UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Computació Paral·lela (CPA)

Pràctica 3: Paralel·lització amb MPI

Curso 2017/18

Índex

Introducció

Aquesta pràctica consta de 3 sessions. La següent taula mostra el material de partida per a realitzar cadascun dels apartats:

Sessió 1	Càlcul de Pi	`mpi_pi.c`
Sessió 2	Fractals	`newton.c`
Sessió 3	Producte matriu-vector	`mxv1.c`, `mxv2.c`

1 Primers passos amb MPI

L'objectiu de la primera sessió de la pràctica és familiaritzar-se amb la compilació i execució de programes MPI senzills.

1.1 Hola món

Comencem amb el típic programa "hola món" en el qual cada procés imprimeix un missatge per l'eixida estàndard. El codi de la Figura 1 mostra un programa mínim, amb inicialització i finalització de MPI, i un `printf` mostrant l'identificador del procés i el nombre de processos.

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

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

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  printf("Hello world from process %d of %d\n", rank, size);
  MPI_Finalize();
  return 0;
}
```

Figura 1: Programa "hola món" en MPI.

Compila i executa el programa. Per a compilar, cal usar el comando `mpicc` de forma similar a un compilador normal (`mpicc` és una utilitat que simplement invoca el compilador amb les opcions apropiades per a la instal·lació particular de MPI – amb "`mpicc -show`" es mostren aquestes opcions).

```
$ mpicc -o hello hello.c
```

Per a executar cal utilitzar `mpiexec`. En el nostre cas hem de llançar un treball a través del sistema de cues del clúster de pràctiques. Per a açò, el més còmode és escriure un *script* que continga les opcions del sistema de cues seguides pels comandos que es volen executar. En la Figura 2 hi ha un exemple en el qual es reserven 4 nodes amb un temps màxim de 10 minuts, en la cua `cpa` i amb l'opció d'executar en el directori actual (`.`). S'executa el programa `hello` a través de `mpiexec` (en altres sistemes seria necessari indicar com a argument de `mpiexec` el nombre de processos que volem, però el clúster està configurat perquè prenga aquest valor de l'opció `-l nodes` de PBS). La línia `cat $PBS_NODEFILE` no es realmente necesaria, serveix per a que sapigam què nodes del clúster han sigut reservats per a la nostra execució.

```
#!/bin/sh
#PBS -l nodes=4,walltime=00:10:00
#PBS -q cpa
#PBS -d .

cat $PBS_NODEFILE
mpiexec ./hello
```

Figura 2: *Script* per a executar en el sistema de cues.

Modifica el programa perquè, a més de l'identificador de procés, mostre també el nom del node en què està executant-se realment aquest procés. Per a açò, cal utilitzar `MPI_Get_processor_name`. Comprova que el sistema de cues col·loca un procés en cadascun dels nodes reservats.

Si volguérem executar tots els processos en el mateix node, caldria modificar el *script*. Concretament hem de reservar 1 node, indicant el nombre de processos per node (`ppn`), i afegir una opció especial:

```
#PBS -l nodes=1:ppn=4,walltime=00:10:00
#PBS -W x="NACCESSPOLICY:SINGLEJOB"
```

1.2 Càlcul de Pi

Anem ara a treballar amb un programa MPI que realitza un càlcul en paral·lel. En particular, anem a calcular una aproximació del valor de π. El valor de π pot calcular-se de moltes formes, una d'elles és mitjançant la integral definida

$$\int_0^1 \frac{1}{1+x^2} dx = \frac{\pi}{4} .$$

El programa `mpi_pi.c` calcula aquesta integral mitjançant el mètode dels rectangles que vam veure en la primera pràctica. L'interval $[0, 1]$ es descompon en n subintervals (rectangles) i cadascun dels p processos realitza el càlcul associat a n/p rectangles. Aquest càlcul és perfectament paral·lelitzable (independent) i únicament falta acumular les sumes parcials en el resultat final. En el programa donat, la suma global es calcula amb una operació de comunicació col·lectiva: `MPI_Reduce`.

Realitza diverses execucions del programa, per a diferents valors de n i amb diferent nombre de processos. Modifica el programa perquè la reducció es realitze amb comunicació punt a punt (`MPI_Send` i `MPI_Recv` en lloc de `MPI_Reduce`). Es pot implementar una solució simple, per exemple que tots els processos envien el resultat al procés 0, el qual s'encarregarà de calcular la suma global.

1.3 El programa *ping-pong*

Els programes MPI utilitzen la xarxa Infiniband disponible en el clúster de pràctiques. Aquesta xarxa té prestacions molt millors que una xarxa convencional (Ethernet), tant en ample de banda com en temps d'establiment. Encara que es poden consultar les especificacions tècniques de la xarxa, és habitual realitzar un estudi experimental per a obtenir els paràmetres de la xarxa a partir de l'execució d'un programa real. Un programa *ping-pong* consta de dos processos P_0 i P_1 en el qual P_0 envia un missatge a P_1, el qual li retorna el missatge immediatament després. P_0 mesura el temps transcorregut des de l'operació d'enviament fins que acaba la recepció de la resposta.

Implementa un programa *ping-pong* amb les següents característiques:

- Les operacions d'enviament i recepció es realitzen amb les primitives estàndard: `MPI_Send` i `MPI_Recv`.
- El temps es mesura amb la funció `MPI_Wtime`.
- Perquè els temps mesurats siguen significatius, el programa ha de repetir l'operació un cert nombre de vegades (per exemple 100 vegades) i mostrarà el temps mitjà.
- El programa tindrà com a argument la grandària del missatge, n (en bytes), i el nombre de repeticions.

Utilitza aquest programa per a obtenir un model del temps de comunicació $t_c = t_s + t_w n$. Per a açò, cal executar-lo per a diferents grandàries de missatge, per exemple, entre 0 i 1,000,000 amb increments de 100,000.

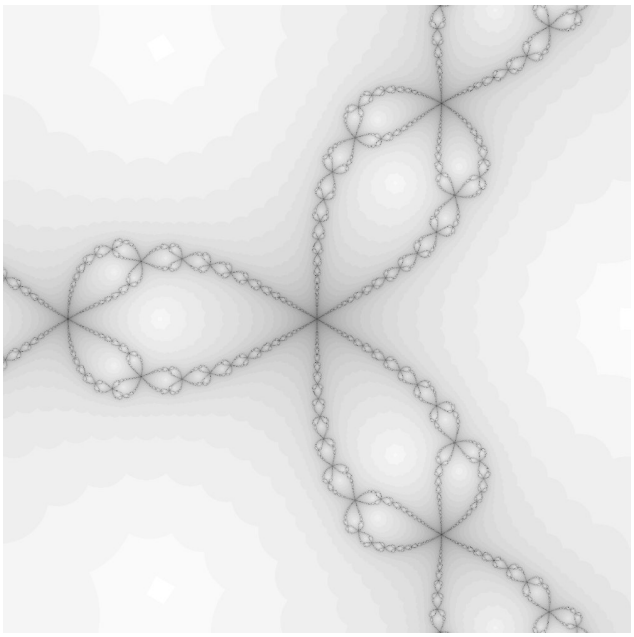
Els resultats mostren el comportament de la xarxa Infiniband. Si volguérem veure les prestacions de la xarxa Ethernet, podríem forçar l'execució del programa MPI perquè no use la Infiniband, afegint la següent opció al `mpiexec`: `--mca btl ^openib` (Nota: aquesta opció és específica de Open-MPI i no està disponible en altres implementacions de MPI).

Repeteix l'estudi executant el programa amb els dos processos dins del mateix node. D'aquesta forma es pot veure el comportament de MPI realitzant el pas de missatges directament a través de memòria compartida.

2 Fractals de Newton

Un fractal és un objecte geomètric l'estructura bàsica del qual es repeteix a diferents escales. En determinats casos, la representació d'un fractal comporta un cost computacional considerable. En aquesta pràctica es va a treballar amb un programa que genera fractals de Newton.

Es proporciona el codi font d'un programa paral·lel que utilitza un esquema mestre-treballadors i la llibreria de comunicacions MPI per a calcular diferents fractals de Newton: `newton.c`. L'objectiu de la



```
Per a y = y1, ..., y2
  Per a x = x1, ..., x2
    col = num_iteracions_Newton(funció,x,y)
    Pintar el punt (x,y) amb el color col
  Fi_Per
Fi_Per
```

Figura 3: Fractal de Newton de la funció $f(z) = z^3 - 1$ (esquerra) i algorisme utilitzat per a la seua generació (dreta). Nota: La imatge del fractal es mostra invertida (es veu el seu negatiu).

pràctica és modificar les comunicacions d'aquest programa, passant a utilitzar comunicacions no bloquejants, per a aconseguir que el procés mestre també processe part de la imatge.

2.1 Algorisme seqüencial

En els fractals de Newton es treballa cercant els zeros de funcions complexes de variable complexa. Aquests zeros es calculen utilitzant el mètode de Newton de càlcul d'arrels, el qual realitza més o menys iteracions depenent del valor inicial. Donada una funció complexa, se selecciona una zona del plànol complex en la qual dibuixar el fractal i se cerca algun zero de la funció partint de tots els punts en eixa zona. El nombre d'iteracions necessàries per a aconseguir la solució des de cada punt dictamina el color amb el qual es pintarà eixe punt en el fractal.

Per exemple, en la Figura 3 (esquerra) pot veure's el fractal de Newton per a la funció $f(z) = z^3 - 1$ en la zona amb x i y entre -1 i 1. Un algorisme en pseudo-codi per al dibuix d'un fractal de Newton es mostra en la Figura 3 (dreta).

2.2 Algorisme mestre-treballadors clàssic

El programa proporcionat (`newton.c`) és una implementació paral·lela seguint l'esquema mestre-treballadors per a generar fractals de Newton.

Aquest programa permet calcular fractals de Newton de diferents funcions. Té múltiples opcions que afecten al fractal generat (poden veure's directament en el codi font). Cal destacar l'opció `-c` a la qual se li proporciona el nombre de funció a utilitzar per al fractal. Actualment hi ha 4 funcions, identificades d'1 a 4, encara que el paràmetre permet indicar també el cas 5. El cas 5 en realitat és una ampliació d'una zona blanca del cas 4. No és "molt bonic", però té un cost major i és més útil per a veure el guany en paral·lel. S'aconsella usar els casos 1-4 per a comprovar (veient la imatge generada) que el programa paral·lel funciona bé i després el cas 5 per a fer mesura de prestacions.

El programa emmagatzema el fractal calculat en cada ocasió en el fitxer `newton.pgm` (si no es canvia usant l'opció `-o`). Per defecte genera una imatge en escala de grisos on el color blanc es correspon amb el major

```
Si jo=0 aleshores (soc el mestre)

  següent_fila <- 0
  Per a proc = 1 ... np
    envia al procés proc petició de fer la fila número següent_fila
    següent_fila <- següent_fila + 1
  fi_per

  files_fetes <- 0
  Mentres files_fetes < files_totals
    rep de qualsevol procés una fila calculada
    proc <- procés que ha enviat el missatge
    num_fila <- número de fila
    envia al procés proc petició de fer la fila número següent_fila
    següent_fila <- següent_fila + 1
    copia fila calculada al seu destí, que es la fila num_fila de la imatge
    files_fetes <- files_fetes + 1
  fin_mentres

si_no (soc un treballador)

  rep número de fila a fer en num_fila
  Mentres num_fila < files_totals
    processa la fila número num_fila
    envia la fila calculada al mestre
    rep número de fila a fer en num_fila
  fi_mentres

fi_si
```

Figura 4: Pseudo-codi del mestre-treballadors.

nombre d'iteracions necessitat en un punt de la imatge. No obstant açò, com a curiositat, el programa permet "acolorir" els fractals generats. Per a açò, ha d'usar-se l'opció `-p` seguida d'un nombre positiu (increment a usar entre les components de colors consecutius de la paleta) o negatiu (s'usa de llavor per a generar una paleta de colors aleatoris). En aquest cas el fitxer d'eixida tindrà l'extensió `.ppm` corresponent a una imatge en color.

Exercici 1: Compila i executa el programa observant els 5 fractals bàsics que pot generar (hauràs d'executar-ho almenys 5 vegades: amb les opcions `-c1`, `-c2`, `-c3`, `-c4` i `-c5`). Pots generar algun fractal amb colors "psicodèlics". Prova amb l'opció `-p50` o amb `-p-11`.

En el programa `newton.c` és la funció `fractal_newton` l'encarregada de realitzar l'algorisme de fractals de Newton proporcionat anteriorment (Figura 3).

Aquesta funció dibuixa en la matriu A de $w \times h$ píxels un fractal de Newton en l'àrea amb coordenades x, y en el rectangle amb extrems (x_1, y_1) i (x_2, y_2). Se cerquen zeros de la funció amb una tolerància donada per `tol` i amb un nombre màxim d'iteracions donat per `maxiter`. A més, es calcula i es retorna el nombre màxim d'iteracions que apareix en la imatge, que s'utilitzarà per al color blanc.

El codi proporcionat ja implementa una versió paral·lela de l'algorisme, seguint l'esquema tradicional mestre-treballadors. El pseudo-codi corresponent es mostra en la Figura 4. Recorda que en l'esquema clàssic mestre-treballadors un dels processos actua com a mestre i va encarregant treball als altres, els treballadors, que retornaran al mestre els resultats de cada treball realitzat.

És convenient que estudies i entengues el codi bàsic mestre-treballadors que s'ha utilitzat en el codi proporcionat.

```
següent_fila <- 0
Per a proc = 1 ... np
  envia al procés proc petició de fer la fila número següent_fila
  següent_fila <- següent_fila + 1
fi_per

files_fetes <- 0
Mentres files_fetes < files_totals
  inicia la recepció no bloquejant d'una fila calculada de qualsevol procés
  Mentres no s'ha rebut res i següent_fila < files_totals
    processa la fila número següent_fila
    següent_fila <- següent_fila + 1
    files_fetes <- files_fetes + 1
  fi_mentres
  Si no s'ha rebut res aleshores
    espera (de forma bloquejant) a rebre un missatge
  fi_si
  proc <- procés que ha enviat el missatge
  num_fila <- número de fila
  envia al procés proc petició de fer la fila número següent_fila
  següent_fila <- següent_fila + 1
  copia fila calculada al seu destí, que es la fila num_fila de la imatge
  files_fetes <- files_fetes + 1
fi_mentres
```

Figura 5: Pseudo-codi del mestre fent que també treballe (el codi dels teballadors no canvia).

Exercici 2: Repassa l'algorisme utilitzat (Figura 4) i la seua implementació en llenguatge C realitzada en la funció `fractal_newton` del programa proporcionat.

2.3 Algorisme mestre-treballadors amb el mestre treballant

En l'algorisme mestre-treballadors clàssic, el mestre va manant treballs que fer als treballadors i arreplegant els resultats, però ell no realitza cap treball. Si executem reservant un processador per a aquest mestre, estarem desaprofitant la potència de càlcul d'aquest processador.

Una forma de no desaprofitar aquesta potència extra és fent que el procés mestre també realitze treballs. Per a açò, cal fer que les seues comunicacions siguen no bloquejants. D'aquesta manera, en lloc de quedar-se bloquejat esperant la resposta d'algun treballador, podrà anar treballant al mateix temps que espera aquesta resposta.

En la Figura 5 tens un algorisme en pseudo-codi per a què el mestre es comporte d'aquesta manera. El codi dels treballadors no canvia.

Exercici 3: Llig i entén l'algorisme mostrat en la Figura 5 i implementa'l sobre una còpia del programa original, per a després poder comparar tots dos programes. Utilitza un dels fractals originals per a comprovar que la nova versió és correcta (ix igual el dibuix). Després, utilitza algun fractal més costós (el cas 5, per exemple) per a traure mesura de prestacions comparant els resultats de tots dos programes. Per a altres casos costosos pots passar-li aquestes opcions al programa:

```
-c4 -r0.01 -x0.1 -y0.2
-c4 -r0.5 -x0.4 -y0.4
-c4 -r0.5 -x-0.12 -y0.4
```

Fes taules i gràfiques de temps, speed-ups, eficiències... per a ambdós programes.

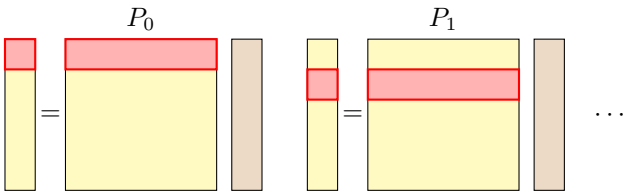


Figura 6: Esquema de repartiment de la matriu entre els processos en el cas de distribució per blocs de files.

3 Producte matriu-vector

En aquesta sessió anem a aprendre a utilitzar les operacions de comunicació col·lectiva de MPI. Per a açò, utilitzarem un nucli computacional molt utilitzat: el producte d'una matriu per un vector.

3.1 Descripció del problema

Molts problemes en computació numèrica permeten la seua resolució mitjançant el que es coneix com a mètodes iteratius. En aquests mètodes, es parteix d'una aproximació a la solució i mitjançant alguna operació que es repeteix durant múltiples iteracions es van obtenint successives aproximacions de la solució, que sota unes determinades condicions van acostant-se cada vegada més a la solució cercada.

Anem a treballar amb un mètode iteratiu per a la resolució de *sistemes d'equacions lineals*, és a dir, donats una matriu quadrada $A \in \mathbb{R}^{n \times n}$ i un vector $b \in \mathbb{R}^n$, es vol calcular un altre vector $x \in \mathbb{R}^n$ que complisca

$$Ax = b.$$

En el cas de resolució de sistemes d'equacions lineals, l'expressió habitual utilitzada en el procés iteratiu és

$$x^{k+1} = Mx^k + v,$$

on x^k representa la solució aproximada en la iteració k, i M i v són una matriu i un vector obtinguts a partir de A i b, respectivament. En aquest cas, es va calculant cada nova aproximació mitjançant un producte matriu-vector i una suma vectorial, a partir de l'anterior aproximació.

L'habitual és fer tantes iteracions com siguen necessàries fins que es complisca un determinat criteri de parada, que garantisca que l'aproximació obtinguda estiga suficientment prop de la solució. No obstant açò, per simplicitat, en el nostre cas anem a realitzar un nombre fix d'iteracions.

Partim d'un codi ja paral·lelitzat que realitza tot el procés fins a obtenir el vector x final aproximació de la solució. En aquest codi treballem amb una matriu M i un vector v aleatoris, però construïts de tal forma que el sistema convergisca (vaja acostant-se a la solució). Al final de tot, el programa mostra la 1-norma del vector x resultat. Aquest valor es pot utilitzar a manera de *hash* per a comprovar que diferents execucions són correctes.

Disposem de dues versions que difereixen en la forma d'emmagatzemar i repartir les dades entre els diferents processadors. (El resultat de les dues versions no és comparable entre si, perquè cadascuna treballa amb un problema diferent.)

- En la primera versió (`mxv1.c`) emmagatzemem la matriu M per files i la repartim per blocs de files entre els processadors (veure Figura 6).
- En la segona versió (`mxv2.c`) la matriu s'emmagatzema per columnes i es reparteix per blocs de columnes entre els processadors.

Aquestes formes diferents de repartir la matriu fan que les comunicacions per a efectuar els càlculs siguen diferents en cadascun dels casos. Es recomana a l'alumne que analitze amb calma ambdues versions per a entendre com es realitzen en tots dos casos el producte matriu-vector i la suma de vectors.

3.2 Tasca a realitzar

Les versions proporcionades de l'algorisme paral·lel per a resoldre un sistema d'equacions mitjançant un mètode iteratiu han sigut desenvolupades usant solament operacions de comunicació punt a punt. No obstant açò, com l'alumne ja sap, en MPI hi ha moltes funcions de comunicació col·lectiva que faciliten la programació quan es requereixen aquest tipus de comunicacions.

L'alumne ha de substituir totes les comunicacions que són susceptibles de ser realitzades mitjançant operacions col·lectives per les corresponents crides a les funcions de MPI de comunicació col·lectiva. En el codi estan marcades aquestes operacions amb un comentari previ que indica `COMUNICACIONS`.

En general, açò és el que hauria d'haver-se fet al principi. Normalment, és convenient utilitzar sempre que es puga les operacions col·lectives de MPI, ja que estaran optimitzades per a realitzar les comunicacions de forma eficient.

Una vegada que es tinguen ambdues versions amb i sense operacions col·lectives, és interessant comparar els temps de cadascuna d'elles. Encara que és probable que no es troben moltes diferències en aquest problema en concret.

Com sempre, s'aconsella a l'alumne que no espere a tenir totes les comunicacions canviades per a provar el programa. És convenient anar provant el programa després de canviar cada comunicació i així assegurar-se que no s'ha alterat el resultat amb el canvi.