bastant similars per les tres peticions i independentment de l'aplicació i el nombre de 'workers'. L'únic valor que destaca és el cas de la petició DAST_WORK_DONE del Test 1 amb 12 'workers'. S'ha de considerar que aquest programa, amb els paràmetres d'execució donats, executa unes tasques molt petites que acaben molt ràpidament i per tant s'efectuen peticions constantment. Aquest ritme suposa un estrès per la jerarquia de memòria que es veu reflectit en la gran dispersió d'aquest valor i en l'augment del valor mitjà.

	#workers	Original	DAST	DAST + Petició
Test 1	1	2624 ± 91 ns	2241 ± 74 ns	2539 ± 38 ns
(1, 25000)	12	6112 ± 243 ns	3600 ± 375 ns	3938 ± 191 ns
Test 2	1	1776 ± 75 ns	1938 ± 69 ns	2238 ± 35 ns
(1, 25000)	12	4205 ± 77 ns	2895 ± 74 ns	3111 ± 38 ns
Matrix Multiply	1	2024 ± 30 ns	2665 ± 77 ns	2834 ± 40 ns
(512, 32)	12	6085 ± 292 ns	3543 ± 149 ns	3855 ± 76 ns

Taula 6.5: Temps d'execució del codi de la petició DAST_WORK_NEW

	#workers	Original	DAST	DAST + Petició
Test 1	1	4736 ± 201 ns	3854 ± 168 ns	4166 ± 86 ns
(1, 25000)	12	11149 ± 442 ns	2714 ± 239 n	5816 ± 960 ns
Test 2	1	39547 ± 722 ns	34057 ± 597 ns	34332 ± 305 ns
(1, 25000)	12	4406 ± 166 ns	1290 ± 8 ns	2135 ± 111 ns
Matrix Multiply	1	2609 ± 29 ns	1998 ± 31 ns	2302 ± 16 ns
(512, 32)	12	11098 ± 590 ns	3052 ± 33 ns	3395 ± 17 ns

Taula 6.6: Temps d'execució del codi de la petició DAST_WORK_DONE

	#workers	Original	DAST	DAST + Petició
Test 1	1	335 ± 120 ns	952 ± 149 ns	1244 ± 76 ns
(1, 25000)	12	2815 ± 347 ns	2319 ± 329 ns	2596 ± 168 ns
Test 2	1	627 ± 477 ns	1041 ± 238 ns	1281 ± 121 ns
(1, 25000)	12	1361 ± 241 ns	786 ± 138 ns	1165 ± 70 ns
Matrix Multiply	1	449 ± 10 ns	1570 ± 82 ns	1910 ± 42 ns
(512, 32)	12	2246 ± 113 ns	2830 ± 145 ns	3143 ± 74 ns

Taula 6.7: Temps d'execució del codi de la petició DAST_WORK_DELETE

Les mètriques de les taules 6.5, 6.6 i 6.7 mostren la considerable reducció, similar a les de la resta de taules d'aquesta secció en execucions amb més d'un 'worker' del codi de les peticions. En el cas de la petició d'alliberament de la memòria, aquesta millora no és plausible perquè s'han de considerar els diferents costos d'accés a la memòria que varien en funció del 'core' on s'estigui executant i de si les posicions estan a la 'cache' d'aquest o no.

6.5 Anàlisi del rendiment a les aplicacions

Els diferents temps d'execució obtinguts per cada aplicació i entorn estan detallats a continuació agrupats per aplicació o programa. Cada programa conté els resultats de les execucions a Arvei i a Knigths.

6.5.1 Test 1

Els valors, o paràmetres d'execució, que canvien en les diferents gràfiques i/o traces d'aquesta aplicació són `PARALLEL_CREATIONS` i `TASK_SIZE`. Aquest s'indiquen o bé a la descripció inferior de cada figura o bé a la llegenda de cadascuna de les gràfiques en el mateix odre.

Execucions a Arvei

La possible saturació del gestor i per tant la limitació del paral·lelisme, tenint els 'workers' en estat IDLE (sense executar tasques d'usuari, codi 'útil'), és el fet que s'observa en la primera de les dues gràfiques de la figura 6.5. En aquesta s'aprecia que si les tasques són relativament petites (`TASK_SIZE` de 25000) en incrementar el nombre de 'workers' i per tant el volum de peticions que arriben per unitat de temps, arriba un punt on la velocitat d'inserció és superior a la de consum. Aquesta diferència suposa que fins que el DAST no satisfà les peticions de finalització de les tasques i per tant allibera les tasques dependents d'ella, aquestes no poden ser executades. Això manté l'speedup pràcticament constant sense importar el nombre de 'workers'. En canvi, en repetir l'execució amb una granularitat de tasca superior (`TASK_SIZE` de 50000, gràfica dreta

de la figura 6.5) el ritme de peticions és inferior i ja no apareix aquesta limitació, com també s'observa a la figura 6.6. En aquestes gràfiques, l'speedup del DAST amb bastants 'workers' (més de 15) és millor que l'original gracies a la menor pressió sobre el DAST i les estructures amb les dependències.

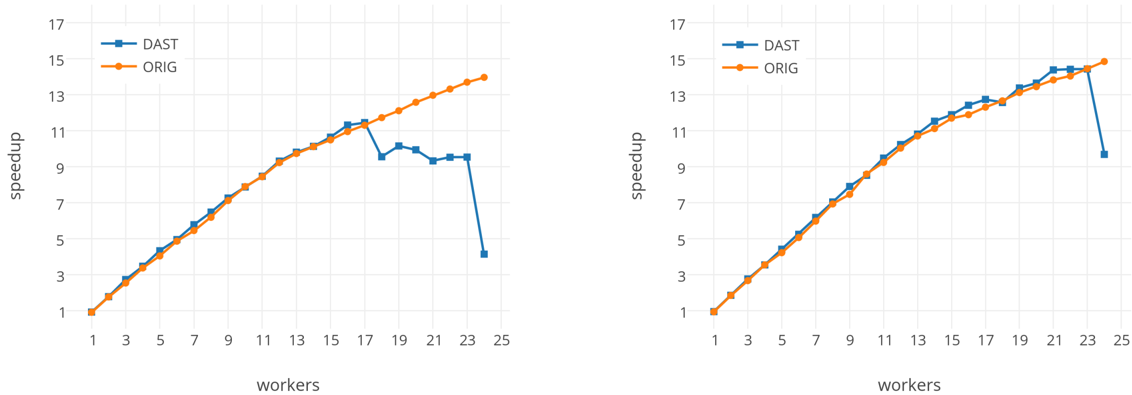


Figura 6.5: Speedup 'Test 1' a Arvei. `PARALLEL_CREATIONS`: 2. `TASK_SIZE`: 25000 (esq.) i 50000 (dta.)

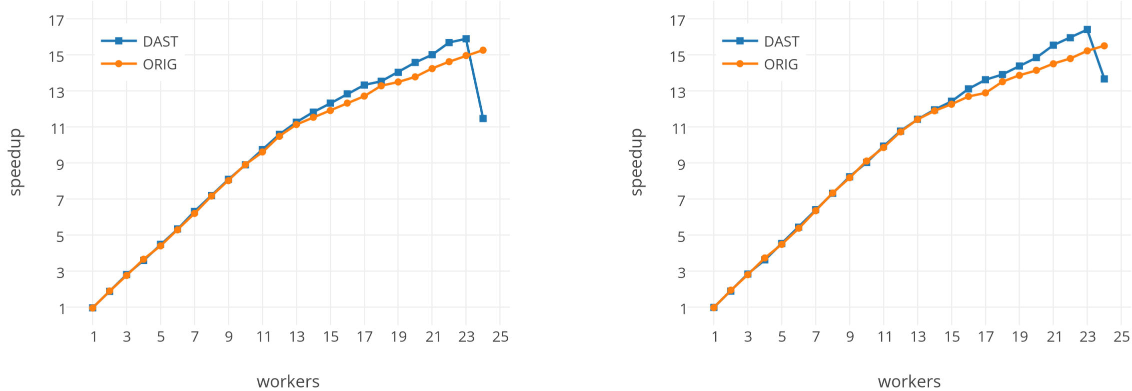


Figura 6.6: Speedup 'Test 1' a Arvei. `PARALLEL_CREATIONS`: 2. `TASK_SIZE`: 100000 (esq.) i 200000 (dta.)

Les implicacions que té el valor `PARALLEL_CREATIONS` s'observen a la figura 6.7. Com més tasques de primer nivell existeixin, el rendiment de la versió original millora lleugerament amb molts de 'workers'. Això és perquè la contenció per la finalització i creació de les tasques disminueix al tenir més grups de tasques independents entre ells que poden ser executats en paral·lel.

Les execucions amb altres valors per `TASK_SIZE` i `PARALLEL_CREATIONS` són coherents, pel que fa a nombre de tasques total creat i la dispersió de les mateixes en el temps d'execució del mateix, amb els resultats explicitats aquí.

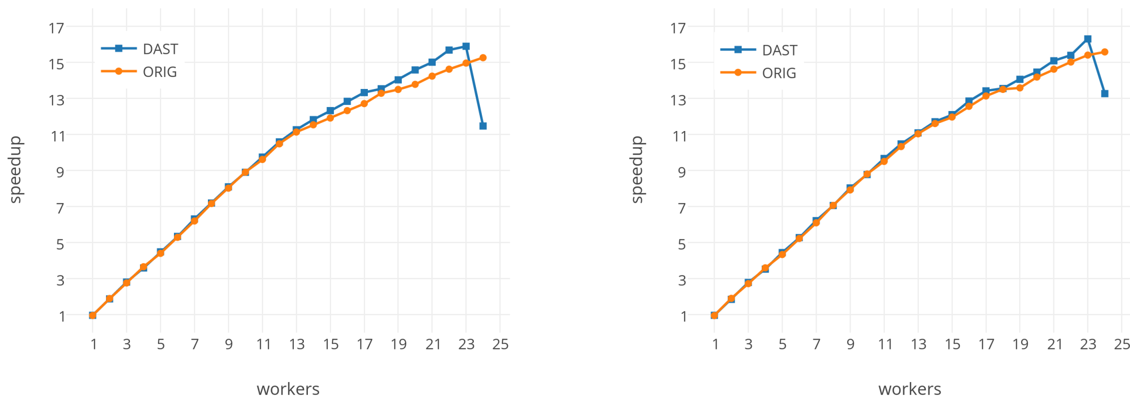


Figura 6.7: Speedup 'Test 1' a Arvei. `PARALLEL_CREATIONS`: 3 (esq.) i 4 (dta.). `TASK_SIZE`: 100000

Execucions a Knights

La saturació del DAST torna a ser un aspecte present en les execucions en aquest entorn. A les figures 6.8 i 6.9 s'observa que en augmentar el nombre de 'cores' utilitzats l'speedup no augmenta, en aquest entorn inclús disminueix, degut a la contenció en l'accés a certs recursos de l'arquitectura com la memòria. Aquesta limitació ve marcada per la duració de les tasques i/o el nombre de 'workers' en cada 'core', a més del nombre de `PARALLEL_CREATIONS` de l'execució.

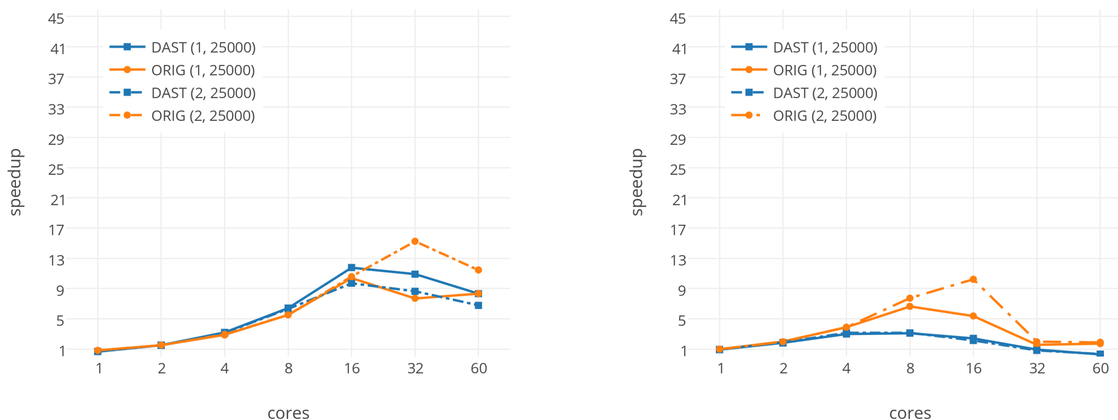


Figura 6.8: Speedup 'Test 1' a Knights. `TASK_SIZE`: 25000. Workers per core: 1 (esq.) i 4 (dta.)

Amb aquest programa el DAST suposa una gran limitació en el paral·lelisme. En la versió original de Nanos, cada conjunt de tasques està en un graf independent i per tant la seva creació és paral·lela. Al utilitzar el DAST aquesta creació s'està serialitzant i amb tasques de curta duració suposa una baixada del paral·lelisme. Per aquest motiu en les segones gràfiques de les

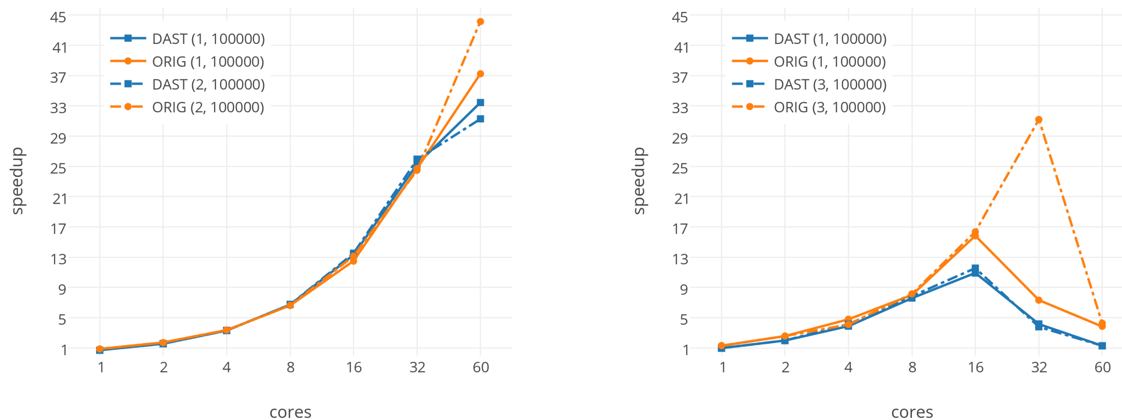


Figura 6.9: Speedup 'Test 1' a Knights. TASK_SIZE: 100000. Workers per core: 1 (esq.) i 4 (dta.)

figures 6.8 i 6.9, veiem com amb valors superiors a 1 de PARALLEL_CREATIONS la versió original aconsegueix speedups punta molt superiors.

6.5.2 Test 2

Els valors, o paràmetres d'execució, que canvien en les diferents gràfiques i/o traces d'aquesta aplicació són PARALLEL_CREATIONS i TASK_SIZE. Aquest s'indiquen o bé a la descripció inferior de cada figura o bé a la llegenda de cadascuna de les gràfiques en el mateix odre.

Execucions a Arvei

Una de les motivacions d'aquest projecte és reduir la contenció en l'accés a les estructures amb les dependències entre tasques mitjançant el model plantejat. A les gràfiques de la figura 6.10 s'observa com en la versió original de Nanos, arribat a un cert nombre de 'workers' l'speedup es manté constant i en canvi la versió de Nanos amb el DAST millora el rendiment considerablement. L'asímptota de la traça canvia en funció de la duració mitjana de les tasques (TASK_SIZE), en el primer cas està al voltant d'11 i en el segon 13. Considerant que la diferència entre la versió original i la nova està en el model d'accés i gestió de les dependències aquesta limitació del paral·lelisme que s'observa és atribuïble a l'accés a les mateixes.

Al seguir augmentant la mida mitjana de les tasques, casos de la figura 6.11, la diferència entre ambdues versions es va reduint incrementant l'speedup de la versió original. A diferència d'aquest, el 'runtime' amb el DAST no canvia significativament pel que fa a rendiment en incrementar el valor de TASK_SIZE. Un augment en el valor de PARALLEL_CREATIONS empitjora lleugerament el rendiment del Nanos original a diferència del nou model que manté resultats similars, com es veu a la figura 6.12.

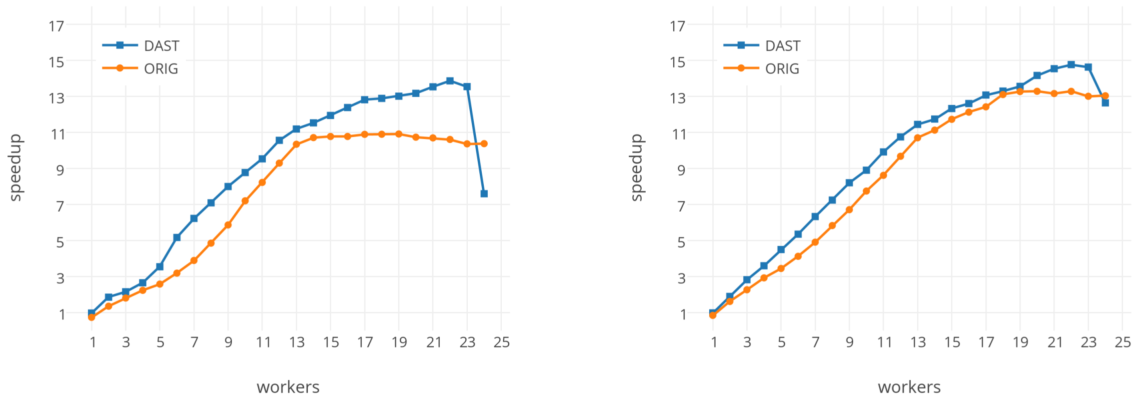


Figura 6.10: Speedup 'Test 2' a Arvei. PRALLEL_CREATIONS: 2. TASK_SIZE: 25000 (esq.) i 50000 (dta.)

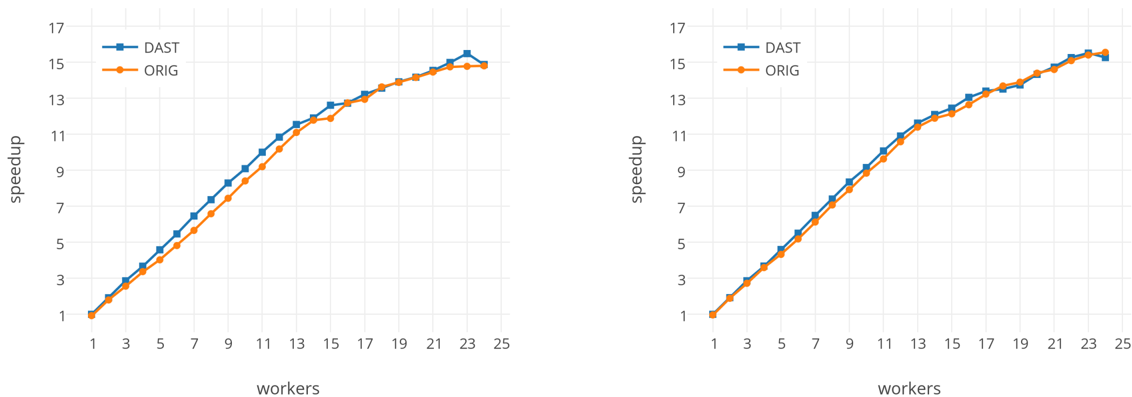


Figura 6.11: Speedup 'Test 2' a Arvei. PRALLEL_CREATIONS: 2. TASK_SIZE: 100000 (esq.) i 200000 (dta.)

En les execucions amb valors petits de TASK_SIZE, apreciem una diferència considerable entre els 'speedups' de les versions quan incrementem el PARALLEL_CREATIONS, variació que apareix a la figura 6.13. A més en el cas del DAST s'observa un comportament peculiar del model implementat: a major valor de PARALLEL_CREATIONS major és el nombre de 'workers' necessaris per tenir una millora en l'speedup del programa.

Aquest comportament és degut al sistema de cues i l'execució asíncrona de les peticions. El nombre de creacions paral·leles correspon al nombre de tasques creadores de la resta de tasques amb dependències que s'acaben executant posteriorment. La inclusió d'aquestes en el graf de dependències s'anirà efectuant en ordre en el DAST. Si el nombre de 'workers' és superior al valor de PARALLEL_CREATIONS, la resta de 'workers' aniran executant tasques internes paral·lelament a la creació tasques amb les dependències satisfetes i sol·licitant al DAST que n'alliberi les

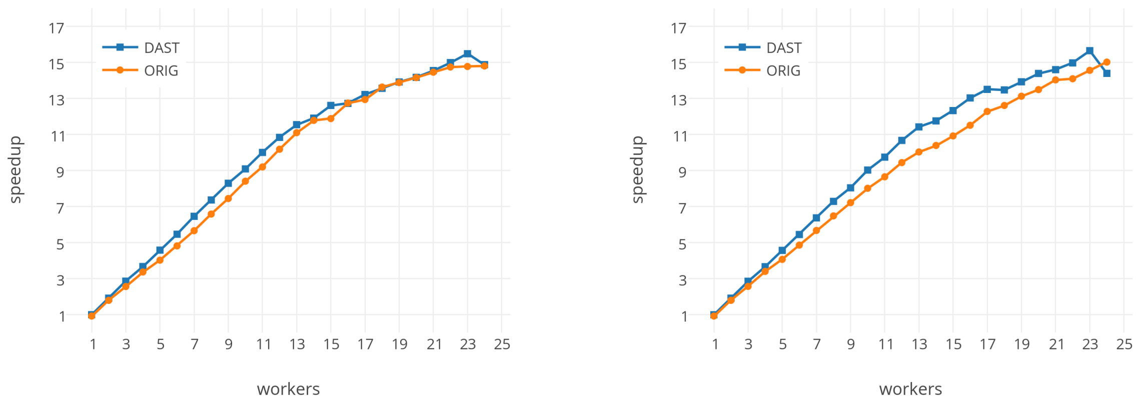


Figura 6.12: Speedup 'Test 2' a Arvei. PRALLEL_CREATIONS: 3 (esq.) i 4 (dta.). TASK_SIZE: 100000

seves successores. Les peticions de tots els 'workers', tant els que creen tasques com els que n'executen, es van satisfent en Round-Robin d'acord amb una de les millores proposades en el capítol d'implementació. En la versió original les dependències s'alliberen al mateix ritme que s'executen les tasques, independentment de la creació. En el cas on PARALLEL_CREATIONS val 4 i el nombre de 'workers' és 5, el DAST estaria dedicant únicament 1 de cada 5 accions a la finalització de tasques i la resta a la creació d'unes que no tenen per què ser executables (fins després de satisfer-se una petició de finalització). Aquesta relació fa que el graf de dependències sigui molt més gran del que seria en el 'runtime' original, fent més costos el càlcul de dependències. El mateix succeeix quan totes les peticions de creació s'han generat i tots els 'workers' executen tasques, fins que totes les tasques no s'hagin creat no s'alliberaran els successors de moltes tasques executades i per tant torna a aparèixer l'augment de complexitat en el graf. Tot això porta a un desaprofitament dels recursos, incrementar el nombre de recursos no suposa una millora, i una clara limitació de paral·lelisme com s'observa a la gràfica de la figura 6.13. En aquesta també s'aprecia que quan el nombre de 'workers' és superior al doble de creacions paral·leles l'speedup augmenta ràpidament.

El comportament del DAST descrit que limita el paral·lelisme final del programa és també apreciable la traça de la figura 6.14 on apareix el nombre de tasques llestes per ser executades (eix Y) al llarg del temps d'execució de l'aplicació (eix X). En el cas de Nanos original (traça superior) el nombre va oscil·lant i arriba fins a pics de 12 WorkDescriptors i pràcticament en tot el transcurs del programa no és inferior a 2-4, per contra en el nou Nanos (traça inferior) el patró és incrementar en un WD i immediatament decrementar-lo (en la imatge pot aparentar una línia constant a un). En aquesta figura les escales temporals de les dues traces són diferents i cadascuna agafa el període amb paral·lelisme de l'execució; l'eix Y si que està a la mateix escala.

També s'aprecia a la figura 6.15 on els 'workers' (eix Y) del Nanos amb DAST (traça inferior) estan majoritàriament en color blau que indica estat (IDLE) durant el temps d'execució (eix X), menys els que estan de color rosa que són les tasques que estan creant més tasques. Les regions de color marró són l'execució de les tasques del segon nivell, les quals estan molt més disperses

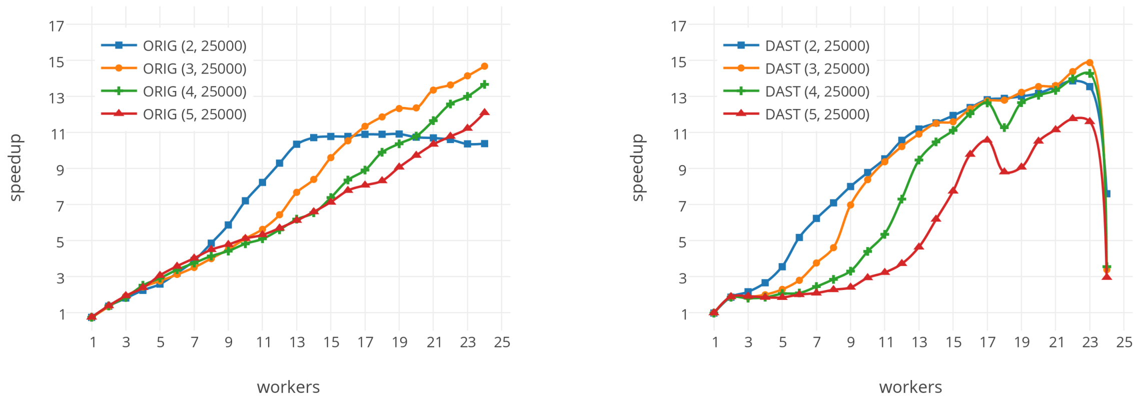


Figura 6.13: Speedup 'Test 2' a Arvei. Comparativa diferents `PARALLEL_CREATIONS` (esq.: original, dta.: DAST)

en el nou 'runtime' que en l'original. Això és perquè els 'workers' estan executant al mateix ritme que les tasques es creen o s'alliberen per a la seva execució en el DAST. A diferencia de la versió original (traça superior) que té una execució molt més compacta.

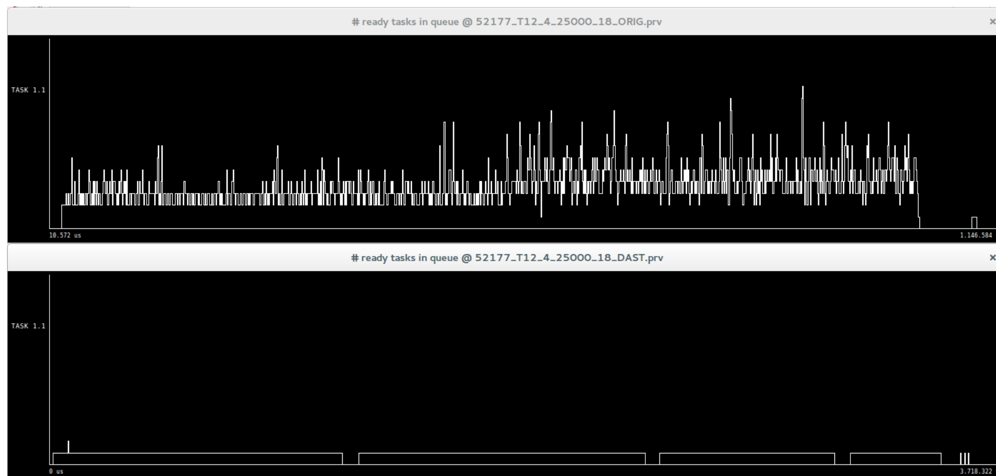


Figura 6.14: Traces del 'Test 2' a Arvei amb el nombre de 'ready tasks' (dalt: original, baix: DAST)

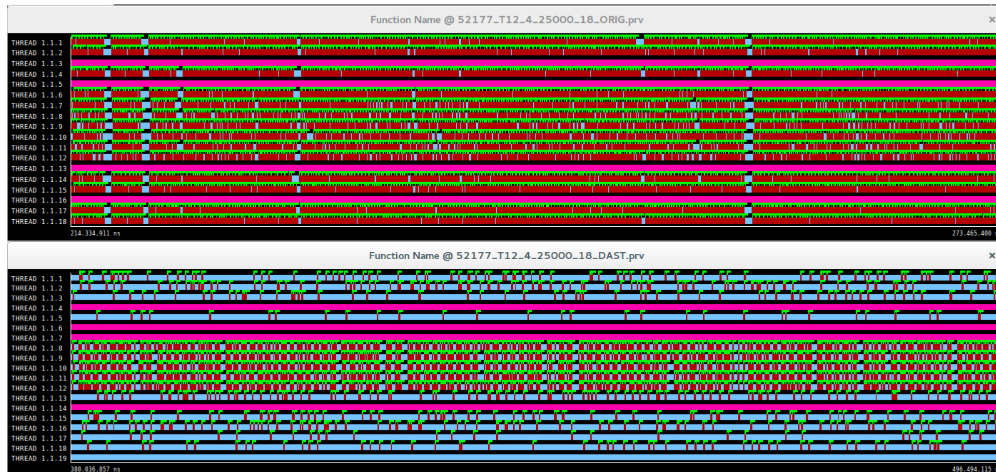


Figura 6.15: Traces del 'Test 2' a Arvei amb les tasques en execució (dalt: original, baix: DAST)

Execucions a Knights

Aquest programa mostra uns resultats molt similars en aquest entorn als d'Arvei com apareix a les figures 6.16 i 6.17. Amb un 'worker' per cada nucli i amb un `PARALLEL_CREATIONS` de 25000, el 'runtime' amb el DAST obté uns 'speedups' superiors als de la versió original; excepte pel cas amb 60 cores, on la densitat més gran de peticions suposa que l'original sigui millor. Amb 4 'workers' per cada 'core', el rendiment amb molts de nuclis és molt pobre a causa de la gran contenció que es genera a les estructures del 'runtime' i als recursos del sistema, com la memòria, etc. En augmentar la mida mitjana de les tasques els resultats canvien lleugerament. Amb un 'worker' per 'core', el 'runtime' original millora els seus resultats considerablement igualant, i superant en alguns punts, el nou Nanos que té un rendiment global similar. Amb 4 'workers' per 'core', el que té uns resultats similars als de la figura 6.16 és l'original i, en canvi, el 'runtime' del DAST millora perquè està menys saturat i augmenta el paral·lelisme, en una certa regió.

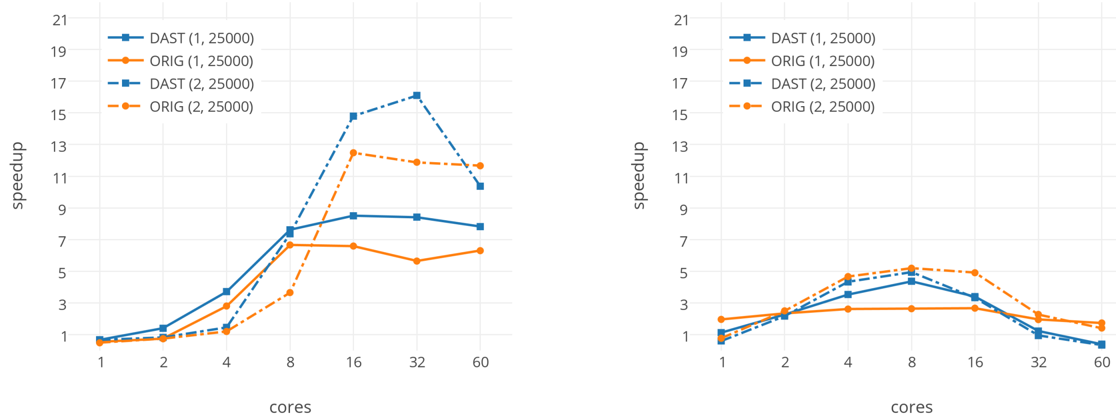


Figura 6.16: Speedup 'Test 2' a Knights. `TASK_SIZE`: 25000. Workers per core: 1 (esq.) i 4 (dta.)

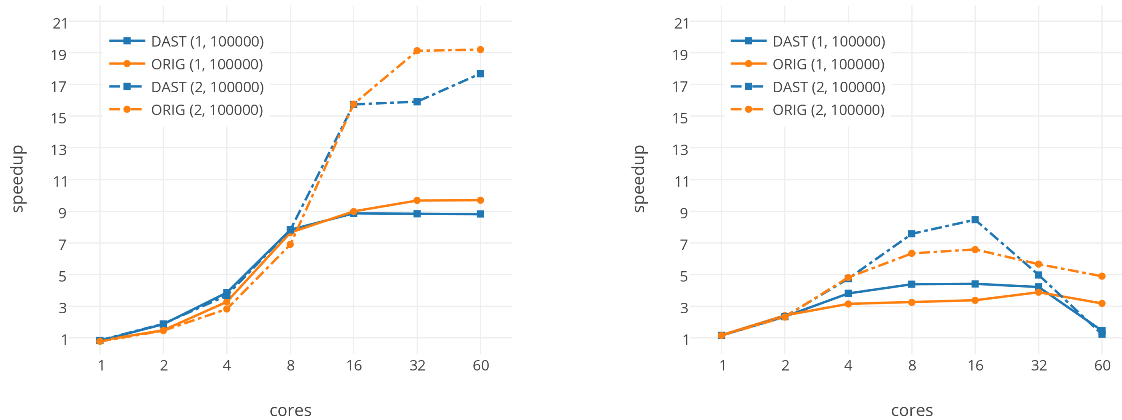


Figura 6.17: Speedup 'Test 2' a Knights. TASK_SIZE: 100000. Workers per core: 1 (esq.) i 4 (dta.)

6.5.3 Cholesky

Els valors, o paràmetres d'execució, que canvien en les diferents gràfiques i/o traces d'aquesta aplicació són MATRIX_SIZE i BLOCK_SIZE. Aquest s'indiquen o bé a la descripció inferior de cada figura o bé a la llegenda de cadascuna de les gràfiques en el mateix odre.

Execucions a Arvei

Com es veu a la figura 6.18 el rendiment dels dos 'runtimes' en aquesta aplicació és relativament similar, l'evolució i els valors en augmentar el nombre de 'workers' és pròxima. Amb alguns valors aquesta diferència és significativament superior com s'observa a la primera de les dues gràfiques, tot i això l'evolució global és anàloga. El la segona s'aprecia que amb més d'11-12 'workers', quan es comença a utilitzar el segon fil d'execució de cada 'core', en rendiment baixa perquè s'estan compartint recursos com la 'cache L2' la qual genera un desaprofitament de la localitat de dades i fa improductiu l'increment de 'workers'.

Amb altres valors de MATRIX_SIZE l'evolució de l'speedup és el mateixa que el de la figura 6.18 d'acord amb el nombre de tasques creades en l'execució. En aquests casos l'aplicació és lleugerament més lenta amb la nova versió de Nanos (figura 6.19) un cop arribat al màxim rendiment pel cost de comunicar els 'workers' i el DAST.

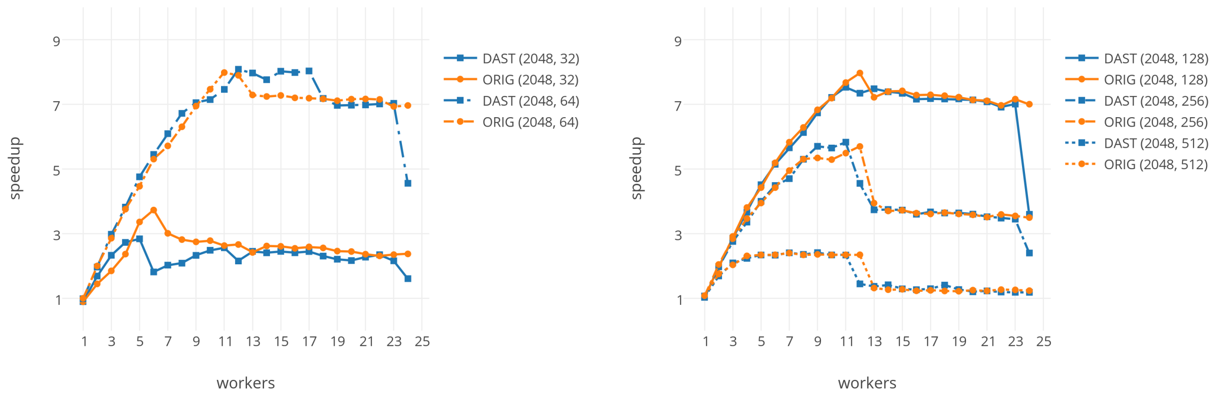


Figura 6.18: Speedup 'Cholesky' a Arvei amb diferents BLOCK_SIZE. MATRIX_SIZE: 2048

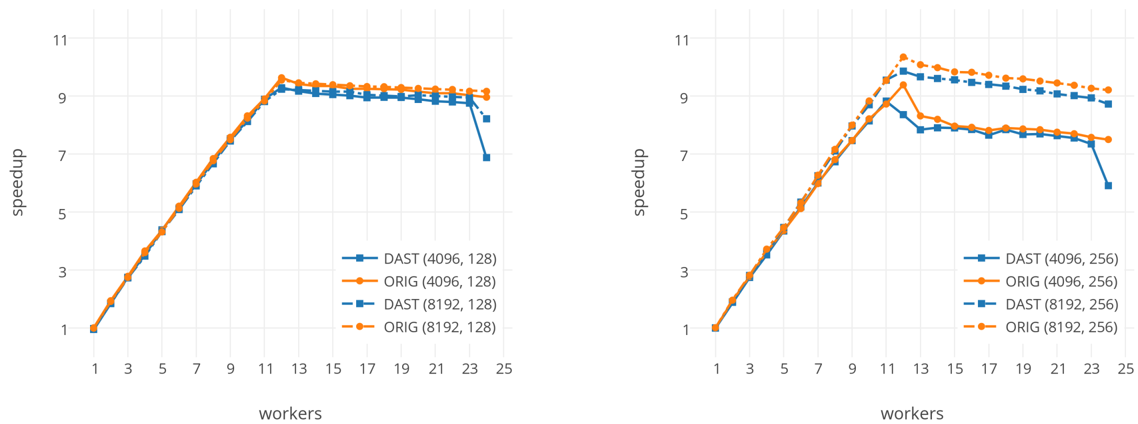
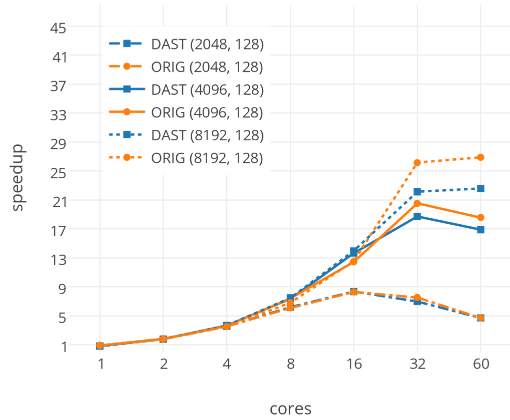


Figura 6.19: Speedup 'Cholesky' a Arvei amb diferents MATRIX_SIZE i BLOCK_SIZE

Execucions a Knights

Les execucions d'aquesta aplicació a Knights han donat uns resultats de rendiment molt similars en ambdues versions com es veu a les figures 6.20, 6.21 i 6.22. Les majors diferencies entre els dos 'runtimes' estan a la figura 6.20 amb la menor granularitat de les tasques (BLOCK_SIZE de 128). A aquesta figura i amb un 'worker' per nucli, apareix el mateix patró que en la primera gràfica de la figura 6.19 on arribat un punt, i existint encara més paral·lelisme per explotar, la millora ve limitada per l'accés als recursos de la màquina, mantenint l'speedup constant o minvant lleugerament i amb una lleugera diferència deguda al sobrecost de comunicació 'workers'-DAST. A la mateixa figura i amb quatre 'workers' per nucli la problemàtica és la mateixa però més plausible gràcies al gran nombre de fils d'execució existents.



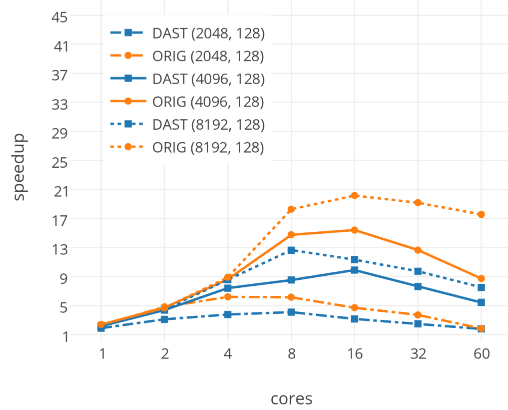
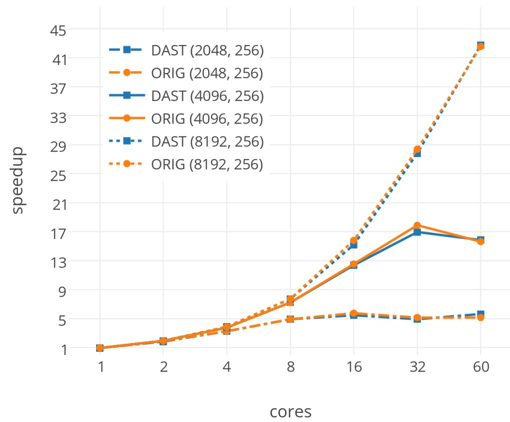


Figura 6.20: Speedup 'Cholesky' a Knights. `BLOCK_SIZE`: 128. Workers per core: 1 (esq.) i 4 (dta.)



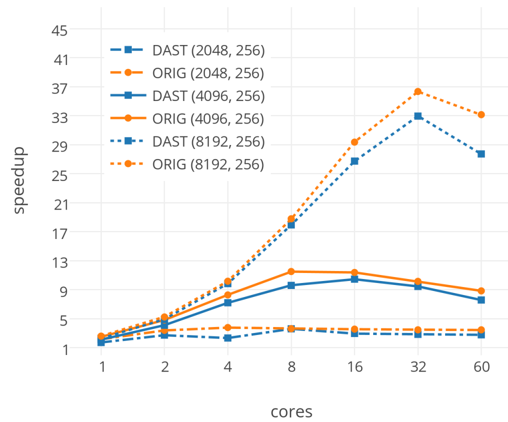


Figura 6.21: Speedup 'Cholesky' a Knights. `BLOCK_SIZE`: 256. Workers per core: 1 (esq.) i 4 (dta.)

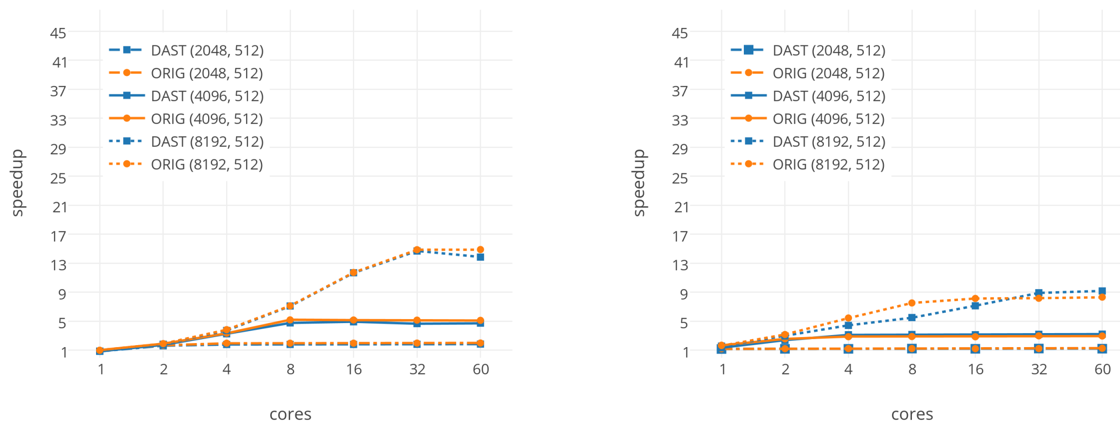


Figura 6.22: Speedup 'Cholesky' a Knights. BLOCK_SIZE: 512. Workers per core: 1 (esq.) i 4 (dta.)

6.5.4 Matrix Multiply

Els valors, o paràmetres d'execució, que canvien en les diferents gràfiques i/o traces d'aquesta aplicació són MATRIX_SIZE i BLOCK_SIZE. Aquest s'indiquen o bé a la descripció inferior de cada figura o bé a la llegenda de cadascuna de les gràfiques en el mateix odre.

Execucions a Arvei

Una de les consideracions de les execucions en Arvei resultava del 'mapeig' del DAST en un dels 'cores' del node d'execució. Les execucions d'aquest programa mostren com varia l'speedup de l'aplicació en funció d'aquest fet.

A la figura 6.23 i en ambdues gràfiques s'observen 4 punts significatius sobre el 'mapeig' del DAST. Això és gràcies a la petita granularitat de les tasques que suposa una dependència dels 'workers' del gestor però sense implicar una saturació total del mateix. Els quatre punts són els següents:

- 6 'workers'. En aquest punt el DAST passa a 'mapejar-se' en el segon processador del node a diferència del 'master worker', qui usualment executa la tasca principal del programa. La versió original té un pendent constant mentre que la nova versió té un pendent menor a partir d'aquest punt. Això és degut al cost més gran d'accés a la memòria de l'altre processador que a la del mateix.
- 12 'workers'. En aquest punt el DAST passa a 'mapejar-se' en el mateix processador que el 'master worker' però compartint recursos amb un altre fil d'execució. L'speedup en aquest punt repunta considerablement i millora el valor de la versió original tot i l'empitjorament observat en altres programes.

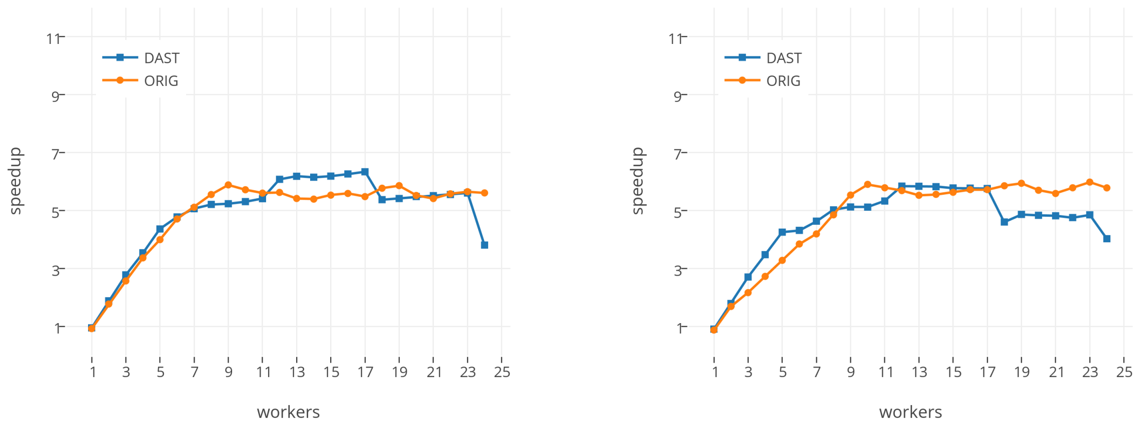


Figura 6.23: Speedup 'Matrix Multiply' a Arvei. BLOCK_SIZE: 64. MATRIX_SIZE: 2048 (esq.) i 4096 (dta.)

- 18 'workers'. En aquest punt el DAST torna a 'mapejar-se' en el segon processador i segueix compartint recursos amb un altre 'worker'. Aquest canvi suposa una abrupta baixada del rendiment.
- 24 'workers'. En aquest punt el DAST és 'mapeja' en el primer processador però executant-se concurrentment amb un altre 'worker'. Aquesta compartició del temps d'execució entre els dos provoca una baixada de l'speedup.

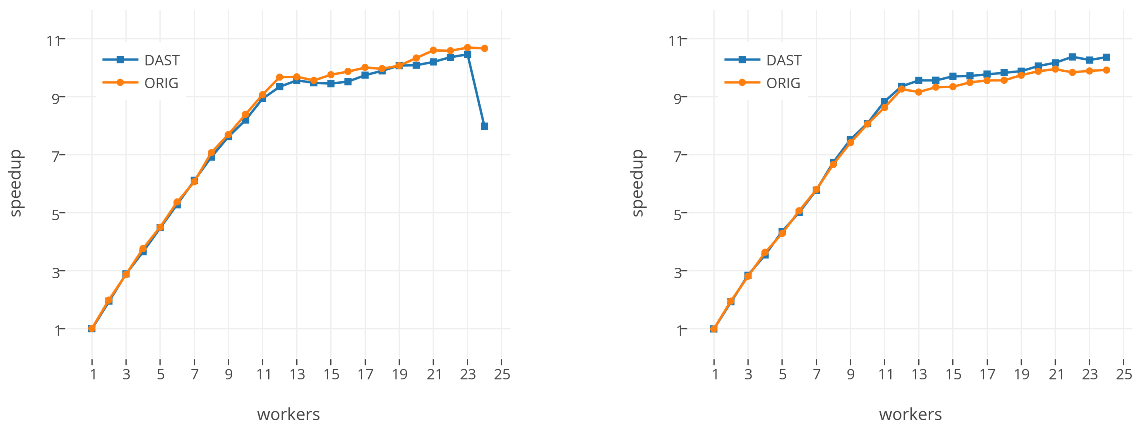


Figura 6.24: Speedup 'Matrix Multiply' a Arvei. BLOCK_SIZE: 128. MATRIX_SIZE: 2048 (esq.) i 4096 (dta.)

En canvi, en augmentar el BLOCK_SIZE i com a conseqüència la duració de cada tasca, aquests punts ja no s'aprecien a les gràfiques de les figures 6.24 i 6.25. En augmentar aquest valor, s'està reduint el nombre de peticions que el DAST ha de satisfer i per tant dispersar-les en el temps

d'execució facilitant la ràpida satisfacció. En aquests casos, el principal aspecte que limita la reducció el temps d'execució és l'accés a les dades a tractar i les dues versions obtenen resultats molt pròxims.

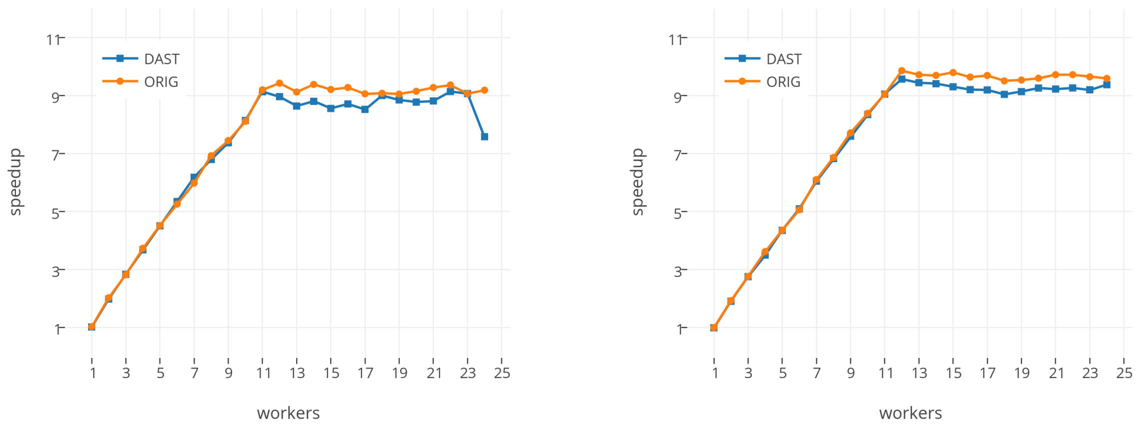


Figura 6.25: Speedup 'Matrix Multiply' a Arvei. BLOCK_SIZE: 256. MATRIX_SIZE: 2048 (esq.) i 4096 (dta.)

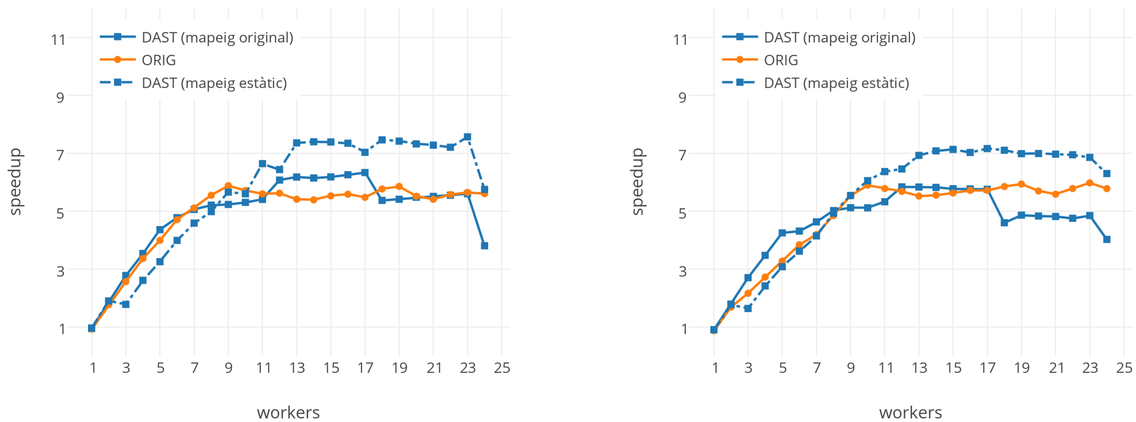


Figura 6.26: Speedup 'Matrix Multiply' a Arvei. Comparativa mapeig DAST. BLOCK_SIZE: 64. MATRIX_SIZE: 2048 (esq.) i 4096 (dta.)

Per comprovar les observacions efectuades a les execucions amb BLOCK_SIZE 64 es van repetir les mateixes forçant el 'mapeig' del DAST en un 'core' del primer processador, a l'igual que el 'master worker', i sense utilitzar el segon fil d'execució d'aquest. Els resultats d'aquesta prova (figura 6.26) van confirmar les observacions perquè mantenen la mateixa pendent de creixement fins al voltant dels 12 'workers', no s'observa un repunt en aquest valor i no abaixa l'speedup amb 18 'workers'. En el cas de 24 'workers', amb els 'mappings' forçats no s'evita la concurrència d'aquest punt i, per tant, la baixada en aquest punt és normal. A més, com també és veu a la figura 6.26, la millora del DAST amb 'mapeig' estàtic sobre la versió original és considerablement

superior; al voltant de 1.4x i 1.17x amb més de 13 'workers' per un MATRIX_SIZE de 2048 i 4096, respectivament.

Execucions a Knights

Les execucions d'aquesta aplicació a Knights han donat uns resultats de rendiment diferents segons la granularitat de les tasques utilitzada en les execucions. La llegenda de cada traça existent a les figures d'aquesta secció indica els paràmetres d'execució MATRIX_SIZE i BLOCK_SIZE, en aquest ordre, a més de la versió de Nanos a la qual correspon.

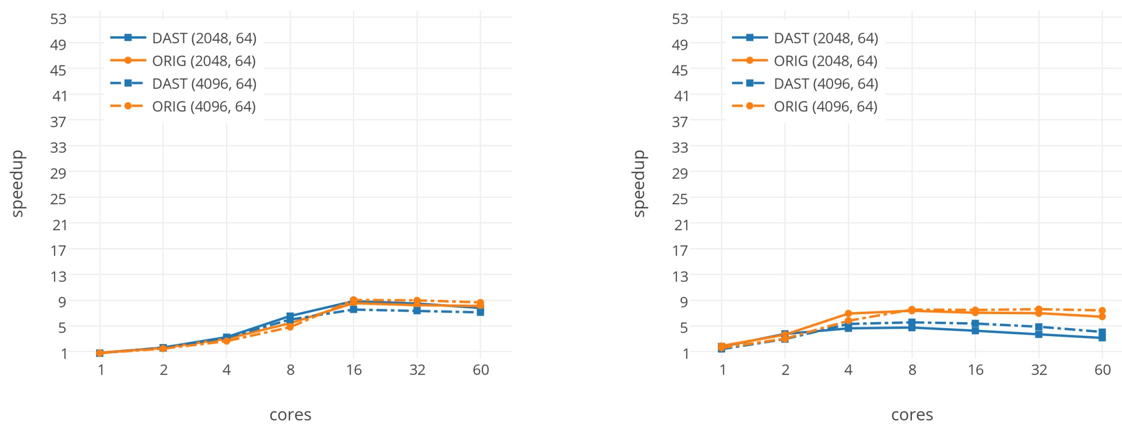


Figura 6.27: Speedup 'Matrix Multiply' a Knights. BLOCK_SIZE: 64. Workers per core: 1 (esq.) i 4 (dta.)

La granularitat BLOCK_SIZE igual a 64 (figura 6.27) mostra uns 'speedup' molt pobres tot i el gran paral·lelisme existent perquè les tasques tenen una execució molt breu i l'overhead de creació està limitant aquest paral·lelisme. Amb un 'worker' per nucli els resultats són similars i més distants amb quatre per nucli a causa de la contenció de l'arquitectura i la saturació del gestor.

La granularitat BLOCK_SIZE igual a 128 (figura 6.28) mostra una millora respecte a la de 64. També s'observa com amb 32 'workers' (32 i 8 'cores', segons la gràfica i per tant el nombre de 'workers' per nucli) el DAST s'estanca i comença a limitar el paral·lelisme a diferència de la versió original que segueix augmentant. La versió original també mostra un punt de saturació per culpa de la falta de paral·lelisme que manté l'speedup constant.

La granularitat BLOCK_SIZE igual a 256 (figura 6.29) mostra uns resultats molt similars per la gràfica amb un 'worker' per cada nucli i més distants en el cas de quatre 'workers' per nucli. Aquesta diferència és el resultat de la contenció que apareix en tenir grans quantitats de fils d'execució i de la no existència de suficient paral·lelisme a l'aplicació.

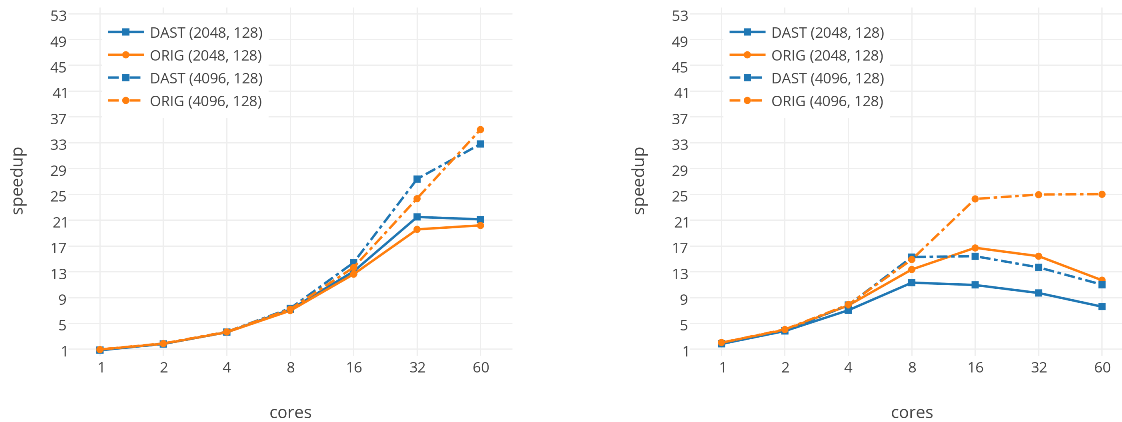


Figura 6.28: Speedup 'Matrix Multiply' a Knights. BLOCK_SIZE: 128. Workers per core: 1 (esq.) i 4 (dta.)

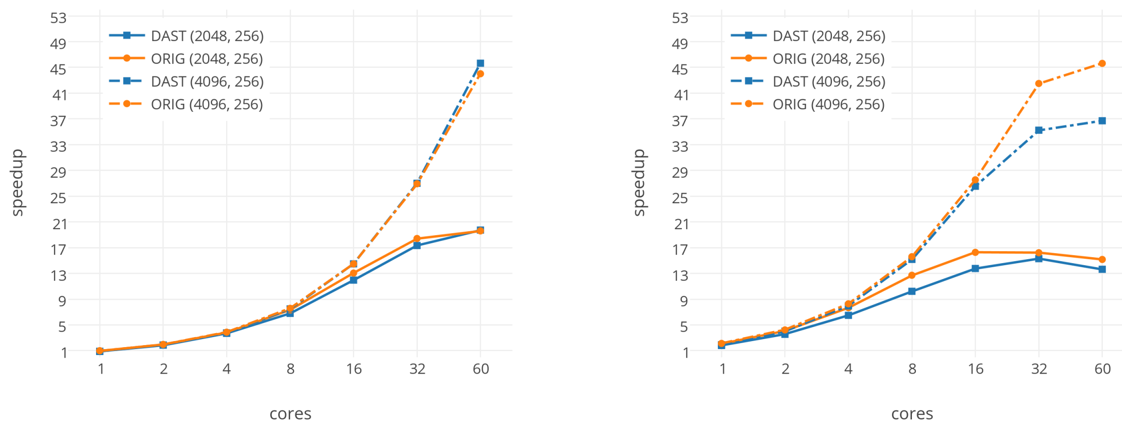


Figura 6.29: Speedup 'Matrix Multiply' a Knights. BLOCK_SIZE: 256. Workers per core: 1 (esq.) i 4 (dta.)

6.5.5 Sparse LU

Els valors, o paràmetres d'execució, que canvien en les diferents gràfiques i/o traces d'aquesta aplicació són NB i BSIZE. Aquest s'indiquen o bé a la descripció inferior de cada figura o bé a la llegenda de cadascuna de les gràfiques en el mateix odre.

Execucions a Arvei

Les execucions d'aquest programa (figura 6.30) de prova han resultat molt similars a les prèvies. La majoria d'elles tenen uns resultats similars pels dos 'runtimes'. Únicament s'aprecia

una major diferència en les execucions amb NB 16 i BSIZE 50 on el DAST amb més d'11 'workers' comença a saturar-se i tenir un comportament més variable.

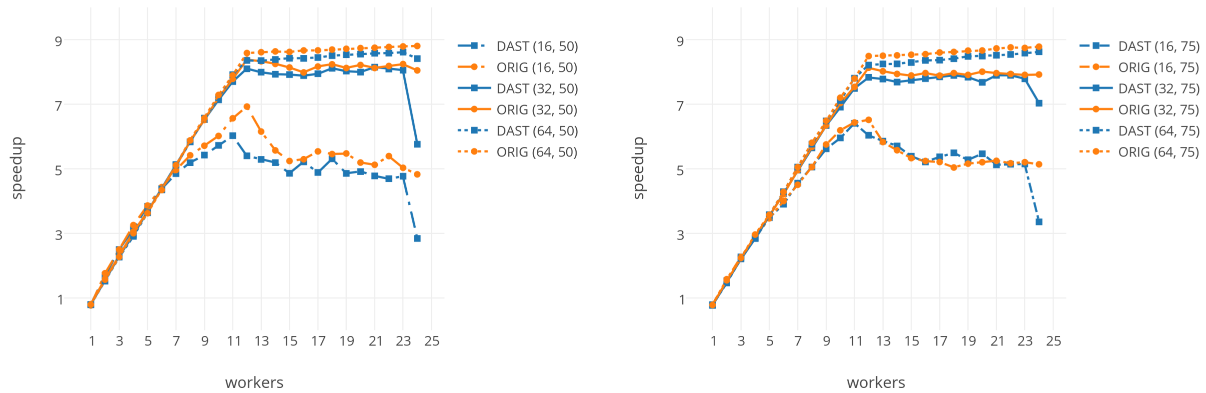


Figura 6.30: Speedup 'Sparse LU' a Arvei amb diferents paràmetres

Execucions a Knights

Les execucions d'aquesta aplicació a Knights han donat uns resultats de rendiment diferents segons el nombre de 'workers' a cada nucli d'execució. Les primeres gràfiques de les figures 6.31 i 6.32 presenten un rendiment molt pròxim entre els dos 'runtimes'. En els casos de quatre 'workers' a cada nucli, l'speedup està limitat per la contenció d'accés als recursos de l'arquitectura i el paral·lelisme de l'aplicació perquè en augmentar el valor de NB també ho fa l'speedup mostrant així la possibilitat d'aprofitar millor els recursos.

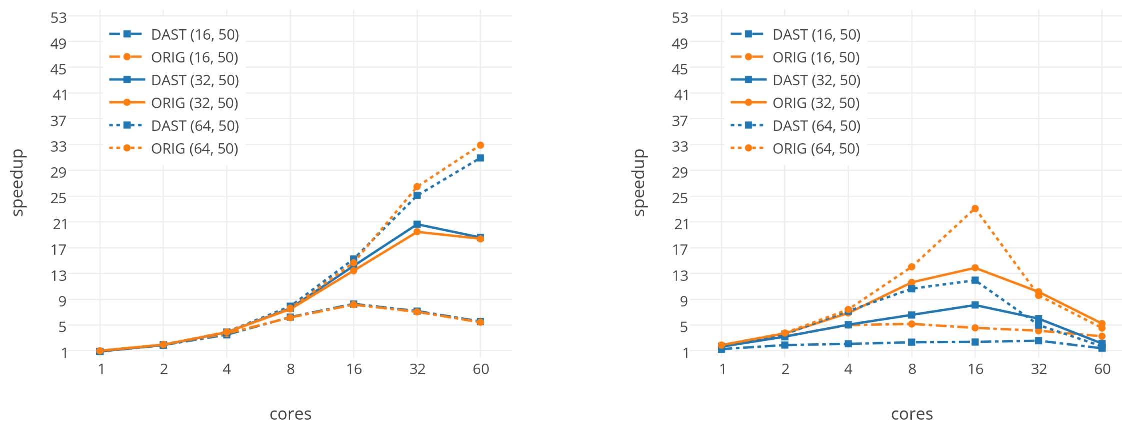
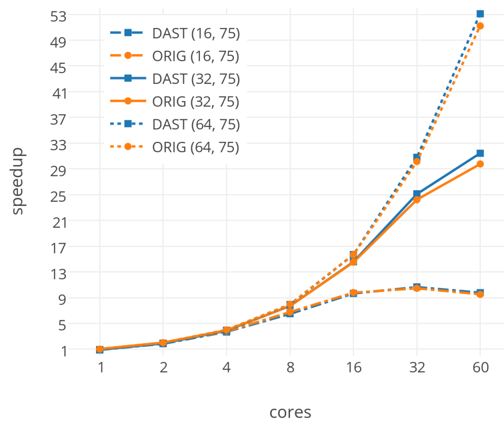


Figura 6.31: Speedup 'Sparse LU' a Knights. BSIZE: 50. Workers per core: 1 (esq.) i 4 (dta.)



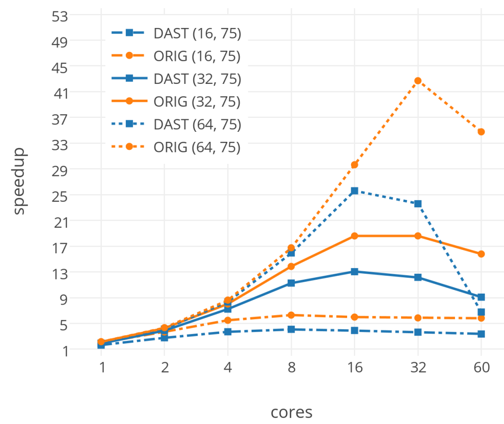


Figura 6.32: Speedup 'Sparse LU' a Knights. `BSIZE`: 75. Workers per core: 1 (esq.) i 4 (dta.)

Capítol 7

Conclusions

Aquest capítol presenta les conclusions extretes del projecte des del punt de vista dels resultats que s'han obtingut, les limitacions observades i es discuteix el possible treball futur a realitzar sobre aquest projecte.

7.1 Resultats

El nou model de gestió de dependències entre tasques plantejat en aquest projecte és una prova de concepte per comprovar-ne la viabilitat i el seu rendiment. S'ha dissenyat un sistema d'accés centralitzat al graf en un únic 'thread' i dissenyat un sistema de peticions relacionades amb aquest. Això permet eliminar la contenció d'accés a aquesta estructura, la qual és present en la versió original on l'accés es basa en regions d'exclusió mútua entre els 'workers'.

La implementació del nou model s'ha efectuat sobre la versió original de Nanos++, efectuant-hi els canvis necessaris per aconseguir el correcte funcionament del model. Amb aquesta implementació s'ha comprovat la possibilitat d'utilitzar el model teòric en un entorn real.

L'execució de les diferents aplicacions de prova en els dos entorns 'hardware' utilitzats ha permès comprovar el correcte funcionament i la millora en certes execucions del temps d'execució. Com apareix en el capítol 6, el nou model amb un nombre considerable de 'workers' triga en mitjana el mateix temps a executar-se, o similar, que la versió original. En els millors casos el model centralitzat arriba a igualar el rendiment de l'original amb dos 'workers' més. En alguns casos no hi ha millora o s'empitjora el temps, això contrasta amb la reducció general observada en el temps d'execució del codi de les peticions (secció 6.4). Aquest fet és atribuïble al cost que implica la comunicació dels 'workers' amb el gestor centralitzat i el temps que triga aquest a començar a satisfer la petició, a més de les limitacions del model.

Les execucions efectuades en el coprocessador del clúster Knights han mostrat que majoritàriament és millor tenir 1 fil de processament per 'core' que no 4 d'aquests, això vol dir que les execucions són més ràpides en el primer cas. Aquest fet s'observa en les dues versions del 'runtime', fent que els resultats més significatius d'ambdós siguin els que utilitzen menys recursos.

Aquests resultats són molt millors pel DAST que els que utilitzen més d'un fil per 'core'.

Aquest projecte, el model proposat i els resultats del mateix, serveixen com a pont per la integració del model de programació OmpSs en un 'hardware' accelerador per a la gestió de tasques. Aquest 'hardware' específic és el principal objectiu del projecte RoMol: crear una arquitectura paral·lela dissenyada des de la perspectiva d'un 'runtime' com Nanos [27].

En general, el projecte ha assolit l'objectiu del mateix, descrit en el capítol 1, dintre de l'abast definit. S'han disseny i implementat el model per poder comprovar-ne el comportament en un entorn real. Respecte a la part d'anàlisi del rendiment s'han ampliat les proves inicialment plantejades, ja que s'han portat a terme en dos entorns 'hardware' quan només estaven previstes en un.

L'assoliment dels objectius no hagués estat possible sense els diversos coneixements adquirits al llarg del grau sobre programació, programació paral·lela, automatització de tasques, i sobre tot gràcies a la capacitat de raonament i enfrontament dels problemes adquirida al llarg de moltes assignatures. Sense totes aquestes capacitats la finalització del projecte hagués sigut impossible d'aconseguir al no poder enfrontar les adversitats que apareixen en aquest tipus de projectes.

7.2 Limitacions

El nou model centralitzat té implícites diverses limitacions com s'ha comprovat en el capítol 6. L'arrel d'aquestes és la capacitat limitada del gestor per satisfer les peticions que va rebent dels 'workers'. El nombre de peticions que pot acceptar per unitat de temps depèn del programa en execució perquè cadascun d'ells té tasques diferents (nombre de dependències, relacions entre elles, volum de tasques, etc.) que fan variar el temps necessari per satisfer-les.

Les conseqüències d'aquest problema i, per tant, les limitacions en les execucions són:

- 'Workers' en estat IDLE.
- Limitació del paral·lelisme.
- Serialització de parts paral·leles.

7.3 Treball futur

L'avanç del projecte ha plantejat diferents qüestions que quedaven fora de l'abast del projecte i que té sentit considerar-les com a treball futur, a més d'altres millores que poden efectuar-se en el sistema.

El principal aspecte a investigar que permet una millora directe respecte al rendiment és el 'mapeig' del DAST. Com s'ha observat en algunes execucions del capítol 6, segons el 'core' on estigui el DAST el temps d'execució varia. El problema és que aquest 'mapeig' és difícil de

controlar perquè la informació disponible en crear-lo és limitada, s'hauria d'analitzar si es pot automatitzar d'alguna forma o s'ha d'obtenir informació de l'usuari en iniciar les execucions.

Altres aspectes a investigar són el comportament del model en un entorn menys controlat. Nanos té moltes opcions que permeten a l'usuari, en iniciar l'execució del programa, definir certs aspectes del comportament en aquella execució com les 'throttle policies', 'scheduling policies', 'dependence managers', etc. Les proves efectuades en aquest projecte han estat sempre amb les mateixes condicions, les quals podrien no ser valides per l'execució d'algunes aplicacions.

Una evolució possible del model seria la possibilitat d'utilitzar més d'un 'thread' per executar les funcions del DAST. Aquest canvi tindria sentit en entorns amb molts 'cores' com el cas del Knights i permetria que el DAST deixes de ser el coll d'ampolla que limita les execucions. Tot i això, s'hauria d'estudiar amb cura la distribució del treball entre els 'threads' per evitar tornar a introduir contenció en aquesta part.

Més enllà de la versió 'software' del model, la integració d'aquest model en una versió 'hardware', com el que proposa en projecte RoMol, aconseguiria efectuar les diferents operacions del gestor en paral·lel i molt més ràpidament. Aquesta línia de treball permetria resoldre les limitacions de rendiment observades en la versió 'software'.

Bibliografia

[1] BSC Programming Models Group. Ompss user guide. Març 2015.

[2] Xavier Martorell Rosa M. Badia. Pumps 2014 - programming with ompss. 2014.

[3] Mic hardware and software architecture. In *High-Performance Computing on the Intel® Xeon Phi™*. 2014. ISBN 978-3-319-06485-7. doi: 10.1007/978-3-319-06486-4_2.

[4] Gordon E. Moore. Cramming more components onto integrated circuits. 1965.

[5] Wikipedia. Dennard scaling - wikipedia, the free encyclopedia, 2015. URL `http://en.wikipedia.org/w/index.php?title=Dennard_scaling&oldid=642400904month=`.

[6] J. Darlington, M. Ghanem, and H. W. To. Structured parallel programming. In *In Programming Models for Massively Parallel Computers*, pages 160–169. IEEE Computer Society Press, 1993.

[7] Productive cluster programming with ompss. In Emmanuel Jeannot, Raymond Namyst, and Jean Roman, editors, *Euro-Par 2011 Parallel Processing*, volume 6852 of *Lecture Notes in Computer Science*. 2011. ISBN 978-3-642-23399-9. doi: 10.1007/978-3-642-23400-2_52.

[8] BSC Programming Models Group. The ompss programming model, Març 2015. URL `http://pm.bsc.es/ompss`.

[9] Extending the openmp tasking model to allow dependent tasks. In Rudolf Eigenmann and BronisR. de Supinski, editors, *OpenMP in a New Era of Parallelism*, volume 5004 of *Lecture Notes in Computer Science*. 2008. ISBN 978-3-540-79560-5. doi: 10.1007/978-3-540-79561-2_10.

[10] Eduard Ayguadé. Implementing the openmp programming model. 2015.

[11] BSC Programming Models Group. Mercurium, Març 2015. URL `http://pm.bsc.es/mcxx`.

[12] BSC Programming Models Group. Nanos++, Març 2015. URL `http://pm.bsc.es/nanox`.

[13] Thomas Bolstad Martinsen. Energy efficient task pool scheduler in ompss. 2013.

[14] Marcos Maroñas. Extending ompss to support dynamic programming. 2015.

[15] Antonio Filgueras. Ompsscl: entorno para la programación automática de aplicaciones opencl. 2012.

[16] Vicerectorat de Recerca i Innovació. Normativa sobre els drets de propietat industrial i intel·lectual a la upc. 2008.

[17] Serveis TIC del DAC. Altas prestaciones, 2014. URL `https://www.ac.upc.edu/serveis-tic/altas-prestaciones`. Accedit: 15/03/2015.

[18] ASUS. U36sd, 2011. URL `http://www.asus.com/es/Notebooks_Ultrabooks/U36SD/specifications/`. Accedit: 15/03/2015.

[19] On the instrumentation of openmp and ompss tasking constructs. In Ioannis Caragiannis, Michael Alexander, RosaMaria Badia, Mario Cannataro, Alexandru Costan, Marco Danelutto, Frédéric Desprez, Bettina Krammer, Julio Sahuquillo, StephenL. Scott, and Josef Weidendorfer, editors, *Euro-Par 2012: Parallel Processing Workshops*, volume 7640 of *Lecture Notes in Computer Science*. 2013. ISBN 978-3-642-36948-3. doi: 10.1007/978-3-642-36949-0_47.

[20] Vincent Pillet, Vincent Pillet, Jesús Labarta, Toni Cortes, Toni Cortes, Sergi Girona, and Sergi Girona. Paraver: A tool to visualize and analyze parallel code. Technical report, In WoTUG-18, 1995.

[21] Robert M. Love. taskset - linux user's manual, 2003. URL `hhttp://linuxcommand.org/man_pages/taskset1.html`. Accedit: 20/06/2015.

[22] Microsoft. Lvalues and rvalues, 2013. URL `https://msdn.microsoft.com/en-us/library/f90831hc.aspx`. Accedit: 1/06/2015.

[23] Code scheduling for optimizing parallelism and data locality. In Pasqua D'Ambra, Mario Guarracino, and Domenico Talia, editors, *Euro-Par 2010 - Parallel Processing*, volume 6271 of *Lecture Notes in Computer Science*. 2010. ISBN 978-3-642-15276-4. doi: 10.1007/978-3-642-15277-1_20.

[24] Utilizing multiple xeon phi coprocessors on one compute node. In Xian-he Sun, Wenyu Qu, Ivan Stojmenovic, Wanlei Zhou, Zhiyang Li, Hua Guo, Geyong Min, Tingting Yang, Yulei Wu, and Lei Liu, editors, *Algorithms and Architectures for Parallel Processing*, volume 8631 of *Lecture Notes in Computer Science*. 2014. ISBN 978-3-319-11193-3. doi: 10.1007/978-3-319-11194-0_6.

[25] Intel Corporation. Ark — intel® xeon phi™ coprocessor 7120p, 2013. URL `http://ark.intel.com/products/75799/Intel-Xeon-Phi-Coprocessor-7120P-16GB-1_238-GHz-61-core`. Accedit: 16/06/2015.

[26] J.C. Nash. *Compact Numerical Methods for Computers: Linear Algebra and Function Minimisation*. ISBN 9780852743195.

[27] Mateo Valero, Miquel Moreto, Marc Casas, Eduard Ayguade, and Jesus Labarta. Runtime-aware architectures: A first approach. *Supercomputing frontiers and innovations*, 1(1), 2014. ISSN 2313-8734.

Apèndix A

Temps d'execució a Arvei

workers	Original		DAST	
	cycles	speedup	cycles	speedup
seq	1.298×10^9	1.000	–	–
1	1.437×10^9	0.903	1.418×10^9	0.914
2	7.385×10^8	1.756	7.320×10^8	1.771
3	5.129×10^8	2.528	4.764×10^8	2.721
4	3.855×10^8	3.363	3.747×10^8	3.460
5	3.215×10^8	4.033	3.000×10^8	4.321
6	2.672×10^8	4.852	2.625×10^8	4.939
7	2.382×10^8	5.443	2.248×10^8	5.767
8	2.099×10^8	6.178	2.005×10^8	6.467
9	1.824×10^8	7.110	1.788×10^8	7.249
10	1.644×10^8	7.886	1.649×10^8	7.861
11	1.538×10^8	8.432	1.532×10^8	8.460
12	1.406×10^8	9.219	1.394×10^8	9.301
13	1.333×10^8	9.724	1.324×10^8	9.795
14	1.283×10^8	10.104	1.281×10^8	10.122
15	1.237×10^8	10.500	1.219×10^8	10.633
16	1.185×10^8	10.940	1.147×10^8	11.305
17	1.148×10^8	11.292	1.133×10^8	11.440
18	1.107×10^8	11.717	1.359×10^8	9.544
19	1.071×10^8	12.100	1.278×10^8	10.144
20	1.032×10^8	12.568	1.306×10^8	9.929
21	1.000×10^8	12.959	1.392×10^8	9.313
22	9.744×10^7	13.305	1.362×10^8	9.516
23	9.473×10^7	13.686	1.362×10^8	9.522
24	9.292×10^7	13.953	3.137×10^8	4.133

Taula A.1: Execucions del Test 1 (PARALLEL_CREATIONS: 2, TASK_SIZE: 25000)

workers	Original		DAST	
	cycles	speedup	cycles	speedup
seq	2.571×10^9	1.000	–	–
1	2.770×10^9	0.928	4.073×10^9	0.945
2	1.398×10^9	1.840	2.103×10^9	1.830
3	9.666×10^8	2.661	1.416×10^9	2.718
4	7.273×10^8	3.536	1.099×10^9	3.502
5	6.104×10^8	4.213	8.785×10^8	4.380
6	5.089×10^8	5.054	7.428×10^8	5.181
7	4.310×10^8	5.967	6.342×10^8	6.068
8	3.716×10^8	6.920	5.521×10^8	6.970
9	3.451×10^8	7.453	4.894×10^8	7.862
10	2.997×10^8	8.580	4.458×10^8	8.631
11	2.788×10^8	9.226	4.078×10^8	9.437
12	2.566×10^8	10.022	3.795×10^8	10.141
13	2.405×10^8	10.696	3.533×10^8	10.892
14	2.316×10^8	11.107	3.384×10^8	11.373
15	2.200×10^8	11.688	3.203×10^8	12.015
16	2.165×10^8	11.879	3.135×10^8	12.274
17	2.091×10^8	12.298	3.006×10^8	12.800
18	2.033×10^8	12.649	2.992×10^8	12.861
19	1.962×10^8	13.110	2.890×10^8	13.314
20	1.914×10^8	13.436	2.886×10^8	13.334
21	1.862×10^8	13.810	2.747×10^8	14.010
22	1.834×10^8	14.026	2.627×10^8	14.650
23	1.782×10^8	14.435	2.657×10^8	14.482
24	1.734×10^8	14.834	3.776×10^8	10.191

Taula A.2: Execucions del Test 1 (`PARALLEL_CREATIONS`: 2, `TASK_SIZE`: 50000)

workers	Original		DAST	
	cycles	speedup	cycles	speedup
seq	5.119×10^9	1.000	–	–
1	5.446×10^9	0.940	5.379×10^9	0.951
2	2.717×10^9	1.884	2.750×10^9	1.861
3	1.863×10^9	2.747	1.830×10^9	2.796
4	1.404×10^9	3.646	1.430×10^9	3.580
5	1.166×10^9	4.388	1.144×10^9	4.474
6	9.696×10^8	5.278	9.604×10^8	5.329
7	8.273×10^8	6.186	8.125×10^8	6.299
8	7.155×10^8	7.153	7.127×10^8	7.181
9	6.392×10^8	8.007	6.332×10^8	8.083
10	5.755×10^8	8.893	5.759×10^8	8.887
11	5.331×10^8	9.600	5.259×10^8	9.733
12	4.890×10^8	10.467	4.839×10^8	10.576
13	4.602×10^8	11.121	4.550×10^8	11.248
14	4.441×10^8	11.524	4.333×10^8	11.811
15	4.301×10^8	11.899	4.158×10^8	12.310
16	4.157×10^8	12.311	3.990×10^8	12.827
17	4.030×10^8	12.701	3.845×10^8	13.312
18	3.856×10^8	13.273	3.785×10^8	13.523
19	3.795×10^8	13.485	3.651×10^8	14.019
20	3.717×10^8	13.768	3.514×10^8	14.566
21	3.598×10^8	14.225	3.412×10^8	14.999
22	3.503×10^8	14.610	3.266×10^8	15.670
23	3.424×10^8	14.949	3.223×10^8	15.881
24	3.357×10^8	15.248	4.468×10^8	11.456

Taula A.3: Execucions del Test 1 (`PARALLEL_CREATIONS`: 2, `TASK_SIZE`: 100000)

workers	Original		DAST	
	cycles	speedup	cycles	speedup
seq	1.020×10^{10}	1.000	–	–
1	1.063×10^{10}	0.960	1.052×10^{10}	0.970
2	5.287×10^{9}	1.930	5.438×10^{9}	1.876
3	3.658×10^{9}	2.790	3.610×10^{9}	2.827
4	2.745×10^{9}	3.717	2.826×10^{9}	3.611
5	2.282×10^{9}	4.471	2.258×10^{9}	4.520
6	1.901×10^{9}	5.368	1.880×10^{9}	5.426
7	1.609×10^{9}	6.340	1.595×10^{9}	6.399
8	1.394×10^{9}	7.320	1.397×10^{9}	7.305
9	1.248×10^{9}	8.174	1.241×10^{9}	8.223
10	1.122×10^{9}	9.096	1.131×10^{9}	9.019
11	1.036×10^{9}	9.846	1.028×10^{9}	9.925
12	9.525×10^{8}	10.713	9.485×10^{8}	10.757
13	8.947×10^{8}	11.405	8.940×10^{8}	11.414
14	8.586×10^{8}	11.884	8.545×10^{8}	11.942
15	8.332×10^{8}	12.246	8.220×10^{8}	12.414
16	8.045×10^{8}	12.683	7.789×10^{8}	13.101
17	7.922×10^{8}	12.880	7.497×10^{8}	13.610
18	7.558×10^{8}	13.500	7.343×10^{8}	13.896
19	7.366×10^{8}	13.851	7.101×10^{8}	14.370
20	7.220×10^{8}	14.132	6.876×10^{8}	14.839
21	7.040×10^{8}	14.493	6.572×10^{8}	15.527
22	6.903×10^{8}	14.781	6.402×10^{8}	15.939
23	6.708×10^{8}	15.211	6.222×10^{8}	16.399
24	6.587×10^{8}	15.490	7.474×10^{8}	13.653

Taula A.4: Execucions del Test 1 (`PARALLEL_CREATIONS`: 2, `TASK_SIZE`: 200000)

workers	Original		DAST	
	cycles	speedup	cycles	speedup
seq	7.663×10^{9}	1.000	–	–
1	8.170×10^{9}	0.938	8.071×10^{9}	0.949
2	4.066×10^{9}	1.884	4.122×10^{9}	1.859
3	2.819×10^{9}	2.718	2.756×10^{9}	2.779
4	2.116×10^{9}	3.621	2.155×10^{9}	3.555
5	1.753×10^{9}	4.371	1.722×10^{9}	4.449
6	1.458×10^{9}	5.253	1.444×10^{9}	5.305
7	1.243×10^{9}	6.162	1.224×10^{9}	6.257
8	1.075×10^{9}	7.129	1.073×10^{9}	7.138
9	9.622×10^{8}	7.961	9.495×10^{8}	8.068
10	8.671×10^{8}	8.834	8.708×10^{8}	8.797
11	8.008×10^{8}	9.567	7.908×10^{8}	9.687
12	7.363×10^{8}	10.404	7.280×10^{8}	10.522
13	6.898×10^{8}	11.106	6.817×10^{8}	11.237
14	6.560×10^{8}	11.677	6.515×10^{8}	11.758
15	6.255×10^{8}	12.247	6.215×10^{8}	12.326
16	6.079×10^{8}	12.601	5.925×10^{8}	12.929
17	5.872×10^{8}	13.045	5.714×10^{8}	13.408
18	5.768×10^{8}	13.282	5.613×10^{8}	13.649
19	5.604×10^{8}	13.669	5.445×10^{8}	14.070
20	5.395×10^{8}	14.199	5.305×10^{8}	14.440
21	5.305×10^{8}	14.440	5.050×10^{8}	15.169
22	5.116×10^{8}	14.975	4.937×10^{8}	15.516
23	5.044×10^{8}	15.187	4.730×10^{8}	16.196
24	4.939×10^{8}	15.511	5.638×10^{8}	13.587

Taula A.5: Execucions del Test 1 (`PARALLEL_CREATIONS`: 3, `TASK_SIZE`: 100000)

workers	Original		DAST	
	cycles	speedup	cycles	speedup
seq	1.019×10^{10}	1.000	–	–
1	1.090×10^{10}	0.935	1.070×10^{10}	0.952
2	5.386×10^{9}	1.891	5.550×10^{9}	1.835
3	3.761×10^{9}	2.709	3.672×10^{9}	2.774
4	2.841×10^{9}	3.586	2.901×10^{9}	3.512
5	2.356×10^{9}	4.323	2.299×10^{9}	4.431
6	1.953×10^{9}	5.216	1.936×10^{9}	5.262
7	1.675×10^{9}	6.083	1.642×10^{9}	6.205
8	1.445×10^{9}	7.050	1.446×10^{9}	7.043
9	1.288×10^{9}	7.908	1.270×10^{9}	8.021
10	1.159×10^{9}	8.793	1.163×10^{9}	8.761
11	1.073×10^{9}	9.491	1.055×10^{9}	9.655
12	9.876×10^{8}	10.315	9.737×10^{8}	10.462
13	9.241×10^{8}	11.023	9.197×10^{8}	11.077
14	8.787×10^{8}	11.593	8.712×10^{8}	11.693
15	8.521×10^{8}	11.955	8.427×10^{8}	12.088
16	8.117×10^{8}	12.550	7.932×10^{8}	12.844
17	7.763×10^{8}	13.123	7.599×10^{8}	13.406
18	7.551×10^{8}	13.490	7.525×10^{8}	13.537
19	7.505×10^{8}	13.573	7.251×10^{8}	14.050
20	7.187×10^{8}	14.173	7.051×10^{8}	14.448
21	6.973×10^{8}	14.610	6.757×10^{8}	15.077
22	6.788×10^{8}	15.008	6.624×10^{8}	15.379
23	6.618×10^{8}	15.392	6.254×10^{8}	16.290
24	6.543×10^{8}	15.568	7.686×10^{8}	13.253

Taula A.6: Execucions del Test 1 (`PARALLEL_CREATIONS`: 4, `TASK_SIZE`: 100000)

workers	Original		DAST	
	cycles	speedup	cycles	speedup
seq	9.383×10^{9}	1.000	–	–
1	1.293×10^{10}	0.725	9.713×10^{9}	0.966
2	6.974×10^{9}	1.345	5.086×10^{9}	1.845
3	5.216×10^{9}	1.799	4.374×10^{9}	2.145
4	4.217×10^{9}	2.225	3.547×10^{9}	2.645
5	3.647×10^{9}	2.573	2.655×10^{9}	3.534
6	2.950×10^{9}	3.180	1.819×10^{9}	5.159
7	2.416×10^{9}	3.884	1.509×10^{9}	6.218
8	1.937×10^{9}	4.844	1.325×10^{9}	7.083
9	1.603×10^{9}	5.855	1.175×10^{9}	7.988
10	1.304×10^{9}	7.195	1.071×10^{9}	8.762
11	1.142×10^{9}	8.215	9.852×10^{8}	9.525
12	1.011×10^{9}	9.285	8.893×10^{8}	10.552
13	9.081×10^{8}	10.333	8.393×10^{8}	11.180
14	8.767×10^{8}	10.703	8.144×10^{8}	11.522
15	8.717×10^{8}	10.765	7.865×10^{8}	11.931
16	8.716×10^{8}	10.765	7.585×10^{8}	12.371
17	8.622×10^{8}	10.884	7.329×10^{8}	12.804
18	8.618×10^{8}	10.888	7.286×10^{8}	12.878
19	8.607×10^{8}	10.902	7.218×10^{8}	13.000
20	8.749×10^{8}	10.725	7.131×10^{8}	13.159
21	8.781×10^{8}	10.686	6.940×10^{8}	13.521
22	8.853×10^{8}	10.600	6.774×10^{8}	13.852
23	9.061×10^{8}	10.355	6.934×10^{8}	13.532
24	9.052×10^{8}	10.366	1.237×10^{9}	7.587

Taula A.7: Execucions del Test 2 (`PARALLEL_CREATIONS`: 2, `TASK_SIZE`: 25000)

workers	Original		DAST	
	cycles	speedup	cycles	speedup
seq	1.868×10^{10}	1.000	–	–
1	2.232×10^{10}	0.837	1.917×10^{10}	0.974
2	1.161×10^{10}	1.609	9.900×10^{9}	1.887
3	8.274×10^{9}	2.257	6.634×10^{9}	2.815
4	6.400×10^{9}	2.918	5.203×10^{9}	3.590
5	5.431×10^{9}	3.439	4.159×10^{9}	4.491
6	4.540×10^{9}	4.114	3.496×10^{9}	5.342
7	3.812×10^{9}	4.899	2.954×10^{9}	6.322
8	3.209×10^{9}	5.821	2.581×10^{9}	7.236
9	2.787×10^{9}	6.702	2.279×10^{9}	8.196
10	2.414×10^{9}	7.737	2.100×10^{9}	8.892
11	2.170×10^{9}	8.607	1.885×10^{9}	9.906
12	1.932×10^{9}	9.665	1.739×10^{9}	10.739
13	1.747×10^{9}	10.691	1.633×10^{9}	11.435
14	1.680×10^{9}	11.117	1.592×10^{9}	11.731
15	1.594×10^{9}	11.717	1.517×10^{9}	12.314
16	1.542×10^{9}	12.114	1.483×10^{9}	12.592
17	1.505×10^{9}	12.410	1.430×10^{9}	13.065
18	1.427×10^{9}	13.091	1.406×10^{9}	13.283
19	1.409×10^{9}	13.258	1.379×10^{9}	13.546
20	1.407×10^{9}	13.273	1.320×10^{9}	14.152
21	1.421×10^{9}	13.148	1.286×10^{9}	14.526
22	1.408×10^{9}	13.269	1.266×10^{9}	14.754
23	1.437×10^{9}	12.996	1.278×10^{9}	14.616
24	1.434×10^{9}	13.024	1.479×10^{9}	12.629

Taula A.8: Execucions del Test 2 (`PARALLEL_CREATIONS`: 2, `TASK_SIZE`: 50000)

workers	Original		DAST	
	cycles	speedup	cycles	speedup
seq	3.744×10^{10}	1.000	–	–
1	4.111×10^{10}	0.911	3.803×10^{10}	0.984
2	2.102×10^{10}	1.781	1.966×10^{10}	1.905
3	1.469×10^{10}	2.548	1.316×10^{10}	2.845
4	1.116×10^{10}	3.354	1.023×10^{10}	3.658
5	9.334×10^{9}	4.011	8.195×10^{9}	4.568
6	7.796×10^{9}	4.802	6.869×10^{9}	5.450
7	6.630×10^{9}	5.647	5.807×10^{9}	6.446
8	5.697×10^{9}	6.571	5.093×10^{9}	7.351
9	5.038×10^{9}	7.431	4.523×10^{9}	8.278
10	4.460×10^{9}	8.394	4.124×10^{9}	9.077
11	4.076×10^{9}	9.184	3.747×10^{9}	9.990
12	3.682×10^{9}	10.167	3.458×10^{9}	10.825
13	3.377×10^{9}	11.086	3.247×10^{9}	11.529
14	3.181×10^{9}	11.769	3.148×10^{9}	11.892
15	3.155×10^{9}	11.867	2.972×10^{9}	12.597
16	2.940×10^{9}	12.733	2.945×10^{9}	12.713
17	2.896×10^{9}	12.926	2.835×10^{9}	13.203
18	2.749×10^{9}	13.618	2.763×10^{9}	13.549
19	2.700×10^{9}	13.866	2.693×10^{9}	13.901
20	2.647×10^{9}	14.141	2.645×10^{9}	14.156
21	2.592×10^{9}	14.443	2.576×10^{9}	14.533
22	2.541×10^{9}	14.732	2.500×10^{9}	14.975
23	2.536×10^{9}	14.760	2.420×10^{9}	15.468
24	2.531×10^{9}	14.789	2.519×10^{9}	14.862

Taula A.9: Execucions del Test 2 (`PARALLEL_CREATIONS`: 2, `TASK_SIZE`: 100000)

workers	Original		DAST	
	cycles	speedup	cycles	speedup
seq	7.484×10^{10}	1.000	–	–
1	7.886×10^{10}	0.949	7.595×10^{10}	0.985
2	3.995×10^{10}	1.873	3.919×10^{10}	1.910
3	2.765×10^{10}	2.706	2.628×10^{10}	2.848
4	2.091×10^{10}	3.579	2.041×10^{10}	3.666
5	1.735×10^{10}	4.313	1.632×10^{10}	4.585
6	1.447×10^{10}	5.172	1.363×10^{10}	5.491
7	1.225×10^{10}	6.110	1.155×10^{10}	6.479
8	1.060×10^{10}	7.059	1.013×10^{10}	7.391
9	9.462×10^{9}	7.910	8.978×10^{9}	8.336
10	8.481×10^{9}	8.824	8.191×10^{9}	9.137
11	7.785×10^{9}	9.614	7.441×10^{9}	10.058
12	7.081×10^{9}	10.569	6.873×10^{9}	10.889
13	6.576×10^{9}	11.382	6.447×10^{9}	11.608
14	6.302×10^{9}	11.877	6.193×10^{9}	12.085
15	6.174×10^{9}	12.123	6.018×10^{9}	12.436
16	5.928×10^{9}	12.625	5.741×10^{9}	13.037
17	5.661×10^{9}	13.220	5.591×10^{9}	13.385
18	5.474×10^{9}	13.671	5.545×10^{9}	13.497
19	5.391×10^{9}	13.882	5.452×10^{9}	13.727
20	5.204×10^{9}	14.381	5.228×10^{9}	14.316
21	5.132×10^{9}	14.585	5.086×10^{9}	14.714
22	4.964×10^{9}	15.077	4.909×10^{9}	15.246
23	4.864×10^{9}	15.388	4.830×10^{9}	15.496
24	4.816×10^{9}	15.539	4.908×10^{9}	15.249

Taula A.10: Execucions del Test 2 (`PARALLEL_CREATIONS`: 2, `TASK_SIZE`: 200000)

workers	Original		DAST	
	cycles	speedup	cycles	speedup
seq	5.602×10^{10}	1.000	–	–
1	6.171×10^{10}	0.908	5.690×10^{10}	0.984
2	3.154×10^{10}	1.776	2.942×10^{10}	1.904
3	2.183×10^{10}	2.566	1.975×10^{10}	2.837
4	1.662×10^{10}	3.370	1.534×10^{10}	3.652
5	1.386×10^{10}	4.042	1.228×10^{10}	4.560
6	1.162×10^{10}	4.821	1.032×10^{10}	5.427
7	9.951×10^{9}	5.629	8.742×10^{9}	6.408
8	8.641×10^{9}	6.483	7.640×10^{9}	7.332
9	7.785×10^{9}	7.195	6.782×10^{9}	8.260
10	6.985×10^{9}	8.020	6.186×10^{9}	9.056
11	6.470×10^{9}	8.658	5.626×10^{9}	9.956
12	5.895×10^{9}	9.502	5.198×10^{9}	10.776
13	5.439×10^{9}	10.299	4.886×10^{9}	11.464
14	5.182×10^{9}	10.810	4.697×10^{9}	11.926
15	4.876×10^{9}	11.488	4.512×10^{9}	12.416
16	4.727×10^{9}	11.852	4.376×10^{9}	12.801
17	4.475×10^{9}	12.519	4.201×10^{9}	13.336
18	4.326×10^{9}	12.949	4.226×10^{9}	13.256
19	4.197×10^{9}	13.346	4.062×10^{9}	13.791
20	4.005×10^{9}	13.986	3.958×10^{9}	14.155
21	3.921×10^{9}	14.288	3.752×10^{9}	14.929
22	3.856×10^{9}	14.529	3.693×10^{9}	15.169
23	3.723×10^{9}	15.048	3.589×10^{9}	15.610
24	3.624×10^{9}	15.459	3.979×10^{9}	14.077

Taula A.11: Execucions del Test 2 (`PARALLEL_CREATIONS`: 3, `TASK_SIZE`: 100000)

workers	Original		DAST	
	cycles	speedup	cycles	speedup
seq	7.467×10^{10}	1.000	–	–
1	8.207×10^{10}	0.910	7.608×10^{10}	0.982
2	4.193×10^{10}	1.781	3.923×10^{10}	1.903
3	2.921×10^{10}	2.556	2.630×10^{10}	2.839
4	2.205×10^{10}	3.387	2.047×10^{10}	3.647
5	1.842×10^{10}	4.054	1.639×10^{10}	4.556
6	1.541×10^{10}	4.847	1.374×10^{10}	5.434
7	1.319×10^{10}	5.660	1.174×10^{10}	6.358
8	1.154×10^{10}	6.469	1.027×10^{10}	7.274
9	1.037×10^{10}	7.203	9.303×10^{9}	8.027
10	9.334×10^{9}	8.000	8.286×10^{9}	9.012
11	8.646×10^{9}	8.637	7.676×10^{9}	9.729
12	7.922×10^{9}	9.426	7.007×10^{9}	10.657
13	7.456×10^{9}	10.016	6.546×10^{9}	11.407
14	7.196×10^{9}	10.377	6.361×10^{9}	11.740
15	6.847×10^{9}	10.906	6.064×10^{9}	12.313
16	6.495×10^{9}	11.497	5.737×10^{9}	13.016
17	6.088×10^{9}	12.266	5.535×10^{9}	13.490
18	5.928×10^{9}	12.597	5.548×10^{9}	13.458
19	5.696×10^{9}	13.110	5.368×10^{9}	13.910
20	5.541×10^{9}	13.476	5.195×10^{9}	14.374
21	5.327×10^{9}	14.017	5.120×10^{9}	14.585
22	5.304×10^{9}	14.079	4.992×10^{9}	14.959
23	5.134×10^{9}	14.545	4.775×10^{9}	15.637
24	4.977×10^{9}	15.003	5.193×10^{9}	14.380

Taula A.12: Execucions del Test 2 (PARALLEL_CREATIONS: 4, TASK_SIZE: 100000)

workers	Original		DAST	
	cycles	speedup	cycles	speedup
seq	1.403×10^{10}	1.000	–	–
1	1.949×10^{10}	0.720	1.464×10^{10}	0.958
2	1.061×10^{10}	1.322	7.589×10^{9}	1.849
3	7.346×10^{9}	1.910	7.603×10^{9}	1.845
4	5.823×10^{9}	2.409	7.093×10^{9}	1.978
5	5.109×10^{9}	2.746	6.160×10^{9}	2.278
6	4.511×10^{9}	3.110	5.045×10^{9}	2.781
7	4.015×10^{9}	3.495	3.748×10^{9}	3.744
8	3.521×10^{9}	3.985	3.054×10^{9}	4.593
9	3.117×10^{9}	4.502	2.015×10^{9}	6.963
10	2.755×10^{9}	5.093	1.677×10^{9}	8.368
11	2.501×10^{9}	5.609	1.499×10^{9}	9.360
12	2.185×10^{9}	6.420	1.375×10^{9}	10.204
13	1.831×10^{9}	7.664	1.287×10^{9}	10.897
14	1.674×10^{9}	8.382	1.221×10^{9}	11.491
15	1.463×10^{9}	9.589	1.211×10^{9}	11.581
16	1.333×10^{9}	10.525	1.141×10^{9}	12.298
17	1.239×10^{9}	11.328	1.101×10^{9}	12.741
18	1.184×10^{9}	11.850	1.098×10^{9}	12.779
19	1.139×10^{9}	12.314	1.062×10^{9}	13.214
20	1.136×10^{9}	12.351	1.037×10^{9}	13.534
21	1.052×10^{9}	13.338	1.032×10^{9}	13.592
22	1.030×10^{9}	13.625	9.766×10^{8}	14.366
23	9.928×10^{8}	14.131	9.439×10^{8}	14.864
24	9.567×10^{8}	14.665	4.163×10^{9}	3.370

Taula A.13: Execucions del Test 2 (PARALLEL_CREATIONS: 3, TASK_SIZE: 25000)

workers	Original		DAST	
	cycles	speedup	cycles	speedup
seq	1.869×10^{10}	1.000	–	–
1	2.571×10^{10}	0.727	1.949×10^{10}	0.959
2	1.372×10^{10}	1.362	1.016×10^{10}	1.840
3	1.022×10^{10}	1.828	1.043×10^{10}	1.791
4	7.450×10^{9}	2.508	1.014×10^{10}	1.843
5	6.432×10^{9}	2.905	9.143×10^{9}	2.044
6	5.539×10^{9}	3.374	8.986×10^{9}	2.079
7	4.993×10^{9}	3.743	7.670×10^{9}	2.436
8	4.526×10^{9}	4.129	6.594×10^{9}	2.834
9	4.243×10^{9}	4.404	5.678×10^{9}	3.291
10	3.886×10^{9}	4.808	4.261×10^{9}	4.386
11	3.672×10^{9}	5.088	3.503×10^{9}	5.334
12	3.347×10^{9}	5.583	2.563×10^{9}	7.290
13	3.025×10^{9}	6.177	1.976×10^{9}	9.457
14	2.861×10^{9}	6.531	1.786×10^{9}	10.464
15	2.534×10^{9}	7.375	1.682×10^{9}	11.107
16	2.239×10^{9}	8.347	1.553×10^{9}	12.031
17	2.099×10^{9}	8.901	1.480×10^{9}	12.629
18	1.890×10^{9}	9.889	1.660×10^{9}	11.257
19	1.804×10^{9}	10.358	1.478×10^{9}	12.641
20	1.732×10^{9}	10.788	1.432×10^{9}	13.049
21	1.605×10^{9}	11.643	1.403×10^{9}	13.321
22	1.487×10^{9}	12.566	1.338×10^{9}	13.961
23	1.439×10^{9}	12.990	1.310×10^{9}	14.263
24	1.368×10^{9}	13.656	5.283×10^{9}	3.537

Taula A.14: Execucions del Test 2 (`PARALLEL_CREATIONS`: 4, `TASK_SIZE`: 25000)

workers	Original		DAST	
	cycles	speedup	cycles	speedup
seq	2.388×10^{10}	1.000	–	–
1	3.247×10^{10}	0.736	2.449×10^{10}	0.975
2	1.747×10^{10}	1.367	1.276×10^{10}	1.871
3	1.239×10^{10}	1.928	1.259×10^{10}	1.897
4	1.005×10^{10}	2.377	1.303×10^{10}	1.833
5	7.829×10^{9}	3.050	1.307×10^{10}	1.827
6	6.709×10^{9}	3.560	1.203×10^{10}	1.985
7	5.953×10^{9}	4.012	1.154×10^{10}	2.070
8	5.324×10^{9}	4.486	1.057×10^{10}	2.260
9	5.008×10^{9}	4.769	9.964×10^{9}	2.397
10	4.683×10^{9}	5.100	8.170×10^{9}	2.923
11	4.498×10^{9}	5.310	7.430×10^{9}	3.214
12	4.205×10^{9}	5.679	6.436×10^{9}	3.711
13	3.911×10^{9}	6.106	5.167×10^{9}	4.622
14	3.629×10^{9}	6.582	3.870×10^{9}	6.171
15	3.358×10^{9}	7.111	3.090×10^{9}	7.730
16	3.075×10^{9}	7.766	2.444×10^{9}	9.770
17	2.962×10^{9}	8.063	2.262×10^{9}	10.557
18	2.876×10^{9}	8.303	2.716×10^{9}	8.795
19	2.635×10^{9}	9.063	2.636×10^{9}	9.060
20	2.458×10^{9}	9.715	2.274×10^{9}	10.500
21	2.312×10^{9}	10.331	2.145×10^{9}	11.136
22	2.217×10^{9}	10.773	2.032×10^{9}	11.752
23	2.130×10^{9}	11.215	2.061×10^{9}	11.586
24	1.976×10^{9}	12.084	8.119×10^{9}	2.942

Taula A.15: Execucions del Test 2 (`PARALLEL_CREATIONS`: 5, `TASK_SIZE`: 25000)

workers	Original		DAST	
	cycles	speedup	cycles	speedup
seq	2.005×10^9	1	–	–
1	2.253×10^9	0.890	2.243×10^9	0.893
2	1.390×10^9	1.442	1.190×10^9	1.685
3	1.087×10^9	1.844	8.613×10^8	2.327
4	8.480×10^8	2.363	7.354×10^8	2.725
5	5.970×10^8	3.357	7.070×10^8	2.835
6	5.379×10^8	3.726	1.107×10^9	1.811
7	6.669×10^8	3.005	9.913×10^8	2.022
8	7.127×10^8	2.812	9.624×10^8	2.083
9	7.310×10^8	2.742	8.625×10^8	2.324
10	7.211×10^8	2.779	8.076×10^8	2.482
11	7.632×10^8	2.626	7.833×10^8	2.559
12	7.536×10^8	2.660	9.324×10^8	2.150
13	8.300×10^8	2.415	8.187×10^8	2.448
14	7.668×10^8	2.614	8.343×10^8	2.402
15	7.697×10^8	2.604	8.188×10^8	2.448
16	7.886×10^8	2.542	8.335×10^8	2.404
17	7.750×10^8	2.586	8.205×10^8	2.443
18	7.845×10^8	2.555	8.705×10^8	2.302
19	8.159×10^8	2.457	9.088×10^8	2.205
20	8.205×10^8	2.443	9.264×10^8	2.163
21	8.490×10^8	2.361	8.832×10^8	2.269
22	8.683×10^8	2.308	8.553×10^8	2.343
23	8.541×10^8	2.347	9.293×10^8	2.157
24	8.459×10^8	2.369	1.250×10^9	1.604

Taula A.16: Execucions del programa Cholesky (`MATRIX_SIZE`: 2048, `BLOCK_SIZE`: 32)

workers	Original		DAST	
	cycles	speedup	cycles	speedup
seq	1.025×10^9	1.000	–	–
1	1.028×10^9	0.997	1.043×10^9	0.983
2	5.131×10^8	1.997	5.222×10^8	1.962
3	3.591×10^8	2.854	3.442×10^8	2.977
4	2.731×10^8	3.752	2.685×10^8	3.816
5	2.297×10^8	4.462	2.156×10^8	4.753
6	1.933×10^8	5.302	1.882×10^8	5.445
7	1.794×10^8	5.711	1.683×10^8	6.088
8	1.626×10^8	6.302	1.526×10^8	6.716
9	1.476×10^8	6.942	1.454×10^8	7.046
10	1.373×10^8	7.462	1.435×10^8	7.142
11	1.284×10^8	7.979	1.374×10^8	7.457
12	1.298×10^8	7.893	1.268×10^8	8.080
13	1.406×10^8	7.286	1.286×10^8	7.965
14	1.416×10^8	7.238	1.321×10^8	7.757
15	1.409×10^8	7.271	1.278×10^8	8.019
16	1.424×10^8	7.197	1.284×10^8	7.980
17	1.426×10^8	7.185	1.276×10^8	8.027
18	1.430×10^8	7.164	1.427×10^8	7.179
19	1.442×10^8	7.108	1.471×10^8	6.963
20	1.432×10^8	7.156	1.470×10^8	6.969
21	1.431×10^8	7.162	1.469×10^8	6.977
22	1.433×10^8	7.148	1.462×10^8	7.008
23	1.479×10^8	6.930	1.458×10^8	7.027
24	1.472×10^8	6.961	2.250×10^8	4.553

Taula A.17: Execucions del programa Cholesky (`MATRIX_SIZE`: 2048, `BLOCK_SIZE`: 64)

workers	Original		DAST	
	cycles	speedup	cycles	speedup
seq	8.627×10^8	1.000	–	–
1	8.287×10^8	1.041	8.410×10^8	1.025
2	4.425×10^8	1.949	4.374×10^8	1.972
3	2.963×10^8	2.910	3.022×10^8	2.854
4	2.267×10^8	3.804	2.357×10^8	3.659
5	1.951×10^8	4.421	1.912×10^8	4.510
6	1.663×10^8	5.187	1.677×10^8	5.142
7	1.481×10^8	5.825	1.526×10^8	5.651
8	1.373×10^8	6.281	1.408×10^8	6.124
9	1.263×10^8	6.826	1.281×10^8	6.731
10	1.200×10^8	7.187	1.196×10^8	7.210
11	1.125×10^8	7.669	1.146×10^8	7.523
12	1.083×10^8	7.965	1.175×10^8	7.339
13	1.196×10^8	7.213	1.153×10^8	7.478
14	1.167×10^8	7.391	1.168×10^8	7.385
15	1.164×10^8	7.412	1.175×10^8	7.337
16	1.184×10^8	7.281	1.205×10^8	7.155
17	1.183×10^8	7.291	1.202×10^8	7.172
18	1.187×10^8	7.264	1.204×10^8	7.161
19	1.195×10^8	7.218	1.203×10^8	7.166
20	1.209×10^8	7.132	1.209×10^8	7.133
21	1.214×10^8	7.105	1.219×10^8	7.074
22	1.238×10^8	6.964	1.248×10^8	6.908
23	1.205×10^8	7.157	1.232×10^8	7.003
24	1.232×10^8	7.000	2.400×10^8	3.594

Taula A.18: Execucions del programa Cholesky (MATRIX_SIZE: 2048, BLOCK_SIZE: 128)

workers	Original		DAST	
	cycles	speedup	cycles	speedup
1	8.219×10^8	1.075	–	–
1	7.646×10^8	1.075	7.705×10^8	1.067
2	4.025×10^8	2.043	4.162×10^8	1.975
3	2.893×10^8	2.842	2.984×10^8	2.755
4	2.375×10^8	3.462	2.453×10^8	3.352
5	2.085×10^8	3.943	2.057×10^8	3.996
6	1.860×10^8	4.419	1.833×10^8	4.485
7	1.662×10^8	4.945	1.750×10^8	4.698
8	1.551×10^8	5.301	1.552×10^8	5.297
9	1.540×10^8	5.338	1.443×10^8	5.697
10	1.555×10^8	5.286	1.457×10^8	5.644
11	1.499×10^8	5.486	1.411×10^8	5.825
12	1.444×10^8	5.693	1.807×10^8	4.551
13	2.087×10^8	3.940	2.202×10^8	3.733
14	2.223×10^8	3.698	2.196×10^8	3.744
15	2.210×10^8	3.720	2.207×10^8	3.725
16	2.265×10^8	3.630	2.286×10^8	3.597
17	2.281×10^8	3.605	2.242×10^8	3.667
18	2.255×10^8	3.646	2.262×10^8	3.634
19	2.280×10^8	3.605	2.261×10^8	3.636
20	2.298×10^8	3.578	2.283×10^8	3.600
21	2.344×10^8	3.508	2.337×10^8	3.519
22	2.290×10^8	3.590	2.365×10^8	3.477
23	2.320×10^8	3.544	2.387×10^8	3.445
24	2.354×10^8	3.493	3.430×10^8	2.397

Taula A.19: Execucions del programa Cholesky (MATRIX_SIZE: 2048, BLOCK_SIZE: 256)

workers	Original		DAST	
	cycles	speedup	cycles	speedup
seq	7.894×10^8	1.000	–	–
1	7.330×10^8	1.077	7.556×10^8	1.045
2	4.515×10^8	1.749	4.678×10^8	1.688
3	3.890×10^8	2.030	3.779×10^8	2.090
4	3.421×10^8	2.308	3.538×10^8	2.232
5	3.377×10^8	2.339	3.382×10^8	2.335
6	3.372×10^8	2.342	3.389×10^8	2.330
7	3.297×10^8	2.395	3.286×10^8	2.403
8	3.376×10^8	2.339	3.355×10^8	2.354
9	3.354×10^8	2.354	3.279×10^8	2.408
10	3.381×10^8	2.336	3.379×10^8	2.337
11	3.373×10^8	2.341	3.383×10^8	2.334
12	3.371×10^8	2.342	5.476×10^8	1.442
13	6.009×10^8	1.314	5.782×10^8	1.366
14	6.255×10^8	1.262	5.590×10^8	1.413
15	6.194×10^8	1.275	6.131×10^8	1.288
16	6.429×10^8	1.228	6.256×10^8	1.262
17	6.359×10^8	1.242	6.113×10^8	1.292
18	6.458×10^8	1.223	5.622×10^8	1.405
19	6.501×10^8	1.215	6.238×10^8	1.266
20	6.326×10^8	1.248	6.594×10^8	1.198
21	6.438×10^8	1.226	6.446×10^8	1.225
22	6.245×10^8	1.265	6.638×10^8	1.190
23	6.283×10^8	1.257	6.683×10^8	1.182
24	6.407×10^8	1.232	6.693×10^8	1.180

Taula A.20: Execucions del programa Cholesky (`MATRIX_SIZE`: 2048, `BLOCK_SIZE`: 512)

workers	Original		DAST	
	cycles	speedup	cycles	speedup
seq	6.342×10^9	1.007	–	–
1	6.298×10^9	1.007	6.668×10^9	0.951
2	3.267×10^9	1.942	3.408×10^9	1.861
3	2.279×10^9	2.783	2.307×10^9	2.749
4	1.734×10^9	3.658	1.801×10^9	3.522
5	1.451×10^9	4.372	1.447×10^9	4.384
6	1.219×10^9	5.203	1.235×10^9	5.135
7	1.053×10^9	6.025	1.062×10^9	5.971
8	9.270×10^8	6.842	9.427×10^8	6.728
9	8.370×10^8	7.578	8.430×10^8	7.524
10	7.626×10^8	8.317	7.792×10^8	8.140
11	7.127×10^8	8.899	7.137×10^8	8.887
12	6.586×10^8	9.630	6.831×10^8	9.285
13	6.741×10^8	9.408	6.917×10^8	9.169
14	6.787×10^8	9.346	6.979×10^8	9.088
15	6.792×10^8	9.338	7.008×10^8	9.051
16	6.857×10^8	9.249	7.037×10^8	9.013
17	6.859×10^8	9.246	7.093×10^8	8.942
18	6.858×10^8	9.248	7.082×10^8	8.956
19	6.875×10^8	9.226	7.086×10^8	8.951
20	6.935×10^8	9.145	7.140×10^8	8.883
21	6.969×10^8	9.102	7.190×10^8	8.821
22	6.968×10^8	9.102	7.210×10^8	8.796
23	7.024×10^8	9.030	7.245×10^8	8.754
24	7.080×10^8	8.959	9.224×10^8	6.876

Taula A.21: Execucions del programa Cholesky (`MATRIX_SIZE`: 4096, `BLOCK_SIZE`: 128)

workers	Original		DAST	
	cycles	speedup	cycles	speedup
seq	4.974×10^{10}	1.000	–	–
1	4.989×10^{10}	0.997	5.067×10^{10}	0.981
2	2.593×10^{10}	1.917	2.698×10^{10}	1.842
3	1.811×10^{10}	2.745	1.822×10^{10}	2.728
4	1.380×10^{10}	3.602	1.428×10^{10}	3.480
5	1.152×10^{10}	4.314	1.149×10^{10}	4.325
6	9.695×10^{9}	5.128	9.775×10^{9}	5.086
7	8.329×10^{9}	5.968	8.424×10^{9}	5.901
8	7.344×10^{9}	6.769	7.461×10^{9}	6.663
9	6.633×10^{9}	7.495	6.673×10^{9}	7.450
10	6.035×10^{9}	8.237	6.122×10^{9}	8.121
11	5.630×10^{9}	8.830	5.643×10^{9}	8.810
12	5.206×10^{9}	9.549	5.383×10^{9}	9.235
13	5.258×10^{9}	9.455	5.393×10^{9}	9.218
14	5.277×10^{9}	9.421	5.415×10^{9}	9.180
15	5.294×10^{9}	9.390	5.434×10^{9}	9.149
16	5.314×10^{9}	9.354	5.436×10^{9}	9.145
17	5.330×10^{9}	9.327	5.496×10^{9}	9.044
18	5.333×10^{9}	9.322	5.516×10^{9}	9.012
19	5.350×10^{9}	9.292	5.533×10^{9}	8.984
20	5.368×10^{9}	9.262	5.503×10^{9}	9.033
21	5.381×10^{9}	9.238	5.525×10^{9}	8.998
22	5.388×10^{9}	9.226	5.545×10^{9}	8.966
23	5.424×10^{9}	9.165	5.560×10^{9}	8.941
24	5.426×10^{9}	9.162	6.052×10^{9}	8.214

Taula A.22: Execucions del programa Cholesky (`MATRIX_SIZE`: 8192, `BLOCK_SIZE`: 128)

workers	Original		DAST	
	cycles	speedup	cycles	speedup
1	5.911×10^{9}	1.000	–	–
1	5.870×10^{9}	1.007	5.915×10^{9}	0.999
2	3.040×10^{9}	1.944	3.132×10^{9}	1.887
3	2.127×10^{9}	2.778	2.153×10^{9}	2.745
4	1.625×10^{9}	3.636	1.677×10^{9}	3.523
5	1.356×10^{9}	4.359	1.360×10^{9}	4.345
6	1.154×10^{9}	5.118	1.141×10^{9}	5.178
7	9.880×10^{8}	5.980	9.843×10^{8}	6.003
8	8.677×10^{8}	6.809	8.778×10^{8}	6.731
9	7.915×10^{8}	7.465	7.918×10^{8}	7.462
10	7.193×10^{8}	8.214	7.256×10^{8}	8.143
11	6.773×10^{8}	8.723	6.696×10^{8}	8.825
12	6.296×10^{8}	9.384	7.070×10^{8}	8.357
13	7.107×10^{8}	8.314	7.541×10^{8}	7.836
14	7.208×10^{8}	8.197	7.475×10^{8}	7.904
15	7.420×10^{8}	7.963	7.478×10^{8}	7.901
16	7.459×10^{8}	7.922	7.532×10^{8}	7.845
17	7.563×10^{8}	7.813	7.724×10^{8}	7.650
18	7.482×10^{8}	7.897	7.535×10^{8}	7.841
19	7.512×10^{8}	7.866	7.699×10^{8}	7.675
20	7.537×10^{8}	7.840	7.680×10^{8}	7.693
21	7.621×10^{8}	7.753	7.740×10^{8}	7.634
22	7.674×10^{8}	7.700	7.824×10^{8}	7.552
23	7.801×10^{8}	7.574	8.042×10^{8}	7.348
24	7.877×10^{8}	7.501	9.999×10^{8}	5.909

Taula A.23: Execucions del programa Cholesky (`MATRIX_SIZE`: 4096, `BLOCK_SIZE`: 256)

workers	Original		DAST	
	cycles	speedup	cycles	speedup
seq	4.662×10^{10}	1.000	–	–
1	4.611×10^{10}	1.011	4.640×10^{10}	1.005
2	2.377×10^{10}	1.960	2.450×10^{10}	1.903
3	1.653×10^{10}	2.820	1.657×10^{10}	2.813
4	1.253×10^{10}	3.720	1.295×10^{10}	3.599
5	1.045×10^{10}	4.462	1.043×10^{10}	4.468
6	8.768×10^{9}	5.316	8.725×10^{9}	5.342
7	7.433×10^{9}	6.270	7.457×10^{9}	6.251
8	6.508×10^{9}	7.162	6.559×10^{9}	7.107
9	5.830×10^{9}	7.995	5.851×10^{9}	7.966
10	5.278×10^{9}	8.831	5.358×10^{9}	8.699
11	4.887×10^{9}	9.537	4.882×10^{9}	9.547
12	4.506×10^{9}	10.345	4.729×10^{9}	9.857
13	4.624×10^{9}	10.079	4.823×10^{9}	9.663
14	4.671×10^{9}	9.979	4.853×10^{9}	9.604
15	4.740×10^{9}	9.833	4.876×10^{9}	9.559
16	4.747×10^{9}	9.819	4.922×10^{9}	9.470
17	4.797×10^{9}	9.716	4.959×10^{9}	9.398
18	4.847×10^{9}	9.616	4.991×10^{9}	9.340
19	4.859×10^{9}	9.592	5.050×10^{9}	9.229
20	4.898×10^{9}	9.516	5.078×10^{9}	9.179
21	4.932×10^{9}	9.450	5.137×10^{9}	9.073
22	4.974×10^{9}	9.371	5.170×10^{9}	9.015
23	5.030×10^{9}	9.266	5.216×10^{9}	8.935
24	5.061×10^{9}	9.210	5.343×10^{9}	8.723

Taula A.24: Execucions del programa Cholesky (MATRIX_SIZE: 8192, BLOCK_SIZE: 256)

workers	Original		DAST	
	ms	speedup	ms	speedup
seq	1767.661	1.000	–	–
1	1903.275	0.929	1851.926	0.954
2	995.430	1.776	936.611	1.887
3	687.127	2.573	635.483	2.782
4	524.918	3.368	498.675	3.545
5	442.298	3.997	405.022	4.364
6	375.385	4.709	369.862	4.779
7	345.663	5.114	349.171	5.062
8	318.402	5.552	339.260	5.210
9	300.420	5.884	337.650	5.235
10	309.193	5.717	333.017	5.308
11	315.590	5.601	326.738	5.410
12	314.240	5.625	290.925	6.076
13	326.381	5.416	285.906	6.183
14	327.717	5.394	287.434	6.150
15	319.574	5.531	285.676	6.188
16	316.260	5.589	282.332	6.261
17	322.584	5.480	278.977	6.336
18	306.356	5.770	328.859	5.375
19	301.675	5.859	326.441	5.415
20	320.270	5.519	322.332	5.484
21	326.535	5.413	320.605	5.514
22	316.977	5.577	318.379	5.552
23	312.837	5.650	315.456	5.604
24	315.216	5.608	463.933	3.810

Taula A.25: Execucions del programa Matrix Multiply (MATRIX_SIZE: 2048, BLOCK_SIZE: 64)

workers	Original		DAST	
	ms	speedup	ms	speedup
seq	14761.785	1.000	–	$-N$
1	16794.077	0.879	16201.117	0.911
2	8725.487	1.692	8229.603	1.794
3	6806.894	2.169	5451.386	2.708
4	5408.745	2.729	4243.665	3.479
5	4497.478	3.282	3469.817	4.254
6	3837.019	3.847	3420.422	4.316
7	3519.926	4.194	3185.545	4.634
8	3042.967	4.851	2938.745	5.023
9	2667.697	5.534	2880.996	5.124
10	2500.890	5.903	2884.102	5.118
11	2551.591	5.785	2770.876	5.327
12	2595.601	5.687	2526.719	5.842
13	2671.567	5.526	2529.167	5.837
14	2655.921	5.558	2534.800	5.824
15	2622.129	5.630	2554.864	5.778
16	2580.127	5.721	2556.616	5.774
17	2579.903	5.722	2562.823	5.760
18	2521.513	5.854	3205.826	4.605
19	2485.060	5.940	3033.914	4.866
20	2590.022	5.699	3052.927	4.835
21	2641.486	5.588	3062.676	4.820
22	2551.689	5.785	3102.405	4.758
23	2468.191	5.981	3041.871	4.853
24	2553.058	5.782	3664.053	4.029

Taula A.26: Execucions del programa Matrix Multiply (`MATRIX_SIZE`: 4096, `BLOCK_SIZE`: 64)

workers	Original		DAST	
	ms	speedup	ms	speedup
seq	1272.692	1.000	–	–
1	1257.176	1.012	1267.473	1.004
2	640.030	1.988	651.384	1.954
3	442.659	2.875	439.767	2.894
4	337.788	3.768	347.782	3.659
5	282.713	4.502	283.222	4.494
6	236.784	5.375	241.209	5.276
7	209.685	6.070	208.000	6.119
8	179.906	7.074	183.984	6.917
9	165.348	7.697	166.877	7.627
10	151.580	8.396	155.219	8.199
11	140.277	9.073	142.391	8.938
12	131.511	9.677	136.109	9.351
13	131.377	9.687	133.110	9.561
14	133.024	9.567	134.172	9.486
15	130.374	9.762	134.683	9.450
16	128.880	9.875	133.721	9.517
17	127.103	10.013	130.589	9.746
18	127.619	9.973	128.614	9.895
19	126.365	10.072	126.362	10.072
20	123.118	10.337	126.183	10.086
21	120.012	10.605	124.750	10.202
22	120.152	10.592	122.852	10.360
23	118.966	10.698	121.640	10.463
24	119.284	10.669	159.360	7.986

Taula A.27: Execucions del programa Matrix Multiply (`MATRIX_SIZE`: 2048, `BLOCK_SIZE`: 128)

workers	Original		DAST	
	ms	speedup	ms	speedup
seq	9996.642	1.000	–	–
1	10063.629	0.993	10046.650	0.995
2	5115.966	1.954	5169.470	1.934
3	3553.850	2.813	3511.829	2.847
4	2746.903	3.639	2818.225	3.547
5	2333.309	4.284	2299.330	4.348
6	1971.257	5.071	1993.692	5.014
7	1723.064	5.802	1729.303	5.781
8	1499.736	6.666	1484.768	6.733
9	1347.829	7.417	1327.560	7.530
10	1240.098	8.061	1236.562	8.084
11	1158.469	8.629	1131.492	8.835
12	1078.691	9.267	1068.435	9.356
13	1091.041	9.162	1045.438	9.562
14	1071.051	9.333	1044.580	9.570
15	1069.317	9.349	1029.851	9.707
16	1052.729	9.496	1028.111	9.723
17	1045.663	9.560	1021.729	9.784
18	1044.586	9.570	1016.468	9.835
19	1026.167	9.742	1011.623	9.882
20	1012.109	9.877	993.615	10.061
21	1004.078	9.956	983.072	10.169
22	1015.939	9.840	963.679	10.373
23	1010.701	9.891	973.786	10.266
24	1007.637	9.921	964.610	10.363

Taula A.28: Execucions del programa Matrix Multiply (`MATRIX_SIZE`: 4096, `BLOCK_SIZE`: 128)

workers	Original		DAST	
	ms	speedup	ms	speedup
seq	1098.630	1.000	–	–
1	1069.807	1.027	1076.965	1.020
2	542.155	2.026	554.399	1.982
3	389.533	2.820	387.387	2.836
4	294.151	3.735	298.434	3.681
5	243.276	4.516	243.793	4.506
6	209.084	5.254	205.515	5.346
7	183.670	5.982	177.697	6.183
8	158.639	6.925	161.686	6.795
9	147.505	7.448	149.053	7.371
10	135.462	8.110	134.964	8.140
11	119.451	9.197	120.315	9.131
12	116.529	9.428	122.567	8.964
13	120.436	9.122	127.177	8.639
14	117.083	9.383	124.834	8.801
15	119.294	9.209	128.405	8.556
16	118.394	9.279	126.113	8.711
17	121.250	9.061	128.898	8.523
18	121.013	9.079	122.024	9.003
19	121.400	9.050	124.149	8.849
20	120.104	9.147	125.229	8.773
21	118.420	9.277	124.647	8.814
22	117.307	9.365	120.193	9.141
23	121.189	9.065	121.163	9.067
24	119.604	9.186	144.943	7.580

Taula A.29: Execucions del programa Matrix Multiply (`MATRIX_SIZE`: 2048, `BLOCK_SIZE`: 256)

workers	Original		DAST	
	ms	speedup	ms	speedup
seq	8517.091	1.000	–	–
1	8537.436	0.998	8597.065	0.991
2	4416.967	1.928	4469.522	1.906
3	3082.656	2.763	3092.147	2.754
4	2353.700	3.619	2434.645	3.498
5	1955.019	4.357	1958.227	4.349
6	1682.609	5.062	1672.677	5.092
7	1396.147	6.100	1408.495	6.047
8	1241.136	6.862	1247.665	6.826
9	1104.137	7.714	1120.858	7.599
10	1015.704	8.385	1020.776	8.344
11	941.732	9.044	941.133	9.050
12	863.950	9.858	890.214	9.567
13	876.412	9.718	901.813	9.444
14	878.362	9.697	905.343	9.408
15	869.644	9.794	915.502	9.303
16	883.783	9.637	925.376	9.204
17	878.704	9.693	926.409	9.194
18	896.115	9.504	942.106	9.040
19	893.240	9.535	932.247	9.136
20	887.357	9.598	919.747	9.260
21	876.275	9.720	923.343	9.224
22	876.123	9.721	919.457	9.263
23	882.890	9.647	925.790	9.200
24	888.035	9.591	908.442	9.375

Taula A.30: Execucions del programa Matrix Multiply (`MATRIX_SIZE`: 4096, `BLOCK_SIZE`: 256)

workers	DAST ORIG		DAST STATIC	
	ms	speedup	ms	speedup
seq	1767.661	1.000	1799.235	1.000
1	1851.926	0.954	1856.265	0.969
2	936.611	1.887	943.717	1.907
3	635.483	2.782	1007.763	1.785
4	498.675	3.545	687.924	2.615
5	405.022	4.364	551.845	3.260
6	369.862	4.779	450.601	3.993
7	349.171	5.062	391.776	4.593
8	339.260	5.210	360.672	4.989
9	337.650	5.235	317.816	5.661
10	333.017	5.308	321.022	5.605
11	326.738	5.410	270.803	6.644
12	290.925	6.076	279.396	6.440
13	285.906	6.183	244.416	7.361
14	287.434	6.150	243.183	7.399
15	285.676	6.188	243.423	7.391
16	282.332	6.261	244.836	7.349
17	278.977	6.336	255.747	7.035
18	328.859	5.375	241.010	7.465
19	326.441	5.415	242.428	7.422
20	322.332	5.484	245.632	7.325
21	320.605	5.514	247.035	7.283
22	318.379	5.552	249.551	7.210
23	315.456	5.604	237.686	7.570
24	463.933	3.810	312.642	5.755

Taula A.31: Execucions del programa Matrix Multiply canviant el 'mapeig' del DAST (`MATRIX_SIZE`: 2048, `BLOCK_SIZE`: 64)

workers	DAST ORIG		DAST STATIC	
	ms	speedup	ms	speedup
seq	14761.785	1.000	14778.178	1.000
1	16201.117	0.911	16253.774	0.909
2	8229.603	1.794	8307.464	1.779
3	5451.386	2.708	8992.343	1.643
4	4243.665	3.479	6098.395	2.423
5	3469.817	4.254	4788.427	3.086
6	3420.422	4.316	4077.815	3.624
7	3185.545	4.634	3561.386	4.150
8	2938.745	5.023	2995.523	4.933
9	2880.996	5.124	2664.584	5.546
10	2884.102	5.118	2439.861	6.057
11	2770.876	5.327	2320.034	6.370
12	2526.719	5.842	2287.433	6.461
13	2529.167	5.837	2132.669	6.929
14	2534.800	5.824	2085.563	7.086
15	2554.864	5.778	2070.923	7.136
16	2556.616	5.774	2102.241	7.030
17	2562.823	5.760	2062.709	7.164
18	3205.826	4.605	2079.718	7.106
19	3033.914	4.866	2114.506	6.989
20	3052.927	4.835	2112.739	6.995
21	3062.676	4.820	2119.639	6.972
22	3102.405	4.758	2126.273	6.950
23	3041.871	4.853	2153.760	6.862
24	3664.053	4.029	2344.907	6.302

Taula A.32: Execucions del programa Matrix Multiply canviant el 'mapeig' del DAST (MATRIX_SIZE: 4096, BLOCK_SIZE: 64)

workers	Original		DAST	
	seconds	speedup	seconds	speedup
seq	0.175	1.000	–	–
1	0.229	0.765	0.221	0.792
2	0.099	1.760	0.103	1.698
3	0.070	2.496	0.070	2.495
4	0.054	3.251	0.056	3.130
5	0.045	3.857	0.046	3.831
6	0.040	4.428	0.040	4.383
7	0.035	4.958	0.036	4.846
8	0.032	5.413	0.034	5.185
9	0.031	5.709	0.032	5.421
10	0.029	6.012	0.031	5.720
11	0.027	6.559	0.029	6.021
12	0.025	6.922	0.032	5.397
13	0.028	6.148	0.033	5.285
14	0.031	5.565	0.034	5.186
15	0.033	5.236	0.036	4.856
16	0.033	5.290	0.034	5.210
17	0.032	5.534	0.036	4.882
18	0.032	5.451	0.033	5.305
19	0.032	5.470	0.036	4.853
20	0.034	5.188	0.036	4.909
21	0.034	5.119	0.037	4.775
22	0.032	5.388	0.037	4.688
23	0.035	5.026	0.037	4.765
24	0.036	4.823	0.062	2.837

Taula A.33: Execucions del programa Sparse LU (NB: 16, BSIZE: 50)

workers	Original		DAST	
	seconds	speedup	seconds	speedup
seq	1.237	1.000	–	–
1	1.542	0.802	1.569	0.788
2	0.777	1.591	0.801	1.544
3	0.537	2.303	0.538	2.297
4	0.406	3.046	0.420	2.945
5	0.338	3.662	0.337	3.671
6	0.283	4.363	0.283	4.364
7	0.244	5.076	0.241	5.127
8	0.212	5.841	0.212	5.831
9	0.190	6.507	0.190	6.519
10	0.172	7.204	0.173	7.132
11	0.159	7.781	0.161	7.696
12	0.148	8.371	0.153	8.092
13	0.149	8.324	0.155	7.987
14	0.150	8.244	0.156	7.923
15	0.152	8.133	0.156	7.912
16	0.155	7.979	0.157	7.882
17	0.152	8.162	0.156	7.940
18	0.150	8.231	0.153	8.108
19	0.152	8.116	0.154	8.022
20	0.151	8.209	0.155	7.989
21	0.152	8.116	0.152	8.143
22	0.151	8.179	0.153	8.085
23	0.150	8.238	0.154	8.049
24	0.154	8.043	0.215	5.755

Taula A.34: Execucions del programa Sparse LU (NB: 32, BSIZE: 50)

workers	Original		DAST	
	seconds	speedup	seconds	speedup
seq	9.670	1.000	–	–
1	12.272	0.788	12.348	0.783
2	6.192	1.562	6.379	1.516
3	4.266	2.267	4.278	2.261
4	3.224	2.999	3.331	2.904
5	2.672	3.620	2.662	3.633
6	2.229	4.339	2.227	4.343
7	1.894	5.106	1.892	5.110
8	1.646	5.875	1.659	5.829
9	1.474	6.563	1.473	6.565
10	1.328	7.282	1.343	7.201
11	1.226	7.888	1.223	7.910
12	1.127	8.580	1.158	8.351
13	1.124	8.605	1.159	8.345
14	1.121	8.630	1.154	8.380
15	1.123	8.615	1.149	8.418
16	1.117	8.661	1.149	8.415
17	1.116	8.662	1.145	8.443
18	1.113	8.687	1.138	8.500
19	1.110	8.711	1.134	8.529
20	1.107	8.733	1.132	8.546
21	1.106	8.745	1.128	8.573
22	1.103	8.768	1.127	8.577
23	1.101	8.785	1.123	8.609
24	1.099	8.799	1.150	8.407

Taula A.35: Execucions del programa Sparse LU (NB: 64, BSIZE: 50)

workers	Original		DAST	
	seconds	speedup	seconds	speedup
seq	0.539	1.000	–	–
1	0.688	0.783	0.684	0.788
2	0.342	1.577	0.353	1.527
3	0.238	2.266	0.238	2.267
4	0.188	2.863	0.190	2.836
5	0.156	3.462	0.155	3.481
6	0.135	3.994	0.138	3.899
7	0.120	4.501	0.118	4.549
8	0.107	5.051	0.107	5.049
9	0.094	5.748	0.096	5.616
10	0.087	6.187	0.091	5.952
11	0.084	6.429	0.084	6.405
12	0.083	6.511	0.089	6.034
13	0.093	5.823	0.092	5.851
14	0.097	5.570	0.095	5.702
15	0.101	5.326	0.100	5.385
16	0.103	5.237	0.103	5.208
17	0.104	5.205	0.100	5.366
18	0.107	5.035	0.098	5.489
19	0.105	5.154	0.102	5.275
20	0.104	5.199	0.099	5.461
21	0.103	5.242	0.105	5.119
22	0.104	5.167	0.105	5.137
23	0.104	5.200	0.105	5.144
24	0.105	5.134	0.161	3.351

Taula A.36: Execucions del programa Sparse LU (NB: 16, BSIZE: 75)

workers	Original		DAST	
	seconds	speedup	seconds	speedup
seq	4.093	1.000	–	–
1	5.293	0.773	5.346	0.766
2	2.665	1.536	2.762	1.482
3	1.849	2.214	1.857	2.204
4	1.397	2.931	1.442	2.838
5	1.159	3.530	1.158	3.534
6	0.973	4.208	0.967	4.231
7	0.827	4.950	0.826	4.955
8	0.722	5.672	0.725	5.643
9	0.645	6.341	0.646	6.335
10	0.582	7.032	0.593	6.904
11	0.543	7.531	0.547	7.484
12	0.504	8.125	0.523	7.828
13	0.511	8.015	0.526	7.775
14	0.516	7.933	0.533	7.680
15	0.519	7.885	0.529	7.737
16	0.514	7.959	0.526	7.788
17	0.519	7.881	0.522	7.845
18	0.514	7.959	0.519	7.886
19	0.518	7.903	0.523	7.832
20	0.511	8.002	0.533	7.674
21	0.514	7.957	0.519	7.890
22	0.516	7.932	0.519	7.891
23	0.518	7.904	0.526	7.782
24	0.517	7.914	0.583	7.026

Taula A.37: Execucions del programa Sparse LU (NB: 32, BSIZE: 75)

workers	Original		DAST	
	seconds	speedup	seconds	speedup
seq	32.747	1.000	–	–
1	42.144	0.777	42.499	0.771
2	21.230	1.542	22.426	1.460
3	14.644	2.236	14.728	2.223
4	11.067	2.959	11.450	2.860
5	9.173	3.570	9.161	3.575
6	7.653	4.279	7.640	4.286
7	6.487	5.048	6.487	5.048
8	5.643	5.803	5.685	5.760
9	5.051	6.483	5.056	6.477
10	4.550	7.197	4.608	7.107
11	4.204	7.790	4.198	7.801
12	3.859	8.486	3.989	8.208
13	3.854	8.497	3.973	8.241
14	3.849	8.508	3.971	8.245
15	3.840	8.528	3.952	8.287
16	3.832	8.546	3.920	8.353
17	3.809	8.597	3.917	8.360
18	3.801	8.614	3.897	8.404
19	3.783	8.655	3.863	8.477
20	3.782	8.659	3.859	8.486
21	3.754	8.723	3.847	8.512
22	3.740	8.755	3.836	8.537
23	3.747	8.739	3.817	8.579
24	3.731	8.777	3.799	8.620

Taula A.38: Execucions del programa Sparse LU (`NB`: 64, `BSIZE`: 75)

Apèndix B

Temps d'execució als MIC

cores	workers per core	Original cycles	Original speedup	DAST cycles	DAST speedup
	seq	1.729×10^9	1.000	–	–
1	1	2.118×10^9	0.816	2.559×10^9	0.675
1	2	1.788×10^9	0.966	2.061×10^9	0.838
1	3	1.753×10^9	0.986	1.926×10^9	0.897
1	4	1.748×10^9	0.989	1.876×10^9	0.921
2	1	1.151×10^9	1.501	1.131×10^9	1.528
2	2	9.093×10^8	1.900	9.316×10^8	1.855
2	3	8.852×10^8	1.952	9.069×10^8	1.905
2	4	8.827×10^8	1.958	9.600×10^8	1.800
4	1	6.046×10^8	2.858	5.395×10^8	3.203
4	2	4.818×10^8	3.587	4.496×10^8	3.844
4	3	4.655×10^8	3.713	4.443×10^8	3.890
4	4	4.469×10^8	3.868	5.777×10^8	2.992
8	1	3.129×10^8	5.524	2.692×10^8	6.420
8	2	2.373×10^8	7.283	2.262×10^8	7.640
8	3	2.269×10^8	7.618	2.453×10^8	7.460
8	4	2.599×10^8	6.650	5.571×10^8	3.102
16	1	1.663×10^8	10.391	1.466×10^8	11.790
16	2	1.884×10^8	9.176	1.955×10^8	8.842
16	3	2.323×10^8	7.440	2.852×10^8	6.590
16	4	3.213×10^8	5.379	7.167×10^8	2.411
32	1	2.240×10^8	7.717	1.585×10^8	10.907
32	2	2.349×10^8	7.358	6.408×10^8	2.697
32	3	7.350×10^8	2.351	5.794×10^8	2.983
32	4	1.116×10^9	1.548	1.830×10^9	0.944
60	1	2.069×10^8	8.353	2.152×10^8	8.330
60	2	1.617×10^9	1.690	1.338×10^9	1.291
60	3	1.269×10^9	1.361	1.845×10^9	0.937
60	4	1.006×10^9	1.717	5.790×10^9	0.298

Taula B.1: Execucions del Test 1 (`PARALLEL_CREATIONS`: 1, `TASK_SIZE`: 25000)

cores	workers per core	Original cycles	Original speedup	DAST cycles	DAST speedup
	seq	3.470×10^9	1.000	–	–
1	1	4.234×10^9	0.819	5.103×10^9	0.679
1	2	3.558×10^9	0.975	4.175×10^9	0.831
1	3	3.528×10^9	0.983	3.879×10^9	0.894
1	4	3.500×10^9	0.991	3.716×10^9	0.933
2	1	2.266×10^9	1.531	2.342×10^9	1.481
2	2	1.815×10^9	1.912	1.894×10^9	1.832
2	3	1.770×10^9	1.960	1.821×10^9	1.905
2	4	1.762×10^9	1.969	1.864×10^9	1.861
4	1	1.177×10^9	2.947	1.112×10^9	3.120
4	2	9.216×10^8	3.765	9.069×10^8	3.826
4	3	8.993×10^8	3.858	8.940×10^8	3.881
4	4	8.918×10^8	3.891	1.104×10^9	3.144
8	1	6.276×10^8	5.528	5.503×10^8	6.305
8	2	4.713×10^8	7.362	4.623×10^8	7.505
8	3	4.584×10^8	7.569	5.271×10^8	6.582
8	4	4.488×10^8	7.731	1.108×10^9	3.132
16	1	3.281×10^8	10.575	3.572×10^8	9.714
16	2	2.403×10^8	14.443	4.312×10^8	8.470
16	3	2.434×10^8	14.258	6.111×10^8	5.678
16	4	3.391×10^8	10.233	1.629×10^9	2.130
32	1	2.273×10^8	15.264	4.015×10^8	8.643
32	2	3.108×10^8	11.165	1.204×10^9	2.881
32	3	1.103×10^9	3.144	1.310×10^9	2.649
32	4	1.771×10^9	1.958	4.206×10^9	0.825
60	1	3.027×10^8	11.464	5.119×10^8	6.778
60	2	2.769×10^9	1.253	2.774×10^9	1.250
60	3	2.349×10^9	1.477	3.584×10^9	0.968
60	4	3.183×10^9	1.900	1.066×10^{10}	0.325

Taula B.2: Execucions del Test 1 (PARALLEL_CREATIONS: 2, TASK_SIZE: 25000)

cores	workers per core	Original cycles	Original speedup	DAST cycles	DAST speedup
	seq	6.917×10^9	1.000	–	–
1	1	8.003×10^9	0.864	1.001×10^{10}	0.690
1	2	6.766×10^9	1.220	8.173×10^9	0.846
1	3	6.716×10^9	1.290	7.634×10^9	0.906
1	4	6.704×10^9	1.310	7.158×10^9	0.966
2	1	4.071×10^9	1.699	4.412×10^9	1.567
2	2	3.394×10^9	2.380	3.670×10^9	1.884
2	3	3.371×10^9	2.510	3.558×10^9	1.944
2	4	3.364×10^9	2.560	3.507×10^9	1.972
4	1	2.053×10^9	3.368	2.087×10^9	3.314
4	2	1.703×10^9	4.610	1.755×10^9	3.941
4	3	1.695×10^9	4.800	1.729×10^9	4.000
4	4	1.695×10^9	4.800	1.781×10^9	3.883
8	1	1.046×10^9	6.612	1.023×10^9	6.764
8	2	8.705×10^8	7.945	8.641×10^8	8.500
8	3	8.601×10^8	8.410	8.621×10^8	8.230
8	4	8.662×10^8	7.985	9.102×10^8	7.599
16	1	5.542×10^8	12.480	5.206×10^8	13.285
16	2	4.591×10^8	15.660	4.467×10^8	15.483
16	3	4.371×10^8	15.823	4.432×10^8	15.605
16	4	4.374×10^8	15.814	6.343×10^8	10.904
32	1	2.798×10^8	24.722	2.736×10^8	25.284
32	2	2.249×10^8	30.750	2.772×10^8	24.947
32	3	7.343×10^8	9.419	5.682×10^8	12.173
32	4	9.452×10^8	7.317	1.658×10^9	4.172
60	1	1.868×10^8	37.250	2.068×10^8	33.443
60	2	1.727×10^9	4.400	1.369×10^9	5.500
60	3	1.496×10^9	4.623	1.700×10^9	4.690
60	4	1.792×10^9	3.859	5.345×10^9	1.293

Taula B.3: Execucions del Test 1 (`PARALLEL_CREATIONS`: 1, `TASK_SIZE`: 100000)

cores	workers per core	Original cycles	Original speedup	DAST cycles	DAST speedup
	seq	1.388×10^{10}	1.000	–	–
1	1	1.606×10^{10}	0.864	2.001×10^{10}	0.693
1	2	1.354×10^{10}	1.240	1.635×10^{10}	0.849
1	3	1.346×10^{10}	1.310	1.527×10^{10}	0.909
1	4	1.341×10^{10}	1.350	1.432×10^{10}	0.969
2	1	8.178×10^{9}	1.697	8.911×10^{9}	1.557
2	2	6.791×10^{9}	2.440	7.349×10^{9}	1.888
2	3	6.744×10^{9}	2.580	7.121×10^{9}	1.949
2	4	6.724×10^{9}	2.640	7.127×10^{9}	1.948
4	1	4.116×10^{9}	3.372	4.205×10^{9}	3.300
4	2	3.404×10^{9}	4.770	3.507×10^{9}	3.958
4	3	3.382×10^{9}	4.104	3.456×10^{9}	4.160
4	4	3.379×10^{9}	4.108	3.474×10^{9}	3.995
8	1	2.092×10^{9}	6.634	2.054×10^{9}	6.759
8	2	1.715×10^{9}	8.920	1.737×10^{9}	7.989
8	3	1.706×10^{9}	8.134	1.717×10^{9}	8.840
8	4	1.706×10^{9}	8.134	1.802×10^{9}	7.703
16	1	1.057×10^{9}	13.136	1.026×10^{9}	13.527
16	2	8.706×10^{8}	15.943	8.736×10^{8}	15.890
16	3	8.726×10^{8}	15.907	8.686×10^{8}	15.981
16	4	8.697×10^{8}	15.961	1.177×10^{9}	11.794
32	1	5.668×10^{8}	24.488	5.531×10^{8}	25.960
32	2	4.421×10^{8}	31.401	7.744×10^{8}	17.925
32	3	9.506×10^{8}	14.602	9.557×10^{8}	14.525
32	4	1.509×10^{9}	9.200	3.626×10^{9}	3.828
60	1	3.144×10^{8}	44.152	4.436×10^{8}	31.290
60	2	3.079×10^{9}	4.508	3.059×10^{9}	4.537
60	3	2.207×10^{9}	6.288	3.689×10^{9}	3.763
60	4	3.213×10^{9}	4.320	1.086×10^{10}	1.277

Taula B.4: Execucions del Test 1 (`PARALLEL_CREATIONS`: 2, `TASK_SIZE`: 100000)

cores	workers per core	Original cycles	Original speedup	DAST cycles	DAST speedup
	seq	2.081×10^{10}	1.000	–	–
1	1	2.415×10^{10}	0.861	3.000×10^{10}	0.693
1	2	2.027×10^{10}	1.260	2.451×10^{10}	0.848
1	3	2.021×10^{10}	1.290	2.298×10^{10}	0.905
1	4	2.017×10^{10}	1.310	2.159×10^{10}	0.963
2	1	1.224×10^{10}	1.700	1.334×10^{10}	1.559
2	2	1.032×10^{10}	2.160	1.108×10^{10}	1.878
2	3	1.014×10^{10}	2.510	1.090×10^{10}	1.908
2	4	1.011×10^{10}	2.580	1.050×10^{10}	1.982
4	1	6.345×10^{9}	3.279	6.400×10^{9}	3.251
4	2	5.164×10^{9}	4.290	5.283×10^{9}	3.938
4	3	5.089×10^{9}	4.880	5.193×10^{9}	4.600
4	4	5.074×10^{9}	4.101	5.182×10^{9}	4.140
8	1	3.201×10^{9}	6.499	3.114×10^{9}	6.681
8	2	2.595×10^{9}	8.180	2.592×10^{9}	8.270
8	3	2.553×10^{9}	8.149	2.577×10^{9}	8.730
8	4	2.552×10^{9}	8.153	2.669×10^{9}	7.797
16	1	1.613×10^{9}	12.898	1.555×10^{9}	13.384
16	2	1.313×10^{9}	15.846	1.335×10^{9}	15.590
16	3	1.304×10^{9}	15.960	1.363×10^{9}	15.263
16	4	1.298×10^{9}	16.350	1.801×10^{9}	11.554
32	1	8.561×10^{8}	24.304	9.045×10^{8}	23.400
32	2	6.632×10^{8}	31.371	9.549×10^{8}	21.790
32	3	6.586×10^{8}	31.592	1.150×10^{9}	18.870
32	4	6.669×10^{8}	31.201	5.479×10^{9}	3.797
60	1	4.689×10^{8}	44.369	7.404×10^{8}	28.102
60	2	3.594×10^{9}	5.788	5.091×10^{9}	4.870
60	3	2.490×10^{9}	8.356	5.623×10^{9}	3.700
60	4	4.868×10^{9}	4.274	1.627×10^{10}	1.278

Taula B.5: Execucions del Test 1 (`PARALLEL_CREATIONS`: 3, `TASK_SIZE`: 100000)

cores	workers per core	Original cycles	Original speedup	DAST cycles	DAST speedup
	seq	1.450×10^{10}	1.000	–	–
1	1	2.660×10^{10}	0.545	2.160×10^{10}	0.671
1	2	2.442×10^{10}	0.593	1.690×10^{10}	0.858
1	3	1.443×10^{10}	1.500	1.343×10^{10}	1.790
1	4	1.323×10^{10}	1.960	1.291×10^{10}	1.123
2	1	1.960×10^{10}	0.740	1.394×10^{10}	1.400
2	2	7.322×10^{9}	1.980	6.664×10^{9}	2.176
2	3	6.439×10^{9}	2.252	6.416×10^{9}	2.260
2	4	6.218×10^{9}	2.332	6.407×10^{9}	2.263
4	1	5.172×10^{9}	2.804	3.905×10^{9}	3.714
4	2	3.222×10^{9}	4.501	3.200×10^{9}	4.532
4	3	4.388×10^{9}	3.305	3.998×10^{9}	3.627
4	4	5.553×10^{9}	2.612	4.106×10^{9}	3.532
8	1	2.178×10^{9}	6.660	1.904×10^{9}	7.616
8	2	3.121×10^{9}	4.647	2.951×10^{9}	4.914
8	3	4.406×10^{9}	3.292	4.083×10^{9}	3.552
8	4	5.503×10^{9}	2.635	3.593×10^{9}	4.370
16	1	2.200×10^{9}	6.592	1.802×10^{9}	8.500
16	2	3.164×10^{9}	4.584	2.885×10^{9}	5.260
16	3	4.385×10^{9}	3.308	3.904×10^{9}	3.714
16	4	5.434×10^{9}	2.669	4.266×10^{9}	3.400
32	1	2.570×10^{9}	5.643	1.803×10^{9}	8.420
32	2	3.207×10^{9}	4.522	3.749×10^{9}	3.868
32	3	6.268×10^{9}	2.313	4.328×10^{9}	3.350
32	4	7.449×10^{9}	1.947	1.182×10^{10}	1.226
60	1	2.302×10^{9}	6.300	1.855×10^{9}	7.819
60	2	1.229×10^{10}	1.180	3.477×10^{9}	4.171
60	3	9.937×10^{9}	1.459	1.058×10^{10}	1.370
60	4	8.351×10^{9}	1.736	3.659×10^{10}	0.396

Taula B.6: Execucions del Test 2 (`PARALLEL_CREATIONS`: 1, `TASK_SIZE`: 25000)

cores	workers per core	Original		DAST	
		cycles	speedup	cycles	speedup
	seq	2.904×10^{10}	1.000	–	–
1	1	6.024×10^{10}	0.482	4.339×10^{10}	0.669
1	2	5.170×10^{10}	0.561	3.762×10^{10}	0.771
1	3	4.099×10^{10}	0.708	3.575×10^{10}	0.812
1	4	3.785×10^{10}	0.767	4.828×10^{10}	0.601
2	1	3.910×10^{10}	0.742	3.520×10^{10}	0.824
2	2	2.773×10^{10}	1.470	2.265×10^{10}	1.282
2	3	1.868×10^{10}	1.554	1.288×10^{10}	2.255
2	4	1.416×10^{10}	2.500	1.337×10^{10}	2.171
4	1	2.425×10^{10}	1.197	1.993×10^{10}	1.456
4	2	9.168×10^{9}	3.167	6.446×10^{9}	4.505
4	3	6.522×10^{9}	4.452	6.287×10^{9}	4.619
4	4	6.226×10^{9}	4.664	6.710×10^{9}	4.327
8	1	7.946×10^{9}	3.654	3.942×10^{9}	7.367
8	2	3.253×10^{9}	8.926	3.141×10^{9}	9.245
8	3	4.358×10^{9}	6.663	4.027×10^{9}	7.211
8	4	5.593×10^{9}	5.192	5.885×10^{9}	4.935
16	1	2.327×10^{9}	12.482	1.963×10^{9}	14.794
16	2	3.218×10^{9}	9.250	2.723×10^{9}	10.664
16	3	4.575×10^{9}	6.347	4.059×10^{9}	7.155
16	4	5.919×10^{9}	4.906	8.709×10^{9}	3.334
32	1	2.444×10^{9}	11.884	1.803×10^{9}	16.103
32	2	3.494×10^{9}	8.311	9.103×10^{9}	3.190
32	3	8.868×10^{9}	3.274	7.670×10^{9}	3.786
32	4	1.276×10^{10}	2.275	3.086×10^{10}	0.940
60	1	2.491×10^{9}	11.660	2.801×10^{9}	10.368
60	2	2.161×10^{10}	1.343	1.044×10^{10}	2.781
60	3	1.669×10^{10}	1.740	2.664×10^{10}	1.900
60	4	2.051×10^{10}	1.415	8.513×10^{10}	0.341

Taula B.7: Execucions del Test 2 (PARALLEL_CREATIONS: 2, TASK_SIZE: 25000)

cores	workers per core	Original cycles	Original speedup	DAST cycles	DAST speedup
	seq	5.823×10^{10}	1.000	–	–
1	1	7.046×10^{10}	0.826	6.999×10^{10}	0.832
1	2	5.775×10^{10}	1.800	5.627×10^{10}	1.340
1	3	5.211×10^{10}	1.117	5.346×10^{10}	1.890
1	4	4.979×10^{10}	1.169	5.048×10^{10}	1.153
2	1	3.953×10^{10}	1.473	3.124×10^{10}	1.864
2	2	2.670×10^{10}	2.180	2.604×10^{10}	2.236
2	3	2.457×10^{10}	2.369	2.545×10^{10}	2.288
2	4	2.420×10^{10}	2.406	2.477×10^{10}	2.351
4	1	1.777×10^{10}	3.277	1.511×10^{10}	3.852
4	2	1.235×10^{10}	4.713	1.268×10^{10}	4.592
4	3	1.423×10^{10}	4.920	1.506×10^{10}	3.867
4	4	1.854×10^{10}	3.140	1.525×10^{10}	3.817
8	1	7.617×10^{9}	7.645	7.428×10^{9}	7.839
8	2	9.483×10^{9}	6.140	1.098×10^{10}	5.302
8	3	1.422×10^{10}	4.940	1.518×10^{10}	3.835
8	4	1.797×10^{10}	3.240	1.327×10^{10}	4.388
16	1	6.485×10^{9}	8.979	6.577×10^{9}	8.853
16	2	9.603×10^{9}	6.640	1.050×10^{10}	5.547
16	3	1.419×10^{10}	4.104	1.443×10^{10}	4.340
16	4	1.729×10^{10}	3.368	1.322×10^{10}	4.406
32	1	6.015×10^{9}	9.681	6.589×10^{9}	8.837
32	2	9.632×10^{9}	6.450	1.097×10^{10}	5.307
32	3	1.454×10^{10}	4.400	1.432×10^{10}	4.660
32	4	1.885×10^{10}	3.880	1.380×10^{10}	4.220
60	1	6.009×10^{9}	9.691	6.599×10^{9}	8.824
60	2	1.540×10^{10}	3.780	1.096×10^{10}	5.312
60	3	1.582×10^{10}	3.681	1.361×10^{10}	4.278
60	4	1.832×10^{10}	3.178	4.031×10^{10}	1.444

Taula B.8: Execucions del Test 2 (PARALLEL_CREATIONS: 1, TASK_SIZE: 100000)

cores	workers per core	Original cycles	Original speedup	DAST cycles	DAST speedup
	seq	1.165×10^{11}	1.000	–	–
1	1	1.486×10^{11}	0.784	1.386×10^{11}	0.840
1	2	1.156×10^{11}	1.800	1.129×10^{11}	1.310
1	3	1.038×10^{11}	1.122	1.066×10^{11}	1.920
1	4	1.015×10^{11}	1.148	1.004×10^{11}	1.160
2	1	7.994×10^{10}	1.457	6.194×10^{10}	1.880
2	2	5.823×10^{10}	2.000	5.116×10^{10}	2.277
2	3	5.288×10^{10}	2.202	5.053×10^{10}	2.305
2	4	5.037×10^{10}	2.313	4.974×10^{10}	2.342
4	1	4.125×10^{10}	2.823	3.156×10^{10}	3.691
4	2	2.795×10^{10}	4.167	2.515×10^{10}	4.631
4	3	2.498×10^{10}	4.663	2.474×10^{10}	4.709
4	4	2.424×10^{10}	4.805	2.456×10^{10}	4.743
8	1	1.938×10^{10}	6.900	1.489×10^{10}	7.825
8	2	1.244×10^{10}	9.366	1.241×10^{10}	9.384
8	3	1.412×10^{10}	8.250	1.534×10^{10}	7.595
8	4	1.837×10^{10}	6.340	1.537×10^{10}	7.578
16	1	7.727×10^{9}	15.760	7.401×10^{9}	15.740
16	2	9.709×10^{9}	11.998	1.093×10^{10}	10.659
16	3	1.448×10^{10}	8.470	1.442×10^{10}	8.810
16	4	1.772×10^{10}	6.576	1.376×10^{10}	8.468
32	1	6.087×10^{9}	19.137	7.324×10^{9}	15.906
32	2	9.806×10^{9}	11.880	1.074×10^{10}	10.850
32	3	1.548×10^{10}	7.527	1.549×10^{10}	7.520
32	4	2.059×10^{10}	5.658	2.343×10^{10}	4.972
60	1	6.066×10^{9}	19.205	6.590×10^{9}	17.677
60	2	2.303×10^{10}	5.580	1.075×10^{10}	10.834
60	3	2.158×10^{10}	5.398	1.887×10^{10}	6.173
60	4	2.379×10^{10}	4.897	9.487×10^{10}	1.227

Taula B.9: Execucions del Test 2 (`PARALLEL_CREATIONS`: 2, `TASK_SIZE`: 100000)

cores	workers per core	Original		DAST	
		cycles	speedup	cycles	speedup
	seq	1.720×10^9	1.000	–	–
1	1	1.843×10^9	0.933	2.041×10^9	0.842
1	2	1.071×10^9	1.606	1.216×10^9	1.414
1	3	8.455×10^8	2.340	9.439×10^8	1.822
1	4	7.497×10^8	2.294	9.050×10^8	1.900
2	1	9.306×10^8	1.848	9.640×10^8	1.784
2	2	5.544×10^8	3.102	5.820×10^8	2.955
2	3	4.567×10^8	3.766	4.811×10^8	3.574
2	4	4.211×10^8	4.840	5.538×10^8	3.105
4	1	4.841×10^8	3.553	4.856×10^8	3.542
4	2	3.148×10^8	5.464	3.275×10^8	5.252
4	3	2.841×10^8	6.540	2.911×10^8	5.909
4	4	2.856×10^8	6.220	4.556×10^8	3.775
8	1	2.809×10^8	6.123	2.775×10^8	6.198
8	2	2.230×10^8	7.712	2.249×10^8	7.648
8	3	2.406×10^8	7.148	2.716×10^8	6.331
8	4	2.787×10^8	6.171	4.288×10^8	4.110
16	1	2.071×10^8	8.305	2.061×10^8	8.345
16	2	2.274×10^8	7.565	2.524×10^8	6.815
16	3	2.956×10^8	5.819	3.312×10^8	5.193
16	4	3.636×10^8	4.730	5.428×10^8	3.168
32	1	2.277×10^8	7.552	2.458×10^8	6.996
32	2	3.247×10^8	5.297	3.527×10^8	4.876
32	3	4.271×10^8	4.270	4.821×10^8	3.567
32	4	5.598×10^8	3.720	6.876×10^8	2.501
60	1	3.640×10^8	4.725	3.644×10^8	4.719
60	2	5.210×10^8	3.301	5.526×10^8	3.112
60	3	7.159×10^8	2.402	7.629×10^8	2.254
60	4	9.251×10^8	1.859	9.504×10^8	1.809

Taula B.10: Execucions del programa Cholesky (`MATRIX_SIZE`: 2048, `BLOCK_SIZE`: 128)

cores	workers per core	Original		DAST	
		cycles	speedup	cycles	speedup
	seq	1.659×10^9	1.000	–	–
1	1	1.698×10^9	0.977	1.761×10^9	0.942
1	2	9.917×10^8	1.673	1.130×10^9	1.468
1	3	8.226×10^8	2.170	9.416×10^8	1.762
1	4	7.654×10^8	2.167	9.588×10^8	1.730
2	1	8.837×10^8	1.877	9.036×10^8	1.836
2	2	5.533×10^8	2.998	5.951×10^8	2.788
2	3	4.902×10^8	3.384	5.223×10^8	3.177
2	4	4.873×10^8	3.405	6.041×10^8	2.746
4	1	5.044×10^8	3.290	5.051×10^8	3.284
4	2	3.701×10^8	4.484	3.748×10^8	4.427
4	3	3.822×10^8	4.341	4.232×10^8	3.920
4	4	4.375×10^8	3.792	7.070×10^8	2.346
8	1	3.365×10^8	4.930	3.356×10^8	4.945
8	2	3.182×10^8	5.215	3.359×10^8	4.940
8	3	3.940×10^8	4.211	4.302×10^8	3.857
8	4	4.571×10^8	3.630	5.419×10^8	3.620
16	1	2.873×10^8	5.775	3.021×10^8	5.492
16	2	3.491×10^8	4.752	3.641×10^8	4.557
16	3	4.130×10^8	4.170	4.243×10^8	3.911
16	4	4.647×10^8	3.570	5.578×10^8	2.975
32	1	3.208×10^8	5.172	3.347×10^8	4.957
32	2	3.527×10^8	4.704	3.644×10^8	4.553
32	3	3.972×10^8	4.178	4.256×10^8	3.899
32	4	4.771×10^8	3.478	5.818×10^8	2.852
60	1	3.212×10^8	5.166	3.276×10^8	5.640
60	2	3.614×10^8	4.591	3.544×10^8	4.682
60	3	4.219×10^8	3.933	4.271×10^8	3.885
60	4	4.774×10^8	3.476	5.948×10^8	2.790

Taula B.11: Execucions del programa Cholesky (`MATRIX_SIZE`: 2048, `BLOCK_SIZE`: 256)

cores	workers per core	Original cycles	Original speedup	DAST cycles	DAST speedup
	seq	1.512×10^9	1.000	–	–
1	1	1.528×10^9	0.989	1.714×10^9	0.882
1	2	1.070×10^9	1.413	1.240×10^9	1.219
1	3	1.091×10^9	1.385	1.235×10^9	1.224
1	4	1.248×10^9	1.211	1.282×10^9	1.179
2	1	9.073×10^8	1.666	9.349×10^8	1.617
2	2	9.089×10^8	1.664	1.022×10^9	1.480
2	3	1.078×10^9	1.403	1.212×10^9	1.247
2	4	1.240×10^9	1.220	1.260×10^9	1.199
4	1	7.654×10^8	1.976	8.398×10^8	1.800
4	2	8.991×10^8	1.682	1.026×10^9	1.474
4	3	1.041×10^9	1.452	1.200×10^9	1.260
4	4	1.230×10^9	1.229	1.264×10^9	1.196
8	1	7.660×10^8	1.974	8.376×10^8	1.805
8	2	8.992×10^8	1.681	1.025×10^9	1.476
8	3	1.074×10^9	1.408	1.190×10^9	1.271
8	4	1.224×10^9	1.235	1.228×10^9	1.231
16	1	7.650×10^8	1.976	8.375×10^8	1.805
16	2	8.984×10^8	1.683	9.998×10^8	1.512
16	3	1.073×10^9	1.409	1.195×10^9	1.265
16	4	1.241×10^9	1.218	1.256×10^9	1.204
32	1	7.598×10^8	1.990	8.289×10^8	1.824
32	2	8.931×10^8	1.693	1.002×10^9	1.509
32	3	1.075×10^9	1.406	1.168×10^9	1.294
32	4	1.223×10^9	1.236	1.225×10^9	1.234
60	1	7.563×10^8	1.999	8.158×10^8	1.853
60	2	8.877×10^8	1.703	9.974×10^8	1.516
60	3	1.008×10^9	1.499	1.186×10^9	1.275
60	4	1.187×10^9	1.274	1.240×10^9	1.219

Taula B.12: Execucions del programa Cholesky (`MATRIX_SIZE`: 2048, `BLOCK_SIZE`: 512)

cores	workers per core	Original cycles	Original speedup	DAST cycles	DAST speedup
	seq	1.228×10^{10}	1.000	–	–
1	1	1.344×10^{10}	0.913	1.475×10^{10}	0.832
1	2	7.651×10^{9}	1.604	8.528×10^{9}	1.439
1	3	5.902×10^{9}	2.800	6.515×10^{9}	1.884
1	4	5.133×10^{9}	2.392	5.676×10^{9}	2.163
2	1	6.753×10^{9}	1.818	6.866×10^{9}	1.788
2	2	3.845×10^{9}	3.193	3.974×10^{9}	3.890
2	3	2.971×10^{9}	4.132	3.058×10^{9}	4.150
2	4	2.610×10^{9}	4.705	2.799×10^{9}	4.386
4	1	3.381×10^{9}	3.631	3.333×10^{9}	3.684
4	2	1.965×10^{9}	6.249	1.948×10^{9}	6.304
4	3	1.571×10^{9}	7.817	1.544×10^{9}	7.951
4	4	1.425×10^{9}	8.613	1.657×10^{9}	7.410
8	1	1.743×10^{9}	7.420	1.671×10^{9}	7.345
8	2	1.086×10^{9}	11.301	1.054×10^{9}	11.647
8	3	9.196×10^{8}	13.352	9.196×10^{8}	13.352
8	4	8.310×10^{8}	14.776	1.438×10^{9}	8.541
16	1	9.852×10^{8}	12.463	8.978×10^{8}	13.676
16	2	6.673×10^{8}	18.400	7.272×10^{8}	16.885
16	3	6.720×10^{8}	18.270	9.528×10^{8}	12.887
16	4	7.968×10^{8}	15.409	1.240×10^{9}	9.905
32	1	6.123×10^{8}	20.530	6.554×10^{8}	18.735
32	2	6.717×10^{8}	18.279	8.701×10^{8}	14.111
32	3	8.053×10^{8}	15.248	1.165×10^{9}	10.540
32	4	9.705×10^{8}	12.651	1.609×10^{9}	7.631
60	1	6.604×10^{8}	18.591	7.266×10^{8}	16.898
60	2	8.551×10^{8}	14.359	1.214×10^{9}	10.114
60	3	1.094×10^{9}	11.221	1.765×10^{9}	6.956
60	4	1.404×10^{9}	8.748	2.251×10^{9}	5.454

Taula B.13: Execucions del programa Cholesky (MATRIX_SIZE: 4096, BLOCK_SIZE: 128)

cores	workers per core	Original cycles	Original speedup	DAST cycles	DAST speedup
	seq	1.198×10^{10}	1.000	–	–
1	1	1.222×10^{10}	0.980	1.246×10^{10}	0.961
1	2	6.807×10^{9}	1.760	7.659×10^{9}	1.564
1	3	5.458×10^{9}	2.195	6.169×10^{9}	1.942
1	4	4.753×10^{9}	2.521	5.595×10^{9}	2.142
2	1	6.148×10^{9}	1.949	6.201×10^{9}	1.932
2	2	3.456×10^{9}	3.468	3.657×10^{9}	3.277
2	3	2.806×10^{9}	4.271	2.991×10^{9}	4.600
2	4	2.479×10^{9}	4.833	2.919×10^{9}	4.106
4	1	3.149×10^{9}	3.805	3.162×10^{9}	3.789
4	2	1.819×10^{9}	6.589	1.914×10^{9}	6.263
4	3	1.545×10^{9}	7.757	1.621×10^{9}	7.392
4	4	1.446×10^{9}	8.286	1.663×10^{9}	7.208
8	1	1.656×10^{9}	7.237	1.646×10^{9}	7.279
8	2	1.060×10^{9}	11.309	1.077×10^{9}	11.131
8	3	9.853×10^{8}	12.163	1.041×10^{9}	11.514
8	4	1.040×10^{9}	11.518	1.248×10^{9}	9.604
16	1	9.554×10^{8}	12.544	9.676×10^{8}	12.386
16	2	7.580×10^{8}	15.811	7.868×10^{8}	15.231
16	3	8.754×10^{8}	13.690	9.470×10^{8}	12.655
16	4	1.052×10^{9}	11.387	1.145×10^{9}	10.462
32	1	7.013×10^{8}	17.890	7.073×10^{8}	16.945
32	2	7.852×10^{8}	15.263	7.907×10^{8}	15.157
32	3	9.797×10^{8}	12.233	9.710×10^{8}	12.343
32	4	1.180×10^{9}	10.159	1.267×10^{9}	9.461
60	1	7.657×10^{8}	15.651	7.544×10^{8}	15.885
60	2	9.449×10^{8}	12.683	9.299×10^{8}	12.888
60	3	1.197×10^{9}	10.110	1.229×10^{9}	9.750
60	4	1.355×10^{9}	8.847	1.579×10^{9}	7.589

Taula B.14: Execucions del programa Cholesky (`MATRIX_SIZE`: 4096, `BLOCK_SIZE`: 256)

cores	workers per core	Original cycles	Original speedup	DAST cycles	DAST speedup
	seq	8.783×10^9	1.000	–	–
1	1	8.808×10^9	0.997	1.004×10^{10}	0.874
1	2	5.759×10^9	1.525	6.701×10^9	1.310
1	3	5.170×10^9	1.698	5.843×10^9	1.503
1	4	5.169×10^9	1.699	6.486×10^9	1.354
2	1	4.679×10^9	1.877	4.997×10^9	1.757
2	2	3.175×10^9	2.765	3.406×10^9	2.578
2	3	3.025×10^9	2.903	3.298×10^9	2.663
2	4	3.410×10^9	2.575	3.683×10^9	2.384
4	1	2.610×10^9	3.365	2.683×10^9	3.273
4	2	2.119×10^9	4.145	2.223×10^9	3.951
4	3	2.592×10^9	3.388	2.799×10^9	3.138
4	4	3.060×10^9	2.870	2.913×10^9	3.140
8	1	1.750×10^9	5.190	1.838×10^9	4.777
8	2	2.064×10^9	4.255	2.263×10^9	3.881
8	3	2.586×10^9	3.395	2.727×10^9	3.221
8	4	3.035×10^9	2.894	2.764×10^9	3.177
16	1	1.697×10^9	5.176	1.781×10^9	4.931
16	2	2.032×10^9	4.322	2.291×10^9	3.833
16	3	2.552×10^9	3.441	2.693×10^9	3.260
16	4	3.072×10^9	2.859	2.798×10^9	3.139
32	1	1.703×10^9	5.158	1.878×10^9	4.676
32	2	2.008×10^9	4.373	2.243×10^9	3.916
32	3	2.501×10^9	3.511	2.599×10^9	3.379
32	4	2.938×10^9	2.989	2.740×10^9	3.205
60	1	1.713×10^9	5.126	1.852×10^9	4.742
60	2	2.033×10^9	4.319	2.141×10^9	4.102
60	3	2.524×10^9	3.479	2.692×10^9	3.262
60	4	2.957×10^9	2.970	2.723×10^9	3.225

Taula B.15: Execucions del programa Cholesky (`MATRIX_SIZE`: 4096, `BLOCK_SIZE`: 512)

cores	workers per core	Original cycles	Original speedup	DAST cycles	DAST speedup
	seq	9.375×10^{10}	1.000	–	–
1	1	1.038×10^{11}	0.902	1.134×10^{11}	0.826
1	2	5.893×10^{10}	1.590	6.599×10^{10}	1.420
1	3	4.504×10^{10}	2.810	4.915×10^{10}	1.907
1	4	3.888×10^{10}	2.410	4.228×10^{10}	2.217
2	1	5.261×10^{10}	1.782	5.296×10^{10}	1.770
2	2	2.934×10^{10}	3.195	2.987×10^{10}	3.138
2	3	2.280×10^{10}	4.110	2.282×10^{10}	4.108
2	4	2.000×10^{10}	4.688	2.094×10^{10}	4.476
4	1	2.654×10^{10}	3.531	2.555×10^{10}	3.669
4	2	1.532×10^{10}	6.118	1.459×10^{10}	6.425
4	3	1.196×10^{10}	7.836	1.161×10^{10}	8.720
4	4	1.052×10^{10}	8.914	1.163×10^{10}	8.590
8	1	1.376×10^{10}	6.813	1.250×10^{10}	7.496
8	2	8.162×10^{9}	11.484	7.883×10^{9}	11.892
8	3	6.316×10^{9}	14.843	6.717×10^{9}	13.955
8	4	5.127×10^{9}	18.283	7.417×10^{9}	12.639
16	1	7.482×10^{9}	12.529	6.709×10^{9}	13.973
16	2	4.008×10^{9}	23.391	5.093×10^{9}	18.405
16	3	4.033×10^{9}	23.245	6.055×10^{9}	15.482
16	4	4.647×10^{9}	20.171	8.262×10^{9}	11.346
32	1	3.584×10^{9}	26.159	4.237×10^{9}	22.124
32	2	3.759×10^{9}	24.939	5.491×10^{9}	17.710
32	3	4.257×10^{9}	22.210	6.901×10^{9}	13.584
32	4	4.884×10^{9}	19.193	9.630×10^{9}	9.734
60	1	3.485×10^{9}	26.899	4.152×10^{9}	22.578
60	2	3.885×10^{9}	24.128	6.741×10^{9}	13.907
60	3	4.661×10^{9}	20.113	9.418×10^{9}	9.953
60	4	5.336×10^{9}	17.568	1.249×10^{10}	7.507

Taula B.16: Execucions del programa Cholesky (`MATRIX_SIZE`: 8192, `BLOCK_SIZE`: 128)

cores	workers per core	Original cycles	Original speedup	DAST cycles	DAST speedup
	seq	9.130×10^{10}	1.000	–	–
1	1	9.286×10^{10}	0.983	9.598×10^{10}	0.951
1	2	5.043×10^{10}	1.810	5.666×10^{10}	1.611
1	3	4.056×10^{10}	2.250	4.571×10^{10}	1.997
1	4	3.502×10^{10}	2.607	3.955×10^{10}	2.308
2	1	4.659×10^{10}	1.959	4.681×10^{10}	1.950
2	2	2.516×10^{10}	3.629	2.792×10^{10}	3.270
2	3	2.029×10^{10}	4.499	2.205×10^{10}	4.140
2	4	1.739×10^{10}	5.248	1.854×10^{10}	4.923
4	1	2.356×10^{10}	3.875	2.335×10^{10}	3.910
4	2	1.274×10^{10}	7.167	1.302×10^{10}	7.100
4	3	1.034×10^{10}	8.825	1.055×10^{10}	8.649
4	4	8.968×10^{9}	10.180	9.270×10^{9}	9.849
8	1	1.182×10^{10}	7.724	1.182×10^{10}	7.727
8	2	6.533×10^{9}	13.974	6.686×10^{9}	13.654
8	3	5.492×10^{9}	16.624	5.506×10^{9}	16.580
8	4	4.858×10^{9}	18.792	5.086×10^{9}	17.951
16	1	6.054×10^{9}	15.810	6.011×10^{9}	15.187
16	2	3.534×10^{9}	25.833	3.576×10^{9}	25.527
16	3	3.180×10^{9}	28.713	3.216×10^{9}	28.387
16	4	3.109×10^{9}	29.365	3.413×10^{9}	26.748
32	1	3.256×10^{9}	28.390	3.283×10^{9}	27.809
32	2	2.247×10^{9}	40.636	2.235×10^{9}	40.844
32	3	2.319×10^{9}	39.377	2.386×10^{9}	38.262
32	4	2.512×10^{9}	36.349	2.771×10^{9}	32.953
60	1	2.145×10^{9}	42.565	2.136×10^{9}	42.743
60	2	1.989×10^{9}	45.912	1.963×10^{9}	46.498
60	3	2.367×10^{9}	38.565	2.403×10^{9}	37.999
60	4	2.754×10^{9}	33.155	3.292×10^{9}	27.733

Taula B.17: Execucions del programa Cholesky (`MATRIX_SIZE`: 8192, `BLOCK_SIZE`: 256)

cores	workers per core	Original cycles	Original speedup	DAST cycles	DAST speedup
	seq	5.717×10^{10}	1.000	–	–
1	1	5.747×10^{10}	0.994	6.608×10^{10}	0.865
1	2	3.701×10^{10}	1.544	4.309×10^{10}	1.326
1	3	3.407×10^{10}	1.677	3.784×10^{10}	1.510
1	4	3.457×10^{10}	1.653	3.645×10^{10}	1.568
2	1	2.983×10^{10}	1.916	3.119×10^{10}	1.832
2	2	1.878×10^{10}	3.440	2.033×10^{10}	2.812
2	3	1.755×10^{10}	3.256	1.846×10^{10}	3.970
2	4	1.803×10^{10}	3.171	1.926×10^{10}	2.967
4	1	1.487×10^{10}	3.845	1.532×10^{10}	3.731
4	2	9.990×10^{9}	5.722	1.043×10^{10}	5.479
4	3	9.665×10^{9}	5.914	1.006×10^{10}	5.679
4	4	1.048×10^{10}	5.452	1.289×10^{10}	4.435
8	1	8.024×10^{9}	7.124	8.042×10^{9}	7.108
8	2	6.046×10^{9}	9.455	6.177×10^{9}	9.255
8	3	6.680×10^{9}	8.557	6.608×10^{9}	8.650
8	4	7.593×10^{9}	7.528	1.036×10^{10}	5.517
16	1	4.872×10^{9}	11.733	4.888×10^{9}	11.695
16	2	4.705×10^{9}	12.150	4.629×10^{9}	12.348
16	3	5.900×10^{9}	9.688	5.980×10^{9}	9.559
16	4	7.014×10^{9}	8.150	8.012×10^{9}	7.134
32	1	3.843×10^{9}	14.873	3.891×10^{9}	14.693
32	2	4.635×10^{9}	12.334	4.673×10^{9}	12.232
32	3	5.887×10^{9}	9.710	6.205×10^{9}	9.213
32	4	6.994×10^{9}	8.174	6.396×10^{9}	8.937
60	1	3.837×10^{9}	14.897	4.128×10^{9}	13.850
60	2	4.590×10^{9}	12.455	5.293×10^{9}	10.799
60	3	5.937×10^{9}	9.629	6.411×10^{9}	8.916
60	4	7.116×10^{9}	8.330	6.339×10^{9}	9.180

Taula B.18: Execucions del programa Cholesky (`MATRIX_SIZE`: 8192, `BLOCK_SIZE`: 512)

cores	workers per core	Original		DAST	
		ms	speedup	ms	speedup
	seq	13897.300	1.000	–	–
1	1	17058.100	0.815	17671.700	0.786
1	2	10741.000	1.294	10660.900	1.304
1	3	8505.000	1.634	8525.170	1.630
1	4	7371.430	1.885	7749.700	1.793
2	1	8818.400	1.576	8536.670	1.628
2	2	5404.670	2.571	5186.630	2.679
2	3	4318.630	3.218	4172.870	3.330
2	4	3817.200	3.641	3683.930	3.772
4	1	4648.170	2.990	4269.670	3.255
4	2	3037.880	4.575	2648.130	5.248
4	3	2421.920	5.738	2258.600	6.153
4	4	2002.710	6.939	3004.550	4.625
8	1	2563.280	5.422	2127.930	6.531
8	2	1681.470	8.265	1929.720	7.202
8	3	1782.890	7.795	2262.580	6.142
8	4	1888.970	7.357	2944.080	4.720
16	1	1628.910	8.532	1575.480	8.821
16	2	1717.420	8.092	1994.830	6.967
16	3	1792.730	7.752	2453.040	5.665
16	4	1963.910	7.076	3267.660	4.253
32	1	1691.940	8.214	1636.180	8.494
32	2	1731.670	8.025	2216.690	6.269
32	3	1814.510	7.659	2755.130	5.044
32	4	1998.030	6.955	3767.630	3.689
60	1	1718.580	8.086	1780.710	7.804
60	2	1793.790	7.747	2628.320	5.288
60	3	1953.990	7.112	3593.870	3.867
60	4	2155.940	6.446	4399.870	3.159

Taula B.19: Execucions del programa Matrix Multiply (MATRIX_SIZE: 2048, BLOCK_SIZE: 64)

cores	workers per core	Original		DAST	
		ms	speedup	ms	speedup
	seq	116231.000	1.000	–	–
1	1	148907.000	0.781	155390.000	0.748
1	2	105504.000	1.102	99750.000	1.165
1	3	82371.700	1.411	81802.700	1.421
1	4	72189.300	1.610	72321.700	1.607
2	1	78834.300	1.474	73991.700	1.571
2	2	54466.700	2.134	51055.700	2.277
2	3	44691.000	2.601	39483.700	2.944
2	4	38980.700	2.982	34282.000	3.390
4	1	43866.300	2.650	36223.000	3.209
4	2	31033.200	3.745	27053.700	4.296
4	3	24775.100	4.691	21927.100	5.301
4	4	19940.300	5.829	24662.491	4.713
8	1	24074.000	4.828	19448.200	5.976
8	2	15242.200	7.626	18375.000	6.325
8	3	13446.300	8.644	20812.100	5.585
8	4	15383.100	7.556	24957.240	4.657
16	1	12833.900	9.057	15355.200	7.569
16	2	13400.900	8.673	18534.500	6.271
16	3	13746.800	8.455	21548.700	5.394
16	4	15587.300	7.457	8576.400	13.552
32	1	12971.600	8.960	15784.000	7.364
32	2	13760.900	8.446	20508.600	5.667
32	3	14031.900	8.283	23941.400	4.855
32	4	15267.600	7.613	9282.900	12.521
60	1	13457.800	8.637	16332.600	7.117
60	2	13926.200	8.346	22743.400	5.111
60	3	14715.700	7.898	28791.200	4.037
60	4	15736.700	7.386	22344.400	5.202

Taula B.20: Execucions del programa Matrix Multiply (MATRIX_SIZE: 4096, BLOCK_SIZE: 64)

cores	workers per core	Original ms	Original speedup	DAST ms	DAST speedup
	seq	6967.830	1.000	–	–
1	1	7427.670	0.938	8069.700	0.863
1	2	5049.730	1.380	5502.870	1.266
1	3	3942.730	1.767	4284.500	1.626
1	4	3440.000	2.026	3766.230	1.850
2	1	3758.330	1.854	3848.000	1.811
2	2	2538.560	2.745	2605.590	2.674
2	3	2007.520	3.471	2051.070	3.397
2	4	1730.290	4.027	1815.220	3.839
4	1	1909.690	3.649	1908.040	3.652
4	2	1298.300	5.367	1288.260	5.409
4	3	1022.930	6.812	1018.860	6.839
4	4	893.717	7.796	992.013	7.024
8	1	995.797	6.997	975.510	7.143
8	2	703.260	9.908	674.713	10.327
8	3	563.283	12.370	551.533	12.634
8	4	521.420	13.363	615.030	11.329
16	1	551.590	12.632	534.183	13.044
16	2	411.477	16.934	385.740	18.064
16	3	363.667	19.160	406.010	17.162
16	4	417.253	16.699	634.640	10.979
32	1	355.757	19.586	323.823	21.517
32	2	336.670	20.696	365.477	19.065
32	3	386.983	18.006	463.357	15.038
32	4	451.947	15.417	716.323	9.727
60	1	345.203	20.185	330.166	21.104
60	2	396.453	17.575	442.907	15.732
60	3	478.877	14.550	593.157	11.747
60	4	595.230	11.706	913.593	7.627

Taula B.21: Execucions del programa Matrix Multiply (`MATRIX_SIZE`: 2048, `BLOCK_SIZE`: 128)

cores	workers per core	Original ms	Original speedup	DAST ms	DAST speedup
	seq	62661.000	1.000	–	–
1	1	66289.300	0.945	71369.700	0.878
1	2	51025.700	1.228	54222.300	1.156
1	3	36599.300	1.712	38931.300	1.610
1	4	30728.200	2.039	33904.000	1.848
2	1	33450.300	1.873	34269.000	1.829
2	2	25544.400	2.453	25381.800	2.469
2	3	18415.100	3.403	18634.300	3.363
2	4	15462.900	4.052	16131.700	3.884
4	1	16994.200	3.687	17046.300	3.676
4	2	12508.600	5.009	12583.900	4.979
4	3	9237.600	6.783	9399.370	6.667
4	4	7899.200	7.933	7973.670	7.858
8	1	8701.630	7.201	8540.170	7.337
8	2	6373.530	9.831	6227.200	10.063
8	3	4817.500	13.007	4670.000	13.418
8	4	4201.070	14.915	4101.100	15.279
16	1	4586.870	13.661	4350.700	14.402
16	2	3355.770	18.673	3176.650	19.726
16	3	2571.590	24.367	2829.200	22.148
16	4	2578.430	24.302	4068.430	15.402
32	1	2574.650	24.338	2288.730	27.378
32	2	2046.800	30.614	2455.740	25.516
32	3	2188.550	28.631	3087.680	20.294
32	4	2511.070	24.954	4577.400	13.689
60	1	1787.870	35.048	1909.650	32.813
60	2	2049.590	30.573	2767.910	22.638
60	3	2302.590	27.213	3841.570	16.311
60	4	2504.510	25.019	5699.830	10.993

Taula B.22: Execucions del programa Matrix Multiply (`MATRIX_SIZE`: 4096, `BLOCK_SIZE`: 128)

cores	workers per core	Original ms	Original speedup	DAST ms	DAST speedup
	seq	5527.770	1.000	–	–
1	1	5620.030	0.984	6291.830	0.879
1	2	3863.070	1.431	4364.270	1.267
1	3	3070.540	1.800	3462.170	1.597
1	4	2672.350	2.069	3068.360	1.802
2	1	2834.220	1.950	2978.790	1.856
2	2	1968.420	2.808	2050.100	2.696
2	3	1547.250	3.573	1669.030	3.312
2	4	1355.120	4.079	1561.910	3.539
4	1	1439.360	3.840	1499.510	3.686
4	2	1008.990	5.479	1117.060	4.949
4	3	847.357	6.524	865.247	6.389
4	4	719.793	7.680	851.713	6.490
8	1	754.393	7.327	813.750	6.793
8	2	550.957	10.033	607.190	9.104
8	3	471.717	11.718	481.973	11.469
8	4	434.713	12.716	540.677	10.224
16	1	423.457	13.054	462.140	11.961
16	2	330.246	16.738	366.843	15.069
16	3	327.380	16.885	345.477	16.000
16	4	339.707	16.272	401.613	13.764
32	1	299.994	18.426	318.864	17.336
32	2	271.188	20.384	267.020	20.702
32	3	304.520	18.152	313.202	17.649
32	4	340.907	16.215	361.367	15.297
60	1	281.923	19.607	280.138	19.732
60	2	279.016	19.812	273.902	20.182
60	3	314.109	17.598	332.462	16.627
60	4	364.420	15.169	405.203	13.642

Taula B.23: Execucions del programa Matrix Multiply (`MATRIX_SIZE`: 2048, `BLOCK_SIZE`: 256)

cores	workers per core	Original ms	Original speedup	DAST ms	DAST speedup
	seq	50853.000	1.000	–	–
1	1	51555.300	0.986	56702.300	0.897
1	2	34993.700	1.453	39635.700	1.283
1	3	26993.500	1.884	30026.500	1.694
1	4	23911.900	2.127	26526.700	1.917
2	1	25906.400	1.963	27069.800	1.879
2	2	17642.600	2.882	18759.500	2.711
2	3	13532.300	3.758	14362.300	3.541
2	4	12047.100	4.221	12647.900	4.021
4	1	13134.300	3.872	13529.300	3.759
4	2	9087.400	5.596	9235.430	5.506
4	3	6934.700	7.333	7044.270	7.219
4	4	6140.530	8.282	6392.630	7.955
8	1	6709.870	7.579	6789.730	7.490
8	2	4564.400	11.141	4752.000	10.701
8	3	3568.570	14.250	3666.270	13.870
8	4	3259.940	15.599	3349.930	15.180
16	1	3512.630	14.477	3509.600	14.490
16	2	2386.830	21.306	2409.370	21.106
16	3	1975.400	25.743	1980.900	25.672
16	4	1846.420	27.541	1915.570	26.547
32	1	1888.330	26.930	1883.360	27.001
32	2	1387.970	36.638	1343.560	37.849
32	3	1190.640	42.711	1234.050	41.208
32	4	1196.310	42.508	1442.660	35.249
60	1	1155.240	44.019	1113.920	45.652
60	2	964.767	52.710	953.587	53.328
60	3	982.730	51.747	1042.500	48.780
60	4	1114.440	45.631	1385.010	36.717

Taula B.24: Execucions del programa Matrix Multiply (`MATRIX_SIZE`: 4096, `BLOCK_SIZE`: 256)

cores	workers per core	Original seconds	Original speedup	DAST seconds	DAST speedup
	seq	0.332	1.000	–	–
1	1	0.377	0.880	0.332	1.000
1	2	0.271	1.227	0.224	1.485
1	3	0.231	1.440	0.195	1.703
1	4	0.266	1.248	0.183	1.818
2	1	0.177	1.877	0.169	1.966
2	2	0.127	2.625	0.116	2.863
2	3	0.114	2.909	0.107	3.106
2	4	0.175	1.896	0.102	3.246
4	1	0.095	3.486	0.091	3.665
4	2	0.073	4.563	0.067	4.964
4	3	0.072	4.590	0.065	5.103
4	4	0.158	2.100	0.067	4.980
8	1	0.053	6.231	0.054	6.149
8	2	0.047	7.020	0.046	7.215
8	3	0.055	6.059	0.053	6.221
8	4	0.141	2.349	0.064	5.165
16	1	0.040	8.259	0.041	8.179
16	2	0.046	7.263	0.048	6.919
16	3	0.056	5.906	0.057	5.838
16	4	0.140	2.367	0.073	4.568
32	1	0.046	7.165	0.047	7.045
32	2	0.056	5.958	0.053	6.319
32	3	0.066	5.007	0.067	4.967
32	4	0.129	2.576	0.081	4.111
60	1	0.060	5.542	0.061	5.435
60	2	0.072	4.606	0.074	4.514
60	3	0.085	3.907	0.082	4.034
60	4	0.235	1.416	0.102	3.272

Taula B.25: Execucions del programa Sparse LU (`NB`: 16, `BSIZE`: 50)

cores	workers per core	Original seconds	Original speedup	DAST seconds	DAST speedup
	seq	2.561	1.000	–	–
1	1	2.845	0.900	2.561	1.000
1	2	2.072	1.236	1.718	1.490
1	3	1.744	1.468	1.497	1.710
1	4	1.584	1.617	1.363	1.878
2	1	1.345	1.904	1.296	1.975
2	2	0.931	2.751	0.868	2.951
2	3	0.806	3.178	0.756	3.386
2	4	0.804	3.185	0.694	3.688
4	1	0.658	3.889	0.655	3.909
4	2	0.453	5.655	0.442	5.789
4	3	0.401	6.388	0.393	6.513
4	4	0.507	5.049	0.372	6.880
8	1	0.336	7.619	0.341	7.500
8	2	0.236	10.857	0.236	10.838
8	3	0.221	11.568	0.223	11.495
8	4	0.390	6.559	0.220	11.617
16	1	0.180	14.188	0.191	13.421
16	2	0.148	17.277	0.149	17.204
16	3	0.167	15.294	0.160	16.020
16	4	0.315	8.122	0.184	13.895
32	1	0.124	20.645	0.131	19.488
32	2	0.173	14.806	0.130	19.725
32	3	0.212	12.051	0.210	12.166
32	4	0.430	5.960	0.252	10.177
60	1	0.138	18.595	0.139	18.373
60	2	0.351	7.290	0.355	7.206
60	3	0.425	6.032	0.358	7.146
60	4	1.184	2.163	0.489	5.233

Taula B.26: Execucions del programa Sparse LU (`NB`: 32, `BSIZE`: 50)

cores	workers per core	Original seconds	Original speedup	DAST seconds	DAST speedup
	seq	20.323	1.000	–	–
1	1	22.781	0.892	20.498	0.991
1	2	16.519	1.230	13.819	1.471
1	3	13.877	1.465	11.903	1.707
1	4	11.899	1.708	10.757	1.889
2	1	10.671	1.904	10.273	1.978
2	2	7.344	2.767	6.833	2.974
2	3	6.280	3.236	5.915	3.436
2	4	5.645	3.600	5.389	3.771
4	1	5.177	3.926	5.173	3.929
4	2	3.514	5.783	3.417	5.948
4	3	3.032	6.704	2.986	6.805
4	4	2.842	7.151	2.739	7.419
8	1	2.563	7.929	2.667	7.621
8	2	1.724	11.787	1.784	11.390
8	3	1.582	12.850	1.578	12.876
8	4	1.910	10.639	1.448	14.039
16	1	1.333	15.247	1.386	14.659
16	2	1.036	19.625	0.954	21.313
16	3	1.058	19.213	0.873	23.279
16	4	1.698	11.968	0.880	23.091
32	1	0.809	25.125	0.767	26.480
32	2	1.135	17.901	0.676	30.054
32	3	1.234	16.466	1.647	12.343
32	4	4.080	4.981	2.119	9.593
60	1	0.657	30.934	0.617	32.913
60	2	2.204	9.221	4.152	4.895
60	3	2.445	8.313	3.535	5.749
60	4	12.116	1.677	4.485	4.532

Taula B.27: Execucions del programa Sparse LU (`NB`: 64, `BSIZE`: 50)

cores	workers per core	Original seconds	Original speedup	DAST seconds	DAST speedup
	seq	1.072	1.000	–	–
1	1	1.187	0.903	1.071	1.001
1	2	0.811	1.323	0.675	1.587
1	3	0.670	1.600	0.568	1.887
1	4	0.661	1.623	0.527	2.035
2	1	0.576	1.860	0.548	1.957
2	2	0.379	2.827	0.344	3.121
2	3	0.342	3.134	0.304	3.530
2	4	0.386	2.775	0.292	3.674
4	1	0.292	3.667	0.284	3.771
4	2	0.202	5.311	0.194	5.525
4	3	0.195	5.505	0.183	5.873
4	4	0.289	3.705	0.195	5.485
8	1	0.165	6.510	0.159	6.751
8	2	0.131	8.168	0.124	8.645
8	3	0.150	7.133	0.142	7.541
8	4	0.263	4.082	0.170	6.292
16	1	0.111	9.666	0.109	9.799
16	2	0.112	9.580	0.111	9.622
16	3	0.148	7.227	0.148	7.257
16	4	0.275	3.904	0.179	5.984
32	1	0.100	10.671	0.103	10.454
32	2	0.115	9.286	0.123	8.683
32	3	0.143	7.519	0.148	7.242
32	4	0.294	3.644	0.183	5.858
60	1	0.110	9.774	0.113	9.504
60	2	0.130	8.253	0.132	8.143
60	3	0.160	6.688	0.151	7.114
60	4	0.320	3.353	0.184	5.828

Taula B.28: Execucions del programa Sparse LU (`NB`: 16, `BSIZE`: 75)

cores	workers per core	Original seconds	Original speedup	DAST seconds	DAST speedup
	seq	8.389	1.000	–	–
1	1	9.283	0.904	8.369	1.002
1	2	6.183	1.357	5.168	1.623
1	3	5.085	1.650	4.327	1.939
1	4	4.471	1.876	3.909	2.146
2	1	4.389	1.911	4.200	1.997
2	2	2.804	2.991	2.580	3.252
2	3	2.345	3.578	2.176	3.855
2	4	2.147	3.907	1.988	4.220
4	1	2.147	3.907	2.118	3.961
4	2	1.362	6.159	1.322	6.346
4	3	1.171	7.166	1.130	7.422
4	4	1.161	7.228	1.046	8.020
8	1	1.082	7.750	1.089	7.705
8	2	0.705	11.904	0.689	12.170
8	3	0.629	13.329	0.616	13.628
8	4	0.744	11.272	0.606	13.853
16	1	0.575	14.583	0.578	14.516
16	2	0.397	21.109	0.399	21.051
16	3	0.412	20.371	0.402	20.893
16	4	0.643	13.041	0.451	18.595
32	1	0.334	25.136	0.346	24.215
32	2	0.297	28.223	0.291	28.800
32	3	0.377	22.257	0.350	23.976
32	4	0.690	12.160	0.451	18.599
60	1	0.267	31.435	0.282	29.776
60	2	0.368	22.798	0.346	24.210
60	3	0.502	16.698	0.431	19.472
60	4	0.924	9.080	0.531	15.786

Taula B.29: Execucions del programa Sparse LU (NB: 32, BSIZE: 75)

cores	workers per core	Original seconds	Original speedup	DAST seconds	DAST speedup
	seq	66.790	1.000	–	–
1	1	73.493	0.909	66.775	1.000
1	2	48.633	1.373	41.063	1.627
1	3	39.907	1.674	34.215	1.952
1	4	34.479	1.937	30.862	2.164
2	1	34.808	1.919	33.421	1.998
2	2	22.013	3.034	20.427	3.270
2	3	18.229	3.664	17.033	3.921
2	4	16.235	4.114	15.380	4.343
4	1	16.985	3.932	16.726	3.993
4	2	10.510	6.355	10.142	6.586
4	3	8.813	7.578	8.544	7.817
4	4	8.015	8.333	7.741	8.628
8	1	8.431	7.922	8.387	7.963
8	2	5.157	12.952	5.111	13.069
8	3	4.386	15.229	4.333	15.413
8	4	4.188	15.948	3.987	16.751
16	1	4.254	15.702	4.264	15.665
16	2	2.647	25.236	2.658	25.125
16	3	2.298	29.069	2.310	28.915
16	4	2.609	25.605	2.253	29.651
32	1	2.168	30.803	2.216	30.145
32	2	1.530	43.650	1.464	45.632
32	3	1.502	44.457	1.446	46.193
32	4	2.831	23.590	1.564	42.705
60	1	1.258	53.105	1.304	51.224
60	2	1.563	42.726	1.286	51.936
60	3	2.428	27.509	1.358	49.193
60	4	9.870	6.767	1.921	34.764

Taula B.30: Execucions del programa Sparse LU (NB: 64, BSIZE: 75)