Porting the OmpSs programming model to the Argobots runtime system

Autor
Kevin Sala Penadés

Director
Eduard Ayguadé Parra (DAC)

Co-director
Vicenç Beltran Querol (BSC-CNS)

Barcelona Supercomputing Center - Centro Nacional de Supercomputación (BSC-CNS)
Departament d'Arquitectura de Computadors (DAC)

Grau en Enginyeria Informàtica

Especialitat
Enginyeria de Computadors

21 de Juny de 2016

Agraïments

M'agradaria agrair al meu director, Eduard Ayguadé, per oferir-me l'oportunitat de treballar al BSC i deixar-me fer aquest projecte tan interessant com a treball de final de grau.

També m'agradaria agrair al meu subdirector, Vicenç Beltran, per escoltar totes les meves idees i propostes, creure en mi i guiar-me en els moments més complicats.

Aquest treball hauria sigut molt més difícil per mi si no fos per l'ajuda i el suport del Josep M. Pérez i del Jorge Bellón. Gràcies per tota l'ajuda en els moments més difícils i per haver-me ensenyat quina és la bona manera de fer les coses.

També vull agrair a la resta de companys del BSC el bon tracte rebut i els comentaris d'ànim en tot moment.

Per últim però no menys important, vull agrair a tota la meva família i amics totes les mostres de suport i la confiança que han dipositat en mi.

Resum

Després d'un breu resum sobre el model de programació d'OmpSs i la llibreria d'Argobots, aquest document mostra el disseny i implementació d'un runtime que suporta OmpSs i està implementat usant la llibreria d'Argobots. Primer, es desenvolupa l'estructura del runtime que serà la base del projecte. Després s'implementen el sistema de dependències puntuals i de regió continua. També es dóna suport a l'interoperabilitat amb MPI. També s'implementa la clàusula commutative d'OmpSs. Un cop finalitzada la implementació, s'avaluen els resultats amb benchmarks i aplicacions com Cholesky, Gauss-Seidel i Nbody.

Índex

1 Context

En aquest apartat hi ha la introducció i els agents implicats en el projecte, sigui de forma directa o indirecta.

1.1 Introducció

Aquest projecte és un Treball de Final de Grau de l'especialitat d'Enginyeria de Computadors de la Facultat d'Informàtica de Barcelona (Universitat Politècnica de Catalunya). Es tracta d'un projecte amb col·laboració amb el Barcelona Supercomputing Center - Centro Nacional de Supercomputación (BSC - CNS), on es busca desenvolupar un runtime de baix impacte amb el framework d'Argobots[1] com a *back-end* i que suporti el model de programació paral·lela d'OmpSs[2].

Per una banda, OmpSs és un model de programació paral·lela desenvolupat al Barcelona Supercomputing Center - Centro Nacional de Supercomputación (BSC - CNS). És una extensió del model de programació paral·lela OpenMP amb les funcionalitats més importants del model StarSs. Les característiques més destacades són el suport al paral·lelisme asíncron i la heterogeneïtat amb dispositius com acceleradors i GPUs. L'entorn del model d'OmpSs consta del compilador Mercurium[3], el runtime Nanos++[4] i una sèrie d'eines de *profiling* i *tracing*. Tots aquests components es detallaran en els següents apartats.

Per altra banda, Argobots és un framework que consta d'una infraestructura de baix nivell que suporta un model *lightweight* de threads i tasques. Aquest framework ha sigut desenvolupat en el Argonne National Laboratory i també es detalla en els següents apartats.

1.2 Actors implicats

Hi ha diferents actors implicats en aquest projecte i es detallen a continuació:

- **Desenvolupador:** El desenvolupador sóc jo mateix. Donat que el projecte és complex i complicat, i es desenvoluparà sobre un software existent amb una complexitat elevada, tindré el suport d'un Enginyer de suport a la investigació.
- **Enginyer de suport a la investigació:** Serà l'encarregat d'ajudar al desenvolupador en els moments difícils i amb més dificultat. També protagonitzarà una petita formació sobre les eines i el software existent.
- **Directors del projecte:** La feina tant del director com del co-director és supervisar l'avanç del projecte i revisar que els objectius es compleixen de forma satisfactòria i en els terminis especificats.
- **Barcelona Supercomputing Center:** Donat que el model d'OmpSs ha sigut desenvolupat pel BSC, aquest centre es veurà beneficiat per aquest projecte, ja que afegirà un nou runtime que implementarà les funcionalitats principals d'aquest model de programació paral·lela.
- **Argonne National Laboratory:** El BSC forma part del conveni JLESC[5] (Joint Laboratory on Extreme Scale Computing) juntament amb l'Argonne National Laboratory, la Universitat d'Illinois, l'INRIA (Institut National de Recherche en Informatique et en Automatique) i el Jülich Supercomputing Centre. Aquest conveni està enfocat als reptes de programació que es troben en els computadors d'altes prestacions.

Això, i donat que aquest projecte es durà a terme usant com a back-end el runtime d'Argobots, desenvolupat a l'Argonne National Laboratory, hi haurà una certa comunicació amb els seus desenvolupadors, ja sigui per a intercanviar *feedback* o per a informar de possibles problemes d'Argobots.

- **Usuaris:** Els usuaris podran usar aquest nou runtime per executar aplicacions paral·leles implementades amb OmpSs. Els usuaris podrien ser investigadors que usin OmpSs com a model de programació paral·lela, alumnes que estan en fase d'aprenentatge d'aquest model, etc.

2 Estat de l'art

En aquest apartat es tracta l'estat de l'art i la contextualització, on s'explica de forma detallada el model de programació d'OmpSs i el framework d'Argobots. Seguidament hi ha un recull dels projectes relacionats i per últim, la conclusió de l'estat de l'art.

2.1 OmpSs

El model de programació paral·lela OmpSs[2] és el producte d'estendre OpenMP[6] integrant-lo amb diverses característiques de la família de models de programació StarSs[7]. Desenvolupat en el Barcelona Supercomputing Center - Centro Nacional de Supercomputación (BSC - CNS), busca estendre el model d'OpenMP amb noves característiques per a suportar paral·lelisme asíncron basat en dependències de dades i l'heterogeneïtat de dispositius (com per exemple, targetes gràfiques i acceleradors). OmpSs està disponible tant per Fortran com per C/C++.

```
void bar(int * n) {
  *n = 10;
}

#pragma omp task firstprivate(n)
void foo(int n) {
  if (n > 0) {
    foo(n-1);
    #pragma omp taskwait
  }
}

int main () {
  int n;

  #pragma omp task out(n)
  bar(&n);

  #pragma omp task in(n)
  {
    foo(n);

    #pragma omp for
    for (int i = 0; i < n; ++i) printf("Iteration %d\n", i);

    #pragma omp taskwait
  }

  #pragma omp taskwait
}
```

Figura 1: Codi d'exemple de l'ús de les directives d'OpenMP i OmpSs.

2.1.1 Concepte

L'objectiu principal d'OmpSs és oferir un entorn productiu en el qual es puguin desenvolupar aplicacions per a sistemes de computació d'altes prestacions. Una propietat que necessita OmpSs per tal d'oferir-ho és el rendiment. Ha de tenir la capacitat d'aconseguir un rendiment comparable al d'altres models de programació paral·lela que usin les mateixes arquitectures. L'altra propietat és la senzillesa, i en aquest camp, OmpSs ha sigut dissenyat seguint uns principis elogiats per la seva efectivitat.

D'una banda, OmpSs adopta d'OpenMP la forma de paral·lelitzar les aplicacions introduint directives en el codi font. L'usuari les introdueix en el codi seqüencial i permet una parallelització incremental, és a dir, es comença pel codi seqüencial i es van afegint directives per a paral·lelitzar les diferents parts de l'aplicació.

Per altra banda, OmpSs adopta de StarSs el model d'execució. OpenMP usa un model d'execució *fork-join*, mentre que StarSs usa un model d'execució *thread-pool*. El model *fork-join* consisteix en executar el programa amb un fil d'execució (o thread) fins a arribar a una regió paral·lela (marcada per l'usuari), a on es crea la resta de threads, executen aquesta regió del codi en paral·lel i llavors s'eliminen quan ja han arribat al final de la regió. En canvi, en el model *thread-pool*, el programa s'inicia amb tots els threads creats i preparats per a rebre feina, i no s'eliminen fins al final de l'execució de l'aplicació. Per tant, com s'ha explicat anteriorment, OmpSs té un model d'execució de *thread-pool*. [8]

De StarSs també s'adopten les característiques de suportar arquitectures heterogènies, paral·lelisme asíncron i per últim, però no menys important, la senzillesa del codi que ha de generar l'usuari.

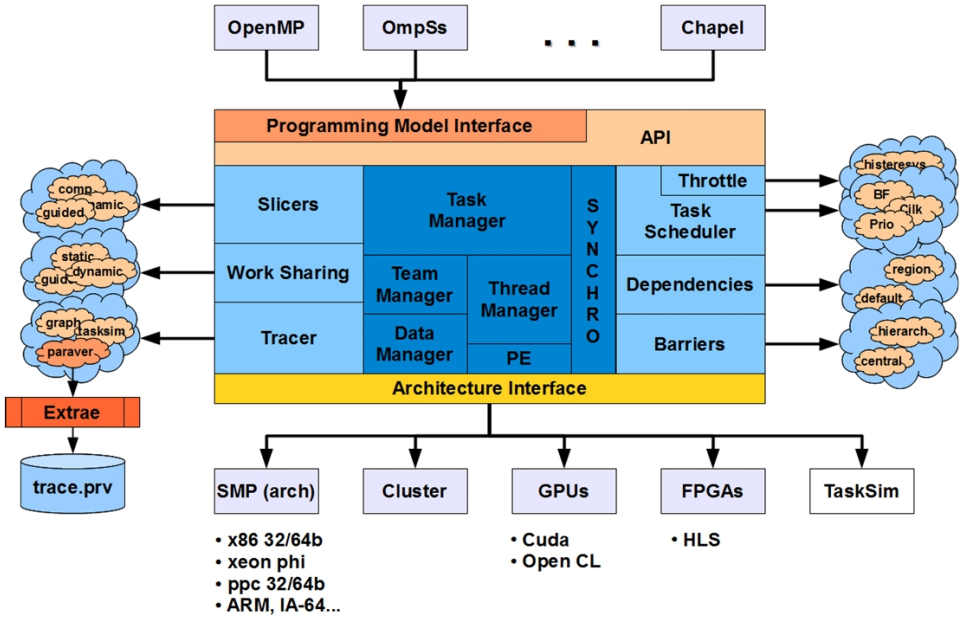

Figura 2: Dibuix on es mostra l'entorn del model OmpSs.

2.1.2 Entorn

L'entorn d'OmpSs consta normalment del compilador Mercurium[3] i del runtime Nanos++[4], ambdós desenvolupats en el Barcelona Supercomputing Center - Centro Nacional de Supercomputación (BSC - CNS). Com es pot veure a la Figura 2, l'ecosistema del model de programació OmpSs és molt variat. A part d'OmpSs en si, també és compatible amb els models d'OpenMP i Chapel[9]. També té suport a una gran varietat d'arquitectures totalment diferents entre elles, com poden ser GPUs, FPGAs, clusters, etc. També compta amb el suport de les eines següents:

- **Paraver:** Aquesta eina és un analitzador de rendiment basat en traces d'execucions. Compta amb una gran quantitat d'opcions de visualització i format de les dades, per a què l'usuari pugui veure en exactitud el que està passant a cada moment de l'execució.[10]
- **Extrae:** Aquesta és l'eina que extreu dades de les execucions de les aplicacions i és l'encarregat de crear les traces.[11]
- **Dimemas:** Aquesta eina és un analitzador de rendiment per a les aplicacions de *message passing*. Permet desenvolupar aplicacions paral·leles en un entorn de treball i proporciona prediccions amb molta precisió del rendiment en una màquina determinada.[12]
- **Tareador:** Aquesta és una eina per a explorar la descomposició de tasques i les dependències entre dades d'un programa, amb forma de graf. L'usuari pot usar Tareador per a saber a on es troba el parel·lelisme potencial de la seva aplicació.[13]

2.1.3 Definint el paral·lelisme

Com s'ha explicat anteriorment, OmpSs permet inserir anotacions i directives en el codi per a poder paral·lelitzar-lo de forma correcta i adient. Un exemple de l'ús de directives és el de la Figura 1.

Una de les característiques més destacades d'OmpSs és que no necessita definir les regions paral·leles de l'aplicació, a diferència d'OpenMP, ja que com s'ha detallat anteriorment, OmpSs utilitza un model *thread-pool* on els threads estan preparats per a rebre feina.

El model d'OmpSs permet expressar el paral·lelisme mitjançant tasques. Les tasques tenen associat una part petita del codi i poden ser executades concurrentment. En temps d'execució, quan el flux arriba a una part declarada com a tasca, es criden les funcions del runtime pertinents per a crear-la i es delega la resta de la feina al runtime. A part d'aquesta forma, també permet expressar paral·lelisme gràcies a la clàusula *for*, que ha d'estar associada a un bucle *for*. Aquesta clàusula encapsula grups d'iteracions del bucle i executa cada grup en una tasca diferent.

Una altra característica important és que OmpSs permet definir múltiples nivells de paral·lelisme, ja que permet crear tasques aniuades, és a dir, unes dins de les altres. Aquesta característica és molt útil per a tenir un rendiment molt alt en els programes i també permet implementar les aplicacions que es doten d'algorismes recursius.

Per últim, OmpSs permet la sincronització de tasques de forma implícita i explícita. Per a la forma implícita, OmpSs proporciona un sistema complex de dependències de dades. Aquestes dependències s'expressen amb les clàusules *in*, *out* i *inout*, juntament amb la clàusula per a definir la tasca. Aquestes clàusules permeten definir quines dades té com a entrada i sortida una tasca. En canvi, per a la forma explícita existeix una directiva anomenada *taskwait* que serveix per a sincronitzar tasques.

2.2 Argobots

Argobots[1] és un framework que consta d'una infraestructura de baix nivell que suporta un model *lightweight* de threads i tasques per a ser usat en un entorn de concurrència massiva. S'aprofita directament dels recursos que ofereix el *hardware* i el sistema operatiu, com poden ser els mecanismes *lightweight* de notificació, motors de moviment de dades, mapeig de memòria i estratègies de posicionament de dades. En resum, Argobots consisteix en un model d'execució i en un model de memòria.

2.2.1 Concepte

Argobots suporta dos nivells de paral·lelisme: *execution streams* i *work units* (a partir d'ara anomenats streams o ES i unitats de treball, respectivament). Un stream és un flux seqüencial d'instruccions que conté una o més unitats de treball. Està assignat a un recurs *hardware* i garanteix el progrés de l'execució. Les unitats de treball són unitats d'execució de baix impacte (o *lightweight*), que poden ser *user-level threads* (ULTs) o tasklets. Cada una està associada a una funció que determina l'usuari i que executa fins a la seva resolució. Una unitat de treball pot estar associada a un stream i, alhora, aquests executen totes les seves unitats de treball associades d'acord amb la seva política de planificació. Per tant, dues o més unitats poden ser executades en paral·lel si i només si, aquestes estan associades a diferents streams, ja que no hi ha concurrència a dins d'un sol stream. A la figura 3 es pot observar l'esquema general. [1, Secció Execution Stream (ES)]

Hi ha dues abstraccions a Argobots. Per una banda, la primera abstracció suporta una execució concurrent amb un baix impacte, com els ULTs o els tasklets, que són capaços d'adaptar-se dinàmicament i eficientment al conjunt de requisits de les aplicacions i dels recursos del *hardware*. Per altra banda, la segona abstracció suporta una activació de les unitats de treball de baixa latència i de baix impacte, controlada principalment per missatges.

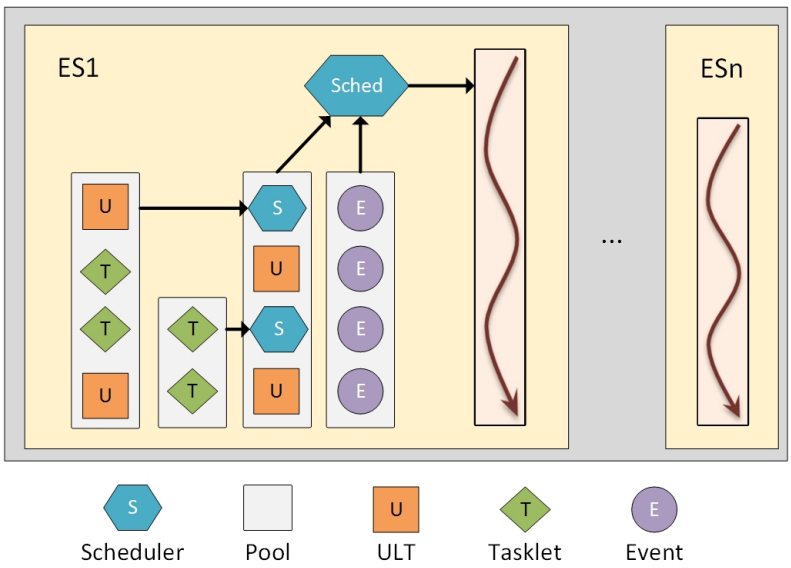

Figura 3: Dibuix on es mostra l'esquema general del runtime Argobots.

2.2.2 Unitats de treball

Els ULTs i els tasklets ofereixen a les aplicacions una abstracció per a paral·lelisme de gra fi. Cada ULT o tasklet és una unitat d'execució independent planificada pel runtime. Aquestes dues unitats de treball però, difereixen en alguns aspectes, que es detallen a continuació:

- **User-level thread (ULT)**: Aquest tipus d'unitat expressa el paral·lisme en termes de contexts persistents, on el flux de control s'atura i es reprèn depenent del flux de dades. Un exemple clar de l'ús d'user-level threads és una aplicació on es programen *futures* (estructura on un thread es bloqueja per esperar una dada) per esperar la recepció de dades remotes. Els ULTs no estan destinats a ser expulsats de l'execució de forma preemptiva, sinó que cedeixen el control al *scheduler* de forma explícita o cridant funcions bloquejants. [1, Secció User-level Thread (ULT)]
- **Tasklet**: Aquesta és una unitat de treball indivisible, amb dependència només de les seves dades d'entrada i típicament amb una producció de dades de sortida un cop ha finalitzat. Expressa la concurrència basada en les condicions de *Bernstein* (la intersecció entre l'entrada d'un i la sortida de l'altre és buida, igual que la intersecció de les dues sortides). Una possible aplicació podria ser el producte de matrius en blocs, que s'executa quan els dos blocs estan disponibles. Els tasklets, a diferència dels user-level threads, no cedeixen de forma explícita el flux de control fins que no acaben d'executar-se. [1, Secció Tasklet]

Una altra característica d'Argobots és que permet la migració d'unitats de treball, tant d'ULTs com de tasklets, entre streams o entre pools de diferents schedulers. La diferència és que un ULT pot migrar mentre s'està executant, en canvi, els tasklets només poden migrar abans de començar la seva execució.

2.2.3 Pools

Una pool és un una estructura de dades que representa un contenidor, que inclou totes les unitats de planificació que han de ser planificades. Aquest contenidor es pot definir com una cua *FIFO* (*First In First Out*), una cua *LIFO* (*Last In First Out*), una cua de prioritats o un arbre, depenent de la política de planificació aplicada. Una pool està assignada a un scheduler, i aquests poden tenir assignades diferents pools.

2.2.4 Schedulers

Un scheduler controla l'ordre d'execució de les unitats de treball planificades en el seu stream. Cada scheduler està associat a un stream concret, tot i que un stream pot tenir associat diversos schedulers i pot canviar entre ells en temps d'execució.

Tot i que Argobots proporciona schedulers bàsics, l'usuari pot definir-ne de nous amb les seves pròpies polítiques de planificació. Aquesta característica li dóna molt potencial, donat que l'usuari pot usar polítiques de planificació totalment personalitzades depenent dels algorismes o aplicacions que executi. Una altra propietat és que els schedulers poden planificar altres schedulers com a unitats de planificació, per tant, es poden apilar uns amb els altres.

Les unitats de treball són planificades usant el canvi de context. Quan una unitat de treball acaba la seva execució o un ULT cedeix el flux de control, hi ha un canvi de context i s'executa l'scheduler. Aquest llavors escull, gràcies a la política de planificació, una altra unitat de treball i l'executa. [1, Secció Scheduler]

2.2.5 Altres funcionalitats

Argobots també implementa altres funcionalitats com *events*, barreres, variables de condició, *eventuals*, *futures*, *mutexes* i *timers*.

2.3 Projectes relacionats

Seguidament es detallen els projectes relacionats amb el desenvolupament de runtimes que suporten el model de programació d'OmpSs.

2.3.1 Runtime Nanos5

Nanos5 és un runtime dissenyat per a proporcionar suport als entorns paral·lels. Ha sigut desenvolupat al Barcelona Supercomputing Center - Centro Nacional de Supercomputación (BSC - CNS). Principalment és usat per a suportar el model d'OmpSs, però també té mòduls per a suportar OpenMP i Chapel.

Proporciona serveis per a suportar paral·lelisme a nivell de tasca usant sincronitzacions basades en dependències de dades. Normalment, les tasques són implementades com a user-level threads. A més, proporciona suport per a mantenir la coherència a través de diferents espais d'adreces, com per exemple GPUs i nodes de clusters.

L'objectiu principal de Nanos5 és ser usat en la recerca en entorns de programació paral·lela. Una de les propietats principals és la possibilitat d'estendre el runtime amb *plugins*, com per exemple, planificadors de tasques, suport a altres dispositius, instrumentació, etc. [4]

2.3.2 Runtime Nanos6

El projecte de Nanos6 consta d'un complex runtime que proporciona ple suport a OmpSs i representa el successor de l'actual Nanos5. Està en fase de desenvolupament en el Barcelona Supercomputing Center - Centro Nacional de Supercomputación (BSC - CNS).

2.4 Conclusió de l'estat de l'art

Com s'ha vist a les seccions 2.1 i 2.2, es farà ús del model de programació paral·lela d'OmpSs i del framework d'Argobots. En el cas d'OmpSs, no hi ha cap necessitat de modificar ni estendre cap característica d'aquest model, per tant, s'usarà l'última versió existent dels seus components. Per altra banda, el framework d'Argobots podria ser modificat, principalment afegint alguns canvis per a facilitar el desenvolupament del projecte. Aquests possibles canvis seran petits i poc importants, per tant, no afectaran l'estructura del framework.

Com s'ha explicat a la secció 2.3, el model de programació paral·lela d'OmpSs és suportat per diversos runtimes, desenvolupats pel Barcelona Supercomputing Center - Centro Nacional de Supercomputación (BSC - CNS). Segons fonts d'aquest centre, mai s'ha desenvolupat un runtime que tingui com a base el framework d'Argobots, per tant, donat que l'objectiu és avaluar si amb l'ajuda d'Argobots és possible desenvolupar un runtime de baix impacte, aquest s'ha de desenvolupar de nou. Cal afegir que algunes característiques del runtime seran adaptades dels que ja existeixen, ja que, independentment de l'ús d'Argobots, es dissenyaran de forma similar.

3 Abast i objectius del projecte

En aquest apartat es defineix el problema, els objectius principals i l'abast del projecte. També s'enumeren els riscos i obstacles que poden sorgir i les seves possibles solucions.

3.1 Formulació del problema

En l'actualitat, l'ús de models de programació paral·lela és molt estès i està en clar augment. Principalment, es busca trobar el rendiment i paral·lelisme màxim de les aplicacions desitjades.

En l'execució d'una aplicació paral·lelitzada, hi intervenen dos factors (entre d'altres). El primer és el nivell d'optimització del codi paral·lelitzat, i el segon, però no menys important, és l'impacte del runtime que implementa les funcionalitats del model que s'està utilitzant. Això significa que, encara que una aplicació sigui implementada de forma òptima, si el runtime té un impacte considerable, podria fer que els temps d'execució de l'aplicació augmentés de forma notable.

3.1.1 Objectiu principal

L'objectiu principal d'aquest projecte és desenvolupar un runtime eficient i de baix impacte gràcies al framework Argobots com a *back-end* i que suporti el model de programació paral·lela d'OmpSs.

Però per aconseguir aquest objectiu hi ha un seguit de requisits molt importants. Aquests s'anomenen a continuació.

- El runtime ha de tenir la capacitat de suportar dependències puntuals i dependències de regió contigua, principalment per poder executar els benchmarks.
- Alguns dels benchmarks usats per avaluar el rendiment usen el model d'MPI[14].
- Les eines que s'utilitzin han de ser presents al MareNostrum III. En cas que no ho siguin, s'han d'instal·lar a la carpeta personal del desenvolupador.
- El runtime ha de poder executar-se en diferents entorns, però hauria de ser dissenyat per executar-se principalment a MareNostrum III.
- El desenvolupament del runtime no ha de provocar un augment de la dificultat d'ús del model d'OmpSs per als usuaris.
- Les funcionalitats implementades s'han d'ajustar als requisits del BSC.

3.1.2 Objectius específics

Per tal que es compleixi l'objectiu principal explicat en la secció 3.1.1 i tenint en compte els requisits, s'han de complir un seguit d'objectius específics, explicats a continuació:

- Dissenyar i implementar l'estructura del runtime de forma clara i senzilla, per a què sigui el màxim d'eficient.
- Dissenyar i implementar un sistema de dependències puntuals eficient i senzill, de forma que la inserció i alliberament de les dependències sigui ràpida i no provoqui la contenció dels threads en les variables de sincronització.

- Dissenyar i implementar un sistema de dependències de regió contigua amb el qual es pugui aprofitar una part del disseny del sistema de dependències puntuals.
- Integrar el runtime amb el model d'MPI per a treure el màxim de rendiment a les aplicacions paral·lelitzades amb OmpSs i MPI.
- Dissenyar i implementar la clàusula *commutative* d'OmpSs, donat que algunes aplicacions de l'avaluació final l'usen.
- Minimitzar l'ús de les operacions més costoses.
- Minimitzar el nombre de canvis de context i el cost de cada un.
- Utilitzar mecanismes de comunicació de baix impacte.
- Minimitzar l'ús de recursos de la màquina.

3.2 Abast

A continuació es detalla l'abast del projecte, tot respectant els requisits i objectius presentats a la secció 3.1. El primer pas serà documentar-se i familiaritzar-se amb les eines, el software i l'entorn on es durà a terme el projecte. Aquesta etapa es compondrà de tres fases:

- **Entendre el concepte de *runtime*:** Entendre la definició de *runtime*, les seves aplicacions i com hauria d'estar estructurat.
- **Familiaritzar-se amb OmpSs:** Consultar la documentació d'OmpSs, entendre les seves funcionalitats i com usar les seves directives. També es paral·lelitzarà alguna aplicació senzilla per a familiaritzar-s'hi.
- **Familiaritzar-se amb Argobots:** Consultar la documentació d'Argobots per a entendre cada una de les seves funcions i si és necessari, consultar el codi font. Per a familiaritzar-se amb la seva API[15], s'implementaran algunes aplicacions.

Un cop acabada aquesta etapa, es podrà començar a dissenyar l'estructura bàsica del nostre runtime. Per això, cal analitzar quines necessitats i requeriments ens demana OmpSs, i quines són les funcionalitats i estructures de dades d'Argobots que ens permeten implementar-les, és clar, de la forma més eficient. En aquesta etapa es desenvoluparan les funcions bàsiques d'OmpSs, com per exemple, la creació de tasques aniuades i la sincronització entre elles (o *taskwait*). També s'haurà d'implementar un conjunt de tests per a comprovar que funcioni correctament.

A la Figura 5 es pot observar l'estructura del runtime que es vol desenvolupar. Aquesta estructura compta amb un nombre determinat de streams, que normalment coincideix amb el nombre de nuclis disponibles de la màquina. A part, hi ha una pool global de tasques a on aquestes són inserides de forma dinàmica. Un cop un stream ha agafat una tasca de la pool global i l'executa, aquesta és assignada a la seva pool privada. Cal dir que la pool privada és més prioritària que la global, així que les tasques que ja han començat a executar-se i estan a una privada, tenen més prioritat que les tasques noves de la pool global. A més, cada stream té un scheduler assignat, que s'encarrega de decidir quina tasca s'executa en cada moment.

Figura 4: Flux de les tasques.

Seguidament, s'entrarà a l'etapa de la implementació de dependències puntuals i dependències de regió contigua i que a continuació es descriuen:

- **Dependències puntuals:** En aquesta fase s'haurà de dissenyar i implementar el model de dependències puntuals, que haurà de ser el màxim d'eficient possible, ja que serà una part molt important del runtime. S'haurà d'implementar un conjunt de tests robustos i fiables per a provar que aquesta implementació sigui correcta i que les tasques s'activin en el moment adient.
- **Dependències de regió contigua:** En aquesta fase s'haurà de dissenyar i implementar el model de dependències de regió contigua. Es buscarà un disseny innovador i eficient, però que alhora, reutilitzi el màxim de codi dels algorismes implementats en la fase de dependències puntuals. S'haurà de desenvolupar un conjunt de tests amb diferents casos complexos, per a poder provar que aquesta implementació és correcta.

  Com es pot veure a la Figura 5, aquest sistema de dependències forma part del runtime. A la Figura 4 es pot veure el cicle de vida de les tasques. Primer de tot, es declara la tasca i seguidament s'insereix en el sistema de dependències. Un cop totes les seves dependències han sigut satisfetes, la tasca s'insereix al sistema d'execució d'Argobots, és a dir, a la pool global.

Un cop acabada l'etapa de les dependències, s'estendrà el runtime d'Argobots dissenyant i implementant planificadors amb polítiques de planificació diferents de les que ja té per defecte.

Després, s'estudiarà com explotar la integració entre Argobots i MPI des d'OmpSs. S'implementarà una política de planificació que processi les peticions d'MPI. Les que siguin bloquejants, com per exemple *MPI_Recv*, seran interceptades i s'aprofitarà aquest temps d'espera executant altres tasques.

Aquesta integració és visible a la Figura 5, on es pot observar que hi ha una pool de peticions MPI. A mesura que es fan peticions bloquejants d'MPI, són inserides a la pool i la tasca es bloqueja. Llavors, els schedulers de forma periòdica, recorren les cues per comprovar si alguna ha acabat. D'aquesta forma, tots els streams poden executar tasques sense interrupció, independentment de si una d'elles està esperant dades o no.

Un cop finalitzada l'etapa anterior, es dissenyarà i implementarà la clàusula *commutative* de les tasques. Les tasques declarades amb aquesta clàusula poden ser executades en qualsevol ordre però no de forma concurrent. Per exemple, si es declaren tres tasques amb la clàusula *commutative* sobre una variable, es podran executar amb qualsevol ordre però en cap moment s'executaran alhora.

Per últim, s'executarà una sèrie de tests i benchmarks per poder comparar els resultats del rendiment i l'ús de recursos amb altres runtimes d'OmpSs, com Nanos++ o Nanos6. També s'executaran aplicacions MPI per veure l'impacte que té la intercepció de crides a MPI.

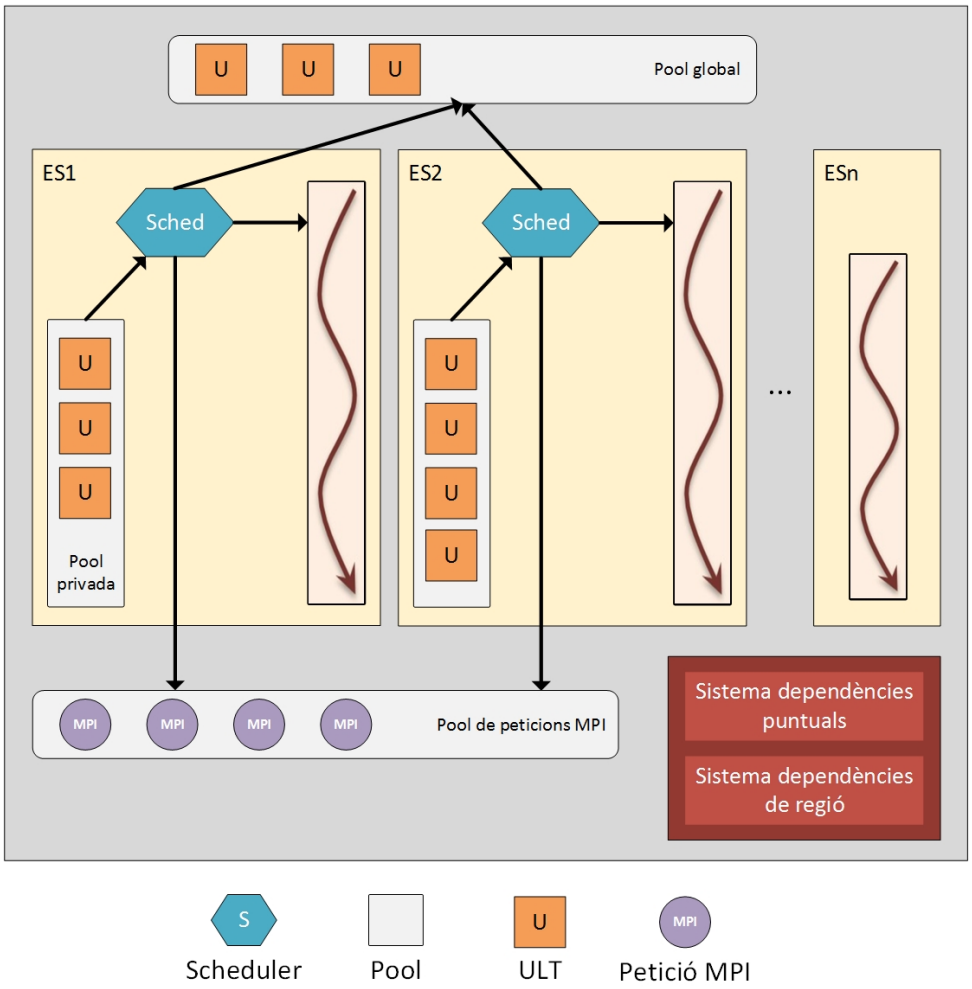

Figura 5: Dibuix on es mostra l'esquema general del runtime a desenvolupar.

3.3 Riscs i possibles solucions

En aquest apartat es detallen tots els possibles riscs i obstacles amb les seves respectives solucions. Abans de tot, cal entendre que en aquest projecte, un error greu i important, com podria ser el disseny erroni de la base o el *core* del runtime, podria comportar una pèrdua de temps molt gran i podria posar en risc la planificació temporal, i en el pitjor dels casos, el no compliment dels objectius i terminis. Per tant, és molt important la prevenció d'errors, i per aquest motiu, s'han programat reunions setmanals amb el director i el co-director, on es supervisarà el treball realitzat durant la setmana i el compliment dels terminis establerts.

3.3.1 Errors de disseny o implementació

Donat que aquest projecte requereix un disseny i una quantitat de codi considerable, poden aparèixer errors tant de disseny com d'implementació.

Solució: S'ha de dur a terme un mètode clar i eficaç per a prevenir, detectar i arreglar errors. Com ja s'ha comentat anteriorment, hi haurà reunions setmanals amb els directors del projecte per a supervisar-lo. A més, es dissenyaran uns casos de prova robustos i fiables per a comprovar que la implementació funciona correctament. Si es troba un error, s'haurà de *debuggar* amb les eines adequades per a detectar-lo i arreglar-lo de forma correcta i amb celeritat.

3.3.2 Augment del temps de les implementacions

Hi ha algunes funcionalitats complexes d'implementar i hi poden sorgir complicacions inesperades. Això pot provocar un augment de les hores dedicades i per tant, retards.

Solució: Una mesura fonamental és la previsió de les hores que se l'hi hauran de dedicar a cada funcionalitat i les possibles complicacions que puguin sorgir. En cas que ocorri, s'haurà de dedicar més hores diàries a solucionar el problema per evitar retards en la planificació.

3.3.3 Bugs en el runtime d'Argobots

Podria ser que es detectés un *bug* en el runtime d'Argobots, i donat que és una part fonamental del projecte, és molt important trobar una solució ràpidament.

Solució: Si es detecta un *bug* en el codi d'Argobots, primer de tot, es redactarà un informe amb la descripció del comportament erroni, un exemple senzill on es pugui veure clarament aquest comportament i les dades necessàries per a fer més fàcil la detecció de l'error. Aquest informe serà enviat immediatament a l'equip de desenvolupadors d'Argobots. En cas que l'error sigui fàcil d'arreglar, es farà una modificació del codi del runtime com a mesura temporal.

3.3.4 No disponibilitat de MareNostrum

Podria ser que degut a problemes tècnics, el supercomputador MareNostrum deixi d'estar disponible temporalment.

Solució: Aquest risc és un problema extern a aquest projecte. En cas que ocorri, no seria un problema crític, donat que només usarem MareNostrum per provar els tests i casos de prova, i només hi haurà guardada una còpia del projecte.

4 Metodologia i rigor

En aquest apartat es descriuen els mètodes de treball, les eines de seguiment, els mètodes de validació i finalment, l'avaluació del resultat final.

4.1 Mètodes de treball

Tenint en compte el poc temps del qual es disposa, la metodologia ideal seria una d'àgil com són *eXtreme Programming* (XP) o *SCRUM*. Tot i que aquestes metodologies estan destinades principalment a treballs en grup i no individuals, es poden extreure conceptes útils com ara:

- **Estratègia de desenvolupament incremental:** Aquest mètode és ideal en aquest projecte, ja que s'han plantejat vàries fases on s'implementaran diferents característiques. Això comporta que es puguin executar aplicacions reals en cada fase del projecte, i així, no s'haurà d'esperar a què estigui totalment acabat.
- **Desenvolupament en cicles curts:** Aquest mètode també és útil en aquest cas, donat que es proposaran objectius setmanals (revisats a cada reunió amb els directors) i així hi haurà un control més exhaustiu del compliment dels objectius i de la correctesa del software.
- **Desenvolupament guiat per proves:** Aquest mètode consisteix a trobar els requisits del software i convertir-los en tests. Seguidament, es modifica el software per a què passi satisfactòriament els tests, i quan ja funcionen, es refactoritza el codi. Aquesta tàctica permet desenvolupar un codi net i clar que funcioni i compleixi tots els requisits demandats.

4.2 Eines de seguiment

En aquest projecte s'usarà principalment el sistema de control de versions GIT[16]. És una eina molt potent, ja que té una gran quantitat de funcionalitats i opcions. Tot i que és difícil de dominar completament, les funcions que es necessitaran en aquest projecte són realment fàcils d'usar i permeten tenir un control clar de totes les versions del software. Una característica indirecta d'usar aquest sistema de control de versions és que t'obliga a documentar tots els canvis efectuats en el repositori remot. També proporciona una eina de *ticketing* on es pot reportar qualsevol tipus d'error o *bug* detectat en el software.

Perquè quedi constància del treball realitzat, s'habilitarà una pàgina compartida de *OneNote* (eina de *Microsoft*) per a poder escriure-hi l'avanç setmanal.

4.3 Mètodes de validació

És molt important plantejar una estratègia correcta de validació, ja que en aquest projecte és necessari comprovar que les diferents fases s'implementen de forma correcta. Si es detecta un error en una fase anterior, podria comprometre greument la correctesa de les fases següents. Per tant, és de vital importància que es desenvolupin uns tests robustos i fiables, que posin a prova tots els casos d'ús possibles.

Els tests de validació que s'usaran en el projecte són els tests que s'utilitzen per a verificar el runtime de Nanos6. Com que en aquest projecte es desenvoluparà una subpart de totes les funcionalitats d'OmpSs, només s'executaran els tests de la part desenvolupada. A més, s'afegiran tests propis en llenguatge C/C++ molt específics per posar a prova la implementació de les funcionalitats.

A part d'això, es planificaran reunions setmanals amb els directors per a comprovar que aquestes fases es duen a terme de forma correcta i sense errors, tant conceptuals com d'implementació. També s'usarà el sistema de correu electrònic per a comunicar-se amb els directors en cas de dubtes ràpids.

4.4 Avaluació del resultat final

Un cop acabades totes les fases de disseny, implementació i testeig, s'avaluarà el rendiment del runtime amb micro-benchmarks desenvolupats per mi mateix. També s'executarà el benchmark N-Body[25], que simula el moviment de partícules gràcies a la interacció entre elles a través de la força física, i un benchmark en Fortran anomenat IFS[26], que és un model global de predicció meteorològica operacional. En aquest moment es podran comparar els resultats amb els altres runtimes que implementen el model d'OmpSs, com Nanos++ i Nanos6.

5 Planificació temporal

En aquest apartat es detalla la descripció de les tasques i l'estimació temporal de cada una. També es descriuen les dependències entre tasques, així com el disseny del diagrama de Gantt. A més s'enumeren els diversos recursos que s'usaran. Per últim, es descriuen les desviacions i el pla d'actuació que es durà a terme.

5.1 Descripció de les tasques

En aquest apartat es descriuen les tasques, en ordre cronològic, que conformen el projecte.

5.1.1 Gestió del projecte

Aquesta tasca encapsula tots els entregables i l'aprenentatge autònom de l'assignatura GEP (Gestió de Projectes). Aquesta assignatura consta d'un total de 6 entregables. El temps de dedicació a l'assignatura és de 75 hores i es desglossa d'aquesta forma:

- **Contextualització i abast del projecte:** 24,50 hores.
- **Planificació temporal:** 8,25 hores.
- **Gestió econòmica i sostenibilitat:** 9,25 hores.
- **Presentació preliminar:** 6,25 hores.
- **Mòdul d'enginyeria de computadors:** 8,5 hores.
- **Presentació oral i document final:** 18,25 hores.

Per dur a terme aquesta tasca, els recursos necessaris són un ordinador, les eines de Microsoft Office i de Google, l'aplicació de ShareLatex.com, Gantter i un mòbil amb càmera.

5.1.2 Familiarització amb els models

El primer pas abans de començar l'anàlisi i disseny del projecte és familiaritzar-se amb els models que s'utilitzaran. En aquest cas, la familiarització ha de ser amb el model de programació paral·lela OmpSs i el del runtime d'Argobots. Per això, s'estudiaran les documentacions de cada programari i s'executarà una sèrie de tests per a entendre cada una de les funcionalitats d'aquests.

Per dur a terme aquesta tasca es necessita un ordinador on s'hi visualitzaran les documentacions, es programaran diversos codis d'exemple, es compilaran i s'executaran. També serà interessant executar-les en el supercomputador MareNostrum III i així, començar a familiaritzar-se amb aquesta computadora. Per tant, en aquest cas, l'ordinador haurà de tenir accés a Internet per establir connexió remota xifrada (ssh) amb el supercomputador.

5.1.3 Anàlisi del projecte

La tasca d'anàlisi del projecte depèn de l'anterior i se centrarà en fer una anàlisi exhaustiva del projecte, definir els objectius i els requisits principals d'aquest. També s'hi detallarà exactament quines funcionalitats d'OmpSs seran dissenyades i implementades.

A més, a partir dels requisits del projecte, es crearà un conjunt de problemes per a comprovar, una vegada acabat el projecte, que els objectius es compleixen.

Per últim, es pensarà a nivell molt conceptual i abstracte, el disseny de les funcionalitats d'OmpSs, que en la següent etapa, servirà com a base per a fer el disseny, en profunditat, i desenvolupament de cada una d'elles.

Per dur a terme aquesta tasca es necessiten les eines de Microsoft Office, les eines de Google i un ordinador amb connexió a Internet.

5.1.4 Configuració, disseny i desenvolupament

En aquest apartat s'hi encapsulen les tasques de configuració de l'entorn, el disseny i desenvolupament del projecte i l'avaluació del resultat final. Aquestes tasques són:

1. **Configuració de l'entorn:** Aquesta tasca depèn de la tasca d'Anàlisi del projecte i se centrarà en descarregar, instal·lar i configurar l'entorn on es desenvoluparà el projecte. També es crearà un repositori remot amb el sistema de control de versions GIT[16]. Donat el desconeixement d'algunes eines, es dedicaran les hores pertinents a l'aprenentatge d'aquestes per assegurar-ne l'ús correcte.
2. **Disseny i desenvolupament:** Un cop acabada la tasca anterior es procedirà a començar el disseny i desenvolupament dels objectius del projecte. Donada la complexitat i la grandària d'aquest, i tal com es va detallar en la secció de metodologia, es durà a terme una estratègia incremental. Per tant, a cada fase es farà el disseny i desenvolupament de les funcionalitats corresponents. Cal dir que, en el disseny de cada fase es tindran en compte les següents, per tal d'evitar al màxim els canvis en aquestes.

   A continuació, s'explica cada fase o tasca en detall:

   (a) **Estructura del runtime:** Aquesta és la primera tasca i se centrarà en dissenyar i implementar una estructura bàsica i òptima per a suportar la creació, l'execució i la sincronització de tasques.

   (b) **Sistema de dependències:** Aquesta tasca depèn de l'anterior, ja que requereix una estructura robusta i òptima del runtime com a base. Aquesta tasca es centrarà principalment en el disseny i desenvolupament de les dependències puntuals i de regió contigua. Cal afegir que el disseny de les dependències de regió dependrà del de les puntuals.

   (c) **Polítiques de planificació:** Aquesta tasca depèn directament de la tasca d'Estructura del runtime, ja que també requereix una estructura com a base. Aquesta tasca se centrarà en dissenyar i implementar una nova política de planificació, la qual Argobots encara no dispongui.

   (d) **Integració amb MPI:** Aquesta tasca se centrarà en dissenyar i desenvolupar una integració eficient entre MPI i Argobots des d'OmpSs. Donat que requerirà dissenyar una nova política de planificació per a comprovar les peticions MPI, es podran reaprofitar els dissenys de la tasca anterior. Això significa que dependrà directament de la tasca de Polítiques de planificació.

   (e) **Clàusula commutative**: Aquesta tasca se centrarà a dissenyar i implementar la clàusula commutative de les tasques. Aquesta tasca és addicional i s'ha afegit donat que l'aplicació Nbody[25], que s'executarà en l'avaluació final, usa aquesta clàusula en les seves tasques. Com és una tasca addicional, a la fita inicial no es va contemplat i només es té en compte a l'apartat 14, que és de revisió del projecte.

Les funcionalitats de cada fase seran validades com s'explicava en la secció de mètodes de validació. En una fase, quan totes les funcionalitats hagin sigut testejades i es confirmi que funcionen correctament, s'avançarà a la fase següent. D'aquesta manera a cada una es dispondrà d'un runtime funcional que podrà executar benchmarks reals, tot i que, limitats a les funcionalitats implementades.

3. **Avaluació del resultat final:** Un cop acabades totes les tasques de disseny i desenvolupament, es procedirà a avaluar els resultats com es descrivia en la secció d'avaluació final.

Per dur a terme aquestes tasques, s'usarà un ordinador amb el sistema operatiu OpenSUSE Leap 42.1 amb connexió a Internet, amb els editors VIM i Kate. A més, s'usarà el model d'OmpSs i Argobots, el compilador Mercurium, els runtimes Nanos++ i Nanos6, el model MPI, el sistema de control de versions GIT i les eines per a debuggar i fer profiling del codi desenvolupat. Per últim, s'accedirà al supercomputador MareNostrum III per executar-hi els tests de validació i fer l'avaluació final.

5.1.5 Etapa final

En la tasca final del projecte, es procedirà a redactar formalment la memòria del treball de final de grau i preparar la presentació oral davant del tribunal.

Per dur a terme aquesta tasca, s'usarà un ordinador amb Microsoft Windows 7 Professional, l'aplicació de ShareLatex.com per a redactar la memòria i les eines de Microsoft Office i de Google.

5.2 Definició de les dependències

A la Taula 1 es poden observar les dependències entre cada tasca del projecte.

Tasca	**Tasca predecessora**
Gestió del projecte	-
Familiarització amb els models	Gestió del projecte
Anàlisi del projecte	Familiarització amb els models
Configuració de l'entorn	Anàlisi del projecte
Disseny i desenvolupament	Configuració de l'entorn
Estructura del runtime	Configuració de l'entorn
Sistema de dependències	Estructura del runtime
Polítiques de planificació	Estructura del runtime
Integració amb MPI	Polítiques de planificació
Clàusula commutative	Integració amb MPI
Avaluació del resultat final	Disseny i desenvolupament
Etapa final	Avaluació del resultat final

Taula 1: Dependències entre tasques del projecte.

5.3 Estimació del temps i els rols de les tasques

A la Taula 2 es pot observar l'estimació del temps dedicat a cada tasca del projecte i els seus rols.

Tasca	**Responsable**	**Dedicació (hores)**
Gestió del projecte	Cap de projecte	75
Familiarització amb els models	Programador	25
Anàlisi del projecte	50% Analista / 50% Dissenyador	100
Configuració de l'entorn	Programador	10
Estructura del runtime		80
Disseny	Dissenyador	*40*
Implementació i test	40% Programador / 60% Tester	*40*
Sistema de dependències		120
Disseny	Dissenyador	*48*
Implementació i test	40% Programador / 60% Tester	*72*
Polítiques de planificació		5
Disseny	Dissenyador	*2*
Implementació i test	40% Programador / 60% Tester	*3*
Integració amb MPI		70
Disseny	Dissenyador	*20*
Implementació i test	40% Programador / 60% Tester	*50*
Clàusula commutative		20
Disseny	Dissenyador	*5*
Implementació i test	40% Programador / 60% Tester	*15*
Avaluació del resultat final	40% Programador / 60% Tester	10
Etapa final	Cap de projecte	60
Total		**575**

Taula 2: Estimació del temps dedicat a cada tasca del projecte i els seus rols.

5.4 Recursos

En aquest apartat es detallen els recursos que s'usaran al llarg de tot el projecte. Aquests recursos són recursos de software, hardware i humans:

5.4.1 Recursos software

Els recursos software que s'utilitzaran en aquest projecte són:

- **Software principal:** S'usarà el model de programació paral·lela OmpSs, el framework d'Argobots, el compilador Mercurium, els runtimes Nanos++ i Nanos6, i el model d'MPI.
- **Sistemes operatius:** El codi es desenvoluparà a OpenSUSE Leap 42.1. Les entregues i la memòria es redactaran a Microsoft Windows 7 Professional.
- **Eines d'ofimàtica:** S'usaran les eines de Microsoft (Word, Excel i Visio), les eines de Google (Drive, Documents i Draw) i l'aplicació de ShareLatex.com per a redactar les entregues i la memòria.
- **Eina de planificació:** S'utilitzarà l'eina Gantter per a dissenyar el diagrama de Gantt.
- **Editors de text:** S'utilitzaran els editors de text VIM i Kate.
- **Eines per a debuggar i de profiling:** Per a debuggar el codi i fer profiling s'utilitzarà el GDB, Extrae juntament amb Paraver i Valgrind.
- **Sistema de control de versions:** Per a tenir un control de versions s'utilitzarà GIT.

5.4.2 Recursos hardware

Els recursos hardware que s'utilitzaran en aquest projecte són:

- **Ordinador portàtil proporcionat pel BSC - CNS:** Dell Latitude E7440 amb quatre Intel® Core™ i7-4600U CPU @ 2.10GHz amb 8GB de memòria RAM.
- **Ordinador portàtil personal:** Acer Aspire V3-571G amb quatre Intel® Core™ i7-3632QM CPU @ 2.20GHz amb 8GB de memòria RAM.
- **Supercomputador MareNostrum III:** A la llista vigent del TOP500[17], la del Novembre de 2015, MareNostrum III està classificat en la 93ª posició. Proporciona un rendiment de pic de 1.1 PetaFLOPS, gràcies als 48,896 nuclis Intel® SandyBridge-EP E5-2670 repartits en un total de 3,056 nodes. També disposa de 42 nodes heterogenis amb un total de 84 acceleradors Xeon Phi 5110P. En el tema de memòria, disposa de 115.5 Terabytes de memòria principal i 2 Petabytes d'emmagatzematge en disc. Per interconnectar els nodes usa Infiniband FDR10 i Gigabit Ethernet.
- **Mòbil amb càmera:** S'usarà per a gravar el vídeo de *Presentació preliminar* de GEP.

5.4.3 Recursos humans

Els recursos humans que intervindran en aquest projecte són:

- **Desenvolupador del projecte:** És l'encarregat de desenvolupar el projecte.
- **Director i co-director del projecte:** Supervisaran el compliment dels objectius i dels terminis del projecte.

- **Enginyer de suport a la investigació:** Donarà suport i ajuda al desenvolupador quan sigui necessari.

5.5 Diagrama de Gantt

El diagrama de Gantt està repartit en dues imatges. A la Figura 6 hi ha la taula detallada del Gantt amb la duració, inici i final, els recursos usats, els rols i el risc de cada tasca. A la figura 7 es pot observar el gràfic del diagrama de Gantt.

5.6 Valoració d'alternatives i pla d'acció

El primer dia de defenses orals dels Treballs de Final de Grau és el dilluns 27 de juny del 2016 i la memòria del projecte s'ha d'entregar una setmana abans, concretament el dilluns 20 de juny del 2016. Per tant, donat que el projecte té planificat acabar-se el dijous 9 de juny de 2016, hi ha una setmana i mitja de marge.

La fase més sensible de patir desviacions temporals és l'etapa de *Disseny i desenvolupament*, concretament la de l'*Estructura del runtime* i la de *Sistema de dependències*. Aquestes desviacions poden ser per excés o bé per defecte, és a dir, es poden dedicar més hores que les planificades, o per contra, es poden dedicar menys. En el cas que es detecti una desviació de més d'una setmana, s'aplicaran mesures correctives i es durà a terme una replanificació.

Les mesures correctives que es podran aplicar són cancel·lar la fase que s'està desenvolupant o simplificar-la, en cas que aquesta fase no sigui cap de les vitals (*Estructura del runtime* i *Sistema de dependències*). En cas de que l'etapa sigui vital, es podran cancelar o simplificar les següents etapes menys importants.

A més, com s'ha explicat a la secció 4.3, s'han planificat reunions setmanals amb els directors per evitar, principalment, aquestes desviacions i les seves conseqüències. Per tant, gràcies al control, el marge de dies i les mesures correctives, s'assegura una correcta finalització del projecte a temps, i per tant, sense una afectació greu als recursos assignats.

Nombre de tarea	Duración	Comienzo	Fin	Nombres de los recursos	Responsabilidad	Riesgo
Gestió del projecte	**40 días?**	**lun 22/02/16**	**vie 01/04/16**	**Eines d'ofimàtica;Gantter;PC Acer;PC Dell;Samsung Galaxy S5;Windows 7**	**Cap de projecte**	**Baix**
Contextualització i abast	10 días	lun 22/02/16	mié 02/03/16		Cap de projecte	Baix
Planificació temporal	6 días?	jue 03/03/16	mar 08/03/16		Cap de projecte	Baix
Gestió econòmica i sostenibilitat	7 días?	mié 09/03/16	mar 15/03/16		Cap de projecte	Baix
Presentació preliminar	5 días	mié 16/03/16	dom 20/03/16		Cap de projecte	Baix
Mòdul d'Enginyeria Computadors	12 días	lun 21/03/16	vie 01/04/16		Cap de projecte	Baix
Presentació oral i document final	12 días	lun 21/03/16	vie 01/04/16		Cap de projecte	Baix
Milestone: Document final i transparències GEP	0 días	vie 01/04/16	vie 01/04/16			
Familiarització amb els models	**4 días**	**sáb 02/04/16**	**mar 05/04/16**	**Editors de text;Eines de debug i profiling;Marenostrum;Models de prog;OpenSUSE;PC Acer;PC Dell**	**Programador**	**Baix**
Anàlisi del projecte	**14 días**	**mié 06/04/16**	**mar 19/04/16**	**Eines d'ofimàtica;OpenSUSE;PC Acer;PC Dell**	**50% Analista / 50% Dissenyador**	**Baix**
Milestone: Presentació oral GEP	0 días	mar 19/04/16	mar 19/04/16			
Configuració de l'entorn	**2 días**	**mié 20/04/16**	**jue 21/04/16**	**Editors de text;GIT;Marenostrum;Models de prog;OpenSUSE;PC Acer;PC Dell**	**Programador**	**Baix**
Disseny i desenvolupament	**39 días?**	**vie 22/04/16**	**lun 30/05/16**	**Editors de text;Eines de debug i profiling;GIT;Marenostrum;Models de prog;OpenSUSE;PC Acer;PC Dell**		
Estructura del runtime	**11 días**	**vie 22/04/16**	**lun 02/05/16**			**Mig**
Disseny	5 días	vie 22/04/16	mar 26/04/16		Dissenyador	Mig
Implementació i test	6 días	mié 27/04/16	lun 02/05/16		40% Programador / 60% Tester	Mig
Sistema de dependències	**17 días**	**mar 03/05/16**	**jue 19/05/16**			**Alt**
Disseny	6 días	mar 03/05/16	dom 08/05/16		Dissenyador	Alt
Implementació i test	11 días	lun 09/05/16	jue 19/05/16		40% Programador / 60% Tester	Alt
Polítiques de planificació	**4 días**	**vie 20/05/16**	**lun 23/05/16**			**Baix**
Disseny	2 días	vie 20/05/16	sáb 21/05/16		Dissenyador	Baix
Implementació i test	2 días	dom 22/05/16	lun 23/05/16		40% Programador / 60% Tester	Baix
Integració amb MPI	**7 días**	**mar 24/05/16**	**lun 30/05/16**			**Baix**
Disseny	3 días	mar 24/05/16	jue 26/05/16		Dissenyador	Baix
Implementació i test	4 días	vie 27/05/16	lun 30/05/16		40% Programador / 60% Tester	Baix
Avaluació del resultat final	**3 días**	**mar 31/05/16**	**jue 02/06/16**	**Editors de text;Eines de debug i profiling;Marenostrum;Models de prog;OpenSUSE;PC Acer;PC Dell**	**40% Programador / 60% Tester**	**Baix**
Etapa final	**7 días**	**vie 03/06/16**	**jue 09/06/16**	**Eines d'ofimàtica;Gantter;PC Acer;PC Dell;Windows 7**	**Cap de projecte**	**Baix**
Milestone: Entrega memòria TFG	0 días	lun 20/06/16	lun 20/06/16			
Milestone: Defensa oral TFG	0 días	lun 27/06/16	lun 27/06/16			

Figura 6: Taula de Gantt amb la duració, dia inicial i final, recursos, rols i risc de les tasques.

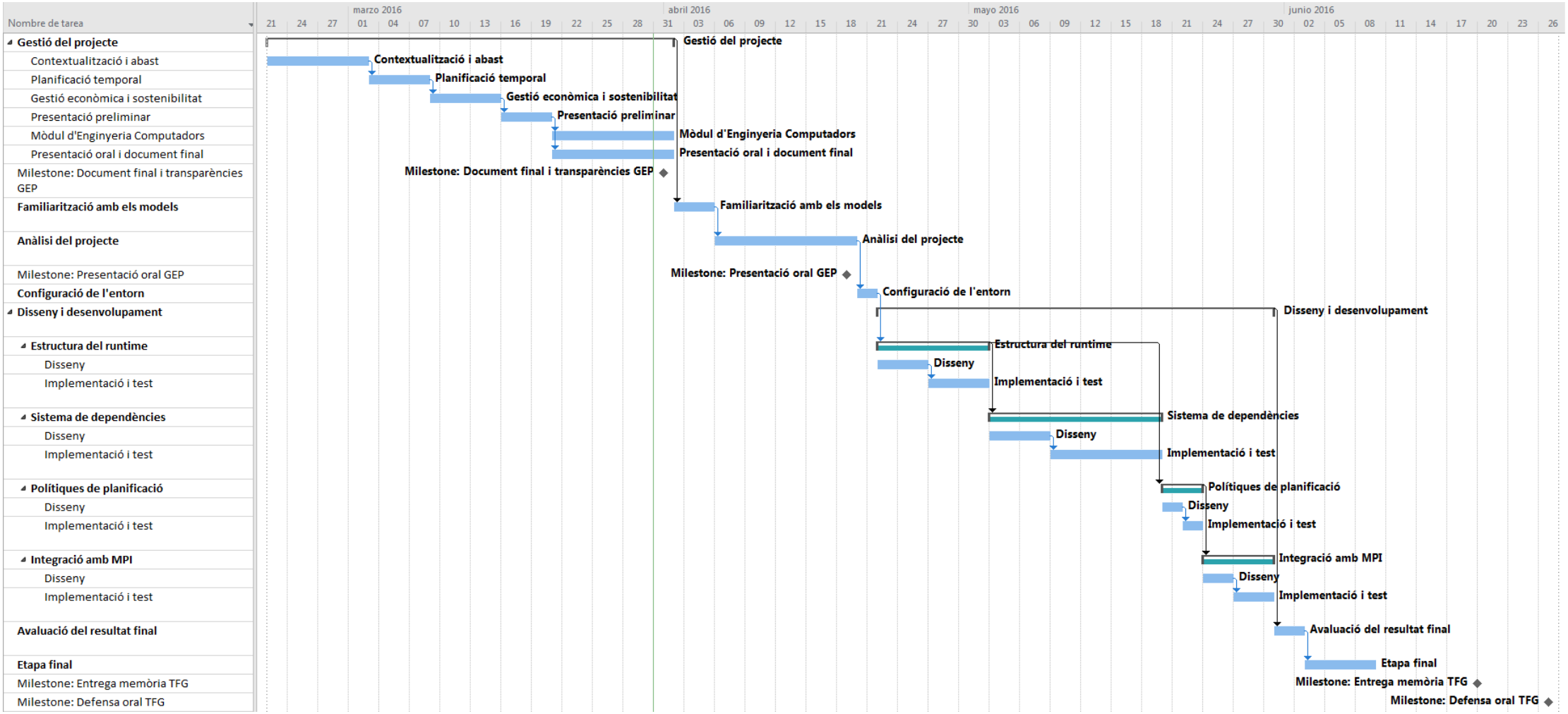

Figura 7: Diagrama de Gantt de la planificació temporal.

6 Gestió econòmica

6.1 Costos del projecte

En aquesta secció es detallen els costos del projecte, desglossats en diversos apartats per organitzar-los clarament. Un cop explicats, es calcula el pressupost final del projecte.

6.1.1 Costos dels recursos software

A la Taula 3 es desglossen els diversos recursos software amb els preus i les amortitzacions.

Software	**Preu(€)**	**Unitats**	**Vida útil**	**Amortització(€/h)**
Microsoft Windows 7 Professional	142.00 [18]	2	3 anys	Inclòs al PC
OpenSUSE Leap 42.1	0	2	3 anys	-
Microsoft Office 365 Home	69.95 [19]	2	3 anys	0.02
Eines de Google (Dirve, Docs...)	0	1	-	-
ShareLatex.com	0	1	-	-
Gantter	0	1	-	-
GIT	0	1	-	-
Editors de text (VIM i Kate)	0	1	-	-
OmpSs	0	1	-	-
Argobots	0	1	-	-
Compilador Mercurium	0	1	-	-
Runtimes Nanos++ i Nanos6	0	1	-	-
MPI	0	1	-	-
Eines de debug (GDB i Valgrind)	0	1	-	-
Eines de profiling (Extrae i Paraver)	0	1	-	-
Total	**423.90**			**0.02**

Taula 3: Costos dels recursos software.

6.1.2 Costos dels recursos hardware

A la Taula 4 es desglossen els diversos recursos hardware amb els preus i amortitzacions corresponents. Cal afegir que l'hora de computació a Marenostrum III no és gratuïta, però com que són dades privades, no s'ha pogut determinar. El preu del Dell Latitude E7440 ha sigut proporcionat pel BSC i el preu del Acer Aspire V3-571G va ser pressupostat per majorista.

Hardware	**Preu(€)**	**Unitats**	**Vida útil**	**Amortització(€/h)**
Dell Latitude E7440	1,600.00	1	4 anys	0.2
Acer Aspire V3-571G	799.00	1	4 anys	0.1
Marenostrum III	-	1	-	-
Samsung Galaxy S5	499.00 [20]	1	4 anys	0.07
Total	**2,898.00**			**0.37**

Taula 4: Costos dels recursos hardware.

6.1.3 Costos dels recursos humans

A la Taula 5 es desglossen els diversos recursos humans amb els costos corresponents.

Recurs humà	**Dedicació(hores)**	**Preu per hora(€/h)**	**Cost total(€)**
Director del projecte	30	50.00 [21]	1,500.00
Co-director del projecte	30	50.00 [21]	1,500.00
Enginyer suport investigació	70	20.00 [21]	1,400.00
Becari de col·laboració (Jo mateix)	560	-	2,000.00 [22]
Total	**690**		**6,400.00**

Taula 5: Costos dels recursos humans.

6.1.4 Costos directes per a cada activitat

Com es va indiciar en el diagrama de Gantt, cada activitat o tasca té una sèrie de recursos necessaris per ser duta a terme. A continuació s'enumeren aquests recursos i a la Taula 6 es formalitzen les necessitats de cada activitat en aquest àmbit.

1. Dell Latitude E7440
2. Acer Aspire V3-571G
3. Marenostrum III
4. Mòbil amb càmera
5. Microsoft Windows 7 Professional
6. OpenSUSE Leap 42.1
7. Microsoft Office 365 Home
8. Eines de Google (Drive, Docs...)
9. ShareLatex.com
10. Gantter
11. GIT
12. Editors de text (VIM i Kate)
13. OmpSs
14. Argobots
15. Compilador Mercurium
16. Runtimes de Nanos++ i Nanos6
17. MPI
18. Eines de debug (GDB i Valgrind)
19. Eines de profiling (Extrae i Paraver)

Tasca/Activitat		**Duració (h)**	**Recursos necessaris**
Gestió del projecte		75	1, 2, 4, 5, 7, 8, 9 i 10
Familiarització amb els models		25	1, 2, 3, 6, 12, 13, 14, 15, 16, 17 i 18
Anàlisi del projecte		100	1, 2, 5, 6, 7 i 8
Configuració de l'entorn		10	1, 2, 3, 6, 11, 12, 13, 14, 15, 16 i 17
Disseny i desenvolupa.	Estructura del runtime	80	1, 2, 3, 6, 11, 12, 13, 14, 15 i 18
	Sistema de dependències	120	1, 2, 3, 6, 11, 12, 13, 14, 15 i 18
	Polítiques de planificació	30	1, 2, 3, 6, 11, 12, 13, 14, 15 i 18
	Integració amb MPI	50	1, 2, 3, 6, 11, 12, 13, 14, 15, 17 i 18
Avaluació del resultat final		10	1, 2, 3, 6, 11, 12, 13, 14, 15, 16, 17 i 19
Etapa final		60	1, 2, 5, 7, 8, 9 i 10

Taula 6: Costos directes per a cada activitat.

6.1.5 Costos indirectes

Els costos de disposar d'una taula a una oficina del BSC i els costos que comporta indirectament, com són l'electricitat i l'aigua, són informació privada i confidencial del centre. Donat que no s'ha pogut recuperar aquesta informació, se n'ha fet una estimació a la Taula 7.

Costos indirectes	**Preu per mes(€/mes)**	**Mesos**	**Cost total(€)**
Despeses generals oficina	40.00	5	200.00
TJove de 6 zones	100.00 [23]	5	500.00
Quota Fibra òptica	27.32 [24]	5	136.60
Total			**836.60**

Taula 7: Estimació dels costos indirectes.

6.1.6 Imprevistos i contingències

En aquest apartat es detallen els possibles imprevistos i el seu cost. Només es calculen els imprevistos de les activitats que són més susceptibles de patir desviacions, ja siguin, en la part de disseny com en la part d'implementació. Aquestes són les activitats *Estructura del runtime* i *Sistema de dependències*. Donada l'alta complexitat i la importància d'ambdues activitats, s'ha fixat un factor de risc de 10% i 15%, respectivament.

En aquest apartat també es calcula el percentatge de la contingència. Per aquest projecte és d'un 10%, ja que, el pressupost és molt detallat i la majoria dels recursos necessaris són gratuïts. A més, les possibles desviacions que s'han descrit podrien comportar com a màxim un augment d'unes 5 hores de dedicació del director i co-director. Per tant, el cost extra podria comportar uns *2 x 5 hores x 50 €/h = 500 €*, que es cobreixen amb aquest percentatge.

6.1.7 Pressupost

A la Taula 8 es pot observar el pressupost del projecte, indicant només els recursos no gratuïts. S'ha tingut en compte que el desenvolupador (jo mateix) compleix tots els rols: cap de projecte, analista, dissenyador, programador i tester. Els costos directes de les tasques Estructura del runtime i Sistema de dependències són respectivament, 24€ i 36€ (calculat a partir del la dedicació en hores pel total de les amortitzacions dels recursos usats per cada tasca).

6.2 Control de gestió

Al llarg del projecte es poden produir desviacions, que es corregiran seguint el pla d'acció de la planificació. Però és important definir uns mecanismes que permetin calcular aquestes desviacions respecte al pressupost.

La millor manera de portar-ho a terme és calcular els costos reals al final de cada fase del projecte, tenint en compte les hores dedicades i comparant-ho amb el pressupost establert en aquest projecte. També és important fer un control exhaustiu a les fases de més risc, que són la d'*Estructura del runtime* i la de *Sistema de dependències*. Si es produeix una diferència considerable, ja sigui per passar-se del pressupost o per quedar-se curt, es realitzarà un estudi per trobar el motiu d'aquesta diferència i s'intentaran corregir les estimacions del pressupost de les següents fases del projecte.

	Unitats	Amortització(€/h)	Temps(hores)	Total(€)
Costos directes				**190.45**
Gestió del projecte			**75**	**30.75**
Dell Latitude E7440	1	0.2		15.00
Acer Aspire V3-571G	1	0.1		7.50
Mòbil amb càmera	1	0.07		5.25
Microsoft Office 365	2	0.02		3.00
Familiarització amb els models			**25**	**7.50**
Dell Latitude E7440	1	0.2		5.00
Acer Aspire V3-571G	1	0.1		2.50
MareNostrum III	1	-		-
Anàlisi del projecte			**100**	**34.00**
Dell Latitude E7440	1	0.2		20.00
Acer Aspire V3-571G	1	0.1		10.00
Microsoft Office 365	2	0.02		4.00
Configuració de l'entorn			**10**	**3.00**
Dell Latitude E7440	1	0.2		2.00
Acer Aspire V3-571G	1	0.1		1.00
MareNostrum III	1	-		-
Disseny i desenvolupament			**280**	**91.80**
Dell Latitude E7440	1	0.2		56.00
Acer Aspire V3-571G	1	0.1		28.00
MareNostrum III	1	-		-
Improvistos				*7.80*
Estructura del runtime		10% del cost de la subtasca: 24.00 €		2.40
Sistema de dependències		15% del cost de la subtasca: 36.00 €		5.40
Avaluació del resultat final			**10**	**3.00**
Dell Latitude E7440	1	0.2		2.00
Acer Aspire V3-571G	1	0.1		1.00
MareNostrum III	1	-		-
Etapa final			**60**	**20.40**
Dell Latitude E7440	1	0.2		12.00
Acer Aspire V3-571G	1	0.1		6.00
Microsoft Office 365	2	0.02		2.40
Costos indirectes				**836.60**
Costos de recursos humans				**6,400.00**
Total acomulat				**7,427.05**
Contingència	**10%**	Aplicat a: **7,427.05 €**		**742.71**
Total sense *IVA*				**8,169.76**
Total amb *IVA*	**21%**	Aplicat a: **8,169.76 €**		**9,885.41**

Taula 8: Pressupost del projecte amb els recursos no gratuïts.

Al final del projecte també es farà un estudi de les diferències de totes les fases, i un cop sumades, si aquesta diferència total sobrepassa el pressupost, s'haurà de fer ús del fons de contingència per cobrir els costos.

Per calcular aquestes desviacions es farà de la següent manera:

- **Desviacions per cada concepte:**
  - Desviació de la mà d'obra per preu = (cost std - cost real) * consum d'hores real.
  - Desviació a la realització d'una tasca per preu = (cost std - cost real) * consum d'hores real.
  - Desviació d'un recurs per preu = (cost std - cost real) * consum real.
  - Desviació a la realització d'una tasca per consum = (consum std - consum real) * cost real.
- **Desviacions per cada import total:**
  - Desviació total a la realització de tasques = cost total std tasca - cost total real tasca.
  - Desviació total en recursos = cost total std recursos - cost total real recursos.
  - Desviació total costs fixos = cost total fix pressupostat - cost total fix real.

Tot i així, donat que les tasques, els recursos i els costos han sigut descrits molt detalladament, és poc probable que al final del projecte la diferència entre el pressupost i el cost real sigui molt diferent.

7 Sostenibilitat i compromís social

En aquest apartat es tracta la sostenibilitat i el compromís social d'aquest projecte. S'avaluarà la sostenibilitat considerant la dimensió ambiental, econòmica i social per separat. Per últim, es presenta la matriu de sostenibilitat de la fita inicial del projecte.

7.1 Dimensió ambiental

En general, l'impacte ambiental que tindrà el desenvolupament del projecte serà baix. Com es pot observar a la Taula 6, els recursos necessaris són de baix consum. Tot i això, cada cop que s'implementi una funcionalitat, serà provada en el supercomputador Marenostrum III. Aquest és un recurs hardware amb un alt consum d'electricitat, però donat que s'hi executen diverses aplicacions en paral·lel, és gairebé impossible computar el consum d'electricitat d'aquest projecte. Tot i això, el cosum dels recursos com els portàtils serà negligible en comparació amb el consum del Marenostrum.

L'objectiu del projecte és avaluar si un runtime basat en Argobots pot ser més eficient que els actuals. Per tant, si mostra alguna millora, l'execució de les aplicacions paral·leles serà més ràpida i el consum elèctric serà inferior, tot i que, aquesta disminució del consum no es pot quantificar. La contaminació que es pugui generar dependrà del mètode de producció energètica que usi el proveïdor.

Durant la vida útil d'aquest projecte segurament s'executarà al Marenostrum, ja que és l'entorn apropiat d'aquest projecte. Tot i que el consum de Marenostrum és considerablement gran, si el runtime aprofita bé els recursos pot ser que millori el temps d'execució i per tant, el consum energètic de les aplicacions disminuirà.

El risc que hi pot haver és que Marenostrum se l'hi actualitzi el hardware i gasti més consum per unitat de computació, tot i que és poc probable, ja que la tendència és de baixar el consum.

7.2 Dimensió econòmica

A la secció 6 es detallen els costos de la realització del projecte, tant dels recursos humans com dels recursos materials. També es tenen en compte els possibles imprevistos de les activitats amb més risc i es destina una contingència adient per evitar que els costos reals sobrepassin el pressupost. Donat que és un projecte d'investigació, orientat a la comunitat científica, no ha de ser competitiu. Per tant, no es pot avaluar la viabilitat en termes de competitivitat. Cal destacar que aquest projecte col·labora amb el BSC - CNS i el JLESC[5].

Per altra banda, aquest projecte es podria realitzar de forma més ràpida però no amb menys recursos. Tot i que la distribució d'hores és la més ajustada possible, un desenvolupador amb més experiència amb els models de programació i arquitectures paral·leles, podria dur-lo a terme més ràpid. No obstant això, implicaria un major cost de recursos humans, ja que aquest desenvolupador tindria un sou major i per tant, el cost total seria igual o més alt. Cal afegir que la dedicació a cada tasca és proporcional a la seva importància en el projecte.

S'ha calculat el cost final del projecte i és el mateix cost que es va calcular a la fita inicia. El cost del projecte és 9,885.41 euros. Donat que s'ha utilitzat el mínim de recursos possibles i de la forma més eficient, no es pot estalviar en tema de costos. Durant la vida útil del projecte, aquest només haurà de ser mantingut per un desenvolupador, ja que poden aparèixer bugs i problemes.

Un dels problemes que poden perjudicar la viabilitat econòmica del projecte és l'aparició d'un error de concepte en el disseny de la solució i que impliqui un canvi considerablement gran de tot el projecte.

7.3 Dimensió social

Actualment, Espanya ha iniciat un creixement econòmic després de diversos anys de crisi profunda, tot i que, encara no és perceptible en l'economia dels ciutadans. A part, a Catalunya hi estan havent canvis molt destacats, ja que s'ha iniciat un procés de ruptura amb Espanya. Aquests dos fets però, sembla que no han afectat de forma negativa al sector informàtic, ja que ha sigut dels únics que han seguit creixent i generant ocupació. Però segurament, aquest projecte no tindrà cap repercussió a la situació actual, com tampoc perjudicarà a ningú.

S'ha vist que l'ús d'Argobots pot comportar una millora en el temps d'execució de les aplicacions dels usuaris i investigadors, i això pot fer que millori la qualitat de vida d'aquests i també dels col·lectius que depenen dels resultats de les aplicacions. Per exemple, si s'està investigant una malaltia, podria ser que es trobessin resultats de forma més ràpida i beneficiar els metges i pacients.

La solució al problema és bona però segurament no serà la millor que hi hagi, donat que l'experiència és un factor molt important en aquests casos. A més, per resoldre un problema d'aquesta magnitud es necessiten molts més mesos dels que hi ha per fer el treball de final de grau.

Personalment, aquest projecte ha canviat la meva manera de veure el consum dels recursos que ens proveeix el hardware, i per ara endavant, portar a terme una programació conscient de l'arquitectura, aprofitant al màxim els recursos.

7.4 Matriu de sostenibilitat

A la Taula 9 es pot observar la matriu de sostenibilitat de la fita inicial i final del TFG.

	Projecte en producció	**Vida útil**	**Riscs**
Ambiental	Consum del disseny	Empremta ecològica	Riscs ambientals
Valoració	*6:10*	*15:20*	*0:0*
Econòmic	Factura	Pla de viabilitat	Riscs econòmics
Valoració	*8:10*	*15:20*	*0:0*
Social	Impacte personal	Impacte social	Riscs socials
Valoració	*10:10*	*12:20*	*0:0*
Valoració parcial	*24:30*	*42:60*	*0:0*
Valoració total	**66:90**		

Taula 9: Matriu de sostenibilitat de la fita inicial i final del TFG.

8 Base inicial del projecte

En aquest apartat es detalla la base inicial sobre on es dissenyarà i desenvoluparà el runtime que suporta OmpSs i les funcionalitats esmentades anteriorment.

8.1 Application Programming Interface (API) del runtime

Una interfície de programació d'aplicacions o API és el conjunt de subrutines i funcions que ofereix una llibreria per ser utilitzada per un altre software com una capa d'abstracció.

Com s'ha explicat anteriorment, el desenvolupament d'aquest runtime s'ha fet paral·lelament amb el del nou runtime Nanos6, desenvolupat al Barcelona Supercomputing Center (BSC). Anteriorment a aquest projecte, al BSC es va dissenyar de nou l'API del runtime Nanos6, que és totalment diferent de la dels runtimes antics.

L'API del runtime és el conjunt de subrutines i funcions que ofereix la biblioteca que forma el runtime i és utilitzat pel compilador Mercurium. Aquest, a l'hora de compilar un codi d'usuari paral·lelitzat amb OmpSs, converteix les directives i clàusules a crides a l'API del runtime, aconseguint d'aquesta manera que el codi d'usuari cridi a les funcions del runtime. És important destacar que aquest procés és totalment transparent per l'usuari.

A la Figura 8 es pot observar un resum de l'API de Nanos6. Aquesta disposa de les funcions `nanos_create_task` i `nanos_submit_task` que permeten la creació de les tasques. Aquestes tasques també poden registrar dependències de lectura, escriptura, lectura-escriptura i commutative amb les funcions `nanos_register_{type}_depinfo` on `type` pot ser `read`, `write`, `readwrite` o `commutative`. A part, també existeix la funció `nanos_taskwait` que permet a les tasques esperar a les tasques filles directes usant la directiva taskwait.

Per suportar la interoperabilitat amb MPI, s'ha afegit la funció `nanos_polling_cond_wait` amb la qual s'afegeix una condició a la cua de condicions del runtime i es duu a terme una espera bloquejant sobre ella. Tant la funció com els seus usos seran explicats amb més detall en els següents apartats.

Per últim, l'API té les funcions de `nanos_preinit`, `nanos_init`, `nanos_wait_unitl_shutdown` i `nanos_notify_ready_for_shutdown` per inicialitzar i finalitzar el runtime, les quals seran explicades en detall en el següent apartat.

8.2 Carregador de la llibreria del runtime

Un dels temes més importants actualment són els de la compatibilitat i la portabilitat del software. En aquest cas, és molt important que un codi d'usuari tingui la possibilitat d'executar-se amb el runtime de Nanos6 o amb el runtime d'aquest projecte sense haver de recompilar-lo. Per aquest motiu, i gràcies a què els dos comparteixen la mateixa API, en el BSC s'ha dissenyat i desenvolupat una llibreria la qual gràcies a una variable d'entorn, carrega un runtime o l'altre. Això permet que un codi amb OmpSs compilat amb Mercurium pugui triar en temps d'execució quin runtime es vol executar. Per exemple, si es vol executar el codi paral·lelitzat `fibonacci.cpp` amb el runtime d'aquest projecte, s'hauran d'executar les següents comandes:

```
$ mcxx fibonacci.cpp -o fibonacci
$ NANOS6=argobots ./fibonacci
```

Si després es vol executar l'aplicació amb el Nanos6 oficial, s'haurà d'executar la comanda:

```
$ NANOS6=nanos6 ./fibonacci
```

Aquesta propietat permet que el procés de comparar execucions d'aplicacions amb els diferents runtimes sigui molt més fàcil i més ràpid, donat que no s'ha de recompilar cap codi.

```
// Allocate space for a task and its parameters
void nanos_create_task(
  nanos_task_info *task_info,
  nanos_task_invocation_info *task_invocation_info,
  size_t args_block_size,
  void **args_block_pointer,
  void **task_pointer
);

// Submit a task
void nanos_submit_task(void *task);

// Block the control flow of the current task waiting all its children
void nanos_taskwait(char const *invocation_source);

// Register a task read access on linear range of addresses
void nanos_register_read_depinfo(void *handler, void *start, size_t len);

// Register a task write access on linear range of addresses
void nanos_register_write_depinfo(void *handler, void *start, size_t len);

// Register a task readwrite access on linear range of addresses
void nanos_register_readwrite_depinfo(void *handler, void *start, size_t len);

// Register a task commutative access on linear range of addresses
void nanos_register_commutative_depinfo(void *handler, void *start, size_t len);

// Add a condition to the condition queue of the runtime and wait for it
void nanos_polling_cond_wait( void *condition )

// Initialize the runtime at least to the point that it will accept tasks
void nanos_preinit(void);

// Continue with the rest of the runtime initialization
void nanos_init(void);

// Wait until the the runtime has shut down
void nanos_wait_until_shutdown(void);

// Notify the runtime that it can begin the shutdown process
void nanos_notify_ready_for_shutdown(void);
```

Figura 8: Resum de l'API del runtime Nanos6.

8.3 Cicle de vida del runtime

Dins de l'execució d'una aplicació d'OmpSs, el runtime té un cicle de vida on s'inicia, s'espera que tota l'aplicació acabi i llavors finalitza. Aquest procés és relativament complex donat que realment és el runtime que inicia l'execució de l'aplicació i no al revés. Primer de tot, el loader carrega la llibreria del runtime i un cop carregada correctament, crida a la funció `nanos_preinit` (veure Figura 8).

Aquesta funció inicialitza totes les estructures internes del runtime i inicialitza els streams, els schedulers i les pools d'Argobots per tal de poder començar a crear i executar tasques. A partir d'ara, l'user-level thread que inicialitza i finalitza les estructures del runtime s'anomenarà tasca runtime, tot i no haver sigut creada explícitament.

Un cop el runtime està llest, el loader crea una tasca anomenada `main` que encapsula una funció especial, explicada en el següent paràgraf. Un cop creada, crida la funció `nanos_init` que en aquest cas no duu a terme cap acció. Llavors, aquesta tasca runtime crida la funció `nanos_wait_until_shutdown`, la qual espera a què la tasca `main` acabi i un cop acabada, allibera totes les estructures internes i les d'Argobots, i finalitza el runtime.

La tasca main té assignada una funció que primer de tot, executa la funció `main` del codi d'usuari, llavors espera a les tasques filles que ha anat creant durant l'execució amb un taskwait i per últim, avisa a la tasca runtime de què ha acabat, cridant la funció `nanos_notify_ready_for_shutdown`.

A la Figura 9 es pot observar de forma gràfica el cicle de vida de la tasca runtime i de la main, explicat anteriorment.

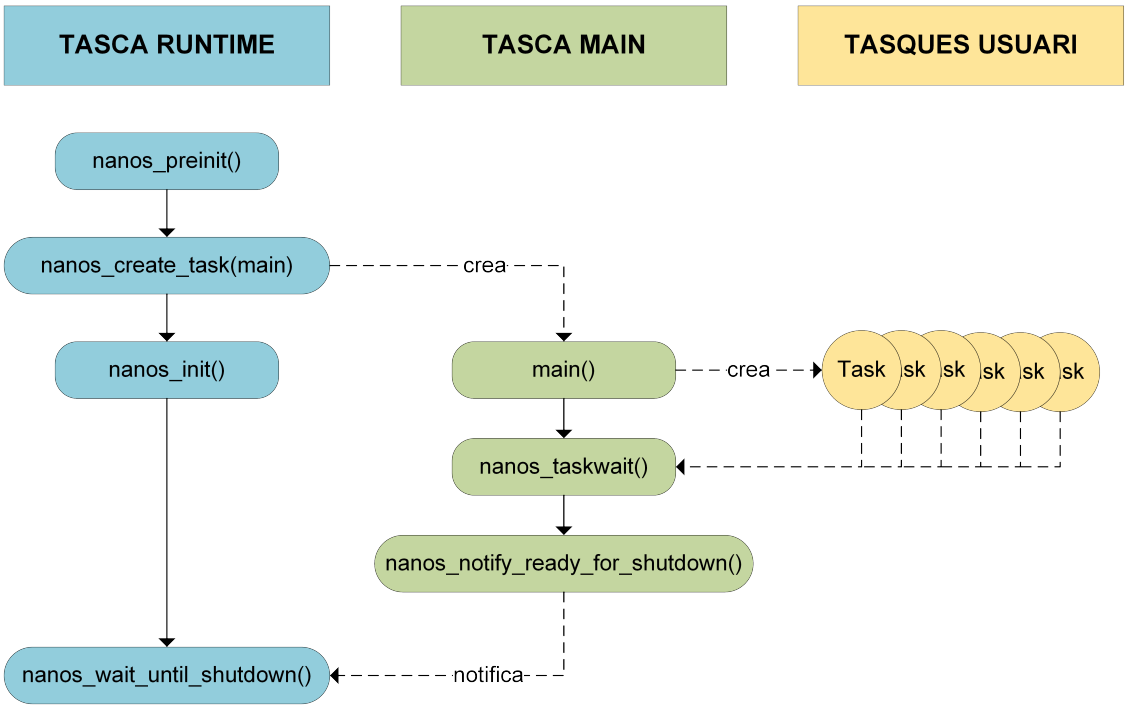

Figura 9: Dibuix del cicle de vida del runtime.

9 Disseny i implementació de les funcionalitats

En aquest apartat s'explica el disseny i la implementació de cada funcionalitat prèviament enumerada. L'apartat es divideix en sis subapartats, en el primer s'expliquen les estructures usades per formar la base estructural del runtime, seguidament es descriu com s'ha implementat el sistema de dependències puntuals i de regió contigua. Llavors s'explica el canvi que s'ha aplicat a l'scheduler per millorar la localitat de les dades. Després es descriu detalladament de què tracta la interoperabilitat amb el model de programació paral·lela de memòria distribuïda MPI i els canvis que s'han dut a terme per adaptar el runtime a aquesta funcionalitat. Per últim s'explica la clàusula d'OmpSs anomenada commutative i la seva implementació.

9.1 Estructura bàsica del runtime

En aquest apartat és on vertaderament comença la part important d'aquest projecte. S'ha de proposar un disseny eficient que sigui capaç de proporcionar una base sòlida en la qual s'implementaran la resta de funcionalitats. Els objectius principals d'aquest apartat són permetre l'execució de tasques, possiblement aniuades, i permetre la sincronització entre tasques gràcies a la directiva taskwait. Per tant, les funcions de l'API que es desenvolupen en aquest apartat són `nanos_create_task`, `nanos_submit_task` i `nanos_taskwait`, però també les d'inicialització i finalització del runtime, les quals són `nanos_preinit`, `nanos_init`, `nanos_wait_until_shutdown` i `nanos_notify_ready_for_shutdown`.

9.1.1 Inicialització del runtime

Aquesta fase es duu a terme a la funció de l'API anomenada `nanos_preinit`. Per poder executar tasques es necessita un conjunt de streams d'Argobots, cada un dels quals ha d'estar assignat a un únic thread hardware. El nombre d'streams no hauria de superar mai el nombre de threads hardware disponibles per aquell procés, donat que si no fos així, podria haver-hi dos o més streams intentant executar-se en els mateixos recursos i per tant, es crearia un overhead no desitjat. Per solucionar aquest problema, el runtime ha d'analitzar quants threads hardware hi ha disponibles pel procés de l'aplicació i crear exactament el mateix nombre d'streams.

Un cop creats tots els streams, s'ha d'analitzar de quina manera aquests rebran i executaran les tasques. El primer pas és definir un scheduler d'Argobots per a cada stream, ja que no es poden compartir entre ells. Bàsicament un scheduler és un tipus d'user-level thread que es dedica a agafar altres user-level threads per a executar-los, seguint una planificació determinada. El fet de què cada stream tingui un scheduler particular, permet la idea de què potencialment cada stream disposi de polítiques de planificació diferents. A part, cada scheduler té assignades una llista de pools, conegudes també com a cues de ready. Per defecte, els schedulers d'Argobots intenten executar els user-level threads que hi ha en la primera pool assignada, després en la segona, etc. Per tant, és important establir l'ordre de les pools correctament, ja que les pools en posicions més petites tenen més prioritat que en les més grans.

En aquest runtime es proposa disposar d'una pool global compartida per tots els streams del sistema. Aquesta pool és on s'insereixen totes les tasques al moment de crear-les i també, és d'on tots els schedulers de cada stream extreuen tasques per executar. A part d'aquesta pool compartida també es proposa l'ús d'una pool privada per a cada tasca. En aquesta pool privada hi pot inserir i extreure tasques només l'stream propietari. Per defecte, les tasques noves sempre s'insereixen en la global, per tant, a les privades mai s'hi insereix cap tasca nova. Per tant, per a què aquesta política funcioni, les tasques hauran de ser canviades de pool el primer cop que es comencin a executar.

És interessant que les pools privades siguin més prioritàries que la global, donat que és més important acabar d'executar les tasques que ja han començat que les noves. Aprofitant la prioritat de les pools als schedulers que s'ha comentat anteriorment, el simple fet d'assignar una llista on la primera pool és la privada i la segona és la global, ja s'aconsegueix aquest objectiu. El flux de les tasques que es vol aconseguir entre les pools es pot observar a la Figura 10.

Amb aquests passos el runtime i la llibreria d'Argobots estan preparats per a començar a rebre les primeres tasques per executar. Tal com s'ha explicat a l'apartat 8.3, l'única tasca que s'executa en aquest punt és la del runtime. Tot seguit, aquesta tasca crea la main per a què s'executi la funció principal de l'usuari i llavors, espera a què aquesta acabi a dins de la funció `nanos_wait_until_shut_down`.

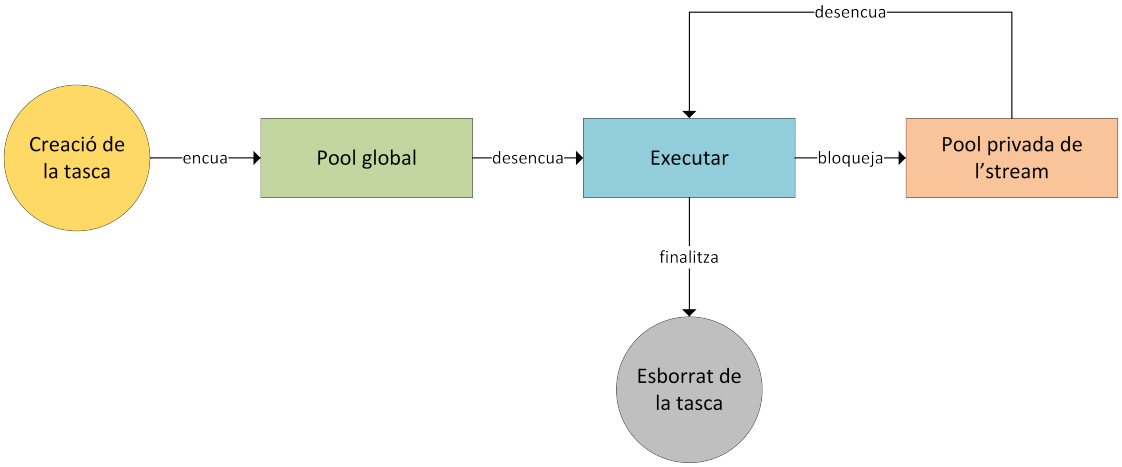

Figura 10: Flux de les tasques a les pools del runtime.

9.1.2 Creació de tasques

Les tasques han d'emmagatzemar una sèrie d'atributs per a què el runtime pugui funcionar amb normalitat. Primer de tot s'ha d'analitzar quins atributs necessita una tasca i llavors trobar un tipus de variable adequada per aquella necessitat, és a dir, si es necessita un variable el qual el rang de valors per exemple sigui de [0, 50], doncs no s'esculli un enter de 32 bits, ja que amb un de 8 bits ja n'hi ha prou. D'aquesta manera, tot i no fer una optimització a fons, es pot millorar el rendiment de l'aplicació, ja que la mida dels objectes és més petit. En aquest punt, una tasca necessita els següents atributs:

- **User-level thread:** Cada tasca ha de tindre assignat un user-level thread d'Argobots. Concretament aquest atribut és un `ABT_thread` que realment és un `typedef` de punter a void (`void *`).
- **Informació:** Cada tasca té una informació concreta, com la funció d'usuari a executar, el nom de la tasca, etc. Aquest atribut és un punter a una estructura del tipus `nanos_task_info`, declarada a l'API.
- **Arguments:** Representa el punter als arguments de la funció d'usuari els quals se li ha de passar com a paràmetre. Realment a la zona reservada de la tasca hi ha primer de tot els paràmetres de la funció i seguidament l'estructura de la tasca en si. Aquest mapeig es pot veure clarament a la Figura 11.

- **Tasca pare:** És necessari que cada tasca tingui un punter a la seva tasca pare i així poder formar un arbre N-ari on l'arrel principal sigui la tasca runtime. En aquest cas el pare no té la necessitat de saber qui és cada fill per tant, amb aquesta informació ja n'hi ha prou.
- **Comptador de fills vius:** Aquest és el comptador de tasques filles que encara no han acabat la seva execució. És útil tant en l'execució d'un taskwait com en la finalització de les tasques. Aquest atribut cal que sigui atòmic i utilitzi funcions atòmiques.
- **Indicador de què està bloquejada en un taskwait:** És el valor booleà que indica si la tasca està bloquejada en un taskwait esperant la finalització de les seves filles.
- **Atributs del taskwait:** Per a la correcta execució del taskwait és necessari una variable de condició. Donat que les variables de condició necessiten un mutex extern per a funcionar, també se n'ha afegit un. Aquests dos tipus estan disponibles a la llibreria d'Argobots, `ABT_cond` i `ABT_mutex` respectivament.

Un cop definida l'estructura de les tasques, a la funció `nanos_create_task` es reserva memòria per crear-ne una de nova i s'assignen els valors corresponents a cada camp. L'atribut tasca pare ha d'apuntar a la tasca que està executant la funció de creació, ja que aquesta és el pare de la nova tasca. També s'ha incrementar el nombre de tasques no finalitzades del pare.

Un cop creada, es crida la funció `nanos_submit_task` la qual crea un nou user-level thread d'Argobots i l'insereix a la cua global del runtime. Aquest user-level thread serà executat en un futur, quan sigui el primer de la pool i algun stream estigui disponible.

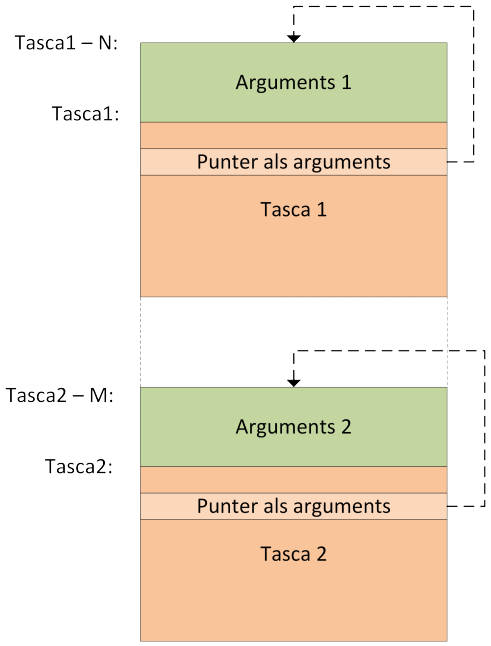

Figura 11: Mapeig de la memòria de les tasques.

9.1.3 Reconeixement de les tasques

En diferents punts del codi és necessari saber quina tasca s'està executant en aquell moment, com per exemple, en la creació de tasques per a saber qui és el pare. En aquest moment només hi ha la possibilitat de saber gràcies a Argobots quin user-level thread s'està executant, però no la tasca. Hi ha una solució poc eficient i que complica considerablement el codi del runtime. Aquesta és la de mantenir una estructura on es pugui consultar a quina tasca pertany un user-

level thread concret. Aquesta solució s'ha descarat totalment, ja que empitjoraria l'estructura del runtime i l'eficiència. Per tant, la solució més bona és saber-ho a través d'Argobots.

Al principi del projecte, Argobots no permetia cap manera de guardar un atribut a dins de cada user-level thread, per tant, es va optar per modificar el codi d'Argobots per afegir un atribut, una consultora i una modificadora. D'aquesta manera, l'user-level thread tenia un punter a la tasca a la qual pertanyia.

No obstant això, unes setmanes més tard els desenvolupadors d'Argobots van afegir una consultora als user-level threads que permetia consultar els arguments de la funció que aquest havia d'executar. La solució definitiva a aquest problema es resol aprofitant aquest canvi de la següent forma: quan es crea l'user-level thread d'una tasca es passa com a paràmetre de la funció un punter a la mateixa tasca, i d'aquesta manera a qualsevol moment de l'execució, un user-level thread pot consultar a quina tasca pertany. Aquesta solució es pot veure gràficament a la Figura 12.

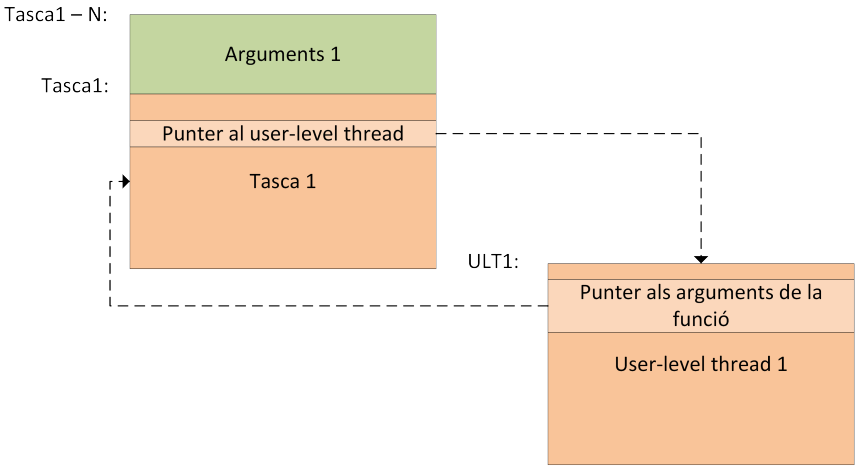

Figura 12: Mètode per guardar el punter de la tasca dins de l'`ABT_thread`.

9.1.4 Execució de les tasques

En el moment de crear l'user-level thread d'una tasca, es passa com a funció a executar una funció genèrica del runtime la qual duu a terme les següents accions:

1. Migrar l'user-level thread a la pool privada de l'stream que l'està executant, tal com s'ha explicat anteriorment.
2. Executar la funció que s'indica a la informació de la tasca amb els paràmetres de la tasca.
3. En el cas de què la tasca sigui la main ja pot acabar la funció. En qualsevol altre cas ha d'executar un taskwait implícit per esperar a què totes les tasques filles directes hagin acabat.
4. Avisar al pare de què ja ha acabat. Aquest procés s'explica detalladament a l'apartat 9.1.5.
5. Alliberar l'estructura de la tasca.

Quan les tasques finalitzen aquesta funció genèrica, Argobots allibera automàticament les estructures relacionades amb l'user-level thread.

9.1.5 Directiva taskwait

Com s'ha explicat en els apartats anteriors, la directiva taskwait es tradueix amb la crida a la funció `nanos_taskwait`. La directiva taskwait té com a objectiu que la tasca que l'executa esperi a tots els seus fills directes, per tant, s'ha de bloquejar i deixar pas a què s'executin altres tasques ready.

Per solucionar el problema del taskwait es proposa que dins la funció `nanos_taskwait` s'executi un codi similar al del pseudocodi presentat a continuació:

```
Function nanos_taskwait():
    tasca->spinlock();
    if tasca->fills_vius != 0 then
        tasca->esperant = true;
        tasca->wait();
        tasca->esperant = false;
    end
    tasca->unlock();
end
```

Algorithm 1: Pseudocodi del taskwait.

Primer de tot, la tasca adquireix el seu propi mutex i seguidament comprova el nombre de filles vives que té. En cas de que sigui zero significa que no ha tingut filles o que ja han acabat totes, per tant, pot alliberar el mutex i seguir amb l'execució. En canvi, si el valor és major que zero significa que hi ha alguna tasca filla executant-se i per tant, ha d'indicar que s'està esperant i esperar sobre la seva variable de condició. Donat que s'usa una variable de condició d'Argobots, només es bloqueja l'user-level thread i l'stream pot continuar executant altres tasques. Un cop despertada, la tasca torna a tindre el mutex adquirit, indica que ja no s'espera, allibera el mutex i continua l'execució.

Un cop dissenyat el taskwait, el següent pas és dissenyar com els fills han de notificar que han acabat l'execució. La solució més senzilla és que cada fill a l'hora de finalitzar l'execució ha de disminuir el comptador de fills vius del seu pare i en el cas de què sigui zero, despertar-lo si aquest està esperant. Per tant, el pseudocodi de la finalització de les tasques és el següent:

```
Function avisa_pare():
    tasca_pare->spinlock();
    avisar = (- -tasca_pare->filles_vives == 0 and tasca_pare->esperant);
    tasca_pare->unlock();
    if avisar then
        tasca_pare->avisar();
    end
end
```

Algorithm 2: Pseudocodi per despertar la tasca pare.

En aquest cas, primer adquireix el mutex del seu pare i seguidament calcula si ha de despertar-lo. Només l'ha de despertar en cas de què estigui esperant en un taskwait i la tasca actual sigui l'última filla. Si cal avisar el pare, es fa després d'haver alliberat el mutex per millorar l'eficiència i l'avisa a través de la variable de condició. Quan desperten la tasca pare s'insereix a la cua de ready. Tot aquest procés és per solucionar el problema del taskwait de forma eficient i usant les eines d'Argobots.

9.1.6 Finalització del runtime

Un cop la funció main ha acabat, s'executa el taskwait implícit per esperar totes les possibles filles. Llavors, aquesta crida a la funció `nanos_notify_ready_for_shutdown` que com s'ha mencionat anteriorment, és la que avisa de la finalització a la tasca runtime. En aquesta funció també s'allibera l'estructura de la tasca main, donat que ja pot donar per finalitzada la seva execució.

En aquest moment la tasca main ja ha despertat la tasca runtime per a què finalitzi. Per tant, aquesta última un cop despertada espera a que tots els streams acabin usant un `join` i seguidament, allibera totes les estructures d'Argobots com són les pools, els schedulers i els streams. En aquest punt el runtime ha finalitzat i ha alliberat tota la memòria prèviament reservada.

9.2 Sistema de dependències puntuals

El sistema de dependències permet als usuaris declarar les tasques amb dependències de dades i d'aquesta manera definir el flux de dades i de les tasques. Un dels objectius principals d'aquest projecte és la creació d'un sistema de dependències de regió contigua, com el que suporten els runtimes Nanos5 i Nanos6. Tot i que es podria passar a implementar directament aquestes dependències, el fet de què sigui un problema complex de resoldre i la falta d'experiència en aquest tema, s'ha planificat implementar primer una versió de dependències puntuals semblant a les que usa OpenMP. Tot i que el llenguatge suporta la definició de dependències de regió, només es té en compte l'adreça inicial i s'obvia la mida de la dependència. Aquesta versió requereix un disseny i implementació molt més fàcil i és un primer pas per complir l'objectiu de les dependències de regió.

Abans d'explicar la solució proposada, el millor que es pot fer és recordar com funcionen les dependències puntuals. A la Figura 13 es poden observar dos codis diferents. El de l'esquerra crea tres tasques, la primera amb una dependència de lectura sobre la variable `a`, la segona té una dependència de lectura-escriptura sobre la mateixa variable i la tercera una dependència de lectura també sobre la mateixa variable. En la part dreta es pot observar un codi també amb tres tasques, però es declaren dependències de més d'un element, per tant, de regió contigua. La primera tasca té una dependència de lectura sobre el primer i segon element del vector `b`, la segona tasca té una dependència de lectura-escriptura sobre el segon del mateix vector i l'última tasca té una dependència de lectura sobre el primer i el segon.

```c
int main () {
    int a = 0;

    #pragma omp task in(a)
    printf("Ini %d\n", a);

    #pragma omp task inout(a)
    a = 1;

    #pragma omp task in(a)
    printf("Fin %d\n", a);

    #pragma omp taskwait
}
```

```c
int main () {
    int b[2]; b[0] = 0; b[1] = 0;

    #pragma omp task in(b[0; 2])
    printf("Ini %d\n", b[0] + b[1]);

    #pragma omp task inout(b[1])
    v[1] = 1;

    #pragma omp task in(b[0; 2])
    printf("Fin %d\n", b[0] + b[1]);

    #pragma omp taskwait
}
```

Figura 13: Codis d'exemple amb dependències puntuals.

És important recalcar un possible problema que poden tindre els usuaris a l'hora d'executar un programa que defineix dependències de regió però l'executa amb un runtime que només implementa dependències puntuals. En aquest cas i en el d'OpenMP això és possible, ja que el compilador permet definir regions tot i que realment, el runtime no les implementa. Normalment aquests runtimes només suporten les dependències puntuals i per tant, l'adreça amb la qual es crea la dependència és l'adreça inicial de la regió. Aquesta tècnica permet que el runtime sigui més senzill però complica significativament la feina de l'usuari, ja que ha de procurar que les seves dependències de regió siguin perfectes, és a dir, sense solapaments parcials.

Aquest problema es pot veure al codi de la dreta de la Figura 13 i és que amb la implementació de dependències puntuals aquest codi s'executa de forma incorrecta. Simplement la primera tasca genera una dependència de lectura a l'adreça del primer element, la segona la genera pel segon element i per tant, no detecta el solapament amb la primera tasca, igual que entre la segona i la tercera tasca.

Un cop recalcats els possibles problemes, es pot començar a explicar el concepte de les dependències. El sistema de dependències puntuals es pot imaginar com un mapa que emmagatzema elements indexats per adreces de memòria i aquests contenen una llista d'accessos a aquella adreça de memòria. Un accés es crea quan una tasca declara una dependència sobre una variable. Els accessos poden ser del tipus lectura, escriptura i lectura-escriptura i també tenen les propietats d'estar ready i topmost. Que un accés sigui ready significa que té permís per executar-se i per altra banda, un accés és topmost quan no té cap accés predecessor. Aquestes propietats estan descrites a la Taula 10. Com es pot observar a la taula, un accés d'escriptura només pot estar ready si no té cap accés predecessor. En canvi, si l'accés és una lectura, pot estar ready si el seu predecessor directe és una lectura i és ready.

Tipus d'accés	**És topmost**	**No és topmost**
Lectura	Sempre	L'anterior és una lectura ready
Escriptura	Sempre	Mai
Lectura-escriptura	Sempre	Mai

Taula 10: Determina quan un accés és ready.

Implementar aquest concepte de dependències és molt senzill un cop entés el mecanisme i les estructures necessàries. Es necessita una nova estructura que representi un accés amb diversos atributs, enumerats a continuació. És important veure que no hi ha cap atribut de topmost, i és que no es necessita guardar-lo perquè es pot calcular gràcies a consultar la llista on està indexat. Només pot ser topmost en el cas de què sigui el primer accés de la seva llista.

- **Tipus d'accés:** Un accés pot ser de lectura, escriptura i lectura-escriptura.
- **Adreça de memòria:** Adreça de l'accés a memòria. Ha de ser el tipus punter a void (`void *`).
- **Ready:** És un valor booleà que indica si l'accés és ready seguint les propietats de la Taula 10.
- **Tasca propietària:** Punter a la tasca propietària de l'accés.
- **Punter al head node:** Hi ha accessos que realment no ho són, sinó que només guarden una llista d'accessos. Aquests accessos s'anomenen head node o node cap, ja que representen el cap de la llista. Aquest camp és un punter al head node que conté aquest accés. En cas de què aquest camp estigui buit significa que aquest accés és un head node.
- **Llista d'accessos:** En cas de què l'accés sigui un head node, té una llista d'accessos. Per millorar l'eficiència s'ha usat una llista intrusiva de Boost. Això significa que cada accés és un node de la llista i cada un té una referència a l'anterior i següent accés de la llista.
- **Mutex:** En cas de què l'accés sigui head node, es pot accedir a la llista dels accessos adquirint aquest mutex. S'usa un mutex d'Argobots per millorar l'eficiència a l'hora d'adquirir-lo.
- **Següent accés de la tasca:** És el punter al següent accés de la tasca propietària. D'aquesta manera es forma una llista d'accessos i la tasca només ha de guardar un punter al primer accés.

Amb aquesta nova estructura ja es poden representar els accessos i els elements que s'indexen en el mapa de les adreces. Seguidament cal definir el mapa d'adreces que pot ser un `unordered map` de C++, implementat com una taula de hash. L'ús d'una taula de hash és molt positiu en aquest cas, ja que permet fer cerques ràpides i eficients.

A part d'afegir aquestes dues estructures cal modificar i afegir alguns camps a l'estructura de la tasca. Aquestes modificacions són:

- **Mapa d'adreces:** Cada tasca té un mapa d'adreces d'accessos de les seves tasques filles. Com s'ha explicat anteriorment s'implementa amb un `unordered map` de C++.
- **Comptador d'accessos no ready:** És el comptador d'accessos de la tasca no satisfets. En cas de què tots els accessos s'hagin inserit i el comptador sigui zero la tasca pot executar-se.
- **Executant-se:** És el valor booleà que indica si la tasca s'està executant. Com s'explicarà en els següents apartats, aquesta variable ha d'usar operacions atòmiques.
- **Inserint accessos:** És el valor booleà que indica si la tasca encara està en la fase de registrar i inserir els accessos.
- **Primer accés:** És un punter al primer accés de la tasca.

Un cop totes les estructures han sigut definides cal explicar les fases de registre, inserció i alliberament dels accessos. Els següents apartats inclouen diferents gràfics per fer més entenedor aquests processos.

		Accés anterior		
		IN	**OUT**	**INOUT**
Accés nou	**IN**	IN	OUT	INOUT
	OUT	OUT	OUT	INOUT
	INOUT	INOUT	INOUT	INOUT

Taula 11: Nou tipus de l'últim accés depenent del seu tipus i del tipus del nou accés.

	És topmost	**No és topmost**
IN	És ready	És ready
OUT	És ready	No és ready
INOUT	És ready	No és ready

Taula 12: Canvis en l'estat de ready de l'últim accés en cas de que fos ready.

9.2.1 Registre i inserció de les dependències

Quan una tasca ha de registrar una dependència, el compilador crida les funcions `nanos_register_{type}_depinfo` on `type` pot ser `read`, `write` o `readwrite`. A dins d'aquestes funcions s'ha de crear un accés amb el tipus indicat, l'adreça que es passa com a paràmetre i ignorar la mida de l'accés. Un cop creat el podem inserir al lloc que pertoca. Aquest accés s'ha d'inserir en el mapa de la tasca pare. Si ja existeix una entrada per aquesta adreça, simplement es guarda una referència al head node. En canvi, si no existeix, es crea un head node i s'insereix en el mapa. Un cop es té el head node, simplement s'ha d'inserir l'accés a la llista d'accessos d'aquest.

En el moment d'inserir el nou accés poden ocórrer tres situacions, la primera és que la llista sigui buida i no hi hagi cap accés anterior, la segona és que l'últim accés de la llista sigui de la mateixa tasca que està inserint i l'última és que l'últim accés de la llista sigui d'una tasca diferent a la que pertanyen els nous accessos.

En la primera situació, en cas de què la llista sigui buida i per tant, no hi hagi cap accés anterior en aquella adreça, el nou accés s'insereix a la llista sempre com a ready i topmost, independentment del seu tipus.

En la segona situació, si l'últim accés és de la mateixa tasca que està inserint s'ha d'aplicar un filtre al nou accés. El filtratge dels accessos és el procés pel qual s'avalua el tipus dels dos accessos, tant del nou com l'últim, es fan les modificacions pertinents a l'últim accés i es descarta la inserció del nou. A la Taula 11 es mostren els canvis que s'han de dur a terme a l'últim accés depenent dels dos tipus d'accés.

A part de canviar el tipus de l'últim accés, el filtratge ha d'encarregar-se d'un altre assumpte que és el de modificar l'atribut de ready depenent del tipus d'accés actualitzat i de si és topmost. A la Taula 12 es poden observar els canvis que s'han de fer depenent de les condicions anteriors en el cas de què l'últim accés fos ready. Si l'accés no era ready no s'ha d'aplicar cap canvi. Un cop aplicats aquests canvis, la llista torna a ser coherent i per tant, el nou accés es pot eliminar i detenir la inserció.

		Accés anterior		
		IN	**OUT**	**INOUT**
Accés nou	**IN**	És ready si l'anterior també ho és	No és ready	No és ready
	OUT	No és ready	No és ready	No és ready
	INOUT	No és ready	No és ready	No és ready

Taula 13: Lògica per decidir si el nou accés és ready depenent de l'anterior.

En l'última situació, si l'últim accés és d'una altra tasca, el procés d'inserció es complica lleugerament. En aquest cas s'ha d'avaluar els tipus del nou accés i l'últim, també l'estat de ready de l'últim i prendre una decisió depenent d'aquestes condicions. Aquesta decisió es veu reflectida a la Taula 13. Per una banda, en cas de què s'insereixi un accés de lectura, aquest podrà ser ready només si l'accés anterior és de lectura i ready. Per altra banda, si l'accés és d'escriptura o lectura-escriptura, en cap cas podrà ser ready.

Un cop una tasca ha inserit tots els seus accessos o dependències, ha d'indicar que ha acabat aquest procés d'inserció posant a fals el camp d'`inserint accessos`. D'aquesta manera, si alguna tasca allibera les dependències i posa a ready l'últim accés no satisfet d'aquesta tasca, podrà posar-la a la cua de ready. Un cop indicat això, la tasca consulta si té tots els accessos satisfets i si és el cas, s'insereix al sistema d'execució tal com s'explicava en els apartats anteriors.

Per últim, perquè quedi clar aquest procés d'inserció, es mostra a continuació un exemple amb el qual s'apliquen aquestes tres situacions. El codi de l'exemple és el de la Figura 14 i l'evolució de la llista a l'adreça de la variable `a` és a la Figura 15. Primer de tot s'insereix l'accés de lectura de la primera tasca i com que la llista és buida, l'accés és ready i la tasca pot començar a executar-se. Llavors s'insereix l'accés de lectura de la segona tasca, i com que l'anterior de la llista és de lectura i ready, la nova també està ready seguint les regles de la Taula 13. Seguidament s'intenta inserir un accés d'escriptura de la mateixa variable i com que l'últim accés és de la mateixa tasca, s'ha de modificar el tipus d'aquesta i posar-la com a lectura-escriptura. A part, donat que s'ha modificat el tipus, ja no pot ser ready seguint les regles de les Taules 11 i 12, i per tant, aquesta segona tasca encara no es pot executar. Per últim, al inserir l'accés d'escriptura de la tercera tasca no es pot posar a ready seguint les regles de la Taula 13.

```
int main () {
  int a = 0;

  #pragma omp task in(a)
  printf("Ini %d\n", a);

  #pragma omp task in(a) out(a)
  a += 1;

  #pragma omp task out(a)
  a += 3;

  #pragma omp taskwait
}
```

Figura 14: Codi d'exemple d'inserció de dependències puntuals.

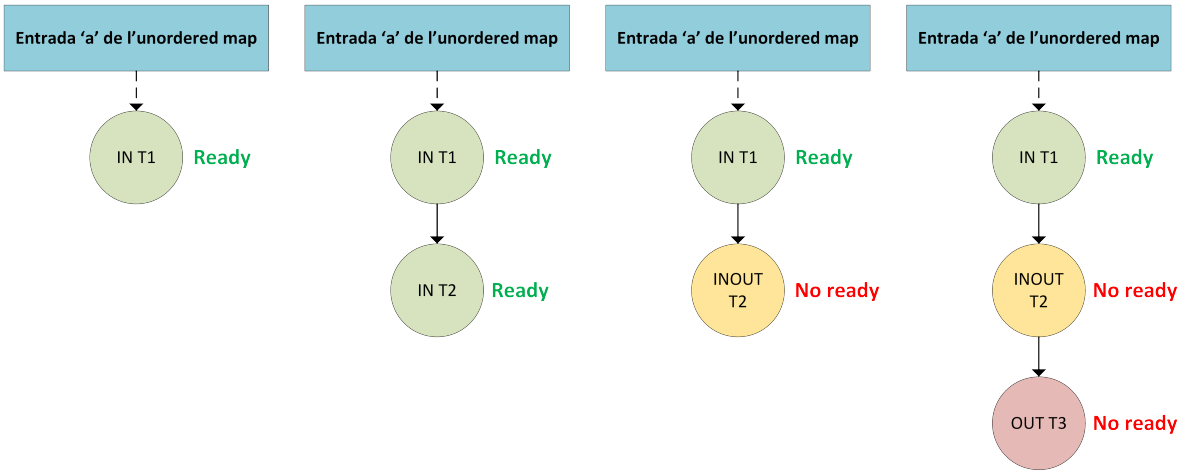

Figura 15: Evolució de la inserció de les dependències del codi d'exemple.

9.2.2 Alliberament de les dependències

El primer pas després de què una tasca finalitza l'execució de la funció d'usuari i fa el taskwait implícit, és l'alliberament dels accessos. L'alliberament és un procés complex pel qual cada accés de la tasca s'esborra i posa a ready els següents accessos corresponents.

Els accessos s'alliberen de forma seqüencial, seguint la llista d'accessos de la tasca que s'ha creat implícitament gràcies al camp `primer accés` de la tasca i `següent accés` dels accessos. Quan l'accés que s'allibera és topmost i el següent és de tipus escriptura o lectura-escriptura, només es posa a ready aquest últim. Si no es compleixen aquestes condicions i l'accés que s'allibera és d'escriptura o lectura-escriptura, es posen a ready tots els següents accessos de lectura. En altre cas, no es produeix cap acció. Un cop acabat aquest procés, s'esborra l'accés de la llista i s'allibera la seva memòria. El pseudocodi d'aquesta fase d'alliberament és:

```
Function alliberar_accés(accés):
    if not accés->últim_de_la_llista() then
        següent_accés = accés->següent;
        if accés->topmost() and not següent_access->lectura() then
            posa_a_ready(següent_access);
        else if not accés->lectura() then
            repeat
                posa_a_ready(següent_access);
                següent_accés = següent_accés->següent;
            until not següent_accés->lectura();
    end
end
```

Algorithm 3: Pseudocodi per posar a ready els accessos després d'un alliberament.

La funció `posa_a_ready` és la funció que posa a ready un accés, decreix el nombre d'accessos no satisfets de la tasca la qual pertany i si és zero, intenta executar-la. A part, també hi ha un canvi en la funció genèrica de les tasques i és que s'ha d'afegir la crida a l'alliberament dels accessos de la tasca després del taskwait implícit i abans de notificar el pare la seva finalització.

Per aclarir l'explicació de l'alliberament de les dependències a la Figura 16 es mostra un exemple que prova que l'algorisme proposat funciona correctament. L'evolució de la llista d'accessos es pot veure a la Figura 17. Es pot suposar que les tasques acaben en ordre seqüencial, és a dir, la primera acaba primer, la segona segon, etc. En el primer cas, s'allibera l'accés de la primera

tasca però no pot posar a ready a cap accés següent, només s'esborra. Quan la segona tasca acaba, allibera el seu accés de lectura el qual és topmost. Com que el següent és un accés d'escriptura el posa a ready i es pot executar la tercera tasca. Quan la tercera tasca acaba, posa a ready els accessos de lectura de la quarta i cinquena tasca.

```
int main () {
  int a = 0;

  #pragma omp task in(a)
  printf("Ini1 %d\n", a);

  #pragma omp task in(a)
  printf("Ini2 %d\n", a);

  #pragma omp task out(a)
  a = 3;

  #pragma omp task in(a)
  printf("Fin1 %d\n", a);

  #pragma omp task in(a)
  printf("Fin2 %d\n", a);

  #pragma omp taskwait
}
```

Figura 16: Codi d'exemple d'alliberament de dependències puntuals.

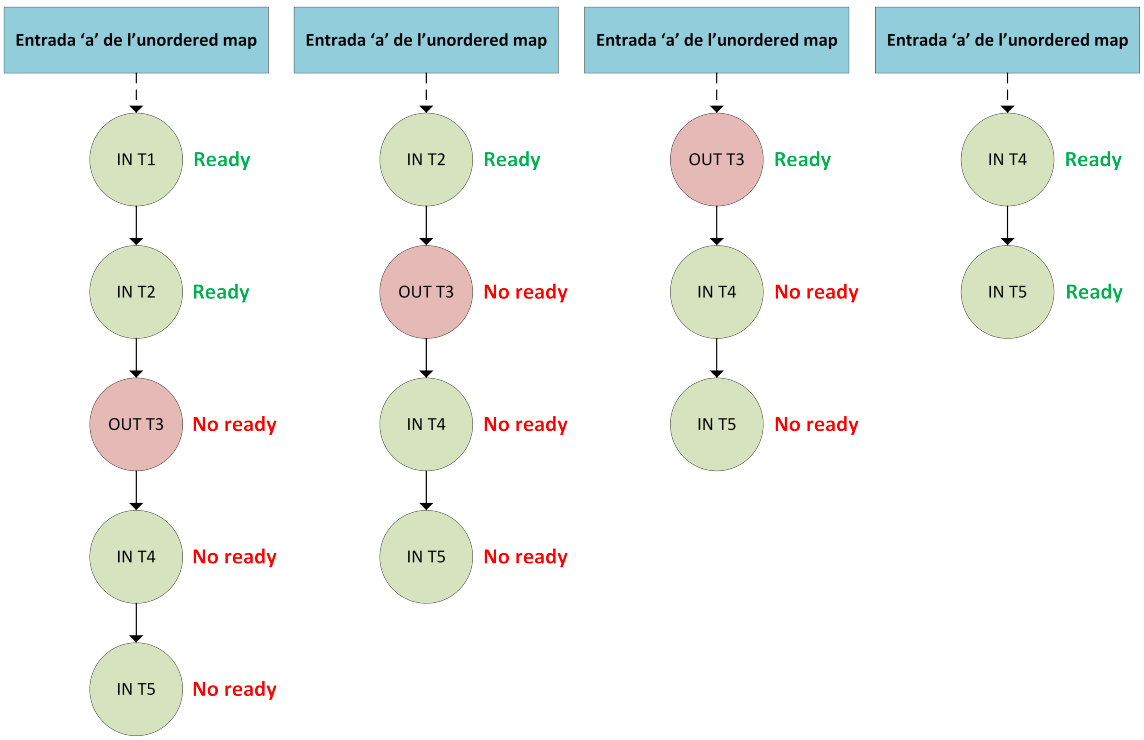

Figura 17: Evolució de l'alliberament de les dependències del codi d'exemple.

9.3 Sistema de dependències de regió contigua

En aquest apartat es descriuen els canvis que s'han efectuat al runtime per tal de suportar les dependències de regió contigua. Es parteix de la versió de dependències planes donat que l'objectiu és buscar una aproximació semblant a la d'aquest sistema. Tot i que la solució és molt més complexa, es busca tindre un disseny on els algorismes d'inserció i alliberament d'accessos siguin molt semblants. Com s'ha vist en els apartats anteriors, aquests dos algorismes són molt senzills i es vol preservar aquesta propietat al màxim.

La principal diferència amb les dependències puntuals és que en aquest cas no s'obvia la mida de l'accés. Per tant, el primer canvi que ve en ment és el de què un accés té una adreça d'inici i final, però això implica un cert nombre de problemes que s'han de solucionar. Els problemes i les propostes de solució es proposen en els següents apartats. Un dels més complexos és el de com emmagatzemar els accessos de forma que es puguin cercar eficientment.

9.3.1 Emmagatzemament dels accessos

Un dels primers problemes és el de triar una estructura per emmagatzemar els accessos. Una característica important és que hauria de ser una estructura on els elements estiguin ordenats. Aquesta propietat és molt important en el cas de regions contigües, ja que és probable que en visitar una regió, també s'hagin de visitar les regions veïnes, principalment per culpa dels solapaments entre regions. Per tant, queda totalment descartat el re-ús de l'estructura de les dependències puntuals, l'`unordered map`. Un altre factor és la importància de què les cerques siguin ràpides i de cost constant, donat que la majoria d'operacions sobre l'estructura són la cerca d'accessos donada una adreça.

Una de les estructures que compleixen aquestes dues condicions és l'arbre AVL. L'arbre AVL ideat per Andelson-Velskii i Landis l'any 1962 és un tipus d'arbre binari ordenat i on el cost de cercar és O(logN) sent N el nombre de nodes de l'arbre. La llibreria de Boost també implementa aquesta estructura de forma eficient i intrusiva. Això és un avantatge respecte a l'`unordered map`, ja que aquest no ho és i normalment una estructura intrusiva és més eficient. Que una estructura sigui intrusiva significa que els elements de l'estructura són els mateixos objectes i els quals s'hi afegeixen els punters per lligar-los entre ells.

La idea és usar un AVL per indexar els accessos amb la direcció inicial i així poder buscar fàcilment si una adreça determinada està continguda en una regió. A la Figura 18 es pot observar un arbre AVL i el que realment representa en forma de regions.

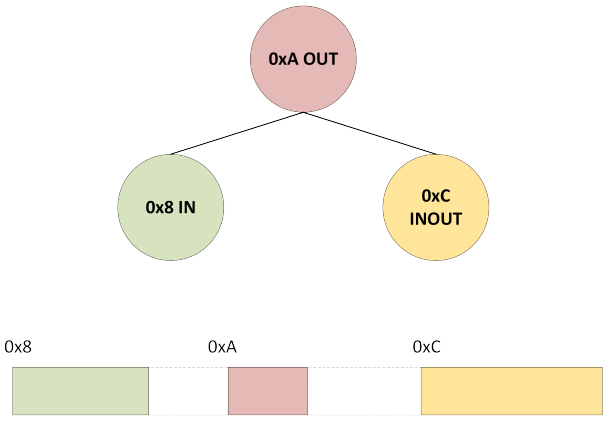

Figura 18: Exemple d'AVL i les regions que representa.

9.3.2 Solapaments parcials entre accessos

Un altre dels problemes presents són els solapaments parcials entre accessos. Tot i que normalment els algorismes estan dissenyats perquè les regions siguin perfectes, hi ha molts exemples els quals aquesta propietat no es compleix i s'han d'usar regions amb solapaments parcials. Per aquest motiu es proposa una solució que compleixi aquestes condicions en el qual es permeten solapaments de qualsevol tipus.

La idea principal és que si es detecta un solapament parcial, els accessos se separen per a formar regions perfectes. Aquesta fragmentació només s'aplica als predecessors directes, i no s'aplica recursivament. És una solució bastant complexa però efectiva, ja que serveix per a qualsevol cas i a més, això provoca que un accés només pugui tindre un successor com a màxim. L'únic problema és el de la memòria, ja que, en cas de solapament es crea un accés nou, tot i que es podria optimitzar amb la tècnica del re-ús d'estructures. Aquesta solució es pot veure en la Figura 19 on s'ensenya un exemple de regions.

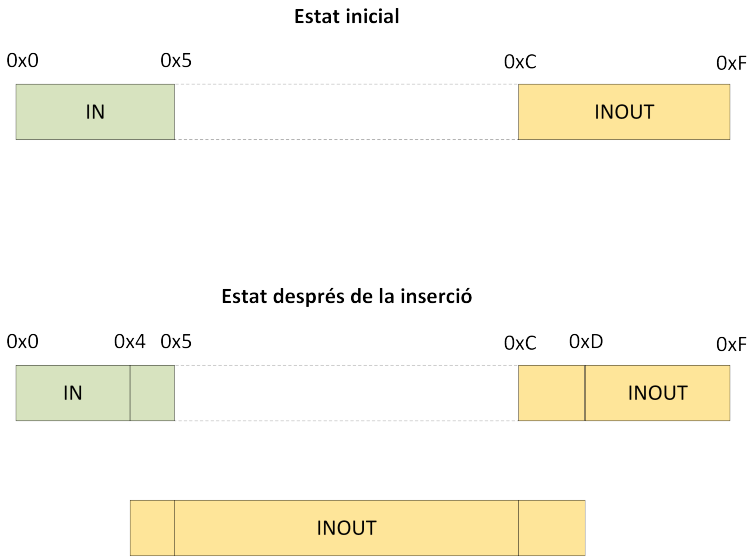

Figura 19: Solució als solapaments parcials entre accessos.

També es va pensar una solució alternativa que posteriorment es va descartar perquè provocava que l'alliberament dels accessos fos molt més difícil. Aquesta idea és totalment contrària a la de fragmentar els accessos en cas de què hi hagi solapament parcial i és que aquesta proposa que no hi hagi cap tipus de fragmentació. La decisió de no fragmentar permet que un accés tingui múltiples successors i això significa que s'hagi de crear una llista de successors per a cada accés, una opció que s'hauria d'evitar.

A més, a la Figura 20 es pot veure un exemple en el qual l'alliberament d'accessos es complica considerablement. Quan T1 allibera el seu accés, transmet la informació als accessos de T2, T3 i T4. En el cas de T4 no es pot posar a ready, ja que depèn de T3. El problema és que un cop s'arriba a un accés que no pot ser ready ha de continuar visitant els seus successors perquè aquests si que podrien posar-se ready, com el que li passa a T5. Per tant, es va decidir descartar aquesta opció que realment facilitava la inserció d'accessos però dificultava l'alliberament i tenia el problema d'haver de guardar punters a tots els successors d'un accés.

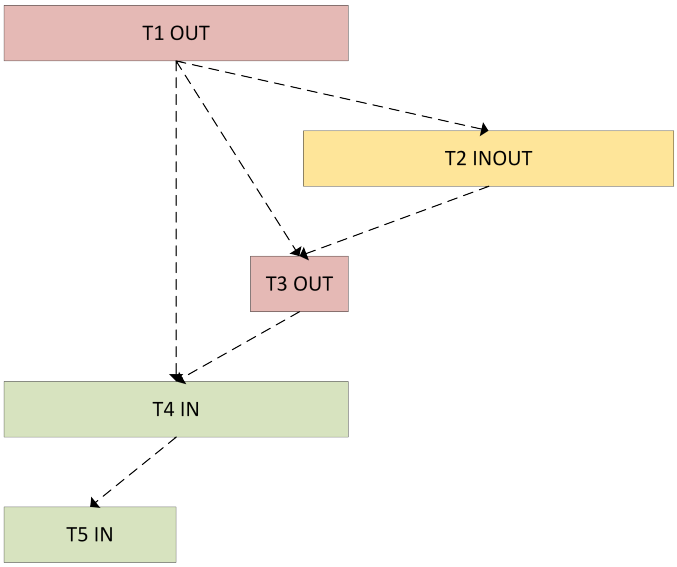

Figura 20: Problema de la solució alternativa.

9.3.3 Manteniment de l'estat de les dependències

Un altre problema és com guardar l'estat actual del mapa de dependències, és a dir, quins són els últims accessos inserits en el mapa, que és on s'hauran d'inserir els nous. La solució més fàcil és crear un altre AVL que guardi l'estat actual, és a dir, la unió de tots els accessos que no tenen successor. Aquest arbre s'anomena `bottom map` o mapa inferior.

9.3.4 Fragmentació i defragmentació

Com ja s'ha explicat anteriorment, la idea proposada provoca un cert nivell de fragmentació en el cas de què hi hagi solapaments parcials. Aquesta fragmentació provoca overhead a l'hora de crear els nous accessos fragmentats, tot i que es podria solucionar amb el re-ús de les estructures dels accessos. Una altra optimització que no s'ha pogut implementar podria ser la defragmentació dels accessos. És a dir, quan dos accessos tenen les mateixes propietats es poden fusionar en un de sol i així reduir la fragmentació.

Per poder suportar la defragmentació cal que els accessos no tinguin com a successors altres accessos, ja que aquests poden ser defragmentats i desaparèixer. La solució que es proposa és que els accessos tinguin com a successor les tasques. En cas de voler trobar els accessos successors només cal buscar en la tasca successora els accessos que hi ha a l'adreça desitjada.

A continuació es descriuen els canvis principals que s'han hagut d'aplicar i els algorismes de registre, inserció i alliberament.

9.3.5 Canvis en les estructures

Els principals canvis són en les estructures que representen els accessos i les tasques. En el cas dels accessos, hi ha diferents canvis i camps afegits respecte a la versió de dependències puntuals. Aquestes modificacions són:

- **Adreça inicial:** Adreça inicial de la regió. És un punter a void (`void *`).
- **Adreça final:** Adreça final de la regió. És un punter a void (`void *`).

- **Tasca successora:** Punter a la tasca successora. A través d'ella es pot arribar als accessos successors.
- **Topmost:** S'ha afegit nou valor booleà com a camp donat que ara no hi ha informació sobre la posició de l'accés en el mapa.
- **Acabat:** Valor booleà que indica si un accés ha acabat. En aquesta versió és necessària, ja que un accés només es pot esborrar si és topmost, en cas contrari, ha d'indicar-se com a finalitzat i quan sigui topmost es podrà esborrar.
- **Ganxo per l'AVL d'accessos de la tasca:** Ganxo per indexar l'accés a l'AVL d'accessos de la seva tasca.
- **Ganxo per l'AVL del bottom map:** Ganxo per indexar l'accés a l'AVL del bottom map de la seva tasca pare.

Els canvis a l'estructura de les tasques són els següents:

- **AVL d'accessos de la tasca:** AVL on s'indexen tots els accessos de la tasca.
- **AVL del bottom map dels fills:** AVL on s'indexen els accessos de les tasques filles que formen el bottom map.
- **Mutex dels accessos:** Mutex que protegeix els accessos tant en l'algorisme d'inserció com en el d'alliberament.

9.3.6 Registre de regions

Quan es crea un nou accés, abans d'inserir-lo en el mapa de dependències, es fa el registre i el filtratge d'aquest. L'etapa de registre tracta sobre guardar l'accés que s'ha creat a l'AVL de la tasca que està inserint l'accés. En aquest punt és quan es detecten solapaments parcials sobre accessos de la mateixa tasca i es corregeixen. En cas de què hi hagi solapament parcial, es parteixen els accessos i es guarden les parts que encara no estaven en el mapa. Cal destacar que aquest procés de registre no necessita adquirir cap mutex, ja que només està accedint als accessos la tasca que ho està executant. A la Figura 21 es pot veure com la tasca registra un accés de lectura i llavors registra un accés d'escriptura amb solapament parcial amb l'anterior. L'acció que es pren és la de separar l'accés antic per on hi ha l'extrem de l'accés nou, la part comuna assignar-li el tipus de lectura-escriptura i indexar la part restant de la nova.

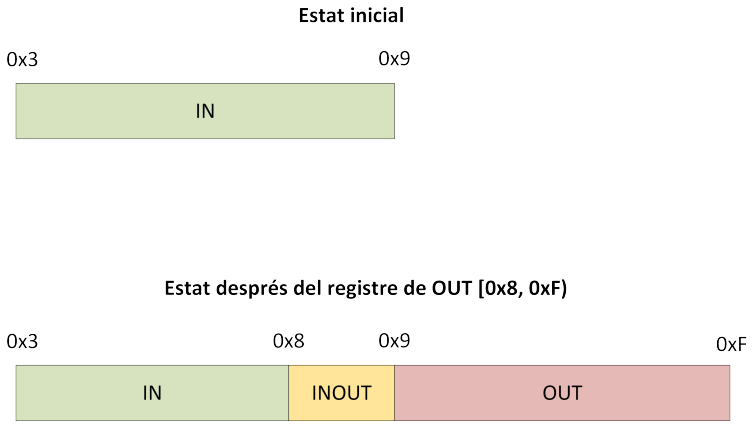

Figura 21: Exemple de registre de regions.

9.3.7 Inserció de regions

Un cop tots els accessos han sigut registrats i filtrats, poden ser inserits al mapa de dependències. Aquest procés necessita el mutex dels accessos de la tasca que està inserint, l'adquireix abans de començar i l'allibera després de processar tots els accessos.

Per cada accés de la tasca es recorre en el bottom map de la tasca pare l'interval de memòria que hi ha entre l'adreça inicial i final de l'accés i es fan les fragmentacions corresponents als accessos del bottom map i l'accés nou. Pot ser que hi hagi parts que no tenen cap accés predecessor i per tant, són ready i topmost. En canvi, si tenen un accés com a successor l'algorisme és més complex.

Si el nou accés té un predecessor, significa que hi ha almenys un accés anterior viu i per tant, aquest accés nou no pot ser topmost. Només pot ser ready en el cas de què el nou accés sigui una lectura i l'accés que hi ha al bottom map sigui de lectura i ready. Com es pot apreciar, la lògica i l'algorisme és molt semblant a la del sistema de dependències puntuals.

9.3.8 Alliberament de regions

Un cop una tasca ha acabat ja pot alliberar tots els seus accessos. Aquest procés també necessita adquirir el mutex al principi i alliberar-lo al final de processar tots els accessos.

En el cas de què l'accés no sigui topmost, cal indicar-lo com a acabat i seguir el procés d'alliberament dels altres accessos. Per tant, és possible que quan es recorre el mapa de dependències hi pot haver accessos ja acabats i per tant, que s'hauran d'esborrar.

En canvi, si l'accés és topmost és quan s'han de prendre accions. Si aquest accés és de lectura s'hauran de recórrer recursivament els successors de l'accés, esborrant els accessos que ja hagin acabat i posant a ready els primers accessos d'escriptura o lectura-escriptura. Cal recordar que per a saber els accessos successors s'ha de visitar la seva tasca i llavors buscar-los al seu AVL d'accessos. També cal destacar que tot i que s'allibera un accés pot ser que es posin ready diferents accessos del mateix nivell. El nivell s'entén com el nombre d'accessos que s'han travessat per a arribar a un accés concret. A la Figura 22 es pot veure clarament que quan s'alliberi T1 podrà posar a ready els dos accessos de T2.

En canvi, si l'accés és topmost i és d'escriptura o lectura-escriptura, en el cas de què l'accés successor sigui de lectura, s'han de posar a ready tots els accessos de lectura següents i posar com a topmost els del primer nivell. Si l'accés successor és escriptura o lectura-escriptura s'ha de posar a ready els accessos d'escriptura del primer nivell.

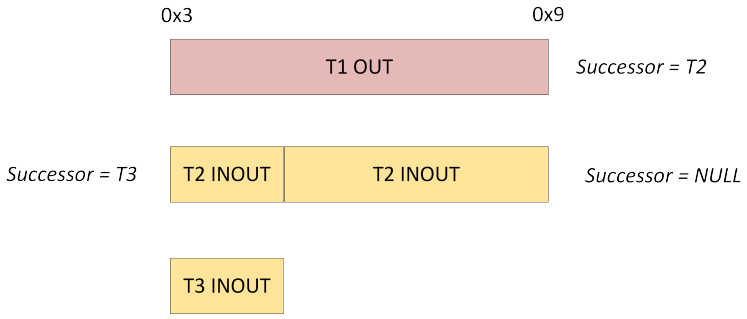

Figura 22: Diferents accessos en un mateix nivell.

9.4 Polítiques de planificació

Un dels aspectes més importants del runtime són els schedulers i les polítiques de planificació. Per defecte, el runtime desenvolupat inserta les noves tasques a la pool global. Però hi ha casos els quals això no interessa, sobretot en el cas en què una tasca i les seves filles treballen sobre un rang de dades molt similar. En aquest cas és molt interessant fer que es pugui explotar la localitat de les dades. Per tant, la política de planificació que es desenvolupa és la de què quan es crea una tasca, s'insereix a la pool privada del stream que l'està creant, i així es pot aprofitar aquesta localitat. Aquesta política es pot aplicar a totes les tasques excepte quan la tasca main està creant altres tasques filles, que aquestes són inserides a la pool global, ja que sinó no hi podrà haver cap tipus de paral·lelisme. És important afegir que l'usuari és responsable de no sobrecarregar de forma desequilibrada els streams.

Per implementar aquesta política de planificació només cal definir una variable d'entorn anomenada `$NANOS6_CUSTOM_SCHED`. En temps d'execució, si la variable no està definida, s'usa l'scheduler per defecte. En canvi, si està definida, s'usa aquest scheduler modificat. Senzillament, el runtime ha de triar a quina pool inserir els user-level threads. Cal afegir que en cas de què aquesta política de planificació modificada estigui activada, la migració a la pool privada que es fa de la tasca ja no s'ha de fer, donat que ja està assignada a la privada des del principi. El flux de les tasques per les pools amb aquesta nova política es pot veure a la Figura 23.

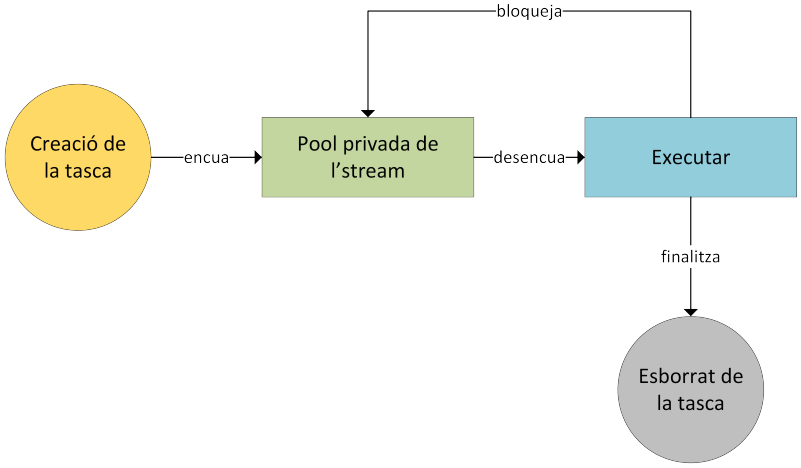

Figura 23: Flux de tasques a les pools del runtime amb la nova política de planificació.

9.5 Integració amb MPI

Un dels objectius més importants del projecte és adaptar el runtime a la llibreria d'interoperabilitat entre els models de programació paral·lela OmpSs i MPI. Aquesta necessitat és deguda al fet que moltes aplicacions actuals combinen els models de programació paral·lela a nivell de memòria compartida, com OmpSs i OpenMP, amb els de memòria distribuïda, com MPI.

La llibreria d'interoperabilitat, desenvolupada al BSC, reimplementa les funcions de comunicació bloquejants com ara MPI_Send, MPI_Recv, etc. Això s'aconsegueix gràcies a què MPI implementa aquestes funcions amb un símbol dèbil, el qual es pot reimplementar amb un símbol fort.

El pseudocodi següent és amb el es re-implementen aquestes funcions de MPI. En el cas de `MPI_Send` s'executa un `MPI_Isend` no bloquejant amb el qual es retorna una referència per poder consultar l'estat de l'enviament. Seguidament es comprova si s'ha completat immediatament

amb `MPI_Test`, i si s'ha completat, ja es pot retornar al codi d'usuari. En canvi, si encara no ha acabat, es crida al runtime amb la funció `nanos_polling_cond_wait`.

```
Function MPI_Send():
    MPI_Isend(..., &petició);
    completat = MPI_Test(petició);
    if not completat then
        nanos_polling_cond_wait(request);
    end
end
```

Algorithm 4: Pseudocodi de la re-implementació del `MPI_Send`.

En aquest punt s'ha d'adaptar la part del runtime perquè tingui una cua de condicions on hi ha tasques bloquejades i les funcions necessàries perquè tot funcioni. La funció `nanos_polling_cond_wait` simplement crea una condició a partir de la petició MPI, l'encua a la cua de condicions i seguidament la tasca fa una espera sobre la condició. Aquesta espera fa que l'user-level thread d'Argobots es bloquegi i deixi pas a l'execució de d'altres user-level threads. El pseudocodi següent mostra la implementació d'aquesta funció.

```
Function nanos_polling_cond_wait(petició):
    condició = crea_condició(petició);
    cua_condicions->afegeix(condició);
    condició->espera();
end
```

Algorithm 5: Pseudocodi de la implementació de `nanos_polling_cond_wait`.

Ara és necessari fer que aquesta cua es visiti periòdicament per comprovar si alguna petició MPI ha acabat. El millor lloc és a la funció que executen els schedulers. Després d'executar un user-level thread, l'scheduler comprova la cua. Aquest codi es pot veure tot seguit.

```
Function executa_scheduler():
    while 1 do
        tasca = agafa_tasca_ready();
        if tasca then
            executa_tasca(tasca);
        end
        comprova_cua_condicions();
    end
end
```

Algorithm 6: Pseudocodi de la funció de l'scheduler.

Per últim, la funció que comprova la cua de condicions la recorre per trobar alguna petició MPI que hagi estat completada. En cas de què una estigui completada, s'envia una notificació per despertar la tasca que s'havia bloquejat per a aquesta execució, s'afegeix a la cua de ready i s'atura la cerca. Quan es comprova una condició s'està executant un `MPI_Test` de la petició

guardada. El següent pseudocodi mostra aquest procés.

```
Function comprova_cua_condicions():
    iterador = cua_condicions->begin();
    while iterador != cua_condicions->end() do
        condició = *iterador;
        completat = comprova_condició(condició);
        if completat then
            condició->notifica();
            break;
        end
        ++iterador;
    end
end
```

Algorithm 7: Pseudocodi de la funció que comprova les condicions.

Amb aquests canvis senzills en el runtime es pot arribar a guanyar molt rendiment i eficiència, ja que realment estem solapant les fases de comunicació amb les fases de computació. A part, també hi ha diversos algorismes que sense aquesta interoperabilitat poden provocar un deadlock i l'usuari ha de tenir-ho en compte per a què no passi. En aquest cas el risc de deadlock baixa, ja que quan es bloqueja una tasca no es bloqueja l'stream, i aquest continua executant altres tasques, cosa que no es permet sense la interoperabilitat, ja que bloqueja tot el thread.

9.6 Clàusula commutative

La clàusula commutative d'OmpSs representa un tipus de dependència entre dades semblant al de lectura-escriptura però amb una característica especial. Totes les tasques definides de forma consecutiva que declaren una dependència commutative sobre una variable poden executar-se en qualsevol ordre però mai al mateix temps. Aquesta clàusula és molt útil a l'hora de programar codis que usen recursos que no es poden compartir però realment no importa l'ordre amb el qual l'executen, reduccions, etc.

A la Figura 24 es pot veure que les dues tasques declaren una dependència del tipus commutative sobre la variable `a`, la tercera una d'escriptura sobre la `a` i la quarta i la cinquena una commutative sobre la mateixa variable `a`. En aquest cas, el runtime pot executar T1 i T2 en qualsevol ordre però no al mateix temps. Un cop les dues han acabat, T3 pot començar a executar-se. Un cop aquesta ha acabat pot despertar les tasques T4 i T5, les quals no es poden executar alhora però si en qualsevol ordre.

En aquest projecte només es dissenya i implementa una versió limitada de la clàusula commutative. Les limitacions són:

- Una tasca no pot tindre la clàusula commutative sobre dues o més variables.
- Les regions amb commutative no es poden solapar parcialment amb altres accessos.

Aquestes limitacions no tenen cap repercussió en cap cas, ja que l'única aplicació de l'avaluació del resultat que usa commutative no fa servir les característiques no implementades d'aquesta clàusula.

```
int main () {
  int a = 0;

  #pragma omp task commutative(a) label(T1)
  a += 1;

  #pragma omp task commutative(a) label(T2)
  a += 2;

  #pragma omp task out(a) label(T3)
  a = 3;

  #pragma omp task commutative(a) label(T4)
  a += 1;

  #pragma omp task commutative(a) label(T5)
  a += 2;

  #pragma omp taskwait
}
```

Figura 24: Codi d'exemple del commutative d'OmpSs.

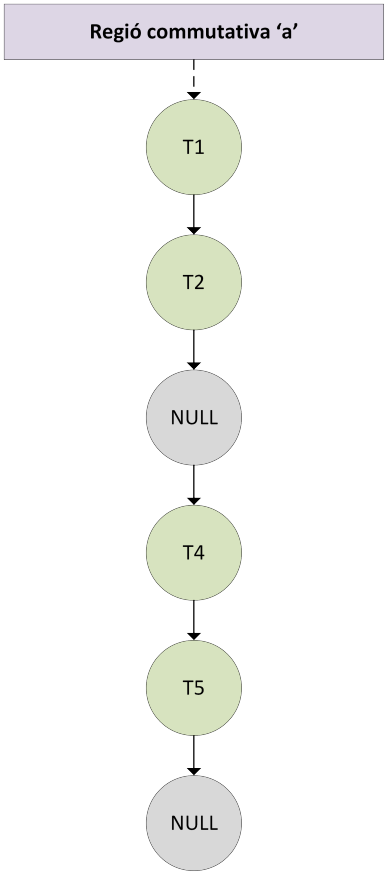

Figura 25: Llista de tasques del commutative amb el codi d'exemple.

La idea és tindre un AVL de regions commutatives on cada regió tingui una llista on es van inserint les tasques amb regions commutatives. El disseny proposat té en compte de separar a dins de cada llista els grups de tasques commutativa consecutives. Es proposa que entre grups d'accessos consecutius de commutative hi hagi marques en la llista. Usant l'exemple de la Figura 24, la llista que resulta es pot veure a la Figura 25. Senzillament hi ha dos grups de commutatives i llavors cada un es guarda a la llista però afegint una marca entre els dos.

Un cop una tasca amb commutative ha acabat, ha de fer l'alliberament dels accessos normalment i posteriorment, recorre la llista de la regió commutativa que pertany fins a trobar una tasca que pugui executar-se. Si ja no hi ha tasques i es troba la marca de separació, senzillament ha d'esborrar-la perquè es pugui reutilitzar l'estructura en altres moments de l'execució.

10 Canvis a la llibreria d'Argobots

La llibreria d'Argobots compta amb un repositori públic de GIT que permet als usuaris descarregar, configurar, compilar i instal·lar Argobots. Tot i que és un repositori públic, només els desenvolupadors tenen dret a escriure en el repositori remot. El que si permet és canviar de forma fàcil el codi i poder afegir commits propis i així poder tenir versió actualitzada i una mica personalitzada.

Durant aquest projecte s'han fet petites modificacions i s'han afegit algunes funcions per facilitar l'ús d'Argobots i per solucionar petits bugs. S'han fet petites modificacions en el codi dels user-level threads, eventuals i variables de condició. També s'han reportat alguns bugs als desenvolupadors d'Argobots que posteriorment han sigut arreglats ràpidament.

```
#pragma omp task inout(A[i*n+j]) label(potrf)
void omp_potrf(int ts, int n, double* A, int i, int j);

#pragma omp task in(A[ai*n+aj]) inout(B[bi*n+bj]) label(trsm)
void omp_trsm(int ts, int n, double *A, int ai, int aj, double *B, int bi, int bj);

#pragma omp task in(A[ai*n+aj]) inout(B[bi*n+bj]) label(syrk)
void omp_syrk(int ts, int n, double *A, int ai, int aj, double *B, int bi, int bj);

#pragma omp task in(A[ai*n+aj]) inout(B[bi*n+bj]) inout(C[ci*n+cj]) label(gemm)
void omp_gemm(int ts, int n, double *A, int ai, int aj, double *B, int bi, int bj,
    double *C, int ci, int cj);

void cholesky_linear(unsigned ts, unsigned n, double *A)
{
    for( unsigned k = 0; k < n; k += ts ) {
        // Diagonal Block factorization
        omp_potrf(ts, n, A, k, k);

        // Triangular systems
        for( unsigned i = k + ts; i < n; i += ts )
            omp_trsm (ts, n, A, k, k, A, k, i);

        // Update trailing matrix
        for( unsigned i = k + ts; i < n; i += ts ) {
            for( unsigned j = k + ts; j < i; j += ts )
                omp_gemm (ts, n, A, k, i, A, k, j, A, j, i);
            omp_syrk (ts, n, A, k, i, A, i, i);
        }
    }
    #pragma omp taskwait
}
```

Figura 26: Codi del kernel del Cholesky.

11 Avalaució del resultat final

En aquest apartat s'expliquen les aplicacions que s'han executat per comprovar que el runtime funciona correctament i també per mesurar el rendiment d'aquestes i fer comparatives. Les aplicacions són la del kernel de Cholesky, Gauss-Seidel, Nbody, IFS i els tests oficials que comproven totes les funcionalitats.

11.1 Cholesky

La descomposició de Cholesky de André-Lous Cholesky aprofita que una matriu simètrica definida positiva pot ser descomposta com el producte d'una matriu triangular inferior i la transposada de la matriu triangular inferior. Aquest algorisme permet resoldre diversos sistemes d'equacions lineals.

Aquesta aplicació utilitza els algorismes lineals: `potrf`, `trsm`, `syrk` i `gemm` de la llibreria d'Intel d'algorismes matemàtics KML. Una part del codi es pot veure a la Figura 26.

Per aquest problema s'ha fet una anàlisi d'strong scaling i d'speedup respecte a la versió seqüencial usant el runtime desenvolupat en aquest projecte. Aquests gràfics es poden veure a la Figura 27 i 28.

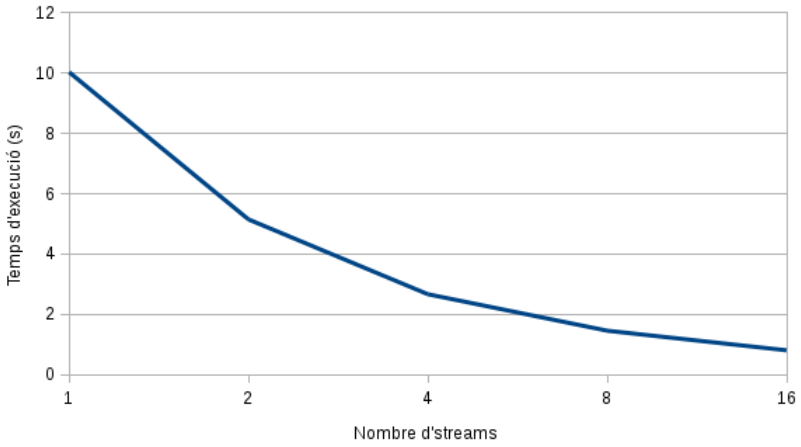

Figura 27: Strong scaling del Cholesky.

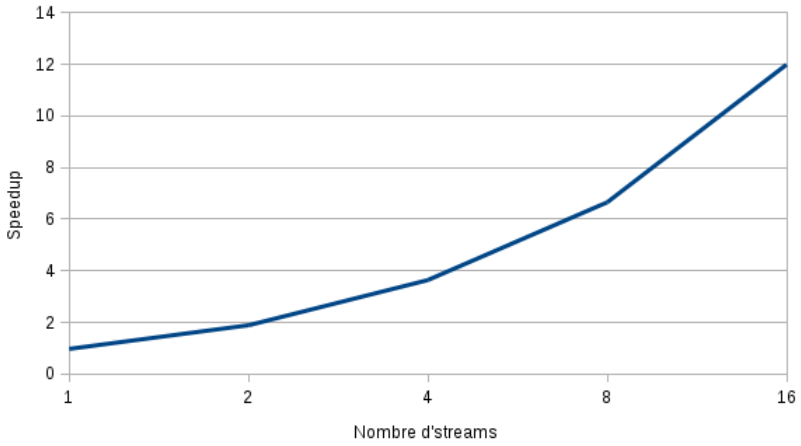

Figura 28: Speedup del Cholesky respecte el seqüencial.

11.2 Gauss-Seidel

El mètode Gauss-Seidel és un mètode iteratiu utilitzat per resoldre sistemes d'equacions lineals similar al mètode de Jacobi. Tot i que pot ser aplicat a qualsevol matriu amb els elements no nuls a la seva diagonal, la convergència només es garanteix si la matriu és diagonalment dominant o simètrica i definida positiva.

El codi principal del Gauss-Seidel es pot veure a la Figura 31. Per aquesta aplicació també s'han extret gràfics de strong scaling i speedup respecte a la versió seqüencial a la Figura 29 i 30.

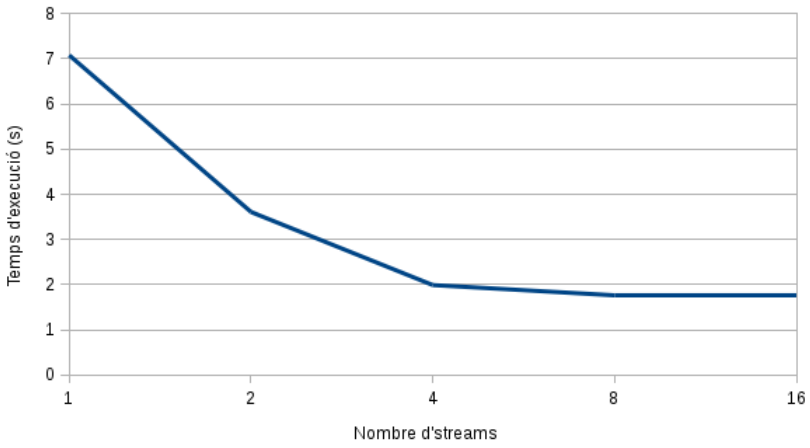

Figura 29: Strong scaling del Gauss Seidel.

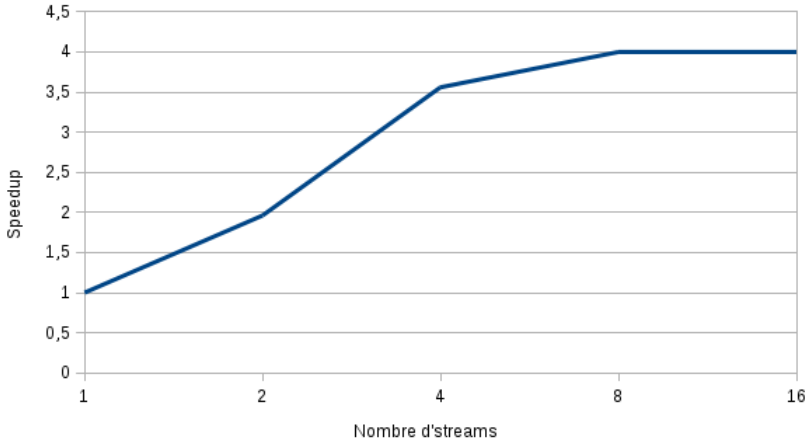

Figura 30: Speedup del Gauss Seidel respecte el seqüencial.

```
void performIteration(double *A, long N, long realN, long TRows) {
    for (long i = 1; i <= N; i += TRows) {
        long rows = min(N+1-i, TRows);
        #pragma omp task in(A[(i-1)*realN;realN], A[(i+rows)*realN;realN])
            inout(A[i*realN;rows*realN])
        for (int ii=0; ii < rows; ii++) {
            for (int jj=1; jj <= N; jj++) {
                A[(i+ii)*realN + jj] = 0.2 * (
                    A[(i+ii)*realN + jj]
                    + A[(i+ii-1)*realN + jj]
                    + A[(i+ii+1)*realN + jj]
                    + A[(i+ii)*realN + jj-1]
                    + A[(i+ii)*realN + jj+1]
                );
            }
        }
    }
}

void main() {
    // ...
    for (int iteration = 0; iteration < IT; iteration++) {
        performIteration(A, N, N+2, TRows);
    }
    #pragma omp taskwait
    // ...
}
```

Figura 31: Codi del kernel del Gauss-Seidel.

11.3 Nbody

L'Nbody és una simulació d'un sistema dinàmic de partícules que normalment està influenciat per forces físiques com la gravetat. Les simulacions de l'Nbody són molt usades en el camp de l'astrofísica.

En aquest cas s'ha utilitzat el problema amb complexitat quadràtica. El problema es divideix en blocs i es calcula cada bloc depenent de tots els altres blocs i llavors hi ha una fase de comunicació. Es crea una tasca OmpSs per cada cop que s'actualitza un bloc a partir de dos blocs, per tant, a cada iteració es creen N^2 tasques de computació. Aquestes tasques es creen amb dependències amb els dos blocs com a entrada i amb una commutative al bloc de sortida. També es creen tasques per enviar els blocs amb MPI. Gràcies a aquestes dependències i a la granularitat fina de les tasques, es pot solapar una part de la computació dels blocs amb la comunicació i això ens ho permet fer la llibreria d'interoperabilitat amb MPI.

En aquest cas s'aplica un retard a les tasques de computació gràcies a dues distribucions aleatòries, la primera és una distribució gamma i una distribució uniforme depenent de la iteració. Aquest retard s'aplica perquè es generi un desbalanceig de càrrega i així poder observar l'efecte d'usar la llibreria d'interoperabilitat amb el runtime d'aquest projecte.

Argobots també disposa d'una llibreria d'interoperabilitat integrada a un una llibreria MPI anomenada MPICH-Argobots. Aquesta interoperabilitat és directament amb Argobots i el runtime del projecte ni s'adona. En canvi, la llibreria d'interoperabilitat del BSC interactua directament amb el runtime.

A les Figures 32, 33 i 34 es fa una comparativa dels speedups de les dues interoperabilitats respecte a l'execució d'OpenMP + MPI sense cap interoperabilitat. La llibreria del BSC és la línia verda i la d'Argobots és la línia vermella. Com es pot veure, les dues interoperabilitats són molt semblants i quan s'executa l'aplicació amb més de 2 nodes es comença a notar la millora respecte a la versió d'OpenMP.

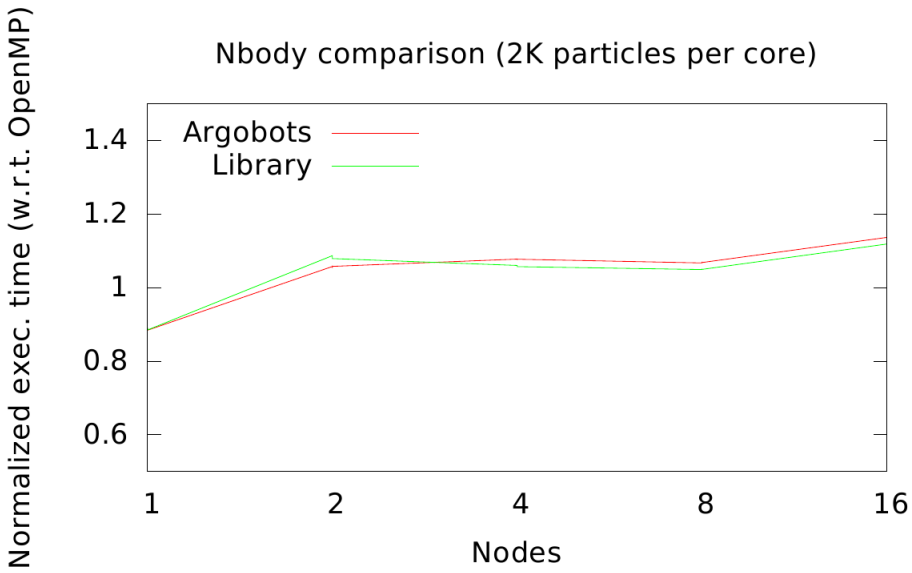

Figura 32: Resultats d'executar els tests de Nanos6 a Marenostrum.

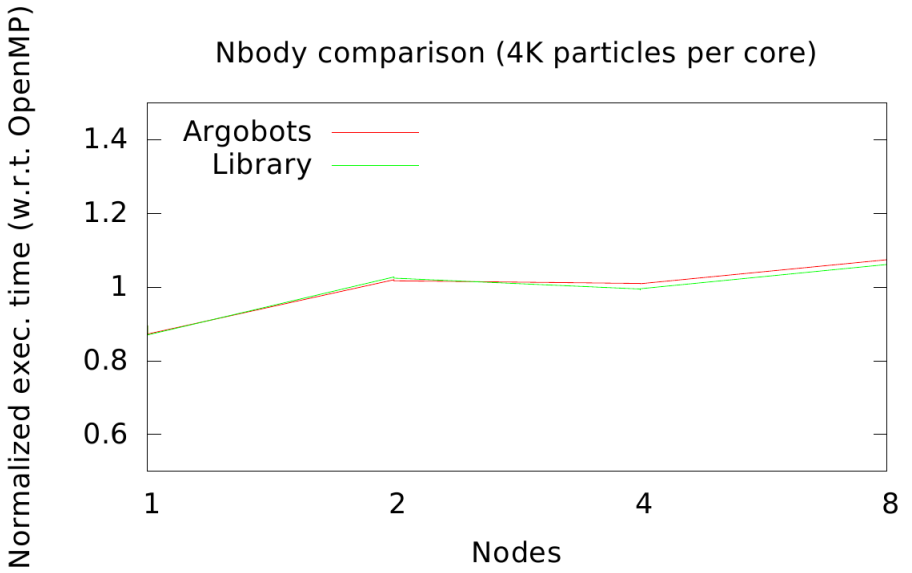

Figura 33: Resultats d'executar els tests de Nanos6 a Marenostrum.

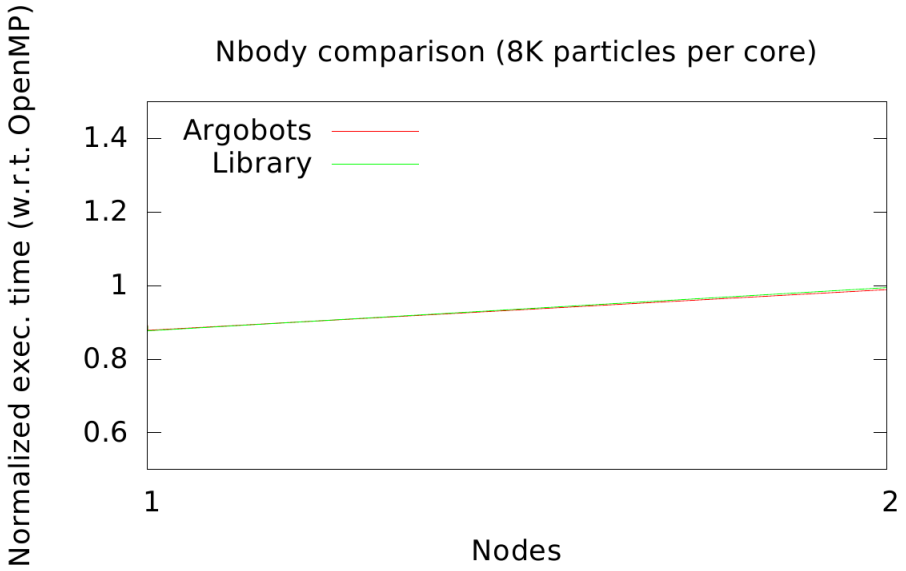

Figura 34: Resultats d'executar els tests de Nanos6 a Marenostrum.

11.4 Integrated Forecast System

L'IFS és un model de predicció meteorològica operacional global desenvolupat i mantingut per l'European Centre for Medium-Range Weather Forecasts (ECMWF). L'IFS és un model que s'executa cada dotze hores i el seu horitzó de pronòstic és de fins a deu dies amb sortides cada 6 hores i amb una resolució de 16 km i 137 nivells verticals. L'IFS és també la base d'un sistema de predicció per conjunts a menys resolució espacial, concretament de 32 km i 91 nivells verticals. Aquest model és conegut internacionalment com el millor model global de predicció meteorològica.

Aquesta aplicació pot usar un model de programació paral·lela a nivell de memòria compartida com és OmpSs i de memòria distribuïda com és MPI. En aquesta aplicació també es podria executar juntament amb la llibreria d'interoperabilitat. La part principal de l'aplicació està programada amb Fortran. El cas és que no s'ha pogut executar donat que hi ha un problema amb el compilador Mercurium per a suportar Fortran amb el runtime de Nanos6.

11.5 Tests de Nanos6

Com ja s'havia explicat anteriorment, Nanos6 disposa d'una col·lecció de tests que comproven cada una de les funcionalitats del runtime amb tots els casos possibles. Es comprova l'inicialització del runtime, la creació de tasques i l'aniuament d'aquestes amb l'algorisme de Fibonacci, les dependències puntuals i les dependències de regió. A la Figura 35 es pot veure que els 1108 tests de la col·lecció s'executen correctament amb 16 streams al supercomputador Marenostrum.

```
PASS: lr-nonest-upgrades.test 4 Evaluating that T2 does not start before T0 finishes
PASS: lr-nonest-upgrades.test 5 Evaluating that when T1 starts T0 has finished
PASS: lr-nonest-upgrades.test 6 Evaluating that when T2 starts T0 has finished
PASS: lr-nonest-upgrades.test 7 Evaluating that T2 does not start before T0 finishes
PASS: lr-nonest-upgrades.test 8 Evaluating that when T2 starts T0 has finished
PASS: lr-nonest-upgrades.test 9 Evaluating that T1 does not start before T0 finishes
PASS: lr-nonest-upgrades.test 10 Evaluating that when T1 starts T0 has finished
PASS: lr-nonest-upgrades.test 11 Evaluating that T1 does not start before T0 finishes
PASS: lr-nonest-upgrades.test 12 Evaluating that when T1 starts T0 has finished
PASS: lr-nonest-upgrades.test 13 Evaluating that T3 does not start before T1 finishes
PASS: lr-nonest-upgrades.test 14 Evaluating that when T3 starts T1 has finished
PASS: lr-nonest-upgrades.test 15 Evaluating that T1 does not start before T0 finishes
PASS: lr-nonest-upgrades.test 16 Evaluating that T2 does not start before T0 finishes
PASS: lr-nonest-upgrades.test 17 Evaluating that when T1 starts T0 has finished
PASS: lr-nonest-upgrades.test 18 Evaluating that when T2 starts T0 has finished
==========================================================================
Testsuite summary for nanos6-argobots 0.1
==========================================================================
# TOTAL: 1108
# PASS:  1108
# SKIP:  0
# XFAIL: 0
# FAIL:  0
# XPASS: 0
# ERROR: 0
==========================================================================
```

Figura 35: Resultats d'executar els tests de Nanos6 a Marenostrum.

12 Conclusions

S'han aconseguit tots els objectius, ja que s'ha dissenyat i implementat una estructura del runtime sòlida i eficient, un sistema de dependències amb un bon disseny a darrere, tot i que optimitzable, una adaptació correcta a l'interoperabilitat amb MPI, i la clàusula commutative, una tasca addicional afegida l'últim mes del treball de final de grau.

Els resultats han sigut correctes, ja que funciona per tots els tests i les aplicacions. Un rendiment correcte però que es podria millorar molt optimitzant una sèrie de parts del codi, com ara la memòria.

Personalment he après molt sobre els models de programació paral·lela d'OmpSs i MPI, sobre com s'implementa un runtime, també una mica sobre els compiladors, sobre el sistema de control de versions GIT, la llibreria d'Argobots i els llenguatges de C i C++.

13 Treball futur

Tot i que s'ha aconseguit acabar el projecte i complir amb tots els objectius encara queda molta feina a fer amb aquest runtime.

Primer de tot es podria implementar una versió completa del commutative d'OmpSs seguint el disseny que s'ha fet fins ara.

Un dels temes més importants a millorar és l'eficiència. Primer de tot caldria aplicar optimitzacions de memòria, per exemple, reutilitzant les estructures de dades com les tasques i els accessos. Altres podria ser l'optimització del codi dels algorismes d'inserció i alliberament dels accessos sobretot amb les dependències de regió.

Un cop optimitzat el runtime es podria fer una avaluació del rendiment més profunda comparant-lo amb altres runtimes d'OmpSs i OpenMP. També es podria buscar més aplicacions amb MPI per poder avaluar el rendiment de la interoperabilitat.

14 Revisió del projecte

En aquest apartat es revisa la planificació i les tasques, les hores dedicades definitives, els recursos i costos, el diagrama de Gannt definitiu i per últim, la revisió de la metodologia.

14.1 Modificacions de les tasques

Durant l'execució del projecte hi ha hagut canvis en les tasques i es presenten a continuació:

- **Clàusula commutative:** Com s'ha explicat a la planificació, s'ha afegit aquesta nova tasca, ja que l'aplicació de l'Nbody necessita aquesta clàusula. Per tant, en els següents apartats es té en compte aquesta nova tasca.

14.2 Revisió de la definició de les dependències

La taula definitiva de dependències entre tasques es pot observar a la Taula 14.

Tasca	**Tasca predecessora**
Gestió del projecte	-
Familiarització amb els models	Gestió del projecte
Anàlisi del projecte	Familiarització amb els models
Configuració de l'entorn	Anàlisi del projecte
Disseny i desenvolupament	Configuració de l'entorn
Estructura del runtime	Configuració de l'entorn
Sistema de dependències	Estructura del runtime
Polítiques de planificació	Estructura del runtime
Integració amb MPI	Polítiques de planificació
Clàusula commutative	Integració amb MPI
Avaluació del resultat final	Disseny i desenvolupament
Etapa final	Avaluació del resultat final

Taula 14: Dependències entre tasques del projecte.

14.3 Revisió de les hores dedicades

A la Taula 15 es poden observar les hores dedicades a cada tasca del projecte i els seus rols. S'ha reduït la dedicació temporal a la tasca de Polítiques de planificació, ja que aquesta tasca és relativament ràpida de dur a terme. En canvi, s'ha augmentat el temps de la tasca d'Integració amb MPI, ja que s'ha arribat a la conclusió que aquesta tasca necessita més dedicació que l'establerta inicialment. També s'ha afegit la tasca de la Clàusula commutative. En total, les hores d'hores dedicades ascendeixen a 575 hores, 15 hores més que en la planificació inicial.

Tasca	**Responsable**	**Dedicació (hores)**
Gestió del projecte	Cap de projecte	75
Familiarització amb els models	Programador	25
Anàlisi del projecte	50% Analista / 50% Dissenyador	100
Configuració de l'entorn	Programador	10
Estructura del runtime		80
Disseny	Dissenyador	*40*
Implementació i test	40% Programador / 60% Tester	*40*
Sistema de dependències		120
Disseny	Dissenyador	*48*
Implementació i test	40% Programador / 60% Tester	*72*
Polítiques de planificació		5
Disseny	Dissenyador	*2*
Implementació i test	40% Programador / 60% Tester	*3*
Integració amb MPI		70
Disseny	Dissenyador	*20*
Implementació i test	40% Programador / 60% Tester	*50*
Clàusula commutative		20
Disseny	Dissenyador	*5*
Implementació i test	40% Programador / 60% Tester	*15*
Avaluació del resultat final	40% Programador / 60% Tester	10
Etapa final	Cap de projecte	60
Total		**575**

Taula 15: Temps dedicat a cada tasca del projecte i els seus rols.

14.4 Revisió dels recursos i costos

Donat que només s'ha afegit la tasca de disseny i desenvolupament de la clàusula commutative i aquesta només necessita un subconjunt dels recursos descrits a la fita inicial, no és necessari afegir-ne ni extreure'n cap.

L'augment de 15 hores dedicades fa que el cost dels recursos de la fase de disseny i desenvolupament augmenti en 0.3 €/hora * 15 hores = 4.5 €. Cal afegir que aquest canvi no ha provocat cap augment d'hores dedicades del director i co-director. Per tant, aquest petit sobrecost es cobreix de sobres amb la contingència del 10% que es va establir a la planificació inicial i no és necessari cap canvi en el pressupost.

14.5 Revisió del diagrama de Gantt

En aquesta secció es presenta el diagrama de Gantt definitiu del projecte. A la Figura 36 hi ha la taula detallada del Gantt amb la duració, inici i final, els recursos usats, els rols i el risc de cada tasca. A la Figura 37 es pot observar el gràfic del diagrama de Gantt.

Com es pot observar en el diagrama, el projecte s'ha acabat el divendres 17 de juny de 2016, tres dies abans de l'entrega de la memòria del treball de fi de grau. Per tant, s'ha complert amb tots els requisits i terminis d'entrega.

14.6 Revisió de la metodologia

A la fita inicial es van proposar com a mètodes de treball, una combinació de les metodologies àgils d'*eXtreme Programming (XP)* i *SCRUM*, que permeten l'aplicació de diferents conceptes. Per una banda, l'estratègia de desenvolupament incremental ha permès executar tests i aplicacions sense haver d'esperar a la finalització del projecte. Per altra banda, el desenvolupament en cicles curts ha permès el control setmanal dels objectius i d'aquesta manera, cenyir-se a la planificació i evitar retards. Per últim, amb el desenvolupament guiat per proves s'ha aconseguit desenvolupar un codi net, estructurat i sobretot, que funcioni correctament i compleixi tots els requisits establerts.

L'eina de seguiment *GIT*, també proposada a la fita inicial, ha sigut molt útil a l'hora de controlar i organitzar clarament les versions del software desenvolupat. L'altra eina de seguiment proposada va ser la de *OneNote* de *Microsoft*, que ha permès controlar l'avanç setmanal del projecte.

A més, els mètodes de validació que es van proposar van ser els d'executar tests propis en llenguatge C/C++ molt específics per a cada funcionalitat i els tests que s'usen per a verificar el runtime de Nanos6. Aquests dos mètodes de validació han permès desenvolupar un software de qualitat i sense errors.

En conclusió, tota la metodologia proposada a la fita inicial ha sigut molt útil i ha proporcionat tots els beneficis exposats anteriorment, i per tant, no ha sigut necessari fer cap canvi en els mètodes.

	Nombre de tarea	Comienzo	Fin	Nombres de los recursos	Responsabilidad	Riesgo
1	**Gestió del projecte**	**lun 22/02/16**	**vie 01/04/16**	**Eines d'ofimàtica;Gantter;PC Acer;PC Dell;Samsung Galaxy S5;Windows 7**	**Cap de projecte**	**Baix**
2	Contextualització i abast	lun 22/02/16	mié 02/03/16		Cap de projecte	Baix
3	Planificació temporal	jue 03/03/16	mar 08/03/16		Cap de projecte	Baix
4	Gestió econòmica i sostenibilitat	mié 09/03/16	mar 15/03/16		Cap de projecte	Baix
5	Presentació preliminar	mié 16/03/16	dom 20/03/16		Cap de projecte	Baix
6	Mòdul d'Enginyeria Computadors	lun 21/03/16	vie 01/04/16		Cap de projecte	Baix
7	Presentació oral i document final	lun 21/03/16	vie 01/04/16		Cap de projecte	Baix
8	Milestone: Entrega document GEP	vie 01/04/16	vie 01/04/16			
9	**Familiarització amb els models**	**sáb 02/04/16**	**mar 05/04/16**	**Editors de text;Eines de debug i profiling;Marenostrum;Models de prog;OpenSUSE;PC Acer;PC Dell**	**Programador**	**Baix**
10	**Anàlisi del projecte**	**mié 06/04/16**	**mar 19/04/16**	**Eines d'ofimàtica;OpenSUSE;PC Acer;PC Dell**	**50% Analista / 50% Dissenyador**	**Baix**
11	Milestone: Presentació oral GEP	mar 19/04/16	mar 19/04/16			
12	**Configuració de l'entorn**	**mié 20/04/16**	**jue 21/04/16**	**Editors de text;GIT;Marenostrum;Models de prog;OpenSUSE;PC Acer;PC Dell**	**Programador**	**Baix**
13	**Disseny i desenvolupament**	**vie 22/04/16**	**mar 07/06/16**	**Editors de text;Eines de debug i profiling;GIT;Marenostrum;Models de prog;OpenSUSE;PC Acer;PC Dell**		
14	**Estructura del runtime**	**vie 22/04/16**	**lun 02/05/16**			**Mig**
15	Disseny	vie 22/04/16	mar 26/04/16		Dissenyador	Mig
16	Implementació i test	mié 27/04/16	lun 02/05/16		40% Programador / 60% Tester	Mig
17	**Sistema de dependències**	**mar 03/05/16**	**jue 19/05/16**			**Alt**
18	Disseny	mar 03/05/16	dom 08/05/16		Dissenyador	Alt
19	Implementació i test	lun 09/05/16	jue 19/05/16		40% Programador / 60% Tester	Alt
20	**Polítiques de planificació**	**vie 20/05/16**	**sáb 21/05/16**			**Baix**
21	Disseny	vie 20/05/16	vie 20/05/16		Dissenyador	Baix
22	Implementació i test	sáb 21/05/16	sáb 21/05/16		40% Programador / 60% Tester	Baix
23	**Integració amb MPI**	**dom 22/05/16**	**jue 02/06/16**			**Baix**
24	Disseny	dom 22/05/16	mar 24/05/16		Dissenyador	Baix
25	Implementació i test	mié 25/05/16	jue 02/06/16		40% Programador / 60% Tester	Baix
26	**Clàusula Commutative**	**vie 03/06/16**	**mar 07/06/16**			**Baix**
27	Disseny	vie 03/06/16	sáb 04/06/16		Dissenyador	Baix
28	Implementació i test	dom 05/06/16	mar 07/06/16		40% Programador / 60% Tester	Baix
29	Milestone: Informe de seguiment	vie 27/05/16	vie 27/05/16			
30	**Avaluació del resultat final**	**mié 08/06/16**	**vie 10/06/16**	**Editors de text;Eines de debug i profiling;Marenostrum;Models de prog;OpenSUSE;PC Acer;PC Dell**	**40% Programador / 60% Tester**	**Baix**
31	**Etapa final**	**sáb 11/06/16**	**vie 17/06/16**	**Eines d'ofimàtica;Gantter;PC Acer;PC Dell;Windows 7**	**Cap de projecte**	**Baix**
32	Milestone: Entrega memòria TFG	lun 20/06/16	lun 20/06/16			
33	Milestone: Defensa oral TFG	lun 27/06/16	lun 27/06/16			

Figura 36: Taula de Gantt amb la duració, dia inicial i final, recursos, rols i risc de les tasques.

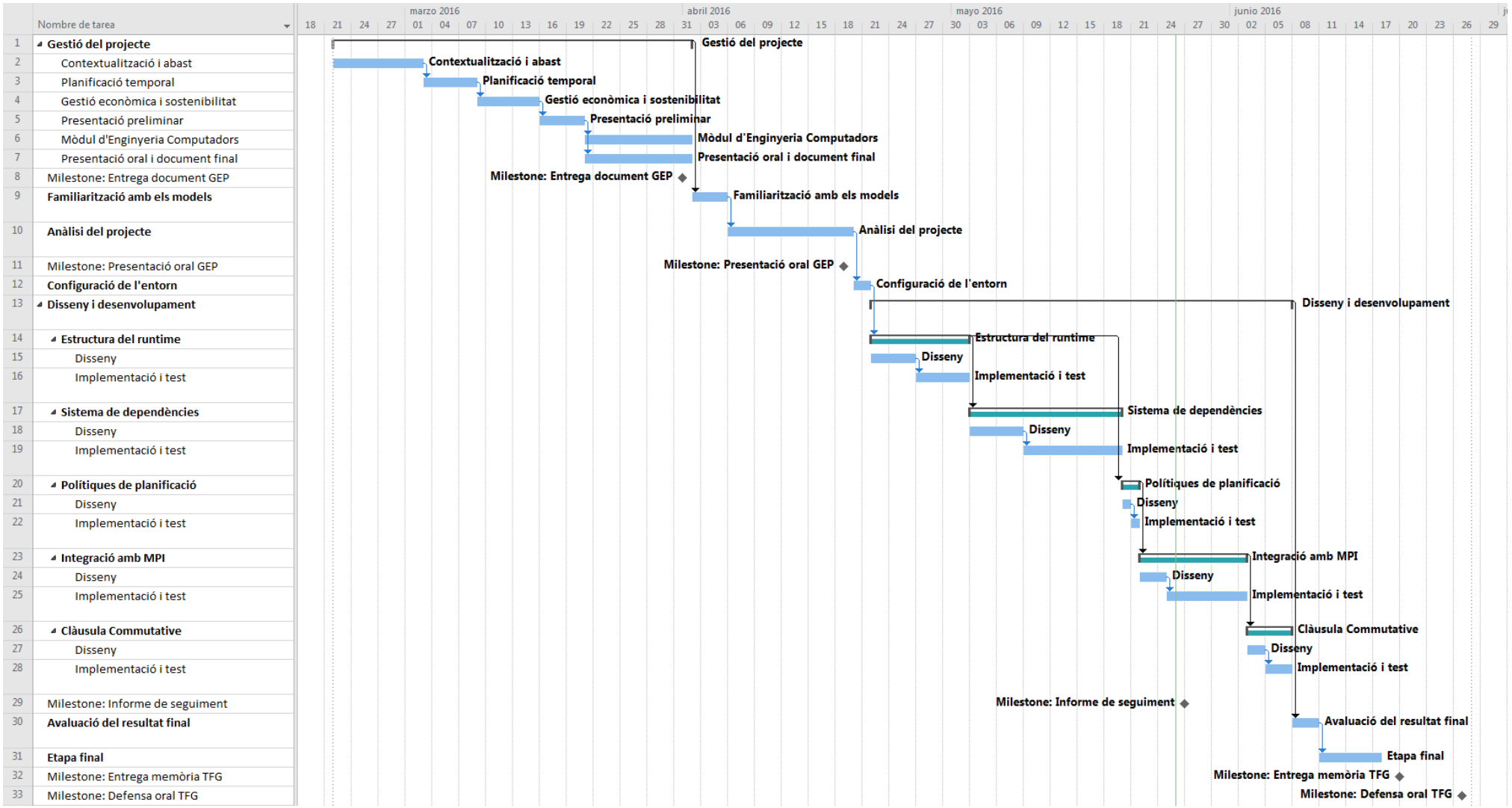

Figura 37: Diagrama de Gantt de la planificació temporal.

15 Referències

[1] Sangmin Seo, Abdelhalim Amer, Pavan Balaji, Pete Beckman, Cyril Bordage, George Bosilca, Alex Brooks, Adrián Castelló, Damien Genet, Thomas Herault, Prateek Jindal, Laxmikant V. Kale, Sriram Krishnamoorthy, Jonathan Lifflander, Huiwei Lu, Esteban Meneses, Marc Snir, and Yanhua Sun, *Argobots: A Lightweight Low-level Threading/Tasking Framework*, Recuperat de `https://collab.cels.anl.gov/display/ARGOBOTS/Argobots+Specification`, 2015.

[2] Barcelona Supercomputing Center - Centro Nacional de Supercomputación (BSC - CNS), *Descripció general del model de programació paral·lela OmpSs*, Recuperat de `https://pm.bsc.es/ompss`.

[3] Barcelona Supercomputing Center - Centro Nacional de Supercomputación (BSC - CNS), *Descripció general del compilador Mercurium*, Recuperat de `https://pm.bsc.es/mcxx`.

[4] Barcelona Supercomputing Center - Centro Nacional de Supercomputación (BSC - CNS), *Descripció general del runtime Nanos++*, Recuperat de `https://pm.bsc.es/nanox`.

[5] BSC, INRIA, Univeristy of Illinois, ANL, JSC , RIKEN/AICS, *Joint Laboratory for Extreme Scale Computing*, Recuperat de `http://publish.illinois.edu/jointlab-esc`.

[6] OpenMP.org (2015, Novembre), *OpenMP Application Programming Interface, Version 4.5*, Recuperat de `http://www.openmp.org/mp-documents/openmp-4.5.pdf`.

[7] Labarta ,J., Barcelona Supercomputing Center (BSC), *StarSs: a Programming Model for the Multicore Era*, Recuperat de `http://www.prace-ri.eu/IMG/pdf/08_starss_jl.pdf`.

[8] Teruel, X., Barcelona Supercomputing Center - Centro Nacional de Supercomputación (BSC - CNS) (2013, Juny), *OmpSs Quick Overview, A practical approach*, Recuperat de `http://www.training.prace-ri.eu/uploads/tx_pracetmo/OmpSsQuickOverviewXT.pdf`.

[9] Chamberlain, B. (2013, Maig 14), *Chapel: Productive Parallel Programming*, Recuperat de `http://www.cray.com/blog/chapel-productive-parallel-programming`.

[10] Barcelona Supercomputing Center - Centro Nacional de Supercomputación (BSC - CNS), *Paraver, Performance Analysis Tools: Details and Intelligence*, Recuperat de `http://www.bsc.es/computer-sciences/performance-tools/paraver`.

[11] Barcelona Supercomputing Center - Centro Nacional de Supercomputación (BSC - CNS), *Extrae, Performance Tools*, Recuperat de `https://www.bsc.es/computer-sciences/extrae`.

[12] Barcelona Supercomputing Center - Centro Nacional de Supercomputación (BSC - CNS), *Dimemas: predict parallel performance using a single cpu machine*, Recuperat de `https://www.bsc.es/computer-sciences/performance-tools/dimemas`.

[13] Subotic, V., Barcelona Supercomputing Center - Centro Nacional de Supercomputación (BSC - CNS) (2014, Octubre 14), *Tareador: understanding and predicting the potential of task decomposition strategies*, Recuperat de `http://www.bsc.es/sites/default/files/public/mare_nostrum/hpc-events/tareador_final_patc14.pdf`.

[14] Message-Passing Interface Forum (2015, Juny 4), *MPI: A Message-Passing Interface Standard, Version 3.1*, Recuperat de http://www.mpi-forum.org/docs/mpi-3.1/mpi31-report.pdf.

[15] Sangmin Seo, Abdelhalim Amer, Pavan Balaji, Pete Beckman, Cyril Bordage, George Bosilca, Alex Brooks, Adrián Castelló, Damien Genet, Thomas Herault, Prateek Jindal, Laxmikant V. Kale, Sriram Krishnamoorthy, Jonathan Lifflander, Huiwei Lu, Esteban Meneses, Marc Snir, and Yanhua Sun, *Argobots Public API*, Recuperat de http://www.mcs.anl.gov/~sseo/public/argobots/modules.html, 2015.

[16] *GIT, system version control*, Recuperat de https://git-scm.com/about

[17] TOP500.org (2015, Novembre), *Top500 List - November 2015*, Recuperat de http://www.top500.org/list/2015/11/.

[18] Microsoft Windows 7 Professional, *Botiga en línia PcComponentes.com* [Consultada: 30 de Març de 2016], Recuperat de http://www.pccomponentes.com/microsoft_windows_7_professional_64bits_oem_service_pack_1.html?gclid=CKjHmNb86MsCFesV0wodcsQFWw.

[19] Microsoft Office 365 Home, *Botiga en línia PcComponentes.com* [Consultada: 30 de Març de 2016], Recuperat de http://www.pccomponentes.com/microsoft_office_365_home_premium.html.

[20] Samsung Galaxy S5, *Botiga oficial en línia de Samsung* [Consultada: 30 de Març de 2016], Recuperat de http://www.samsung.com/es/consumer/mobile-devices/smartphones/galaxy-s/SM-G900FZKAPHE.

[21] Informe de Page Personnel, *Estudis de remuneració de Page Personnel del 2016 a Espanya*, Recuperat de http://www.pagepersonnel.es/sites/pagepersonnel.es/files/er_tecnologia16.pdf.

[22] Beca col·laboració 2015-2016, *Pàgina oficial del Ministeri d'Educació d'Espanya* [Consultada: 30 de Març de 2016], Recuperat de http://www.mecd.gob.es/servicios-al-ciudadano-mecd/catalogo/general/educacion/998142/ficha/998142-2015.html.

[23] Bitllet integrat TJove de 6 zones, *Pàgina oficial de TMB* [Consultada: 30 de Març de 2016], Recuperat de http://www.tmb.cat/es/sistema-tarifari-integrat/-/ticket/TJove.

[24] Fibra òptica 30Mb Movistar, *Pàgina oficial de Movistar* [Consultada: 30 de Març de 2016], Recuperat de http://www.movistar.es/particulares/internet.

[25] Departament de Física de la Universitats de Princeton, *N-body Simulations*, Recuperat de http://physics.princeton.edu/~fpretori/Nbody/intro.htm.

[26] European Center for Medium-Range Weather Forecasts (2015), *CY41R1 Official IFS Documentation*, Recuperat de https://software.ecmwf.int/wiki/display/IFS/Official+IFS+Documentation.