Simular entorns HPC amb aplicacions MPI per avaluar el rendiment fent ús de diferents tecnologies i millores

Treball Final de Màster

Autor: José Antonio Martín Pérez jmartinperez1@uoc.edu
Consultor: Alberto García Villoria agarciavillo@uoc.edu

Gener 2016

FITXA DEL TREBALL FINAL

Títol del treball:	*Simular entorns HPC amb aplicacions MPI per avaluar el rendiment fent ús de diferents tecnologies i millores.*
Nom de l'autor:	*José Antonio Martín Pérez*
Nom del consultor:	*Alberto García Villoria*
Data de lliurament (mm/aaaa):	*01/2016*
Àrea del Treball Final:	*Simulació i Modelatge*
Titulació:	*Màster en Enginyeria Informàtica*

Resum del Treball (màxim 250 paraules):

L'objectiu principal d'aquest treball és modelar un sistema HPC (*High Performance Computing*), fent ús de diferents tecnologies, tant a nivell topològic com d'interconnexió de xarxa, per tal de avaluar com afecten aquestes al rendiment final del sistema.

Algunes de les tecnologies que es volen avaluar són tant a nivell topològic (topologies Fat Tree, 2D Torus i 3D Torus) com a nivell d'interconnexió de nodes (Ethernet e InfiniBand). D'igual manera, es preveu estudiar quin tipus de topologia de xarxa és la més adient per al sistema final.

Tant el modelatge del sistema com les simulacions amb aquest es faran fent servint l'eina *SimGrid* (http://simgrid.gforge.inria.fr/). Aquesta eina permet fer tant el modelatge de sistemes de càlcul intensiu com avaluar-los fent servir programari tipus MPI (*Message Passing Interface*).

Aquest tipus de programari serà l'utilitzat per realitzar les simulacions, principalment buscant aplicacions de molt càlcul per als nodes com de molta comunicació entre ells.

Finalment, s'obtindrà tot un conjunt de dades amb temps d'execució I latències obtingudes desprès d'utilitzar les diferents tecnologies i millores aplicades, cosa que ens permetrà extraure les conclusions finals.

Abstract (in English, 250 words or less):

The main objective of this work is to model an HPC system (*High Performance Computing*) using different kind of technologies, both in a topological way and network interconnection, with the goal to evaluate how they affect the performance of the system.

Some of the technologies that we would like to check are related with different topological architectures (*Fat Tree, 2D Torus & 3D Torus*) and technologies applied to the interconnection between them (*Ethernet* & *InfiniBand*).

SimGrid (http://simgrid.gforge.inria.fr/) would be the tool used to model the system and simulate it. With this tool is possible to design HPC systems and evaluate them using MPI (*Message Passing Interface*) applications.

This type of software would be used to perform the simulations, mainly using applications with intensive calculation in the nodes and intensive communication between them.

Finally, the data obtained with the simulations will show executions times and latency performance using the different technologies. This data would help us to redact some final conclusions.

Paraules clau (entre 4 i 8):

HPC , MPI , SimGrid , simulation , network topology , link connection , FLOPs

Índex

Llista de figures

Llista de gràfiques

1. Introducció

1.1 Context i justificació del Treball

Les necessitats cada cop més habituals de gran càlcul intensiu porten a les universitats i al sector privat a necessitar accés a sistemes de tipus HPC (*High Performance Computing*), per tal de poder satisfer aquestes demandes.

Aquests tipus de sistemes basen el seu funcionament, principalment, en dividir tasques complexes entre diferents nodes, cosa que permet que treballin de forma paral·lela, reduint molt considerablement els temps d'execució finals.

La simulació i modelatge és una eina molt interessant per dissenyar aquests tipus de sistemes, atès que permet avaluar el rendiment final que s'obtindrà amb aquest, com a pas previ a l'elecció entre diferents tecnologies i/o topologies.

Aquest treball vol poder avaluar l'impacte de diferents tecnologies i tendències de mercat sobre un sistema final, per veure com l'elecció entre elles afecta al rendiment final que s'obté i, d'aquesta manera, servir com a eina que ajudi a extreure conclusions abans de fer una elecció final.

1.2 Objectius del Treball

- Estudi de diferents tecnologies utilitzades en sistemes HPC.
- Estudi de les topologies més habituals per aquest tipus de sistemes.
- Simulació i modelatge d'un sistema HPC.
- Desenvolupament de programari MPI per fer realitzar les simulacions.
- Proves de rendiment aplicant diferents configuracions sobre el sistema.
- Extreure unes conclusions finals.

1.3 Enfocament i mètode seguit

Primerament es vol mostrar d'una forma teòrica les diferents tecnologies i topologies habituals avui en dia en aquest tipus d'entorns. Es vol que aquest primer enfoc teòric serveixi com a punt de partida del treball.

Seguidament, es farà el modelatge d'un sistema HPC, fent servir l'eina *SimGrid* i programari tipus MPI per poder realitzar diferents simulacions sobre el sistema.

Per tant, aquest és el mètode escollit: modelatge del sistema fent servir programari per tal d'obtindre dades executant diferents simulacions.

Finalment, i en base a les dades que s'obtingui, es mostraran unes dades i gràfiques on es podrà avaluar com afecten les diferents tecnologies al rendiment final del sistema.

1.4 Planificació del Treball

	Nombre de tarea	Duración	Comienzo	Fin	Prede
1	0. Proposta de TFM	5 días	lun 21/09/15	dom 27/09/15	
2	**1. Fase 1**	**20 días**	**lun 28/09/15**	**dom 25/10/15**	
3	1.1. Introducció	10 días	lun 28/09/15	dom 11/10/15	
4	1.2. Tecnologies a nivell d'enla	5 días	lun 12/10/15	dom 18/10/15	3
5	1.3. Topologies LAN	5 días	lun 19/10/15	dom 25/10/15	4
6	**2. Fase 2**	**30 días**	**lun 26/10/15**	**dom 06/12/15**	
7	2.1. Modelatge amb SimGrid	10 días	lun 26/10/15	dom 08/11/15	
8	2.2. Desenvolapemt eines MPI	10 días	lun 09/11/15	dom 22/11/15	7
9	2.3. Primeres simulacions	10 días	lun 23/11/15	dom 06/12/15	8
10	**3. Fase 3**	**25 días**	**lun 07/12/15**	**dom 10/01/16**	
11	3.1. Implementació millores sis	10 días	lun 07/12/15	dom 20/12/15	
12	3.2. Segones simulacions	10 días	lun 21/12/15	dom 03/01/16	11
13	3.3. Conclusions i punt final	5 días	lun 04/01/16	dom 10/01/16	12

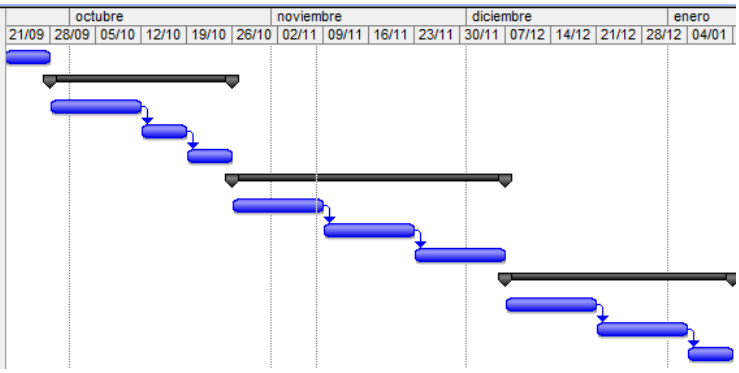


Figura 1. Diagrama de Gantt

1.5 Breu sumari de productes obtinguts

- Sistema HPC modelat amb l'eina *SimGrid*.
- Programari MPI per realitzar diferents simulacions.
- Dades de temps d'execució per cada una de les proves fetes.
- Gràfiques comparatives de les diferents dades.

1.6 Breu descripció dels altres capítols de la memòria

- Modelatge i plataforma utilitzada

Breu descripció de la generació de models per l'eina SimGrid i detall de la plataforma computacional feta servir per fer les simulacions.

- Codi font del programari MPI

Codi font de les diferents aplicacions tipus MPI fetes servir durant el procés de simulació amb l'eina SimGrid.

2. Conceptes teòrics

2.1. Tecnologies d'interconnexió

2.1.1. Gigabit Ethernet

Aquesta tecnologia és una evolució històrica de l'estàndard Ethernet ja present, sobre la qual es va millorar la velocitat de transmissió per tal d'augmentar molt significativament el rendiment, sense que hi hagués canvis molt significatius a nivell de maquinari o infraestructura.

A nivell físic, es defineix l'ús de tres tipus de cablejat:

- 1000BASE-X: Fibra òptica
- 1000BASE-T: Parell de coure sense apantallament
- 1000BASE-CX: Parell de coure amb apantallament

Encara que és possible trobar cablejat de fibra òptica en equipament específic, més enfocat al sector professional IT, el cablejat fent servir parell de coure s'ha imposat com l'estàndard a nivell d'infraestructures IT, incloses sistemes d'alt rendiment i clústers HPC.

El protocol 1000BASE-T treballa amb cablejat de categoria 5 o superior, que consta de quatre fils de coure. La gran diferència amb estàndards anteriors és que, en aquest cas, tots quatre cables són utilitzats, tant en l'enviament com la recepció de dades, fent servir tècniques d'equalització i modulació. Altres característiques no es veuen modificades en comparació amb el protocol previ, com són el nivell de modulació (125 megabauds) i la immunitat al soroll.

Precisament aquest fet el fa molt popular al món IT actualment, donat que no cal grans canvis a nivell d'infraestructura per realitzar el seu desplegament i, per altra banda, el preu és molt atractiu, tant a nivell d'equips de xarxa com del propi cablejat en sí.

Encara que el seu ús a sistemes HPC cada cop tendeix a ser menys habitual, donant pas a l'ús de protocols més eficients i que donen millors prestacions, em volgut usar aquest a mode de referència comparativa amb la resta de protocols. Per tant, serà una de les tecnologies que farem servir amb el simulador per tal de veure el rendiment del sistema amb aquest protocol.

Necessitem dues dades per a l'emulador, referents als protocols de comunicacions, que són la velocitat de transmissió i la latència. Referent als valors utilitzats, farem servir 1000 Mbps i 4,25 µs, respectivament.

2.1.2. 10 Gigabit Ethernet

Aquesta tecnologia és una evolució de l'anterior, augmentant la velocitat de transmissió de les trames *Ethernet* fins a una velocitat de 10 Gbps. Només es defineix l'ús de protocol *full-duplex* en enllaços punt a punt, mitjançant l'ús de switches com a mitjà d'interconnexió. Queden descartats altres tipus de protocols (*half-duplex*) i el concepte de '*hub*'.

A nivell de cablejat, es mantenen tant el parell de coure com la fibra òptica. Donat l'alta velocitat a la qual treballa el protocol, es necessita parell de coure d'una qualitat superior (categoria 6a o superior).

L'adopció d'aquesta tecnologia ha sigut més gradual que altres revisions anteriors, atès que l'abaratiment dels dispositius no ha sigut l'esperat, ja que la tecnologia anterior (*Gigabit Ethernet*) també s'ha vist molt reduïda de preu. Aquest fet ha fet que encara sigui força habitual trobar-se majoritàriament amb el protocol anterior en infraestructures IT.

A nivell físic, es defineixen diferents possibilitats als connectors, inclús es dóna la possibilitat d'usar connectors diferents en un enllaç punt a punt, permetent que cada extrem tingui connectors diferents. Connectors típics solen ser XENPAK, XFP i SFP+, sent aquest últim un dels més habituals actualment.

Parlant sobre cablejat de fibra òptica, es tenen dues classificacions: SMF i MMF. Aquesta darrera és l'habitual a nivell de xarxes locals, sent la primera més adient per llargues distàncies. Per tant, SMF queda descartada per al nostre propòsit en aquest treball.

Referint-nos al parell de coure, tenim fins a tres possibilitats compatibles amb el protocol: parell de coure avançat, cable tipus '*twin-axial*' (tipus InfiniBand) i *backplane*.

Ens centrarem únicament en dues possibilitats per a les nostres simulacions:

- SFP+ Direct Attach
  Ús d'un cable tipus 'twin-axial' passiu, connectat directament contra un connector SFP+. Els cables varien en la seva llargària, segons si son passius (7 metres màxim) o actius (15 metres màxim). Aquesta configuració és molt popular actualment.
- 10GBASE-T
  Ús de parell trenat de coure, amb o sense apantallament, i una longitud màxima de fins a 100 metres. Aquest fet facilita la migració de xarxes existents al nou estàndard, sempre i quan el cablejat sigui de bona qualitat (categoria 6A o superior). Per contra, l'ús d'aquest cablejat penalitza la latència obtinguda, fins a 4 microsegonds.

Per al nostre treball, suposarem la màxima velocitat de transmissió que dóna l'estàndard (10Gbps) i una latència de 0,3 µs per SFP+ i 2.25 µs per 10GBASE-T, respectivament.

2.1.3. 40 i 100 Gigabit Ethernet

Aquesta evolució de les dues tecnologies anteriors apareix l'any 2010 i pretén millorar la velocitat de transmissió, definint dues velocitats: 40 Gbps i 100 Gbps, respectivament.

Aquest nou estàndard, per primera vegada, defineix dues possibles velocitats de transmissió. L'idea és que cada velocitat de transmissió s'adopti a una part diferent de la infraestructura, quedant la menor velocitat par la connexió de servidors i equips de xarxa locals i reservant la velocitat de 100Gbps per a 'backbones' troncals de xarxa.

A nivell físic, es defineixen diferents tipus de connectors, de tipus connectable als equips d'electrònica de xarxa. El connectors de tipus òptic (fibra òptica) no s'han estandarditzat físicament oficialment, encara que hi ha acords entre fabricants per tal de fer compatibles els seus productes. Per exemple, a nivell de xarxes locals, s'han adoptat els connectors tipus QSFP i CXP.

El protocol és de tipus 'full-duplex' i manté, a nivell elèctric, les següents característiques:

- Es manté el format Ethernet a les trames (802.3 MAC).
- Mateixa mide mínima i màxima a les trames.
- Suport a OTN (Optical Transport Network).
- Correcció d'errors millor o igual a 10exp(-12).
- Proporcionat especificacions a nivell físic per operar sobre SMF (Single-Mode Optical Fiber), MMF (Multi-Mode Optical Fiber) OM3 i OM4, parell trenat de coure i 'backplane'.

Nomenclatura feta servir, a nivell físic:

Nivell físic	40 Gigabit Ethernet	100 Gigabit Ethernet
Backplane	40GBASE-KR4	100GBASE-KP4
Backplane millorat	-	100GBASE-KR4
7 m. sobre cable tipus twinaxial	40GBASE-CR4	100GBASE-CR10 100GBASE-CR4
30 m sobre cable parell trenat de Categoria 8	40GBASE-T	-
100 m. sobre OM3 MMF	40GBASE-SR4	100GBASE-SR10 100GBASE-SR4
125 m. sobre OM4 MMF		
2 km. sobre SMF	40GBASE-FR	100GBASE-CWDM4
10 km. sobre SMF	40GBASE-LR4	100GBASE-LR4
40 km. sobre SMF	40GBASE-ER4	100GBASE-ER4

Tipus de ports per l'estàndard 100 Gigabit Ethernet:

Nom	Clàusula	Medi físic
100GBASE-CR10	85 (802.3ba)	Cable tipis Twin-Axial
100GBASE-CR4	92 (802.3bj)	
100GBASE-SR10	86 (802.3ba)	Fibra Òptica Multi-Mode, 850 nm
100GBASE-SR4	95 (802.3bm)	
100GBASE-LR4	88 (802.3ba)	Fibra Òptica Single-Mode, WDM: 1295.56 nm, 1300.05 nm, 1304.59 nm, 1309.14 nm
100GBASE-ER4		
100GBASE-CWDM4	Non-IEEE	Fibra Òptica Single-Mode, WDM: 1271 nm, 1291 nm, 1311 nm, 1331 nm
100GBASE-PSM4	Non-IEEE	4 × Fibra Òptica Single-Mode 1310 nm
100GBASE-ZR	Non-IEEE	Single-mode fiber, 1546.119 nm
100GBASE-KR4	93 (802.3bj)	Backplane
100GBASE-KP4	94 (802.3bj)	

Per al nostre procés de simulació, treballarem amb les dues velocitats de transmissió que ens ofereix el protocol. Per als nodes computacionals, farem servir la velocitat de 40Gbps i, per contra, deixarem la velocitat superior (100 Gbps) per la interconnexió entre switches.

Referent a la latència, farem servir el mateix valor que dóna la versió anterior del protocol (10 Gigabit Ethernet).

2.1.4. InfiniBand

Aquesta tecnologia neix l'any 1999, no com una revisió o millora d'un estàndard previ, sinó com una definició totalment nova i enfocada a un sector molt específic: *clustering* i sistemes d'alt rendiment HPC.

La motivació que va propiciar el seu desenvolupament va ser l'alt cost de les connexions de fibra òptica en aquell moment. Aquest fet va portar a treballar sobre el parell de coure existent per tal d'augmentat el seu rendiment, amb un enfocament totalment diferent.

La tendència del moment era treballar amb un parell de coure amb cada cop més fils i més paral·lelisme sobre un mateix canal. Per contra, es va optar per un altre enfocament, on es volia millorar les velocitats sobre un únic parell, deixant de banda altres tipus de pretensions.

Per tant, es va treballar amb el parell de coure, fent que l'enviament sobre el segon cable del parell es fes amb una fase diferent. Amb aquest fet es van aconseguir velocitats al voltant del GHz però amb un cost molt inferior a la fibra òptica. De fet, i gràcies a aquesta aproximació, altres tecnologies es van veure també millorades fent servir aquesta aproximació (SATA i PCI-X, per exemple).

Durant el desenvolupament de la tecnologia, van aparèixer fins a dos grans grups ven diferenciats, recolzats per grans fabricants: NGIO (Intel, Sun & Dell) i FIO (IBM, HP & Compaq). La unió d'aquests dos va crear el consistori *InfiniBand Trade Association*, que va promoure el desenvolupament i l'estandardització d'aquest.

De manera inicial, es pretenia que aquesta nova tecnologia substituís altres tecnologies presents al sector IT, com Ethernet i FC, i, per tant, s'adoptés com a tecnologia única d'interconnexió. Però factors externs a ella, com són de tipus econòmic, competència amb tecnologies ja existents (Ethernet i FC), poc recolzament de part de la indústria (Intel va deixar el desenvolupament), van fer que els hipotètics clients no mostressin molt entusiasme en la seva adopció.

Encara que pogués semblar que aquest estàndard estava destinat a desaparèixer (part de la indústria així ho creia), van sorgir sectors molt específics on va tindre un petit èxit i, actualment, és la tecnologia d'interconnexió més utilitzada a superordinadors e infraestructures HPC.

La topologia que fa servir l'estàndard és lleugerament diferent a l'habitual sobre xarxes tipus Ethernet, atès que és possible que un mateix node connecti contra diferents switches, fent servir diferents canals de comunicació, cosa que proporciona redundància i augment del rendiment. L'habitual és que es tindre agregacions tipus 4x o 16x, es a dir, quatre o setze canals de comunicació per cada node, respectivament.

Relació de velocitats i latències amb les diferents revisions de l'estàndard (agregacions 4x):

	SDR	DDR	QDR	FDR	EDR	NDR
Velocitat (Gbps)	10	20	40	56	100	180
Latència (µs)	5	2.5	1.3	0.7	0.5	0.35

2.2. Topologies de xarxa

2.2.1. Topologies clàssiques

Punt a punt
La connexió més senzilla que existeix. Simplement consisteix en connectar dos enllaços directament entre ells, sense passar per cap element intermedi.
Al nostre cas concret, serà el mètode utilitzat per connectar els diferents nodes de computació amb l'electrònica de xarxa.

Bus
Topologia en la qual els diferents elements de xarxa connecten contra un canal comú de comunicació, anomenat '*bus*'.
Per tant, tota la informació viatja de forma comuna sobre aquest canal i són els diferents nodes que han de comprovar si aquesta s'adreça a ells. En el que cas que no sigui així, les dades són ignorades.
La seva gran avantatge és la simplicitat a nivell estructural i el baix cost. Per contra, es necessita un cost elevat per mantenir de forma adequada la infraestructura i la dependència d'un únic canal de comunicació suposa un risc.

Estrella
Aquesta topologia sorgeix quan diferents elements punt a punt connecten contra un element central que, a la vegada, realitza la tasca d'encaminament de la informació entre ells. Per tant, els nodes connecten punt a punt i, per tant, estan tots connectats entre d'ells d'una forma indirecta.
Aquesta configuració és la més senzilla d'implementar i dissenyar. De la mateixa manera, és força senzill que escali sense complicacions. Per contra, la dependència d'un element central d'interconnexió afegeix un risc en cas que aquest provoqui fallida.

Anell
Tots els nodes de la xarxa connecten entre ells formant un anell tancant, on les dades viatgen en un mateix sentit. Per tant, cada element de la xarxa actua com a client i com a repetidor d'aquestes, en cas que no siguin per a ell.
Per tant, la viabilitat de xarxa recau en els diferents nodes, atès que una fallida en un d'ells faria que la xarxa quedés trencada.

Malla
Tots els elements de la xarxa connecten entre ells, fent servir connexions punt a punt.
La gran avantatge és que no es depèn de cap element central o node per mantindre la integritat de la xarxa, però el fet de tindre tots el nodes interconnectats directament suposa que sigui impracticable quan el nombre de nodes és elevant.

Arbre
Aquesta topologia és, en realitat, una combinació de les topologies bus i estrella. S'utilitza un canal comú (*bus*) per interconnectar diferents topologies en estrella, mantenint els inconvenients de la topologia bus amb els avantatges de la topologia estrella.

2.2.2. Topologies específiques per entorns HPC

Centrant-nos ja en topologies específiques per entorns HPC, actualment ens trobem amb dos tipus d'estratègies per la interconnexió d'un gran nombre de nodes computacionals, cadascuna amb les seves avantatges e inconvenients.

Fat Tree

Aquesta topologia va ser desenvolupada des del MIT i és àmpliament utilitzada al món de les comunicacions, no solament amb sistemes HPC. És una evolució de la topologia en arbre, fent més eficient aquesta.

Es fa una distribució en forma d'arbre, on els nodes computacionals queden connectats al nivell més baix, fent servir únicament un enllaç. A mesura que pugem per l'arbre, els nombre d'enllaços d'interconnexió que queden al nivell inferior és el mateix que el nombre per connectar amb el nivell superior

Per tant, a mesura que pugem de nivell, tenim més requisits en ample de banda, per poder satisfer aquesta regla. L'arrel de l'arbre serà l'element del disseny amb el major nombre de connexions i, per tant, d'ample de banda a gestionar.

Aquesta topologia és força utilitzada i la ideal per sistemes tipus '*Networks-on-Chip*', però pressenta certa limitacions quan volem fer un desplegament d'un nombre elevat de nodes.

L'escenari ideal és mantindre aquesta relació d'enllaços 1:1 entre nivells, però la realitat sol ser força diferent i és molt habitual tindre escenaris on la relació mantinguda passa a ser 1:2 o superior, cosa que fa aprofitar més els elements de xarxa existents i abaratir costos però penalitza a nivell de rendiment, sobretot quan la comunicació és entre nodes situats a diferent banda del node arrel.

Aquesta topologia es cataloga segons el nombre de nivells que tenim a l'arbre, sent molt habitual trobar-nos amb un escenari de dos nivells, amb un o varis nodes centrals (que fan la tasca d'arrel de l'arbre) i varis *switches* a segon nivell, encarregats de connectar els nodes de nivell inferior (servidors) amb els nodes centrals.

Per aquest tipus d'escenari, el nombre màxim de nodes que podem connectar segueix la següent fórmula matemàtica:

$$Nmax = (Pe \cdot Pc)/2$$

Sent *Pe* el nombre de ports disponibles al nivell 1 i *Pc* al nivell 2, respectivament.

Per tant, i per a una configuració de dos nivells, si fem servir enllaços (*switches*) amb 36 ports, tenim que el màxim nombre de nodes que podem connectar serien 648 ((36*36)/2)

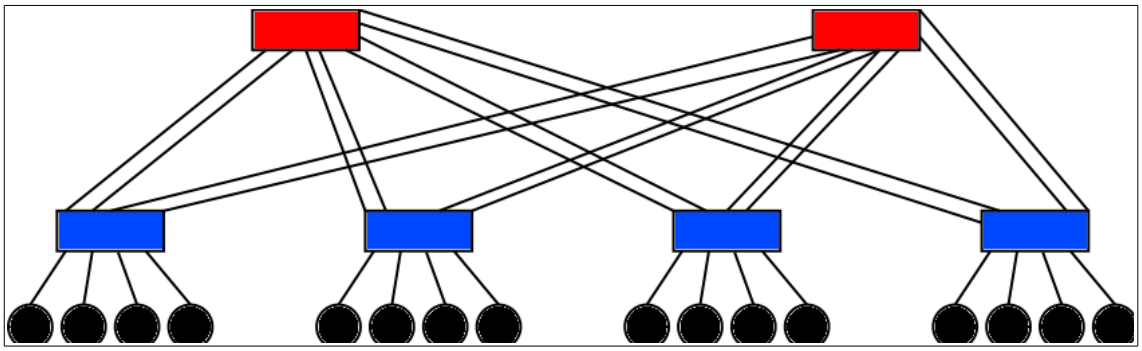


Figura 2. Topologia Fat Tree de dos nivells

A la figura anterior podem veure com es manté la relació d'enllaços entre nivells: quatre nodes al nivell més inferior (color negre) connecten amb els *switches* de segon nivell (color blau) i, aquestos fan servir el mateix nombre d'enllaços per connectar amb les dues arrel (color vermell).

Aquest escenari ideal, on es té una relació 1:1, suposa que la xarxa està totalment equilibrada a nivell d'enllaços i, per tant, no es produeixen bloquejos a nivell de xarxa. Seria similar a tindre una xarxa de commutació de circuits.

La realitat és ben diferent, i mantindre aquesta relació ideal és costós, cosa que fa que s'opti per no mantindre la relació 1:1 entre nivells, passant a un escenari similar a les xarxes de commutació de paquets, on apareix el concepte de bloqueig.

El bloqueig queda produït quan ens trobem que el nombre d'enllaços del nivell inferior és superior als que tenim al següent nivell, atès que s'han de compartir els enllaços i, per tant, és necessari establir un sistema bloquejant per tal que es pugui produir aquesta compartició del medi.

A mesura que la relació augmenta, i ens allunyem del la relació ideal 1:1, el bloqueig augmenta, cosa que pot provocar una pèrdua de rendiment molt significativa al sistema. Per tant, és important tindre en compte aquest factor i saber treballar amb ell, per tal de minimitzar-lo, en la mesura que sigui possible.

Cal destacar també el concepte de latència amb el problema dels bloquejos, ja que si un paquet queda bloquejat a l'espera de poder ser enviat, tindre una latència reduïda ens assegura que el temps de bloqueig serà menor. D'aquí que tindre latències reduïdes és vital en sistemes HPC, on el rendiment final obtingut és clau.

Un altre concepte que és important tindre present és la redundància i resiliència a fallides. Si bé tindre tot el sistema redundat a nivell d'enllaços ens garanteix alta disponibilitat, també perdre algun dels enllaços penalitza el rendiment, donat que tot el tràfic que aquest enllaç suporta ha de ser traspassat als enllaços restants i, per tant, augmenten la congestió de xarxa i els bloquejos.

En cas que es tingui un sistema amb poca càrrega a nivell de xarxa, perdre un enllaç no significa cap problema important i, segurament, no es notaria a nivell de rendiment, però amb un escenari HPC, on l'alt rendiment i la càrrega són vitals, perdre un enllaç si que suposa una important penalització de rendiment i s'ha de minimitzar al màxim, sempre que es permeti.

Torus

Aquesta topologia de xarxa suposa un canvi radical de concepte amb les aproximacions vistes anteriorment i és freqüentment utilitzada en sistemes HPC de grans dimensions, on suposa un benefici important a nivell de cost.

Oblidem el concepte de elements de xarxa que interconnecten els diferents nodes e imaginem un nou escenari, on els propis nodes queden connectats directament entre ells. Semblaria una aproximació similar al concepte de malla, vist anteriorment.

S'ha d'imaginar aquesta topologia com una variació de la topologia clàssica en malla, però una malla connectada parcialment i no totalment, formant una estructura de vàries dimensions on els tots el nodes connecten directament amb els veïns més propers.

Donat que ara treballem directament amb nodes i eliminem el concepte d'encaminament amb elements externs, apareix el concepte de dimensió, segons la disposició dels nodes i les connexions entre ells.

Les dues disposicions bàsiques i més habituals són xarxes *Torus* Bidimensionals (2D) i Tridimensionals (3D). Encara que existeixen disposicions que augmenten aquest nombre de dimensions, fins 6D, veure les dues primeres ens donarà una visió més concreta del concepte.

La topologia *Torus* de dues dimensions (2D) disposa els nodes computacionals sobre un taulell, a mode de malla, i on cada element connecta directament amb els veïns. Per tant, tenim que cada node té quatre connexions directes, amb els veïns que té a la seva part superior, inferior, dreta i esquerra. Els nodes situats a l'extrem del taulell connectaran amb l'altre extrem del taulell, a mode de tancament, cosa que farà que tot node tingui aquestes quatre connexions.

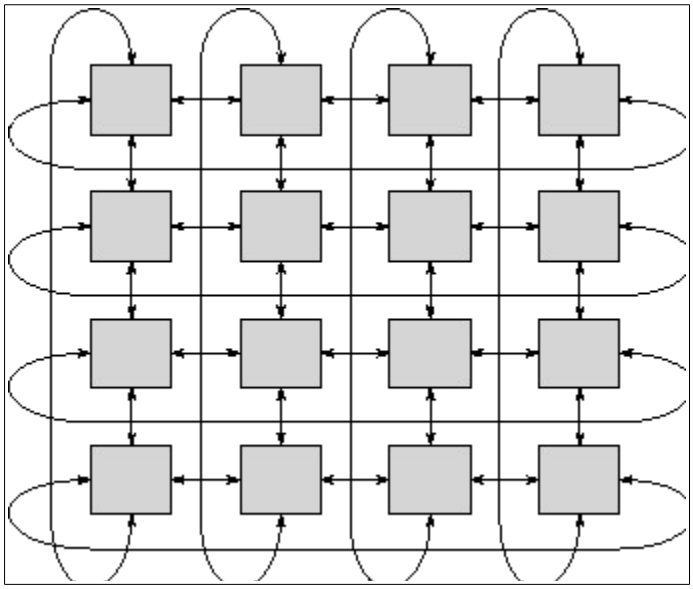


Figura 3. Estructura Torus de dues dimensions (4x4)

La topologia Torus 3D agafa el concepte anterior i l'expandeix, afegint una dimensió addicional, que seria com donar-li profunditat al nostre taulell anterior, per tal de formar un element amb forma de cub, on cada node connecta de la mateixa manera amb els seus veïns, fent servir en aquest sis connexions.

De forma similar, els nodes que quedarien situats a les cares del cub, fan el tancament connectant amb el node homònim a la cara contraria de l'estructura. La següent figura renderitzada mostra molt clarament com s'ha d'imaginar l'estructura que formen els nodes.

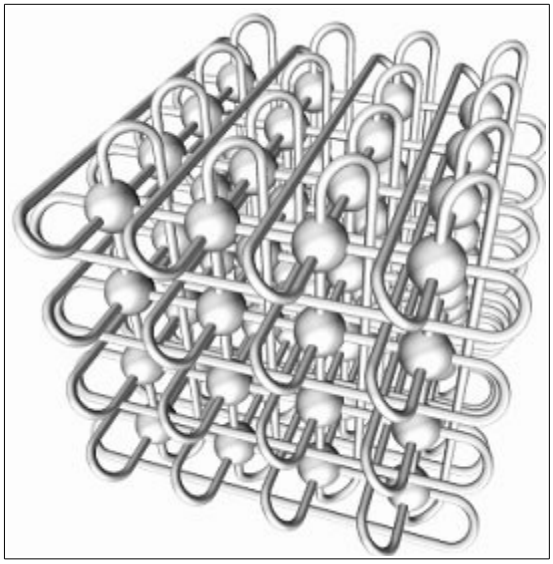


Figura 4. Estructura Torus de tres dimensions (4x4x4)

L'idea d'augmentar la xarxa Torus de dimensions és mantindre una latència reduïda, ja que moure's entre els diferents nodes de la xarxa implica un retard i aconseguir passar pel menor nombre de nodes fins arribar d'un extrem a l'altra suposarà un millor rendiment.

Per exemple, per una xarxa Torus 2D d'un miler de nodes (32x32) es necessitarien 32 salts per arribar al centre de la malla. En canvi, per una xarxa Torus amb el mateix nombre de nodes (10x10x10) només caldrien 15 salts, es a dir, menys de la meitat i, per tant, una latència molt inferior.

El fet que els nodes computacionals de la xarxa connectin directament entre ells, afegeix beneficis pel fet d'eliminar els elements de xarxa addicionals (*switches*) però implica tindre altre tipus en compte altres consideracions.

Els beneficies més immediats que obtenim és una reducció de la latència i, per tant, millor rendiment a nivell de xarxa, sobretot si fem servir un determinat tipus d'algoritmes de processament paral·lel on la comunicació és principalment amb els nodes veïns. A més a més, tenim una reducció del consum elèctric i menor càrrega calorífica.

Per contra, no tot són avantatges, atès que necessitem donar a cada node de processament un major nombre de connexions, cosa que es tradueix en tindre un major cost a nivell de cablejat i d'unes targetes de xarxa específiques, per tal de poder satisfer aquest major nombre de cablejat.

A més a més, el fet de tindre un major nombre de cables per cada node fa que la infraestructura sigui més complexa, afegint cablejat que cal tindre controlat i ben organitzat en cas de fallida. Si s'opta per cablejat de coure, el volum que aquest ocupa és major però amb un menor cost. Per contra, si volem reduir el volum d'aquest es pot optar per cablejat òptic (fibra òptica) amb un conseqüent augment de cost.

Mirant sobre les possibilitats d'escalar la infraestructura que tenim, la topologia *Fat Tree* no presenta cap problema, sempre i quan estigui ben dissenyat. Per contra, l'aproximació més senzilla per escalar aquesta seria augmentar una de les dimensions del cub. Per exemple, en una xarxa Torus 3D de 64 nodes (4x4x4) passar a 5x4x4 o superior.

Aquesta solució, encara que vàlida, suposaria trencar amb l'equilibri de l'estructura i, per tant, encara que s'augmentés el nombre de nodes, hi hauria penalitzacions a nivell de latència que s'haurien de tindre en compte.

3. Model computacional

3.1. Modelatge dels sistemes base

3.1.1. Elements bàsics

El modelatge del sistema HPC es realitza fent servir arxius tipus XML, on es defineix tota l'estructura del sistema que volem avaluar. Aquesta és la metodologia que fa servir el programari *SimGrid*.

Dins del fitxer de modelatge, es creen els diferents elements que composen el sistema, com són els nodes computacionals, els '*switches*' per interconnectar-los i els diferents enllaços per crear aquestes connexions.

Encara que n'hi ha d'altres opcions possibles però intranscendents per aquest projecte, als nodes computacionals del nostre model s'han definit les següents característiques:

1. Nom del node
2. Potència de càlcul, mesurada en FLOPS.
3. Nombre de nuclis o processadors al node.
4. Percentatge de disponibilitat per fer càlculs.

Exemple de definició d'un node en *SimGrid*:

```
<host id="node00" power="150Gf" core="1" availability="95"/>
```

Les característiques d'aquest processador són les següents:

Processador	
Fabricant	Intel
Model	Xeon E5-2687W
Data llançament	2012
Cache	20 MB
Velocitat Intel QPI	8 GT/s
Enllaços QPI	2
Conjunt instruccions	64 bits
Litografia	32 nm
Temperatura max.	67º
Voltatge	0.60V – 1.35V
Nª nuclis	8
Nº fils	16
Velocitat rellotge	3.1 GHz
Memòria max.	256 GB
Tipus memòria	DDR3
Ample banda max.	51.2 GB/s
Revisió PCI-X	3.0
Nª max. busos	40
Nº max CPU's	20 MB
GFLOPs	150

Els nodes generen una potència de càlcul de 150 GFLOPS, que és la que proporciona el processador anterior. Encara que el processador té un total de 8 nuclis, a la definició d'aquest a *SimGrid*, s'indica que només hi tenim un processador, ja que la potència de càlcul és total, a nivell de processadors i no de nucli.

Els '*switches*' que interconnecten els nodes no tenen cap definició especial, ja que només cal identificar-los. No es defineix cap característica addicional.

Exemple de definició d'un '*switch*':

```
<router id="router00"/>
```

Per cada connexió física que hi hagi al sistema HPC, s'ha de definir aquest enllaç de forma explícita i, a més a més, donar-li les característiques que li pertoquin.

Les característiques que es defineixen als enllaços de xarxa venen donats per la tecnologia que tingui cadascun, sent necessari donar-li una velocitat de transmissió i una latència. La velocitat s'ha de expressar en Bytes/s, per tant, es necessari transformar els valors convencionals (bits/s) a aquest format. Per contra, la latència s'expressa seguint les unitats estàndards de µs.

Exemple de definició d'un enllaç de comunicacions:

```
<link id="link000" bandwidth="14000MBps" latency="0.5us"/>
```

Encara que menys necessaris per al nostre cas, també s'han de definir les connexions tipus '*localhost*' dels nodes, es a dir, l'enllaç de comunicació amb ell mateix. Per aquestes connexions, i atès que es fan a nivell local, s'ha optat per donar-li un valor molt alt en velocitat de transmissió i molt reduït en latència.

Exemple de definició de la connexió entre ell mateix (*localhost*):

```
<link id="loopback" bandwidth="100GBps" latency="0.001us"
                sharing_policy="FATPIPE"/>
```

3.1.2. Enllaços d'interconnexió

Per als nostres sistema base s'han escollit les següent tecnologies a nivell d'enllaç:

- Gigabit Ethernet
- 10 Gigabit Ethernet
- InfiniBand SDR
- InfiniBand DDR
- InfiniBand QDR
- InfiniBand FDR
- InfiniBand EDR

Cadascuna d'aquestes tecnologies té les seves pròpies característiques, tant a nivell de velocitat de transmissió com de latència. Els valors escollits han sigut els següents:

	Velocitat transmissió (Mbits/s)	Velocitat transmissió (Mbytes/s)	Latència (µs)
Gigabit Ethernet	1000	125	4.25
10 Gigabit Ethernet	10000	1250	2.25
InfiniBand SDR	8000	1000	5
InfiniBand DDR	16000	2000	2.5
InfiniBand QDR	32000	4000	1.3
InfiniBand FDR	56000	7000	0.7
InfiniBand EDR	100000	14000	0.5

S'ha afegit la columna de velocitat de transmissió en les unitats de bytes/s, ja que és aquest format el que fa servir el programari *SimGrid* per la definició dels enllaços.

3.1.3. Topologies

S'han definit tres topologies per al modelatge del sistema base. Són les següents:

- Fat Tree
- 2D Torus
- 3D Torus

Totes tres topologies tenen en comú el nombre de nodes computacionals que formen el sistema HPC. S'ha definit sempre el sistema fent servir 64 nodes, que queden interconnectats de forma diferent per cada topologia.

Per a la topologia Fat Tree, els nodes s'agrupen en quatre grups, amb 16 nodes per cadascun d'ells. Els nodes de cada grup connecten directament al mateix switch, anomenat '*Edge Switch*'.

Els quatre switches tipus '*Edge*' connecten amb un switch central que els connecta entre ells i que hem anomenat '*Core Switch*'.

Els nodes sempre connecten amb el seu switch '*Edge*' amb un enllaç que té les característiques marcades per la tecnologia que faci servir. Per contra, els quatre enllaços entre els switches '*Edge*' i el '*Core*' tenen unes característiques diferents, segons la tecnologia que fem servir.

Per la tecnologia Ethernet (Gigabit i 10 Gigabit) es fa una relació tipus 1:1 entre nivells, cosa que fa que aquest enllaç entre switches sigui una suma equivalent a la suma dels 16 nodes que té connectat el switch amb la mateixa latència que marqui la tecnologia.

	Velocitat transmissió (Mbits/s)	**Velocitat transmissió (Mbytes/s)**	**Latència (µs)**
Gigabit Ethernet	16000	2000	4.25
10 Gigabit Ethernet	160000	20000	2.25

Amb el protocol InfiniBand s'ha fet servir una aproximació diferent. Donat que el mateix protocol defineix dues velocitats de transmissió (4x i 16x) segons si són connexió per a nodes o entre switches. A nivell de nodes, s'han definit connexions tipus 4x i s'ha deixat el tipus 16x per a la interconnexió entre switches '*Edge*' i el switch '*Core'*.

	Velocitat transmissió (Mbits/s)	**Velocitat transmissió (Mbytes/s)**	**Latència (µs)**
InfiniBand SDR – 12x	24000	3000	5
InfiniBand DDR – 12x	48000	6000	2.5
InfiniBand QDR – 12x	96000	12000	1.3
InfiniBand FDR – 12x	168000	21000	0.7
InfiniBand EDR – 12x	504000	42000	0.5

Les topologies Torus tenen un enfocament totalment diferent, atès que desapareix el concepte de switch i els nodes connecten directament entre ells.

Donat que la definició de la topologia amb l'eina *SimGrid* no permet connectar nodes directament entre ells, s'ha optat per definir un switch per cada node i serà aquest element el que s'encarregarà de connectar amb els altres nodes.

Per la connexió 'local' entre node i el seu propi switch s'ha definit una connexió de molt alta velocitat, amb les següent característiques:

```
<link id="local00" bandwidth="10000GBps" latency="0.000001us"/>
```

Una vegada tenim definits els switches per cada node, hem de definir els enllaços entre ells i veure com s'interconnecten aquests.

La definició dels enllaços és de la mateixa forma que amb la topologia Fat Tree. Per cada enllaç s'ha de definir tant la velocitat de transmissió com la latència d'aquest. Totes dues característiques venen donades per la tecnologia que fem servir.

	Velocitat transmissió (Mbits/s)	Velocitat transmissió (Mbytes/s)	Latència (µs)
Gigabit Ethernet	1000	125	4.25
10 Gigabit Ethernet	10000	1250	2.25
InfiniBand SDR	8000	1000	5
InfiniBand DDR	16000	2000	2.5
InfiniBand QDR	32000	4000	1.3
InfiniBand FDR	56000	7000	0.7
InfiniBand EDR	100000	14000	0.5

Es pot apreciar com, per la tecnologia InfiniBand, hem fet servir els valors que dóna la interconnexió tipus 4x, donat que aquesta és la utilitzada habitualment per la connexió de nodes computacionals.

Referent a la topologia o manera d'interconnectar els nodes entre ells, tenim dues aproximacions fetes: 2D Torus i 3D Torus. Veiem com queden distribuïts segons la topologia utilitzada.

En la topologia 2D Torus tenim 64 nodes (igual que amb Fat Tree) però distribuïts en forma de malla de dues dimensions, amb unes mides de 8 x 8. D'aquesta manera tenim una malla totalment equilibrada. Els nodes situats a l'extrem de la malla també connecten amb el seu homònim a l'altra part de la malla, per tal de fer el tancament d'aquesta.

La topologia 3D Torus connecta els 64 nodes a mode de cub tridimensional 4x4x4, es a dir, amb quatre nivells de profunditat i totalment interconnectat, fent que tots els nodes situats a les cares del cub connectin amb el seu homònim, per tal de fer el tancament.

Amb la topologia 2D Torus tenim un total de 128 connexions entre nodes. Per contra, amb 3D Torus, tenim 176 connexions.

3.2. Programari MPI

Una de les característiques que té el programari *SimGrid* és la possibilitat de simular programari tipus MPI per a un sistema HPC definit prèviament, per tal d'avaluar rendiments i temps d'execució, simulant com es comportaria un sistema HPC d'una determinades característiques.

Una vegada tenim definit el nostre sistema d'alt rendiment, com ja hem vist anteriorment, toca escollir el programari que farem servir durant les simulacions.

MPI són una sèrie de directives i llibreries per al llenguatge C per tal de treballar amb sistemes distribuïts de memòria distribuïda, es a dir, nodes computacionals connectats entre ells per tal de distribuir entre ells treballs complexos i poder treballar de forma paral·lela.

Al nostre cas, una tasca complexa es divideix entre 64 nodes i s'avalua en temps d'execució per completar aquesta. D'aquesta manera es pot avaluar el rendiment que donaria un determinat sistema HPC. El sistema HPC a avaluar queda prèviament definit en un arxiu tipus XML, tal i com ja s'ha vist anteriorment.

Les aplicacions MPI escollides per fer les simulacions han sigut les següents:

- *Latency*
  S'envia un missatge d'un byte de mida des del node 0 a tota la resta de nodes i s'espera la resposta d'un altre missatge d'un byte per part de cada node. La tasca es repeteix 1.000 i 100.000 vegades.
  Aquesta aplicació avalua el rendiment de xarxa entre el node 0 i tota la resta, però no té en compte el rendiment computacionals dels nodes.

- *Array Assignment*
  Un vector de 960.000.000 s'inicialitza pel node 0 i es descompon en parts iguals i s'envia a la resta de nodes, per tal que cadascun d'ells faci la suma dels elements de la part del vector rebuda. Una vegada computada, s'envia de tornada al node 0, que recopila totes les dades i genera la suma de tots els elements del vector.
  Aquesta aplicació avalua tant el rendiment de xarxa entre el node 0 i la resta com el rendiment de càlcul de cada node, atès que cada node ha de computar el valor que li correspon i enviar-lo al node 0.

- *Matrix Multiply*
  La tasca de multiplicar la matriu A de 248x30 elements per la matriu B de 30x14. El node 0 reparteix les tasques entre la resta de nodes i n'espera el resultat. La tasca es repeteix 1.000 i 100.000 vegades.
  Aquesta aplicació, de forma similar a l'anterior, avalua tant el rendiment de la topologia de xarxa com la potència de càlcul dels diferents nodes.

- *Prime Generator*
  Generador de números primers, dividint la tasca entre els diferents nodes. Es generen 1.000.000 i 100.000.000 números primers.
  Aquesta aplicació avalua tant el rendiment de la topologia de xarxa com del rendiment de càlcul dels diferents nodes.

- *Wave Equation*
  Resolució de l'equació d'ona, que descriu la propagació d'ones, ja sigui de so, llum o l'aigua.

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}$$

Figura 5. Equació d'ona

L'aplicació de latència és l'única de les escollides que només avalua el rendiment de xarxa amb cada topologia, sense tindre en compte el rendiment de càlcul dels nodes. L'elecció d'aquesta es deu precisament a aquest fet: avaluar el rendiment de cada topologia de xarxa amb un programari molt senzill.

Per contra, la resta d'aplicacions MPI escollides si fan un ús més intensiu de cada node, donat que cadascun d'ells rep una porció de la tasca complexa a completar, computa la seva part i n'envia la solució.

S'han escollit diferents tasques complexes amb conceptes clàssics de la programació paral·lela, atès que és la millor manera de veure com es comporta cada programari paral·lel en diferents entorns.

El fet que algunes de les aplicacions es repeteixin durant certs cicles es deu al fet d'incrementar la complexitat d'aquestes i veure com escalen.

3.3. Plataforma

Tot el conjunt de proves i simulacions fetes durant el desenvolupament d'aquest treball s'han fet sobre una mateixa màquina, sempre fent servir el mateix conjunt de maquinari i programari.

Les dades del maquinari utilitzat són les següents:

Maquinari	
Fabricant	IBM
Model	X3550
Tipus/Model	7978-C3G
Processador	Intel Xeon X5355
Velocitat rellotge	2.66 GHz.
Memòria cau L2	8 MB.
Velocitat bus	1333 MHz.
Nº nuclis / processador	4
Nº processadors	2
Nº total nuclis	8
Hyper-Threading	No
Memòria RAM	4 GB.
Tipus memòria	DDR2
Velocitat memòria	PC2 – 5300
Disc intern	80 GB.
Tipus connexió	SAS
Redundància	RAID1
Nº total discs	2

Dades del programari fet servir:

Programari	
Sistema Operatiu	Linux
Detall	Debian
Versió	7.9
Kernel	3.2
SimGrid version	3.7.1

Donat que treballem amb simulacions, el resultat d'aquestes sempre són les mateixes, de forma independent a la plataforma que es faci servir. Per tant, la diferència de maquinari no afecta als resultats finals obtinguts, però el temps en arribar a aquest si dista molt segons la màquina feta servir.

Aquest fet s'ha vist reflectit de forma molt notable quan s'ha volgut comparar l'execució de probes sobre una Raspberry Pi. El temps necessari per completar les simulacions és molt superior, encara que el resultats finals obtinguts són els mateixos.

3.4. Exemple d'ús

El codi MPI, abans de ser utilitzat pel programari *SimGrid* ha de ser compilat per aquest, fent servir la pròpia eina que proporciona el programari per aquesta tasca.

Exemple de compilació de codi MPI amb *SimGrid*:

```
$ smpicc -o executable codi_font.c
```

Una vegada tenim compilat el nostre programari MPI, podem fer-lo servir amb el simulador, indicant-li a aquest una sèrie de paràmetres, com són el nombre de processos concurrents, llistat dels nodes disponibles i el fitxer amb la configuració del sistema HPC a avaluar.

Exemple d'execució del programari MPI amb *SimGrid*:

```
$ smpirun -np 64 -platform HPC_system.xml -hostfile hosts ./executable
```

Cada aplicació MPI, una vegada finalitzada l'execució, mostra per pantalla el temps d'execució.

```
Elapsed wallclock time was 233724 seconds
```

4. Avaluació del sistema base

4.1. Test de latència

4.1.1. Temps d'execució

Test de latència entre el primer node (node 0) i la resta de nodes de la xarxa, enviant un byte de dades a cada node i rebent les mateixes dades de cadascun d'ells.

Taules de temps d'execució (en microsegons (µs)):

Latency - 1.000 reps.			
	Fat Tree	**2D Torus**	**3D Torus**
1G	10227866	2305045	1928302
10G	1846391	1268925	1069473
SDR	3295186	2692826	2249598
DDR	1647593	1398346	1176732
QDR	846361	776995	661755
FDR	464294	466320	404268
EDR	299840	362762	318439

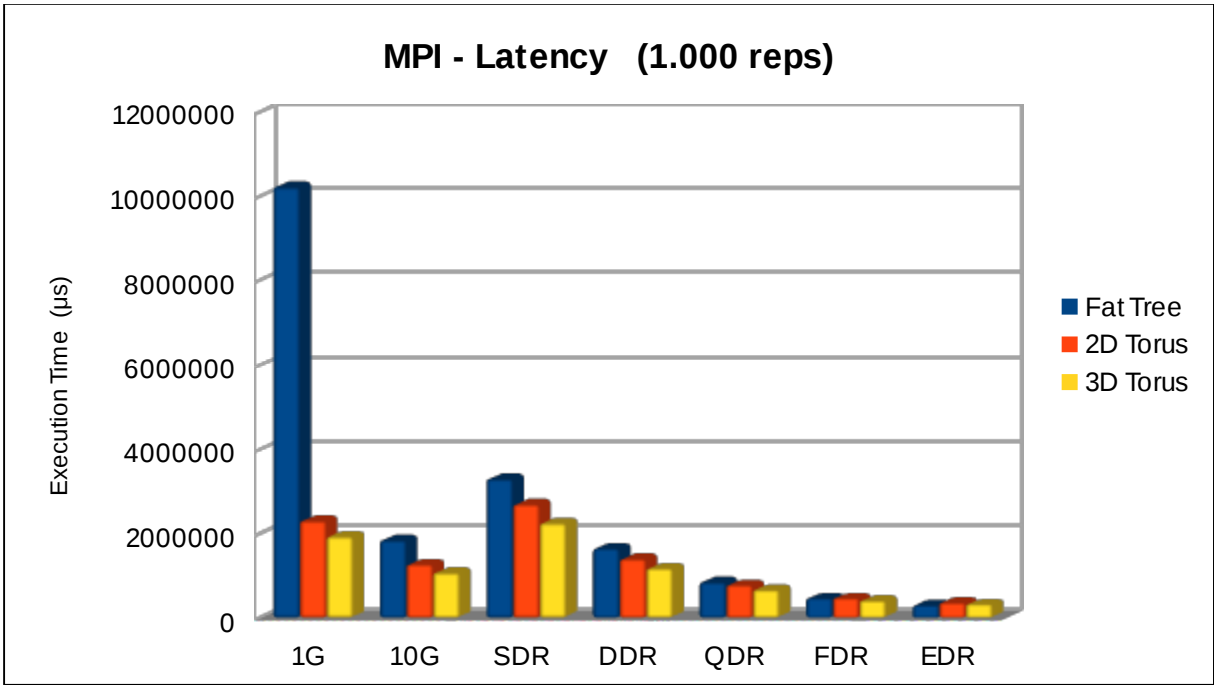
Test de latència – 1.000 repeticions

Latency - 100.000 reps.			
	Fat Tree	**2D Torus**	**3D Torus**
1G	83351184999	1259607026	1221932697
10G	8417478243	1155996232	1135976310
SDR	10620566637	1298384627	1254061887
DDR	5310283334	1168938124	1146776754
QDR	2657398113	1106803724	1095279812
FDR	1516579137	1075736665	1069531481
EDR	765058876	1065380922	1060948648

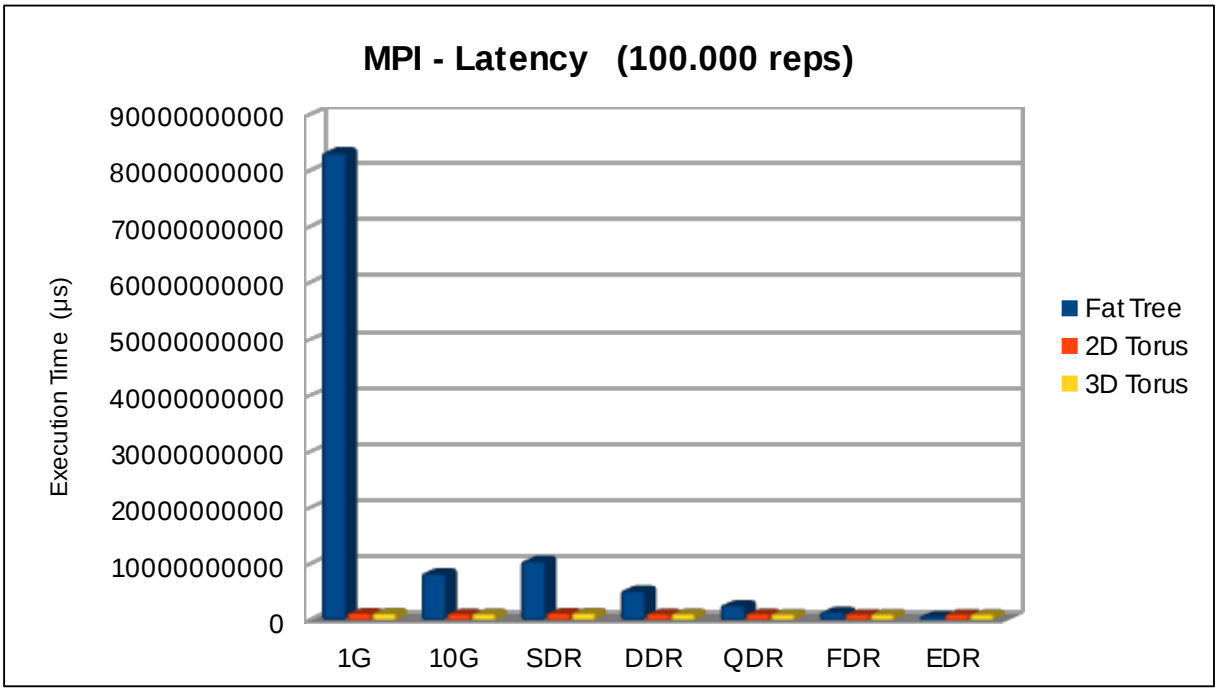
Test de latència – 100.000 repeticions

4.1.2. Gràfiques comparatives

Gràfiques comparatives dels temps d'execució del test de latència per cada topologia i tecnologia d'interconnexió, per 1.000 i 100.000 repeticions.



Test de latència – 1.000 repeticions



Test de latència – 100.000 repeticions

4.2. Array Assignment

4.2.1. Temps d'execució

Descomposició d'un vector gran (960.000.000 elements) en parts iguals i enviament de cada part al node corresponent per al seu processament. Rebem les dades processades de cada node per part del node mestre (node 0).

Taules de temps d'execució (en microsegons (µs)):

Array Assignment - 1 repetició			
	Fat Tree	**2D Torus**	**3D Torus**
1G	1939678887.84444	87764698.9694468	87759590.7780331
10G	193978968.113927	29887042.0170438	29884337.6822189
SDR	242486983.249023	31513602.4727805	31507592.8414294
DDR	121243494.232825	27477804.6528053	27474799.8376446
QDR	60622053.140754	25460255.4921535	25458693.003292
FDR	34640915.4023032	24594991.99722	24594150.656856
EDR	17321370.5551637	24249166.4012273	24248565.4354612

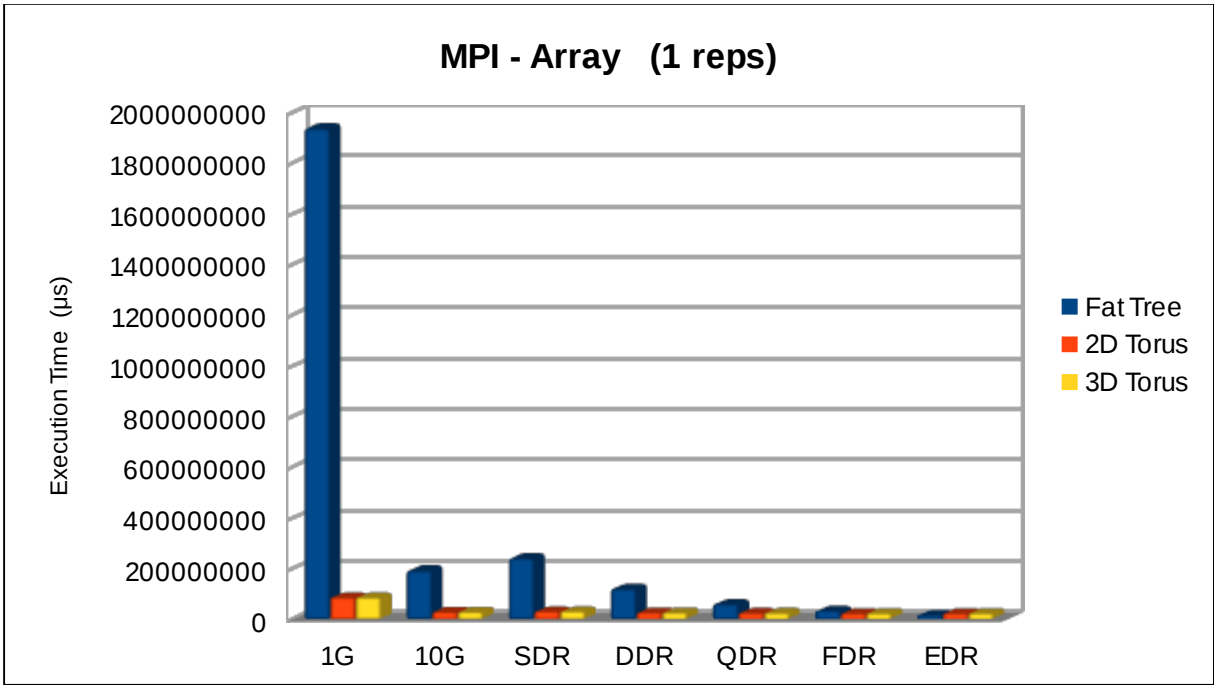
Test Array Assignment – Una repetició

Array Assignment - 10 repeticions			
	Fat Tree	**2D Torus**	**3D Torus**
1G	19396788960.0106	877646967.738956	877595885.82106
10G	1939789668.44986	298870398.131636	298843354.74628
SDR	2424869822.44826	315136002.762289	315075906.380012
DDR	1212434925.64465	274778024.553931	274747976.360595
QDR	606220511.389775	254602532.974807	254586907.902765
FDR	346409132.454265	245949897.964734	245941484.476776
EDR	173213683.100748	242491641.97693	242485632.349031

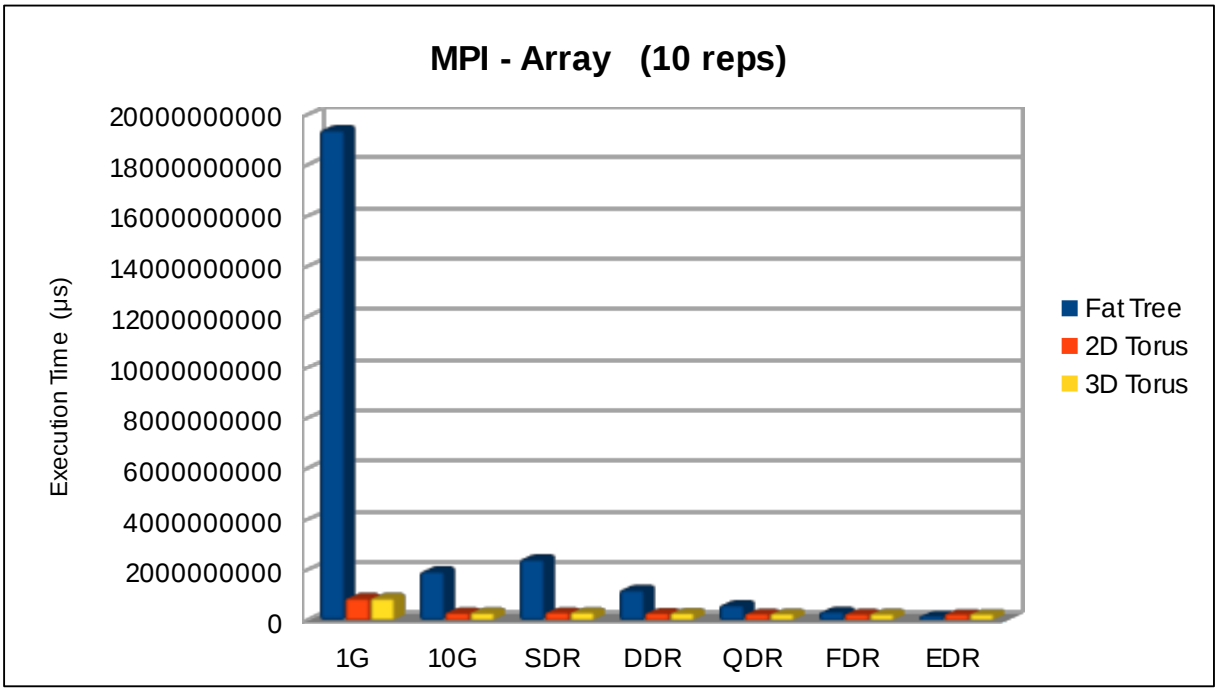
Test Array Assignment – 10 repeticions

4.2.2. Gràfiques comparatives

Gràfiques comparatives pel test de descomposició d'un vector gran, per 1 i 10 repeticions i totes les topologies i tecnologies d'interconnexió.



Test Array Assiignment – Una repetició



Test Array Assignment – 10 repeticions

4.3. Matrix Multiply

4.3.1. Temps d'execució

Multiplicació de dues matrius de mida gran per generar una tercera matriu resultant. La tasca es divideix entre tots els nodes, que computaran la seva part, per després enviar el resultat al node mestre (node 0).

Taules de temps d'execució (en microsegons (µs)):

Matrix Multiply - 1.000.000 repeticions			
	Fat Tree	**2D Torus**	**3D Torus**
1G	4350188215	61891384	61102011
10G	437470199	57847978	57268762
SDR	549776284	62103605	60799214
DDR	274888143	58204660	57550768
QDR	137511242	56332558	55991171
FDR	78520279	55394257	55212248
EDR	39461642	55083197	54953734

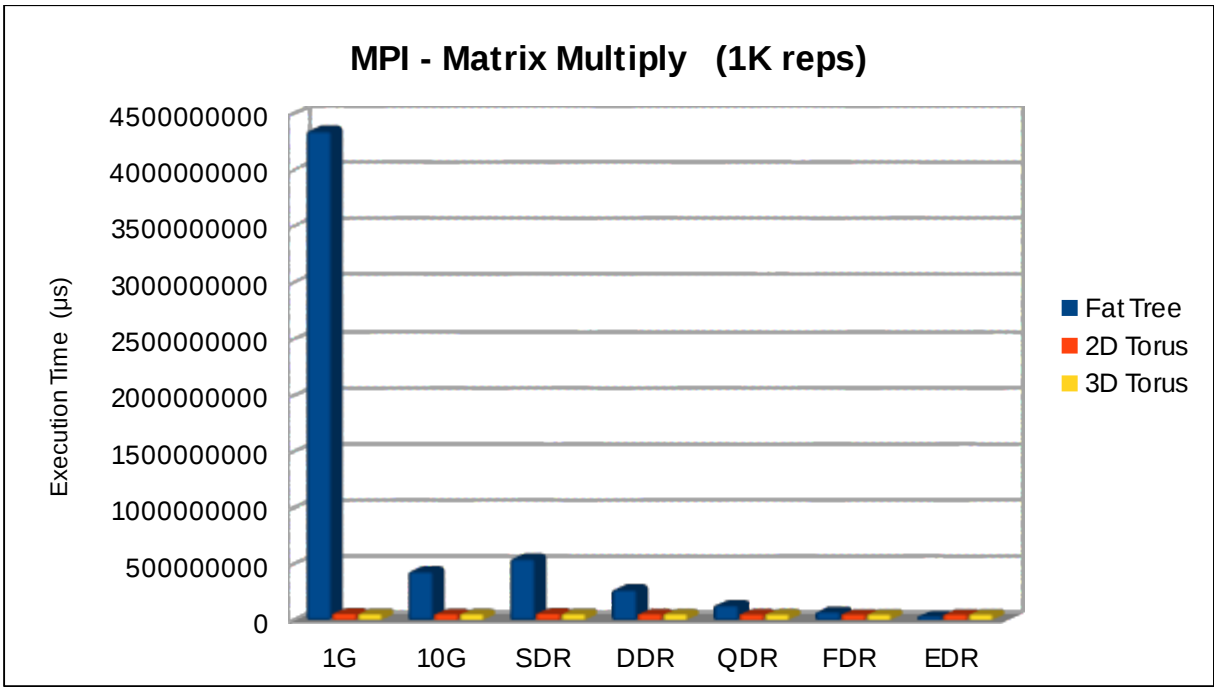
Test Matrix Multiply – 1.000.000 repeticions

Matrix Multiply - 100.000.000 repeticions			
	Fat Tree	**2D Torus**	**3D Torus**
1G	43472932464292	544162746527	544084018353
10G	4347538386080	543758702489	544880711688
SDR	5434716837634	544184265656	544053840856
DDR	2717358418976	543794370787	543728997657
QDR	1358685926606	543607177145	543573038085
FDR	776386200809	543513346322	543495145633
EDR	388213250759	543482239912	543469294122

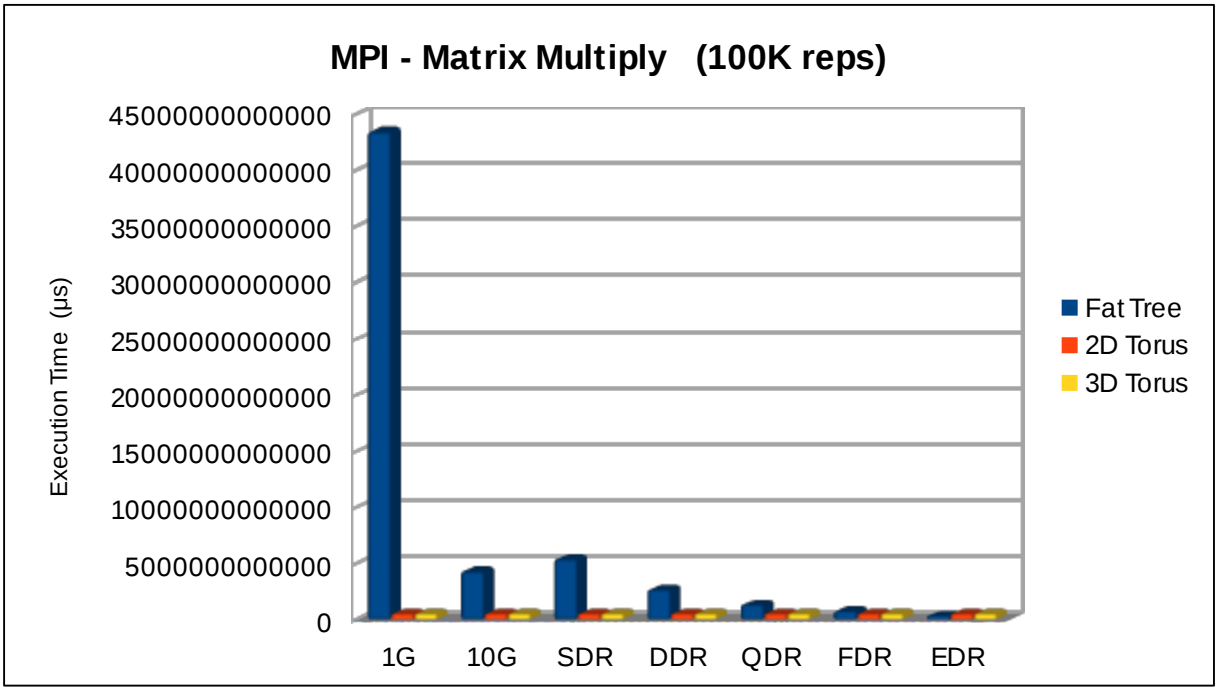
Test Matrix Multiply – 100.000.000 repeticions

4.3.2. Gràfiques comparatives

Gràfiques comparatives pel test de multiplicació de matrius, per 1.000.000 i 100.000.000 repeticions i totes les topologies i tecnologies d'interconnexió.



Test Matrix Multiply – 1.000.000 repeticions



Test Matrix Multiply – 100.000.000 repeticions

4.4. Prime Generator

4.4.1. Temps d'execució

Generador d'un nombre elevat de nombres primers (1.000.000 i 100.000.000, respectivament). La computació es fa de forma col·laborativa entre tots el nodes del sistema.

Taules de temps d'execució (en segons):

Prime Generator - 1.000.000			
	Fat Tree	**2D Torus**	**3D Torus**
1G	716	1370	1028
10G	365	725	544
SDR	809	1611	1209
DDR	404	805	604
QDR	210	419	314
FDR	113	225	169
EDR	80	161	120

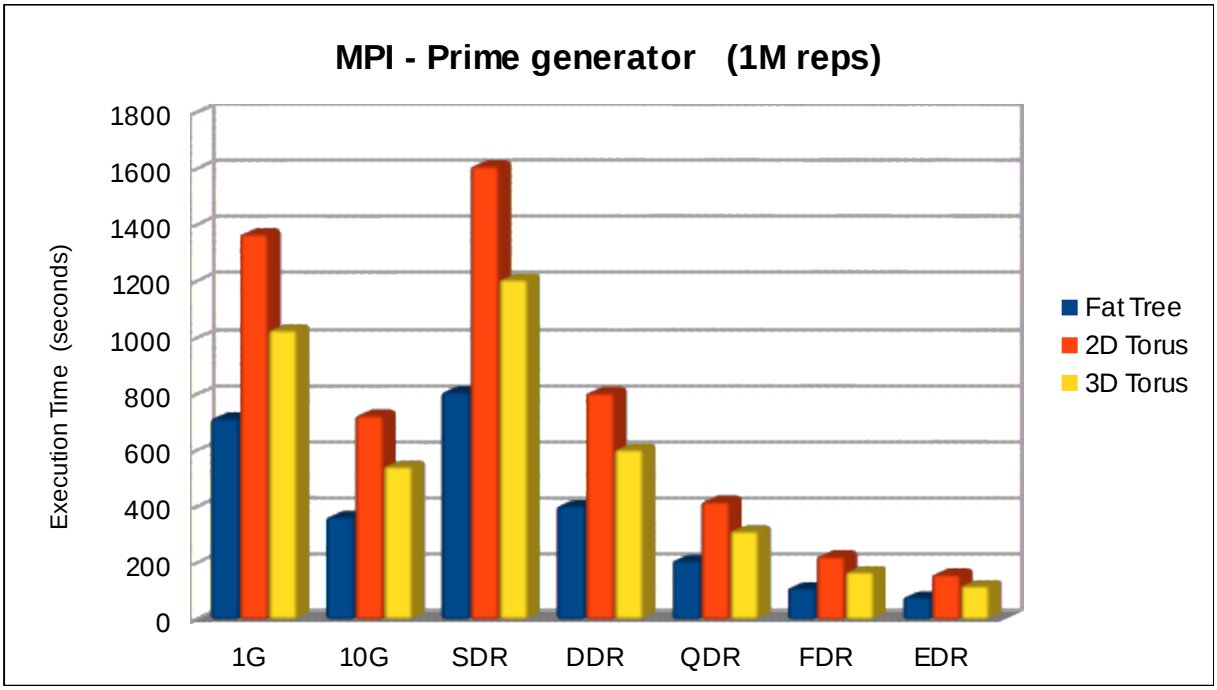
Test Prime Generator – 1.000.000 nombres primers

Prime Generator - 100.000.000			
	Fat Tree	**2D Torus**	**3D Torus**
1G	771	1426	1084
10G	421	780	599
SDR	865	1667	1264
DDR	460	861	660
QDR	266	474	369
FDR	168	281	224
EDR	136	216	176

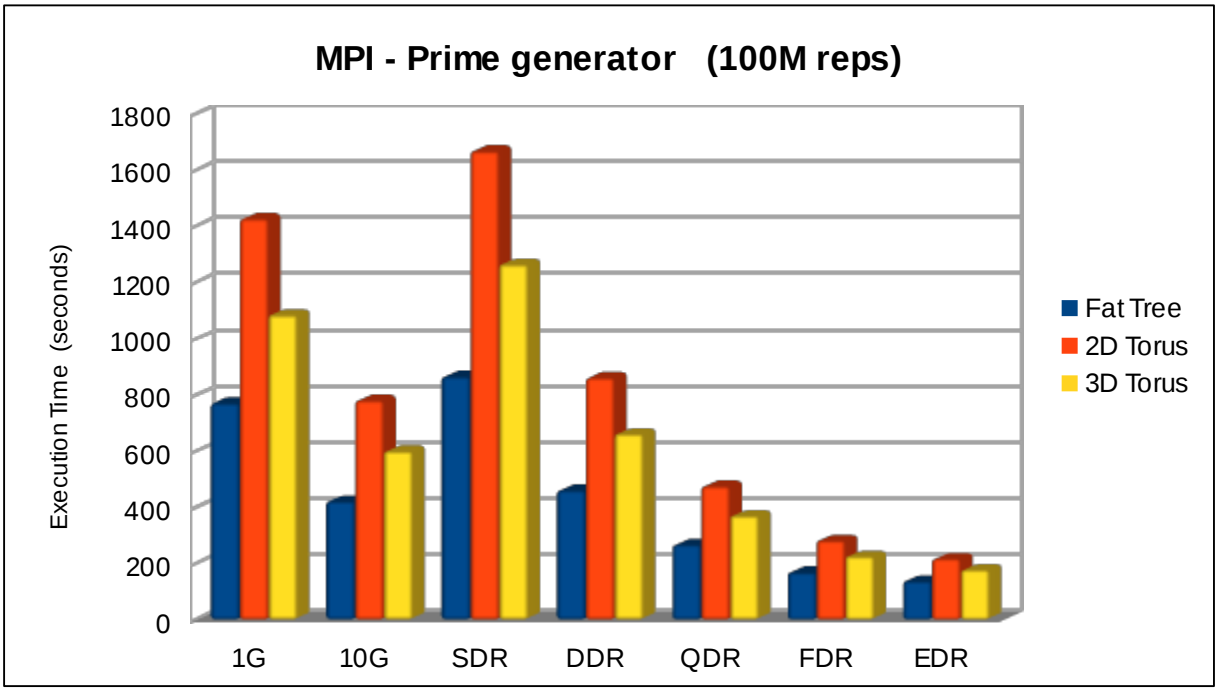
Test Prime Generator – 1.000.000 nombres primers

4.4.2. Gràfiques comparatives

Gràfiques comparatives pel generador de nombres primers i totes les topologies i tecnologies d'interconnexió. Generació de 1.000.000 i 100.000.000 de nombres primers.



Test Prime Generator – 1.000.000 nombres primers



Test Prime Generator – 100.000.000 nombres primers

4.5. Wave Equation

4.5.1. Temps d'execució

Ús de programari paral·lel MPI per resoldre l'equació d'ona, de forma col·laborativa entre tots els nodes del sistema.

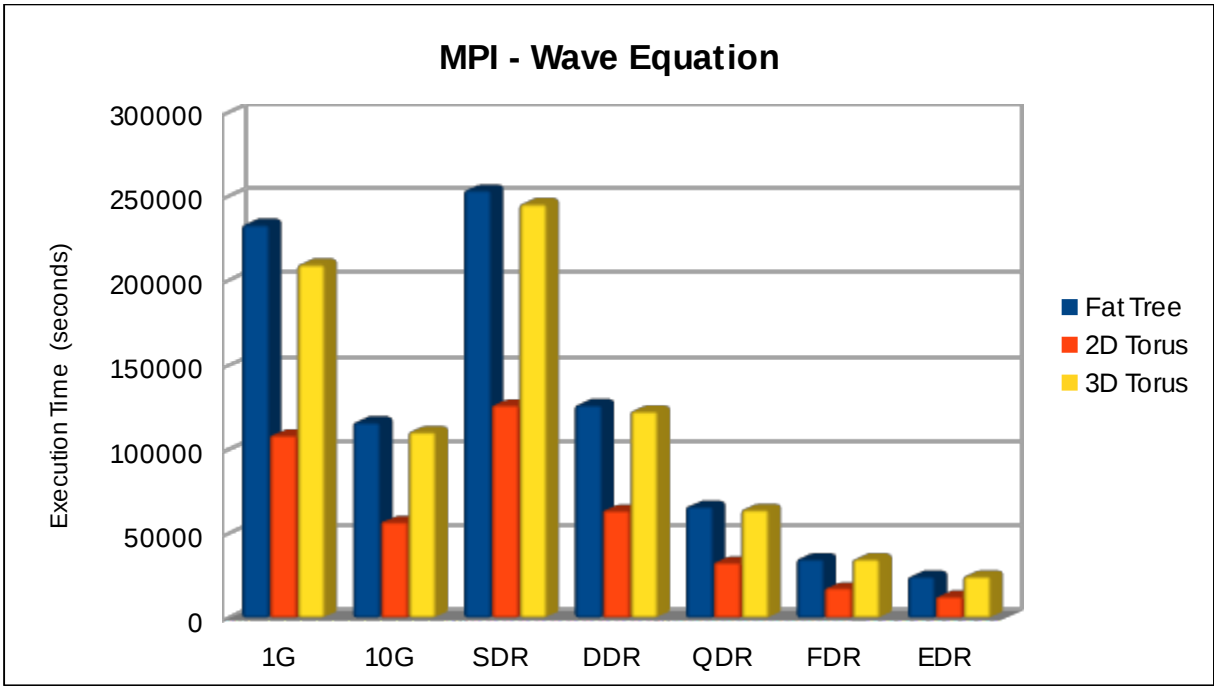
Taules de temps d'execució (en segons):

Wave Equation			
	Fat Tree	**2D Torus**	**3D Torus**
1G	233724	108636	209937
10G	116341	57316	110729
SDR	254068	126556	245875
DDR	126709	64111	122957
QDR	66423	33376	64465
FDR	35328	18026	35065
EDR	24867	12929	25050

Test Wave Equation amb MPI

4.5.2. Gràfiques comparatives

Gràfica comparativa pel càlcul de l'equació d'ona, amb totes les topologies i tecnologies d'interconnexió.



Test Wave Equation amb programari MPI

4.6. Conclusions

El test de latència no dóna resultats que siguin sorprenents. Es por apreciar com a cada evolució tecnològica, tant amb Ethernet com amb InfiniBand el salt qualitatiu en la reducció del temps d'execució és significatiu, especialment amb la tecnologia Ethernet, on el pas d'una velocitat de transmissió de 1Gbps a 10Gbps redueix considerablement el temps necessari per completar el test.

D'igual manera, es pot apreciar com la topologia que millor resultat dóna, de forma general, és 3D Torus. Però, si es mira amb detall, es pot apreciar com a mida que augmentem la velocitat i reduïm la latència (InfiniBand), Fat Tree passa a ser la topologia que proporciona el menor temps d'execució.

Per contra, i mirant el resultats obtinguts amb el test de descomposició del vector (*Array Assignment*), podem apreciar com la topologia Fat Tree és la que dóna el pitjor resultat, ja que les topologies Torus es mantenen força equilibrades amb les diferents tecnologies d'interconnexió.

El resultat per l'execució de la multiplicació de matrius (*Matrix Multiply*) ens aporta un resultats semblants al test anterior, on les topologies Torus es mantenen amb uns resultats molt estables i reduïts, en contraposició amb Fat Tree, que aporta els pitjors resultats de forma generalitzada, donant el millor valor amb la millor tecnologia d'interconnexió (InfiniBand EDR).

Cal destacar, amb els dos tests anterior, el baix rendiment que dóna la tecnologia Gigabit Ethernet, que té el llastre d'una latència massa elevada, en comparació amb la resta de tecnologies d'interconnexió.

El resultats que dóna el generador de nombres primers és força interessant, atès que la topologia Fat Tree es comporta com la millor, superant clarament a 2D Torus, i proporcionant una reducció de temps d'un 50%, en comparació amb aquesta última. També millora en tots els cassos a 3D Torus, inclòs amb Gigabit Ethernet.

En aquest cas, la pròpia natura del test fa que hi hagi molta diferència en els temps d'execució entre topologies, independentment de la tecnologia d'interconnexió existent.

D'igual manera que ha succeït anteriorment, l'execució del test d'equació d'ona aporta uns resultats molt interessant, ja que en aquest cas la millor topologia és per a 2D Torus, que aporta uns resultats un 50% millors que la resta. Per contra, Fat Tree i 3D Torus donen unes xifres similars, per totes les execucions.

En el cas d'aquest darrer test, obtindre un millor rendiment amb la topologia 2D Torus es deu, segurament, al fet que el test treballa molt amb els nodes veïns i és precisament aquest fet el que propicia que aquesta topologia es comporti així de bé i doni els menors temps d'execució.

Remarcar com, segons el tipus de test, la topologia influeix i proporciona uns millors temps d'execució, per a un mateix nombre de nodes computacionals.

D'igual manera, cada evolució en la tecnologia d'interconnexió proporciona millor velocitat de transmissió i menor latència, cosa que repercuteix en el temps d'execució, reduint-se aquest.

5. Millores proposades

5.1. Tecnologies d'interconnexió

5.1.1. 10 Gigabit Ethernet amb connexions SFP+

Al sistema base ja vam treballar amb la tecnologia 10 Gigabit Ethernet, vam modelar-lo i vam poder avaluar-lo, amb les diferents simulacions, per tal de veure el seu rendiment amb les diferents topologies proposades.

En aquest cas mantenim el mateix estàndard amb la mateixa velocitat de transmissió (10 Gbps) però milloren la latència dels enllaços de xarxa amb la introducció de la tecnologia SFP+.

Aquesta tecnologia base el seu funcionament en substituir la clàssica tecnologia basada en cables tipus parell de coure trenat (10GBASE-T) per cablejat de fibra òptica. D'aquesta manera es redueix considerablement la latència, passant d'un temps de 2.25 a 0.3 µs.

Modificarem els valors de latència per al model base, per cadascuna de les topologies proposades. La resta de paràmetres i aspectes topològics no reben cap modificació, atès que només es vol avaluar com afecta al rendiment el canvi de latència en la tecnologia d'interconnexió.

5.1.2. 40 / 100 Gigabit Ethernet

Aquesta modificació del sistema base implica afegir el nou estàndard Ethernet sobre el model base, per tal de de veure quin rendiment dóna per cadascuna de les topologies proposades.

En aquest cas, a més de modificar aspectes com són la velocitat de transmissió i latència, s'han de tindre en compte altres consideracions de tipus topològic, especialment amb la topologia Fat Tree, ja que serà en aquest escenari on es faran servir les dues velocitats de transmissió que proporciona l'estàndard.

Les connexions entre els nodes i els switches tipus '*Edge*' es faran amb connexions de 40 Gbps. Per la connexió entre cada switches *Egde* i *Core* switch farem servir sis enllaços de 100 Gbps agregats, amb una velocitat total de 600 Gbps i una latència de 0,3 µs, per a tots els enllaços.

El fet d'utilizar sis enllaços per les connexions troncals amb el switch central ve motivat pel fet que necessitem aquesta velocitat per tal de tindre equilibrat l'arbre. Recordem que tenim connectats 16 nodes a cada switch *Edge* i que necessitaríem connexions de 640 Gbps entre switches per tal de tindre un equilibri perfecte. (16 nodes x 40 Gbps = 640 Gbps).

A les topologies tipus Torus només s'implementaran amb la velocitat de transmissió de 40 Gbps. El motiu d'aquesta decisió és que en aquesta topologia no fem servir electrònica de xarxa i, per tant, només són connexions entre nodes computacionals, precisament on s'utilitza aquesta velocitat de transmissió.

5.1.3. InfiniBand NDR

Aquesta evolució en l'estàndard no està disponible actualment i es preveu que passi a estar accessible després de l'any 2020, per tant, farem servir els valors teòrics amb els quals s'està treballant durant el seu desenvolupament i posterior estandardització.

Com cada evolució en el protocol, es millora tant la velocitat de transmissió com la latència, passant a proporcionar uns valor de 180 Gbps i 0.35 µs, respectivament.

Aquesta velocitat de transmissió correspon per agregacions 4x, es a dir, les que fem servir per connectar els nodes, ja sigui amb els switches o directament entre ells. En el cas de la topologia Fat Tree, farem servir les agregacions 12x per la interconnexió entre els switches *Edge* amb el *Core*.

5.2. Topologies de xarxa

5.2.1. Fat Tree 1:1 amb InfiniBand

Per tal d'oferir el millor rendiment possible, aquesta topologia ha d'estar el més equilibrada possible, es a dir, seguir el patró 1:1 entre nivells o bé aproximant-se a ell el màxim possible.

Per les tecnologies *Ethernet* si que s'ha seguit aquest patró i s'ha simulat sempre amb aquesta relació 1:1 entre nivells. Per cada 16 nodes que connecten contra un switch *Edge*, sortien 16 connexions del mateix tipus cap al switch *Core*.

Per contra, quan hem treballat amb la tecnologia *InfiniBand* no s'ha seguit aquest patró, ja que s'ha treballat amb agregacions tipus 4x per a les connexions dels nodes amb els switches Edge i només una agregació tipus 12x per a connectar els switches Edge amb el Core. Aquesta elecció ens dóna un arbre força desequilibrat, amb una relació 1:5 entre els nivells.

Millorarem la relació entre nivells augmentant a 5 el nombre d'agregacions tipus 12x entre switches Edge i Core, cosa que ens equilibra l'arbre en una relació quasi de 1:1.

5.2.2. 2D Torus

Partim d'aquesta topologia clàssica en entorns d'alt rendiment i mirem una possible millora a nivell de topologia per tal de veure l'augment de rendiment que obtenim amb les execucions MPI proposades.

La millora que apliquem sobre la topologia és la d'augmentar el nombre d'interconnexions entre els nodes presents a la malla de 8 x 8 que formen els diferents nodes computacionals.

Amb la disposició clàssica de la topologia només es té present la connexió d'aquestos de forma vertical i horitzontal, es a dir, amb enllaços amb el node a esquerra/dreta o damunt/avall.

Per tant, la millora proposada conserva aquestes interconnexions que defineix la topologia clàssica Torus de dues dimensions però em definit més associacions entre nodes, de mode que afegim un nou concepte de connexió: les connexions en diagonal.

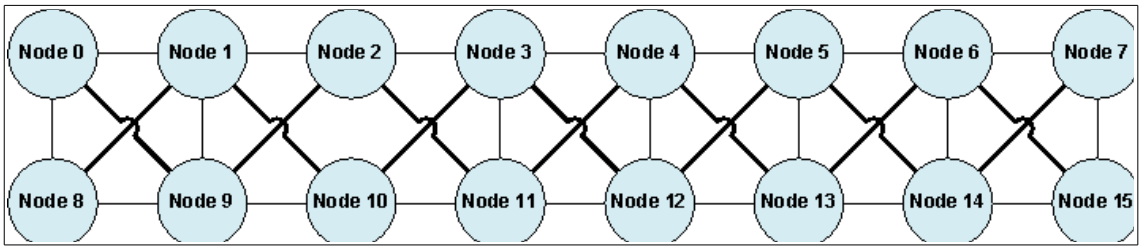


Figura 6. Topologia 2D Torus millorada

Com es pot veure a la figura anterior, s'afegeixen connexions en diagonal entre nodes, i es conserven les ja existents, per tal de reduir el nombre de salts entre nodes i fer que la comunicació entre ells sigui més directa, augmentant el nombre de possibles enllaços.

Per tant, s'augmenta el nombre d'interconnexions afegint un total de 98 noves connexions a les ja existents prèviament amb la definició clàssica de la topologia.

5.2.3. 3D Torus

Per aquesta topologia s'ha definit dues possibles millores per augmentar les interconnexions entre nodes i així millorar el rendiment en l'execució d'aplicacions MPI.

La primera de les millores consisteix en definir dos nous enllaços per cadascun dels cubs que formen la distribució dels nodes. D'aquesta manera es redueix el nombre de salts necessaris per moure's dintre de l'arquitectura.

A la figura 7 es poden veure les dues noves connexions per cadascun dels cubs que formen la topologia (enllaços de color vermell i blau, respectivament). Recordem que aquesta topologia tridimensional està formada per 27 cubs com el següent. A la figura només es mostra el primers d'ells.

La segona millora proposada manté aquest enllaços dins dels 27 cubs i afegeix 16 connexions addicionals, entre els nodes externs que formen el cub (nodes 0, 3, 12, 15, 48, 51, 60 i 63) i connexions directes entre els nodes més allunyats.

Aquest fet d'augmentar les connexions hauria de reduir els temps d'execució dels tests, ja que el nombre de salts necessaris per moure's per dins de l'estructura es veu reduït.

Finalment, i a mode de resum, mostrem com queden els nombres d'enllaços abans i després d'aplicar aquestes millores:

- Sistema base: 176 connexions
- Primera millora: 230 connexions (176 existents + 54 noves)
- Segona millora: 246 connexions (176 + 54 + 16 noves)

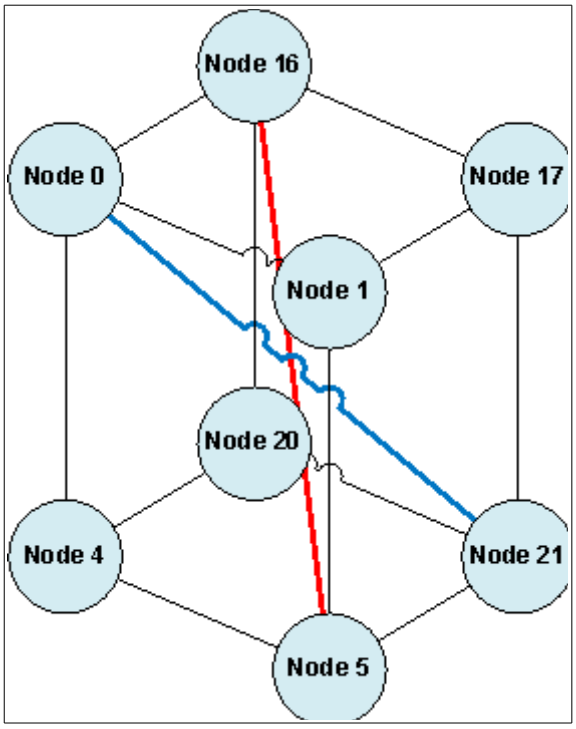


Figura 7. Topologia 3D Torus millorada

6. Avaluació de les millores proposades

6.1. 10 Gigabit Ethernet amb connexions SFP+

6.1.1. Temps d'execució

Taules comparatives dels temps d'execució entre les tecnologies 10 Gigabit Ethernet, fent servir les connexions tipus SFP+ en el segon cas i donant el percentatge de millora aportat.

Fat Tree Topology			
	10G	**10G – SFP+**	**Improvement - %**
Latency – 1K reps	1846391	966375	47.66
Latency – 100K reps	8417478243	8329477450	1.05
Array Assignment – 1 rep	193978968.113927	193967135	0.01
Array Assignment – 10 rep	1939789668.44986	1939671336	0.01
Matrix Multiply – 1K rep	437470199	434851079	0.60
Matrix Multiply – 100K rep	4347538386080	4347276470869	0.01
Prime Generator – 1M	365	106	70.96
Prime Generator – 100M	421	51	87.89
Wave Equation	116341	16814	85.55

Temps d'execució per la topologia Fat Tree

2D Torus Topology			
	10G	**10G – SFP+**	**Improvement - %**
Latency – 1K reps	1268925	259262	79.57
Latency – 100K reps	1155996232	1055031034	8.73
Array Assignment – 1 rep	29887042.0170438	29873401	0.05
Array Assignment – 10 rep	298870398.131636	298733994	0.05
Matrix Multiply – 1K rep	57847978	54859487	5.17
Matrix Multiply – 100K rep	543758702489	543459862934	0.05
Prime Generator – 1M	725	96	86.76
Prime Generator – 100M	780	152	80.51
Wave Equation	57316	8000	86.04

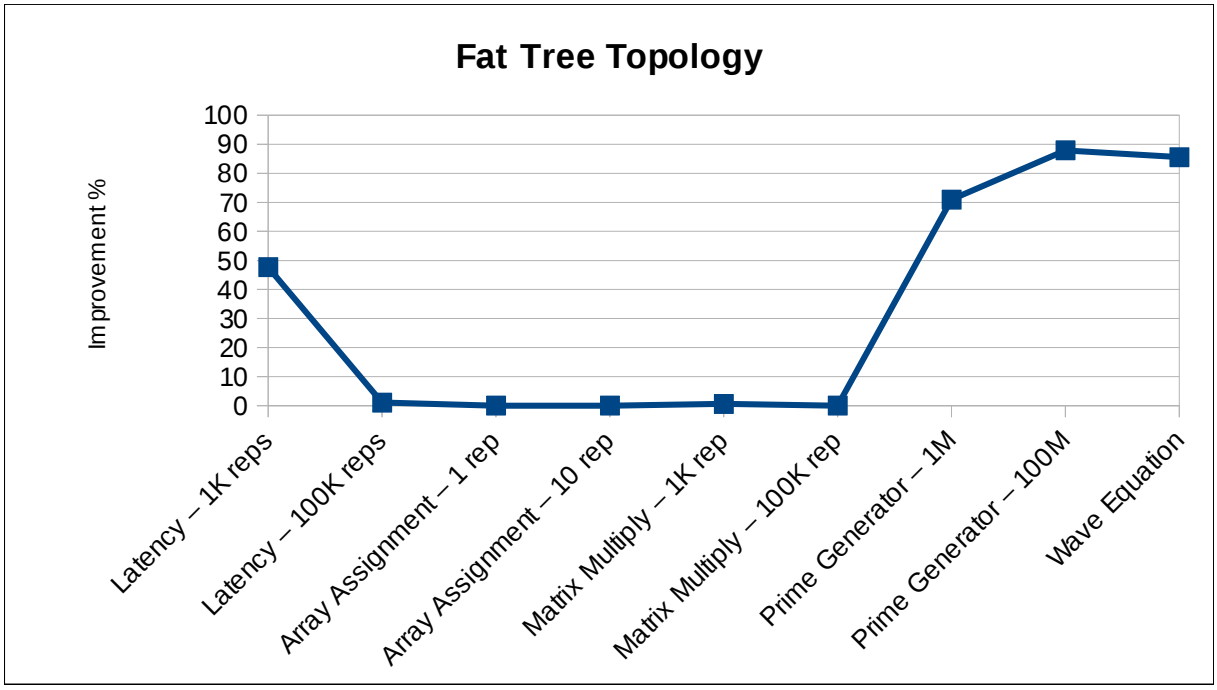
Temps d'execució per la topologia 2D Torus

3D Torus Topology			
	10G	**10G – SFP+**	**Improvement - %**
Latency – 1K reps	1069473	232668	78.24
Latency – 100K reps	1135976310	1052335107	7.36
Array Assignment – 1 rep	29884337.6822189	29873041	0.04
Array Assignment – 10 rep	298843354.74628	298730388	0.04
Matrix Multiply – 1K rep	57268762	54823121	4.27
Matrix Multiply – 100K rep	544880711688	545242666565	0.00
Prime Generator – 1M	544	72	86.76
Prime Generator – 100M	599	128	78.63
Wave Equation	110729	14902	86.54

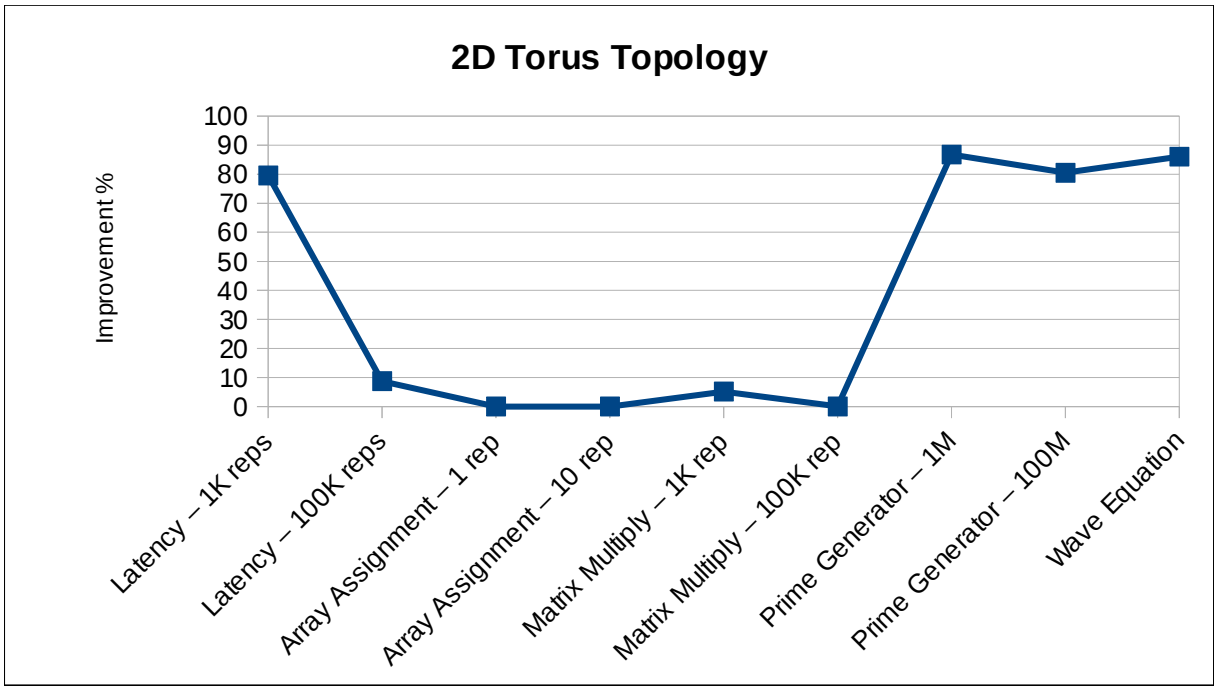
Temps d'execució per la topologia 3D Torus

6.1.2. Gràfiques comparatives

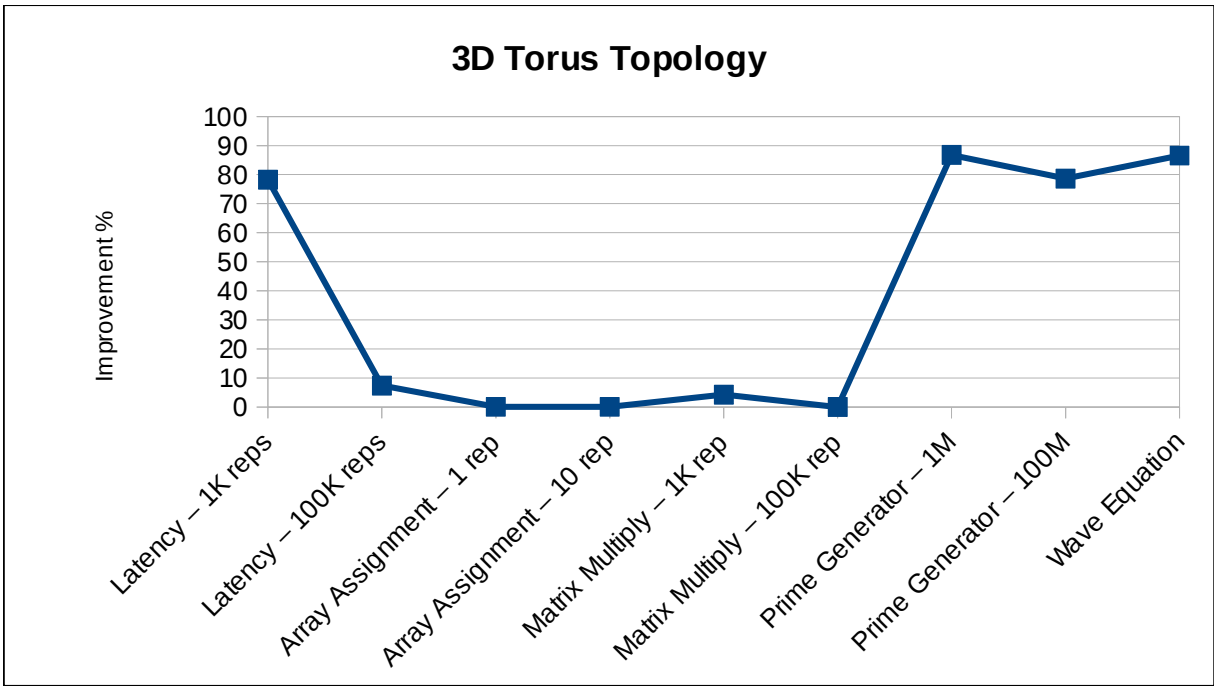
Gràfiques comparatives entre els dos tipus de connexions, mostrant el percentatge de millora que aporta utilitzar connexions tipus SFP+ amb els mateixos test MPI.



Gràfica amb percentatge millora – Topologia Fat Tree



Gràfica amb percentatge millora – Topologia 2D Torus



Gràfica amb percentatge millora – Topologia 3D Torus

6.2. 40 / 100 Gigabit Ethernet

6.2.1. Temps d'execució

Taules comparatives dels temps d'execució entre les tecnologies 10 Gigabit Ethernet i el nou estàndard 40 / 100 Gigabit Ethernet, mostrant el percentatge de millora que dóna aquesta segona opció.

Fat Tree Topology			
	10G	**40G**	**Improvement %**
Latency – 1K reps	1846391	343134	81.42
Latency – 100K reps	8417478243	2092523323	75.14
Array Assignment – 1 rep	193978968.113927	48493153	75.00
Array Assignment – 10 rep	1939789668.44986	484931509	75.00
Matrix Multiply – 1K rep	437470199	109014932	75.08
Matrix Multiply – 100K rep	4347538386080	1086849334863	75.00
Prime Generator – 1M	365	49	86.58
Prime Generator – 100M	421	104	75.30
Wave Equation	116341	15725	86.48

Temps d'execució per la topologia Fat Tree

2D Torus Topology			
	10G	**40G**	**Improvement %**
Latency – 1K reps	1268925	259214	79.57
Latency – 100K reps	1155996232	1055026305	8.73
Array Assignment – 1 rep	29887042.0170438	25051429	16.18
Array Assignment – 10 rep	298870398.131636	250514272	16.18
Matrix Multiply – 1K rep	57847978	54786560	5.29
Matrix Multiply – 100K rep	543758702489	543452576422	0.06
Prime Generator – 1M	725	96	86.76
Prime Generator – 100M	780	152	80.51
Wave Equation	57316	7967	86.10

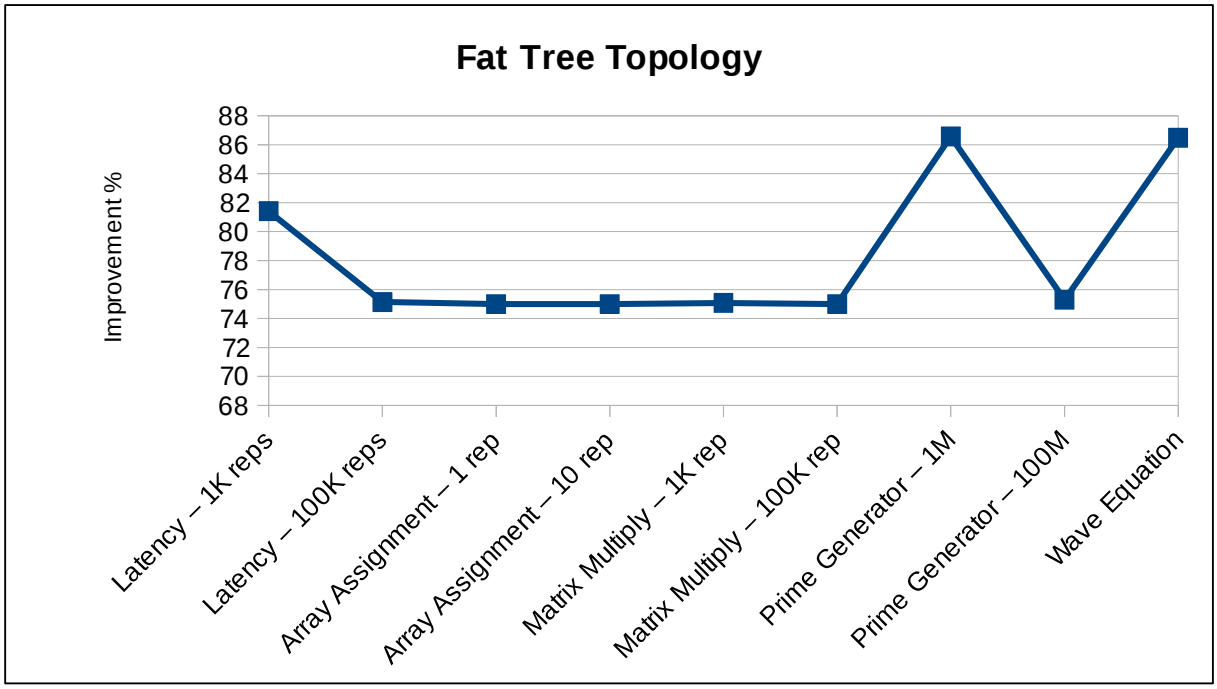
Temps d'execució per la topologia 2D Torus

3D Torus Topology			
	10G	**40G**	**Improvement %**
Latency – 1K reps	1069473	232621	78.25
Latency – 100K reps	1135976310	1052366941	7.36
Array Assignment – 1 rep	29884337.6822189	25051068	16.17
Array Assignment – 10 rep	298843354.74628	250510666	16.17
Matrix Multiply – 1K rep	57268762	54708456	4.47
Matrix Multiply – 100K rep	544880711688	543444766005	0.26
Prime Generator – 1M	544	72	86.76
Prime Generator – 100M	599	128	78.63
Wave Equation	110729	15219	86.26

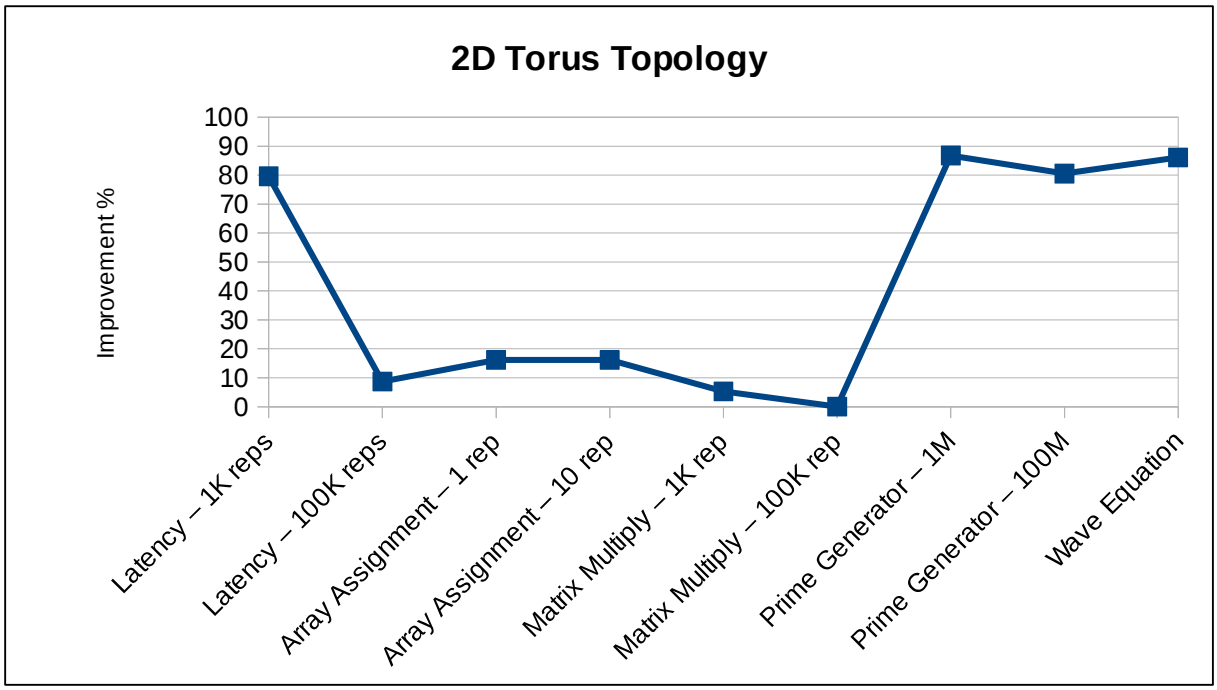
Temps d'execució per la topologia 3D Torus

6.2.2. Gràfiques comparatives

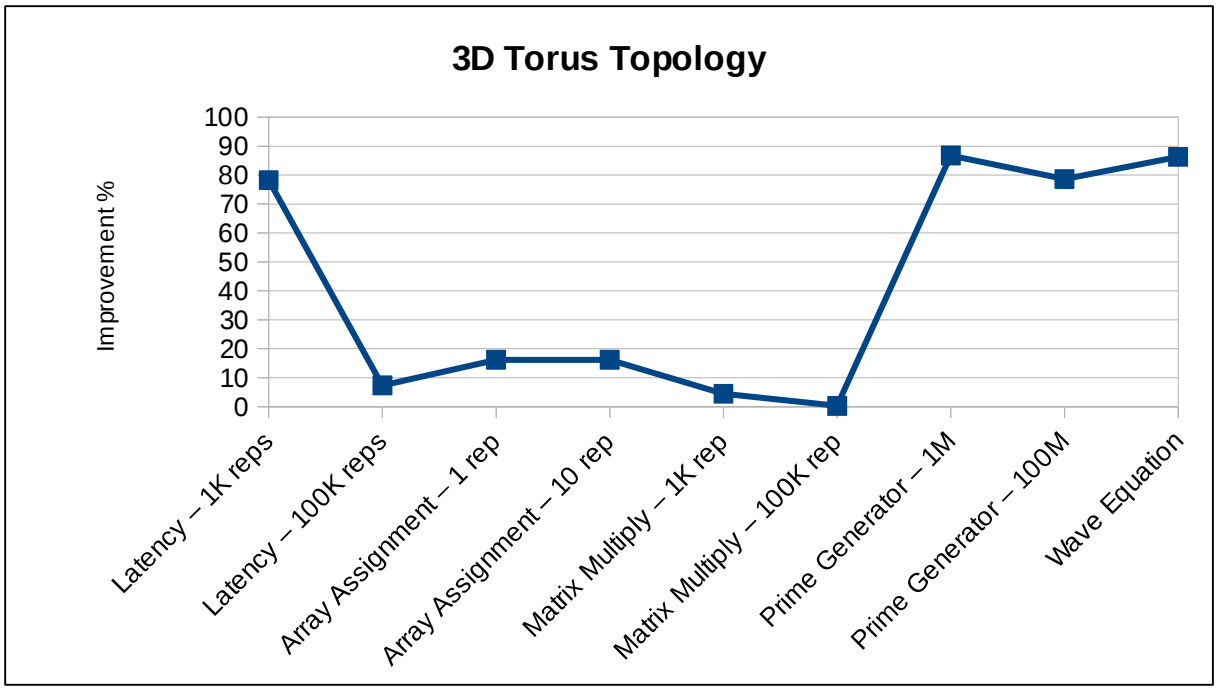
Gràfiques comparatives entre les tecnologies d'interconnexió 10Gigabit-Ethernet i 40/100 Gigabit-Ethernet, mostrant el percentatge de millora que dóna l'execució dels mateixos test MPI amb cadascuna d'elles.



Gràfica amb percentatge millora – Topologia Fat Tree



Gràfica amb percentatge millora – Topologia 2D Torus



Gràfica amb percentatge millora – Topologia 3D Torus

6.3. InfiniBand NDR

6.3.1. Temps d'execució

Taules comparatives dels temps d'execució entre les tecnologies InfiniBand EDR i NDR, amb el percentatge de millora que aporta la segona.

Fat Tree Topology			
	InfiniBand – EDR	**InfiniBand – NDR**	**Improvement - %**
Latency – 1K reps	299840	203035	32.29
Latency – 100K reps	765058876	466963601	38.96
Array Assignment – 1 rep	17321370.5551637	10525420	39.23
Array Assignment – 10 rep	173213683.100748	105254183	39.23
Matrix Multiply – 1K rep	39461642	24040466	39.08
Matrix Multiply – 100K rep	388213250759	235899671948	39.23
Prime Generator – 1M	80	56	30.00
Prime Generator – 100M	136	112	17.65
Wave Equation	24867	17652	29.01

Temps d'execució per la topologia Fat Tree

2D Torus Topology			
	InfiniBand – EDR	**InfiniBand – NDR**	**Improvement - %**
Latency – 1K reps	362762	285095	21.41
Latency – 100K reps	1065380922	1057614368	0.73
Array Assignment – 1 rep	24249166.4012273	24248117	0.00
Array Assignment – 10 rep	242491641.97693	242481149	0.00
Matrix Multiply – 1K rep	55083197	54854004	0.42
Matrix Multiply – 100K rep	543482239912	543459320644	0.00
Prime Generator – 1M	161	112	30.43
Prime Generator – 100M	216	168	22.22
Wave Equation	12929	9239	28.54

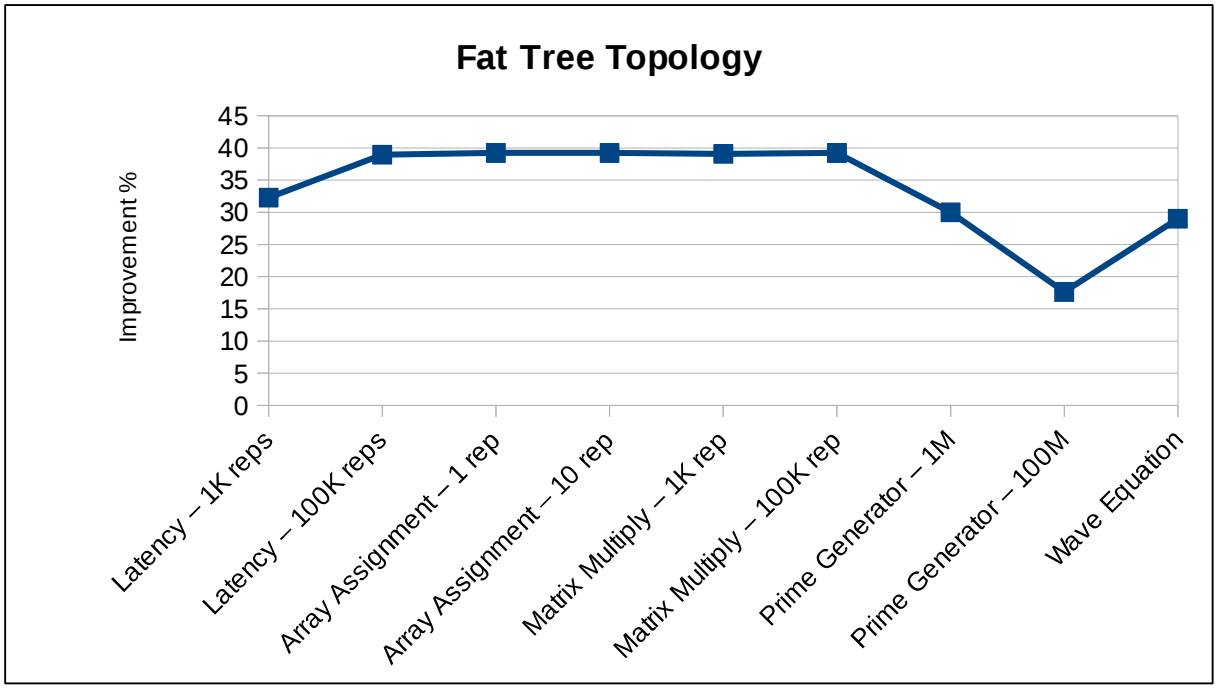
Temps d'execució per la topologia 2D Torus

3D Torus Topology			
	InfiniBand – EDR	**InfiniBand – NDR**	**Improvement - %**
Latency – 1K reps	318439	254069	20.21
Latency – 100K reps	1060948648	1054511776	0.61
Array Assignment – 1 rep	24248565.4354612	24247696	0.00
Array Assignment – 10 rep	242485632.349031	242476942	0.00
Matrix Multiply – 1K rep	54953734	54762762	0.35
Matrix Multiply – 100K rep	543469294122	543450196879	0.00
Prime Generator – 1M	120	84	30.00
Prime Generator – 100M	176	140	20.45
Wave Equation	25050	17843	28.77

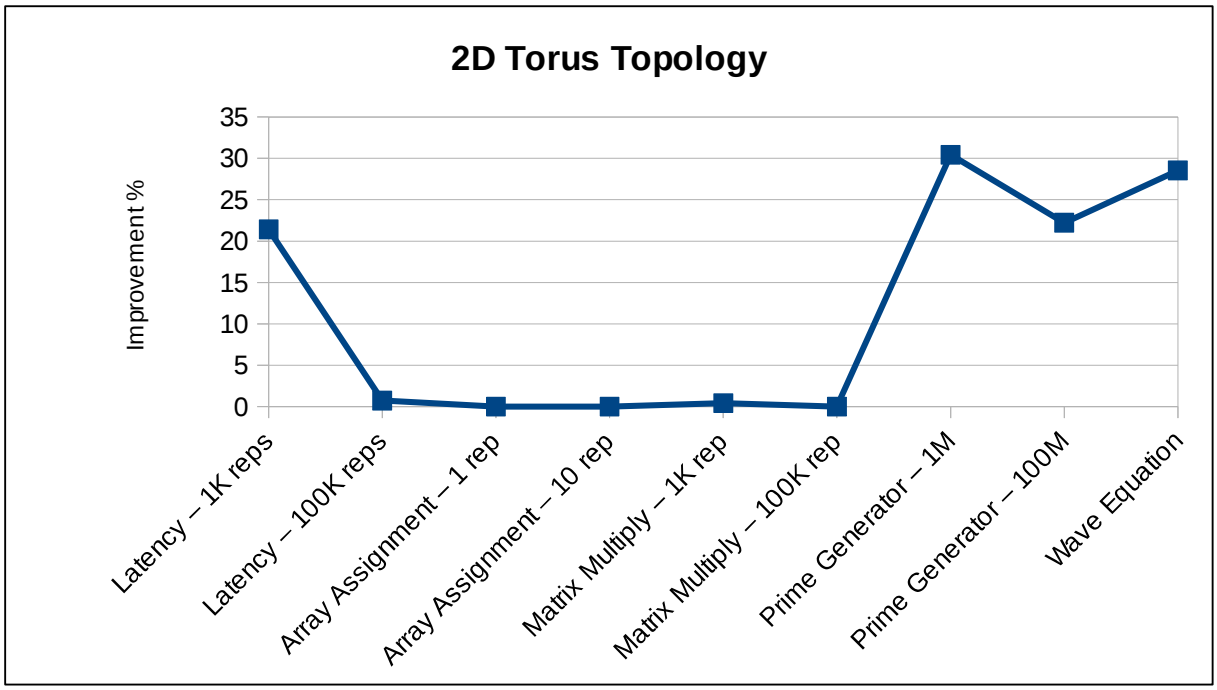
Temps d'execució per la topologia 3D Torus

6.3.2. Gràfiques comparatives

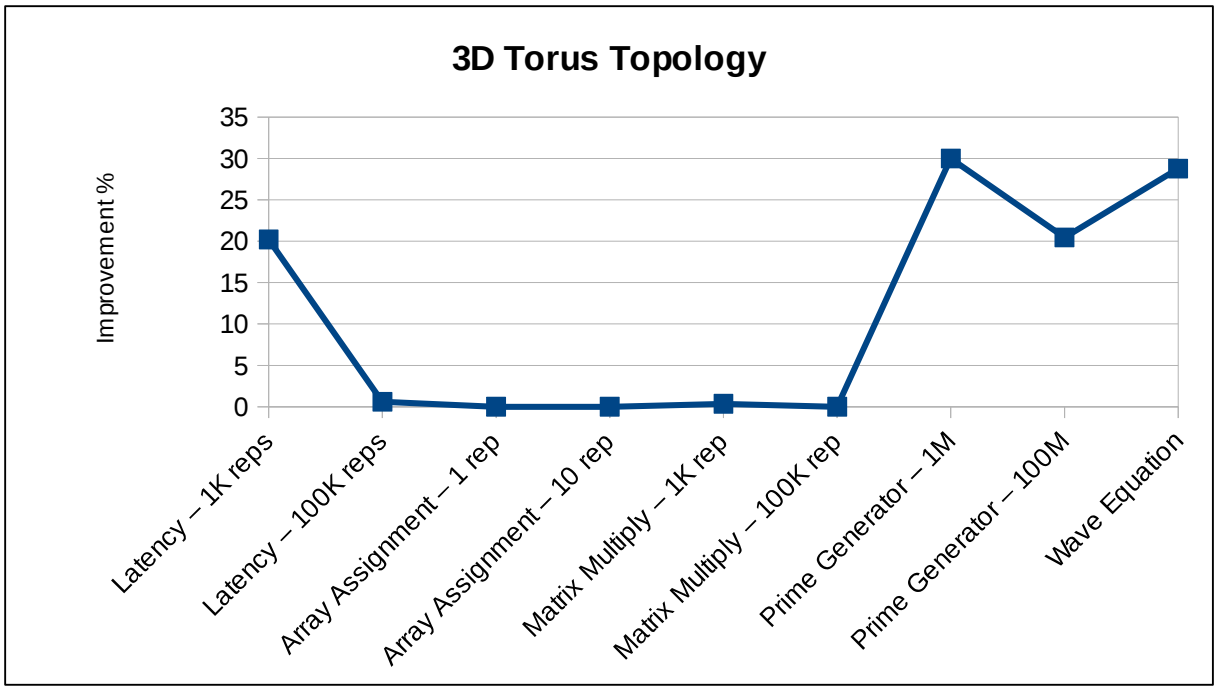
Gràfiques comparatives entre la millor de les tecnologies actuals (InfiniBand EDR) amb la propera evolució d'aquesta als pròxims anys, anomenada NDR. Es mostra el percentatge de millora amb les tres topologies estudiades.



Gràfica amb percentatge de millora – Topologia Fat Tree



Gràfica amb percentatge de millora – Topologia 2D Torus



Gràfica amb percentatge de millora – Topologia 3D Torus

6.4. Fat Tree 1:1 amb InfiniBand

6.4.1. Temps d'execució

Taules comparatives dels temps d'execució entre la disposició original i millorada (relació 1:1 entre nivells) amb les diferents tecnologies d'interconnexió InfiniBand.

Fat Tree Topology - InfiniBand SDR			
	Original	**Improved**	**% improvement**
Latency – 1K reps	3295186	3295186	0
Latency – 100K reps	10620566637	10620566637	0
Array Assignment – 1 rep	242486983.249023	242486983	1.02695409509579E-007
Array Assignment – 10 rep	2424869822.44826	2424869822	1.84859345608857E-008
Matrix Multiply – 1K rep	549776284	549776288	0
Matrix Multiply – 100K rep	5434716837634	5434716838016	0
Prime Generator – 1M	809	809	0
Prime Generator – 100M	865	865	0
Wave Equation	254068	254038	0.0118078625

Temps d'execució per la topologia Fat Tree amb connexions InfiniBand SDR

Fat Tree Topology - InfiniBand DDR			
	Original	**Improved**	**% improvement**
Latency – 1K reps	1647593	1647593	0
Latency – 100K reps	5310283334	5310283334	0
Array Assignment – 1 rep	121243494.232825	121243494	0.000000192
Array Assignment – 10 rep	1212434925.64465	1212434925	5.31698560735094E-008
Matrix Multiply – 1K rep	274888143	274888145	0
Matrix Multiply – 100K rep	2717358418976	2717358419185	0
Prime Generator – 1M	404	404	0
Prime Generator – 100M	460	460	0
Wave Equation	126709	127009	0

Temps d'execució per la topologia Fat Tree amb connexions InfiniBand DDR

Fat Tree Topology - InfiniBand QDR			
	Original	**Improved**	**% improvement**
Latency – 1K reps	846361	846361	0
Latency – 100K reps	2657398113	2657398113	0
Array Assignment – 1 rep	60622053.140754	60622053	2.32182827630822E-007
Array Assignment – 10 rep	606220511.389775	606220511	6.42959179231184E-008
Matrix Multiply – 1K rep	137511242	137511242	0
Matrix Multiply – 100K rep	1358685926606	1358685926703	0
Prime Generator – 1M	210	210	0
Prime Generator – 100M	266	266	0
Wave Equation	66423	66039	0.578113003

Temps d'execució per la topologia Fat Tree amb connexions InfiniBand QDR

Fat Tree Topology - InfiniBand FDR			
	Original	**Improved**	**% improvement**
Latency – 1K reps	464294	464294	0
Latency – 100K reps	1516579137	1516579137	0
Array Assignment – 1 rep	34640915.4023032	34640915	1.16135267091977E-006
Array Assignment – 10 rep	346409132.454265	346409132	1.31135408310001E-007
Matrix Multiply – 1K rep	78520279	78520279	0
Matrix Multiply – 100K rep	776386200809	776386200848	0
Prime Generator – 1M	113	113	0
Prime Generator – 100M	168	168	0
Wave Equation	35328	35390	0

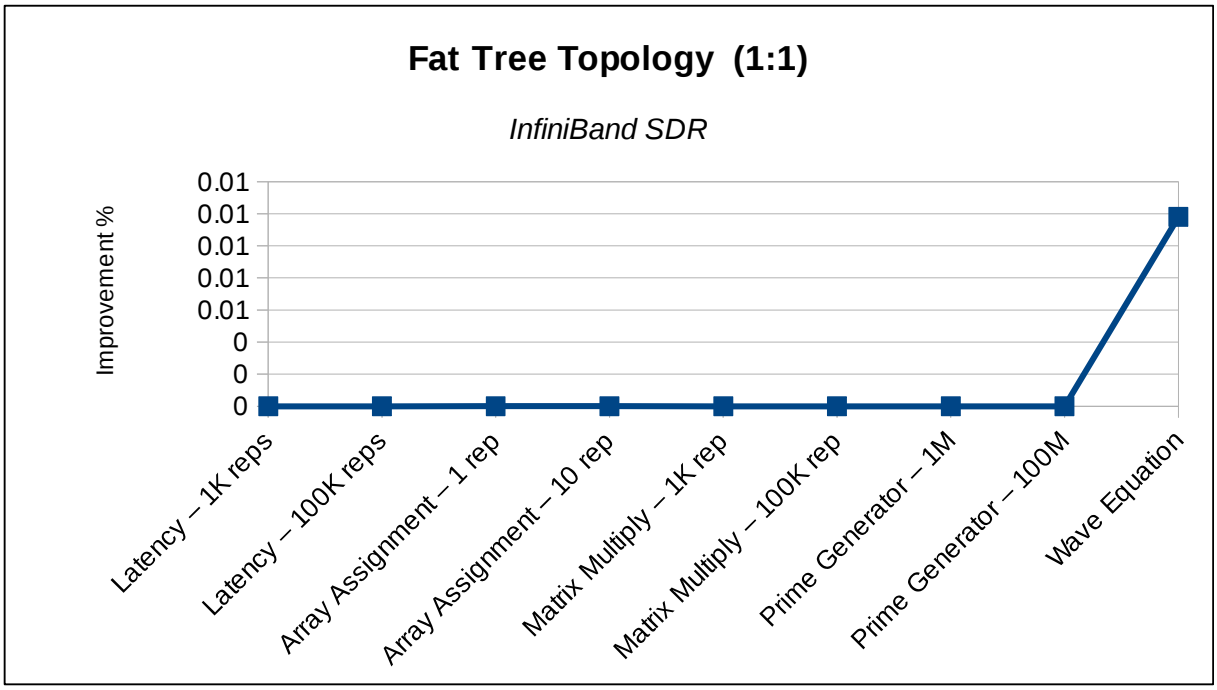
Temps d'execució per la topologia Fat Tree amb connexions InfiniBand FDR

Fat Tree Topology - InfiniBand EDR			
	Original	**Improved**	**% improvement**
Latency – 1K reps	299840	299840	0
Latency – 100K reps	765058876	765058876	0
Array Assignment – 1 rep	17321370.5551637	17321370	3.20507952267235E-006
Array Assignment – 10 rep	173213683.100748	173213683	5.81639909569276E-008
Matrix Multiply – 1K rep	39461642	39461643	-2.53410641448681E-006
Matrix Multiply – 100K rep	388213250759	388213250770	-2.83348811080941E-009
Prime Generator – 1M	80	80	0
Prime Generator – 100M	136	136	0
Wave Equation	24867	25190	-1.2989102023

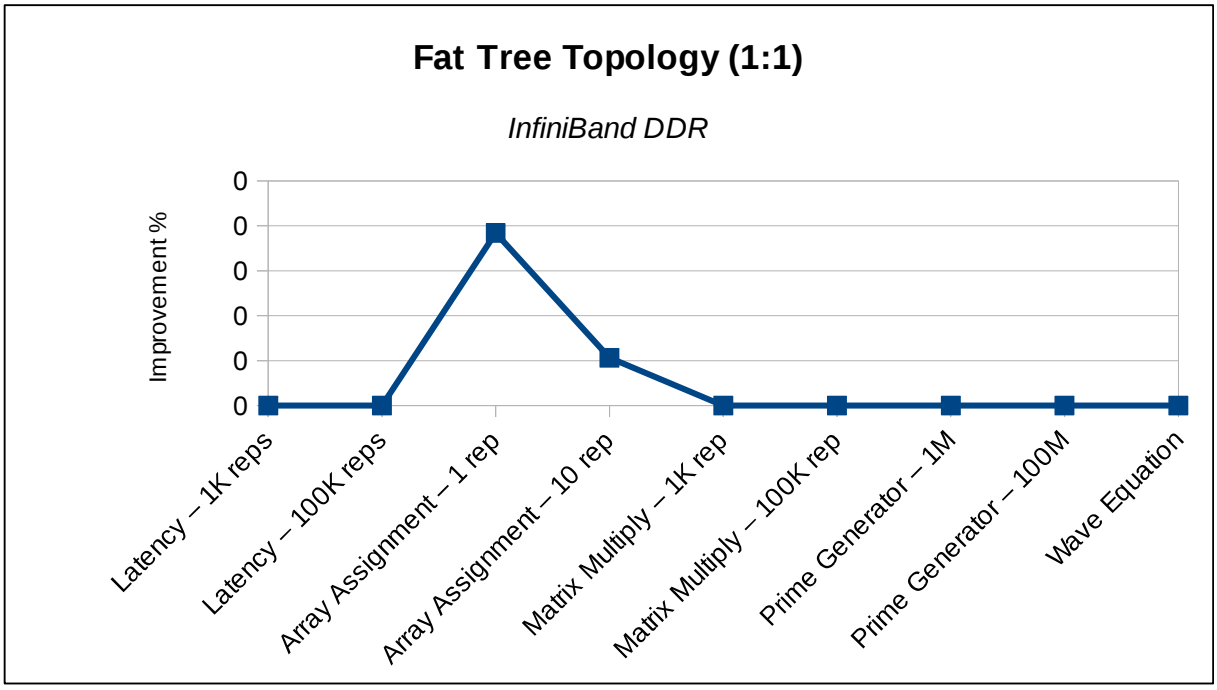
Temps d'execució per la topologia Fat Tree amb connexions InfiniBand FDR

6.4.2. Gràfiques comparatives

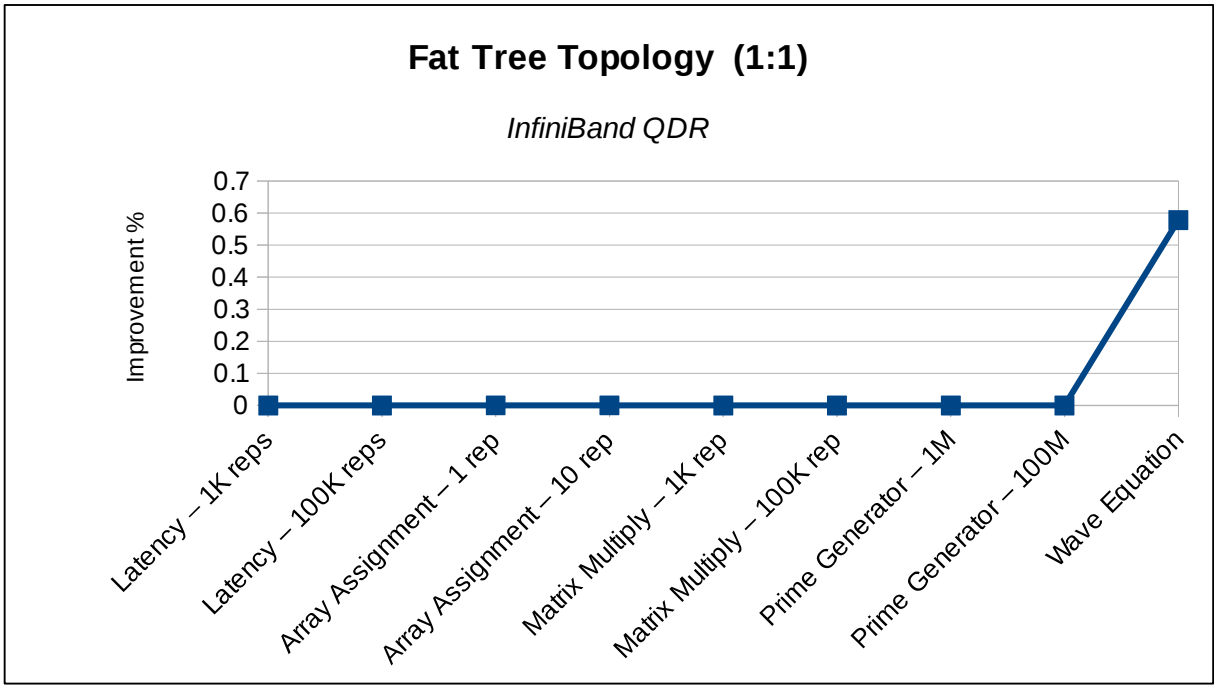
Gràfiques comparatives entre la topologia *Fat Tree* no equilibrada i equilibrada 1:1, per cadascuna de les tecnologies InfiniBand estudiades. Es mostra el percentatge de millora obtingut amb els mateixos test MPI.



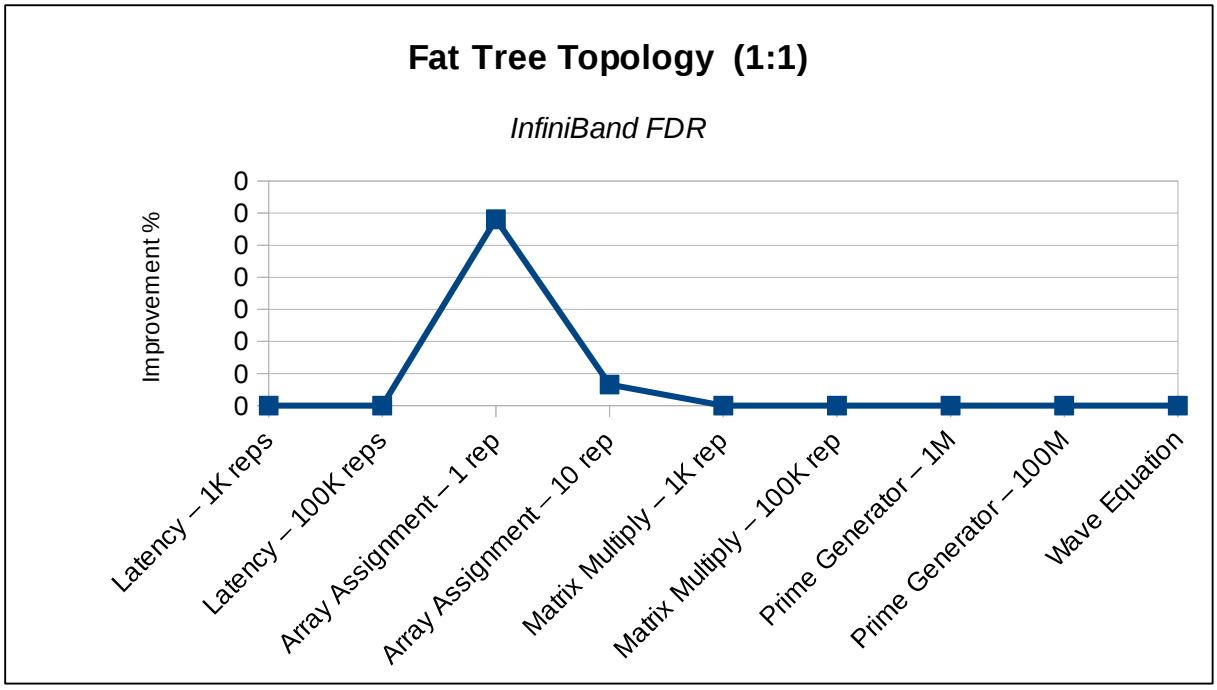
Gràfica amb percentatge de millora – InfiniBand SDR



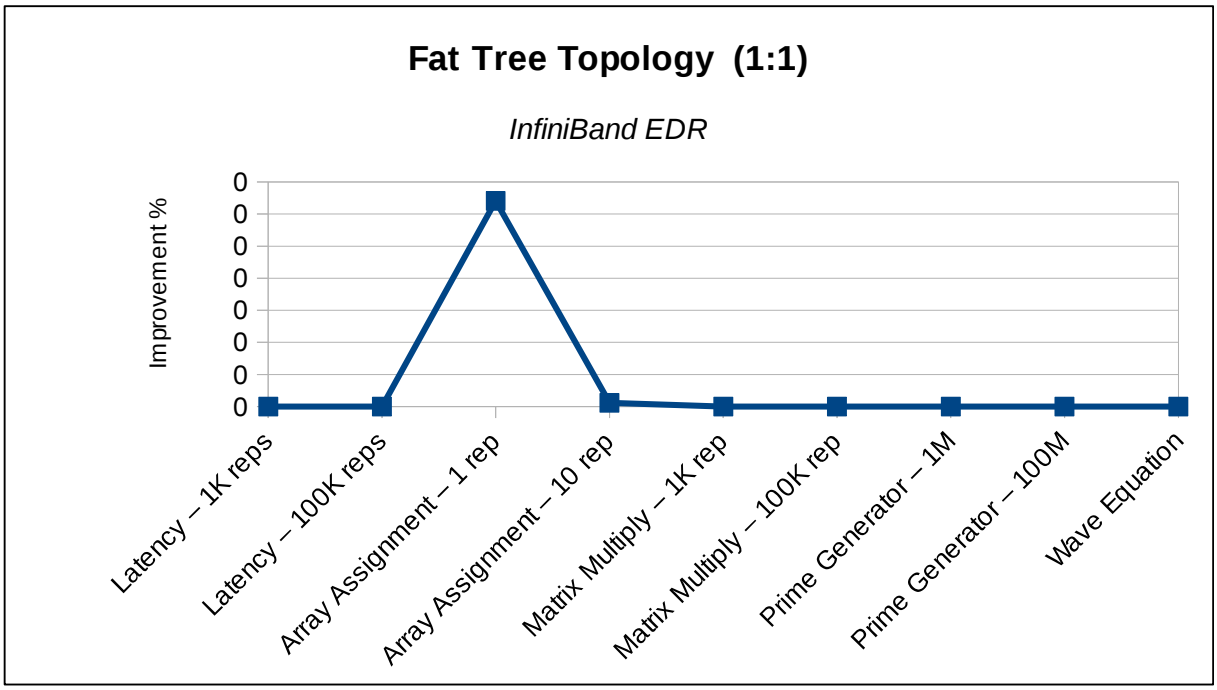
Gràfica amb percentatge de millora – InfiniBand DDR



Gràfica amb percentatge de millora – InfiniBand QDR



Gràfica amb percentatge de millora – InfiniBand FDR



Gràfica amb percentatge de millora – InfiniBand EDR

6.5. 2D Torus

6.5.1. Temps d'execució

Taules comparatives dels temps d'execució entre la topologia 2D Torus clàssica i la topologia millorada amb més enllaços d'interconnexió entre nodes.

2D Torus Topology - Ethernet 1Gbps			
	Classic	**Enhanced**	**% Improvement**
Latency – 1K reps	2305045	1816983	21.17
Latency – 100K reps	1259607026	1210801636	3.87
Array Assignment – 1 rep	87764698.9694468	87758081	0.01
Array Assignment – 10 rep	877646967.738956	877580793	0.01
Matrix Multiply – 1K rep	61891384	60315780	2.55
Matrix Multiply – 100K rep	544162746527	544005485221	0.03
Prime Generator – 1M	1370	857	37.45
Prime Generator – 100M	1426	913	35.97
Wave Equation	108636	107946	0.64

Temps d'execució entre topologies 2D Torus – Connexions 1Gigabit-Ethernet

2D Torus Topology - Ethernet 10Gbps			
	Classic	**Enhanced**	**% Improvement**
Latency – 1K reps	1268925	1010539	20.36
Latency – 100K reps	1155996232	1130158085	2.24
Array Assignment – 1 rep	29887042.0170438	29883538	0.01
Array Assignment – 10 rep	298870398.131636	298835364	0.01
Matrix Multiply – 1K rep	57847978	57080637	1.33
Matrix Multiply – 100K rep	543758702489	543681965517	0.01
Prime Generator – 1M	725	453	37.52
Prime Generator – 100M	780	508	34.87
Wave Equation	57316	56807	0.89

Temps d'execució entre topologies 2D Torus – Connexions 10Gigabit-Ethernet

2D Torus Topology - InfiniBand SDR			
	Classic	**Enhanced**	**% Improvement**
Latency – 1K reps	2692826	2118635	21.32
Latency – 100K reps	1298384627	1240966522	4.42
Array Assignment – 1 rep	31513602.4727805	31505817	0.02
Array Assignment – 10 rep	315136002.762289	315058150	0.02
Matrix Multiply – 1K rep	62103605	60399790	8.09
Matrix Multiply – 100K rep	544184265656	544013829853	0.09
Prime Generator – 1M	1611	1007	37.49
Prime Generator – 100M	1667	1063	36.23
Wave Equation	126556	125900	0.52

Temps d'execució entre topologies 2D Torus – Connexions InfiniBand SDR

2D Torus Topology - InfiniBand DDR			
	Classic	**Enhanced**	**% Improvement**
Latency – 1K reps	1398346	1111250	20.53
Latency – 100K reps	1168938124	1140229072	2.46
Array Assignment – 1 rep	27477804.6528053	27473912	0.01
Array Assignment – 10 rep	274778024.553931	274739098	0.01
Matrix Multiply – 1K rep	58204660	57352408	1.46
Matrix Multiply – 100K rep	543794370787	543709142697	0.02
Prime Generator – 1M	805	503	37.52
Prime Generator – 100M	861	559	35.08
Wave Equation	64111	62874	1.93

Temps d'execució entre topologies 2D Torus – Connexions InfiniBand DDR

2D Torus Topology - InfiniBand QDR			
	Classic	**Enhanced**	**% Improvement**
Latency – 1K reps	776995	627705	19.21
Latency – 100K reps	1106803724	1091875017	1.35
Array Assignment – 1 rep	25460255.4921535	25458231	0.01
Array Assignment – 10 rep	254602532.974807	254582291	0.01
Matrix Multiply – 1K rep	56332558	55889758	0.79
Matrix Multiply – 100K rep	543607177145	543562888161	0.01
Prime Generator – 1M	419	262	37.47
Prime Generator – 100M	474	317	33.12
Wave Equation	33376	33070	0.92

Temps d'execució entre topologies 2D Torus – Connexions InfiniBand QDR

2D Torus Topology - InfiniBand FDR			
	Classic	**Enhanced**	**% Improvement**
Latency – 1K reps	466320	385934	17.24
Latency – 100K reps	1075736665	1067698130	0.75
Array Assignment – 1 rep	24594991.99722	24593902	0.00
Array Assignment – 10 rep	245949897.964734	245938998	0.00
Matrix Multiply – 1K rep	55394257	55158852	0.42
Matrix Multiply – 100K rep	543513346322	543489805328	0.00
Prime Generator – 1M	225	141	37.33
Prime Generator – 100M	281	196	30.25
Wave Equation	18026	17904	0.68

Temps d'execució entre topologies 2D Torus – Connexions InfiniBand FDR

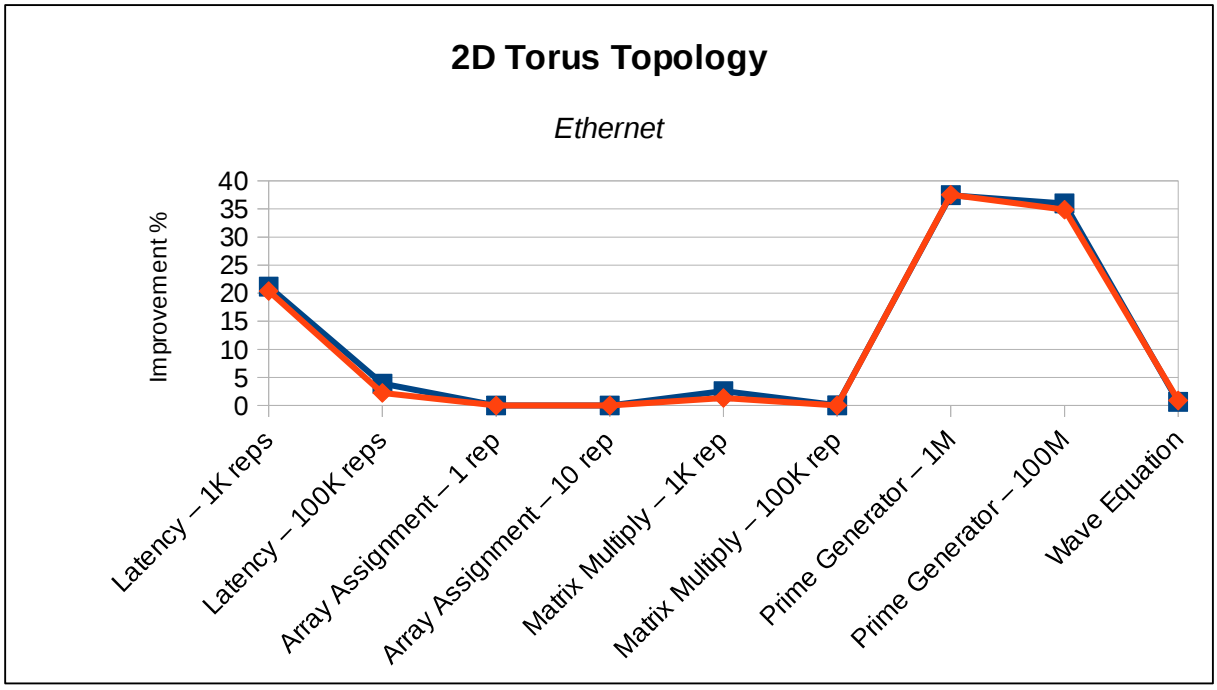
2D Torus Topology - InfiniBand EDR			
	Classic	**Enhanced**	**% Improvement**
Latency – 1K reps	362762	305343	15.83
Latency – 100K reps	1065380922	1059639111	0.54
Array Assignment – 1 rep	24249166.4012273	24248387	0.00
Array Assignment – 10 rep	242491641.97693	242483856	0.00
Matrix Multiply – 1K rep	55083197	54918025	0.30
Matrix Multiply – 100K rep	543482239912	543465722489	0.00
Prime Generator – 1M	161	100	37.89
Prime Generator – 100M	216	156	27.78
Wave Equation	12929	12973	-0.34

Temps d'execució entre topologies 2D Torus – Connexions InfiniBand EDR

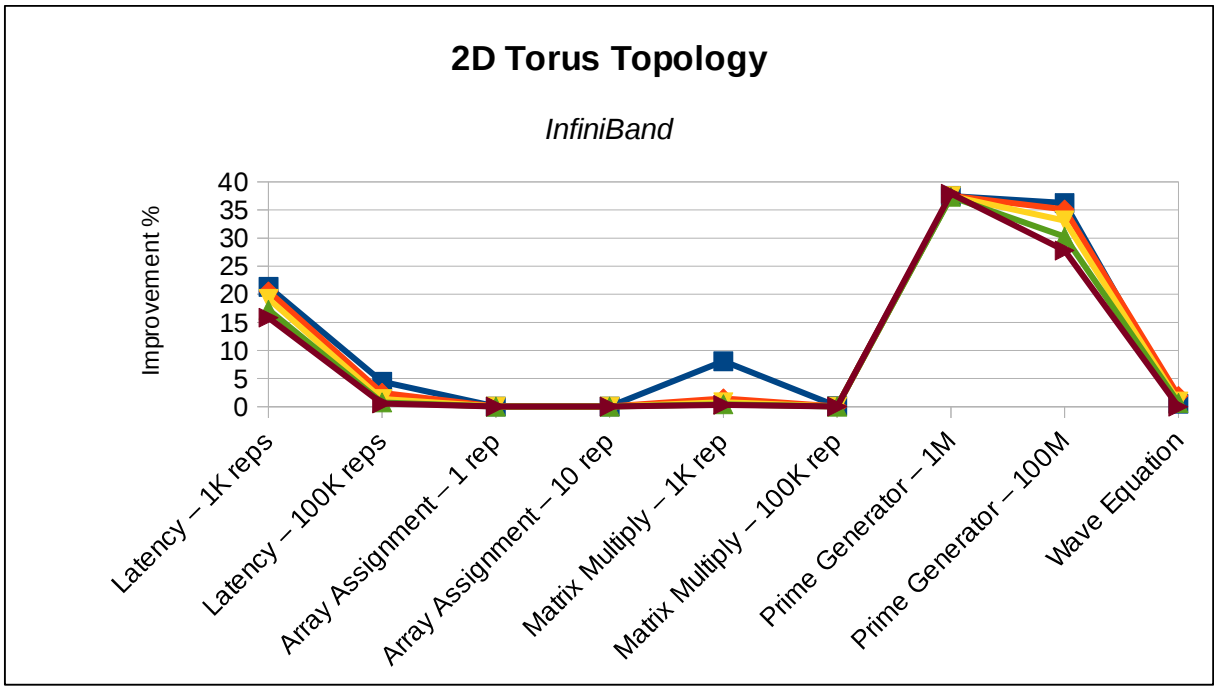
6.5.2. Gràfiques comparatives

Gràfiques comparatives entre la topologia clàssica 2D Torus prèviament analitzada i la millora proposada sobre aquesta, augmentant el nombre d'enllaços entre nodes. Es mostra el percentatge de millora per cada tecnologia d'interconnexió, tant Ethernet com InfiniBand.

S'han fet dues agrupacions: tecnologies Ethernet e InfiniBand.



Gràfica amb percentatge de millora – Tecnologies Ethernet



Gràfica amb percentatge de millora – Tecnologies InfiniBand

6.6. 3D Torus

6.6.1. Temps d'execució

Taules comparatives amb els temps d'execució amb les dues millores proposades sobre la topologia clàssica, augmentant el nombre d'enllaços entre nodes.

3D Torus Topology - Ethernet 1Gbps					
	Classic	**Enhanced**	**Enhanced (2)**	**% Improv. (1)**	**% Improv. (2)**
Latency – 1K reps	1928302	1620057	1303250	15.99	32.41
Latency – 100K reps	1221932697	1191108246	1159427560	2.52	5.12
Array Assignment – 1 rep	87759590.7780331	87755411	87751115	0.00	0.01
Array Assignment – 10 rep	877595885.82106	877554091	877511136	0.00	0.01
Matrix Multiply – 1K rep	61102011	60273810	58871612	1.36	3.65
Matrix Multiply – 100K rep	544084018353	544000432008	543861078475	0.02	0.04
Prime Generator – 1M	1028	857	518	16.63	49.61
Prime Generator – 100M	1084	913	574	15.77	47.05
Wave Equation	209937	175808	177825	16.26	15.30

Temps d'execució entre topologies 3D Torus – Connexions 1Gigabit-Ethernet

3D Torus Topology - Ethernet 10Gbps					
	Classic	**Enhanced**	**Enhanced (2)**	**% Improv. (1)**	**% Improv. (2)**
Latency – 1K reps	1069473	906285	738563	15.26	30.94
Latency – 100K reps	1135976310	1119704917	1102960045	1.43	2.91
Array Assignment – 1 rep	29884337.6822189	29882125	29879850	0.01	0.02
Array Assignment – 10 rep	298843354.74628	298821228	298798487	0.01	0.02
Matrix Multiply – 1K rep	57268762	56786114	56277703	0.84	1.73
Matrix Multiply – 100K rep	544880711688	544126041320	544318808614	0.14	0.10
Prime Generator – 1M	544	453	272	16.73	50.00
Prime Generator – 100M	599	509	328	15.03	45.24
Wave Equation	110729	93314	94380	15.73	14.76

Temps d'execució entre topologies 3D Torus – Connexions 10Gigabit-Ethernet

3D Torus Topology - InfiniBand SDR					
	Classic	**Enhanced**	**Enhanced (2)**	**% Improv. (1)**	**% Improv. (2)**
Latency – 1K reps	2249598	1886958	1514244	16.12	32.69
Latency – 100K reps	1254061887	1217797827	1180526432	2.89	5.86
Array Assignment – 1 rep	31507592.8414294	31502675	31497622	0.02	0.03
Array Assignment – 10 rep	315075906.380012	315026736	314976201	0.02	0.03
Matrix Multiply – 1K rep	60799214	59727321	58611423	6.60	3.60
Matrix Multiply – 100K rep	544053840856	543946634082	543835061715	-0.01	0.04
Prime Generator – 1M	1209	1007	605	16.71	49.96
Prime Generator – 100M	1264	1063	660	15.90	47.78
Wave Equation	245875	205428	205829	16.45	16.29

Temps d'execució entre topologies 2D Torus – Connexions InfiniBand SDR

3D Torus Topology - InfiniBand DDR					
	Classic	**Enhanced**	**Enhanced (2)**	**% Improv. (1)**	**% Improv. (2)**
Latency – 1K reps	1176732	995412	809055	15.41	31.25
Latency – 100K reps	1146776754	1128644724	1110009027	1.58	3.21
Array Assignment – 1 rep	27474799.8376446	27472341	27469814	0.01	0.02
Array Assignment – 10 rep	274747976.360595	274723391	274698123	0.01	0.02
Matrix Multiply – 1K rep	57550768	57010471	56458688	0.94	1.90
Matrix Multiply – 100K rep	543728997657	543674949314	543619789096	0.01	0.02
Prime Generator – 1M	604	503	302	16.72	50.00
Prime Generator – 100M	660	559	358	15.30	45.76
Wave Equation	122957	103005	102497	16.23	16.64

Temps d'execució entre topologies 2D Torus – Connexions InfiniBand DDR

3D Torus Topology - InfiniBand QDR					
	Classic	**Enhanced**	**Enhanced (2)**	**% Improv. (1)**	**% Improv. (2)**
Latency – 1K reps	661755	567469	470563	14.25	28.89
Latency – 100K reps	1095279812	1085851156	1076160593	0.86	1.75
Array Assignment – 1 rep	25458693.003292	25457414	25456100	0.01	0.01
Array Assignment – 10 rep	254586907.902765	254574123	254560984	0.01	0.01
Matrix Multiply – 1K rep	55991171	55709228	55425136	0.50	1.01
Matrix Multiply – 100K rep	543573038085	543544843299	543516434415	0.01	0.01
Prime Generator – 1M	314	262	157	16.56	50.00
Prime Generator – 100M	369	317	212	14.09	42.55
Wave Equation	64465	54273	54255	15.81	15.84

Temps d'execució entre topologies 2D Torus – Connexions InfiniBand QDR

3D Torus Topology - InfiniBand FDR					
	Classic	**Enhanced**	**Enhanced (2)**	**% Improv. (1)**	**% Improv. (2)**
Latency – 1K reps	404268	353499	301319	12.56	25.47
Latency – 100K reps	1069531481	1064454513	1059236517	0.47	0.96
Array Assignment – 1 rep	24594150.656856	24593462	24592754	0.00	0.01
Array Assignment – 10 rep	245941484.476776	245934600	245927525	0.00	0.01
Matrix Multiply – 1K rep	55212248	55064046	54910925	0.27	0.55
Matrix Multiply – 100K rep	543495145633	543480325279	543465013458	0.00	0.01
Prime Generator – 1M	169	141	84	16.57	50.30
Prime Generator – 100M	224	196	140	12.50	37.50
Wave Equation	35065	29946	30751	14.60	12.30

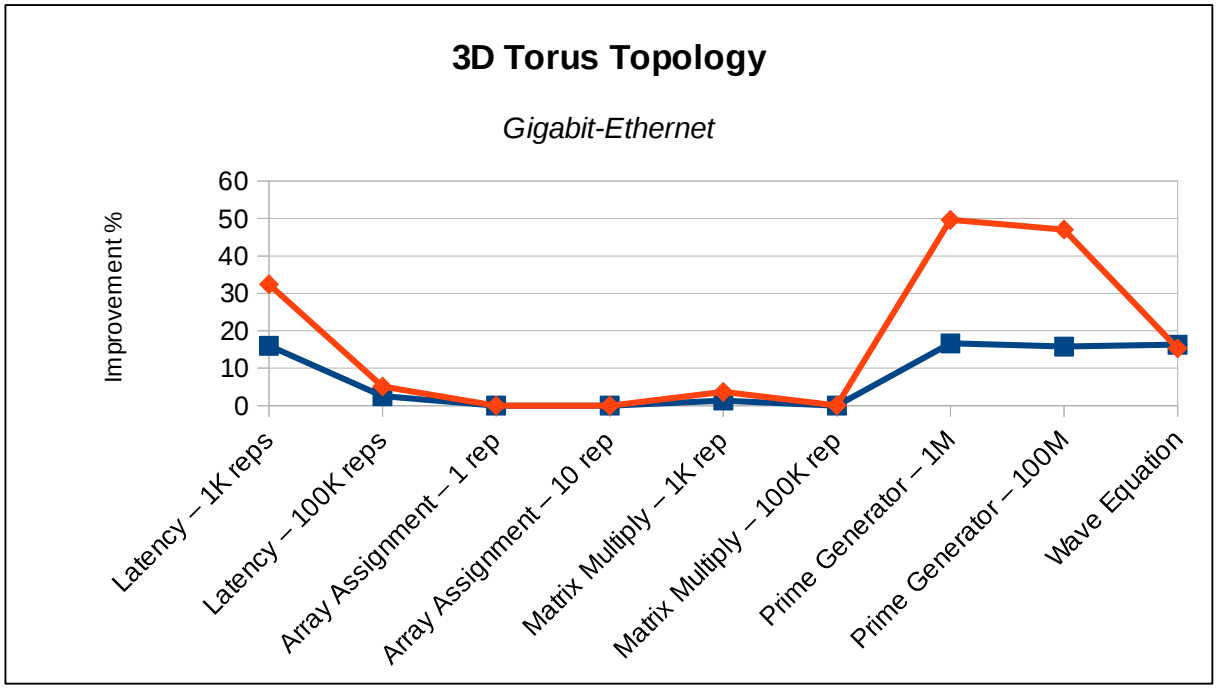
Temps d'execució entre topologies 2D Torus – Connexions InfiniBand FDR

3D Torus Topology - InfiniBand EDR					
	Classic	**Enhanced**	**Enhanced (2)**	**% Improv. (1)**	**% Improv. (2)**
Latency – 1K reps	318439	282175	244904	11.39	23.09
Latency – 100K reps	1060948648	1057322242	1053595102	0.34	0.69
Array Assignment – 1 rep	24248565.4354612	24248073	24247568	0.00	0.00
Array Assignment – 10 rep	242485632.349031	242480715	242475661	0.00	0.00
Matrix Multiply – 1K rep	54953734	54848641	54738944	0.19	0.39
Matrix Multiply – 100K rep	543469294122	543458784604	543447815342	0.00	0.00
Prime Generator – 1M	120	100	60	16.67	50.00
Prime Generator – 100M	176	156	116	11.36	34.09
Wave Equation	25050	21019	21487	16.09	14.22

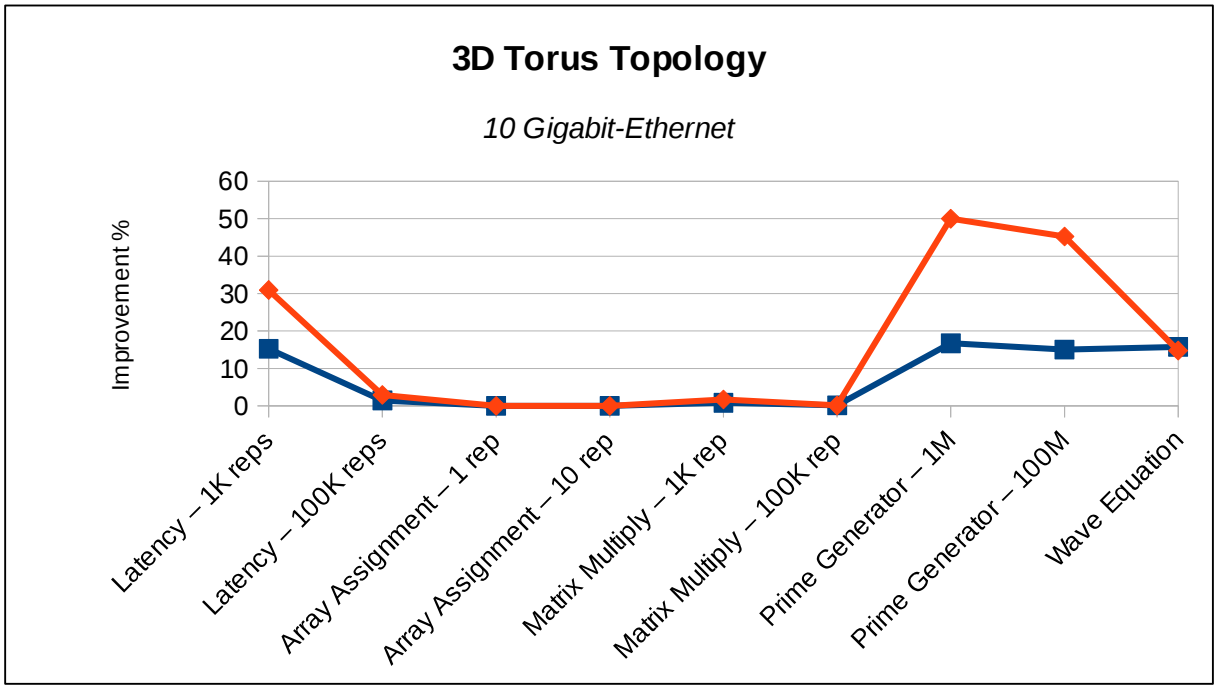
Temps d'execució entre topologies 2D Torus – Connexions InfiniBand EDR

6.6.2. Gràfiques comparatives

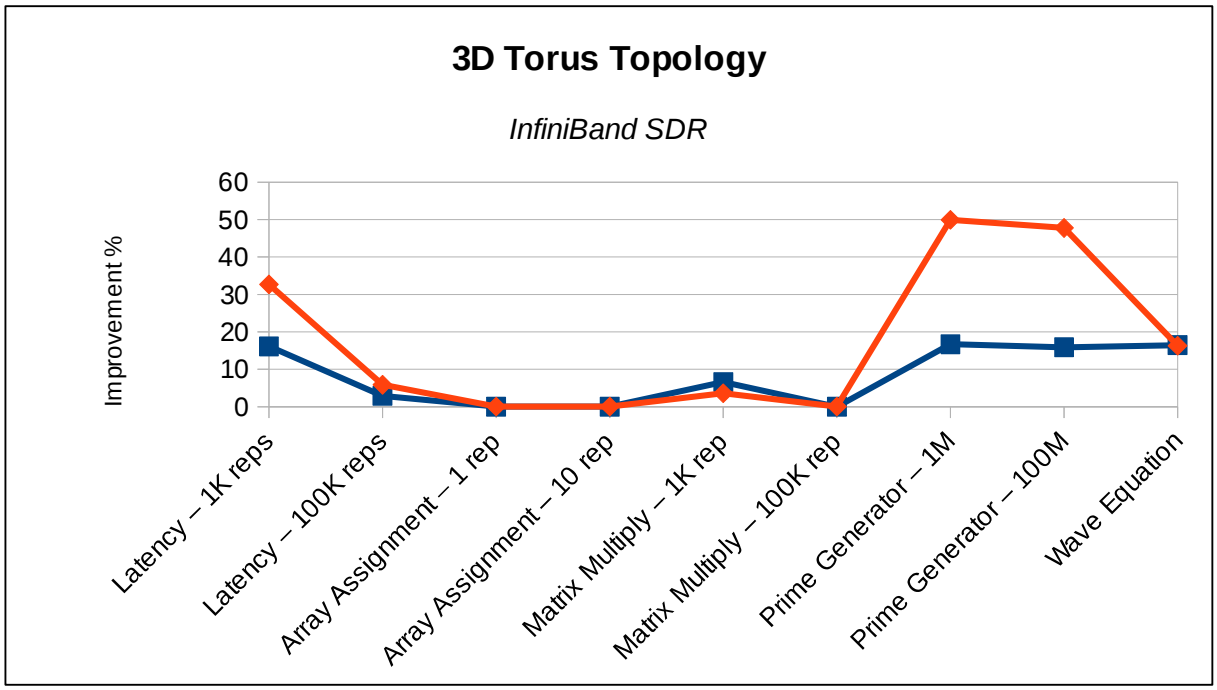
Gràfiques comparatives per cada tecnologia d'interconnexió on es mostra el percentatge de millora que aporta les modificacions fetes sobre la topologia clàssica per tal d'augmentar el rendiment d'aquesta.



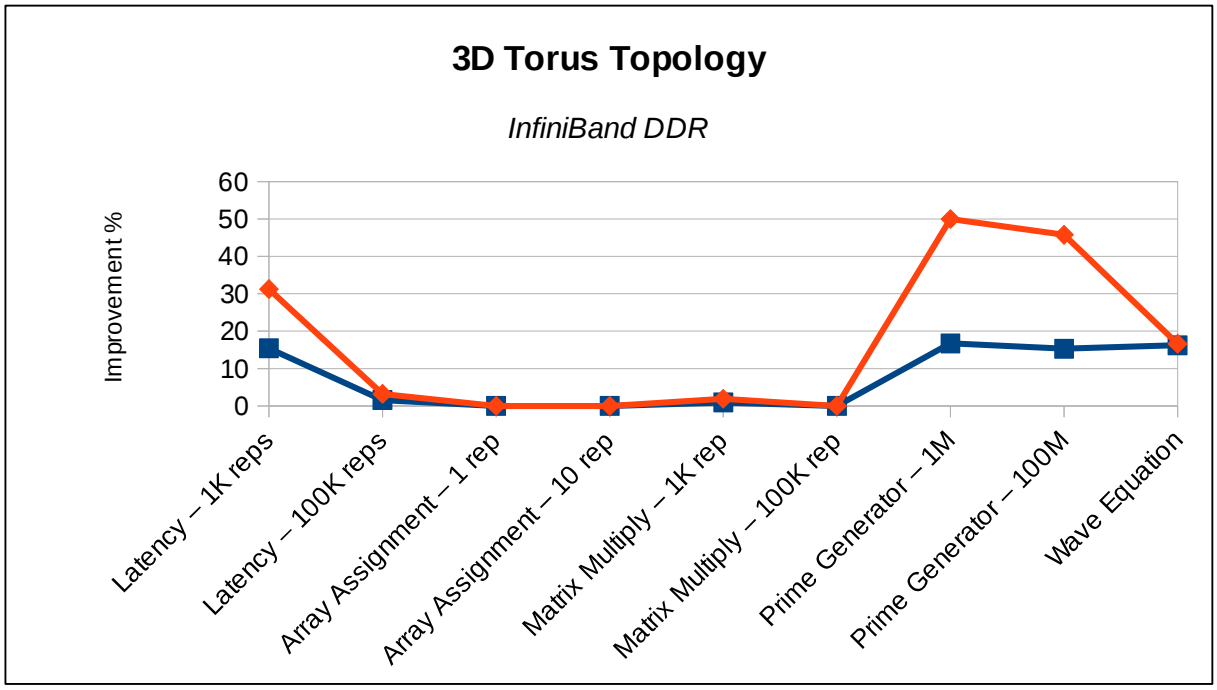
Gràfica amb percentatge de millora – Tecnologia Gigabit-Ethernet



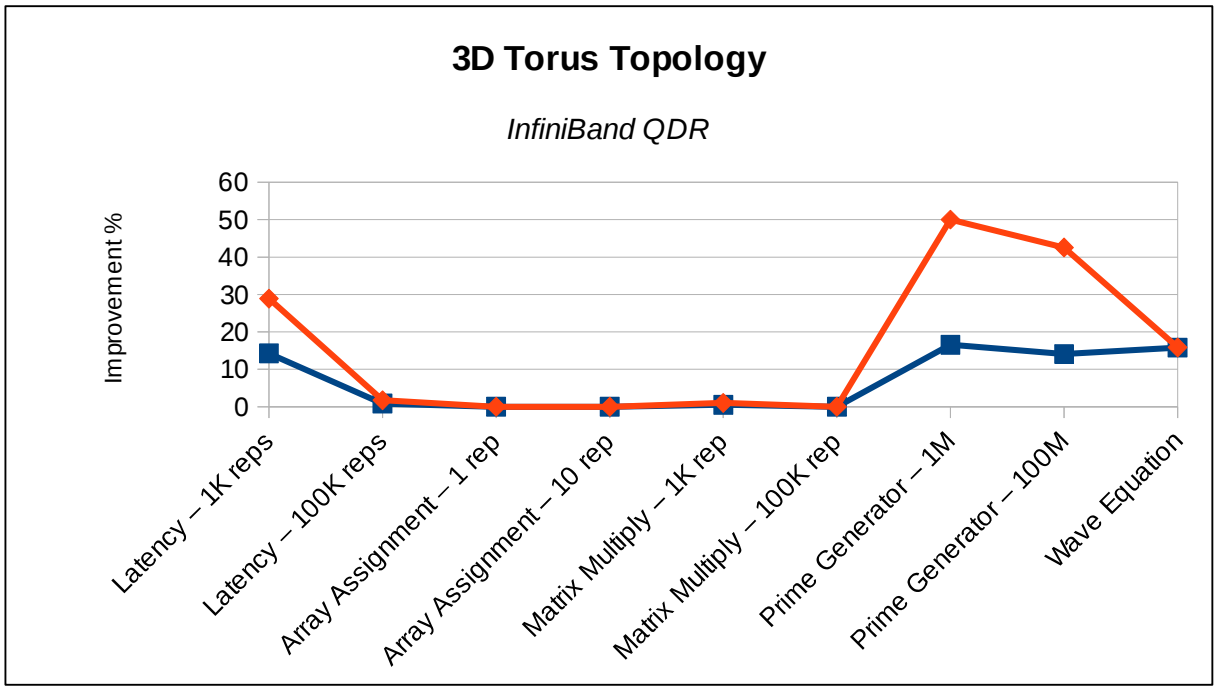
Gràfica amb percentatge de millora – Tecnologia 10 Gigabit-Ethernet



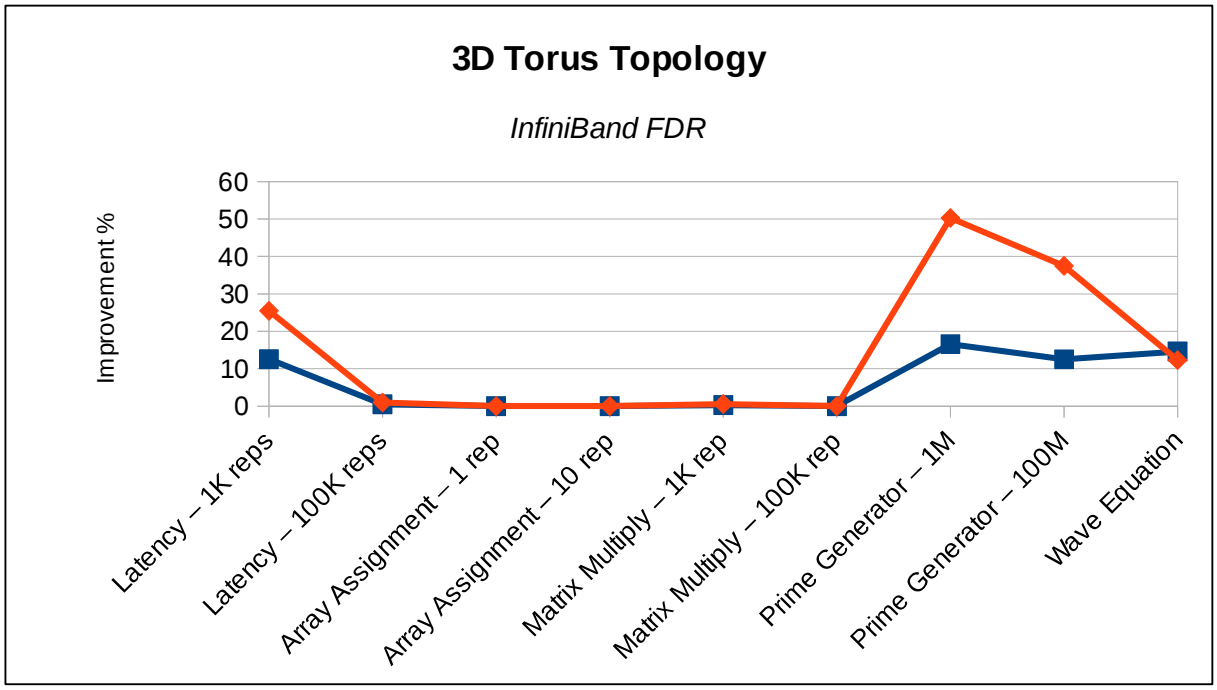
Gràfica amb percentatge de millora – Tecnologia InfiniBand SDR



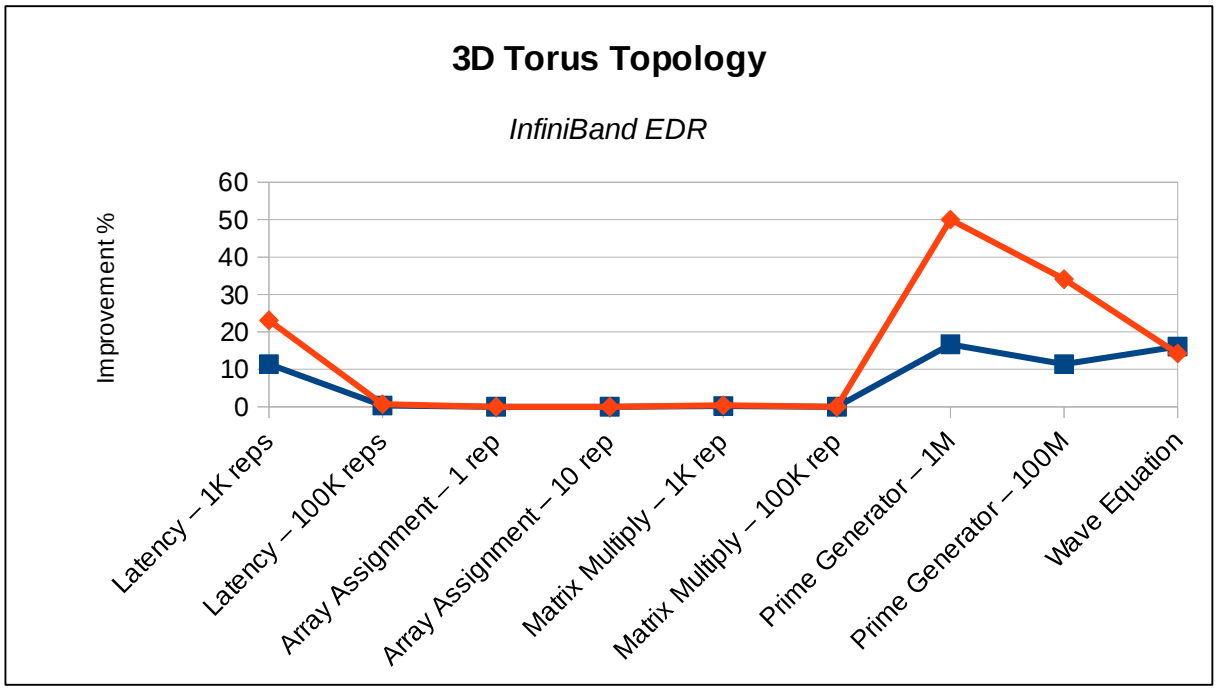
Gràfica amb percentatge de millora – Tecnologia InfiniBand DDR



Gràfica amb percentatge de millora – Tecnologia InfiniBand QDR



Gràfica amb percentatge de millora – Tecnologia InfiniBand FDR



Gràfica amb percentatge de millora – Tecnologia InfiniBand EDR

7. Conclusions

7.1. Millores en les tecnologies d'interconnexió

La primera de les millores proposades, per la tecnologia 10 Gigabit Ethernet, relacionada amb la incorporació de tecnologia SFP+, ens ha donat un resultats molt similar per les tres topologies.

Per contra, i si mirem cada un dels tests MPI al detall, es pot apreciar uns resultats molt dispars, segons la natura del test fer servir. Hi ha una sèrie de tests MPI on la millora de rendiment és insignificant i, de forma contrària, amb els tests de generació de nombres primers i equació d'ona, l'augment de rendiment i reducció del temps d'execució del test ha sigut espectacular, reduint molt significativament els temps.

Per tant, sobre aquesta millora proposada, es pot concloure que dependria molt del tipus de programari MPI al qual es vol destinar el sistema HPC, ja que la millora en el rendiment final no és homogènia i depèn de la natura del test.

Amb aquesta millora només es redueix el temps de latència en els enllaços, atès que aquesta és la millora que dóna la tecnologia SFP+, però mantenint la velocitat de transmissió.

Aquest fet també ens fa concloure que hi ha tests MPI on la latència té més repercussió en els temps d'execució. Més concretament, amb la generació de nombres primers i l'equació d'ona.

La segona millora proposada ha sigut adoptar la tecnologia 40/100 Gigabit Ethernet per les tres topologies fetes servir, tant per les connexions dels nodes com les connexions troncals entre switches.

En aquest cas, la topologia Fat Tree és la que més es beneficia en la reducció de temps, donant un resultats molt interessants per tots els tests MPI i obtenint una millora d'un 75% en el pitjor dels cassos.

D'igual manera, les dues topologies Torus també reben una millora significativa de rendiment, però menys homogènia, ja que hi ha varis tests on l'augment no és tant significatiu però, de forma contrària, amb la generació de nombres primers i equació d'ona, la reducció de temps necessari per completar el test si és molt significativa, ja que es redueix molt considerablement aquest.

En aquest cas passa de forma similar a la primera millora, on la natura del test MPI determina si l'evolució tecnològica en les connexions aporta realment unes millores significatives a nivell de temps d'execució.

La darrera de les millores ha sigut avaluar el rendiment que proporcionarà la pròxima evolució en la família InfiniBand, amb el futur protocol NDR.

Amb la topologia Fat Tree, la millora de rendiment en la reducció de temps és significativa, donat l'augment de la velocitat de transmissió i reducció de latència que proporcionarà aquesta propera evolució. En tots els tests MPI ha proporcionat una bona reducció de temps i es podria justificar una evolució cap a aquesta, camí de la propera generació de superordinadors (Exascale).

Per contra, a les topologies Torus, torna a passar el mateix que amb les millores anteriors. Hi ha una sèrie de tests MPI on l'evolució no es veu justificada però, per contra, amb altres si que es veu reflectida aquesta millora tecnològica.

7.2. Millores topològiques

La primera millora aplicada sobre les topologies ha sigut equilibrar la topologia Fat Tree amb connexions InfiniBand, per tal que hi hagués una relació 1:1 entre els dos nivells (nodes/switches edge i switches edge/switch core).

El fet d'equilibrar l'arbre i passar d'una connexió InfiniBand 12x a cinc connexions d'aquest tipus no ha millorat en res el rendiment, aportant un resultats molt pobres i que no justificarien en cap cas l'augment d'enllaços.

L'augment del rendiment mai ha arribat al 1%, sempre quedant per sota d'aquest valor, cosa molt decebedora, ja que es pensava que augmentant el flux als troncals, augmentaria el rendiment del sistema.

Per tant, i pels tests MPI analitzats, el flux que proporciona agregacions tipus 12x als troncals entre switches seria més que suficient per a adoptar-les a un hipotètic sistema final.

La següent millora topològica ha sigut relaciona amb un augment de connexions entre nodes per l'arquitectura 2D Torus, afegint 98 noves connexions i avaluant com aquestes milloren el rendiment del sistema.

Els resultats han sigut molt similars, tant amb les tecnologies Ethernet com InfiniBand, mostrant un augment de rendiment significatiu només amb un test MPI (generació de nombres primers). La resta de tests MPI no es beneficien d'aquesta augment de connexions i, per tant, la conclusió que es podria treure és que hauria d'analitzar-se més al detall amb quin tipus de programari MPI es vol treballar, atès que augmentar els enllaços entre nodes només es pot justificar amb un dels tests MPI utilitzats.

La darrera de les millores fetes ha sigut sobre la topologia 3D Torus, en dues fases, augmentant les connexions entre nodes: primer afegint connexions internes al cub entre els nodes i, en segona fase, afegint connexions extres a la part exterior del cub.

Per totes les tecnologies de connexió (Ethernet e InfiniBand) els resultats han sigut similar als cassos anteriors, on els tests MPI de generació de primers i l'equació d'ona si es veuen millorats però no així amb la resta, cosa que fa difícil justificar aquest augment de connexions a la topologia.

7.3. Conclusions generals

Seguidament fem un resum general de les conclusions obtingudes:

- La millora en les tecnologies d'interconnexió augmenta el rendiment, sobretot amb la topologia Fat Tree, però en menor mesura amb les topologies Torus.
- Segons la natura del test MPI, els valors de latència i velocitat de transmissió tenen major o menor rellevància. No es pot generalitzar amb els valor obtinguts.
- La natura del test MPI determina si aquestes millores es veuen reflectides o no. No es pot generalitzar i caldria estudiar a quin tipus de tests es dedicarà el sistema HPC.
- Les agregacions InfiniBand 12x són més que suficients per connexions troncals amb la topologia Fat Tree per als nostres tests MPI. Equilibrar l'arbre buscant una relació 1:1 no millora el rendiment en els temps d'execució.
- Augmentar el nombre d'enllaços amb la topologia 2D Torus no garanteix un millor rendiment, ja que, i segons les nostres simulacions, només es veu millorat en un dels tests MPI (*Prime Generator*). Per tant, la natura de les execucions MPI determina si aquesta millora té sentit aplicar-la.
- Amb l'augment de connexions amb la topologia 3D Torus passa una cosa semblant: depèn del test MPI. Amb la generació de nombres primers i l'equació d'ona si es veu reduït considerablement el temps d'execució però, per contra, amb els altres tests aquesta millora no té sentit.

7.4. Treball futur

Algunes de les possibilitats que es podrien estudiar en un futur amb els diferents conceptes i programari vistos durant aquest treball.

Tecnologies d'interconnexió:

- Avaluar el rendiment amb connexions passades, per veure l'evolució històrica dins d'aquest camp: Ethernet, Fast Ethernet, Token Ring, Myrinet, etc.
- Avaluar rendiment de futures connexions, encara no presents: 400 Gigabit Ethernet.
- Fat Tree mixte: 10 Gigabit Ethernet per als nodes e InfiniBand per les connexions troncals.

Topologies:

- Noves possibilitats amb Fat Tree: augmentar/reduir el nombre de switches Edge, augmentar el nombre de switches Core, etc.
- Millores sobre 2D Torus: més connexions entre nodes, connexions de major velocitat entre diferents punts de la malla, etc.
- Millores sobre 3D Torus: més connexions dins del cub, connexions de major velocitat entre diferents punts del cub, etc.
- Avaluar sistemes hypercubs diferents: 4D Torus, 5D Torus.
- Augmentar topologia 2D Torus: malla 5 x 5 o superior.
- Escalar 2D Torus: desequilibrar la malla i avaluar impacte en el rendiment.
- Escalar 3D Torus: desequilibrar el cub i avaluar impacte en el rendiment.

Nodes computacionals:

- Augmentar el nombre de nodes al sistema.
- Augmentar nombre de nuclis per node.
- Augmentar rendiment de càlcul dels nodes (FLOPS).
- Computar amb targetes gràfiques (GPGPU).

Programari MPI:

- Definir nous tests per avaluar rendiments.
- Adaptar tests existents a SimGrid: NAS Parallel Benchmarks, Intel MPI Benchmarks, OMB MPI Tests, SPEC MPI 2007, etc.

8. Glossari

- High Performance Computing (HPC)

Terme anglès per identificar aquelles aplicacions i entorns computacionals d'alt rendiment, utilitzats per recerca o tasques específiques que requereixen càlcul intensiu fent ús de maquinari d'altes prestacions (clusters i superordinadors).

- Node

Element de xarxa final, encarregat de rebre unes directives, tractar-les i retornar el resultat obtingut.

- Encaminador

Element de xarxa encarregat d'interconnectar els diferents nodes, per tal que pugui haver comunicació entre ells.

- Enllaç

Camí d'interconnexió entre dos elements a una xarxa.

- Velocitat de transmissió

Quantitat de dades transmeses per unitat de temps.

- Latència

Temps necessari per tal que un bit viatgi d'un punt de la xarxa a un altre.

- Message Passing Interface (MPI)

Conjunt de directives per programació paral·lela en entorns distribuïts, fent servir el llenguatge C.

- SimGrid

Programari per simular entorns distribuïts heterogenis, com Grids, P2P o Cloud, fent servir programari distribuït, com MPI.

- Fat Tree

Topologia de xarxa on els diferents elements queden distribuïts formant un arbre, on les fulles són els nodes computacionals i les arrels, els encaminador de xarxa.

- 2D Torus

Topologia de xarxa, sense fer servir elements d'encaminament addicionals, on els diferents nodes computacionals queden distribuïts formant una malla e interconnectats entre ells.

- 3D Torus

Topologia de xarxa, sense fer servir elements d'encaminament addicionals, on els diferents nodes computacionals queden distribuïts formant un cub tridimensional e interconnectats entre ells.

- Ethernet

Tecnologia d'interconnexió de xarxa molt utilitzada i que s'ha convertit en l'estàndard actualment, tant en entorns computacionals d'alt rendiment com en infraestructures IT convencionals.

- InfiniBand

Tecnologia d'interconnexió de xarxa, desenvolupada per entorns d'alt rendiment, que ofereix una alta velocitat de transmissió i una baixa latència.

- FLOPS

Unitat de mesura computacional i que s'utilitza per calcular el rendiment de càlcul de processadors i sistemes informàtics. Dóna el nombre d'operacions en coma flotant que es poden computar per cada segon de temps.

9. Bibliografia

Wikipedia, the free encyclopedia: http://en.wikipedia.org
Octubre / Novembre 2015

ClusterDesign.org: http://clusterdesign.org
Octubre / Novembre 2015

Brief History of InfiniBand:
https://blogs.oracle.com/RandomDude/entry/history_hype_to_pragmatism
Octubre 2015

InfiniBand Essentials every HPC Expert must know:
http://es.slideshare.net/mellanox/1-mellanox
Octubre 2015

Baera, Bjorn. *"Benefits of Deploying SFP+ Fiber vs. 10GBase-T"*
http://www.datacenterknowledge.com/archives/2012/11/27/data-center-infrastructure-benefits-of-deploying-sfp-fiber-vs-10gbase-t/
Octubre 2015

Dennis Abts & John Kim . *High Performance Datacenter Networks: Architectures, Algorithms, and Opportunities.* Morgan & Claypool Publishers, 2011. 978-1608454020

Al-Fares, Loukissas, Vahdat, "*A Scalable, Commodity Data Center Network Architecture*" , Proc. of ACM SIGCOMM '08, 38(4):63-74, Oct. 2008.

Mellanox Corp. "*Deploying HPC Cluster with Mellanox InfiniBand Interconnect Solutions ".* Juny 2014.

Sebastian Kalcher . "*Don't forget the 'Fabric'. The Role of High-Bandwidth, Low-Latency Interconnects in High Performance Clusters* ". ADTECH Global.
Octubre 2015

Swamy N. Kandadai & Xinghong He. "*Performance of HPC Applications over InfiniBand, 10 Gb and 1 Gb Ethernet* ". IBM Corp.

Pavan Balaji. "*Network Architecture Trends*". Argonne National Laboratory. ATPESC Workshop. Març 2015.

Brice Goglin, Joshua Hursey & Jeffrey M. Squyres. "*netloc: Towards a Comprehensive View of the HPC System Topology* " . Juny 2014.
https://hal.inria.fr/hal-01010599

HyperTransport Consortium . "*Why Torus-Based Clusters?* ". 2011
http://www.hypertransport.org/docs/uploads/Why_Torus_Main.pdf

10. Annexos

10.1. Codi font del programari MPI

10.1.1. Test de latència

```
/*********************************************************/
/*                                                       */
/*    Programari MPI                                     */
/*    Nom del programari:   Latency                      */
/*    Descripció:   Test de latència entre tots el enllaços  */
/*                                                       */
/*    Estudiant:   José Antonio Martín Pérez             */
/*    email:   jmartinperez1@uoc.edu  /  jamartin@protonmail.ch  */
/*                                                       */
/*********************************************************/


#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <time.h>
#define NUMBER_REPS 1000
#define NODES           64


int main (int argc, char *argv[])
{
int reps,                   /* nombre de repeticions  */
    tag,                    /* MPI message tag parameter  */
    numtasks,               /* nombre de tasques MPI  */
    rank,                   /* Identificador MPI  */
    dest, source,           /* font/destí  */
    avgT,                   /* temps promig per repetició (microseconds)  */
    rc,                     /* codi retorn  */
    i,
    n;
double T1, T2, T3, T4,      /* inici/fi  temps per repeticio */
    sumT,                   /* suma de totes les execucions */
    total_time,
    deltaT;                 /* temps per una repeticio */
char msg;                   /* buffer que conte missatge de 1 byte */
MPI_Status status;          /* rutina recepcio parametre MPI */

MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
MPI_Barrier(MPI_COMM_WORLD);

sumT = 0;
msg = 'x';
tag = 1;
reps = NUMBER_REPS;


if (rank == 0) {
```

```
   printf("Inici del test de latencia:  Nombre de reps = %d.\n", reps);
   printf("*****************************************\n");
   printf("Rep#       T1               T2            deltaT\n");
   dest = 1;
   source = 1;

T3 = MPI_Wtime();

   for (n = 1; n <= reps; n++) {
      T1 = MPI_Wtime();     /* Temps inicial */

      for (i = 1; i < NODES; i++)
      {
         rc = MPI_Send(&msg, 1, MPI_BYTE, i, tag, MPI_COMM_WORLD);
         if (rc != MPI_SUCCESS) {
            printf("Send error in task 0!\n");
            MPI_Abort(MPI_COMM_WORLD, rc);
            exit(1);
         }
       }

      for (i = 1; i < NODES; i++)
      {

         rc = MPI_Recv(&msg, 1, MPI_BYTE, i, tag, MPI_COMM_WORLD,
                    &status);
         if (rc != MPI_SUCCESS) {
            printf("Error recepcio en tasca 0!\n");
            MPI_Abort(MPI_COMM_WORLD, rc);
            exit(1);
         }
      }

      T2 = MPI_Wtime();     /* Temps final */
      deltaT = T2 - T1;
      sumT += deltaT;
   }

   T4 = MPI_Wtime();
   total_time = T4 - T3;
   printf("\n\n*** Temps total = %8.8f microsegonds\n\n", total_time);
}


else if (rank > 0) {

   dest = 0;
   source = 0;
   for (n = 1; n <= reps; n++) {
      rc = MPI_Recv(&msg, 1, MPI_BYTE, source, tag, MPI_COMM_WORLD,
                    &status);
      if (rc != MPI_SUCCESS) {
         printf("Error en tasca 1!\n");
         MPI_Abort(MPI_COMM_WORLD, rc);
         exit(1);
      }
      rc = MPI_Send(&msg, 1, MPI_BYTE, dest, tag, MPI_COMM_WORLD);
      if (rc != MPI_SUCCESS) {
         printf("Error enviament en tasca 1!\n");
         MPI_Abort(MPI_COMM_WORLD, rc);
         exit(1);
      }
   }
```

```
}

MPI_Finalize();

}
```

10.1.2. Array Assignment

```
/*********************************************************/
/*********************************************************/
/*                                                       */
/*    Programari MPI                                     */
/*    Nom del programari:   Array Assignment             */
/*    Descripció:   Descomposició d'un vector entre nodes */
/*                  1 repetició                          */
/*                                                       */
/*    Estudiant:   José Antonio Martín Pérez             */
/*    email:   jmartinperez1@uoc.edu  /  jamartin@protonmail.ch */
/*                                                       */
/*********************************************************/
/*********************************************************/


#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define  ARRAYSIZE 960000000     /* Mida de l'array a tractar */
#define  MASTER    0

#define ROUNDS  1       /* Nombre de repeticions */


float  data[ARRAYSIZE];

int main (int argc, char *argv[])
{
int   numtasks, taskid, rc, dest, offset, i, j, tag1,
      tag2, source, chunksize;
int index;
float mysum, sum;
float update(int myoffset, int chunk, int myid);
MPI_Status status;

double T1, T2, total_time;

MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &numtasks);
if (numtasks % 4 != 0) {
   printf("Es requereix un nombre de tasques multiple de 4. Sortint...\n");
   MPI_Abort(MPI_COMM_WORLD, rc);
   exit(0);
   }
MPI_Comm_rank(MPI_COMM_WORLD,&taskid);
chunksize = (ARRAYSIZE / numtasks);
tag2 = 1;
tag1 = 2;


/****************/
/*     Node 0     */
/****************/

if (taskid == MASTER){

  T1 = MPI_Wtime();

  for (index=0; index<ROUNDS; index++) {
```

```
  /* Inicialització del vector */
  sum = 0;
  for(i=0; i<ARRAYSIZE; i++) {
    data[i] =  i * 1.0;
    sum = sum + data[i];
    }

   /* Enviament a cada node de la seva porció del vector. Node 0 es queda la
primera d'aquestes */
  offset = chunksize;
  for (dest=1; dest<numtasks; dest++) {
    MPI_Send(&offset, 1, MPI_INT, dest, tag1, MPI_COMM_WORLD);
    MPI_Send(&data[offset], chunksize, MPI_FLOAT, dest, tag2, MPI_COMM_WORLD);
    offset = offset + chunksize;
    }

  /* Node 0 fa la seva part del treball */
  offset = 0;
  mysum = update(offset, chunksize, taskid);

  /* Rebem resultat de cada node */
  for (i=1; i<numtasks; i++) {
    source = i;
    MPI_Recv(&offset, 1, MPI_INT, source, tag1, MPI_COMM_WORLD, &status);
    MPI_Recv(&data[offset], chunksize, MPI_FLOAT, source, tag2,
      MPI_COMM_WORLD, &status);
    }

  /* Rebem suma final */
  MPI_Reduce(&mysum, &sum, 1, MPI_FLOAT, MPI_SUM, MASTER, MPI_COMM_WORLD);
  offset = 0;
  }

  T2=MPI_Wtime();
  total_time = T2 - T1;
  printf("\n\n Temps total:  %8.8f \n\n" , total_time );

}


/************************/
/*     Resta de nodes     */
/************************/

if (taskid > MASTER) {

 for (index=0; index<ROUNDS; index++) {

  /* Cada node rep la seva porció del vector */
  source = MASTER;
  MPI_Recv(&offset, 1, MPI_INT, source, tag1, MPI_COMM_WORLD, &status);
  MPI_Recv(&data[offset], chunksize, MPI_FLOAT, source, tag2,
    MPI_COMM_WORLD, &status);

  mysum = update(offset, chunksize, taskid);

  /* Enviament del resultat computat al node 0 */
  dest = MASTER;
  MPI_Send(&offset, 1, MPI_INT, dest, tag1, MPI_COMM_WORLD);
  MPI_Send(&data[offset], chunksize, MPI_FLOAT, MASTER, tag2, MPI_COMM_WORLD);

  MPI_Reduce(&mysum, &sum, 1, MPI_FLOAT, MPI_SUM, MASTER, MPI_COMM_WORLD);

  }
```

```
}

MPI_Finalize();

}



//  Tasca per computar la suma valors de la porció del vector rebuda
//  ----------------------------------------------------------------

float update(int myoffset, int chunk, int myid) {
  int i;
  float mysum;
  mysum = 0;
  for(i=myoffset; i < myoffset + chunk; i++) {
    data[i] = data[i] + i * 1.0;
    mysum = mysum + data[i];
    }
  return(mysum);
}
```

10.1.3. Matrix Multiply

```
/*********************************************************/
/*********************************************************/
/*                                                       */
/*    Programari MPI                                     */
/*    Nom del programari:   Matrix Multiply              */
/*    Descripció:   Multiplicació de dues matrius grans  */
/*                  1.000 repeticions                    */
/*                                                       */
/*    Estudiant:   José Antonio Martín Pérez             */
/*    email:   jmartinperez1@uoc.edu  /  jamartin@protonmail.ch   */
/*                                                       */
/*********************************************************/
/*********************************************************/


#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define NRA 248                /* Nombre files matriu A */
#define NCA 30                 /* Nombre columnes matriu A */
#define NCB 14                 /* Nombre columnes matriu B */
#define MASTER 0               /* Identificador primera tasca (0) */
#define FROM_MASTER 1
#define FROM_WORKER 2

#define ROUNDS 1000            /*  Nombre de repeticions */


int main (int argc, char *argv[])
{
int numtasks,              /* Nombre de tasques */
    taskid,                /* Identificador tasca */
    numworkers,            /*  */
    source,                /* task id of message source */
    dest,                  /* task id of message destination */
    mtype,                 /* message type */
    rows,                  /* rows of matrix A sent to each worker */
    averow, extra, offset, /* used to determine rows sent to each worker */
    i, j, k, rc;           /* misc */
double  T1, T2, total_time,
        a[NRA][NCA],           /* matrix A to be multiplied */
    b[NCA][NCB],           /* matrix B to be multiplied */
    c[NRA][NCB];           /* result matrix C */
MPI_Status status;

MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD,&taskid);
MPI_Comm_size(MPI_COMM_WORLD,&numtasks);
if (numtasks < 2 ) {
  printf("Need at least two MPI tasks. Quitting...\n");
  MPI_Abort(MPI_COMM_WORLD, rc);
  exit(1);
  }
numworkers = numtasks-1;


/*******************   Node 0 - Inici del procés   *******************/
   if (taskid == MASTER)
   {
```

```
    T1 = MPI_Wtime();   // Marca temps inicial

    int index;
    for (index=0; index<ROUNDS; index++)  {

      // Inicialització de les matrius
      for (i=0; i<NRA; i++)
         for (j=0; j<NCA; j++)
            a[i][j]= i+j;
      for (i=0; i<NCA; i++)
         for (j=0; j<NCB; j++)
            b[i][j]= i*j;

      /* Enviament de les dades de la matriu a cada node */
      averow = NRA/numworkers;
      extra = NRA%numworkers;
      offset = 0;
      mtype = FROM_MASTER;
      for (dest=1; dest<=numworkers; dest++)
      {
         rows = (dest <= extra) ? averow+1 : averow;
         MPI_Send(&offset, 1, MPI_INT, dest, mtype, MPI_COMM_WORLD);
         MPI_Send(&rows, 1, MPI_INT, dest, mtype, MPI_COMM_WORLD);
         MPI_Send(&a[offset][0], rows*NCA, MPI_DOUBLE, dest, mtype,
                  MPI_COMM_WORLD);
         MPI_Send(&b, NCA*NCB, MPI_DOUBLE, dest, mtype, MPI_COMM_WORLD);
         offset = offset + rows;
      }

      /* Rebem les dades de cada node */
      mtype = FROM_WORKER;
      for (i=1; i<=numworkers; i++)
      {
         source = i;
               MPI_Recv(&offset,  1,  MPI_INT,  source,  mtype,  MPI_COMM_WORLD,
&status);
         MPI_Recv(&rows, 1, MPI_INT, source, mtype, MPI_COMM_WORLD, &status);
         MPI_Recv(&c[offset][0], rows*NCB, MPI_DOUBLE, source, mtype,
                  MPI_COMM_WORLD, &status);
      }

    }

      T2=MPI_Wtime();   // Marca temps final
      total_time = T2 - T1;
      printf("\n\n Temps total:  %8.8f \n\n" , total_time );

   }


/********************   Resta de nodes   ********************/
   if (taskid > MASTER)
   {
    int index;
    for (index=0; index<ROUNDS; index++)  {

      // Recepció de dades a computar
      mtype = FROM_MASTER;
      MPI_Recv(&offset, 1, MPI_INT, MASTER, mtype, MPI_COMM_WORLD, &status);
      MPI_Recv(&rows, 1, MPI_INT, MASTER, mtype, MPI_COMM_WORLD, &status);
          MPI_Recv(&a,  rows*NCA,  MPI_DOUBLE,  MASTER,  mtype,  MPI_COMM_WORLD,
&status);
```

```
          MPI_Recv(&b,  NCA*NCB,  MPI_DOUBLE,  MASTER,  mtype,  MPI_COMM_WORLD,
&status);

      // Comput de la seva part de la matriu
      for (k=0; k<NCB; k++)
         for (i=0; i<rows; i++)
         {
            c[i][k] = 0.0;
            for (j=0; j<NCA; j++)
               c[i][k] = c[i][k] + a[i][j] * b[j][k];
         }

      // Enviament de les dades computades al node 0
      mtype = FROM_WORKER;
      MPI_Send(&offset, 1, MPI_INT, MASTER, mtype, MPI_COMM_WORLD);
      MPI_Send(&rows, 1, MPI_INT, MASTER, mtype, MPI_COMM_WORLD);
      MPI_Send(&c, rows*NCB, MPI_DOUBLE, MASTER, mtype, MPI_COMM_WORLD);
    }
}

   MPI_Finalize();

}
```

10.1.4. Prime Generator

```
/*********************************************************/
/*********************************************************/
/*                                                       */
/*    Programari MPI                                     */
/*    Nom del programari:   Prime Generator              */
/*    Descripció:   Generador de nombres primers         */
/*                  1.000.000 nombres primers a generar  */
/*                                                       */
/*    Estudiant:   José Antonio Martín Pérez             */
/*    email:   jmartinperez1@uoc.edu  /  jamartin@protonmail.ch   */
/*                                                       */
/*********************************************************/
/*********************************************************/


#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define LIMIT     1000000      /*  Nombre de primers a generar  */
#define FIRST     0            /*  Identificador de la tasca 0  */

int isprime(int n) {
int i,squareroot;
if (n>10) {
   squareroot = (int) sqrt(n);
   for (i=3; i<=squareroot; i=i+2)
      if ((n%i)==0)
         return 0;
   return 1;
   }
else
   return 0;
}


int main (int argc, char *argv[])
{
int   ntasks,               /* Nombre total de tasques */
      rank,                 /* Identificador de tasca */
      n,                    /* Variable per a bucles */
      pc,                   /* Comptador de nombres primers */
      pcsum,                /* Nombre de primers trobats per totes les tasques
*/
      foundone,             /* Primer més recenment trobat */
      maxprime,             /* Primer més gran trobat */
      mystart,              /* On comença a calcular cada tasca */
      stride;

double start_time,end_time;

MPI_Init(&argc,&argv);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
MPI_Comm_size(MPI_COMM_WORLD,&ntasks);
if (((ntasks%2) !=0) || ((LIMIT%ntasks) !=0)) {
   printf("Aquesta aplicacio requereix un nombre de tasques parell.\n");
   MPI_Finalize();
   exit(0);
}
```

```
start_time = MPI_Wtime();     /* Initialize start time */
mystart = (rank*2)+1;         /* Find my starting point - must be odd number */
stride = ntasks*2;            /* Determine stride, skipping even numbers */
pc=0;                         /* Initialize prime counter */
foundone = 0;                 /* Initialize */

/********************  Tasca 0 - Inicia el procés  ********************/
if (rank == FIRST) {
   pc = 4;
   for (n=mystart; n<=LIMIT; n=n+stride) {
      if (isprime(n)) {
         pc++;
         foundone = n;
      }
   }
   MPI_Reduce(&pc,&pcsum,1,MPI_INT,MPI_SUM,FIRST,MPI_COMM_WORLD);
   MPI_Reduce(&foundone,&maxprime,1,MPI_INT,MPI_MAX,FIRST,MPI_COMM_WORLD);
   end_time=MPI_Wtime();
   printf("Temps total: %.2lf segonds\n",(end_time-start_time)*10);
}


/********************  Resta de tasques  ************************/
if (rank > FIRST) {
   for (n=mystart; n<=LIMIT; n=n+stride) {
      if (isprime(n)) {
         pc++;
         foundone = n;
      }
   }
   MPI_Reduce(&pc,&pcsum,1,MPI_INT,MPI_SUM,FIRST,MPI_COMM_WORLD);
   MPI_Reduce(&foundone,&maxprime,1,MPI_INT,MPI_MAX,FIRST,MPI_COMM_WORLD);
   }

MPI_Finalize();

}
```

10.1.5. Wave Equation

```
/*********************************************************/
/*********************************************************/
/*                                                       */
/*    Programari MPI                                     */
/*    Nom del programari:   Wave Equation                */
/*    Descripció:   Equació propagació ones              */
/*                  https://en.wikipedia.org/wiki/Wave_equation   */
/*                                                       */
/*    Estudiant:   José Antonio Martín Pérez             */
/*    email:   jmartinperez1@uoc.edu  /  jamartin@protonmail.ch   */
/*                                                       */
/*********************************************************/
/*********************************************************/


# include <stdio.h>
# include <stdlib.h>
# include <math.h>
# include <time.h>

# include "mpi.h"

int main ( int argc, char *argv[] );
double *update ( int id, int p, int n_global, int n_local, int nsteps,
  double dt );
void collect ( int id, int p, int n_global, int n_local, int nsteps,
  double dt, double u_local[] );
double dudt ( double x, double t );
double exact ( double x, double t );
void timestamp ( );


int main ( int argc, char *argv[] )
{
  double dt = 0.00125;
  int i_global_hi;
  int i_global_lo;
  int id;
  int n_global = 401;
  int n_local;
  int nsteps = 4000;
  int p;
  double *u1_local;
  double wtime;
/*
  Inicialització procés MPI
*/
  MPI_Init ( &argc, &argv );
  MPI_Comm_rank ( MPI_COMM_WORLD, &id );
  MPI_Comm_size ( MPI_COMM_WORLD, &p );

  if ( id == 0 )
  {
    timestamp ( );
  }

  wtime = MPI_Wtime ( );

/*
  Determinar N_LOCAL
*/
```

```
  i_global_lo = (   id       * ( n_global - 1 ) ) / p;
  i_global_hi = ( ( id + 1 ) * ( n_global - 1 ) ) / p;
  if ( 0 < id )
  {
    i_global_lo = i_global_lo - 1;
  }
  n_local = i_global_hi + 1 - i_global_lo;

/*
  Actualitzar valors N_LOCAL
*/
  u1_local = update ( id, p, n_global, n_local, nsteps, dt );

/*
  Recol·lecció valors en un array
*/
  collect ( id, p, n_global, n_local, nsteps, dt, u1_local );

/*
  Temps total necessari per fer el càlcul
*/
  wtime = MPI_Wtime ( ) - wtime;
  if ( id == 0 )
  {
    printf ( "\n" );
    printf ( "  Elapsed wallclock time was %g seconds\n", wtime );

  }

/*
  Fi del procés MPI
*/
  MPI_Finalize ( );

/*
  Alliberem memòria
*/
  free ( u1_local );

/*
  Fi
*/
  if ( id == 0 )
  {
    timestamp ( );
  }

  return 0;
}


// Funció UPDATE
// S'avança la solució un determinat nombre de passos.

double *update ( int id, int p, int n_global, int n_local, int nsteps,
  double dt )
{
  double alpha;
  double c;
  double dx;
  int i;
  int i_global;
  int i_global_hi;
  int i_global_lo;
```

```
  int i_local;
  int i_local_hi;
  int i_local_lo;
  int j;
  int ltor = 20;
  int rtol = 10;
  MPI_Status status;
  double t;
  double *u0_local;
  double *u1_local;
  double *u2_local;
  double x;
/*
  Es determina el valor ALPHA.
*/
  c = 1.0;
  dx = 1.0 / ( double ) ( n_global - 1 );
  alpha = c * dt / dx;

  if ( 1.0 <= fabs ( alpha ) )
  {
    if ( id == 0 )
    {
      fprintf ( stderr, "\n" );
      fprintf ( stderr, "UPDATE - Warning!\n" );
      fprintf ( stderr, "  1 <= |ALPHA| = | C * dT / dX |.\n" );
      fprintf ( stderr, "  C = %g\n", c );
      fprintf ( stderr, "  dT = %g\n", dt );
      fprintf ( stderr, "  dX = %g\n", dx );
      fprintf ( stderr, "  ALPHA = %g\n", alpha );
      fprintf ( stderr, "  Computation will not be stable!\n" );
    }
    MPI_Finalize ( );
    exit ( 1 );
  }
/*
  El vector de punts N_GLOBAL s'ha de dividir entre els diferents nodes.
*/
  i_global_lo = (   id       * ( n_global - 1 ) ) / p;
  i_global_hi = ( ( id + 1 ) * ( n_global - 1 ) ) / p;
  if ( 0 < id )
  {
    i_global_lo = i_global_lo - 1;
  }

  i_local_lo = 0;
  i_local_hi = i_global_hi - i_global_lo;

  u0_local = ( double * ) malloc ( n_local * sizeof ( double ) );
  u1_local = ( double * ) malloc ( n_local * sizeof ( double ) );
  u2_local = ( double * ) malloc ( n_local * sizeof ( double ) );

  t = 0.0;
  for ( i_global = i_global_lo; i_global <= i_global_hi; i_global++ )
  {
    x = ( double ) ( i_global ) / ( double ) ( n_global - 1 );
    i_local = i_global - i_global_lo;
    u1_local[i_local] = exact ( x, t );
  }

  for ( i_local = i_local_lo; i_local <= i_local_hi; i_local++ )
  {
    u0_local[i_local] = u1_local[i_local];
  }
```

```
/*
  Agafa nombre de passos (NSTEPS).
*/
  for ( i = 1; i <= nsteps; i++ )
  {
    t = dt * ( double ) i;
/*
  Primer pas: Usem la informació inicial de la derivada.
*/
    if ( i == 1 )
    {
      for ( i_local = i_local_lo + 1; i_local < i_local_hi; i_local++ )
      {
        i_global = i_global_lo + i_local;
        x = ( double ) ( i_global ) / ( double ) ( n_global - 1 );
        u2_local[i_local] =
          +         0.5 * alpha * alpha   * u1_local[i_local-1]
          + ( 1.0 -       alpha * alpha ) * u1_local[i_local]
          +         0.5 * alpha * alpha   * u1_local[i_local+1]
          +                             dt * dudt ( x, t );
      }
    }
/*
  Després del primer pas, es podem fer servir les estimacions.
*/
    else
    {
      for ( i_local = i_local_lo + 1; i_local < i_local_hi; i_local++ )
      {
        u2_local[i_local] =
          +               alpha * alpha   * u1_local[i_local-1]
          + 2.0 * ( 1.0 - alpha * alpha ) * u1_local[i_local]
          +               alpha * alpha   * u1_local[i_local+1]
          -                                 u0_local[i_local];
      }
    }
/*
  Intercanvi d'informació amb el node veí de la part esquerra.
*/
    if ( 0 < id )
    {
      MPI_Send ( &u2_local[i_local_lo+1], 1, MPI_DOUBLE, id - 1, rtol,
        MPI_COMM_WORLD );
      MPI_Recv ( &u2_local[i_local_lo],   1, MPI_DOUBLE, id - 1, ltor,
        MPI_COMM_WORLD, &status );
    }
    else
    {
      x = 0.0;
      u2_local[i_local_lo] = exact ( x, t );
    }
/*
  Intercanvi d'informació amb el node veí de la part dreta.
*/
    if ( id < p - 1 )
    {
      MPI_Send ( &u2_local[i_local_hi-1], 1, MPI_DOUBLE, id + 1, ltor,
        MPI_COMM_WORLD );
      MPI_Recv ( &u2_local[i_local_hi],   1, MPI_DOUBLE, id + 1, rtol,
        MPI_COMM_WORLD, &status );
    }
    else
    {
      x = 1.0;
```

```
      u2_local[i_local_hi] = exact ( x, t );
    }
/*
  Ens movem al següent pas.
*/
    for ( i_local = i_local_lo; i_local <= i_local_hi; i_local++ )
    {
      u0_local[i_local] = u1_local[i_local];
      u1_local[i_local] = u2_local[i_local];
    }
  }
/*
  Alliberem memòria.
*/
  free ( u0_local );
  free ( u2_local );

  return u1_local;
}


// Funció COLLECT
// Recol·lecció de dades i enviament al node mestre

void collect ( int id, int p, int n_global, int n_local, int nsteps,
  double dt, double u_local[] )
{
  int buffer[2];
  int collect1 = 10;
  int collect2 = 20;
  int i;
  int i_global;
  int i_global_hi;
  int i_global_lo;
  int i_local;
  int i_local_hi;
  int i_local_lo;
  int j;
  int n_local2;
  MPI_Status status;
  double t;
  double *u_global;
  double x;

  i_global_lo = (   id       * ( n_global - 1 ) ) / p;
  i_global_hi = ( ( id + 1 ) * ( n_global - 1 ) ) / p;
  if ( 0 < id )
  {
    i_global_lo = i_global_lo - 1;
  }

  i_local_lo = 0;
  i_local_hi = i_global_hi - i_global_lo;
/*
  Node mestre recull resultats al vector U_GLOBAL.
*/
  if ( id == 0 )
  {
/*
  Creació del vector global.
*/
    u_global = ( double * ) malloc ( n_global * sizeof ( double ) );
/*
  Es copia el resultat del node mestre al vector global.
```

```
*/
    for ( i_local = i_local_lo; i_local <= i_local_hi; i_local++ )
    {
      i_global = i_global_lo + i_local - i_local_lo;
      u_global[i_global] = u_local[i_local];
    }
/*
  Es contacta amb cada node
*/
    for ( i = 1; i < p; i++ )
    {
/*
  Missatge "collect1", amb l'índex global i el nombre de valors.
*/
      MPI_Recv ( buffer, 2, MPI_INT, i, collect1, MPI_COMM_WORLD, &status );
      i_global_lo = buffer[0];
      n_local2 = buffer[1];

      if ( i_global_lo < 0 )
      {
        fprintf ( stderr, "  Illegal I_GLOBAL_LO = %d\n", i_global_lo );
        exit ( 1 );
      }
      else if ( n_global <= i_global_lo + n_local2 - 1 )
      {
        fprintf ( stderr, "  Illegal I_GLOBAL_LO + N_LOCAL2 = %d\n",
          i_global_lo + n_local2 );
        exit ( 1 );
      }
/*
  Missatge "collect2", que conté els valors.
*/
      MPI_Recv ( &u_global[i_global_lo], n_local2, MPI_DOUBLE, i, collect2,
        MPI_COMM_WORLD, &status );
    }
/*
  Es mostra el resultat.
*/
    t = dt * ( double ) nsteps;
    free ( u_global );
  }

/*
  Resta de nodes envien els resultats al node mestre (node 0).
*/
  else
  {
/*
  Missatge "collect1", amb l'índex global i el nombre de valors.
*/
    buffer[0] = i_global_lo;
    buffer[1] = n_local;
    MPI_Send ( buffer, 2, MPI_INT, 0, collect1, MPI_COMM_WORLD );
/*
  Missatge "collect2", que conté els valors.
*/
    MPI_Send ( u_local, n_local, MPI_DOUBLE, 0, collect2, MPI_COMM_WORLD );
  }

  return;
}


//  Funció EXACT
```

```
//  Càlcul de la solució exacta.

double exact ( double x, double t )
{
  const double c = 1.0;
  const double pi = 3.141592653589793;
  double value;

  value = sin ( 2.0 * pi * ( x - c * t ) );

  return value;
}


// Funció DUDT
// Evaluació de la derivada parcial.

double dudt ( double x, double t )
{
  const double c = 1.0;
  const double pi = 3.141592653589793;
  double value;

  value = - 2.0 * pi * c * cos ( 2.0 * pi * ( x - c * t ) );

  return value;
}


// Funció TIMESTAMP
// Marca de temps

void timestamp ( )
{
# define TIME_SIZE 40

  static char time_buffer[TIME_SIZE];
  const struct tm *tm;
  time_t now;

  now = time ( NULL );
  tm = localtime ( &now );

  strftime ( time_buffer, TIME_SIZE, "%d %B %Y %I:%M:%S %p", tm );

  printf ( "%s\n", time_buffer );

  return;
# undef TIME_SIZE
}
```