Tasca de laboratori HPC

Laboratori 1: Entorn d'execució paral·lel

JR Herrero i MA Senar

Primavera 2023

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Continguts

Entregable

Sessió 1

Muntatge experimental

L'objectiu d'aquesta primera sessió és familiaritzar-se amb l'entorn de maquinari i programari que utilitzarà durant el semestre per fer totes les tasques de laboratori en HPC. Del teu PC/terminal arrencat amb Linux1 accedireu a boada, un servidor multiprocessador situat a l'ordinador Departament d'Arquitectura de la UPC. Per connectar-s'hi haureu d'establir una connexió mitjançant el ordre d'intèrpret d'ordres segur: "ssh -X hpc1XYZ@boada.ac.upc.edu", sent hpc1XYZ el nom d'usuari assignat a tu. L'opció -X és necessària per reenviar les dades X11 i poder obrir finestres remotes el vostre escriptori local2 . Hauríeu de canviar la contrasenya del vostre compte utilitzant "ssh -t hpc1XYZ@boada.ac.upc.edu passwd"3 .

Un cop hàgiu iniciat sessió, us trobareu a qualsevol dels nodes interactius: boada-6 a boada-8, on podeu executar treballs interactius i des d'on podeu enviar treballs d'execució a la resta de els nodes de la màquina. De fet, boada es compon de diversos nodes (anomenats boada-1 a boada-15), equipat amb cinc generacions de processadors diferents, tal com es mostra a la taula següent:

Nom del node	Generació del processador	Partició interactiva	
boada-1 a 4	Intel Xeon E5645	No	execució 2
boada-6 a 8	Intel Xeon E5-2609 v4	Sí	interactiu
boada-9	Intel Xeon E5-1620 v4 + Nvidia K40c	No	cuda9
boada-10	Intel Xeon Silver 4314 + 4 x Nvidia GeForce RTX 3080	No	cuda
Boada-11 a 14	Intel Xeon Silver 4210R	No	execució
boada-15	Intel Xeon Silver 4210R + ASUS AI CRL-G116U-P3DF	No	iacard

Tanmateix, en aquest curs només utilitzareu els nodes boada-6 a boada-8 de manera interactiva i nodes boada-11 a boada-14 a través de la cua d'execució, tal com s'explica a la subsecció següent. La resta dels nodes tenen accés restringit i els usuaris d'HPC no poden enviar treballs als seus corresponents cues.

Tots els nodes tenen accés a un disc NAS (Network-Attached Storage) compartit; podeu accedir-hi a través /scratch/nas/1/hpcXXYY (de fet, aquest és el vostre directori personal, comproveu-ho escrivint pwd a l'ordre línia). A més, cada node de boada té el seu propi disc local que es pot utilitzar per emmagatzemar fitxers temporals no visible per a altres nodes; podeu accedir-hi a través de /scratch/1/hpcXXYY.

Publicarem tots els fitxers necessaris per fer cada tasca de laboratori a /scratch/nas/1/hpc0/sessions. Per a la sessió d'avui, heu d'extreure els fitxers del fitxer lab1.tar.gz des d'aquesta ubicació descomprimint-lo al vostre directori d'inici a boada.

Des de l'arrel del vostre directori d'inici, desempaqueteu els fitxers amb la següent línia d'ordres: "tar -zxvf /scratch/nas/1/hpc0/sessions/lab1.tar.gz".

1També podeu accedir des del vostre ordinador portàtil, arrencat amb Linux, Windows o MacOS X, si hi ha instal·lat un client shell segur.

2Opció -Y si us connecteu des d'un ordinador portàtil MacOS X amb XQuartz. Amb Windows, si feu servir massilla (https://www.chiark.greenend.org.uk/ÿsgtatham/putty/latest.html), llavors també necessita xming: (https://wiki.centos.org/HowTos/Xming). Alternativament, podeu utilitzar MobaXterm (https://mobaxterm.mobatek.net/download.html) que ho té tot integrat.

3L'ordre passwd s'executarà a boada. Després d'introduir correctament la contrasenya antiga dues vegades, se us demanarà la teva nova contrasenya també dues vegades.

Per configurar totes les variables d'entorn, heu de processar el fitxer environment.bash ara disponible al vostre directori d'inici amb "source ÿ/environment.bash". Nota: com que ho heu de fer cada vegada que inicieu sessió al compte o obriu una nova finestra de consola, us recomanem que afegiu aquesta línia d'ordres al fitxer .bashrc del vostre directori d'inici, un fitxer que s'executa cada vegada que es fa una nova sessió. s'inicia. Si heu desempaquetat els fitxers lab1.tar.gz al vostre directori d'inici, aquest fitxer ja s'ha afegit automàticament al vostre directori d'inici i s'utilitzarà des de la vostra propera connexió a Boada.

En cas que necessiteu transferir fitxers de boada a la vostra màquina local (ordinador portàtil o escriptori a la sala de laboratori), o viceversa, heu d'utilitzar l'ordre de còpia segura scp. Per exemple, si escriviu l'ordre següent "scp hpc1XYZ@boada.ac.upc.edu:lab1/pi/pi seq.c". a la vostra màquina local estareu copiant el fitxer font pi seq.c situat al directori lab1/pi del vostre directori d'inici a boada al directori actual, representat amb el ".", a la màquina local, amb el mateix nom.

1.1 Arquitectura de nodes i memòria

El primer que fareu és investigar l'arquitectura dels nodes disponibles a boada. Executeu l'ordre sbatch submit-arch.sh al directori lab1/arch. Aquesta ordre posarà en cua l'script submit-arch.sh.
Més detalls sobre les execucions en cua a continuació. Aquest script executarà les ordres lscpu i lstopo per tal d'obtenir informació sobre el maquinari en un dels nodes de la cua d'execució (boada-11 a 14). L'execució d'aquestes dues ordres mitjançant l'script submit-arch.sh genera tres fitxers, on el nombre pot ser 11, 12, 13 o 14: 1) lscpu-boada-number, 2) lstopo-boada-number i map-boada- nombre.fig.
Podeu utilitzar l'ordre xfig per visualitzar el fitxer de sortida generat (map-boada-number.fig) i exportar-lo a un format diferent (PDF o JPG, per exemple) mitjançant Fitxer ÿ Exporta per incloure'l al vostre lliurament per a aquest laboratori. tasca 4 . Aquests fitxers us ajudaran a esbrinar:

- el nombre de sòcols, nuclis per sòcol i fils per nucli en un node específic;
- la quantitat de memòria principal en un node específic i cada NUMAnode;
- la jerarquia de memòria cau (L1, L2 i L3), privada o compartida per a cada nucli/sòcket.

Suggeriment: observeu quin és el nombre de fil(s) per nucli que retorna l'ordre lscpu. Si el número és 2, això significa que el maquinari de cada nucli manté el context de dos fils d'execució i pot recollir instruccions de qualsevol dels dos fils, encara que no simultàniament. Això s'anomena Hyper-Threading per a processadors Intel. Tot i que lscpu informa que té 40 CPU (nuclis), en realitat només n'hi ha 20. Fins i tot quan Hyper-Threading està habilitat i el sistema pot prendre instruccions ràpidament de 40 fils, només hi ha 20 nuclis disponibles!

Ompliu la taula de la secció "Entregables" amb les característiques principals del node. Dibuixa l'arquitectura del node a partir de la informació generada per les eines anteriors. l'opció "--of fig map.fig" per a lstopo pot ser molt útil per a aquest propòsit. llavors podeu utilitzar l'ordre xfig per visualitzar el fitxer de sortida generat (map.fig) i exportar a un format diferent (pdf o jpg, per exemple) mitjançant fitxer ÿ export[5] .

[4] A la distribució de Linux boada podeu utilitzar xpdf per obrir fitxers pdf i mostrar-los per visualitzar fitxers gràfics. També podeu utilitzar l'ordre "fig2dev -L pdf map.fig map.pdf" per convertir de .fig a .pdf; cerqueu llenguatges gràfics de sortida alternatius escrivint "man fig2dev".

[5] a la distribució boada linux podeu utilitzar xpdf per visualitzar fitxers pdf i mostrar-los per a altres formats gràfics. També podeu utilitzar l'ordre "fig2dev -l pdf map.fig map.pdf" per convertir de .fig a .pdf; cerqueu llenguatges gràfics de sortida alternatius escrivint "man fig2dev".

1.2 Modes d'execució: interactiu vs en cua

Hi ha dues maneres d'executar els vostres programes a boada:

1. mitjançant un sistema de cues (en un dels nodes boada-11 a boada-14);
2. de manera interactiva (en qualsevol dels nodes d'inici de sessió boada-6 a boada-8). En aquest cas, el sistema limita nombre de nuclis que s'utilitzaran en execucions paral·leles a dos.

És obligatori utilitzar l'opció 1 quan voleu executar scripts que requereixen diversos processadors en un node, assegurant-vos que el vostre treball s'executa de manera aïllada (i, per tant, informant de resultats de rendiment fiables) i per evitar afegir càrrega addicional al node interactiu al qual accedeixen tots. usuaris; l'execució comença tan bon punt hi ha un node disponible. Quan utilitzeu l'opció 2, la vostra execució s'inicia immediatament, però compartirà recursos amb altres programes i treballs interactius, sense garantir resultats representatius de temps. Normalment, es proporcionaran scripts per a ambdues opcions (submit-xxxx.sh i run-xxxx.sh, respectivament):

- Posar en cua un treball per a l'execució: "sbatch [-p partition] ./submit-xxxx.sh" . Es poden especificar paràmetres addicionals, si ho necessita l'script, després del nom de l'script. Si no especifiqueu el nom de la partició amb "-p partition", el vostre script s'executarà a la partició d'execució per defecte. Utilitzeu "cua" per preguntar al sistema sobre l'estat de la presentació de la vostra feina. Podeu utilitzar "escancel·la" seguit de l'identificador de la feina per eliminar una feina del sistema de cua. Tingueu en compte que els noms de partició associats a cada nom de node es mostren a l'última columna de la taula anterior.
  Després de l'execució en un node disponible associat a la partició especificada, a més dels fitxers generats per l'script, es crearan dos fitxers addicionals. El seu nom tindrà el nom de l'script seguit d'un ".e" i un ".o" i l'identificador de la feina. Contindran els missatges enviats a l'error estàndard i la sortida estàndard, respectivament, durant l'execució del treball. Hauríeu de comprovar-los per assegurar-vos que els resultats tinguin sentit.

- Execució interactiva: ./run-xxxx.sh. Es poden especificar paràmetres addicionals després del nom de l'script. Els treballs executats de manera interactiva tenen un límit de temps curt per executar-se.

1.3 Compilació i execució en sèrie

A continuació, us familiaritzareu amb els passos de compilació i execució tant per a aplicacions seqüencials com paral·leles. Per a això, utilitzarem alguns codis que calculen els dígits del número Pi. Com a referència, la figura 1.1 mostra un codi python que calcula l'àrea sota un quadrant del cercle unitari, és a dir, un cercle de radi igual a 1, definit per la corba y = ÿ 1 ÿ x 2.

El temps d'execució d'aquest codi a boada-1 va trigar 522,1 segons a calcular l'estimació 3,14159266367805 (el valor real és 3,141592653589793). Com podeu veure, va trigar molt de temps mentre només calculava 7 dígits correctes de Pi. Pretenem estimar els dígits de Pi molt més ràpidament. Per a això farem servir alguns codis C que paral·lelitzem. Fareu servir un codi molt senzill, pi seq.c, que podeu trobar dins del directori lab1/OpenMP/pi. pi seq.c realitza el càlcul del nombre Pi calculant la integral de l'equació de la figura 1.2. Aquesta vegada l'algorisme utilitza la derivada de la funció arctangent de x, que és igual a 1 dividit per (1 + x Pi. De nou, l'equació es pot resoldre calculant l'àrea definida per la funció, que al seu torn pot ser s'aproxima dividint l'àrea en petits rectangles i sumant-ne l'àrea. Aquesta és una de les moltes maneres d'estimar ^{2}) que és una altra manera d'estimar els dígits del nombre Pi. Els dos algorismes anteriors es multipliquen per 4 perquè les funcions calculen ÿ/4. La figura 1.3 mostra una versió simplificada del nombre Pi. codi que teniu a pi seq.c. La variable num steps defineix el nombre de rectangles, l'àrea dels quals es calcula en cada iteració del bucle i.

La figura 1.4 mostra el flux de compilació i execució d'un programa seqüencial. Sempre compilarem programes per generar fitxers executables binaris mitjançant un Makefile, amb múltiples objectius que especifiquen les regles per compilar cada versió del programa. En aquest curs farem servir gcc (el front-end C de la col·lecció de compiladors GNU) o icc (el front-end C de la col·lecció Intel Compilers)6 . Podeu escriure "gcc -v" o "icc -v" per saber quina versió específica del compilador esteu utilitzant. I definiu la variable CC al Makefile per utilitzar-ne una o una altra.

6 Tingueu en compte que, però, icc no es troba per defecte i només estarà disponible si la variable d'entorn PATH està definida correctament. Quan sigui necessari, l'afegirem al nostre fitxer de configuració de l'entorn. Només llavors estarà disponible un cop s'hagi fet "source ÿ/environment.bash", ja sigui manualment o automàticament a l'inici de sessió si s'especifica al fitxer ÿ/.bashrc.

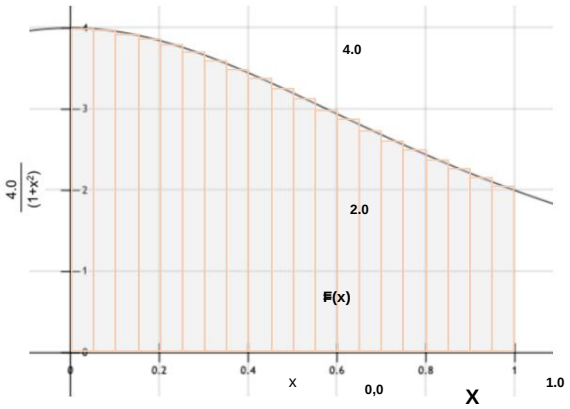
```
def midpoint_riemann_sum():
    N = 1000000000

    delta_x = (1 - 0) / N
    x = 0
    pi = 0
    inici = temps.hora()

    mentre que x < (1-delta_x):
        x_next = x + delta_x
        x_mid = (x_següent + x) / 2
        f_x = math.sqrt (1 - math.pow (x_mid, 2))
        pi += f_x * delta_x
        x += delta_x

    pi = 4 * pi
    final = temps.hora()
    imprimir (final - començament)
    print("Estimació: " + str(pi))
    print("Actual: " + str(math.pi))
```

Figura 1.1: Codi Python per calcular Pi



Matemàticament, sabem que:

$$\int_0^1 \frac{4.0}{(1+x2)}\,dx = \pi$$

Podem aproximar la integral com una suma de rectangles:

$$\sum_{i=0}^{N} F(x_i)\Delta x \approx \pi$$

On cada rectangle té una amplada Δx i una alçada $F(x_i)$ al centre de l'interval i.

Figura 1.2: Una manera de calcular Pi

```
num_passos llargs estàtics = 100000;
void principal () {
    doble x, pi, pas, suma = 0,0;

    pas = 1,0/(doble) nombre_passos;
    for (long int i=0; i<nombre_passos; ++i) {
            x = (i+0,5)*pas;
            suma += 4,0/(1,0+x*x);
    }
    pi = pas * suma;
}
```

Figura 1.3: Codi de sèrie per a Pi

En els passos següents compilareu pi seq.c utilitzant el Makefile i l'executareu de manera interactiva i mitjançant el sistema de cues, amb les ordres de cronometratge adequades per mesurar el seu temps d'execució:

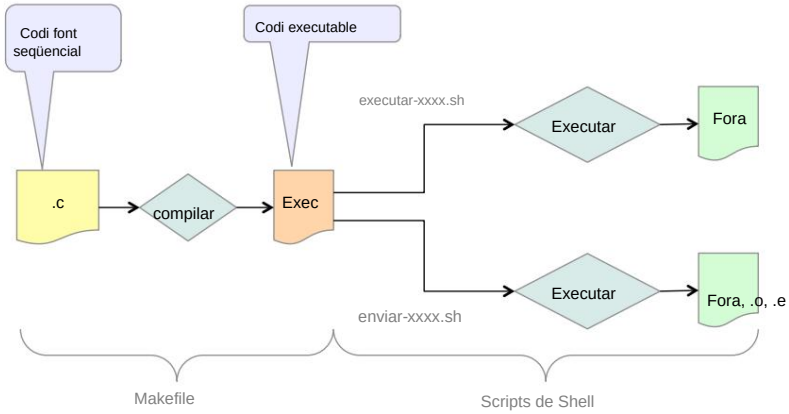


Figura 1.4: Flux de compilació i execució del programa seqüencial.

1. Obriu el fitxer Makefile, identifiqueu l'objectiu que heu d'utilitzar per compilar el codi seqüencial. Observeu com s'invoca el compilador. Executeu la línia d'ordres "make target Identified" per generar el fitxer executable binari. Alternativament, només podeu escriure "make" que ho compilarà tot
   programes.

2. Executeu interactivament el binari generat per calcular el número pi seqüencialment fent 1.000.000.000 d'iteracions mitjançant l'script run-seq.sh ("run-seq.sh pi seq 1000000000") que retorna el temps de CPU de l'usuari i del sistema, el temps transcorregut , també anomenat temps de paret, i el % de CPU utilitzada (utilitzant el programa d'hora GNU a /usr/bin/time). A més, el mateix programa també informa del temps d'execució transcorregut mitjançant gettimeofday. Mireu el codi font i identifiqueu les invocacions de funcions i les estructures de dades necessàries per mesurar el temps d'execució. Tingueu en compte que el temps de CPU és la quantitat de temps de processador que triga el procés. Això no indica la durada. El temps transcorregut representa la durada total de l'execució. Si aprofitem el paral·lelisme, el temps transcorregut pot ser inferior al temps total de la CPU perquè s'utilitzen diversos nuclis per fer el treball i es pot fer més ràpid.

3. Envieu l'execució al sistema de cua mitjançant l'ordre "sbatch submit-seq.sh" i utilitzeu "queue" per veure que el vostre script s'està executant. Mireu l'script submit-seq.sh i els resultats generats (la sortida estàndard i l'error de l'script i el fitxer time-pi seq-boada{11-14}).

1.4 Compilació i execució de programes OpenMP

En aquesta secció utilitzarem OpenMP, l'estàndard de programació paral·lela amb memòria compartida, per expressar el paral·lelisme en el llenguatge de programació C. Tingueu en compte que OpenMP s'explicarà detalladament a la segona part del curs, i l'objectiu ara és tenir un primer contacte i observar que podem accelerar l'execució dels nostres programes. Per tant, en aquest apartat veurem com compilar i executar programes paral·lels a OpenMP i observarem les millores de rendiment. La figura 1.5 mostra el flux de compilació i execució d'un programa OpenMP. La principal diferència amb el flux que es mostra a la figura 1.4 és que ara el Makefile inclourà la marca de compilació adequada per habilitar l'OpenMP.

1.4.1 Compilació de programes OpenMP

1. Al mateix directori lab1/OpenMP/pi trobareu una versió OpenMP del codi per fer el càlcul de pi en paral·lel (pi omp.c). Compileu el codi OpenMP utilitzant l'objectiu adequat al Makefile). Què et diu el compilador? El compilador emet un avís o un missatge d'error? El compilador està generant un fitxer executable? La compilació funciona sense problemes perquè hem utilitzat el senyalador -fopenmp, però si elimineu el senyalador -fopenmp i forceu la recompilació veureu la diferència.

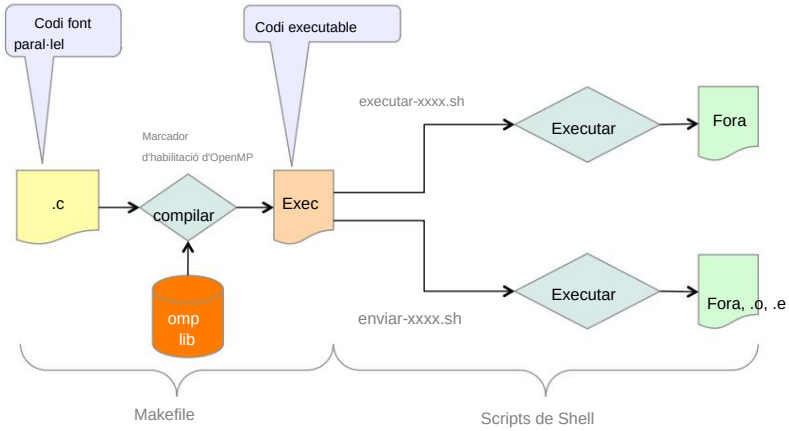


Figura 1.5: Flux de compilació i execució per a OpenMP.

1.4.2 Execució de programes OpenMP

1. Executeu interactivament el codi OpenMP amb 8 fils (processadors) i el mateix nombre d'iteracions (1.000.000.000) mitjançant l'script run-omp.sh ("run-omp.sh pi omp 1000000000 8"). Què us indica l'ordre time sobre el temps de CPU de l'usuari i del sistema, el temps transcorregut i el % de CPU utilitzat? Fes una ullada a l'script per descobrir com especifiquem el nombre de fils a utilitzar a OpenMP. Tingueu en compte que, com que qualsevol node d'inici de sessió boada-[678] és compartit per molts usuaris i està reservat per a ús interactiu, s'ha configurat de manera que els nostres processos només tinguin 2 nuclis disponibles per a la seva execució. Així, no veurem cap millora a causa de la paral·lelització.

2. Utilitzeu l'script submit-omp.sh per posar en cua l'execució del codi OpenMP ("sbatch submit-omp.sh") i mesura el temps de la CPU, el temps transcorregut i el % de CPU quan s'executa el programa OpenMP utilitzant 8 fils aïllats. Veieu una diferència important entre l'execució interactiva i la cua?

1.4.3 Escalabilitat forta versus feble

Finalment, en aquesta secció explorareu l'escalabilitat del codi pi omp.c en variar el nombre de fils utilitzats per executar el codi paral·lel. Per avaluar l'escalabilitat es calcularà la relació entre els temps d'execució seqüencial i paral·lel. Es consideraran dos escenaris diferents: escalabilitat forta i feble.

- En una escalabilitat forta, el nombre de fils es canvia amb una mida de problema fixa. En aquest cas, el paral·lelisme s'utilitza per reduir el temps d'execució del vostre programa.
- En escalabilitat feble, la mida del problema és proporcional al nombre de fils. En aquest cas, el paral·lelisme s'utilitza per augmentar la mida del problema per al qual s'executa el programa.

Us proporcionem dos scripts, submit-strong-omp.sh i submit-weak-omp.sh, que s'han d'enviar al sistema de cua amb l'ordre sbatch. Els scripts executen el codi paral·lel utilitzant des d'1 (np NMIN) fins a 20 (np NMAX). La mida del problema per a una escalabilitat forta és d'1.000.000.000 d'iteracions; per a una escalabilitat feble, la mida inicial del problema és de 100.000.000 que creix proporcionalment amb el nombre de fils. L'execució trigarà un temps perquè es fan diverses execucions per a cada prova (per tal d'aconseguir un temps mínim), si us plau, tingueu paciència! I, molt important, NO executeu-los de manera interactiva sinó que envieu-los per a l'execució a la cua d'execució. Com a resultat, cada script genera diversos fitxers, inclòs un gràfic (un fitxer amb un nom que acaba en .ps en format Postscript) que mostra el temps d'execució paral·lel resultant i l'eficiència d'acceleració o paral·lel. Visualitzeu[7] les trames generades i raoneu com canvia l'acceleració amb el nombre de fils en els dos escenaris.

Opcional: canvieu el valor de np NMAX a submit-strong-omp.sh de 20 a 40 i torneu a executar-lo. Pots explicar el comportament observat en el diagrama d'escalabilitat? Fins i tot quan Hyper-Threading està habilitat i el sistema pot rebre instruccions ràpidament de 40 fils, només hi ha 20 nuclis disponibles!

7 A la distribució de Linux boaada podeu utilitzar l'ordre ghostscript gs per visualitzar fitxers Postscript o convertir el fitxer fitxers PDF mitjançant l'ordre ps2pdf i utilitzeu xpdf per visualitzar fitxers PDF.

1.5 Programació, compilació i execució de programes MPI

1.5.1 Aprenentatge MPI

En aquesta secció utilitzarem MPI, l'estàndard de programació paral·lela amb sistemes de memòria distribuïda, per expressar el paral·lelisme en el llenguatge de programació C. Contràriament a l'apartat anterior, en aquesta part del curs hem d'entendre com desenvolupar un codi mitjançant MPI. Introduïm MPI als vídeos disponibles a Aula-ESCI. Si us plau, mireu-los primer i feu els qüestionaris associats. A continuació, trobareu un conjunt d'exemples al directori lab1/MPI/Tutorial. Utilitzeu-los per provar diferents funcionalitats que ofereix MPI. Venen de https://www.jics.utk.edu/mpi-tutorial de la Universitat de Tennessee.

Hi podeu trobar explicacions. A la majoria de subdirectoris hi ha un fitxer amb el nom acabat en .c.soln amb la solució perquè pugueu comprovar la vostra solució amb una bona. Podeu ignorar qualsevol cosa relacionada amb PBS, que és un altre sistema de cua diferent de SLURM8 que s'utilitza a Boada.

Tingueu en compte que l'execució d'un codi paral·lel pot canviar d'execució a execució. En conseqüència, la sortida que s'imprimeix pot reflectir de manera diferent d'una execució a una altra i pot superposar-se missatges escrits des de diferents processos MPI. De vegades és necessari executar diverses vegades per entendre el comportament. En resum, si us plau, consulteu els vídeos sobre MPI i intenteu entendre el codi i el comportament dels programes del Tutorial, almenys per als programes següents (i el nombre de processos MPI entre parèntesis): hello1 (4), hello2 (4), pical (4), pingpong (2), bloqueig (2), bloqueigf (2), col·lectius (4). Recordeu que heu de compilar el programa escrivint make dins del directori corresponent; i executeu-los amb mpirun -np P MPIEXECFILE, on P representa el nombre de processos MPI que volem crear, i MPIEXECFILE s'ha de substituir pel nom del fitxer executable MPI generat després de la compilació. A més, és important utilitzar el manual en línia per entendre les trucades MPI. Per exemple, quan estudieu el fitxer deadlock.c, us pot interessar llegir sobre algun nou primitiu MPI (man MPI Ssend). A més, inspeccioneu el document sobre els modes d'enviament de MPI a Aula-ESCI. A l'informe:

- Expliqueu el problema de bloqueig i com es pot solucionar;
- Mostrar el fragment de codi relacionat amb les operacions col·lectives de recollida i dispersió. A més, mostra la sortida de la seva execució, comentant-ho.

1.5.2 Forta escalabilitat amb processos i fils

A continuació, provarem com compilar i executar programes paral·lels en MPI, o una combinació de MPI i OpenMP. Per fer-ho, seguiu els mateixos passos que per a OpenMP a la secció anterior, però treballant amb els fitxers del directori lab1/MPI/pi. El codi calcula el nombre Pi amb un algorisme diferent basat en l'anomenat mètode de Monte Carlo. Els mètodes de Monte Carlo són una àmplia classe d'algorismes computacionals que es basen en mostres aleatòries repetides per obtenir resultats numèrics. La idea és simular punts aleatoris (x, y) en un pla 2-D amb el domini com un quadrat de costat 1 unitat. Imagineu un cercle dins del mateix domini amb el mateix diàmetre i inscrit al quadrat. Aleshores calculem la relació entre el nombre de punts que es trobaven dins del cercle i el nombre total de punts generats. Com més gran sigui el nombre de mostres, millor serà la precisió obtinguda.

Podem obtenir un gràfic del temps d'execució i de l'acceleració que s'obté en variar el nombre de processos MPI (escalabilitat forta) enviant un treball a la cua d'execució amb sbatch submit-strong-mpi.sh). A més, volem observar la combinació de MPI i OpenMP. Per a això, executeu sbatch submit-mpi2-omp.sh) i estudieu els resultats presentats al fitxer de sortida creat un cop finalitzada l'execució. Tingueu en compte que com que la prova utilitza 2 processos MPI, l'execució implica utilitzar un nombre parell de fils entre 2 i 40 fins i tot quan la sortida presenta els resultats d'1 a 20 fils: hi ha entre 1 i 20 fils per procés MPI; amb 2 processos això fa que hi hagi el doble de fils.

Esperem que, amb aquests experiments, us adoneu que podem accelerar el codi paral·lelitzant un codi amb OpenMP, amb MPI o una combinació d'ambdós per utilitzar diversos nodes i diversos nuclis per node!

8Veureu SLURM a la 2a part d'aquest curs.

1.5.3 Escriu el teu propi codi MPI

Finalment, comproveu la vostra comprensió creant un programa MPI propi que inclogui trucades a diverses primitives MPI. Aquí tens l'oportunitat de ser creatiu! No és necessari que el codi sigui realment útil, però el codi hauria d'incloure alguns intercanvis de missatges amb 1) primitives punt a punt i 2) almenys una primitiva de comunicació col·lectiva.

Per a aquest últim, observeu que tots els processos han d'anomenar un primitiu col·lectiu de la mateixa manera al programa; i la manera com podem especificar quin dels processos s'ha de comportar de manera diferent és a través d'un dels paràmetres. Sovint, aquest procés s'anomena procés mestre o arrel a la literatura.

Al vídeo d'introducció i en diversos Makefiles dins del subarbre MPI de directoris proporcionat a lab1 podeu veure com compilar programes C ampliats amb MPI amb mpicc. Però, si preferiu codificar el vostre programa MPI amb C++, podeu compilar-lo amb mpic++. Si preferiu Fortran, podeu utilitzar mpif77, mpiF90 o mpifort depenent del sabor de Fortran.

Per exemple, si el meu programa MPI és hello2.c, podem compilar amb "mpicc -o hello2 hello2.c".
Com que el codi que produireu és probable que s'executi en molt poc temps, podeu executar-lo de manera interactiva amb mpirun. Per exemple, podem executar-lo amb 4 processos amb "mpirun -np 4 hello2".

Nota: si el vostre codi implica molt de temps de CPU, heu de crear un script d'intèrpret d'ordres i llançar-lo a l'execució al sistema de cua amb l'ordre sbatch. Si aquest fos el cas, podeu inspirar-vos en els múltiples fitxers submit-xxxx.sh de lab1. Tanmateix, aquest NO és l'objectiu i no esperem que aquí desenvolupi un codi complex ni un que impliqui un cost computacional elevat.

Si us plau, proveu el vostre codi executant-lo amb diversos processos. Al vostre informe, haureu de mostrar el codi i explicar breument la vostra implementació i la seva funcionalitat, aclarint què ha de fer el vostre programa i com compilar-lo i executar-lo, possiblement complementat amb la sortida que produeix.

Entregable

Després de l'última sessió d'aquest treball de laboratori, i abans de començar la següent, hauràs de lliurar dos fitxers:

1. Un informe en format PDF (no s'acceptaran altres formats) amb les respostes a les preguntes indicat al final d'aquest document;
2. Per a la pregunta 5, també heu de lliurar el fitxer font del vostre codi C ampliat amb MPI.

El vostre professor obrirà el treball al lloc web de moodle aula ESCI i establirà el que correspongui dates de lliurament del lliurament. Enviament individual a través del web de moodle aula ESCI.

Important: a la portada del document, si us plau, indiqueu clarament el vostre nom i cognoms, l'identificador del grup (nom d'usuari hpc1XYZ), títol del treball, data, curs acadèmic/semestre, ... i qualsevol altra informació que considereu necessària. .

Com a part del document, podeu incloure qualsevol fragment de codi, figura o trama que necessiteu per donar suport a les vostres explicacions. En cas que necessiteu transferir fitxers de Boada a la vostra màquina local (ordinador portàtil o escriptori a la sala de laboratori), o viceversa, podeu utilitzar l'ordre de còpia segura scp. Per exemple, "scp hpc1XYZ@boada.ac.upc.edu:lab1/foobar.txt local/directory/". executat localment a la vostra màquina per copiar el fitxer foo.txt des del directori lab1 del vostre directori inicial de boada (el sistema remot) al directori local/directory/ de la vostra màquina local amb el mateix nom.

Arquitectura de nodes i memòria

1. Completa la taula següent amb les característiques arquitectòniques rellevants dels diferents nodes tipus disponibles a boada:

	boada-11 a boada-14
Nombre de sòcols per node	
Nombre de nuclis per sòcol	
Nombre de fils per nucli	
Freqüència màxima del nucli	
Mida de la memòria cau L1-I (per nucli)	
Mida de la memòria cau L1-D (per nucli)	
Mida de la memòria cau L2 (per nucli)	
Mida de la memòria cau d'últim nivell (per sòcol)	
Mida de la memòria principal (per sòcol)	
Mida de la memòria principal (per node)	

2. Incloeu en el document l'esquema arquitectònic d'un dels nodes boada-11 a boada-14.

Temporització d'execucions seqüencials i paral·leles

Per a cadascun dels apartats següents mostreu les trames o taules requerides. Però, a més, si us plau, raoneu els resultats obtinguts. Per exemple, comenta si l'escalabilitat és bona o no i per què.

3. Traceu el temps d'execució i l'acceleració que s'obté en variar el nombre de fils (forta escalabilitat) enviant els treballs a la cua d'execució (secció 1.4.3). Si feu la part opcional, mostreu la trama resultant i comenteu el motiu d'aquest comportament. Mostra l'eficiència paral·lela obtinguda en executar la prova d'escala feble. Expliqueu a què es refereixen l'escalabilitat forta i feble, exemplificant la vostra explicació amb les trames que presenteu.

4. Grafiqueu el temps d'execució i l'acceleració que s'obté en variar el nombre de processos MPI d'1 a 20 (forta escalabilitat) enviant els treballs a la cua d'execució (secció 1.5.2).
   A més, mostra en una taula el temps d'execució transcorregut quan s'executa amb 2 processos MPI en variar el nombre de fils d'1 a 20 [9]. Quina era l'hora seqüencial original?; el temps amb 20 processos MPI?; i el temps amb 2 processos MPI cadascun amb 20 fils OpenMP?
   Podeu recuperar aquesta informació per als processos 1 i 20 MPI del fitxer elapsed.txt; i per a 2 processos MPI cadascun amb 20 fils dins del fitxer de sortida creat després de l'execució de sbatch submit-mpi2-omp.sh.

MPI

Comprensió dels codis MPI

- Expliqueu el problema de bloqueig i com es pot solucionar;
- Mostra l'extracte de codi relacionat amb les operacions col·lectives MPI Gather i MPI Scatter. A més, mostreu la sortida de la seva execució en col·lectius de programes (només la sortida d'aquestes dues operacions, no tota la sortida del programa), explicant-ho breument.

Escriu el teu propi codi MPI

5. Creeu un programa MPI propi que inclogui trucades a diverses primitives MPI. Aquí tens l'oportunitat de ser creatiu! El codi hauria d'incloure alguns intercanvis de missatges amb 1) primitives punt a punt i 2) almenys una primitiva de comunicació col·lectiva. Prova-ho amb diversos processos. Al vostre informe, haureu de mostrar el codi i explicar breument la vostra implementació i la seva funcionalitat.

9Tingueu en compte que per a 2 processos MPI la prova d'escalabilitat implica utilitzar un nombre parell de fils entre 2 i 40.