5913-1: Sistema distribuït amb Raspberry Pi's enfocat a docència

Memòria del Projecte Fi de Carrera
d'Enginyeria en Informàtica
realitzat per
Roger Pastor Ortiz
i dirigit per
Juan Carlos Moure Lopez
Bellaterra, 26 de Gener de 2015

Índex

Índex de figures

Capítol 1

1 Introducció

La societat actual està rodejada de computadors i aparells electrònics i el seu nombre no para de créixer. Avui en dia ens fan falta els ordinadors per a una gran majoria d'activitats tant de caire domèstic com empresarial. La capacitat de còmput que tenen els fan ideals per a gran quantitat de tasques, ja siguin tractament d'imatges, càlcul numèric o processament de dades entre moltes d'altres. Els ordinadors són capaços de dur a terme totes aquestes tasques més ràpida i eficientment que una persona. El seu ús, però comporta certs inconvenients. Tant el disseny com el manteniment de qualsevol sistema suposa una despesa econòmica i, com més capaç i potent es vulgui aquest, més elevada és la despesa. Fins fa relativament pocs anys, la llei de Moore[1] es complia cada dos anys sens falta, abaratint els preus dels ordinadors que anaven quedant tecnològicament desfasats. Sembla ser, però, que aquest creixement del hardware està desaccelerant. La tecnologia s'està estancant i és difícil millorar el hardware actual. Per fer-ho s'han de gastar molts diners per aconseguir una millora no massa significativa. Conseqüentment, la línia d'avui en dia va dirigida a la replicació de hardware més que en la millora del mateix.

Sota aquestes premisses apareixia fa uns anys el concepte de *clúster* (principis dels 60). En primera instància, per augmentar la potència dels ordinadors, es pot optar per replicar el hardware dintre d'una mateixa computadora posant diversos components hardware (com processadors) en una sola màquina, però això no és sempre suficient i té certes limitacions. Sobretot gràcies a Internet les dades actuals que s'han de gestionar són de mides gegantines. Són tals que un sol ordinador no podria gestionar-les totes ni en anys tot i tenir diversos mòduls de hardware replicats. Per aquests motius i altres aquí no esmentats, els clústers han esdevingut la solució. Agrupacions de diversos computadors interconnectats, que es poden veure des de l'exterior com un sol sistema ja que, de fet, es comporten com un sol sistema lògic. Hi ha, doncs, una gran varietat d'opcions, cadascuna més o menys adequada a unes demandes o tasques. Tot i així, principalment el que decideix quin sistema s'adquireix és el pressupost. Com més elevat, millor sistema es pot adquirir per realitzar la feina requerida.

Donat aquest context, en aquest projecte s'han escollit plaques Raspberry Pi per muntar un petit clúster domèstic. Tot i tenir la clara desavantatge que les plaques tenen una potència de còmput reduïda (com veurem en la secció d'*estat de l'art*), el seu baix preu les converteix en una alternativa ideal per entrar en el món dels clústers sense requerir una inversió econòmica important. Sota el mateix context, sembla palesa la importància d'uns mínims coneixements de computació paral·lela i de sistemes distribuïts, doncs el futur ens avoca a aquest tipus de programació sobre aquests sistemes. Per això, indirectament, es buscarà augmentar els coneixements de la matèria al llarg d'aquest projecte.

1.1 Objectius

L'objectiu principal d'aquest projecte serà el disseny i implementació d'unes pràctiques acadèmiques que permetin als alumnes adquirir coneixements sobre clústers i programació paral·lela. Per fer-ho es planteja construir un clúster format per plaques Raspberry Pi's i fer-ne ús a l'estudiar la programació paral·lela executada sobre aquest mateix sistema.

[1] Llei de Moore: "http://en.wikipedia.org/wiki/Moore's_law"

Amb la finalitat de complir l'objectiu, es dividirà el projecte en dues parts. La primera consistirà principalment en muntar i posar a punt el sistema format per les Raspberry Pi's i els diferents perifèrics d'entrada/sortida tot buscant adquirir els coneixements necessaris per desenvolupar el contingut de la resta del projecte. La segona part consistirà en plasmar tots els coneixements adquirits en la primera part dissenyant la documentació necessària per a implementar unes pràctiques acadèmiques sobre programació paral·lela en sistemes multinode.

1.1.1 Documentació a generar

En aquest projecte es generarà el material necessari per a realitzar unes pràctiques com les descrites en la secció d'objectius. Aquest material constarà del següent:

- Una guia per a l'estudiant que constarà dels diferents guions de pràctiques i que contindrà un exercici previ proposat als alumnes per a cada sessió.
- Una guia pel professor on s'exposarà l'objectiu de les pràctiques i una breu explicació de les mateixes. Aquesta guia buscarà que la persona que imparteixi les pràctiques sàpiga com enfocar-les i tingui constància dels punts clau, així com de possibles incidències amb les quals els alumnes es puguin topar.

1.2 Estructura de la memòria

La memòria constarà de quatre capítols organitzats per:

- Capítol 2. Anàlisis prèvia i planificació: S'estudia el problema i es presenta una solució a aquest amb la seva planificació.
- Capítol 3. Sistema: S'expliquen els detalls de la solució implementada. També es testeja el sistema obtingut i s'avaluen els resultats dels test.
- Capítol 4. Pràctiques: Concreta com s'han creat les pràctiques. Objectiu principal del projecte.
- Capítol 5. Conclusions: S'avaluen els resultats obtinguts, si s'han assolit els objectius i tenint-ho tot en compte es proposen possibles millores i futures línies de treball

S'ha decidit aquesta organització seguint un esquema bàsic de planificació de projectes. Avaluació corresponent al capítol 2, disseny en el capítol 3 juntament amb testeig. Implementació en el capítol 4 i, finalment, diagnosis al capítol 5.

Capítol 2

2 Anàlisis previ i planificació

Abans d'entrar en matèria, s'ha fet un anàlisis del problema que presenta el projecte i una planificació de la solució amb la qual es pretén abordar. En la primera secció del capítol es descriu breument l'element principal amb el que es treballarà: la placa Raspberry Pi. També es fa un breu resum de l'estat de l'art dels clústers formats per diferents plaques Raspberry Pi. A la següent secció es fa l'estudi de la viabilitat del projecte, i es tanca el capítol amb la planificació del projecte.

2.1 Estat de l'art

Una placa Raspberry Pi és un computador de la mida d'una targeta de crèdit i amb la principal característica que és de baix cost i de baix consum. El hardware de la placa consta d'un SoC (system on a chip) Broadcom (BCM2835) amb un processador ARM (ARM1176JZF-S) que funciona a 700 MHz, una GPU VideoCore VI capaç de reproduir vídeos a 1080p i 512 MB de memòria RAM (model B). No inclou una memòria d'estat sòlid o disc dur, sinó que fa servir una targeta SD per a l'emmagatzematge permanent i per a arrencar el sistema operatiu. El computador pot usar diversos sistemes operatius com per exemple Debian, Fedora, Arch Linux, Risc OS i més. El preu de les Raspberry Pi i el seu baix consum (el model B consumeix entre 700 i 1000 mA), apart de la capacitat d'afegir una gran varietat de perifèrics I/O com una càmera o diferents sensors, les fan ideals per a infinitat de projectes tant diversos con servidors web o de descàrregues, media centers, clústers, o fins i tot projectes tant especialitzats com monitoritzar el transit. Tot i que en un principi van ser creades per a l'educació amb l'objectiu d'apropar els computadors i la programació a totes les edats, el seu ús s'ha escampat a una gran varietat d'àmbits.

Pel que fa a clústers de Raspberry Pi's, s'han creat diversos sistemes, des de connexions de 2 o 4 plaques (nodes) a connexions de fins a 32 nodes, 64 o més. Els resultats, però, sempre són similars. No s'aconsegueix millor rendiment que amb màquines més potents, pel que no suposen una opció viable en el camp dels sistemes multinode. Alguns estudis que podríem destacar són els següents:

(a) Creating a Raspberry Pi-Based Beowulf Cluster [1], per Joshua Kiepert.

(b) Like Magic Apears (40-Node Raspberry Pi) [2] , per David Guill.

(c) Raspberry Pi Supercomputer [3], University of Southampton.

En el primer estudi, una persona realitzant la seva investigació de tesis doctoral es troba que necessita accés a un clúster en el que pugui instal·lar el software que necessiti sense impediments de permisos ni temporals. A més a més vol que sigui un sistema robust que no caigui mai i conseqüentment deixi de donar servei. Que sempre estigui executant l'aplicació que ha implementat. Donat que l'aplicació que vol executar no té un ús intensiu de càlcul, un clúster de Raspberry Pi's li suposa la solució ideal. Tant per preu com per disponibilitat per instal·lar tot el software que vulgui sense demanar permís a un administrador. El segon cas és similar al primer. Degut a la necessitat de treballar amb aplicacions paral·leles, la persona que dur a terme el estudi crea un clúster a base de Raspberry Pi's com a projecte personal, així té una disponibilitat completa del sistema i pot provar les seves aplicacions sense restriccions temporals. Finalment, en el tercer

estudi, uns professors de la universitat de Southampton creen un clúster per permetre als estudiants entendre la programació paral·lela i inspirar-los a que en facin ús, doncs consideren que és un camp en expansió que poc a poc va agafant importància.

La conclusió a la que arriben tots és, a grans trets, que la potència de càlcul en brut que s'obté de muntar aquests sistemes no és comparable amb el rendiment que es pot aconseguir amb d'altres composats per màquines millors que les Raspberry Pi individualment. S'argumenta, però, que són ideals per aprendre sobre sistemes multinode o per executar aplicacions que requereixin poc càlcul. El preu de cada placa és assequible si es compara amb qualsevol ordinador de sobretaula o sistemes similars, el que permet adquirir-ne una quantitat suficient per a poder realitzar un sistema amb les característiques funcionals d'un sistema distribuït, però amb una potència més reduïda que si es construís amb nodes de gama alta.

Seguint els exemples donats anteriorment, en aquest projecte es planteja usar també Raspberry Pi's. El seu baix cost ha permès adquirir una quantitat de plaques suficientment gran com per muntar un clúster que sigui adient de cara a complir els objectius del projecte. Aconseguir la resta d'elements que composen el sistema no ha suposat tampoc cap gran dificultat doncs la majoria, com el teclat, ratolí o la pantalla HDMI, ja es tenien amb anterioritat. La motivació per a fer el projecte prové d'aquests treballs que, tot i muntar sistemes d'aparença senzilla, requereixen un grapat de coneixements de diversos àmbits per dur a terme les construccions. Coneixements que es volen adquirir a títol personal. Apart, la construcció del sistema suposa un repte emocionant a superar, doncs és un problema complex però a la vegada gratificant i amb resultats visibles.

2.2 Viabilitat del projecte

2.2.1 Viabilitat tècnica

La tasca tècnica principal és la construcció del sistema a base de Raspberry Pi's. També s'ha de considerar la creació d'un joc de programes paral·lels. Per a muntar tot el sistema, són necessaris coneixements de xarxes i arquitectures de computadors que han sigut assolits majorment durant la carrera. També serà important tenir coneixements de l'arquitectura de les plaques per a poder muntar el sistema adequadament sense provocar danys al hardware. Aquest fet ja es considera en el projecte i la seva planificació. Per la part de programació, seran necessaris coneixements de C així com de MPI. Ambdós estudiats durant la carrera, pel que no suposaran cap dificultat més que la de repassar antics apunts si fos necessari.

2.2.2 Viabilitat econòmica

La part econòmica queda determinada pels materials que s'usaran per a la construcció del sistema (hardware) i per les llibreries i programes que es faran servir (software) així com per les hores dedicades al projecte. Precisament s'ha decidit crear un clúster amb Raspberry Pi's per minimitzar el cost final del projecte. Les plaques seran proporcionades per l'alumne i la resta de hardware usat serà proporcionat pel departament d'Arquitectura de Computadors i Sistemes Operatius. El software que es farà servir serà bàsicament els sistemes operatius que implementaran les plaques i les llibreries per a la paral·lelització del programa. Tot s'aconseguirà de programari lliure, pel que no suposarà cap cost econòmic addicional. Finalment les hores queden comptabilitzades en els objectius del projecte i en la planificació.

Es poden veure els costos del projecte en la següent figura:

Hardware	Unitats	Cost unitari	Cost total
Raspberry Pi's Model B	2	32	64
Teclat USB	1	6	6
Ratoli USB	1	5	5
Pantalla HDMI	1	100	100
Cable HDMI	1	3	3
Cable Ethernet	3	1	3
Switch 5 ports	1	10	10
Targeta SD 8GB	2	5	10
Carregador micro USB	2	4	8
PREU TOTAL:			209

Figura 1: Taula costos projecte (euros)

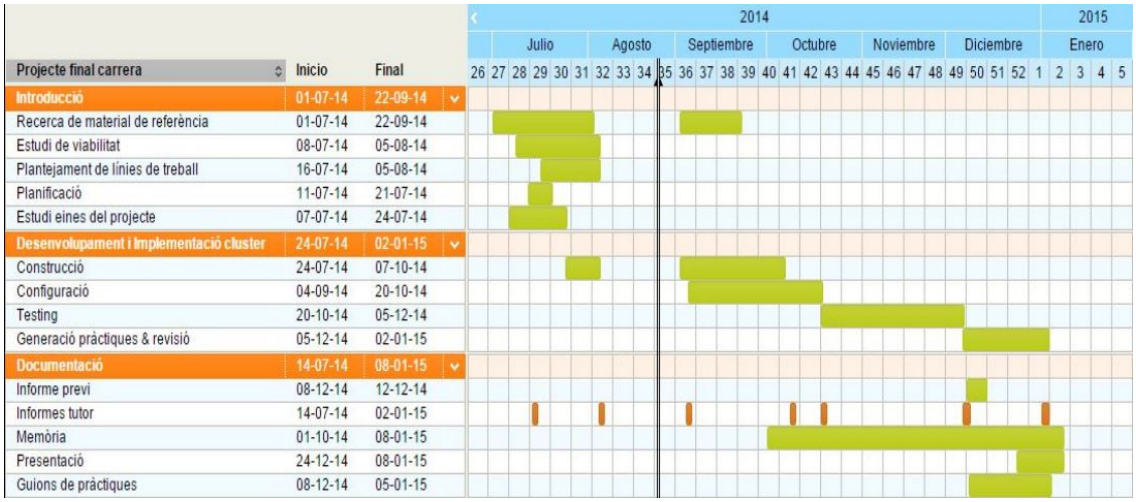
S'ha de comentar que degut a la gran disponibilitat de carregadors micro USB que es té (qualsevol carregador smartphone d'avui en dia amb intensitat 1A ja va bé), no s'ha comprat un hub (15 €). Un hub ens permetria substituir tots els carregadors microUSB ja que podríem connectar totes les Raspberry Pi's al hub, i aquest les alimentaria a totes.

2.2.3 Viabilitat legal

Tot el software que s'usa és programari lliure. És més, les Raspberry Pi's ja varen ser concebudes per a fomentar projectes diversos com el que es planteja. Així doncs no hi ha cap impediment legal.

2.3 Planificació

Aquest projecte s'ha planificat com es mostra en el següent diagrama:



Com es pot apreciar, el projecte es va començar a finals del curs passat, preveient l'entrega al febrer i l'impossibilitat de dedicar moltes hores durant el curs lectiu actual 2014-2015.

El projecte consta d'una primera part d'estudi del problema i els seus components, recerca d'informació, estudi de la viabilitat així com plantejament de possibles solucions i les seves planificacions. A continuació, en una segona part es presenta el treball en sí. Es desenvolupa tot el sistema i es posa a prova. També es fan les pràctiques i es validen. Finalment, el projecte inclou una tercera part de documentació tant del projecte en si com a resultat final com de seguiment del treball realitzat.

Capítol 3

3 Sistema

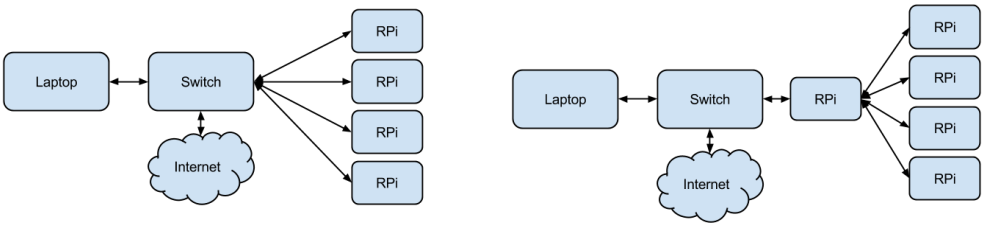
En aquest capítol es tracta la primera part del projecte. S'explica com s'han resolt els objectius que consisteixen en la construcció i configuració del sistema proposat. Després de veure el problema plantejat, l'anàlisi de la solució i la planificació en el capítol anterior, es parla del disseny, la implementació i les proves. Aquest tres temes són els que s'abordaran en aquest capítol. Un per cada subsecció.

3.1 Disseny

En aquest apartat es comentarà quin disseny s'ha plantejat pel problema. Per quins motius i amb quina finalitat s'ha fet.

Donat el pressupost del projecte i les limitacions que aquest comporta de cara al nombre de nodes formats per Raspberry Pi's dels quals es pot disposar, la topologia que s'ha implementat és la de Master-Slave de cara a l'usuari, i de connexió punt a punt via una xarxa LAN entre els diferents nodes. Aquesta distribució del sistema permet executar qualsevol tipus de programa paral·lel, però amb l'avantatge que l'usuari només ha d'interactuar amb un node Master. La distribució esmentada és ideal per a sistemes amb pocs nodes i una xarxa de comunicació ràpida, com és el cas d'aquest projecte. Cal tenir en compte, però, que quan s'escala el nombre de nodes, la xarxa es col·lapsa i surt més a compte crear diversos subsistemes amb el seu propi Master, i que siguin aquests Masters els que es comuniquin entre ells, repartint així la carrega del Master en diversos nodes. Idealment el Master s'hauria de dedicar a gestionar les tasques però, com no es compta amb masses nodes, s'ha implementat una solució en la que el Master esdevé un node de treball més.

En els següents diagrames veiem la vista lògica del sistema per part de l'usuari i la configuració real del sistema.



(a) Diagrama de la vista lògica del sistema (b) Diagrama de la vista física del sistema

Figura 2: Diagrames del sistema

S'ha plantejat el sistema d'aquesta manera per a maximitzar el còmput dels nodes. Implementant la comunicació punt a punt reduïm al mínim el nombre de missatges entre les aplicacions, fent la comunicació directe. Alhora, obligant al Master a calcular augmentem en un el nombre de nodes de càlcul. Es descarta estudiar altres tipus de topologies i configuracions del sistema per falta de nodes. Es considera que no s'obtindria cap millora respecte la solució plantejada que sembla ser l'òptima quan es disposa d'un nombre de nodes tant reduït.

3.2 Construcció i configuració

Aquesta secció entra en el detall de la construcció i configuració del cluster format per plaques Raspberry Pi. Es descriu com s'han realitzat aquestes operacions en el projecte tot donant les instruccions per poder seguir el procediment o adaptar-lo convenientment.

Per a dur a terme el muntatge, s'ha configurat primerament la placa que fa de Master instal·lant-hi un sistema operatiu i les llibreries necessàries per executar les comandes paral·leles, així com creant les claus RSA necessàries per a poder establir connexions SSH sense necessitat de fer inicis de sessió amb usuari i contrasenya (*login*). A continuació s'ha fet el mateix amb la resta de plaques, però partint d'una imatge que estableix la configuració del Master. Finalment, s'ha muntat físicament el cluster, i s'ha configurat per a que la llibreria paral·lela pugui executar programes fent ús de totes les plaques.

3.2.1 Guia ràpida

En aquesta subsecció es comentarà la solució al problema de forma breu, sense entrar en detalls ni explicacions extenses del passos.

Com ja s'ha dit amb anterioritat, primerament s'haurà d'instal·lar un sistema operatiu a la placa que farà de Master. En aquest treball s'ha usat l'eina BerryBoot per a instal·lar el sistema operatiu Raspbian. Un cop instal·lat el sistema operatiu i actualitzat, els passos i les comandes que s'han executat són els que es veuen a continuació:

```
1  ##  Generar claus RSA
2
3      $  cd ~
4      $  ssh-keygen -t rsa -C "pi@raspberry"
5      ## L'opció de passphrase la deixarem en blanc.
6      $  cat /home/pi/.shh/id_rsa.pub >> /home/pi/-shh/authorized_keys
```

Figura 3: Claus RSA

```
1  ##  Instal·lació MPICH
2
3      $  cd ~
4      $  sudo apt-get install gfortran
5      $  mkdir mpich3
6      $  cd /mpich3
7      $  mkdir install build
8      $  wget http://www.mpich.org/static/downloads/3.1.2/mpich-3.1.2.tar.gz
9      ##  Cal assegurar-se que és la última versió disponible
10     ##  a la web http://www.mpich.org/downloads/
11     $  tar xvfz mpich-3.1.2.tar.gz
12     $  home/pi/mpich3/mpich-3.1.2/configure -prefix=/home/pi/mpich3/install
13     $  cd /build
14     $  make
15     $  make install
16     $  export PATH=$PATH:/home/pi/mpich3/install/bin
17     ##  O si es vol fer de forma permanent, afegir al final
18     ##  del document /home/pi/.profile les següents dues línies:
19             #MPI
20             export PATH="$PATH:/home/pi/mpich3/install/bin"
```

Figura 4: Instal·lació MPICH

El que s'ha fet tot seguit és crear una imatge de la targeta SD del Master. En aquest projecte s'ha fet ús de l'eina *Win32 Disk Imager*. Un cop obtinguda la imatge, s'ha copiat a totes les altres targes. L'últim pas ha consistit en configurar els Slaves. Com que des del Master s'ha de fer una connexió SSH al Slave, ha calgut esbrinar la direcció IP del Slave amb anterioritat.

```
##  Configuració Slave

  $  shh -A pi@192.168.Y.X
  ## On X i Y corresponen a la IP del Slave
  $  cd /home/pi/.shh
  $  rm id_rsa id_rsa.pub
  $  cat ~/.ssh/id_rsa.pub | ssh pi@192.168.Y.X "cat >> .ssh/authorized_keys"
  ## És opcional canviar el hostname del Slave:
  $  ssh pi@192.168.Y.X 'sudo echo "Slave1" | sudo tee /etc/hostname; sudo shutdown -r now'
```

Figura 5: Configuració slave

Com a resultat s'ha obtingut físicament el sistema descrit amb anterioritat i amb les llibreries necessàries per executar programes paral·lels usant directives MPI.

3.2.2 Pas a pas

En aquesta subsecció s'aprofundirà una mica més en el procés d'instal·lació i configuració del sistema, donant més detalls i valoracions.

Es comença per llistar els elements necessaris per portar a terme la construcció del sistema:

```
Raspberry Pi + targeta SD
Font d'energia micro-USB d' 1A a 5V
Teclat i ratolí USB
Monitor HDMI i cable HDMI-HDMI
Cable Ethernet
Switch o router
Connexió a internet
```

Figura 6: Llista de hardware necessari

El primer pas ha consistit en instal·lar un sistema operatiu a una targeta SD per a poder treballar amb la Raspberry Pi que esdevindrà el Master. Aquesta tasca s'ha dut a terme descarregant l'eina *BerryBoot* [4] que es pot trobar per Internet de forma gratuïta.

Amb l'ajut d'un ordinador capaç de llegir i escriure a la targeta, s'ha formatat amb el format del sistema d'arxius FAT32. A continuació s'ha extret el contingut del .zip que conté l'eina BerryBoot a la mateixa targeta i s'ha muntat el sistema com es mostra en el diagrama. Tot seguit s'ha donat corrent per arrancar BerryBoot.

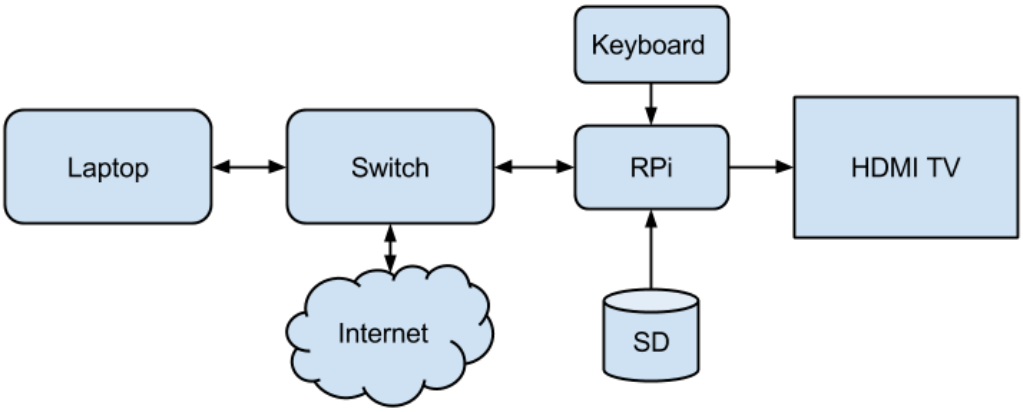


Figura 7: Diagrama del sistema al instal·lar l'eina BerryBoot

BerryBoot és una eina que ens permet instal·lar diversos sistemes operatius. Entre les opcions que tenim està la distribució Raspbian, sistema operatiu que es farà servir en aquest treball.

El funcionament de l'eina d'instal·lació de SO és a base de menús on es presenten les diferents opcions. Primerament, l'eina presenta un menú de benvinguda que en el projecte s'ha omplert com es mostra a la següent figura.

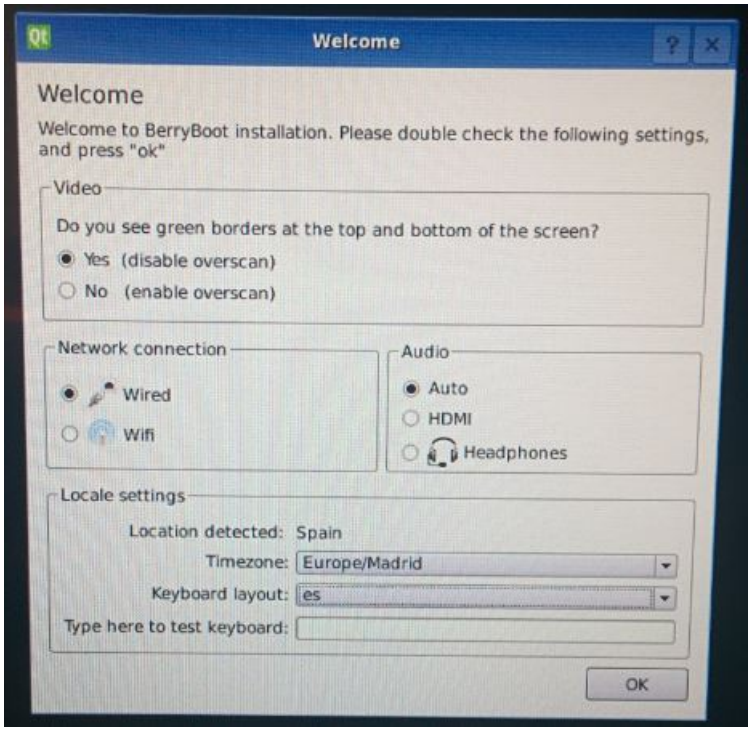


Figura 8: Menú de benvinguda

El menú que ve a continuació demana la unitat en la qual es vol instal·lar el SO ja que l'eina ha

de formatar la unitat i la deixar-la apte per a instal·lar el sistema operatiu. Existeix la possibilitat de guardar els arxius del sistema operatiu a una unitat de memòria diferent a la targeta SD, com podria ser, per exemple, un pendrive, o un disc dur extern. En tot cas, s'ha escollit la unitat que correspon a la targeta, doncs independentment d'on guardem els arxius, es necessita la SD per arrancar el sistema operatiu. Amb aquesta elecció el que busquem és deixar lliures els dos ports USB.

El següent menú permet seleccionar un sistema operatiu a instal·lar.

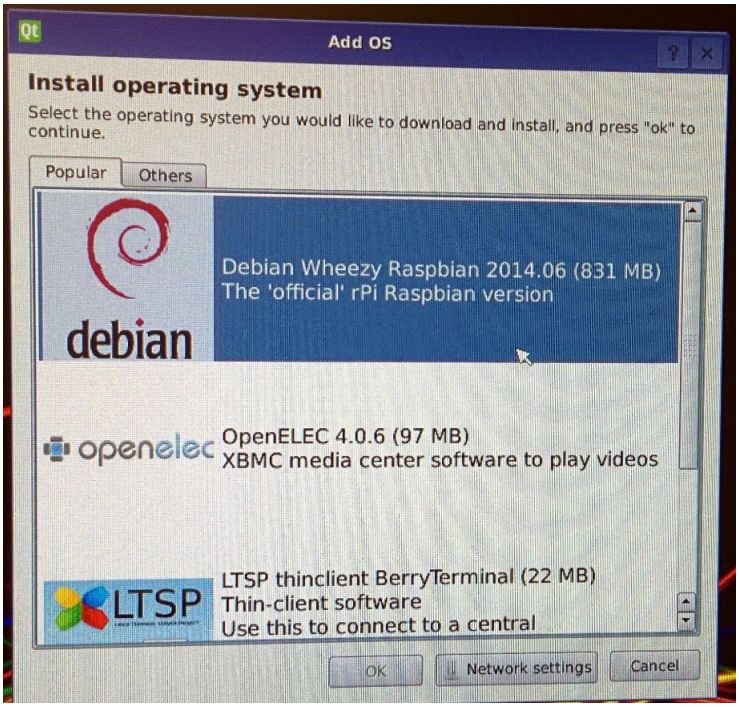


Figura 9: Selecció del SO

Com es veu en la figura anterior, s'ha escollit Debian Wheezy Raspbian 2014.06, sistema que ocupa 831MB. Conseqüentment l'ample de banda de la connexió a Internet és un fet a tenir en compte quan es descarrega. Un cop instal·lat el sistema operatiu, i com en el projecte no es requerirà cap altre, s'ha configurat la SD per a que arranqui sempre Raspbian per defecte.
Ja l'últim que ens demana BerryBoot abans de que Raspbian s'executi és una configuració de la Raspberry Pi.

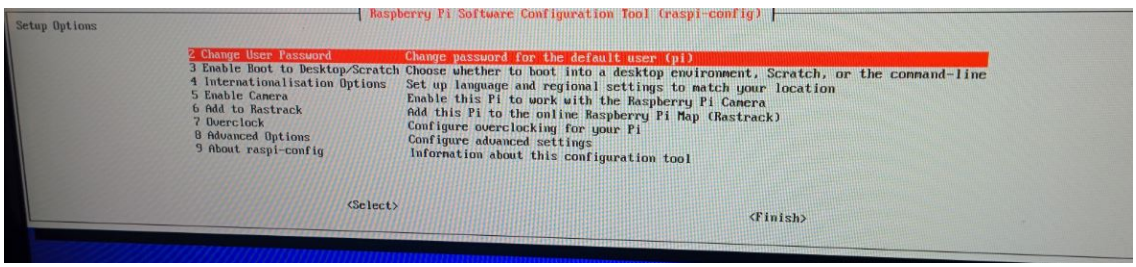


Figura 10: Configuració de la RPi

El menú és bastant explícit i ens permet fer el que cada una de les entrades diu. Pel projecte interessa habilitar les connexions SSH per a que les diferents plaques es puguin comunicar entre

elles. També pot ser còmode habilitar que al donar potència a la placa, aquesta carregui directament el escriptori de Raspbian. Un cop finalitzada la configuració, la placa es reinicia i podem fer un login al sistema operatiu[2].

El que s'ha fet servir en el projecte és la terminal però, com es pot apreciar, hi han diversos programes preinstal·lats.

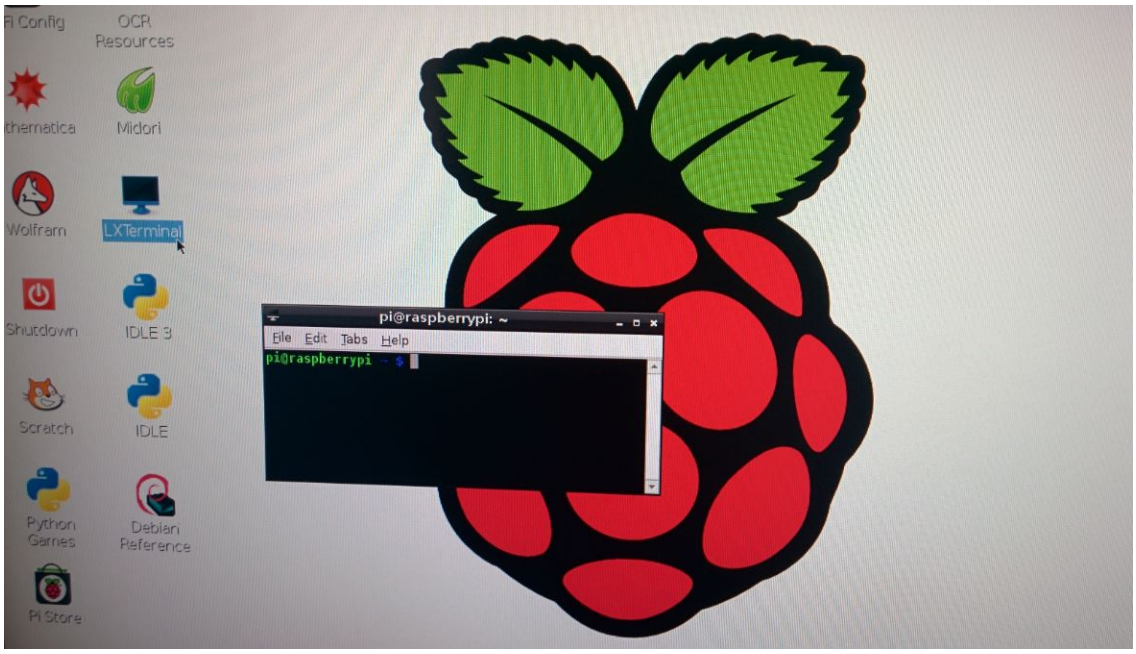


Figura 11: Escriptori Raspbian

Si la versió descarregada de BerryBoot és recent, es tindrà el sistema operatiu al dia. En tot cas, no és mala idea actualitzar-lo.

```
$ sudo apt-get update
```

Altre cop, el temps que s'haurà d'esperar dependrà de la velocitat del Internet connectat a la placa.

Per comoditat, en aquest projecte, a l'hora d'interactuar amb les diferents plaques, s'ha usat l'eina de Windows gratuïta PuTTY, que permet fer connexions SSH a cada placa i, per tant, treballar en un ordinador extern al sistema, ja que cada connexió permet executar comandes a la terminal com si ho féssim des de la mateixa Raspberry Pi. Per tant, el diagrama és el mateix que a la figura 7.

L'únic que es necessita per a la connexió, apart de que físicament estiguin connectades a la mateixa xarxa LAN, és la adreça IP de la placa, que fàcilment s'obté com es veu a la següent figura. En aquest cas 192.168.1.101

```
pi@raspberrypi ~ $ ip addr show eth0
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether b8:27:eb:cf:02:b0 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.101/24 brd 192.168.1.255 scope global eth0
       valid_lft forever preferred_lft forever
```

Figura 12: Direcció IP

[2]Per defecte el nom d'usuari és pi i la contrasenya raspberry, però es pot canviar en el menú de configuració o, més tard, per terminal.

I amb això es pot omplir el menú que demana PuTTY i fer un login a la placa.

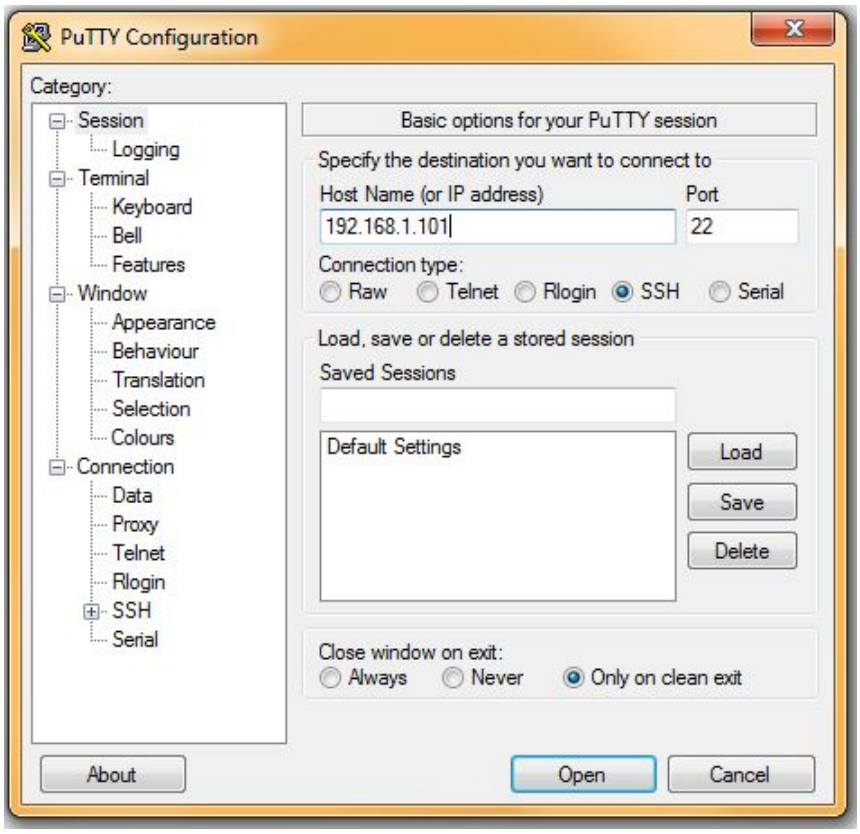


Figura 13: Menú PuTTY

El següent pas ha consistit en la creació de les claus RSA. Aquest pas serveix per a que les plaques que formin el sistema es puguin comunicar entre elles sense haver de fer cap login, fet que és vital per a la rapidesa en les comunicacions de programes paral·lels.

Com es pot apreciar en les dues primeres comandes del bloc de codi *Claus RSA* (figura 3), el que s'ha fet és generar tant una clau pública com una de privada. Pel que fa a l'opció del passphrase (*contrassenya*), s'ha decidit deixar-la en blanc. Donaria més seguretat a la connexió però, donat que es treballarà en una xarxa LAN local, no s'ha considerat necessari. Amb la tercera comanda s'aconsegueix copiar la clau pública que s'ha generat a la llista de claus autoritzades. Això permet fer connexions localhost (*aquest ordinador*), és a dir, permet que des d'una màquina ens puguem connectar a la mateixa màquina fent servir, per exemple, una connexió SSH. El que interessa serà tenir aquesta clau pública a les claus autoritzades de la resta de plaques.

En el segon bloc de codi, *Instal·lació MPICH* (figura 4), les dues primeres comandes instal·len Fortran, que a la seva vegada, ajuda en la instal·lació de MPICH. Les comandes que venen a continuació creen diferents carpetes, que han permès mantenir organitzat el projecte.

La comanda de la línia 8 s'usa per descarregar MPICH. En el moment d'aquest projecte, la versió que s'oferia era la 3.1.2. (es pot comprovar la versió actual des de la web de MPICH). La comanda que ve just a continuació permet descomprimir l'arxiu que s'ha descarregat amb la comanda anterior. Abans de poder finalitzar la instal·lació, s'han de configurar els paràmetres per a que es dugui a terme com volem. La comanda de la línia 12 modifica aquesta configuració per tal de que el resultat del procés d'instal·lació es guardi al directori que s'ha anomenat *Install*. Les comandes de la línia 13 a la 15 permeten compilar i instal·lar MPICH. Cal considerar que el make i el make install són ambdues comandes que triguen un temps considerable en executar-se.

La primera de l'ordre d'una hora i la segona aproximadament la meitat. Una altre consideració és la variable PATH. Per a que, quan executem un programa, la llibreria on estan les directives MPI sigui accessible, cal que s'executi la comanda de la línia 16. Si no es vol executar la mateixa comanda sempre que s'utilitzin directives de la llibreria de directives paral·leles, es pot optar per editar el document /home/pi/.profile i afegir-hi al final les dues línies 19 i 20. D'aquesta manera no caldria executar la comanda de la línia 16 doncs ja es farà automàticament.

L'últim pas de la construcció i configuració del sistema ha consistit en posar a punt la resta de plaques apart de la que correspon al Master. Per dur-ho a terme, s'ha tingut que crear una imatge de la targeta SD del Master amb l'ajut d'una eina gratuïta com és el programa Win32 Disk Imager. La mateixa eina s'ha emprat també per a copiar la imatge a la resta de targes, obtenint un sistema com el que es mostra al diagrama de la figura 2b. Tot i ser possible treballar directament amb cada Raspberry Pi, en aquest projecte s'ha optat per treballar seguint el diagrama de la figura 2a. Mitjançant una connexió SSH al Master i, a partir d'aquesta placa, llançar les comandes i que s'executin en les diferents Raspberry Pi Slaves fent servir també una connexió SSH per a la comunicació entre plaques. Amb aquest objectiu en ment, ha sigut necessari editar alguns dels fitxers que resideixen en les imatges dels Slaves i que s'han passat del Master al fer la copia de la imatge. Les diferents comandes que s'han dut a terme per a l'edició es poden veure en el bloc de codi *Configuració Slave* (figura 5). Per a executar les esmentades comandes és necessari conèixer la direcció IP de cada Slave ja que es necessita establir una connexió SSH entre el Master i cada altra placa. Un cop s'ha fet la connexió i s'ha accedit al directori on resideixen les claus RSA, s'han eliminat. Aquest pas ha estat necessari ja que s'han traspassat, com es comentava abans, al copiar la imatge feta de la SD corresponent a la placa Master. A continuació s'ha copiat la clau pública del Master a les claus autoritzades de cada Slave fent us de la comanda de la línia 7. De manera totalment opcional, però per comoditat, s'ha executat la comanda de la línia 9, que permet canviar el hostname i diferenciar fàcilment totes les plaques així com reconèixer-les ràpidament.

Aquí conclou la secció corresponent a la construcció i configuració del sistema on s'ha vist quin procediment s'ha seguit en aquest projecte per posar a punt els diferents components del sistema.

3.3 Test

L'última secció del capítol consistirà en el joc de proves implementat per veure com es comporta el sistema davant de diferents programes variats.

3.3.1 Com fer

Aquest incís en el capítol descriu quins són els passos per a escriure, compilar i executar un programa adequadament pel sistema proposat.

La redacció d'un programa es pot fer en el format més còmode per l'usuari. Una opció seria crear un nou document amb l'editor nano (o vi si es prefereix) des de la terminal. També es pot connectar els perifèrics adients a una RaspberryPi i obrir un editor de text des de l'escriptori, o fins i tot escriure el programa externament al sistema, i introduir-lo a aquest fent una copia per ssh. Només és necessita compilar el codi en un node, però per a poder executar-lo paral·lelament cal que tots els nodes continguin una copia del codi en el mateix PATH.

Suposem que tenim el codi prova.c en un directori del Master tal com /home/pi/code i que hi estem a dintre. Per compilar el codi ens farà falta la comanda:

```
$ mpicc -g -o prova prova.c
```

el resultat serà un executable anomenat prova i és el que haurem de copiar a tots els nodes. Per fer-ho ràpidament es pot fer servir la comanda:

```
$ scp /home/pi/code/prova pi@192.168.1.X:/home/pi/code
```

que copiara el executable al mateix directori del node amb IP 192.168.1.X i usuari pi. És important copiar en el mateix PATH tots els compilats ja que la comanda per executar el algorisme és la mateixa per a tots els nodes. L'execució es dur a terme amb:

```
$ mpiexec -f pifile -n 2  prova
```

En els paràmetres d'aquesta comanda veiem el nom de l'executable, però també un fitxer (-f) anomenat pifile i el nombre de nodes (-n) que en aquest cas és dos, indicant que el programa s'executarà en paral·lel en dos nodes, la direcció IP dels quals està en el fitxer pifile i que és necessària per a poder fer servir les comandes mpi.

```
1 192.168.1.X
2 192.168.1.Y
3
```

Figura 14: contingut fitxer pifile

Vist com llançar un programa, en la següent secció s'explica quins programes s'han creat com a joc de proves del sistema tractat en el projecte.

3.3.2 Joc de proves

El joc de proves proposat per aquest projecte consta de dos programes que augmenten progressivament de complexitat. Tot i ser un joc a priori no massa ampli, tracta els punts crítics d'un sistema distribuït com són la comunicació entre nodes i el còmput en cada un d'aquests nodes.

Es pot veure el codi de cada un dels programes a l'annex. Tot i així en la subsecció es farà un petit comentari del que fa cadascun i es valorarà la seva utilitat i rendiment.

MonteCarlo

Aquest programa calcula una aproximació del nombre 'pi' fent ús del mètode de MonteCarlo [5]. A grans trets, el programa genera unes coordenades aleatòries a l'espai de coordenades [-1,1]x[-1,1], i va acumulant totes les que, partint del punt (0,0), tinguin mòdul menor que 1 o, altrament dit, les que caiguin al cercle de radi 1 inscrit al quadrat [-1,1]x[-1,1]. Un cop generat un nombre determinat de punts N, el programa estima el valor de 'pi' fent ús de la fórmula de l'àrea del cercle que ens permet obtenir el resultat al dividir 4 cops (àrea del quadrat) el nombre total de punts que han caigut dintre el cercle (aproximació de pi) entre el nombre total de punts generats (aproximació àrea del quadrat).

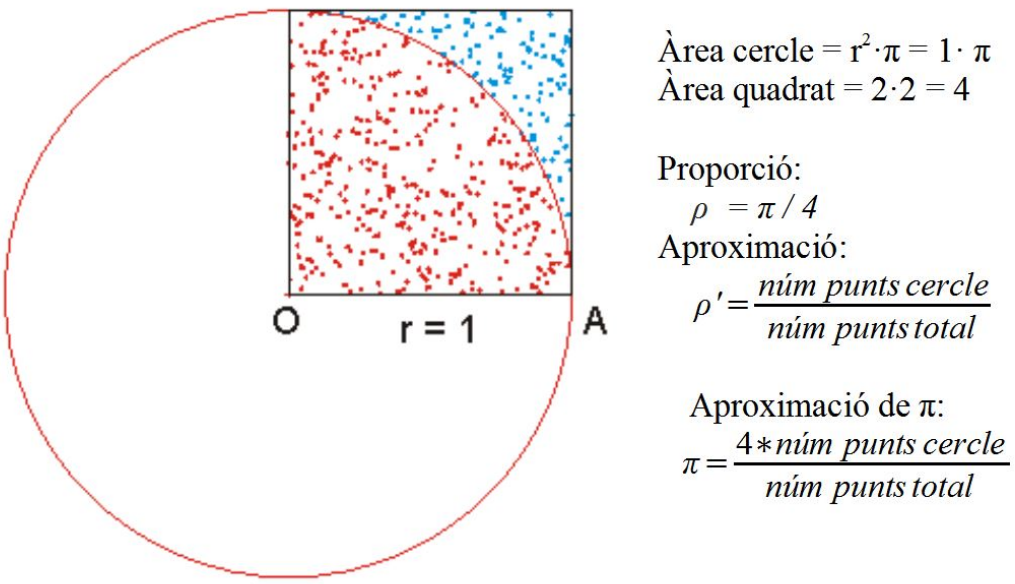


Figura 15: Esquema del mètode de MonteCarlo per aproximar el nombre π

El programa és escandalosament paral·lel. Si s'usen N nodes per realitzar el càlcul, el temps de execució es reduirà en N. Això vol dir que, si tenim R plaques Raspberry Pi, podríem fer una aproximació de pi usant N punts en una R-enèsima part del temps que ens portaria el càlcul amb una sola placa. Tot i així, donat aquest tipus de programa en el que es busca una aproximació com a resultat, es pot plantejar el paral·lelisme de manera diferent a l'enfoc més habitual. En canvi de reduir el temps d'execució R vegades, es pot fer servir el mateix temps per fer el càlcul de RxN punts i augmentar la precisió del resultat. O també es pot implementar una combinació de les dues opcions.

L'algorisme fa servir les comandes MPI per a que cadascú dels diferents nodes calculi N punts i, al final de tot, es fa servir una comanda del tipus reduce que ens permet sumar tots els resultats dels diferents nodes quan retornen el valor parcial al Master. Per tant el programa ens permet repartir el càlcul de l'aplicació totalment, obtenint així un programa distribuït.

Simulació de la difusió de calor en una dimensió (Stencil 1D)

El segon dels programes del joc de proves és un model discret simplificat de la difusió de calor en una sola dimensió mitjançant la fórmula [6]:

$$u_k^{j+1} = u_k^j + (u_{k+1}^j + u_{k-1}^j - 2 \cdot u_k^j)$$

On cada posició de l'array u ve determinada pel subíndex i la iteració del model ve donada pel superíndex. Cada nova iteració del mètode consisteix en aplicar l'equació a totes les posicions del vector fent servir els valors de l'estat anterior.

Per paral·lelitzar aquest mètode iteratiu, es fan tantes particions del vector de dades on es vol aplicar com Raspberry Pi's estiguin involucrades en la execució del programa que implementa el mètode, i es reparteixen. A continuació, cada Raspberry Pi aplica una iteració a la partició que li ha sigut assignada per cada unitat de temps que es vulgui avançar. La dificultat de la

paral·lelització resideix en els extrems de cada partició o subvector. Com es veu de la fórmula són necessàries les dues posicions adjacents a cada element del vector per calcular el nou estat. Es dona el fet que els valors adjacents dels extrems del subvector resideixen en un altre node, pel que aquest algorisme requereix d'una comunicació entre nodes per a cada càlcul del seu subvector. Tot i donar-se aquest fet, la comunicació és només d'una posició del vector, és a dir, d'una unitat del tipus de dada amb el que treballem. Conseqüentment el missatge és lleuger, i no es produeix una gran limitació en el rendiment del per comunicació al executar el programa en el nostre sistema.

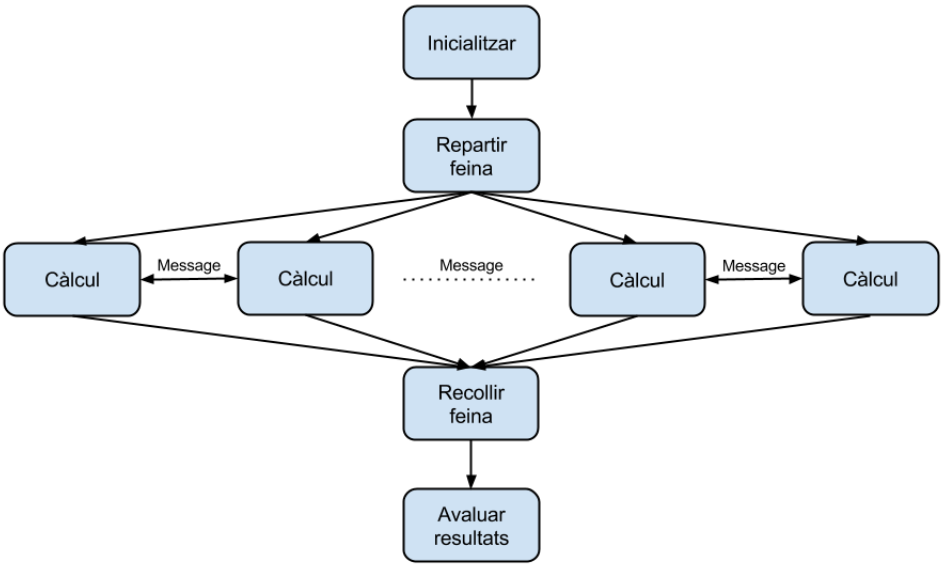


Figura 16: Diagrama de flux del programa

Existeix també una millora fent ús de les comandes que ens proporciona MPI. Aquesta millora consisteix en calcular primerament els nous valors dels extrems i enviar-los immediatament fet el càlcul de forma no blocant. Això ens permet calcular la resta dels valors alhora que s'envia el missatge aprofitant així l'espera. Finalment, es reben els nous valors per poder començar amb la següent iteració.

Estudi dels resultats

En aquesta subsecció es valoren els resultats obtinguts de l'execució dels programes del joc de proves. Fent us del software estadístic R s'han avaluat els resultats d'ambdós programes per generar models que ajustin les dades, amb l'objectiu de fer prediccions sobre el nombre de punts necessàris per obtenir una aproximació amb un error menor a una certa tolerància fixada.

Pel que fa al mètode de MonteCarlo tenim dos paràmetres d'entrada. Un corresponent al nombre de punts generat en cada node i tenim també el nombre de nodes usats. Com a resultats obtenim el error de l'aproximació del mètode i el temps d'execució del programa (en ms). Per avaluar els resultats s'ha decidit utilitzar potències de deu com a paràmetre d'entrada de la variable corresponent als punts a generar. Pel que fa als nodes s'han avaluat els resultats del programa

sèrie (un sol node) i del programa paral·lel amb dos i quatre nodes. Veiem una gràfica (figura 17) dels valors absoluts dels errors respecte els punts usats.

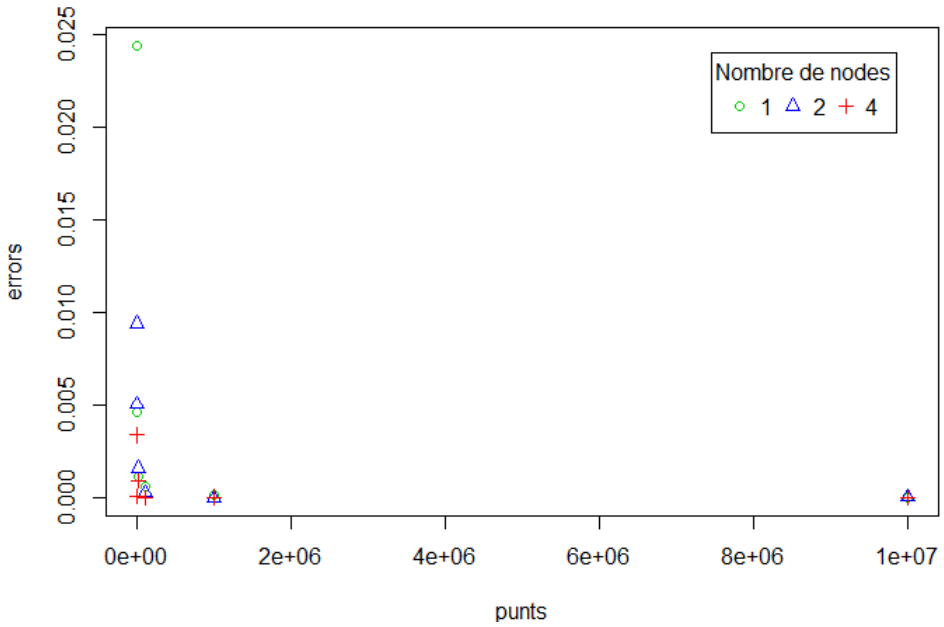


Figura 17: Plot dels errors del mètode de MonteCarlo

L'únic que es pot apreciar en aquest plot és que aquesta escala no serveix per visualitzar els resultats.

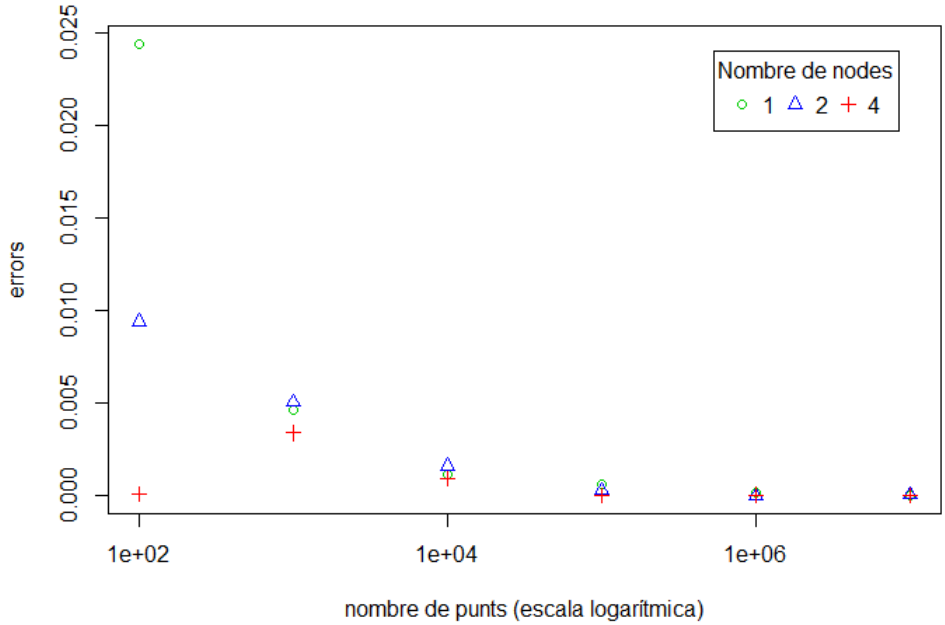


Figura 18: Plot dels errors del mètode de MonteCarlo respecte el log10 dels punts

Donat que els punts generats són potències de deu, s'ha canviat l'escala fent un logaritme. S'aprecia que els errors també tenen un comportament exponencial pel que sembla bona idea

aplicar també un logaritme als errors.

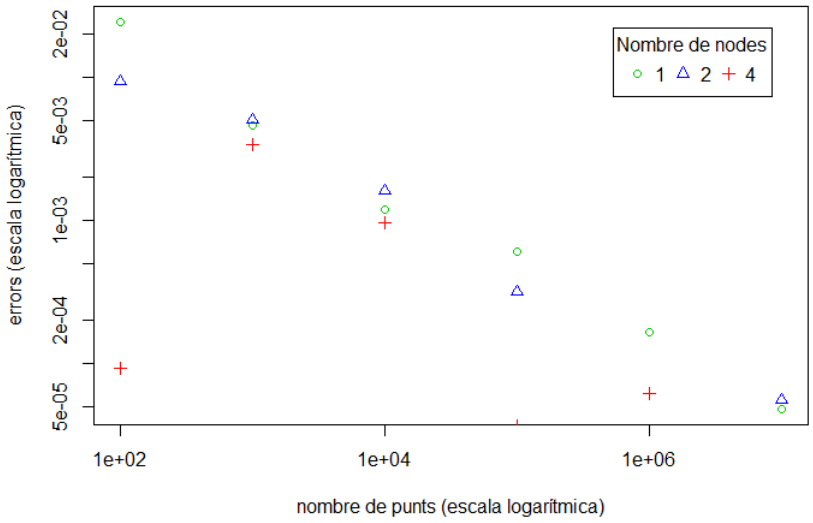


Figura 19: Plot de log10 errors del mètode de MonteCarlo respecte el log10 dels punts

En aquesta nova gràfica es pot veure un comportament clarament lineal entre els errors i els punts generats. El software estadístic R ens permet ajustar les dades dels errors per un model lineal en funció del punts i una variable aleatòria discreta que representi els nodes (figura 20).

```
> fit = lm( lerror ~ lpunts + nodes-1)
> summary(fit)

Call:
lm(formula = lerror ~ lpunts + nodes - 1)

Residuals:
     Min       1Q   Median       3Q      Max
-0.65265 -0.28449 -0.08907  0.10750  1.36372

Coefficients:
       Estimate Std. Error t value Pr(>|t|)
lpunts  0.45209    0.07498   6.029  3.1e-05 ***
nodes1  0.99564    0.40380   2.466 0.027209 *
nodes2  1.25975    0.40380   3.120 0.007531 **
nodes4  1.76524    0.40380   4.372 0.000639 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.5433 on 14 degrees of freedom
Multiple R-squared:  0.9814, Adjusted R-squared:  0.976
F-statistic: 184.3 on 4 and 14 DF,  p-value: 6.156e-12
```

Figura 20: Model lineal dels errors del mètode de MonteCarlo

La comanda *lm* (*linear model*) de *R* ens indica com afecta cada variable aleatòria a l'ajust de les dades per una recta. En aquest cas les dades s'ajusten per la recta:

$$error_k = 0.45209 \cdot punts_k + 0.99564 \cdot nodes1_k + 1.25975 \cdot nodes2_k + 1.76524 \cdot nodes4_k$$

On la variable *lpunts* indica el pendent de la recta i la variable discreta *nodes* indica el desplaçament. Es pot veure de la figura 20 que totes les variables considerades són significatives en

el model. La comanda *lm* realitza automàticament un test *t-student* per a contrastar la hipòtesis que una variable aleatòria sigui zero. En l'última columna veiem que es rebutja la hipòtesis nul·la del test (la variable aleatòria és zero) amb una significació indicada pel nombre d'asterisc ja que el p-valor resultant d'aplicar el test és menor als nivells de significació indicats (0.01 com a mínim en aquest model).
A més a més, si ens fixem en el coeficient de correlació (*Adjusted R-squared*) veiem que és molt pròxim a u, fet que indica que el model ajusta molt bé les dades i que el residu dels valors ajustats (valor mesurat menys valor predit) és pròxim a zero.

Cal notar un parell de fets. Primerament en aquest model s'ha decidit treure l'*intercept* del model (desplaçament de la recta respecte l'origen en l'eix d'ordenades) ja que aquest va inclòs en els coeficients de les variables nodes això s'ha fet amb el -1 de la comanda. Altrament, cal comentar que s'ha treballat amb el valor absolut del logaritme dels errors per obtenir el model. Això comporta una simetria respecte l'eix d'abscisses però, per comoditat, s'ha escollit el model positiu.

S'ha fet el mateix estudi relacionant el temps i els punts generats, tenint en compte els nodes usats. Aplicats els logaritmes, s'ha obtingut la següent gràfica:

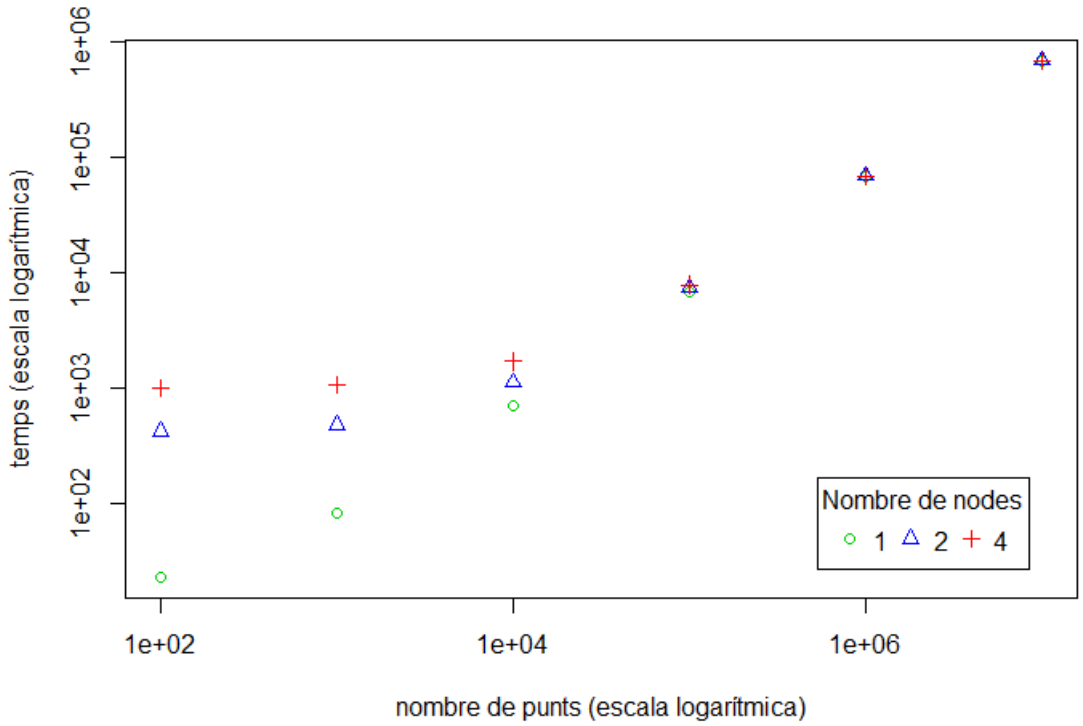


Figura 21: Plot de log10 del temps del mètode de MonteCarlo respecte el log10 dels punts

En verd (punts) s'aprecien els temps del programa sèrie, en blau (triangles) usant dos nodes i en vermell (creus) quatre. La diferència de temps per a pocs punts s'explica per l'*overhead* (temps d'execució addicional) produït de la paral·lelització. Tot i així, es pot veure que per a un nombre de punts prou gran els temps són iguals, fet que indica com ja se sabia que el programa es totalment distribuït.

Si es genera amb R un model lineal els resultats s'ajusten també molt bé.

```
> fit = lm( ltemps ~ lpunts + nodes-1)
> summary(fit)

Call:
lm(formula = ltemps ~ lpunts + nodes - 1)

Residuals:
    Min      1Q  Median      3Q     Max
-0.4370 -0.3009 -0.0545  0.2010  0.8306

Coefficients:
       Estimate Std. Error t value Pr(>|t|)
lpunts  0.72027    0.05898  12.213 7.46e-09 ***
nodes1  0.19201    0.31759   0.605   0.5551
nodes2  0.57241    0.31759   1.802   0.0931 .
nodes4  0.72578    0.31759   2.285   0.0384 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.4273 on 14 degrees of freedom
Multiple R-squared:  0.9909, Adjusted R-squared:  0.9884
F-statistic: 382.9 on 4 and 14 DF,  p-value: 3.973e-14
```

Figura 22: Model lineal del temps del mètode de MonteCarlo

Amb l'ajust del model es pot veure com el *overhead* fa que els nodes siguin un paràmetre significatiu però no del tot. A diferència del que veiem en el model anterior dels errors figura 20 en aquest model la variable nodes té un nivell de significació de 0.1 o, fins i tot, quan nodes val 1, la hipòtesis de que la variable aleatòria nodes1 sigui zero no es rebutja. De fet, si no consideréssim els primers valors on el nombre de punts és petit, el model s'ajustaria encara millor i aleshores el nombre de nodes no seria significatiu.

A continuació i, gràcies a ambdós models, es pot fer una predicció de l'error i del temps estimat d'execució del programa donat un nombre de punts i nodes o, el que és millor, es pot fer una predicció inversa i estimar quants punts necessitarem passar com a entrada per generar un error més petit que una certa tolerància, o trigar un cert temps. Suposem que volem una estimació de 3 xifres decimals significatives (volem 3,141...). Això implica un error d'$1 \cdot e^{-4}$ o menys. Calculant amb els paràmetres del model obtenim que hem de generar el còmput amb uns 200k punts. Aplicant el càlcul, ens surt un error de $9 \cdot e^{-5}$ que està lleugerament per sota de l'error que buscàvem. El temps estimat era de 13 minuts i el programa n'ha trigat dos i escaig. Ara bé, si descartem del model la variable dels nodes, ens surten 3 minuts, que és una aproximació molt més acurada. Es veu doncs empíricament el fet que s'argumentava de forma teòrica amb anterioritat. La diferència de temps en la predicció s'explica pel fet de calcular una exponencial fer recuperar el temps original expressat en logaritme al model. Per tant, s'ha vist que usar més nodes permet generar més punts en el mateix temps i reduir l'error del mètode.

Només s'ha fet un estudi del temps (ms) d'execució del programa que implementa el model discret simplificat de la difusió de calor ja que fer una aproximació de l'error respecte a la solució real és un problema més complex. En aquest programa tenim un paràmetre d'entrada que equival a la mida del vector i un altre pel número de nodes. Com a sortida tenim el temps de còmput. S'ha decidit augmentar la mida del vector multiplicant per dos cada cop, és a dir, les mides del vector considerades són potències de dos. Conseqüentment es pot observar un comportament exponencial com al programa anterior.

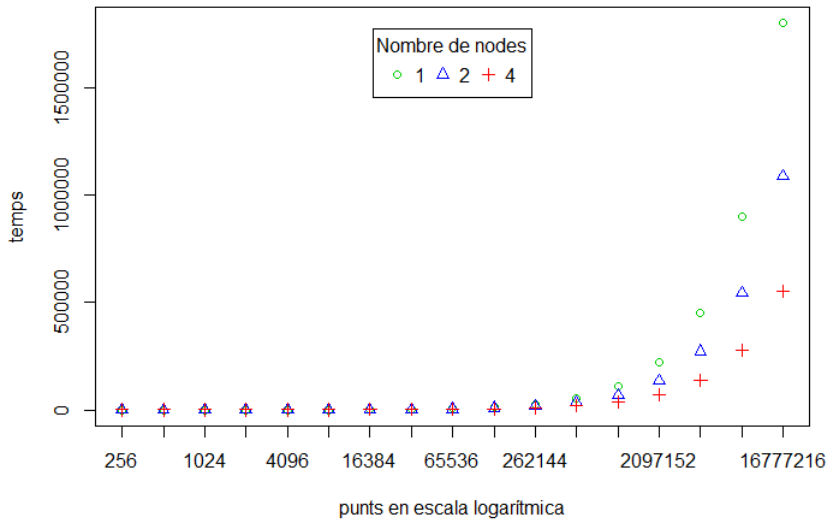


Figura 23: Plot del temps del mètode Stencil en ms respecte el log dels punts

I si s'aplica un logaritme al temps

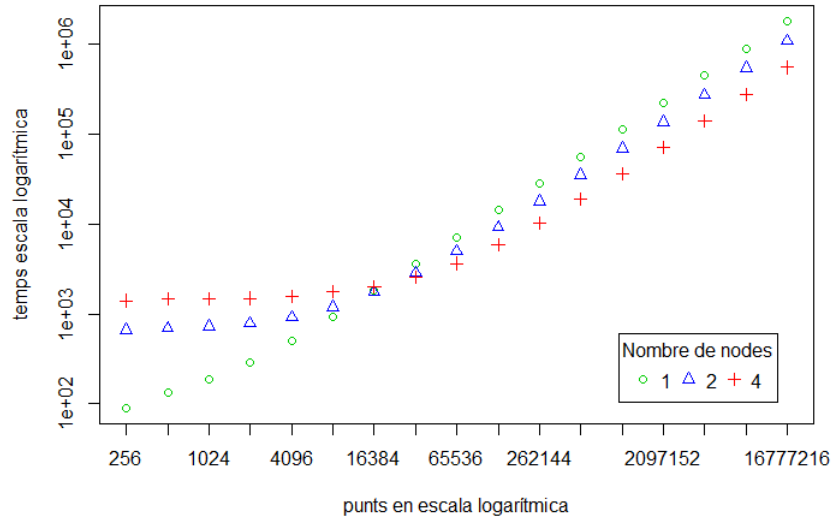


Figura 24: Plot del log del temps del mètode Stencil respecte el log dels punts

Ara sí s'observa una disminució del temps d'execució en funció del nombre de nodes usats per a valors del vector grans. Es pot apreciar gràficament el *overhead* per a mides de vector petites. Fins un valor de $SZ = 16384$ el algorisme sèrie encara és més ràpid que el paral·lel. També es pot veure com s'ajusten les dades. Pel programa sèrie s'aprecia que el comportament és lineal, però per més nodes el *overhead* trenca la linealitat per a valors petits. Usant R s'ha generat el següent model:

```
> fit = lm( ltemps ~ lSZ + nodes-1)
> summary(fit)

Call:
lm(formula = ltemps ~ lSZ + nodes - 1)

Residuals:
    Min      1Q  Median      3Q     Max
-1.3443 -0.8883 -0.2354  0.6711  3.1551

Coefficients:
       Estimate Std. Error t value Pr(>|t|)
lSZ     0.72763    0.03247  22.409  < 2e-16 ***
nodes1  1.40090    0.58806   2.382  0.02131 *
nodes2  1.60559    0.58806   2.730  0.00888 **
nodes4  1.47818    0.58806   2.514  0.01543 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.136 on 47 degrees of freedom
Multiple R-squared:  0.9936, Adjusted R-squared:  0.9931
F-statistic:  1831 on 4 and 47 DF,  p-value: < 2.2e-16
```

Figura 25: Model lineal del temps del mètode Stencil

L'ajust és excel·lent ja que el valor del *Adjuster R-squared* és molt pròxim a u. Com es veia gràficament abans en la figura 24, es veu numèricament que en aquest segon programa el factor nodes si que és rellevant pel temps d'execució (totes les variables 'passen' el test t-student amb una tolerància menor al 0.05).

Per estudiar com afecta el nombre de nodes s'ha decidit calcular el *speed-up* dels programes.

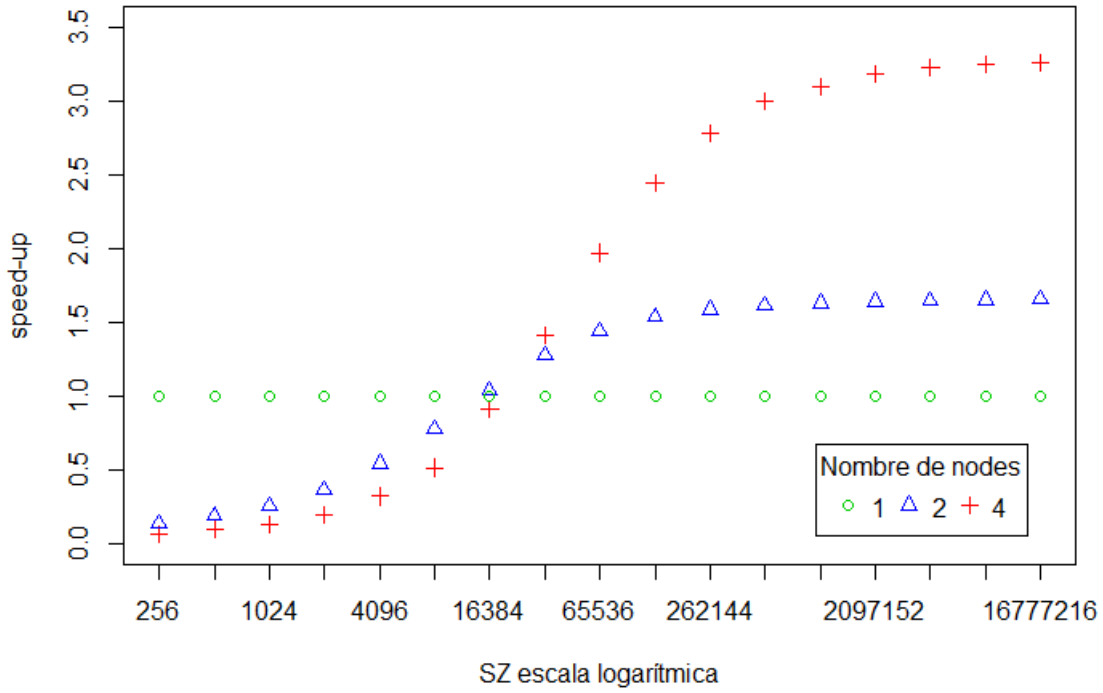


Figura 26: Speed-up del mètode Stencil executant el programa paral·lel amb 2 i 4 nodes, respecte el valor de l'execució en sèrie.

En verd es representa un *speed-up* nul. En blau es veu la comparativa entre el programa sèrie i

el programa paral·lel amb dos nodes i en vermell del programa sèrie amb el paral·lel usant quatre nodes. Podem veure que usant dos nodes s'ha aconseguit un *speed-up* d'una mica més de 1.5 i usant quatre nodes de 3.3 aproximadament. Si es comparen els dos programes paral·lels s'obté un *speed-up* de dos. Que els *speed-up*s no siguin més elevats s'explica principalment per dos motius. El primer és el muntatge del sistema amb components de baixa qualitat però de baix preu. El segon és el programa en sí. Les constants comunicacions entre nodes generen petites esperes ja que les plaques no son síncrones i no van igual de ràpid en fer el càlcul de cada iteració.

Per fer una comparativa similar s'ha decidit calcular el paràmetre de rendiment anomenat *ppt* (punts partit temps) resultant de la divisió dels punts usats en una execució entre el temps en ms d'execució del programa.

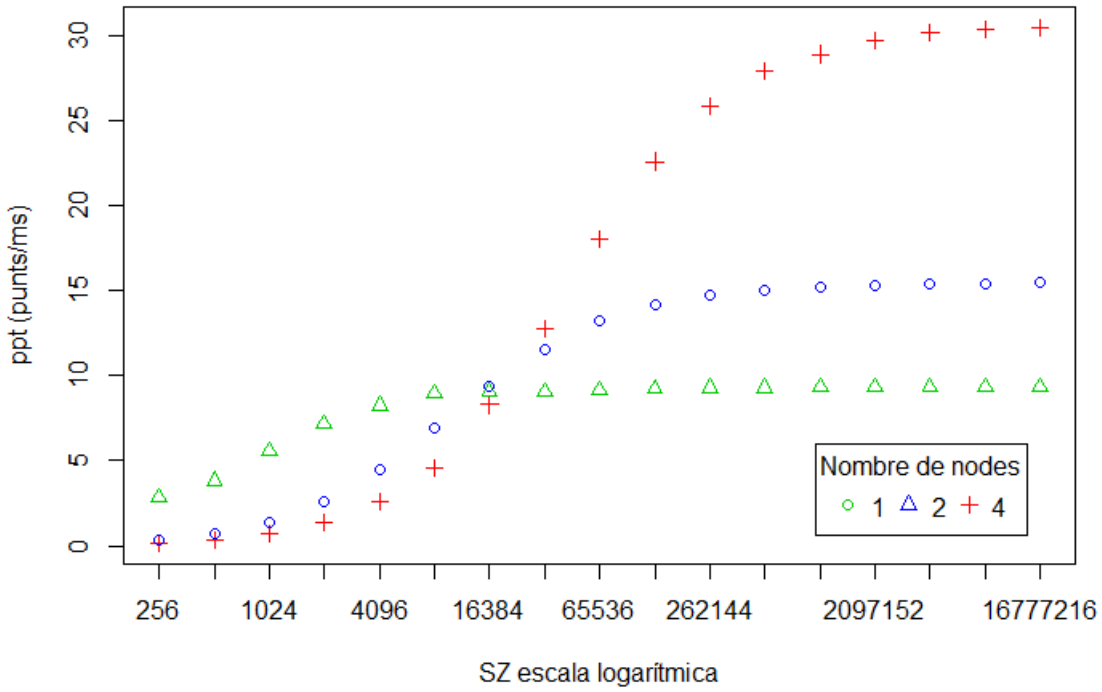


Figura 27: Rendiment mesurat en punts/ms del mètode Stencil

En aquest plot es poden veure tres casos diferenciats. Un cas inicial per valors menors a $2 \cdot e^{14}$ en els que el overhead fa que el programa sèrie sigui capaç de calcular més punts per unitat de temps. Una segona zona fins aproximadament un SZ=$2 \cdot e^{20}$ en la que els programes paral·lels van cada cop més ràpid que el sèrie i, finalment, a partir d'aquest punt, el paràmetre ppt s'estanca i s'arriba a un output màxim/òptim degut segurament a la memòria i la capacitat de còmput de les plaques.

Com a comentari final, remarcar que les diferents mesures s'han pres sobre el temps real de CPU en milisegons utilitzat pel programa i no entre que es llançava i finalitzava. Per fer-ho s'ha usat l'eina *perf* que ens permet fer un *profiling* del programa i mesurar-ne el rendiment.

En el projecte s'han intentat aconseguir mesures de valors com poden ser el nombre de cicles o el nombre d'instruccions del programa. Tot i que les Raspberry Pis tenen uns comptadors que permeten fer mesures d'aquests paràmetres, estan desactivats per defecte. S'han intentat activar per mitjà de mòduls externs al *kernel* de la distribució que es tenia però no s'ha aconseguit. També s'ha provat amb actualitzacions del *kernel* i altres imatges, però tampoc s'ha aconseguit. Tot i ser un camp interessant, quedava fora de l'àmbit del projecte i s'ha decidit deixar-ho estar i avaluar els resultats i els temps.

Capítol 4

4 Pràctiques

L'altre objectiu principal d'aquest projecte és el de dissenyar unes pràctiques acadèmiques prou complertes com per impartir-se a un curs teòric de programació paral·lela o que contingui els fonaments necessaris per a que l'estudiant pugui dur a terme les pràctiques i donar ús al que aprèn.

Per dur a terme l'objectiu s'han creat la documentació necessària per dur a terme dues pràctiques basades en els programes vistos al capítol anterior. Aquesta documentació consta de dos guions de pràctiques pels alumnes i un full de ruta pel professor encarregat d'impartir-les.

Degut a la dificultat per trobar una guia sobre com escriure guions de pràctiques, s'ha seguit un criteri personal basat en les pràctiques realitzades durant la carrera, i s'ha validat preguntant a altres companys. El criteri seguit inclou una introducció per posar el alumne en context, un desenvolupament on es descriu la pràctica ordenadament però sense ser pas a pas, deixant lloc a que els alumnes puguin prendre les seves decisions, i argumentar perquè no han realitzat la pràctica d'una altra manera, així com raonar i valorar si un enfoc diferent de la solució hauria sigut millor. Finalment s'inclou una part d'avaluació per a que es pugui donar un criteri segons el que puntuar el treball dels alumnes.

Els guions de pràctiques estan estructurats per:

- Objectiu: Es planteja la motivació de la pràctica.
- Introducció: S'introdueix el mètode a tractar i es proposa un exercici previ.
- Guió: Es donen indicacions i consells de com realitzar la pràctica.
- Resultats: S'especifiquen els resultats i lliuraments finals que es demanen.
- Opcionals: Es plantegen problemes addicionals que complementen la pràctica.

I s'han escrit per a que els alumnes siguin capaços de desenvolupar-les sense incidències. Estan plantejades per a poder-se adaptar a diferents situacions. Ja sigui temporals podent-se allargar més o menys sessions o de nivell, podent donar més pes als opcionals en funció de la capacitat dels alumnes.

El full de ruta pel professor descriu l'objectiu de les pràctiques, possibles incidències que podrien ocórrer als alumnes i comentaris diversos sobre les puntuacions. Es considera que complementats amb la memòria d'aquest projecte són suficients per impartir les pràctiques.

Tota la documentació es pot trobar recollida a l'annex A d'aquesta memòria.

- Guions de pràctiques
  A) Pràctica 1
  B) Pràctica 2
- Full de ruta.

Capítol 5

5 Conclusions

Aquest treball es va començar amb la motivació personal d'adquirir coneixements en un camp com és el de la programació paral·lela. Com a pretext per treballar-hi es va complementar amb l'objectiu d'escriure unes pràctiques acadèmiques ja que no hi ha millor manera d'assegurar-se que s'entén una matèria que explicant-la.

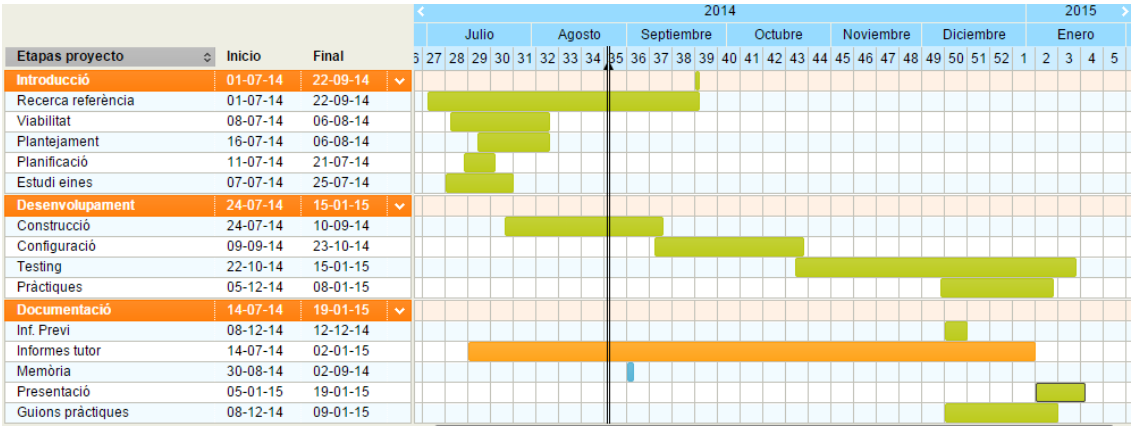
Un cop finalitzat el projecte la conclusió final és que s'han complert tots els objectius tant a nivell acadèmic com personal i que el resultat final és el desitjat. En la següent subsecció s'analitzaran els resultats dels dos objectius principals del projecte. Essent conscient de les dificultats de combinar feina, universitat i projecte, aquest projecte es va començar l'estiu passat. Tot i fer una planificació poc restrictiva i amb marges, aquesta s'ha vist afectada per diferents incidències al llarg del treball. En la subsecció corresponent es revisa la planificació. Finalment, a l'última subsecció del capítol es proposen possibles línies de treball que deixa obertes aquest projecte i millores a realitzar sobre la feina feta.

5.1 Compliment dels objectius

En aquest projecte s'havien plantejat dos objectius. El primer d'ells consistia en construir un sistema capaç d'executar programes paral·lels fent ús de plaques Raspberry Pi's. El segon es basava en el primer per dissenyar unes pràctiques acadèmiques que complementessin un curs acadèmic sobre programació paral·lela.

El sistema s'ha construït satisfactòriament i, tot i obtenir uns resultats potser no excessivament popers als teòricament perfectes, es dóna l'objectiu per complert. Es recalca que el material disponible era limitat i que la potència de les plaques és baixa. Tot i així, això no desmeritza el fet que, com es buscava, s'ha construït un sistema funcional. Els resultats, tot i ser importants, estaven en un segon pla. Pel que fa a l'objectiu principal de disseny d'unes pràctiques, també es considera el resultat com satisfactori. S'han generat dos guions que són concrets i assequibles però sense deixar de ser complets. La combinació que es buscava per les pràctiques. Es pot afirmar doncs que s'han complert els objectius satisfactòriament, aconseguint com a producte final una metodologia per construir un clúster format per plaques Raspberry Pi i la documentació necessària per realitzar unes pràctiques acadèmiques que en facin ús.

5.2 Revisió de la planificació



Etapas proyecto	Inicio	Final
Introducció	01-07-14	22-09-14
Recerca referència	01-07-14	22-09-14
Viabilitat	08-07-14	06-08-14
Plantejament	16-07-14	06-08-14
Planificació	11-07-14	21-07-14
Estudi eines	07-07-14	25-07-14
Desenvolupament	24-07-14	15-01-15
Construcció	24-07-14	10-09-14
Configuració	09-09-14	23-10-14
Testing	22-10-14	15-01-15
Pràctiques	05-12-14	08-01-15
Documentació	14-07-14	19-01-15
Inf. Previ	08-12-14	12-12-14
Informes tutor	14-07-14	02-01-15
Memòria	30-08-14	02-09-14
Presentació	05-01-15	19-01-15
Guions pràctiques	08-12-14	09-01-15

Respecte el planing original hi ha un augment destacable en hores respecte la tasca referent al *testing* del clúster i de la memòria.

5.3 Possibles millores i futures línies de treball

Com a possibles millores es proposa:

- Fer un estudi dels errors del segon programa del joc de proves.
- Ampliar el joc de proves.
- Activar els comptadors hardware i fer un estudi de mètriques com el CPI o nombre de fallades de memòria cau (caché).

Com a futures línies de treball es planteja:

- Passar d'un àmbit local a un àmbit global escalant el sistema i provant diferents topologies.
- Augmentar la freqüència de rellotge i mirar com afecta als resultats.
- Aprofitar el hardware de les Raspberry Pis (pins entrada/sortida) per fer un projecte complet com pot ser construir un robot.

Bibliografia

Referències

[1] Joshua Kiepert, *Creating a Raspberry Pi-Based Beowulf Cluster*.
http://coen.boisestate.edu/ece/files/2013/05/Creating.a.Raspberry.Pi-Based.Beowulf.Cluster_v2.pdf ,
Maig 22, 2013.

[2] David Guill, *Like Magic Apears*.
http://likemagicappears.com/projects/raspberry-pi-cluster/ ,
2014.

[3] Simon J. Cox, James T. Cox, Richard P. Boardman, Steven J. Johnston, Mark Scott, Neil S. O'Brien, *Raspberry Pi at Southampton*.
https://www.southampton.ac.uk/∼sjc/raspberrypi/ ,
Juny 2013.

[4] Floris Bos, *BerryBoot v2.0 - bootloader / universal operating system installer*.
https://github.com/maxnet/berryboot ,
http://www.berryterminal.com/doku.php/berryboot ,
Última modificació Setembre 3, 2014.

[5] G.A. Mikhailov, *Monte Carlo method*.
Encyclopedia of Mathematics, Springer, ISBN 978-1-55608-010-4,
http://www.encyclopediaofmath.org/index.php/Monte-Carlo_method ,
Última modificació Febrer 7, 2011.

[6] Enrique Zuazua, *Métodos Numéricos de resolución de Ecuaciones en Derivadas Parciales*.
http://www.uam.es/personal_pdi/ciencias/ezuazua/informweb/notas-05_065.pdf ,
Febrer 11, 2007.

[7] Andrew K. Denis, *Raspberry Pi Supercluster*.
Novembre 2013.

[8] Sean McManus, Mike Cook, *Raspberry Pi for dummies*.
2013.

A Guions de pràctiques

En aquest apèndix es presenta la documentació generada com a producte del projecte.

A.1 Pràctica 1

Objectiu

Aquesta primera sessió servirà d'introducció al món del còmput paral·lel en un sistema multi-node. Al finalitzar-la, s'haurà vist com desenvolupar i executar programes paral·lels en un clúster.

Introducció

Al laboratori es disposa d'un clúster format per Raspberry Pi's, petits computadors en una sola placa de baix consum i baix cost. Podeu veure un petit esquema a la seguent figura.

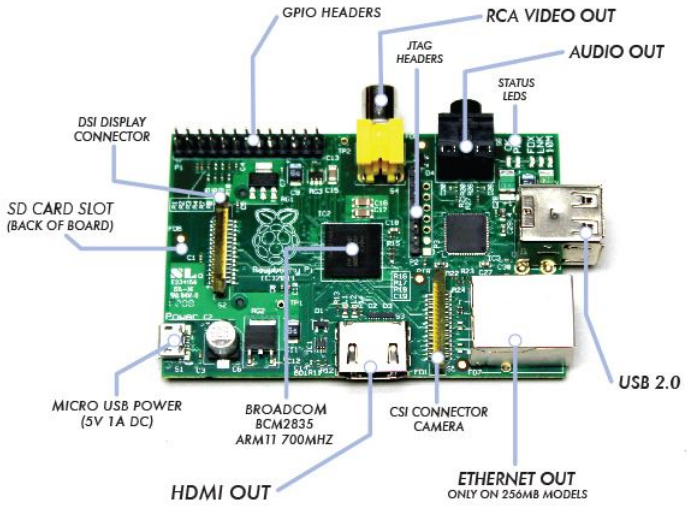


Figura 28: Esquema Raspberry Pi

Com podeu veure, la placa disposa entre d'altres d'un connector HDMI, dos port USB, diversos pins d'entrada/sortida, port ethernet, un procesador ARM, un connector de corrent, un port per la targeta SD i uns leds. Components que fareu servir en més o menys mesura.

Les plaques ja tenen instal·lat el software MPICH. Aquest software implementa les comandes MPI en el llenguatge de programació C que permetran fer fàcilment les comunicacions entre nodes.
La programació paral·lela s'usa sovint en àmbits científics. Aprofitant aquest fet en aquestes pràctiques es planteja programar mètodes de càlcul numèric i paral·lelitzar-los.

Per treballar tots aquests elements i entendre els conceptes de la programació paral·lela en sistemes multi-node en aquesta pràctica es proposa calcular una estimació del nombre pi.
Per dur a terme l'aproximació es farà servir el mètode estadístic de Montecarlo. Aquest consisteix en tirar dards a un tauler quadrat sense apuntar a cap lloc en concret. A continuació, si dibuixem

un cercle inscrit al quadrat i comptem el nombre de dards que han caigut dintre (dc) i el nombre total de dards tirats (dt), podem fer una estimació de 'pi' aplicant la fórmula:

$$\pi = \frac{4 \cdot dc}{dt}$$

Això és gràcies a la relació entre les àrees de les dues figures geomètriques.

Exercici previ: Fent ús de les fórmules de l'àrea del cercle i del quadrat, deduïu la fórmula de l'aproximació de π anterior.

En termes computacionals s'han de generar N punts aleatoris [3] al domini $[-1, 1]x[-1, 1]$ i, fent servir la fórmula anterior, calcular una aproximació de π.

*nGuió

En aquesta pràctica s'ha d'escriure un programa que implementi el mètode de Montecarlo per calcular una aproximació de 'pi'. S'ha de fer tant la versió sèrie com la versió paral·lela. Finalment s'hauran d'avaluar els resultats de les execucions dels dos programes i fer-ne una explicació.

Pel cas sèrie, el programa podria seguir una estructura com la que es mostra a continuació:

MontecarloSerie.c

```
/* Llibreries */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/* Definicio de constants (editables en temps de compilacio) */
#ifndef N
#define N 1000  //punts a generar per cada iteracio
#endif
#ifndef ITER
#define ITER 10 //nombre total d'iteracions
#endif

double calcul_pi(int N){

        do (N vegades){

        /* Generar N punts aleatoris */
                punt = Punt_aleatori();
        /* Determinar si el punt generat cau dintre el cercle i
        sumar en un comptador si cal */
                if (punt_cercle == True) comptador++;
        }
        /* Calcular pi amb la formula i retornar el valor */
        pi=estimar_pi;
        return (pi);
}

main{
        do( ITER vegades){
```

[3] Busqueu informació sobre la funció *random* i altres funcions relacionades amb aquesta.

```
            /* Calcular aproximacio parcial de 'pi' */
            pi_parcial = calcul_pi(N);

            /* Fer la mitja aritmetica dels valors parcials de 'pi' */
            pi = mitja_aritmetica(pi, pi_parcial);
        }

        /* Mostreu els resultats que considereu oportuns */
        printf( RESULTATS );
}
```

[4]

Editeu un fitxer anomenat MonteCarloSerie-*núm.subgrup*.c que implementi el algorisme descrit amb anterioritat. A continuació editeu un fitxer MonteCarlo-*núm.subgrup*.c on implementeu la paral·lelització d'aquest programa.
Per dur-la a terme es demana que feu servir comandes MPI. Trobareu una API amb exemples i totes les funcions disponibles a:

API de comandes MPI: http://mpi.deino.net/mpi_functions/index.htm

En aquesta pràctica us serà imprescindible enviar i rebre missatges.

Per executar els programes en el clúster us farà falta compilar-lo. Podeu dur a terme aquesta tasca amb la comanda:

```
> mpicc -g -DPUNTS=X -DITER=Y -o $PATH/MC $PATH/MonteCarlo.c
```

on $PATH és la ruta al vostre codi i X i Y són els valors donats a les constants PUNTS i ITER respectivament en temps de compilació.

A continuació, per poder executar el vostre binari us farà falta copiar-lo a la resta de plaques. Això ho podeu fer amb la comanda:

```
>scp $PATH/MC pi@<Direccio-IP>:$PATH
```

que us permetrà copiar el vostre executable MC a la mateixa direcció en la que es trobi actualment a la placa. És imprescindible per poder executar el programa en paral·lel que els executables es trobin a la mateixa ruta.

Per obtenir les direccions IP de la resta de plaques del vostre clúster podeu mirar els diferents fitxers pifile en el que trobareu les direccions i que són necessaris per executar el programa en paral·lel. Per dur a terme l'execució empreu la comanda:

```
> mpiexec -f $PATH/pifile<#Nodes> -n <#Nodes> $PATH/MC
```

On *Nodes* és el nombre de nodes que voleu fer servir per l'execució. Teniu en compte que per executar el programa s'usaran els N nodes definits al fitxer pifile*Nodes* així que aquests han de contenir una copia de l'executable.

Resultats

S'haurà d'entregar una memòria autocontinguda en la que expliqueu quina solució heu implementat i perquè. S'hauran de descriure també els resultats obtinguts i fer-ne una petita avaluació.

[4] Mitja aritmètica: http://en.wikipedia.org/wiki/Arithmetic_mean

Opcionals

- Estudieu la funció MPI_Reduce i implementeu-la al programa paral·lel.
- Quines utilitats creieu que pot tenir paral·lelitzar un programa similar a aquest si tenim un temps limitat d'execució?
- Penseu en el càlcul necessari per fer una predicció meteorològica d'un dia per un altre.

A.2 Pràctica 2

Objectiu

En aquesta segona pràctica es busca tractar els punts principals de la programació paral·lela. Es treballaran aspectes com el solapament de comunicació i còmput o la repartició de la feina a realitzar entre diferents tasques.
Al finalitzar la pràctica s'hauran tractat les nocions bàsiques de còmput paral·lel.

Introducció

En aquesta sessió s'implementarà un mètode per modelitzar la transmissió de calor en una dimensió, que anomenarem Stencil.

Aplicant el mètode de les diferencies finites [5] es pot discretitzar una equació diferencial que calculi la temperatura en un vector d'una sola dimensió d'un material donat.

Originalment tenim la equació diferencial:

$$\frac{\partial u(x,t)}{\partial t} = \alpha \cdot \frac{\partial^2 u(x,t)}{\partial x^2}$$

Aplicant diferències finites podem fer una partició del vector i aplicar la substitució:

$$\frac{\partial u}{\partial t} = \frac{u_k^{j+1} - u_k^j}{\Delta t}$$

i també podem substituir

$$\frac{\partial^2 u(x,t)}{\partial x^2} = \frac{u_{k+1}^j - 2 \cdot u_k^j + u_{k-1}^j}{\Delta x^2}$$

aïllant a l'equació original podem obtenir una fórmula recursiva que aproxima el valor de la solució a l'equació diferencial en els punt de la partició.

Exercici previ:Trobeu la fórmula recursiva que calculi u_k^{j+1} a partir de la mateixa posició en el temps anterior i les posicions dels costats.

Nota: us farà falta substituir

$$\lambda = \frac{\alpha \cdot \Delta t}{\Delta x^2}$$

En termes computacionals tindrem un vector amb tantes posicions com particions es desitgin i anirem iterant la fórmula recursiva en un bucle per calcular l'aproximació de la solució.

[5] http://en.wikipedia.org/wiki/Finite_difference_method

Guió

En aquesta pràctica s'ha d'escriure un programa que implementi el mètode Stencil tant en la seva versió sèrie com en la versió paral·lela. Finalment s'hauran d'avaluar els resultats de les execucions dels dos programes i fer-ne una explicació.

El programa consta de diversos paràmetres: la mida del vector SZ (recordeu que equival a les particions que fem), el nombre d'iteracions que realitzarem, el paràmetre lambda i finalment el valor inicial del focus de calor.

El programa consta de varies parts

- Inicialització: actualitzareu tot el vector de mida SZ a zero menys les posicions centrals que inicialitzarem segons el paràmetre d'entrada. Un node haurà d'encarregar-se'n
- Repartició: En funció del nombre de nodes amb els que treballeu haureu de particionar i distribuir el vector
- Iterar: Haureu d'aplicar la fórmula recursiva tantes vegades com s'indiqui al paràmetre corresponent. És important que penseu en els vectors dels extrems
- Recollida: un node haurà de recollir tota la informació de la resta
- Finalització: Heu d'alliberar els recursos creats i mostrar el resultat final

Nota: és important que feu servir memòria dinàmica per poder treballar amb mides de vector grans.

La complexitat de la pràctica es troba en els iterats. És important que plantegeu bé les comunicacions entre nodes per no crear deadlocks i que pugeu fer el càlcul del valor dels extrems sense problemes.

Resultats

S'haurà d'entregar una memòria autocontinguda en la que expliqueu quina solució heu implementat i perquè. S'hauran de descriure també els resultats obtinguts i fer-ne una petita avaluació.

Opcionals

- Implementeu una versió del programa en la que utilitzeu comunicacions entre nodes no blocants. Com afecta als temps d'execució?
- Quins paràmetres creieu que afecten a l'error del mètode?
- Com implementaríeu un checkpoint? Quines utilitats li veieu?

A.3 Full de ruta

Objectiu

Aquesta primera sessió servirà d'introducció al món del còmput paral·lel en un sistema multi-node. Al finalitzar-la, s'haurà vist com desenvolupar i executar programes paral·lels en un clúster.

Incidències

Cal remarcar que per a les execucions multinode és important que els executables es trobin en totes les maquines sota el mateix *path*. Que es tinguin permisos d'execució i que el *pifile* que s'utilitzi sigui correcte.

Existeix la possibilitat que els alumnes col·lapsin la memòria d'una placa o corrompin el sistema de fitxers. Seria recomanable tenir *spares* (recanvi de components hardware) per poder fer un canvi i que continuïn treballant.

Comentaris

Es demana que els alumnes tinguin unes nocions mínimes de C i que tinguin o estiguin aprenent els conceptes teòrics de la programació paral·lela. Tot i així es poden adaptar les pràctiques per fer servir un altre llenguatge de programació. Aquest fet implicaria buscar una eina similar a MPI per fer la paral·lelització o treballar amb sockets del sistema. Les pràctiques i la configuració del sistema descrita al PFC són suficientment genèriques com per permetre aquest canvi.

Dels alumnes només s'espera que aprenguin a programar pel que si hi ha qualsevol problema de configuració del clúster no se'n haurien d'encarregar. Per qualsevol incidència referenciar-se al PFC.

Els alumnes poden treballar en grups de dos i s'espera que les sessions siguin presencials. S'ha d'anar amb compte si els alumnes porten codi de casa ja que poden haver-hi problemes de format. Eines com *dos2unix* poden resultar molt útils en aquest cas.

Pel que fa a la puntuació, es pot donar més o menys valor a les parts opcionals en funció de les sessions disponibles per dur a terme les pràctiques.

B Codi del joc de proves

B.1 Monte Carlo

Sèrie

```
/*      llibreries usades.      */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*      Constants/parametres    */
#ifndef PUNTS
#define PUNTS 10000      // nombre de punts per iteracio
#endif
#ifndef ITER
#define ITER 100         // nombre d'iteracions
#endif
#ifndef MTYPE
#define MTYPE double
#endif


double calcul_pi(int punts)
{

        int i;
        double x,y, pi;      /*  coordenades del punt    */
        double counter=0;     /* comptador de punts que cauen al cercle */
        unsigned int max;
        for (i=0; i<punts; i++){
                x= 2*((double)random() / ((double)RAND_MAX + 1.0))-1;
                y= 2*((double)random() / ((double)RAND_MAX + 1.0))-1;
                if( (x*x + y*y)<=1.0 ){counter++;}
        }
        pi = 4.0 * (double)counter/(double)punts;
        return(pi); /*  retornem el valor aproximat de PI.      */
}


int main (int argc, char *argv[])
{
/*      variables.      */
double  local_pi,           //  valor de pi de cada tasca
        total_local_pi,     //  suma dels valors locals de pi
        parcial_pi,         //  valor de pi parcial de cada iteracio
        total_pi;           //  valor de pi de l'acumulat d'iteracions
int     rank,               //  rang de cada tasca i llavor
        world,              //  nombre total de tasques
        codi_retorn,        //  codi retorn per les comandes MPI
        i;
```

```
srandom (time(NULL));
total_pi = 0.0;
for (i = 0; i < ITER; i++) {//pel total d'iteracions
    local_pi = calcul_pi(PUNTS);
    total_pi = ((total_pi * (i*1.0)) + local_pi)/(i + 1.0);

}

    printf("\n Fets servir %8d PUNTS, el valor de pi es: %10.16f\n",
                (PUNTS * (i)),total_pi);
    printf ("\n Error del metode de: %10.16f\n i
                err. per n m punts %10.16f\n",
                (3.1415926535897- total_pi),
                (3.1415926535897- total_pi)*(PUNTS * (i)));

return 0;
}
```

Paral·lel

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <mpi.h>

#ifndef PUNTS
#define PUNTS 10000      // nombre de punts per iteracio
#endif
#ifndef ITER
#define ITER 100         // nombre d'iteracions
#endif
#ifndef MTYPE
#define MTYPE double
#endif

double calcul_pi(int punts)
{

        int i;
        double x,y, pi;
        double counter=0;
        unsigned int max;
        for (i=0; i<punts; i++){
                x= 2*((double)random() / ((double)RAND_MAX + 1.0))-1;
                y= 2*((double)random() / ((double)RAND_MAX + 1.0))-1;
                if( (x*x + y*y)<=1.0 ){counter++;}
        }
        pi = 4.0 * (double)counter/(double)punts;
        return(pi);
}

int main (int argc, char *argv[])
{

double  local_pi,
        total_local_pi,
        parcial_pi,
        total_pi;
int     rank,
        world,
        codi_retorn,
        i;

MPI_Status stat;        /*      flag per a les comunicacions    */

/*      Creem les tasques MPI   */
MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&world);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);
printf ("Tasca %d creada.\n", rank);
```

```
srandom ((rank)+time(NULL));


total_pi = 0.0;
for (i = 0; i < ITER; i++) {


     local_pi = calcul_pi(PUNTS);
     codi_retorn = MPI_Reduce(&local_pi, &total_local_pi, 1,
                                    MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
    if (codi_retorn != MPI_SUCCESS){
                printf("Error en el reduce. Tasca: %d\n", rank);
          }

    if (rank == 0) {
       parcial_pi = total_local_pi/(world*1.0);
       total_pi = ((total_pi * (i*1.0)) + parcial_pi)/(i + 1.0);
    }
}
if (rank == 0){
         printf("\n Fets servir %8d PUNTS i dos nodes, el valor de pi es:
                     %10.16f\n", (PUNTS * (i)),total_pi);
      printf ("\n Error del m tode de:
                          %10.16f\n i err. per num punts %10.16f\n",
                 (3.1415926535897- total_pi),
                 (3.1415926535897- total_pi)*(PUNTS * (i)));
}
MPI_Finalize();
return 0;
}
```

B.2 Stencil

Sèrie

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#ifndef SZ // Tamany del vector
#define SZ 8
#endif
#ifndef MTYPE // Tipus de dades de les caselles.
#define MTYPE float
#endif
#ifndef REP // Repeticions
#define REP 256
#endif
#ifndef INIT // Valor inicial de calor
#define INIT 256
#endif
#ifndef CNT // Coeficient de propagaci
#define CNT 256
#endif



float calculH(float L, float M, float R)
{
  float resultat=0;
  resultat= M + CNT*(L + R - 2*M);
  return(resultat);
}

void calcul(float *final, float *inici)
{
  int i;
  for(i=1; i<SZ+1; i++){
      final[i]= inici[i] + CNT*(inici[i-1] + inici[i+1] - 2*inici[i]);
  }
}


int main(int argc, char* argv[])
{
    int i,j,k,       // variables auxiliars de bucle
        matriu=0,    // boolea que indica amb quina matriu treballem
        world=1,     // nombre total de tasques
        repeticions // nombre total de iteracions de calcul que farem
        ;
    float   *mat1, *mat2;   // Vectors on anirem aplicant els calculs

    /*  Alocatem l'espai on farem els calculs   */
        mat1 = (float*) malloc (sizeof (float) * (SZ +2));
        mat2 = (float*) malloc (sizeof (float) * (SZ +2));
```

```
        /*      Inicialitzaci del vector + buffer      */
        for ( i = 0 ; i < SZ+2; i++) {
            mat1[i] = 0;
        mat2[i] = 0;
        }
        if ( SZ % 2 != 0 ){mat1[(int)(SZ)/2+1] = INIT;}
        else {
            mat1[(int)(SZ)/2+1] = INIT ;
            mat1[(int)(SZ)/2] = INIT ;
        }

        for ( repeticions=0 ; repeticions<REP ; repeticions++) {
        /*      Canviem la matriu objectiu del calcul   */
            matriu++;
            matriu = matriu%2;
            /*  Calculem el que resta del vector        */
        if(matriu){
                        calcul(mat2, mat1);
            }else{
                    calcul(mat1, mat2);
            }
        }

    free(mat1);
    mat1=NULL;
    free(mat2);
    mat2=NULL;
    printf("\n Vector de tamany %d, %d-iteracions, %d-nodes.\n",
                SZ ,REP, world);

    return 0;
}
```

Paral·lel

```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


#ifndef SZ
#define SZ 256
#endif
#ifndef MTYPE
#define MTYPE float
#endif
#ifndef REP
#define REP 256
#endif
#ifndef INIT
#define INIT 256
#endif
#ifndef CNT
#define CNT 256
#endif

#ifndef DEBUG
#define DEBUG 0
#endif

  /*    Tags dels missatges
   * 0: Enviar les dades inicials (Master to Slave)/Rebre les dades inicials
   * 1: Enviar Halo L / Rebre Halo R
   * 2: E H R / R H L
   * 3: E resultat final / R resultat final
   */


float calculH(float L, float M, float R)
{
  float resultat=0;
  resultat= M + CNT*(L + R - 2*M);
  return(resultat);

}

void calcul(float *final, float *ini, int world)
{
        int i;
        for(i=2; i<SZ/world; i++){
        final[i]= ini[i] + CNT*(ini[i-1] + ini[i+1] - 2*ini[i]);
        }
}
void calculPare(float *final, float *ini, int limit)
{
```

```
        int i;
        for(i=2; i<limit; i++){
                final[i]= ini[i] + CNT*(ini[i-1] + ini[i+1] - 2*ini[i]);
        }
}


int main(int argc, char* argv[])
{

  /*    variables.      */
    int i,j,k,          //      variables auxiliars de bucle
    matriu=0,           //      boolea que indica amb quina matriu treballem
    rank,               //      rang de cada tasca
    world,              //      nombre total de tasques
    codi_retorn,        //      codi retorn per les comandes MPI
        repeticions,
    lim;
    float   *mat1, *mat1f, // Vectors on anirem aplicant els calculs
    *mat2, *mat,
    *buffer,
    haloR[1], haloR2[1],
    haloL[1], haloL2[1];


    /* flags per a les comunicacions   */
    MPI_Request requestSendL,requestSendR, requestRecieve;
    MPI_Status stat;

    /* Creem les tasques MPI   */
    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&world);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);

    if (rank==0){/*     Codi de la tasca Pare   */
        /*      Alocatem l'espai on farem els calculs en funcio de SZ
            i dues caselles adicionals de halo  */
        mat = (float*) malloc (sizeof (float) * (SZ+2));
        mat1 = (float*) malloc (sizeof (float) * (SZ/world +2));
        mat2 = (float*) malloc (sizeof (float) * (SZ/world +2));
        buffer = (float*) malloc (sizeof (float) * (SZ/world +2));

        /*      Inicialitzacio del vector + buffer      */
        for ( i = 0 ; i < SZ+2; i++) {  mat[i] = 0;}
        for ( i = 0 ; i < SZ/world+2; i++) {
           mat1[i] = 0;
           mat2[i] = 0;
        }

        /*      Posem el focus de calor a la matriu     */
        if ( SZ % 2 != 0 )  {  mat[(int)(SZ)/2+1] = INIT ;  }
        else  {
          mat[(int)(SZ)/2+1] = INIT ;
          mat[(int)(SZ)/2] = INIT ;
```

```
}


/*      Repartim el treball entre els diferents processos */
for(i=0; i<world-1; i++){
        for(j=0 ; j<SZ/world+2; j++){
            buffer[j] = mat[(j+i*(SZ/world))];
        }
        MPI_Send(buffer, (SZ/world)+2, MPI_FLOAT,
                    i+1, 0, MPI_COMM_WORLD);
}

lim= (SZ+1)-((SZ/world)*(world-1)-1);
for ( j = 0 ; j < lim; j++){
        mat1[j] = mat[(SZ/world)*(world-1)-1 +j];
}

for ( repeticions=0 ; repeticions<REP ; repeticions++)
{
        matriu++;
        matriu = matriu%2;
        if(matriu){
                mat2[1]=calculH(mat1[0], mat1[1], mat1[2]);
                haloL[0]=mat2[1];
        }else{
                mat1[1]=calculH(mat2[0], mat2[1], mat2[2]);
                haloL[0]=mat1[lim-2];
        }
        /*      Enviem el halo L */
        if (repeticions!=0){
        MPI_Wait(&requestSendL, MPI_STATUS_IGNORE);
        }
        MPI_Isend(haloL, 1 , MPI_FLOAT, world-1,
                    1, MPI_COMM_WORLD, &requestSendL);

        if(matriu){ calculPare(mat2, mat1, lim);}
        else{ calculPare(mat1, mat2, lim);}

        MPI_Recv(haloL2, 1, MPI_FLOAT, world-1,
                    2, MPI_COMM_WORLD, &stat);
        if(matriu){ mat2[0]=haloL2[0]; }
        else{ mat1[0]=haloL2[0]; }
}

/*      Rebem el treball dels diferents processos       */
for(i = 0 ; i < world-1; i++){
        MPI_Recv(buffer, (SZ/world), MPI_FLOAT,
                    MPI_ANY_SOURCE, 3, MPI_COMM_WORLD, &stat);

        if(matriu){
        for ( j = 0 ; j < SZ/world+2; j++){
            mat[(SZ/world)*(stat.MPI_SOURCE-1)+j] = buffer[j];
        }
        }else{
```

```
                for ( j = 0 ; j < SZ/world+2; j++) {
                    mat[(SZ/world)*(stat.MPI_SOURCE-1)+j] = buffer[j];
                }
                }
        }
        /* Copiem el treball del pare a la matriu resultat */
        for ( j = 1 ; j < lim-1; j++)
        {
            if(matriu){
                mat[(SZ/world)*(world-1) +j] = mat2[j] ;
            }
            else{
                mat[(SZ/world)*(world-1) +j] = mat1[j] ;
            }
        }

        free(buffer);
        buffer=NULL;
        free(mat1);
        mat1=NULL;
        free(mat2);
        mat2=NULL;
        free(mat);
        mat=NULL;
        }/*     Fi codi de la tasca Pare        */
/*************************************************************/
    else{ /*    Codi de la tasca fill   */
        /*      Alocatem l'espai on farem els calculs en funcio
            de SZ/world i dues caselles adicionals de halo      */
        mat1 = (float*) malloc (sizeof (float) * (SZ/world+2));
        mat2 = (float*) malloc (sizeof (float) * (SZ/world+2));
        buffer = (float*) malloc (sizeof (float) * ((SZ/world)+2));
        /*      Inicialitzaci del vector        */
        for ( i = 0 ; i < SZ/world+2; i++) {
        mat1[i] = 0;
        mat2[i] = 0;
        }

        MPI_Recv(mat1, (SZ/world) + 2 , MPI_FLOAT,
                    0, 0, MPI_COMM_WORLD, &stat);

        /*      Fem el calcul del treball       */
        for ( repeticions=0 ; repeticions<REP ; repeticions++)
            {
            /* Canviem la matriu objectiu del c lcul */
            matriu++;
            matriu = matriu%2;
            /* Calculem i enviem els halos      */
            if(matriu){
                mat2[(SZ/world)] = calculH(mat1[(SZ/world)-1],
                    mat1[(SZ/world)], mat1[(SZ/world)+1]);
                haloR[0]=mat2[(SZ/world)];
                if (rank !=1){
                        mat2[1] = calculH(mat1[0], mat1[1], mat1[2]);
```

```
                haloL[0]=mat2[1];
            }
        }else{
            mat1[(SZ/world)] =  calculH(mat2[(SZ/world)-1],
                        mat2[(SZ/world)], mat2[(SZ/world)+1]);
            haloR[0]=mat1[(SZ/world)];
            if (rank !=1){
                    mat1[1] = calculH(mat2[0], mat2[1], mat2[2]);
                    haloL[0]=mat1[1];
            }
        }

        /* Enviem els halo assegurant-nos que no hi
            ha cap missatge pendent pel mateix canal        */
        if (repeticions!=0){
            MPI_Wait(&requestSendR, MPI_STATUS_IGNORE);
            if (rank !=1){
                MPI_Wait(&requestSendL, MPI_STATUS_IGNORE);
            }
        }

        MPI_Isend(haloR, 1 , MPI_FLOAT, (rank+1)%world,
                    2, MPI_COMM_WORLD, &requestSendR);
        if (rank !=1){      MPI_Isend(haloL, 1 , MPI_FLOAT, rank-1,
                            1, MPI_COMM_WORLD, &requestSendL);      }

        if(matriu){calcul(mat2, mat1, world);}
        else{calcul(mat1, mat2, world);}

        /* Rebem el halo de l'esquerra que usarem a la seguent iter. */
        if (rank !=1){
            MPI_Recv(haloL2, 1, MPI_FLOAT,rank-1,2, MPI_COMM_WORLD, &stat);
            if(matriu){ mat2[0]=haloL2[0]; }
            else{ mat1[0]=haloL2[0]; }
        }
        MPI_Recv(haloR2, 1, MPI_FLOAT, (rank+1)%world,
                    1, MPI_COMM_WORLD, &stat);
        if(matriu){(SZ/world)+1]=haloR2[0];}
        else{mat1[(SZ/world)+1]=haloR2[0];}
    }/*     Final de les iteracions */

    /*      Enviem el treball al Pare       */
    if(matriu){
            for ( j = 1 ; j < SZ/world+1; j++){buffer[j] = mat2[j];}
    }else{
            for ( j = 0 ; j < SZ/world+1; j++) {buffer[j] = mat1[j];}
    }
    MPI_Send(buffer,SZ/world, MPI_FLOAT, 0, 3, MPI_COMM_WORLD);

    /*      Alliberem l'alocatacio de memoria       */
    free(buffer);
    buffer=NULL;
    free(mat1);
    mat1=NULL;
```

```
        free(mat2);
        mat2=NULL;

    }/* Fi codi de la tasca fill          */

    /*  Totes les tasques          */
    printf("Finalitzacio proces %d,amb REPS %d i SZ %d. \n", rank, REP, SZ);
    MPI_Barrier(MPI_COMM_WORLD);
    MPI_Finalize();
    return 0;
    }
```

C Codi en R

En aquest apèndix es presenta el codi fet servir per avaluar generar els *plots* i els models lineals del capítol 3 secció 3.

```
setwd("C:/Users/usuari/Downloads/PFC/mem")


data <- read.csv(file="resultatsMC.csv",head=TRUE,sep=";")
data
error <- abs(data$error)
lerror <- abs(log10(abs(data$error)))
punts <- (data$punts)
lpunts <- log10(data$punts)
iter <- data$iter
temps <- data$temps
ltemps <- log10(temps)
nodes<- factor(data$nodes)
formes <-data$nodes
colors <- c(3,4,10)


plot(punts[nodes==1], error[nodes==1], pch=1,  col=3, ylab="errors", xlab="punts")
lines(punts[nodes==2], error[nodes==2], col=4, pch=2, type="p")
lines(punts[nodes==4], error[nodes==4], col=10, pch=3, type="p")
legend("topright", inset=.05, title="Nombre de nodes", pch=c(1,2, 3),
        c("1","2","4"), col=colors, horiz=TRUE)

plot(punts[nodes==1], error[nodes==1],log="x", pch=1,  col=3,
        ylab="errors", xlab="nombre de punts (escala logarítmica)" )
lines(punts[nodes==2], error[nodes==2],log="x", col=4, pch=2, type="p")
lines(punts[nodes==4], error[nodes==4], log="x",col=10, pch=3, type="p")
legend("topright", inset=.05, title="Nombre de nodes", pch=c(1,2, 3),
        c("1","2","4"), col=colors, horiz=TRUE)

plot(punts[nodes==1], error[nodes==1],log="yx" , pch=1,  col=3,
        ylab="errors (escala logarítmica)", xlab="nombre de punts (escala logarítmica)" )
lines(punts[nodes==2], error[nodes==2],log="xy" , col=4, pch=2, type="p")
lines(punts[nodes==4], error[nodes==4], log="xy ",col=10, pch=3, type="p")
legend("topright", inset=.05, title="Nombre de nodes", pch=c(1,2, 3),
        c("1","2","4"), col=colors, horiz=TRUE)

fit = lm( lerror ~ lpunts + nodes-1)
fit
summary(fit)
plot(punts[nodes==1], temps[nodes==1],log="yx" , pch=1,  col=3,
        ylab="temps (escala logarítmica)", xlab="nombre de punts (escala logarítmica)" )
lines(punts[nodes==2], temps[nodes==2],log="xy" , col=4, pch=2, type="p")
lines(punts[nodes==4], temps[nodes==4], log="xy ",col=10, pch=3, type="p")
legend("bottomright", inset=.05, title="Nombre de nodes", pch=c(1,2, 3),
        c("1","2","4"), col=colors, horiz=TRUE)
```

```
fit = lm( ltemps ~ lpunts + nodes-1)
fit
summary(fit)


######################################################################
data <- read.csv(file="resultatsStencil.csv",head=TRUE,sep=";")
data

SZ <- data$SZ
lSZ <- log(data$SZ, 2)
iter <- data$iter
temps <- data$temps
ltemps <- log(data$temps, 2)
nodes<- factor(data$nodes)

plot(SZ[nodes==1], temps[nodes==1], pch=1, log="x", xaxt = "n", col=3,
        ylab="temps", xlab="punts en escala logarítmica", xaxt = "n")
axis(1,cbind(2^(1:30))) # draw y axis with required labels
lines(SZ[nodes==2], temps[nodes==2], col=4, pch=2, type="p")
lines(SZ[nodes==4], temps[nodes==4], col=10, pch=3, type="p")
legend("top", inset=.05, title="Nombre de nodes", pch=c(1,2, 3),
        c("1","2","4"), col=colors, horiz=TRUE)


plot(SZ[nodes==1], temps[nodes==1], pch=1, log="xy", xaxt = "n", col=3,
        ylab="temps escala logarítmica", xlab="punts en escala logarítmica", xaxt = "n")
axis(1,cbind(2^(1:30))) # draw y axis with required labels
lines(SZ[nodes==2], temps[nodes==2], col=4, pch=2, type="p")
lines(SZ[nodes==4], temps[nodes==4], col=10, pch=3, type="p")
legend("bottomright", inset=.05, title="Nombre de nodes", pch=c(1,2, 3),
        c("1","2","4"), col=colors, horiz=TRUE)




fit = lm( ltemps ~ lSZ + nodes-1)
fit
summary(fit)
plot(SZ[nodes==1],temps[nodes==1]/temps[nodes==4],pch=3, col=10,log="x", type="p",
        xaxt = "n",ylim=c(0,3.5), ylab="speed-up", xlab="SZ escala logarítmica" )
axis(1,cbind(2^(1:30))) # draw y axis with required labels
lines(SZ[nodes==1],temps[nodes==1]/temps[nodes==2], col=4,pch=2,log="x", type="p")
lines(SZ[nodes==1],temps[nodes==1]/temps[nodes==1], col=3,pch=1,log="x", type="p")
legend("bottomright", inset=.05, title="Nombre de nodes", pch=c(1,2, 3),
        c("1","2","4"), col=colors, horiz=TRUE)



ppt <- SZ/temps
fit = lm( ppt ~ lSZ + nodes - 1)
fit
summary(fit)
plot(SZ[nodes==4], ppt[nodes==4],col=10, pch=3,type="p", log="x",
```

```
        ylab="ppt (punts/ms)", xaxt = "n",xlab="SZ escala logarítmica")
axis(1,cbind(2^(1:30))) # draw y axis with required labels
lines( SZ[nodes==1],ppt[nodes==1], col=3,log="x",pch=2, type="p")
lines(SZ[nodes==2], ppt[nodes==2],col=4,pch=1,log="x", type="p")
legend("bottomright", inset=.05, title="Nombre de nodes", pch=c(1,2, 3),
        c("1","2","4"), col=colors, horiz=TRUE)
```

Resum

Aquest projecte planteja el disseny d'unes pràctiques acadèmiques sobre programació paral·lela. Per realitzar aquest disseny es duu a terme la construcció i configuració d'un clúster format per plaques Raspberry Pi en el que s'avalua el rendiment de l'execució d'uns programes paral·lels que s'han escrit en C. Aquests mateixos programes s'utilitzen posteriorment en l'elaboració de les pràctiques, que es basen en escriure'ls i avaluar els resultats de la seva execució en un sistema com el descrit en aquest projecte.

Resumen

Este proyecto plantea el diseño de unas prácticas académicas sobre programación paralela. Para realizar este diseño se realiza la construcción y configuración de un clúster formado por placas Raspberry Pi en el cual se evalúa el rendimiento de la ejecución de unos programas paralelos que se han escrito en C. Estos mismos programas se utilizan posteriormente en la elaboración de las prácticas, que se basan en escribirlos y evaluar los resultados de su ejecución en un sistema como el descrito en este proyecto.

Abstract

This thesis objective is to design practices guides about parallel computing. To do this design, a construction and configuration of a cluster form by Raspberry Pi's is done. In it, some programs written in C are tested to see how the cluster performs. This same programs are used later to elaborate the practices, which are about writing them and evaluating the results of their execution in a system similar to the one described in the project.