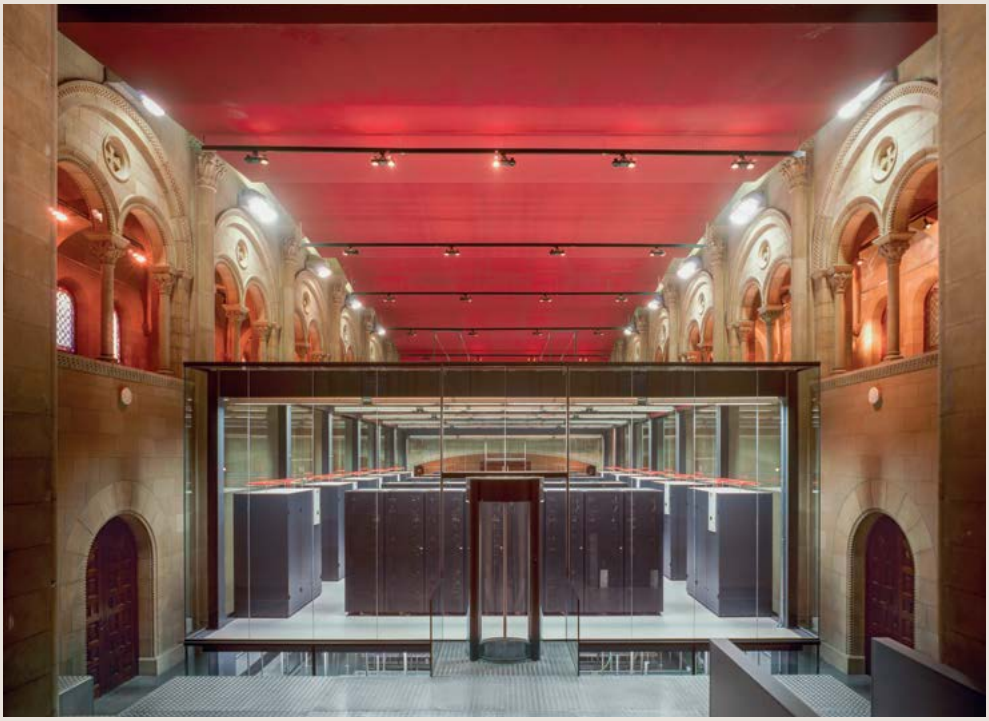
Models de programació per computació distribuïda

Discurs de presentació de Rosa M. Badia i Sala
com a membre numerària de la Secció de Ciències
i Tecnologia, llegit el dia 17 de juny de 2025



Institut d'Estudis Catalans
SECCIÓ DE CIÈNCIES I TECNOLOGIA

Models de programació per computació distribuïda

Models de programació per computació distribuïda

Discurs de presentació de Rosa M. Badia i Sala
com a membre numerària de la Secció de Ciències
i Tecnologia, llegit el dia 17 de juny de 2025

Barcelona, 2025

Institut d'Estudis Catalans
SECCIÓ DE CIÈNCIES I TECNOLOGIA

Biblioteca de Catalunya. Dades CIP

Badia Sala, Rosa M. (Rosa Maria), autor
Models de programació per computació distribuïda. — Primera edició
Bibliografia
ISBN 9788499657974
I. Institut d'Estudis Catalans. Secció de Ciències i Tecnologia II. Títol
1. Computació distribuïda 2. Supercomputadors
004.75
004.382.2

Aquesta publicació ha rebut un ajut de:

Diputació de Girona

Carrer del Carme, 47. 08001 Barcelona

Primera edició: juny de 2025

Edició: Flor edicions, SL
Disseny de la coberta: Azcunce | Ventura
Imatge de la coberta: MareNostrum 1, primer supercomputador del Barcelona Supercomputing Center. Font: Barcelona Supercomputing Center.

Imprès a Sistemes

ISBN: 978-84-9965-797-4
Dipòsit Legal: B 11764-2025

DOI: 10.2436/10.2000.86.1

1. Llenguatges de programació i models de programació

Una de les singularitats dels ordinadors en comparació amb altres màquines és que tenen una part de *hardware* (maquinari) i una part de *software* (programari). El programari es descriu seguint el que s'anomena llenguatge de programació, que no és res més que una sintaxi específica per descriure les instruccions dels ordinadors. Els primers llenguatges de programació apareixen al final de la dècada de 1940, amb l'aparició dels primers ordinadors programables (Eckert *et al.*, 1964). Aquests primers llenguatges de programació, anomenats llenguatges màquina, on un programa és una llista de codis que descriu les instruccions que ha d'executar el processador, queden molt lluny del que avui en dia coneixem com a llenguatge de programació.

Aquells llenguatges primitius van evolucionar durant els primers anys fins al 1957, quan va aparèixer el primer llenguatge de programació compilat, el Fortran (FORmula TRANslation) (Backus, 1978), dissenyat per poder descriure algorismes més entenedors per als humans i per abstraure els detalls del maquinari. L'any 1958 apareix el primer llenguatge funcional, el Lisp (Jones *et al.*, 2012), que permetia expressar recursivitat i expressions condicionals. El mateix any apareix també l'ALGOL, que permetia descriure més fàcilment algorismes i que esdevé l'antecessor de llenguatges moderns com C, Pascal, Ada, C++, Java i C#. I així fins als més de 300 llenguatges de programació actuals, dels quals els més populars són Python, Java, Javascript i C++. La figura 1 ens mostra els deu llenguatges de programació més populars segons la classificació «tendències» (*trending*) de la revista *IEEE Spectrum*.

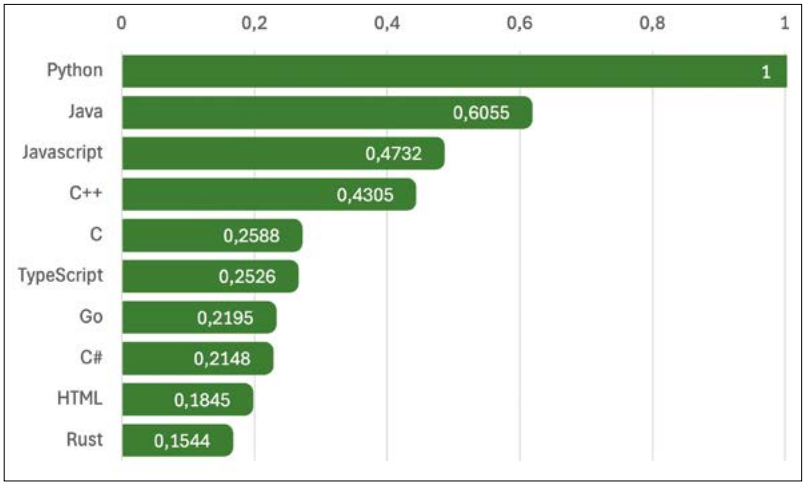


FIGURA 1. Llenguatges de programació més emprats.
FONT: Elaboració pròpia amb dades extretes de «Top Programming Languages 2024».[1]

En les primeres computadores, els llenguatges de programació s'executaven de manera seqüencial, és a dir, en cada moment s'executa una única instrucció. En una execució seqüencial, tècnicament un programa és una llista d'instruccions que s'han d'executar una darrere l'altra, però pot haver-hi salts per donar suport a expressions condicionals i bucles.

En aquest discurs volem diferenciar els llenguatges de programació respecte dels *models de programació*. Si un llenguatge de programació serveix per descriure l'algorisme o aplicació a executar per l'ordinador, el model de programació estén el llenguatge de programació amb un model d'execució, és a dir, la manera com efectivament s'executarà l'algorisme en la plataforma de computació. En aquest sentit, el model de programació seqüencial seria el que trobàvem en aquestes primeres computadores.

En l'evolució de les plataformes de computació cap a sistemes complexos, per exemple *clústers* que connecten un gran nombre de nodes de computació amb una xarxa d'interconnexió ràpida, o en l'evolució de les arquitectures amb l'aparició dels processadors multinucli i els acceleradors, ha aparegut la necessitat d'altres models de programació que donin suport a una execució paral·lela. Aquest discurs fa un recorregut sobre l'evolució de les infraestructures de computació d'altes prestacions i com els models de programació paral·lels han anat evolucionant per aprofitar-ne al màxim la capacitat alhora que faciliten la programació.

1. En línia: <https://spectrum.ieee.org/top-programming-languages-2024>.

2. Plataformes de computació paral·lela i distribuïda: dels supercomputadors al continu digital

2.1. Els primers supercomputadors

Els primers supercomputadors eren ordinadors molt potents per a l'època, però incloïen un nombre limitat de processadors. Per exemple, el supercomputador Cray-1 (any 1976) equipava 1-2 processadors en cada sistema.[2] Una característica dels processadors del Cray-1 és que eren vectorials, és a dir, cada instrucció podia operar amb un vector de dades. En els següents anys els fabricants de supercomputadors van anar afegint processadors als sistemes, però de manera moderada (per exemple, de 8 a 16 processadors). No és fins a la dècada de 1990 que els fabricants comencen a incrementar el nombre de processadors en els supercomputadors amb diferents estratègies, però sempre amb processadors especialitzats. Per exemple, el novembre de 1993 Fujitsu presenta el supercomputador Numerical Wind Tunnel amb 140 processadors vectorials i un rendiment màxim de 236 Gflops[3] de pic, que va passar a ser la màquina més potent llistada en el TOP500.[4] El TOP500 es publica dos cops l'any i llista en ordre de capacitat els cinc-cents supercomputadors més potents del món segons uns programes de referència. En l'edició anterior del TOP500, el juny de 1993, les quatre primeres posicions de la llista eren ocupades per Connection Machines (CM-5), fabricades per Thinking Machines, amb un nombre de processadors que oscil·lava entre 512 i 1.024, amb un rendiment de 131 Gflops de pic la més gran. Els processadors de la Connection Machine eren de tipus sistòlic, uns processadors molt simples que operaven en un sol bit. En aquest cas, els diferents processadors feien tots la mateixa operació en múltiples dades. En canvi, la màquina en cinquena posició era la SX-3/44 de NEC, amb només quatre processadors vectorials i un rendiment de 25,60 Gflops de pic.

El CEPBA-UPC, centre precursor del BSC, va ser seu de dos ordinadors singulars: KEOPS,[5] instal·lat el 1991, una Connection Machine 2 (CM2), amb 2.048 processadors sistòlics i 256 MB de memòria (figura 2), amb un rendiment de 0,64 GFlops, i KEFREN, instal·lat el mateix any, una CONVEX C3480 amb vuit processadors vectorials i 1 GB de memòria, amb un rendiment de 0,8 GFlops (figura 3).

2. *Cray history*. En línia: <https://cray-history.net/faq-1-cray-supercomputer-families/>.
3. El *flops* (*floating operation per second*) és la unitat de referència en computació d'altes prestacions i mesura el nombre d'operacions de punt flotant per segon que pot realitzar el sistema.
4. Lloc web del TOP500, en línia: <https://top500.org>.
5. Els computadors del CEBA-UPC tenien noms de faraons o de temples egipcis.

2.2. Sistemes de memòria distribuïda i de memòria compartida

El 1994, com a sistema alternatiu a aquests supercomputadors, es proposa la primera granja d'ordinadors o clúster al Centre de Vols Espacials Goddard de la NASA (Becker *et al.*, 1995). Aquest sistema estava pensat per resoldre problemes computacionals en àrees de ciències de la Terra i de l'Espai (*the Beowulf parallel workstation*). Els pioners d'aquest projecte foren els doctors Thomas Sterling i Donald Becker, que connectaren entre si setze ordinadors (nodes) de propòsit general (tipus PC), cadascun dels quals estava equipat amb un microprocessador Intel 486, controlats pel sistema operatiu Linux i interconnectats amb una xarxa Ethernet. El clúster desenvolupat tenia una eficiència de 70 Mflops, una capacitat avui en dia ridícula però relativament comparable (si es tenia en compte el nombre de processadors) a l'obtinguda pels supercomputadors de l'època, i gastant només 45.000 dòlars.

Tot i que aquests ordinadors que componen un clúster podrien ser diferents entre si (clústers heterogenis), en la seva evolució la majoria dels clústers es componen de nodes idèntics o molt similars. Els clústers són típicament sistemes de memòria distribuïda.

Per memòria distribuïda entenem una arquitectura de computadors en la qual cada unitat de processament (com un node en un clúster) té la seva pròpia memòria local i no comparteix memòria física directament amb altres unitats de processament. És a dir, des d'un node determinat podem accedir a la memòria del node, però no a la memòria d'altres nodes. Els supercomputadors actuals poden considerar-se clústers on els nodes i les xarxes d'interconnexió són molt especialitzats i d'alta gamma.

L'aparició d'aquests tipus de màquines va portar també nous models de programació paral·lels per sistemes de memòria distribuïda, essent els més rellevants el Parallel Virtual Machine (PVM) (Geist, 1994) i Message Passing Interface (MPI) (MPI Forum, 1994), que explicarem amb més detall en la secció 4.1.

Si els clústers són sistemes amb memòria distribuïda, cap al 1995 apareixen també sistemes amb arquitectures multiprocessador de memòria compartida (en anglès, *scalable shared memory multiprocessor architectures*, SSMP) (Lenoski *et al.*, 1995). En contraposició als sistemes de memòria distribuïda, aquests sistemes permeten accedir a qualsevol posició de la memòria des de qualsevol node. Aquesta arquitectura facilita la comunicació i la sincronització entre els processos d'una aplicació, ja que poden llegir i escriure directament en àrees de memòria compartida.

Un exemple de màquina de memòria compartida que va estar en funcionament al CEPBA-UPC és KARNAK (figura 4), una Origin 2000 de Silicon Graphics installada l'any 1997. Karnak tenia 64 processadors de tipus MIPS R1000 a 200 MHz, una

Figura 2. Ordinador KEOPS, Connection Machine 2 del CEPBA-UPC. Fotografia: Rosa M. Badia i Sala.

Figura 3. Ordinador KEFREN, Convex C3840 del CEPBA-UPC. Fotografia: Rosa M. Badia i Sala.

capacitat pic de 32 GFlops i una memòria de 12 GB accessible des de qualsevol dels processadors.

Els nodes de les màquines Origin 2000 estaven connectats entre ells per una xarxa d'interconnexió anomenada NUMAlink (Non-Uniform Memory Access Link). Sota una arquitectura NUMA, un processador pot accedir a la seva pròpia memòria local de forma més ràpida que a la memòria no local (memòria local d'un altre processador o memòria compartida entre processadors). Aquesta màquina era molt popular entre els usuaris, ja que el seu sistema operatiu IRIX era molt accessible i fàcil d'usar.

FIGURA 4. Ordinador KARNAK, Origin 2000 del CEPBA-UPC. Fotografia: Rosa M. Badia i Sala.

Per programar aquestes màquines de memòria compartida es va proposar OpenMP (Dagum *et al.*, 1998), que serà descrit amb més detall en la secció 4.1.

2.3. Supercomputació moderna

Els supercomputadors són tradicionalment clústers amb una capacitat de càlcul molt alta compostos per un gran nombre de nodes de computació d'altes prestacions connectats per una xarxa molt ràpida. Al BSC, el primer supercomputador MareNostrum 1 (vegeu la figura 5) es va instal·lar l'any 2004, amb un rendi-

ment de 42,3 Tflops (42,3 bilions d'operacions de punt flotant per segon). Aquesta capacitat va ser doblada amb MareNostrum 2 (94,2 Tflops) l'any 2006, posteriorment actualitzada a MareNostrum 3 amb 1,1 Petaflops (any 2012), i augmentada de nou a MareNostrum 4 (11,1 Petaflops) l'any 2017.

Figura 5. MareNostrum 1. Fotografia: BSC.

Voldria destacar que aquests quatre supercomputadors van ser tots instal·lats a la capella de Torre Girona, fet pel qual han estat coneguts i fins i tot han rebut el premi al centre de computació més bonic.[6]

El BSC ha tingut un impacte molt important en la supercomputació a escala nacional i europea. Amb MareNostrum 1, per primera vegada un supercomputador espanyol entrava en les deu primeres posicions del TOP500, més concretament en la posició 4. Poc després de la creació del BSC es va fundar la Xarxa Espanyola de Supercomputació (RES), que inclou MareNostrum 2 i altres clústers més petits distribuïts per tot el país, amb l'objectiu de donar servei a tots els investigadors espanyols. I no tan sols això: el BSC va influir en la creació, en la creació el 2010 de la xarxa europea de supercomputació PRACE (Partnership for Advanced Computing in Europe), que ha evolucionat el 2018 cap a l'European High-Performance Computing Joint Undertaking (EuroHPC JU).

EuroHPC JU té com a objectiu principal desplegar una infraestructura de supercomputadors a Europa de diferents capacitats: Petaescala, Pre-Exascala i

6. En línia: <https://www.bsc.es/news/bsc-news/marenostrum-4-chosen-the-most-beautiful-data-centre-the-world>.

Exascala. En concret, ha gestionat la instal·lació de nou supercomputadors fins al moment. A Barcelona s'ha instal·lat el supercomputador MareNostrum 5 (vegeu la figura 6), de 249,44 Pflops de pic, que a diferència dels seus predecessors ha estat instal·lat en un centre de dades del nou edifici BSC-Repsol.

Figura 6. MareNostrum 5. Fotografia: BSC.

2.4. Computació en grid *i en núvol*

En contraposició a la supercomputació, on una gran capacitat de càlcul es concentra en un centre de dades, tenim la computació en *grid* (o *grid computing*, en anglès), on la capacitat de càlcul està geogràficament distribuïda.

La computació en *grid* es va originar a principis dels anys noranta fent analogia amb la idea que l'energia dels ordinadors pogués ser tan fàcil d'accedir com la xarxa elèctrica. És a dir, que els usuaris es poguessin endollar a la xarxa de computació de la mateixa manera que endollen un electrodomèstic (Foster *et al.*, 1998). Un altre concepte també conegut com a *grid computing* és el que s'ha anomenat en anglès *volunteer computing*, que fa referència a quan els propietaris dels equips informàtics els posaven a l'abast de la xarxa quan no els feien servir, amb l'objectiu d'unir tota la seva capacitat per resoldre problemes d'investigació. Per exemple, va esdevenir popular el projecte SETI@home (Korpela *et al.*, 2001), que buscava una prova d'intel·ligència extraterrestre mitjançant el processament de senyals de ràdio. El projecte es basava en la infraestructura BOINC (Berkeley Open Infrastructure for Network Computing) (Anderson, 2020), un sistema de programari de codi obert per a *volunteer computing*, que posteriorment va dedicar-se a altres projectes de recerca com estudis climàtics, física d'altes energies o estudis matemàtics (cerca de nombres primers).

En l'àrea de *grid*, la comunitat científica es va organitzar al voltant del Global Grid Forum, GGF (anys 2001-2006), després anomenat Open Grid Forum, OGF (a partir de l'any 2006). L'objectiu principal de la iniciativa era la definició

d'estàndards per a *grid*. Organitzats en grups de treball, es feien reunions presencials trimestrals on es discutien les diferents propostes. Exemples de resultats d'aquestes reunions són els estàndards GridFTP, per permetre la transferència de fitxers grans entre nodes geogràficament distants, o l'Open Grid Services Architecture (OGSA), una arquitectura de serveis en *grid* definida com a extensió de l'arquitectura de serveis web.

Pel que fa al programari de base per als sistemes *grid* (*grid middleware*) van aparèixer múltiples entorns d'execució. Entre ells, els més populars van ser el Globus Toolkit (Foster *et al.*, 1997), que implementava múltiples estàndards definits en l'OGF, oferint serveis per executar treballs o moure fitxers, i el sistema Condor (Epema *et al.*, 1996), que tenia com a objectiu orquestrar treballs en granges d'ordinadors seguint el model de *volunteer computing*.

A Europa va haver-hi molts projectes *grid*, molts d'ells finançats per la Comissió Europea. Per exemple, la xarxa d'excel·lència CoreGRID,[7] que va agrupar durant quatre anys (de 2004 a 2008) 155 investigadors i 169 estudiants de doctorats a quaranta-una institucions europees, entre elles la UPC amb la participació del nostre grup. CoreGRID donava suport a la recerca bàsica i fomentava la col·laboració entre socis.

Un altre projecte rellevant que va agrupar 96 institucions va ser el projecte BEinGRID (2006-2009), que tenia com a objectiu principal fomentar l'adopció de les tecnologies *grid* mitjançant la realització d'experiments industrials i la creació d'un repositori d'eines de programari de *grid*. Es van desenvolupar 25 experiments aplicats a un ampli espectre de sectors industrials europeus (entreteniment, financer, industrial, química, jocs, comerç minorista, tèxtil, etc.).

D'altra banda, el CERN va coordinar una sèrie de projectes en *grid* per donar servei als seus experiments de física d'altes energies. El primer, el projecte DataGrid, va ser concebut al CERN per abordar la manca d'una capacitat informàtica i de processament de dades necessària per afrontar els enormes requisits del Large Hadron Collider (més de 10 petabytes de dades cada any). El projecte, participat per gairebé cent socis, va construir una *grid* distribuïda que unia granges d'ordinadors de múltiples centres internacionals a Europa fent servir Globus com a programari de base (LHC Grid). Altres projectes posteriors que van continuar aquests desenvolupaments van ser CrossGrid, EGEE, EGEE-II i EGEE-III.

Al final d'aquests projectes es va crear una nova organització (EGI.eu) per continuar la coordinació i l'evolució de la infraestructura de xarxa europea (EGI) a partir de la xarxa EGEE. EGI és un pas important per garantir que la comunitat investigadora europea tingui accés a una infraestructura de computació distribuïda

7. Lloc web del projecte CoreGRID: <http://coregrid.ercim.eu/>.

i pugui mantenir la seva posició de lideratge en recerca, i per donar suport al seu treball en col·laboracions globals durant molts anys.

D'altra banda, el CERN ha seguit treballant en el concepte de *grid* i actualment processa les dades dels seus experiments en la Worldwide LHC Computing Grid (WLCG), composta per dues-centes granges d'ordinadors a trenta-nou països de tot el món, molts d'ells gestionats amb HT-Condor, una evolució del programari Condor. En aquest sentit, per a l'experiment ALICE es fa servir un programari específic al qual la meva estudiant de doctorat Marta Bertran està contribuint amb millores a la gestió de recursos de computació.

Pel que fa al programari Globus, tot i que ja no es manté, hi ha components com el GridFTP que encara es fan servir per a grans transferències de dades.

La computació Cloud, o *Cloud computing* en anglès, també traduïda per «núvol» en català, apareix cap a l'any 2008. A diferència dels dos anteriors (supercomputació i *grid*), per a mi és més un model de negoci que una tecnologia. La computació en *cloud* el que cerca és oferir la computació com a servei, però amb un cost econòmic: és a dir, comercialitzar els serveis de computació. En realitat, és el que havíem dit que era la computació en *grid*, quan l'hem definida com la possibilitat d'obtenir serveis de computació com qui es connecta a la xarxa elèctrica.

És cert que, per poder oferir la computació com a servei, les tecnologies de virtualització de la computació han estat molt importants. Una màquina virtual (VM) és una tecnologia informàtica que utilitza programari en lloc d'un ordinador físic per executar programes i desplegar aplicacions. Una o més màquines «hostes» virtuals s'executen en una màquina «amfitrió» física. Cada màquina virtual executa el seu propi sistema operatiu de manera separada respecte d'altres màquines virtuals, encara que s'executin totes al mateix ordinador amfitrió. Tot i que les VM hostes comparteixen recursos, la tecnologia de virtualització aïlla de les altres les dades i aplicacions que conté.

El model d'execució en núvol tradicionalment s'ha realitzat amb màquines virtuals. Tot i que es poden tenir *núvols* privats, la tecnologia és principalment coneguda pels grans proveïdors comercials, com Amazon Web Services, Microsoft Azure o Google Cloud Platform.

De manera semblant a l'àrea de *grid*, hi ha hagut múltiples projectes finançats per la Comissió Europea. El projecte OPTIMIS, participat pel nostre grup al BSC, tenia com a objectiu optimitzar tot el cicle de vida dels serveis, començant per la construcció del servei en la qual el model de programació hi juga un aspecte important. Un altre exemple és el projecte ASCETIC, també participat pel nostre grup, que estenia la idea d'OPTIMIS tenint en compte l'eficiència energètica de tots els passos del cicle de vida dels serveis executats en el núvol, incloent-hi la construcció i l'execució del servei, on de nou el model de programació és clau.

2.5. El continu digital

Hem vist en les seccions anteriors com la computació ha anat evolucionant al llarg dels anys. Hem comentat les diferents alternatives, els clústers utilitzats en supercomputació i la computació en núvol.

Recentment, aquestes grans infraestructures de computació s'han vist esteses incorporant altres dispositius com ara sensors (càmeres, instruments de mesura) i grans instruments (telescopis, satèl·lits) que són fonts de dades molt nombroses i heterogènies. També s'han incorporat processadors *edge* propers a aquests dispositius, que processen part de la informació *in situ*, però que requereixen la connexió amb aquestes infraestructures de computació més grans per a determinats càlculs.

Aquest nou paradigma ha vingut a anomenar-se continu digital. Compost de dispositius grans i petits, heterogenis i dinàmics, integra no només capacitats de càlcul, sinó també d'emmagatzematge i connectivitat, obrint nous reptes de recerca tant en la seva programació com en la gestió dels recursos, la gestió de les dades i molts altres aspectes.

Similarment a les àrees de *grid* i *cloud*, la Comissió Europea està invertint en un nombre significatiu de projectes, actualment aglutinats en la iniciativa EU-CloudEdgeIoT.eu.[8] Aquesta iniciativa té com a objectiu posar mitjans per a la comprensió i el desenvolupament del continu de Cloud, Edge i IoT (CEI), promovent la cooperació entre una àmplia gamma de projectes de recerca, desenvolupadors i proveïdors, usuaris empresarials i possibles adoptants d'aquest nou paradigma tecnològic. Entre els projectes participats pel nostre grup, hi trobem AI-SPRINT (anys 2021-2023), que tenia com a objectiu la definició d'un nou entorn per al disseny i l'operació d'aplicacions d'intel·ligència artificial en el continu digital. Actualment, el grup participa en el projecte ICOS, que dissenya, desenvolupa i valida un metasistema operatiu per al continu digital, tenint en compte les diferents característiques de volatilitat i heterogeneïtat de l'entorn i paràmetres de rendiment i consum d'energia, entre altres. El grup també participa en DISCOVER-US, una acció de col·laboració entre Europa i els Estats Units, que busca definir temes de futur en el continu digital i en intel·ligència artificial, organitzant reunions amb participació internacional i posant mitjans per donar suport a la col·laboració internacional.

8. En línia: <https://eucloudedgeiot.eu>.

3. Evolució dels processadors: multicore, Cell i GPU

Però la complexitat de les infraestructures de computació no ve només pel nombre de components i per com s'organitzen, sinó també per l'evolució dels mateixos components. Una de les revolucions en el món del paral·lelisme ha estat l'aparició dels processadors *multicore* (o multinucli, en català). Com a resposta a una combinació d'obstacles tecnològics que cal superar, com ara la dissipació de potència, la latència de memòria i el poc paral·lelisme que quedava per explotar a nivell d'instrucció, els principals fabricants de xips van adoptar els dissenys multinucli com l'únic mitjà per explotar el nombre creixent de transistors dels xips.

Un processador multinucli és un sistema que inclou en un sol xip múltiples instàncies del processador (*cores* en anglès, o *nuclis* en català). Això va ser l'any 2005, quan Intel, seguint l'exemple del processador Power 4 d'IBM i el processador Niagara de Sun Microsystems, va anunciar que els seus microprocessadors d'alt rendiment inclourien d'ara endavant diversos nuclis (Asanovic *et al.*, 2006). Així doncs, anomenem *processador* el sistema complet inclòs en el xip o circuit integrat, i *core* (*o* nucli) cadascun dels processadors que el componen. La paraula *multicore* esdevé la moda en la indústria, i proposa duplicar el nombre de nuclis a cada circuit integrat en cada generació de processadors.

Aquest gran canvi en la indústria suposa també una de les revolucions (o crisis) en la programació. Per descomptat que aquest canvi ajuda a nivell de sistema, on ja es fa servir la multiprogramació, és a dir, múltiples processos s'executen simultàniament per resoldre diferents aspectes necessaris per al funcionament del sistema. Però, pel que fa a l'aplicació, la norma eren els programes seqüencials i, per tant, sorgeix la necessitat de nous models de programació que explotin el paral·lelisme més enllà de la programació manual amb fils, tal com explicarem en la secció 4.

Al principi, el nombre de nuclis per processador era limitat (*dual-core* o *quad-core*, és a dir, dos o quatre nuclis per processador). Però el progrés ha portat a incrementar aquest nombre i, per exemple, el processador Intel Sapphire Rapids, dels nodes de la partició de propòsit general del MareNostrum 5, té 56 nuclis (el punt més a la dreta en la figura 7), amb dos processadors d'aquest tipus per node. També es parla de *manycore* quan arribem a un nombre gran de nuclis per processador. En la figura 7 veiem l'evolució del nombre de nuclis de la família de processadors Intel Xeon al llarg dels anys, començant amb versions d'un sol nucli fins a l'esmentat Sapphire Rapids, amb 56 nuclis.

La revolució en les arquitectures no s'ha limitat els processadors multinucli, sinó que també han aparegut diferents tipus de processadors. El processador Cell Broadband Engine (CBE) (Hofstee, 2005) d'IBM, popularment conegut com a Cell, era un xip multinucli (vegeu la figura 8), que constava d'un processador PowerPC de propòsit general (PPE) que orquestrava les execucions i de múltiples

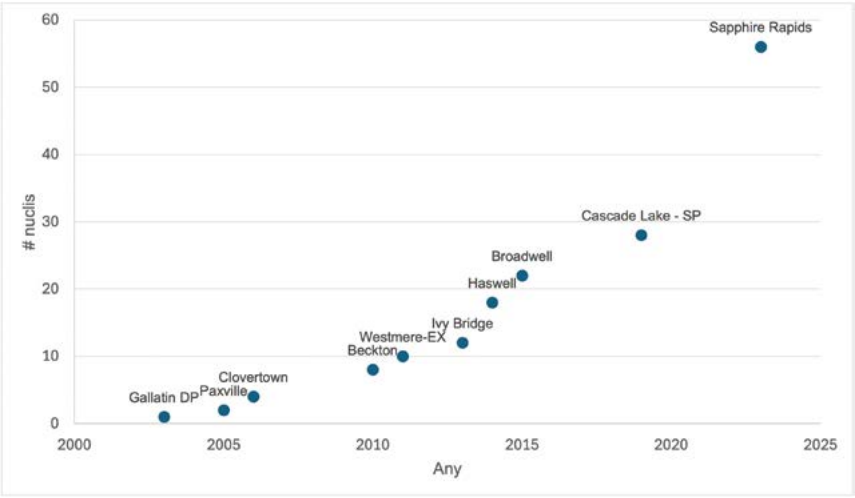


FIGURA 7. Evolució del nombre de nuclis de la família de processadors Intel Xeon (l'etiqueta indica el nom en clau del processador).
FONT: Elaboració pròpia.

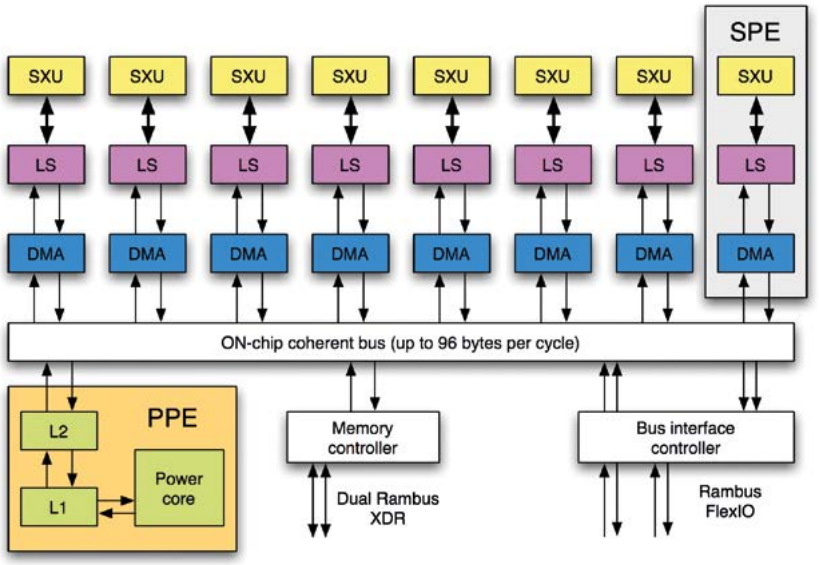


FIGURA 8. Arquitectura del processador Cell.
FONT: Hellips.

processadors *sinèrgics* o acceleradors (SPE), potents en càlcul (6 o 8 nuclis). Tots ells estaven connectats a un bus d'interconnexió, que també s'acoblava a la memòria principal i als dispositius d'entrada/sortida. Els SPE només podien accedir a la memòria principal mitjançant transferències d'accés directe a memòria (DMA) programant els seus controladors de memòria individuals. Per a cada SPE, les dades i el codi residien en la seva memòria local de 256 KB. Totes aquestes característiques feien molt interessant el processador Cell per l'eficiència que tenia, però també molt difícil de programar.

Per programar el processador Cell, IBM posava a la disposició dels programadors l'entorn Software Development Kit (SDK). Aquest SDK era de molt baix nivell i els programadors necessitaven programar els moviments de dades i gestionar totes les sincronitzacions entre processos paral·lels, a més de programar amb instruccions especials els càlculs.

Els anys 2005-2011 el BSC i IBM van mantenir la col·laboració MareIncognito, que tenia com a objectiu la construcció d'un supercomputador basat en el processador Cell i l'exploració del disseny d'arquitectura, models i eines de programació i aplicació per a l'arquitectura del processador Cell. En el capítol següent explicarem com vam treballar des del punt de vista del model de programació.

Pel que fa al maquinari, el prototip MariCel es va instal·lar l'any 2008 amb una arquitectura similar al sistema Roadrunner nord-americà (primer en la llista del TOP500 del juny de 2008). L'ordinador MariCel, exemple d'ordinador híbrid, disposava de 72 servidors QS22 IBM Blade, cadascun amb dos processadors PowerXCell a 3,2 GHz, i 12 GB de memòria (la versió d'alt rendiment del processador Cell amb operacions de coma flotant de doble precisió). A més, la màquina disposava de 12 servidors JS22 IBM Blade, cadascun amb dos processadors Power6 de propòsit general i 8 GB de memòria. Els nodes estaven connectats mitjançant una xarxa InfiniBand (16 Gb) i el rendiment màxim era de 14,4 TFlops, tot instal·lat en dos bastidors.

El processador Cell, a més d'aplicar-se a aplicacions de la supercomputació, era el processador de la consola de joc PlayStation 3, i fabricat per un consorci format per Sony, Toshiba i IBM. La PlayStation 3 va patir molts retards en la fabricació i a causa d'aquest motiu, entre d'altres, el novembre de 2009 el consorci va decidir aturar la fabricació de la línia de processadors Cell i el projecte MareIncognito també es va veure afectat.

Un altre exemple d'arquitectura que ha aparegut en els darrers anys el trobem en les unitats de processament gràfic (en anglès, *graphic processing unit*, GPU), que van ser inicialment dissenyades, com diu el seu nom, per al processament específic d'aplicacions gràfiques (visualitzacions, jocs...). Cap a l'any 2007 es comencen a fer servir les GPU com a acceleradors en supercomputadors gràcies a l'aparició de la sèrie NVIDIA GeForce 8, equipada amb unitats de processament més genèriques.

Semblant al cas del Cell, en el node de computació hi podem trobar un o més processadors de propòsit general (CPU) i un o més acceleradors (GPU). És el que es va anomenar *General-purpose computing on graphics processing units* (GPGPU). Les GPU són molt eficients per realitzar el mateix càlcul en paral·lel sobre un conjunt de dades, per exemple càlculs matricials. S'ha aplicat en camps tan diversos com l'aprenentatge automàtic, l'exploració de petroli, el processament d'imatges científiques, l'àlgebra lineal, l'estadística, la reconstrucció 3D i el càlcul del preu de les opcions d'accions. De manera semblant al Cell, la programació de les GPU és més complexa que la d'un processador de propòsit general. Aquesta complexitat prové del fet que les GPU generalment no comparteixen el sistema de memòria amb la CPU i, per tant, calen transferències de dades explícites entre els dos espais de memòria, i a més requereixen instruccions especials per programar els càlculs. En el cas de NVIDIA, el fabricant proporciona el llenguatge de programació CUDA[9] i un conjunt de llibreries específiques per a les GPU. El codi en CUDA només programa el càlcul a realitzar en les GPU, i la resta de l'aplicació cal programar-la en C/C++ o un altre llenguatge de propòsit general. Un altre inconvenient de CUDA és que és propietari de NVIDIA i específic per a les seves GPU, per això es van proposar també altres solucions obertes, com OpenCL,[10] proposat per Khronos Group i adoptat per un grup nombrós d'empreses, però CUDA ha seguit dominant. Posteriorment s'ha proposat SYCL,[11] un llenguatge inspirat en OpenCL i també proposat pel mateix Khronos Group, que proporciona una capa d'abstracció per programar no només GPU, sinó també altres acceleradors, com els FPGA.[12]

A diferència del processador Cell, que va deixar de produir-se, les GPU han revolucionat el mercat de la supercomputació, i altres fabricants com AMD o Intel en produeixen. Avui en dia pràcticament tots els supercomputadors equipen GPU, i són presents en nou de les deu primeres màquines llistades en el TOP500 de juny del 2024.

4. Models de programació

4.1. Models de programació paral·lels tradicionals: distribuïts i de memòria compartida

Hem comentat prèviament que amb l'aparició dels clústers i els supercomputadors de memòria distribuïda apareixen nous models de programació paral·lels

9. NVIDIA CUDA Toolkit. En línia: <https://developer.nvidia.com/cuda-toolkit>.
10. Lloc web d'OpenCL: <https://www.khronos.org/opencl/>.
11. Lloc web de SYCL: <https://www.khronos.org/sycl/>.
12. Un FPGA (Field Programmable Gate Array) és un dispositiu *hardware* que es pot configurar.

per a aquests sistemes, els més rellevants dels quals són Parallel Virtual Machine (PVM) i Message Passing Interface (MPI).

PVM va ser creat per l'Oak Ridge National Laboratory l'any 1989 i reescrit l'any 1991 a la Universitat de Tennessee. Aquest entorn de programació dona suport a xarxes heterogènies que es poden construir a partir d'una col·lecció de computadors distribuïts, i ofereix la il·lusió que componen una única màquina virtual paral·lela. És a dir, permet utilitzar com si fos un sol ordinador paral·lel gran una col·lecció de màquines heterogènies. PVM té interfícies en llenguatges de programació tals com C/C++ i FORTRAN.

D'altra banda, la Message Passing Interface (MPI) va aparèixer el 1994. És una especificació d'una llibreria que adreça el model de computació paral·lel basat en el pas de missatges. Sota aquest paradigma, processos paral·lels amb espais de memòria separats es poden comunicar entre ells mitjançant l'intercanvi de missatges. Desenvolupat per un consorci amb participants acadèmics, industrials i de laboratoris nacionals, MPI proporciona un estàndard àmpliament acceptat i estable del model de pas de missatges. En un programa MPI, típicament les dades es divideixen i reparteixen entre els diferents processos i cadascun opera amb una partició (normalment fent les mateixes operacions). Quan és necessari compartir informació, els processos intercanvien missatges. Per exemple, per enviar una dada que és necessària en un altre procés per als càlculs següents. MPI té operacions de tipus col·lectiu que programen operacions sobre tots els processos (per exemple, una operació *gather* recull dades de tots els processos, o en una *bcast* s'envien dades a tots els processos).

En comparació amb PVM, s'espera que MPI sigui més ràpid dins d'un sistema multiprocessador gran (Geist *et al.*, 1996). Té moltes més opcions de comunicació, punt a punt i col·lectives, que PVM. Això pot ser important si un algorisme depèn de l'existència d'una opció de comunicació especial.

PVM és millor quan les aplicacions s'executen en xarxes heterogènies. Té una bona interoperabilitat entre nodes diferents. PVM permet el desenvolupament d'aplicacions tolerants a fallades que poden sobreviure a errors dels nodes o de la tasca. Com que el model PVM es construeix al voltant del concepte de màquina virtual (no present en el model MPI), proporciona un conjunt molt potent de funcions dinàmiques de gestió de recursos i control de processos.

Tot i que PVM va esdevenir l'estàndard *de facto* durant un temps curt, MPI va esdevenir molt més popular entre els desenvolupadors d'aplicacions, se n'han implementat diferents versions i segueix essent molt comú en les aplicacions paral·leles, principalment escrites en C i Fortran. Les raons del domini de MPI davant de PVM van ser diverses, entre elles el desenvolupament d'un estàndard obert, l'existència d'implementacions molt optimitzades, la manca d'escalabilitat de PVM, el catàleg de funcions que ofereix MPI, molt més extens i avançat, la seva

portabilitat i el que és segurament més important, l'existència d'una comunitat forta i activa que ha anat estenent les funcionalitats i proveint implementacions.

El model de programació paral·lel per memòria compartida més reconegut és OpenMP (Open Multi-Processing), que va aparèixer el 1997.[13] Es va desenvolupar com una forma estandarditzada i fàcil d'utilitzar que permetés la programació paral·lela en sistemes de memòria compartida i en diversos llenguatges de programació com Fortran, C i C++. OpenMP ofereix directrius, funcions de llibreries i variables d'entorn per facilitar la programació paral·lela. Abans de la seva aparició, la programació paral·lela solia ser complexa i requeria un coneixement profund de les arquitectures de maquinari i les tècniques de sincronització.

Per exemple, OpenMP permet, amb una única anotació en el codi, la paral·lelització de bucles *for*, fent que una o més iteracions d'un bucle s'executin per un fil diferent. Un altre tipus d'anotació molt usada és la *reducció*, que permet també paral·lelitzar operacions de tipus acumulatiu (per exemple, la suma de tots els elements d'un vector o el producte escalar).

L'OpenMP Architecture Review Board (ARB) és l'organització responsable del desenvolupament i el manteniment de l'estàndard OpenMP. El seu objectiu principal és guiar l'evolució d'OpenMP, assegurant que l'estàndard segueixi sent rellevant i útil per a la comunitat de programació paral·lela. L'OpenMP ARB està format per diverses organitzacions membres, que inclouen fabricants de maquinari (com ara Intel o AMD), proveïdors de programari (empreses que desenvolupen compiladors i altres eines de programació que suporten OpenMP) i institucions acadèmiques que contribueixen amb recerca i desenvolupament en l'àrea de la programació paral·lela, entre elles el BSC.

A pesar que el model de programació de memòria compartida és més senzill i més intuïtiu que el de memòria distribuïda, donat que els sistemes distribuïts (clústers) han dominat els sistemes comercials per sobre dels de memòria compartida, MPI ha continuat essent un dels sistemes més utilitzats.

Tot i això, OpenMP s'ha mantingut, i també s'ha combinat donant peu a models de programació híbrids (MPI+OpeMP).

4.2. El model de programació superscalar

Per donar resposta a les necessitats de programació de les diferents plataformes de computació paral·lela s'ha proposat StarSs (Star superscalar) (Planas *et al.*, 2009), un model de programació proposat pel BSC i la UPC basat en tasques amb dos objectius principals: permetre l'explotació automàtica del paral·lelisme funcional

13. *The OpenMP API specification for parallel programming.* En línia: <https://www.openmp.org>.

(a nivell de tasca) i mantenir les aplicacions desconeixedores de la plataforma d'execució. El punt de partida de StarSs és una aplicació escrita de manera seqüencial en un llenguatge de programació tradicional (és a dir, C/C++ o Fortran) i la plataforma d'execució objectiu pertany a una sèrie de recursos paral·lels, com ara un xip *multicore* homogeni/heterogeni, un sistema de memòria compartida, un clúster o fins i tot un *grid* computacional.

Un primer pas és identificar en l'aplicació seqüencial les tasques que componen l'aplicació i la direcció dels paràmetres. Això es fa anotant en el codi (amb *pragmas*, decoradors, depenent del llenguatge de programació triat) funcions que en execució esdevindran tasques. L'anotació també indica l'ús que es fa dels arguments d'aquestes funcions (és a dir, si l'argument és de lectura o d'escriptura). Aquesta informació sobre les dades operades per les funcions es fa servir per calcular dinàmicament en temps d'execució del programa les dependències entre tasques. Això es registra en un graf de tasques (vegeu la figura 9) on cada vèrtex del graf representa la invocació d'una tasca i els arcs entre els vèrtexs representen dependències de dades entre elles. Aquest graf de tasques és una representació de l'aplicació que expressa el paral·lelisme potencial entre tasques. Les tasques que tenen una dependència entre elles s'han d'executar una darrere l'altre. En canvi, les tasques que no tenen dependències de dades entre elles es poden executar en paral·lel. Quan una tasca acaba, allibera les dependències que té amb tasques successores. Les tasques que no tenen cap dependència més es poden executar. Per exemple, en el graf de la figura 9, inicialment només la tasca 1 de color groc pot ser executada. Un cop acaba aquesta tasca, les tasques de color rosa poden executar-se, i si tenim prou processadors es poden executar totes en paral·lel. El model StarSs també dona suport a una execució asíncrona de les tasques: no cal que esperem que totes les tasques roses acabin per començar a executar les tasques blaves o la tasca vermella que depenen d'elles. Si, per exemple, la tasca 2 acaba, la tasca 9 pot començar a executar-se. Si després acaba la tasca 5, la tasca 8 podria executar-se, i així successivament. El color en el graf de tasques indica el tipus de funció que executen i l'identificador numèric es correspon amb l'ordre en què han estat invocades, que, tal com hem vist, no es correspon amb l'ordre d'execució.

En finalitzar la tasca, el graf de tasques s'actualitza i es prenen noves decisions de quines tasques executar.

A més d'executar les tasques en els diferents recursos paral·lels de la plataforma d'execució, l'entorn d'execució és capaç de realitzar altres activitats, com ara la transferència de dades quan sigui necessari i la sincronització. Depenent de la infraestructura computacional, les clàusules sobre la direccionalitat dels arguments també impliquen moviment de dades. És a dir, si es tracta d'un sistema distribuït, en temps d'execució es transferiran les dades d'entrada necessàries als espais de memòria accessibles pel processador responsable d'executar la tasca. El sistema

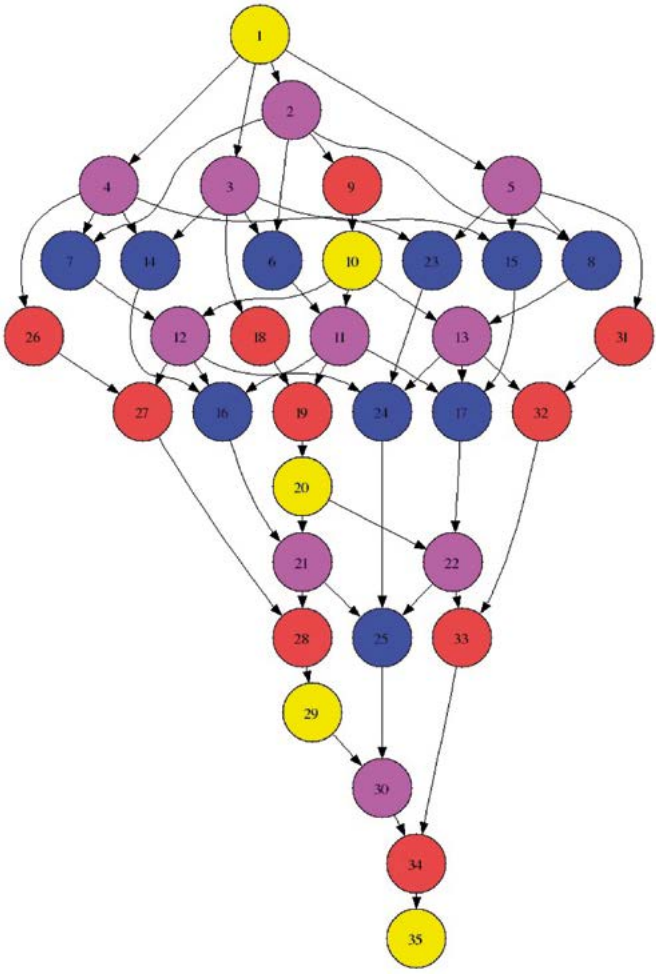


FIGURA 9. Graf de tasques.
FONT: Elaboració pròpia.

farà el mateix amb les dades de sortida, movent-les al node on s'hagi decidit que hagi d'executar la tasca que llegeixi aquesta dada. En general, aquest model d'execució redueix la necessitat de sincronitzacions globals que requereixen altres models com MPI i permet anar executant les diferents tasques d'una manera asíncrona a mesura que les dades s'han generat i els recursos estan disponibles. Així i tot, cal fer crides explícites per sincronitzar, que poden ser necessàries per esperar que un conjunt de tasques acabi o per transferir el resultat d'una o diverses tasques al node computacional que executa el programa principal.

4.3. Versions inicials de StarSs: Grid superscalar, SMPSs i CellSS

Aquest model de programació descrit a la secció anterior ha estat la meva contribució principal en recerca i, en particular, les múltiples implementacions que se n'han fet, per tal d'adaptar-se a les especificitats de cada infraestructura computacional.

El primer entorn per al qual es va desenvolupar el model va ser per computació en *grid* i es va anomenar Grid superscalar (GridSs) l'any 2003. Aquesta primera implementació, feta en el context de la tesi de Raül Sirvent (Sirvent, 2009), feia servir com a llenguatge de programació C/C++. La identificació de les tasques es feia amb un fitxer separat a partir del qual es generava codi addicional que feia de «goma» per generar el programa que es podia executar en paral·lel i en *grids* computacionals formades d'un conjunt de computadors geogràficament distribuïts. A més de la identificació de les tasques, la sintaxi de GridSs incloïa una petita API per llançar les tasques i sincronitzar-ne l'execució. Aquesta sincronització, en general, no és necessària gràcies al mecanisme de detecció de les dependències entre tasques, però sí que ho és si des del programa principal volem accedir a dades generades per les tasques. Això permet, per exemple, programar algun comportament condicional després d'executar un conjunt de tasques o realitzar alguna operació en el node local amb les dades generades.

Una característica d'aquesta versió era que les tasques eren de gra gruixut. El terme *gra* fa referència a la durada de les tasques, i dir que és *gruixut* vol dir que les tasques són de durada llarga. Per exemple, una tasca podia ser un càlcul o simulació sobre totes les dades d'un fitxer. Això era necessari, ja que les interaccions en el *grid* requerien temps addicionals: latència de comunicació entre els nodes distribuïts, temps de transferir les dades entre els nodes, etc. Una altra característica de GridSs era que les dades que intercanviaven les dades sempre eren fitxers. Cal tenir en compte que aquests fitxers viatjaven entre els diferents nodes de la *grid*, per tal que fossin accessibles per les tasques que operaven sobre ells.

GridSs comptava amb un motor d'execució potent que era capaç de prendre les decisions de planificació —quan i en quin node del *grid* executar cada tasca—, realitzava totes les transferències de fitxers entre nodes i implementava funcionalitats de tolerància a fallades, entre altres prestacions.

El motor feia servir el programari de *grid* Globus Toolkit per realitzar les funcions bàsiques en el *grid*, i també es va fer una versió sobre el protocol *ssh* per executar aplicacions GridSs dins de supercomputadors, fent-ne ús en producció als supercomputadors MareNostrum 1 i MareNostrum 2 (anys 2004-2011).

Un dels reptes importants en fer propostes de recerca en models de programació és trobar comunitats d'usuaris que en facin ús i que validin les propostes realitzades. En aquestes primeres versions inicials, el nombre d'usuaris externs

era relativament baix. GridSs va tenir un grup d'usuaris i col·laboradors de la Universitat de Castella - la Manxa (UCLM), liderat per Camelia Muñoz i Alfonso Niño, que van desenvolupar una aplicació en GridSs per al mapeig d'hipersuperfícies d'energia potencial molecular (Reyes *et al.*, 2007). L'ús de GridSs permetia a aquests investigadors automatitzar els càlculs i realitzar-los en paral·lel, accelerant l'obtenció de resultats científics. La figura 10 mostra una hipersuperfície d'energia potencial de dues dimensions calculada amb aquesta aplicació per la molècula de l'acetona quan es varien els valors dels angles θ1 i θ2. A més de col·laborar com a usuaris, Sebastián Reyes va realitzar la tesi a la UCLM amb noves contribucions a GridSs.

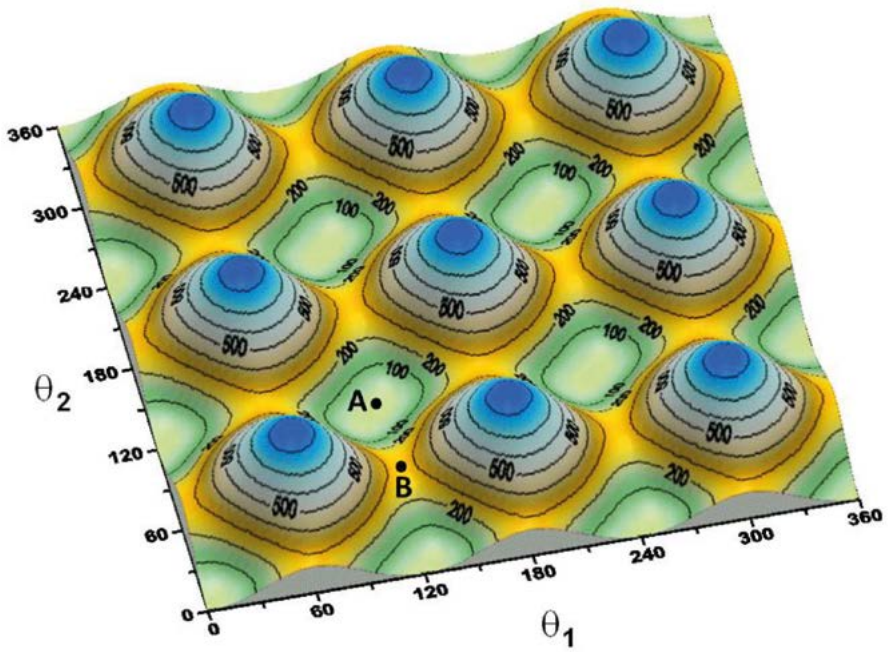


FIGURA 10. Hipersuperfície d'energia potencial bidimensional per a l'acetona en funció dels angles θ1 i θ2.
FONT: Reyes *et al.*, 2007.

Cap a l'any 2005 es van realitzar dues versions més del model StarSs: SMPSs i CellSs (Bellens *et al.*, 2006), que van formar part de les tesis de Josep M. Pérez (Pérez, 2015) i de Pieter Bellens (Bellens, 2012). SMPSs era una versió específica per a màquines de memòria compartida o per a processadors multinucli (tot i que en aquells moments els processadors tenien molt pocs *cores*), i CellSs era

una versió específica per al processador Cell. SMPSs i CellSs compartien la mateixa infraestructura de compilació i inicialment el mateix motor d'execució, però aquest segon es va haver d'especialitzar per al processador Cell, força diferent.

La sintaxi de SMPSs i CellSs es va definir com a anotacions tipus *pragma* en els llenguatges de programació C/C++ i Fortran. Així, a diferència de GridSs, el codi de l'aplicació ja tenia les anotacions incorporades en el codi. També, a diferència de GridSs, que es basava en generació de codi, SMPSs i CellSs es basaven en el fet de compilar el codi per generar un nou codi font que incloïa crides a la llibreria del motor d'execució i permetia generar els programes finals que explotaven el paral·lelisme.

Una altra diferència de GridSs amb SMPSs i CellSs és que les dades que precisaven les dependències de les tasques eren fitxers en el primer cas, però en els altres dos eren dades (objectes o variables del programa) emmagatzemades a la memòria. També, les tasques en CellSs podien ser de gra més fi que en GridSs, i encara podien ser més fines en el cas de SMPSs, ja que no hi havia transferències. En qualsevol cas, per tal d'amagar els temps addicionals que implica el model de tasques, les dades en què s'operava normalment eren blocs de dades. Per exemple, si l'aplicació volia fer un càlcul sobre totes les dades d'una matriu, les tasques operaven sobre blocs d'aquesta matriu.

Pel que fa al model d'execució, SMPSs, a diferència de GridSs, no necessitava moure dades entre els processadors, ja que estava enfocat a sistemes amb memòria compartida. En canvi, CellSs sí que necessitava moure dades de la memòria principal a les memòries locals dels SPE. Això simplificava moltíssim la programació d'aquest processador. Per exemple, en col·laboració amb el grup del catedràtic Enric Quintana, aleshores a la Universitat Jaume I, es va validar que amb SMPSs es podien paral·lelitzar fàcilment llibreries seqüencials d'àlgebra lineal (Badia *et al.*, 2009). Pel que fa a CellSs, en col·laboració amb un grup del centre de supercomputació de Jülich es va desenvolupar l'algorisme basat en partícules Multiparticle Collision Dynamics (MPC) per al càlcul de propietats hidrodinàmiques dels fenòmens de fluids i flux (Schiller *et al.*, 2010). Si la fabricació del processador Cell no s'hagués aturat, és probable que CellSs hagués tingut un impacte significatiu en la comunitat de desenvolupadors, ja que s'havia inclòs en la distribució de Linux Yellow Dog, específica per a la PlayStation 3.

Una altra col·laboració amb el grup del professor Enrique Quintana va proposar GPU superscalar (GPUSs) (Ayguadé *et al.*, 2009), una extensió del model de programació StarSs que tenia com a objectiu la paral·lelització d'aplicacions en plataformes de computació que incloïen diversos processadors gràfics (GPUs). GPUSs es fa càrrec de l'heterogeneïtat de l'arquitectura i dels espais de memòria separats, mantenint la simplicitat i la portabilitat en la programació.

4.4. La convergència amb OpenMP: OmpSs

Com hem vist fins ara, durant els anys 2005-2010 es van desenvolupar diverses versions del model StarSs per a processadors multinucli i acceleradors (Cell i GPUs). Per tal que les contribucions de recerca guanyessin en adopció per part de la comunitat, una de les contribucions importants del BSC a l'àrea es va centrar en influenciar l'estàndard OpenMP amb el model de tasques de StarSs.[14] Per aconseguir aquest objectiu, els conceptes que s'havien anat desenvolupant en les versions anteriors (principalment SMPSs i GPUSs) es van incorporar a OmpSs (Duran *et al.*, 2011), una implementació pròpia d'OpenMP desenvolupada al BSC, amb suport per C/C++ i Fortran (vegeu la figura 11).

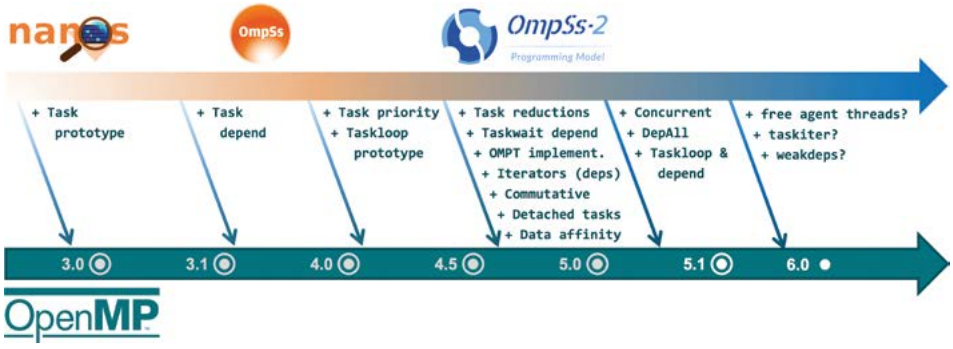


Figura 11. Contribucions del BSC a l'estàndard OpenMP.
Font: BSC.

La primera versió d'OpenMP que va incorporar el model de tasques va ser la versió 3.0 (any 2008). Val a dir que aquesta incorporació del model de tasques mantenia també el model de bucles inicial d'OpenMP, i així han seguit coexistint en les diferents versions. En aquesta primera versió amb tasques, només es va acceptar el concepte de tasca sense dependències de dades entre elles, és a dir, el model permet definir un conjunt de tasques que poden executar-se entre elles en paral·lel i en el motor d'execució s'encarrega que els diferents fils d'execució les vagin processant. La sintaxi defineix també anotacions que permeten indicar quan cal fer sincronitzacions en el programa principal per esperar que les tasques acabin.

No és fins a la versió 4.0 que OpenMP (any 2013) adopta també el model de dependències de dades definit a StarSs. Aquesta mateixa versió també incorpora

14. En línia: <https://pm.bsc.es/ftp/ompss-2/doc/spec/introduction/openmp.html>.

el model d'*offload* que permet delegar tasques a un accelerador (per accelerador es consideren dispositius tipus GPU, FPGA, etc.) amb un mecanisme semblant al que vam proposar primer a CellSs i després a GPUSs.

La versió 4.5 (any 2015) va incorporar la construcció *taskloop* que permet implementar un bucle amb tasques. Amb aquesta implementació es dona, doncs, suport a dues maneres de paral·lelitzar els bucles de les aplicacions: amb fils o amb tasques. Això dona més flexibilitat, ja que depenent de l'aplicació una o altra versió serà més eficient.

El BSC continua contribuint als grups de treball d'OpenMP proposant noves idees que es validen prèviament en OmpSs-2,[15] el successor d'OmpSs.

4.5. L'altra part de la família StarSs: COMPSs

La primera instància de StarSs per *grid* va ser GridSs, però en iniciar les activitats de recerca i col·laboració en l'entorn del projecte Europeu CoreGRID es va voler adaptar el model a la infraestructura de components Grid Component Model (GCM) dissenyada i implementada en el projecte. L'estudiant de doctorat Enric Tejedor, que actualment forma part del personal investigador del CERN, va dissenyar aquesta nova versió, anomenada COMP superscalar (COMPSs), que consta de diversos components, cadascun d'ells encapsulant una funcionalitat identificada en GridSs. Com a resultat, el motor d'execució va guanyar en capacitat de reutilització, millor desplegament, flexibilitat i separació de conceptes.

Si GridSs estava escrit en el llenguatge de programació C/C++, per tal de poder integrar-se amb la implementació del GCM de ProActive, es va triar Java com a llenguatge de programació per a COMPSs. En aquest sentit, la sintaxi de COMPSs se simplifica, ja que necessita només una interfície Java externa que declara els mètodes que esdevindran tasques i altres anotacions de les metadades necessàries per a les tasques. A part d'això, no requereix utilitzar cap API o anotació addicional en l'aplicació; tot és pura sintaxi i crides a funcions de Java estàndard.

Això era possible gràcies a un carregador de classes Java personalitzat que modifica l'aplicació en el moment de l'execució inserint-hi algunes crides necessàries; aquestes funcions de manipulació del codi font són proporcionades per Javassist, una biblioteca Java per a l'edició de classes.

Si bé en la versió inicial de COMPSs els paràmetres de les tasques només podien ser fitxers (igual que amb GridSs), en versions posteriors (Tejedor *et al.*, 2011)

15. Barcelona Supercomputing Center (2021). *OmpSs-2 releases.* En línia: <https://github.com/bsc-pm/ompss-releases>.

es va afegir el suport a objectes. Per tal de mantenir l'asincronia en la generació de tasques que retornen un objecte, es va implementar un mecanisme per a la gestió d'aquests objectes (objectes futurs).

Amb aquesta primera versió de COMPSs, i en col·laboració amb Josep Lluís Gelpí, del Departament de Ciències de la Vida del BSC, es va desenvolupar una versió del mètode *hmmpfam* en COMPSs (Tejedor *et al.*, 2010). Aquest mètode implementa una comparació de seqüències de proteïnes àmpliament utilitzada per la comunitat. Vam tenir coneixement, a través de la relació entre el Departament de Ciències de la Vida del Centre de Supercomputació de Barcelona (BSC) i l'Institut Europeu de Bioinformàtica (EBI), que l'EBI estava interessat a realitzar grans sèries d'execucions de *hmmpfam*, que és costós en càlcul, a MareNostrum 2. Per tal d'accelerar-lo, vam desenvolupar aquesta versió paral·lela de *hmmpfam* amb COMPSs.

Vam comparar el rendiment de la versió COMPSs amb la versió disponible en MPI, i vam comprovar que la nostra versió escalava millor amb el nombre de processadors i era, per tant, molt competitiva amb l'estàndard MPI (vegeu la figura 12). I el que és més important, aquesta aplicació va ser àmpliament utilitzada per la comunitat científica i va tenir l'impacte corresponent. COMPSs-HMMPfam es va utilitzar per fer proves enormes, processant més de set milions de seqüències de proteïnes i utilitzant prop de deu mil processadors. Aquests càlculs es van realitzar a MareNostrum 2, i des d'aleshores diferents versions de COMPSs han estat en producció a MareNostrum 3, 4 i 5, disponibles per als seus usuaris.

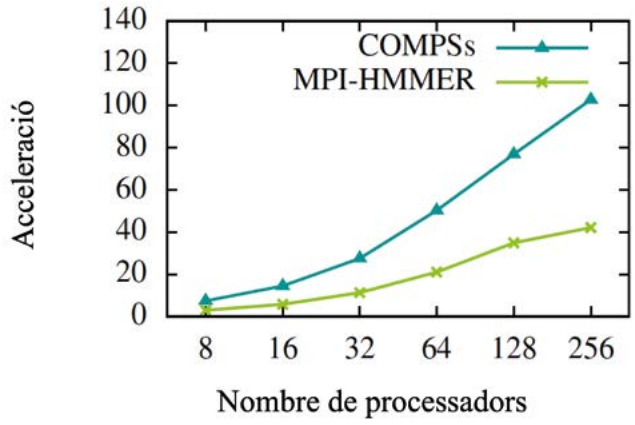


Figura 12. Comparativa de l'eficiència de COMPSs *versus* MPI.
Font: Tejedor *et al.*, 2010.

Amb l'aparició de la computació en núvol hi ha una tendència a substituir aplicacions monolítiques per aplicacions orientades a serveis. Un dels problemes que s'han de resoldre en l'existència de múltiples solucions per al núvol és que no són interoperables, la qual cosa obliga l'usuari a limitar-se a un proveïdor concret o a adaptar contínuament les aplicacions per a diferents proveïdors de *cloud*. Amb l'objectiu de simplificar els reptes dels programadors es proposa ServiceSs (Lordan *et al.*, 2014), una evolució de COMPSs que proporciona un model de programació per a aplicacions orientades a serveis executats en el núvol i un entorn d'execució que ajuda a abstraure les aplicacions de l'entorn d'execució real.

Un aspecte que cal ressaltar de les aplicacions ServiceSs és que poden estar compostes per dos tipus diferents de tasques: mètodes i serveis. En el primer cas es tracta de mètodes habituals de l'aplicació que són seleccionats per ser executats de forma remota. D'altra banda, els serveis corresponen a invocacions a serveis web tipus SOAP. Això permet combinar en una mateixa aplicació tasques de codi normal i invocacions a serveis web. A més, l'aplicació sencera podia ser desplegada com un servei que podia ser instal·lat en un node del núvol. Cada invocació a un servei de ServiceSs genera un *workflow* que inclou diferents tasques (vegeu la figura 13). Si s'invoca més d'un cop, el motor d'execució de COMPSs és capaç d'executar els diferents *workflows* generats de manera simultània. Pel que fa al

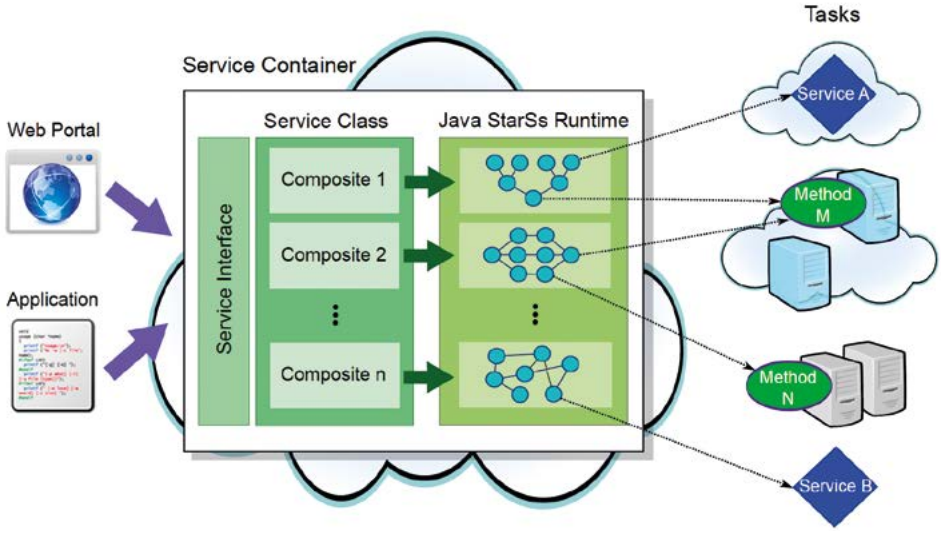


FIGURA 13. Arquitectura de ServiceSs.
FONT: Tesi d'Enric Tejedor.

desplegament, ServiceSs permet executar les aplicacions en múltiples núvols de diferents proveïdors, tant núvols públics (per exemple, Amazon Web Services) com privats (OpenStack). Aquest tipus d'execució utilitzant múltiples núvols és el que s'anomena federació, i el motor d'execució de ServiceSs dona suport a la interoperabilitat entre ells mitjançant connectors que internament ofereixen una interfície única, però externament s'adapten a les particularitats de cada proveïdor de núvol.

La implementació de ServiceSs estava basada en serveis SOAP, que van anar quedant obsolets ja que tothom utilitzava serveis REST. Recentment s'ha afegit el suport a serveis REST per poder donar suport a aquesta versió més moderna dels serveis web. Tots els desenvolupaments per donar suport a ServiceSs es van integrar en la infraestructura de COMPSs, de manera que el mateix entorn permet les aplicacions tradicionals de COMPSs i les orientades a serveis.

4.6. Convergència HPC, Big Data i AI

Cap a la dècada de 2010 es comença a parlar de Big Data i, tot i que inicialment el concepte el fan servir principalment empreses com Google, Amazon i Facebook, poc després és àmpliament reconegut com l'inici d'una nova transformació digital en la investigació científica i l'enginyeria en particular, però també en tota la societat en general. El primer model de programació per Big Data que va aparèixer és MapReduce (Dean *et al.*, 2008), inicialment desenvolupat per Google com a programari propietari, que es basa a organitzar les aplicacions de processament de dades en dues fases: una fase *map*, on totes les dades font són transformades per una operació, i la fase *reduce*, on els diferents resultats de la fase *map* són combinats per obtenir el resultat final. MapReduce permet processar i generar grans quantitats de dades realitzant les operacions de forma paral·lela i distribuïda en un clúster. Per tal de distribuir les dades, MapReduce fa servir un sistema de fitxers distribuït —en el cas de la solució de Google, Google File System (GFS). Poc després de publicar-se MapReduce van sorgir múltiples implementacions obertes, la més popular de les quals va ser la versió d'Apache Hadoop.[16] Posteriorment va aparèixer Spark (Zaharia *et al.*, 2010), un model de programació que va reemplaçar MapReduce en la majoria d'aplicacions de Big Data. Si bé MapReduce emmagatzema les dades en un sistema de fitxers distribuït, Spark es basava inicialment en l'estructura de dades RDD (Resilient Data Descriptions), que representen conjunts de dades immutables sobre els quals opera l'aplicació. Si MapReduce defineix els dos operadors Map i Reduce, Spark defineix múltiples

16. Apache Hadoop. Pàgina web: <http://hadoop.apache.org/>.

transformacions i accions que es poden aplicar sobre els RDD (cap a trenta operadors diferents). Les transformacions generen un nou RDD a partir del primer (*map*, filtrar, etc.) i les accions executen una *query* sobre els RDD (comptar, etc.). Posteriorment als RDD, Spark va definir altres tipus de dades: els *dataframe* i els *datasets*.

En els dos casos, MapReduce i Spark ofereixen un model de programació que limita el programador a fer servir els seus operadors. En canvi, la programació fent servir COMPSs dona més flexibilitat al programador, ja que pot definir les aplicacions sense aquesta restricció. Tot i que les aplicacions en Spark moltes vegades poden programar-se en menys línies de codi que en COMPSs, aquestes línies de codi requereixen una expertesa en Spark que no és necessària per a un programador de COMPSs, ja que ofereix un model més assequible als programadors no experts. Pel que fa als tipus de dades, COMPSs és també més flexible, ja que pot utilitzar qualsevol tipus de dada definit pel programador, i sistemes de fitxers distribuïts o els discos locals.

Pel que fa al rendiment, vam fer un estudi en què vam poder comprovar que COMPSs és competitiu amb Spark (Conejero *et al.*, 2018). La figura 14 mostra una comparativa del rendiment de l'aplicació WordCount en Java COMPSs i Spark, on veiem que el temps d'execució i escalabilitat és molt millor en el cas de COMPSs.

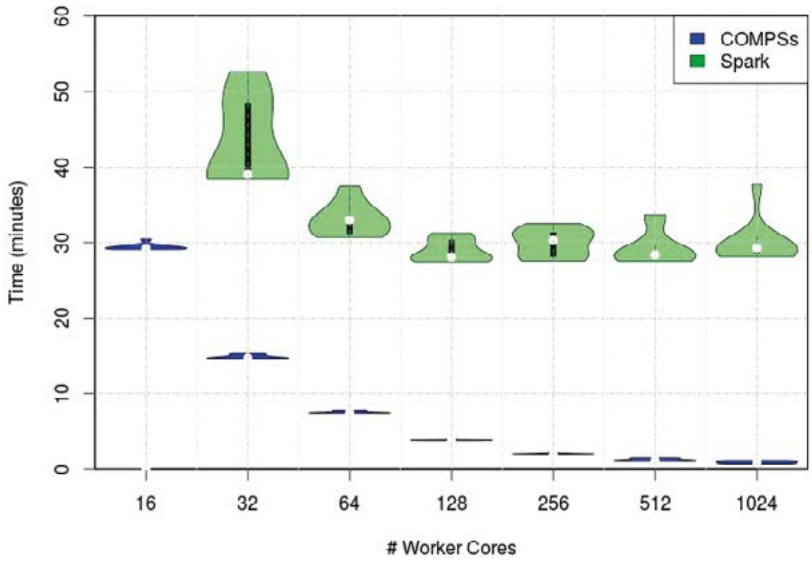


Figura 14. Comparativa entre COMPSs i Spark amb l'aplicació *wordcount*.
Font: Conejero *et al.*, 2018.

Tenint en compte que la comunitat d'usuaris de Spark és molt gran, es va estendre la interfície de COMPSs amb la Distributed Data Set (DDS) (Mammadli *et al.*, 2022), una biblioteca per facilitar el desenvolupament d'aplicacions PyCOMPSs amb la sintaxi de Spark. L'avantatge addicional és que el programador pot combinar l'ús dels operadors de DDS amb les tasques de COMPSs tradicionals.

Una altra revolució important que es fa patent no només en les tecnologies de la informació sinó en la societat en general es troba en l'ús massiu d'eines i mètodes d'intel·ligència artificial (IA). D'una banda, la supercomputació és instrumental per a la IA, ja que, per exemple, per entrenar els grans models de llenguatge (LLM) —com ara ChatGPT, per mencionar el que és probablement més conegut— és necessària una capacitat molt gran de computació. D'altra banda, la supercomputació es beneficia de l'evolució de la IA, ja que es pot fer servir per optimitzar els seus processos. I encara més, es poden desenvolupar aplicacions (o *workflows*) que combinin diferents mòduls de supercomputació tradicional amb models d'IA.

Un aspecte important que cal considerar pel que fa a la convergència o la convivència de la supercomputació amb el Big Data i la IA són els llenguatges de programació. Els llenguatges utilitzats en Big Data i IA mostren una gran intersecció, tots dos inclouen, com a més utilitzats, el Python, R, Julia i Java. En canvi, en CAP els llenguatges de programació tradicionalment s'han limitat a dos: Fortran i C/C++ (vegeu la figura 15). Pel que fa a Python i Julia, tot i que no es fan servir per al desenvolupament d'aplicacions de simulació o modelització, sí que són llenguatges també acceptats, per exemple per a aspectes relacionats amb l'anàlisi de dades generades

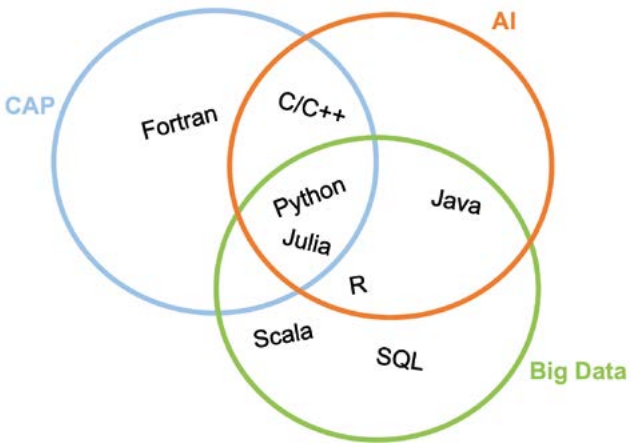


Figura 15. Llenguatges de programació més utilitzats en CAP, AI i Big Data.
Font: Elaboració pròpia.

per les simulacions. És per aquest motiu que cap a l'any 2015 vam decidir desenvolupar una interfície en Python per a COMPSs (Julia era molt incipient en aquells moments), que s'ha acabat anomenant PyCOMPSs (Tejedor *et al.*, 2017).

A més del ja esmentat, Python té alguns avantatges sobre altres llenguatges de programació: el llenguatge en si és compacte, llegible i molt adequat per a la creació ràpida de prototips, alhora que és prou potent per escriure aplicacions grans. Tot i que algunes persones no el consideren prou eficient, Python s'integra molt bé amb C/C++, la qual cosa permet invocar fàcilment mòduls externs programats en aquests llenguatges per obtenir bon rendiment. A més, té un ric conjunt de llibreries científiques, com ara, per exemple, per a càlcul numèric, processament i anàlisi de dades i interfícies gràfiques.

A hores d'ara, Python és la interfície més utilitzada de COMPSs per al desenvolupament d'aplicacions. S'ha anat estenent, donant suport a fallades de tasques i a fallades globals de l'aplicació, adquisició de dades en *streaming* i serveis REST, tal com heu mencionat fa poc, i altres.

El grup també ha proposat la llibreria *dislib* (Álvarez *et al.*, 2019), una llibreria d'aprenentatge automàtic paral·lelitzada amb PyCOMPSs. En aquest cas, el desenvolupament d'aplicacions no necessita conèixer PyCOMPSs, pot desenvolupar codis d'aprenentatge automàtic fent crides a la llibreria *dislib* i les aplicacions s'executaran en paral·lel.

Aquesta llibreria ha estat utilitzada, tal com descriu el discurs de Carme Jordi (2023), per analitzar les dades de Gaia, i ha permès detectar més d'un miler de cúmuls d'estrelles (per exemple, Castro-Ginard *et al.*, 2020 i 2022). L'ús de la *dislib* que paral·lelitza l'execució permet una reducció de fins a 64 vegades (depenent del nombre de processadors usats) el temps d'execució del codi, i requereix canvis mínims en la seva programació.

L'ús de PyCOMPSs ha estat especialment rellevant en les aplicacions (o *workflows*) desenvolupats en el projecte eFlows4HPC,[17] en les àrees de fabricació, modelització del clima o computació urgent per a desastres naturals. L'objectiu del projecte era oferir eines que permetessin que aquestes aplicacions poguessin combinar aspectes de CAP tradicional amb altres de Big Data i d'IA. També es volia demostrar que COMPSs permet aplicacions més dinàmiques, i se'n pot canviar automàticament el comportament segons si hi ha canvis en l'entorn d'execució o en les dades d'entrada.

Podríem ressaltar l'aplicació Urgent Computing Services for Earthquakes (UCIS4EQ) (De la Puente *et al.*, 2020), desenvolupada per l'investigador Josep de la Puente (Departament CASE del BSC) en col·laboració amb altres investigadors, que té com a objectiu generar mapes de com sacseja el sòl un terratrèmol de mo-

17. En línia: <https://eflows4hpc.eu>.

derat a gran, per ajudar a avaluar-ne els impactes. Consisteix en un conjunt de serveis i simulacions, combinant operacions d'anàlisi i processament de dades i simulacions d'altes prestacions, orquestrats per PyCOMPSs, que és capaç de detectar automàticament les alertes de terratrèmols i realitzar els càlculs necessaris per produir els mapes associats a l'alerta. Per a aquest cas s'han demostrat múltiples funcionalitats de COMPSs: la possibilitat de combinar serveis i tasques computacionals, el suport per a dades en *streaming* i la possibilitat de cancel·lar i destruir tasques en funció de noves dades.

En concret, el que fa l'aplicació (vegeu la figura 16) en detectar un terratrèmol és generar conjunts de paràmetres que es fan servir per iniciar l'execució de

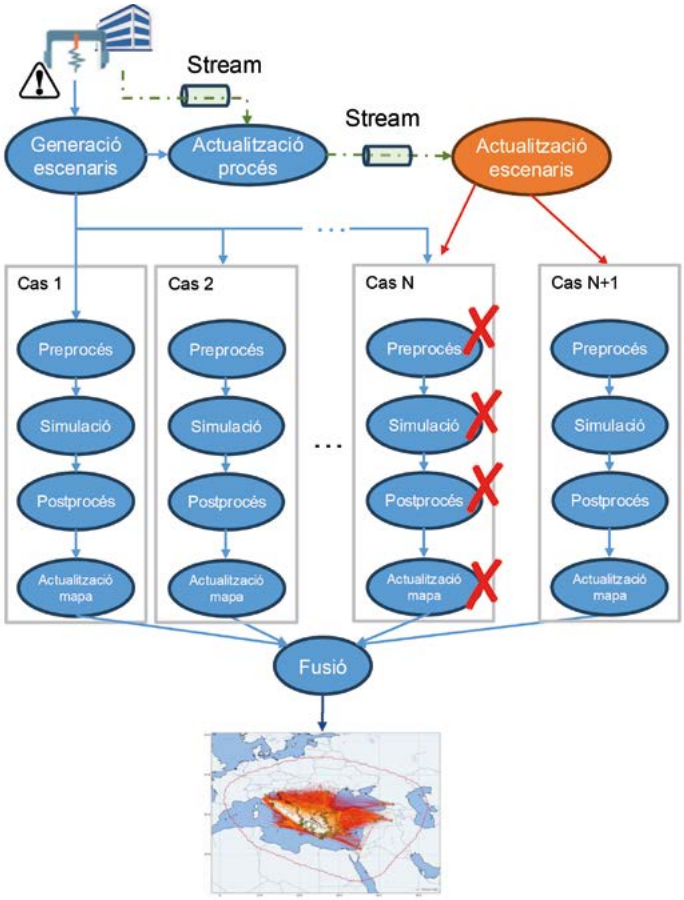


FIGURA 16. Esquema de l'aplicació UCIS4EQ.
FONT: Elaboració pròpia.

múltiples simulacions en un supercomputador amb caràcter d'urgència. Mentre aquestes simulacions s'han començat a executar, se segueix escoltant a la central d'alertes, i si es rep nova informació sobre el terratrèmol es pot decidir de manera dinàmica que algunes simulacions ja no són rellevants. En aquest cas, PyCOMPSs és capaç de cancel·lar-les i, si cal, iniciar l'execució de noves simulacions que són ara necessàries. L'aplicació se segueix desenvolupant en el projecte DT-GEO, que té com a objectiu desenvolupar bessons digitals de la Terra en aspectes de geofísica, en particular terratrèmols, tsunamis, volcans i extrems antropogènics.

4.7. COMPSs al continu digital

El projecte COMPSs també s'ha volgut adaptar als nous escenaris del continu digital. En un escenari tan distribuït i inestable com l'*edge-to-cloud*, la solució tradicional de COMPSs amb un motor d'execució centralitzat no era adequada. Per això, es proposa una solució basada en una infraestructura d'agents autònoms, on cada un conté també el motor d'execució de COMPSs (vegeu la figura 17).

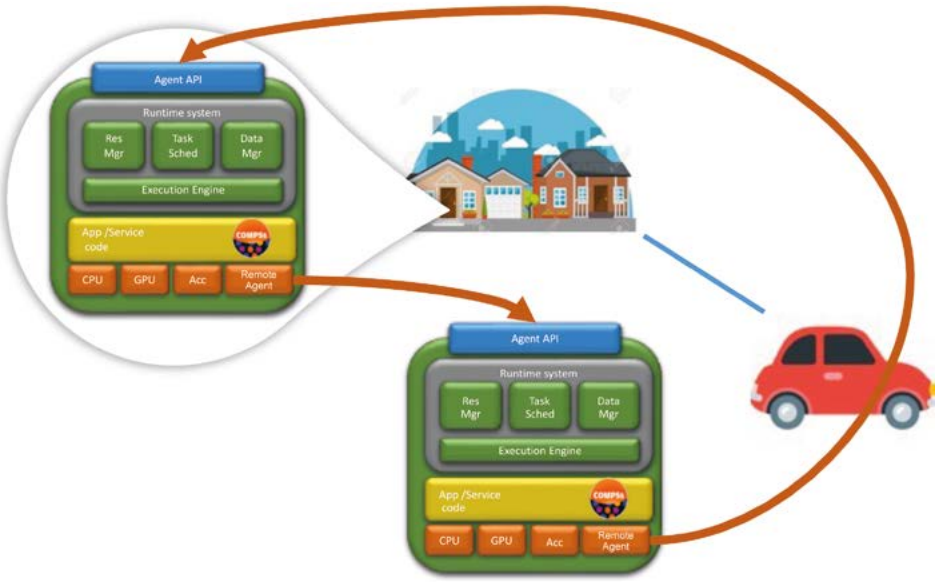


FIGURA 17. Nova arquitectura de COMPSs basada en agents.
FONT: Elaboració de Francesc Lordan [BSC].

Cada agent pot processar esdeveniments/dades en els seus recursos informàtics encastats, alhora que ofereix la seva capacitat de càlcul a la resta de la infraestructura seguint el paradigma de funció com a servei (Function as a Service, FaaS). A diferència d'altres solucions FaaS, la nostra permet convertir de manera transparent la lògica d'aquestes funcions en *workflows* basats en tasques (aplicacions COMPSs); així, els agents que allotgen l'execució del mètode generen el *workflow* corresponent i poden delegar part de la càrrega de treball a altres agents per millorar el rendiment global del servei.

Per donar més capacitat encara a aquest paradigma, s'ha estès també el model de programació amb la capacitat de poder definir tasques «niades» (Lordan *et al.*, 2023). És a dir, la capacitat de definir tasques dins d'una tasca, implicant una nova jerarquia de tasques diferent de la que ja definien les dependències de dades. Aquesta possibilitat és molt important: permet al programador definir com a tasques «grans» parts d'una aplicació que poden anar en paral·lel entre elles, però alhora permet definir un nivell de paral·lelisme intern per explotar millor les infraestructures de computació actuals. En el *continuum*, podem iniciar tasques «grans» en nodes separats, i les tasques més «petites» definides dins d'aquestes tasques es poden executar en els processadors de què disposa aquest node. El mateix tipus de mapeig es pot fer en els supercomputadors, on per exemple tenim la jerarquia *rack-node-processador*, que pot organitzar les aplicacions en jerarquies de tasques adequades per a aquesta arquitectura.

En el projecte espanyol HP2D-DT,[18] liderat per Eduard Prieto Araujo, del laboratori CITCEA de la UPC i participat pel nostre grup, s'ha proposat una aplicació desenvolupada amb el suport d'aquesta solució basada en agents per implementar un bessó digital de la xarxa del sistema elèctric. Aquest bessó permetrà una gestió en temps real de noves xarxes elèctriques d'energia renovable. Aquestes noves xarxes seran molt més dinàmiques i variables i requereixen una gestió en temps real.

El bessó digital, construït en el supercomputador, podria interactuar amb els elements físics de la xarxa elèctrica (per exemple, convertidors de potència, proteccions, etc.) mitjançant un conjunt de nodes en l'*edge*. Aquests nodes podran enviar l'estat del dispositiu al bessó digital, o realitzar computacions de manera local i autònoma (per exemple, un algorisme de detecció i mitigació basat en un model d'IA). En cadascun d'aquests nodes es desplegaria un agent de COMPSs, i tota la infraestructura de la xarxa estaria orquestrada en col·laboració per aquests agents (vegeu la figura 18).

18. En línia: <https://citcea.upc.edu/ca/projectes/hp2c-dt-bsc-aei-ted>.

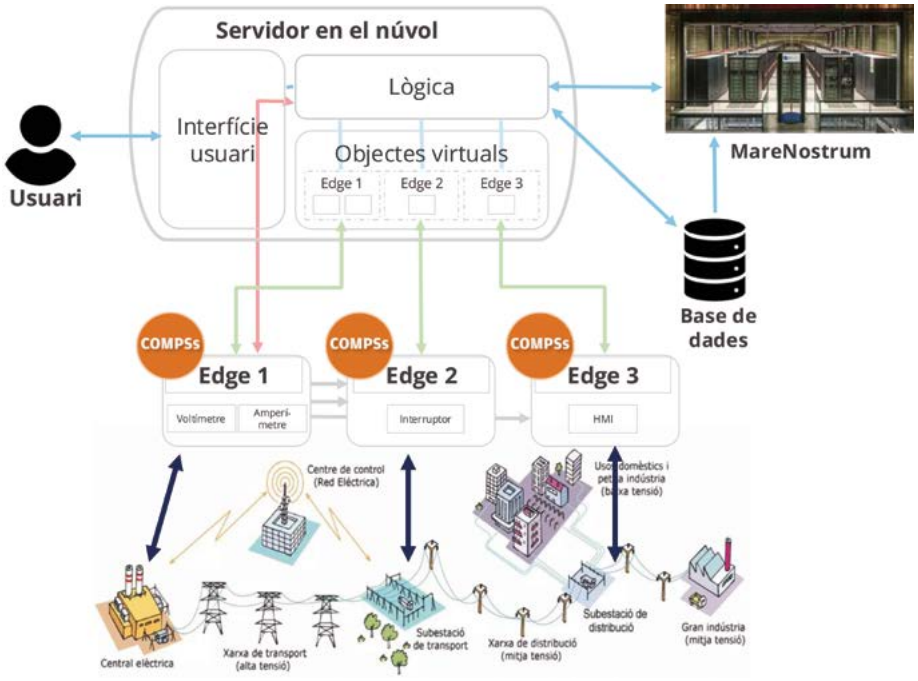


FIGURA 18. Arquitectura del projecte HP2C-DT.
FONTS: BSC i CITCEA-UPC.

5. CONCLUSIONS

Les infraestructures de computació d'alt rendiment han anat evolucionant al llarg dels anys. Els supercomputadors són màquines molt potents que donen suport a la resolució de problemes amb gran impacte per a l'avenç de la ciència i per a la societat. L'evolució de la tecnologia també ha portat diferents tipus d'infraestructures distribuïdes, com els clústers, el *grid*, el *cloud* i el continu digital. El canvi no és només en com s'organitzen aquests sistemes, sinó també en els seus elements bàsics de computació: els processadors i els acceleradors, que han sofert una gran evolució en pocs anys.

Els models de programació també han hagut d'evolucionar per adaptar-se a aquests canvis i per oferir interfícies fàcils d'usar però que alhora permetin a les aplicacions treure el màxim rendiment dels sistemes de computació. Des de la UPC primer i el BSC després, el meu grup hi ha contribuït amb el model de programació StarSs, basat en tasques i que té com a objectiu simplificar el desenvolupament d'aplicacions paral·leles alhora que ofereix un model d'execució que re-

dueix les sincronitzacions globals, permetent així un millor aprofitament dels recursos.

El model StarSs s'ha anat adaptant per a les diferents infraestructures de computació i tipus d'arquitectures, però també s'ha estès per poder integrar computacions de diferent tipus, com simulacions, models d'intel·ligència artificial i analítiques de dades. La taula 1 mostra un resum de les diferents versions i les seves principals característiques.

Taula 1. *Resum de les diverses versions del model de programació StarSs*

Model	*Granularitat de les tasques*	*Exemple de tasca*	*Entorn d'execució*	*Dades intercanviades per les tasques*
GridSs	Gruixuda (ms a dies)	Mètode o simulació externa	Computació en *grid*	Fitxers
SMPSs	Fina (µs a ms)	Mètode o funció en C	Un node d'un clúster	Objectes en memòria
CellSs	Fina (µs a ms)	Mètode executat a un SPE	Un node Cell	Objectes en memòria
GPUSs	Fina (µs a ms)	Mètode executat en GPU	Un node d'un clúster amb múltiples GPUs	Objectes en memòria
OmpSs	Fina (µs a ms)	Mètode o funció en C	Un node d'un clúster	Objectes en memòria
COMPSs en distribuït	Gruixuda (ms a dies)	Mètode o simulació externa	Computació en *grid* o *cloud*	Fitxers i objectes en memòria
PyCOMPSs en super-computador	Intermèdia (ms a minuts)	Mètode en Python	Supercomputador o clúster	Objectes en memòria
ServiceSs	Intermèdia (s a minuts)	Serveis REST	Cloud, continu digital	Fitxers i objectes en memòria
COMPSs Agents	Intermèdia (ms a minuts)	Mètode	Cloud, continu digital	Fitxers i objectes en memòria

Font: Elaboració pròpia.

La cosa més important és que el model ha servit per desenvolupar aplicacions amb impacte científic i social, donant sentit als nostres esforços de recerca i desenvolupament.

Mirant al futur, estem estudiant ja com PyCOMPSs pot ajudar a la computació quàntica dissenyant i desenvolupant una llibreria que faciliti la simulació i l'execució de circuits quàntics, que permeti partir el circuit en dos o més si el nombre de *qbits* és superior al disponible en el xip quàntic. Aquesta recerca volem que evolucioni per poder donar suport a aplicacions híbrides que s'executin en computació tradicional i computació quàntica.

Una altra àrea de recerca actual se centra en el disseny de metodologies per desenvolupar bessons digitals. Els entorns dels bessons digitals són molt heterogenis, amb components de computació tradicional i d'intel·ligència artificial, amb parts que poden ser accelerades en supercomputadors i dades que provenen de sensors i instruments externs, etc. Per tant, les tecnologies que hem desenvolupat amb PyCOMPSs per *workflows* on convergeixen la CAP tradicional amb la IA s'hi adapten molt bé.

Podríem continuar mencionant altres àrees de recerca, com noves extensions del model d'agents pel continu digital o nous models de programació bioinspirats, ja que, en definitiva, la recerca en models de programació és molt versàtil i alhora té un gran potencial per treure el màxim rendiment de les tecnologies que han de venir, amb el corresponent impacte en la ciència i la societat en general.

Bibliografia

Álvarez Cid-Fuentes, Javier [*et al.*] (2019). «dislib: Large scale high performance machine learning in python». *2019 15th International Conference on eScience (eScience)*. IEEE, p. 96-105.

Anderson, David p. (2020). «BOINC: a platform for volunteer computing». *Journal of Grid Computing* 18, núm. 1, p. 99-122.

Asanovic, Krste [*et al.*] (2006). «The landscape of parallel computing research: A view from Berkeley». Technical Report No. UCB/EECS-2006-183, University of California, Berkeley, USA.

Ayguadé, Eduard [*et al.*] (2009). «An extension of the StarSs programming model for platforms with multiple GPUs». A: *Euro-Par 2009 Parallel Processing: 15th International Euro-Par Conference, Delft, The Netherlands, August 25-28, 2009. Proceedings* [Springer Berlin Heidelberg] 15, p. 851-862.

Backus, John (1978). *The history of Fortran I, II, and III. History of programming languages.* Nova York: Association for Computing Machinery, p. 25-74. També disponible en línia: <https://doi.org/10.1145/800025.1198345>.

Badia, Rosa [*et al.*] (2009). «Parallelizing dense and banded linear algebra libraries using SMPSs». *Concurrency and Computation: Practice and Experience* 21, núm. 18, p. 2438-2456.

Becker, Donald J. [*et al.*]. (1995). «BEOWULF: A parallel workstation for scientific computation». *Proceedings, international conference on parallel processing*, vol. 95, p. 11-14.
Bellens, Pieter (2012). «Programming model and run-time optimizations for the cell/bE». Universitat Politècnica de Catalunya (UPC) [PhD diss.].
Bellens, Pieter [*et al.*] (2006). «CellSs: a programming model for the Cell BE architecture». *Proceedings of the 2006 ACM/IEEE Conference on Supercomputing*, p. 5.
Castro-Ginard, Alfred [*et al.*] (2020). «Hunting for open clusters in Gaia DR2: 582 new open clusters in the Galactic disc». *Astronomy & Astrophysics* 635, p. A45.
— (2022). «Hunting for open clusters in Gaia EDR3: 628 new open clusters found with OCfinder». *Astronomy & Astrophysics* 661, p. A118.
Conejero, Javier [*et al.*] (2018). «Task-based programming in COMPSs to converge from HPC to big data». *The International Journal of High Performance Computing Applications* 32, núm. 1, p. 45-60.
Dagum, Leonardo; Menon, Ramesh (1998). «OpenMP: an industry standard API for shared-memory programming». *IEEE computational science and engineering* 5, núm. 1, p. 46-55.
De la Puente, Josep [*et al.*] (2020). «Urgent supercomputing of earthquakes: Use case for civil protection». *Proceedings of the platform for advanced scientific computing conference*, p. 1-8.
Dean, Jeffrey; Ghemawat, Sanjay (2008). «MapReduce: simplified data processing on large clusters». *Communications of the ACM* 51, núm. 1, p. 107-113.
Duran, Alejandro [*et al.*] (2011). «Ompss: a proposal for programming heterogeneous multicore architectures». *Parallel processing letters* 21, núm. 02, p. 173-193.
Eckert, Jr John Presper; Mauchly, John W. (1964). «Electronic numerical integrator and computer». *U.S. Patent* 3 (4 de febrer), 120.606.
Epema, Dick H. J. [*et al.*] (1996). «A worldwide flock of condors: Load sharing among workstation clusters». *Future Generation Computer Systems* 12, núm. 1 (1996), p. 53-65.
Foster, Ian; Kesselman, Carl (1997). «Globus: A metacomputing infrastructure toolkit». *The International Journal of Supercomputer Applications and High Performance Computing* 11, núm. 2, p. 115-128.
— (ed.) (1998). *The grid: blueprint for a new computing infrastructure*. San Francisco, CA: Morgan Kaufmann Publishers Inc.
Geist, Al (1994). *PVM: Parallel virtual machine: a users' guide and tutorial for networked parallel computing*. MIT Press.
Geist, G. A.; Kohl, James A.; Papadopoulos, Phil M. (1996). «PVM and MPI: A comparison of features». *Calculateurs Paralleles* 8, núm. 2, p. 137-150.
Hofstee, H. Peter (2005). «Power efficient processor architecture and the Cell processor». A: *11th International Symposium on High-Performance Computer Architecture*. IEEE, p. 258-262.
Jones, Robin; Maynard, Clive; Stewart, Ian (2012). «The Art of Lisp Programming». *Springer Science & Business Media* (6 de desembre), p. 2.
Jordi, Carme (2023). «*Gaia*: la revolució del segle xxi a l'astronomia». Discurs de presentació com a membre numerària de la Secció de Ciències i Tecnologia. En línia: <https://publicacions.iec.cat/repository/pdf/00000371/00000039.pdf>.

Korpela, Eric [*et al.*] (2001). «SETI@ home-massively distributed computing for SETI». *Computing in science & engineering* 3, núm. 1 (2001), p. 78-83.
Lenoski, Daniel E.; Weber, Wolf-Dietrich (1995). *Scalable Shared-Memory Multiprocessing*. San Francisco: Morgan Kaufmann.
Lordan, Francesc [*et al.*] (2014). «ServiceSs: An interoperable programming framework for the cloud». *Journal of grid computing* 12, p. 67-91.
— (2023). «Hierarchical Management of Extreme-Scale Task-Based Applications». A: *European Conference on Parallel Processing*. Cham: Springer Nature Switzerland, p. 111-124.
Mammadli, Nihad [*et al.*] (2022). «DDS: integrating data analytics transformations in task-based workflows». *Open Research Europe* 2.
Message Passing Interface Forum (1994). «MPI: A Message-Passing Interface standard». *International Journal of Supercomputer Applications*, 8(3/4), p. 165-414.
Pérez, Josep M. (2015). «A dependency-aware parallel programming model». Universitat Politècnica de Catalunya (UPC) [PhD diss.] En línia: <https://upcommons.upc.edu/handle/2117/95636>.
Planas, Judit [*et al.*] (2009). «Hierarchical task-based programming with StarSs». *The International Journal of High Performance Computing Applications* 23, núm. 3, p. 284-299.
Reyes, Sebastián [*et al.*] (2007). «Performance of computationally intensive parameter sweep applications on Internet-based Grids of computers: the mapping of molecular potential energy hypersurfaces». *Concurrency and Computation: Practice and Experience* 19, núm. 4 (2007), p. 463-481.
Schiller, Annika [*et al.*] (2010). «Particle Methods on Multicore Architectures: Experiences and Future Plans». A: *AIP Conference Proceedings* [American Institute of Physics], vol. 1281, núm. 1, p. 1797-1800.
Sirvent Pardell, Raül (2009). «GRID superscalar: a programming model for the Grid». Universitat Politècnica de Catalunya (UPC). [PhD diss.] En línia: <https://upcommons.upc.edu/handle/2117/93329>.
Tejedor, Enric; Lordan, Francesc; Badia, Rosa M. (2011). «Exploiting Inherent Task-Based Parallelism in Object-Oriented Programming». A: *2011 IEEE/ACM 12th International Conference on Grid Computing*. IEEE, p. 74-81.
Tejedor, Enric [*et al.*] (2010). «Enabling HMMER for the Grid with COMP Superscalar». *Procedia Computer Science* 1, núm. 1, p. 2629-2638.
— (2017). «PyCOMPSs: Parallel computational workflows in Python». *The International Journal of High Performance Computing Applications* 31, núm. 1, p. 66-82.
Zaharia, M. [*et al.*] (2010). «Spark: Cluster computing with working sets». A: *2nd USENIX workshop on hot topics in cloud computing* (HotCloud 10).


INSTITVT D'ESTVDIS CATALANS
M CM VII


9 788499 657974