ikd baliabideak 13 (2017)

Programazio Paraleloa: MPI

Ikaslearen kaiera

Olatz Arbelaitz Gallego

Olatz Arregi Uriarte

Agustin Arruabarrena Frutos

José Ignacio Martín Aramburu

Javier Muguerza Rivero

AURKIBIDEA

1. PROIEKTUAREN TESTUINGURUA

1.1. SARRERA

Proiektuak Konputazio Paraleloko Sistemak (Informatika Ingeniaritzako Gradua, 3. maila, Konputagailuen Ingeniaritza) ikasgaiaren bigarren zatiari dagokio: Programazio Paraleloa, MPI. Proiektuak ikasgaiaren % 60a betetzen du, hau da, 3,6 ECTS kreditu; guztira, 36 ordu presentzial eta beste 54 ordu ez-presentzial. Hiru ikasleko taldeak osatuko direnez, lan-karga osoa 270 ordukoa da taldeko.

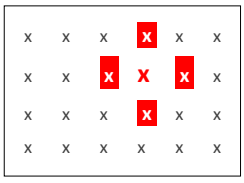
1.2. GALDERA ERAGILEA

Adi, erre egingo baitzara!

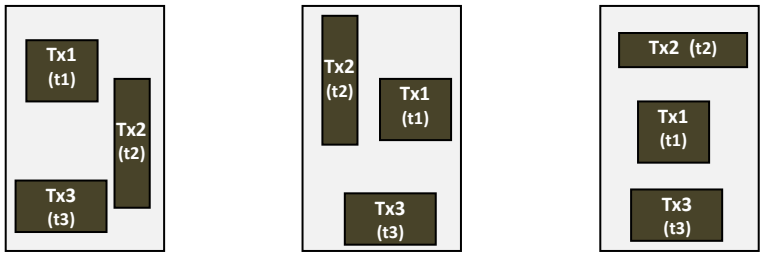
1.3. JOKALEKUA

TXIPSA enpresak zirkuitu inprimatuko txartelak fabrikatzen ditu, zeinetan hainbat txip jarri behar baitira. Txipek bero asko sortzen dute erabiltzen direnean, eta posizio egokienetan kokatu behar dira txarteletan sistema osoaren tenperatura globala ahalik eta baxuena izan dadin, sistemak funtziona dezan plaka erre gabe.

Hori dela eta, zirkuitua fabrikatu baino lehen, beroaren difusioa simulatzen duen algoritmo bat exekutatzen da, txipetarako txartelean posizio desberdinak aztertu eta egokiena aukeratzeko: txartelaren batez besteko tenperatura minimizatzen duena, alegia. Bero-difusioaren algoritmoak hainbat puntutan banatzen du txartela, 2 dimentsioko saretxo bat osatuz, eta sareko puntu bakoitzaren tenperatura kalkulatzen du **auzokoen tenperaturaren arabera**. Eragiketa errepikatzen da harik eta sistema batez besteko tenperatura jakin batera konbergitzen den arte; tenperatura hori bero-iturrien, txipen, posizioen arabera da.



Simulazioa prozesadore batean, seriean, egin da orain arte, hainbat konfigurazio analizatuz egokiena aukeratzeko. Irudian, zirkuitu inprimatuko txartel batean hiru txip kokatzeko hiru aukera ageri dira, zeinetan, beroaren difusioaren simulazioaren arabera batez besteko hiru tenperatura desberdin lortzen diren.



1. konf.: $T_{bb}1$ 2. konf.: $T_{bb}2$ 3. konf.: $T_{bb}3$

Duela gutxi, TXIPSAk 33 prozesadoreko **memoria banatuko** *cluster* bat erosi du, non prozesadoreak Gigabit Etherneten bidez konektatuta baitaude, eta, ondorioz, bero-difusioaren algoritmoa eta batez besteko tenperaturen kalkulua paralelizatu nahi ditu. **Prozesadore bakoitzak txartelaren zati baten portaera termikoa simulatuko du**, prozesadore guztien artean batez besteko tenperatura kalkulatzeko. Helburua garbia da: txartelen konfigurazio gehiago aztertu ahal izatea denbora gutxiagoan, fabrikazio-denbora laburtzeko, eta, agian, etekinak igotzeko.

Enpresak eskatu dizu txipen kokapenen araberako portaera termikoa paralelizatzea, ahalik eta modurik eraginkorrenean.

1.4. LANTZEN DIREN GAIAK ETA IKASTE-PROZESUAREN EMAITZAK

Proiektuarekin ikasgaiaren 3. gaia landuko da, Programazio paraleloa: MPI. Analizatuko ditugu memoria banatuko sistemetarako aplikazio paralelo eraginkorrak sortzeko gainditu behar diren arazoak: prozesuen arteko komunikazioa, bidali behar diren datuen definizio egokiena komunikazioaren kostua minimizatzeko, eta lan-banaketarako hainbat eredu (estatikoa, dinamikoa, *manager*/*worker*). Aplikazioa garatzeko, MPI erabiliko dugu, tresna estandarra *cluster*-etarako aplikazioak sortzeko.

Bukaeran, gauza izan behar zara honakoak egiteko:

- Aplikazio ez oso konplexuak sortzea, MPI erabiliz, memoria banatuko konputagailueta-rako.
- Analizatzea eta ulertzea aplikazio paraleloak exekutatzean lortzen den eraginkortasuna.

Horiez gain, beste gaitasun orokor eta zeharkakoak landuko dira (ikus Informatika Ingeniaritzako Graduko web-orria, “Ikasketa-plana” atalean).

1.5. ENTREGATZEKOAK

E1 Taldea osatzeko agiria eta taldean aritzeko hartutako konpromisoen dokumentua (E1.1); proiektua aurrera eramateko egin diren bileren aktak (E1.2).

E2 Jokalekua aztertzeko egindako posterra.

E3 Puzzlean, taldeko kideak egindako lana: teoria lantzeko egindakoa eta ariketen analisia/ebazpena biltzen dituen txostena (E3.1); puzzlearen aurkezpena (E3.2). Gehituko da taldekide bakoitzak bere atala ikasteko erabili duen material berria (irakasleak emandakoaz aparte).

E4 Berdinen arteko ebaluazioen agiriak. Batetik, puzzlearen aurkezpenaren ebaluazio-agiria (E4.1), eta, bestetik, aplikazioaren aurkezpenarena (E4.2).

E5 Proiektuan zehar gutxieneko ezagutzak lortu direla egiaztatzeko azterketa.

E6 Aplikazioaren aurreneko fasearen bukaera: lortu diren emaitzen txosten laburra (orri bat) (E6.1). Garatutako aplikazioaren azken txosten teknikoa (E6.2), eta bera aurkezteko erabilitako materialak (E6.3).

E7 Proiektua garatzeko sortutako dokumentazioa biltzen duen karpeta. Azkeneko bertsioa proiektua bukatutakoan entregatu behar bada ere, lehenengo bertsio bat entregatu behar da puzzlea bukatu eta gero; bertsio hori errebisatuko da, eta hobetzeko aukera izango da.

Entregatu behar diren dokumentu guztiak taldeari dagozkio, E5 azterketa izan ezik, pertsonala baita.

1.6. EBALUAZIO-SISTEMA

Proiektua (ikasgaiaren bigarren atala) 6 puntutan ebaluatuko da, irizpide hauen arabera:

- Aurkezpen indibiduala. Bi aurkezpen egingo dira: puzzlearena eta aplikazioarena. Talde bakoitzak horietako bat egingo du. Aurkezpena nork egingo duen unean bertan aukeratuko da. Jarduera kooperatiboa denez, lortutako nota talde osoarena izango da. Aurkezpenak irakaslearen eta ikasleen arteen ebaluatuko dira, eta **puntu bat** balioko dute.
- Gutxieneko ezagutzak egiaztatzeko azterketa indibiduala: **2 puntu**. Proiektua gainditzeko, gutxienez % 30a lortu behar da azterketan.
- Garatutako aplikazioa eta lortu den errendimendua azaltzen duen txosten teknikoa: **3 puntu**.
- Proiektuaren karpeta. Kalifikazioa GAI edo EZ GAI izango da. Derrigorrez gainditu behar da, baina ez du eraginik izango proiektuaren kalifikazioan.

Ebaluazioan, eskatutakoaz harantzago doazen jarduerak edo bereziki ondo egindako lanak kontuan hartuko dira gehigarrizko puntuak emateko.

Ikasgaia gainditzeko, proiektua gainditu behar da, hau da gutxienez 3 puntu lortu behar dira. Taula honetan laburbiltzen dira jarduerak, eta haien ebaluazioa.

	Nota	**Irakaslea**		**Ikasleak**	
Banakakoa	2 puntu	Azterketa (E5)	2 p.		
Taldekoa	4 puntu	Aurkezpenak (E3.2/E6.3) Txosten teknikoa (E6.2) Dokumentu-karpeta (E7)	0,5 p. 3 p. -	Aurkezpenak (E4.1/E4.2)	0,5 p.
Guztira	**6 puntu**		**5,5 puntu**		**0,5 puntu**

1.7. LANAREN PLANIFIKAZIOA

Hurrengo taulan, proiektuan zehar garatu behar den lana laburbiltzen da: jarduerak, aurreikusitako lan-karga (eskoletan eta kanpoan), sortu behar den dokumentazioa, eta ebaluazio-une nagusiak, astez aste.

Eskolako jarduerak 1,5 orduko hiru saiotan garatuko dira, astez aste, astelehenetan, astearteetan eta asteazkenetan. Martxoaren 21-23 astean (aste santua) eta maiatzaren 16-20 astean (lanak entregatzeko) ez dago eskolarik, baina lanegunak dira.

Ordutegi trinkoko astean (apirilak 18-22) aurreikusita dago bisita bat DIPCren kalkulu-zentrora (konfirmatzeko dago).

IKASLEEN LANAREN PLANIFIKAZIOA

Astea	Eskola	Aurrez aurreko jarduerak	t/egun	Eskolatik kanpo (astez aste)	t/aste	Entregatzekoak	Ebaluazioa
02/29 -4	1	Proiektuaren jokalekua. Taldekako hausnarketa	40 m			Agiria: taldea osatzea (E1.1)	
		Eztabaida, azken posterra	30 m			Posterra (din-A4) (E2)	
		Proiektuaren planifikazioa	20 m				
	2	MPI: oinarrizko kontzeptuak	90 m		3 h		
	3	MPI: oinarrizko kontzeptuak	60 m	MPIren oinarrizko kontzeptuak ikasi			
		Puzzlearen aurkezpena eta lan-banaketa	30 m	Lan indibiduala, puzzlean esleitutakoa ikasteko			
03/7-11	4	Lan indibiduala, puzzlean esleitutakoa ikasteko	90 m	Lan indibiduala, puzzlean esleitutakoa ikasteko	4 h		
	5		90 m				
	6		90 m				
03/14-18	7	Taldearen bilera, ezagutza partekatzeko	90 m	Puzzlea osatzen duten kontzeptu guztiak ikastea	4 h		
	8	Adituen bilera, ezagutza partekatzeko	90 m	Puzzlearen txostena eta aurkezpena prestatzea			
	9	Taldearen bilera, ezagutza partekatzeko	80 m	Adierazpen grafikoari buruzko ariketak lantzea			
		Ariketa berezia: adierazpen grafikoa	10 m				
03/21-23				Puzzlearen txostena eta aurkezpena prestatzea Adierazpen grafikoari buruzko ariketak lantzea	7 h		
04/4-8	10	Ariketa berezia: adierazpen grafikoa	60 m	Puzzlearen aurkezpena prestatzea	4 h		
		Taldearen bilera, ezagutza partekatzeko	30 m				
	11	Puzzlea: aurkezpena	60 m			Puzzlea: txostena (E3.1), aurk-	[1 p. (E3.2, E4.1)]
			30 m			ezpena (E3.2), ebal.-agiria (E4.1)	
	12	Puzzlea: zalantzak argitzeko debatea	60 m	Aplikazioa garatzea (1. fasea)		Karpeta (berrikuspena) (E7)	
		Aplikazioaren enuntziatua (1. fasea)	30 m				
04/11-15	13	Talde-lana: aplikazioa garatzea	90 m	Aplikazioa garatzea	4 h		
	14		90 m				
	15		90 m				
04/18-22				Aplikazioa garatzea	6 h		
04/25-29	16	Talde-lana: aplikazioa garatzea	90 m	Aplikazioa garatzea	4 h		
	17	Teoria: puntutik punturako kom.; *deadlock*. Teoria: Jumpshot	90 m			Aplikazioren 1. faseko lehenengo emaitzak (E6.1)	
	18	Lehen emaitzei buruzko debatea	50 m	Aplikazioa garatzea (2. fasea)			
		Aplikazioaren enuntziatua (2. fasea)	30 m				
		Teoria: topologiak/eraztuna. Ariketa: MPI+OpenMP	10 m				
05/2-6	19	Talde-lana: aplikazioa garatzea (2. fasea)	90 m	Aplikazioa garatzea	4 h		
	20		90 m				
	21		90 m				
05/9-13	22	Talde-lana: aplikazioa garatzea, dokumentazioa	90 m	Aplikazioa garatzea / dokumentazioa	4 h		
	23		90 m				
	24		90 m				
05/16-20				Aurkezpena prestatu / Gai nagusiak ikasi	7 h	Txosten tekn., 1-2 faseak (E6.2)	3 p.
24/05	25	Egindako lanaren defentsa	90 m		3 h	Agiria: aurkezpen-ebal. (E4.2)	[1 p.(E6.3, E4.2)]
		Gutxieneko ezagutzen azterketa	90 m			Azterketa (E5) - Karpeta (E7)	2 p. (E.5)

2. BALIABIDEAK

Atal honetan, proiektua garatu ahal izateko ikasleei emandako baliabideak bildu ditugu: laborategirako materialak, puzzlearen enuntziatuak, ebatzi beharreko ariketak, eta garatu beharko duten aplikazioaren deskribapena.

2.1. *CLUSTER*-A ETA MPI-REN OINARRIZKO KONTZEPTUAK LANTZEKO LABORATEGIKO MATERIALAK

Puzzlearekin hasi baino lehen, *cluster*-a nola erabili eta MPIren oinarrizko kontzeptuak azalduko ditugu laborategiko bi saiotan.

2.1.1. *Cluster*-a abian jartzeko prozedura

32+1 nodoko *cluster* sinple batekin lan egingo dugu. Nodoen ezaugarriak hauek dira —Intel Core 2 6320 - 1,8 GHz - 2 GB RAM - 4 kB cache— eta Gigabit Ethernet-aren bidez daude konektatuta. 32 nodo kalkulurako erabiliko ditugu, eta bestea garapenerako. Prestazio handiko sistema ez izan arren, aukera ematen digu mezu-ematearen bidezko aplikazio paraleloak exekutatzeko, haien portaera analizatzeko, eta abar.

Garapenerako nodoa fitxategien zerbitzaria da, eta *cluster*-era sartzeko puntua. Kanpo-helbidea **`g002615.gi.ehu.eus`** da, eta bere helbidea *cluster* barnean `servidor01` (`nodo00`). Gainerako nodoak —`nodo01`, `nodo02`... `nodo32`— bakarrik sarrerako nodotik atzi daitezke (ezin da kanpotik sartu). Laborategian *cluster*-arekin konektatzeko, honako komando hau erabiliko dugu Linuxen

```
> ssh kontua@g002615.gi.ehu.eus
```

Erabiliko ditugun adibideak eta programak `templates` direktorioan dituzue, hiru karpetatan: `adibideak`, `puzzlea`, `aplikazioa`. Kopiatu programak zure laneko direktoriora karpeta berri batean; esaterako

```
> mkdir adibideak
> cp templates/adibideak/* adibideak
```

▪ Exekuzio-ingurunea sortu

Makina asko erabiliz aplikazioak exekutatzeko, `ssh`-ren bidez sartuko gara makinetan, baina pasahitza erabili behar izan gabe. Horretarako beharrezkoa da makina bakoitzean gutxienez behin sartzea, eta horrela erabiltzaile baimendu gisa hartuko gaitu. Honako komando hauek exekutatu behar dira,

1. Sarrerako direktorioan:

```
> ssh-keygen -t rsa
```
(`return` sakatu hiru aldiz)

.ssh direktorioa sortzen da, non id.rsa eta id_rsa.pub fitxategiak ageri baitira; pasa direktorio horretara eta exekutatu:

```
> cp id_rsa.pub authorized_keys
> chmod go-rw authorized_keys
```

Berriro sarrerako direktoriora pasa, eta *cluster*-eko makinetan sartu eta atera, banan-banan:

```
> ssh nodoxx            (xx = 01 ... 32)
      (yes)
> exit
```

Eragiketa horiek **cluster-ssh.sh** komando-fitxategian bildu ditugu, denak automatikoki exekutatu ahal izateko.

▪ MPICH2

MPIren MPICH2 bertsioa erabiliko dugu (banaketa librekoa da, eta zure PCan instala dezakezu; beste inplementazio ezagun bat OpenMPI da). Programa paraleloak exekutatu aurretik, honako hau egin behar da:

2. Laneko direktorioan .mpd.conf izeneko fitxategia sortu, honako eduki honekin:

   secretword=xxxx (xxxx = edozein hitz)

   eta baimenak aldatu: > chmod 600 .mpd.conf

3. Erabiliko ditugun makinen izenak dituen fitxategia sortu (izen lehenetsia, mpd.hosts). Nahikoa da fitxategian idaztea:

   ```
   nodo00
   nodo01
   …
   nodo32
   ```

Aurreko kasuan bezala, bi eragiketa horiek **cluster-ssh.sh** komando-fitxategian sartuta daude.

4. MPICH2 ingurunea abiatu (makina bakoitzean exekutatu behar diren mpd *daemon*-ak):

   ```
   > mpdboot -v -n proz_kop [-f makinen_fitxategia]
   ```

   (-f baldin eta makinen zerrenda duen fitxategiaren izena mpd.hosts ez bada).

Beste komando batzuk:

> mpdtrace	"aktibatuta" dauden makinen zerrenda itzultzen ditu
> mpdlistjobs	exekuzioan dauden atazak zerrendatzen ditu
> mpdringtest 2	aktibatuta dauden makinak zeharkatu (2 aldiz) eta denbora itzultzen du
> mpd &	*daemon* bakar bat sortzen du, bertako nodoan
> mpdhelp	komandoei buruzko informazioa

Arazoren bat egon bada *daemon*-ak sortzen, mpdcleanup exekutatu eta prozesua errepikatu.

5. Dagoeneko, programak konpilatu eta exekutatu ditzakegu:

```
> mpicc [-Ox] -o pr1 pr1.c      [x=1-3 optimizazio-maila; lehenetsita, 2]
> mpiexec -n xx pr1             xx = prozesu-kopurua
```

   `pr1` programa exekutatzen da xx prozesadoretan (-1 *flag*-arekin, ez da `nodo00`a erabiltzen)

```
> mpiexec -n 1 -host nodo00 p1 : -n 1 -host nodo01 p2
> mpiexec -configfile prozesuak
```

   Bi programa, `p1` eta `p2`, exekutatzen dira, 00 eta 01 nodoetan, edo `prozesuak` fitxategian adierazten den moduan.

6. Eta lan-saioa bukatzeko:

```
> mpdallexit
```

(komando hauei buruzko informazio gehiago lortzeko: `komandoa --help`)

2.1.2. MPIren oinarrizko kontzeptuak: adibideak eta ariketak

Puzzlean egin behar den lanari ekin baino lehen, MPIrekin lan egiten hasteko behar diren oinarrizko ezagutza aztertuko dugu gardenkien eta laborategiko adibideen bidez.

- Definizioa. Sistemaren arkitektura eta mezu-ematea.
- Oinarrizko MPI funtzioak:

  Programaren hasiera eta bukaera: `MPI_Init`, `MPI_Finalize`

  Prozesuak identifikatzea: `MPI_Comm_rank`, `MPI_Comm_size`

  Mezuak bidali eta hartu, puntutik puntura: `MPI_Send`, `MPI_Recv`, `MPI_Probe`

Honako lau programa hauek —`kaixo.c`, `zirku.c`, `bidali.c` eta `probe.c`— laborategiko lehen saioetan erabili eta aztertuko ditugu.

```
/************************************************************
       kaixo.c
       MPI programa: prozesuak aktibatzea
************************************************************/

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

int main(int argc, char *argv[])
{
  int   izenl;
  char  pr_izena[MPI_MAX_PROCESSOR_NAME];
  int   pid, npr;                       // identifikadorea eta kopurua
  int   A = 21;

  MPI_Init (&argc, &argv);
  MPI_Comm_size (MPI_COMM_WORLD, &npr);
  MPI_Comm_rank (MPI_COMM_WORLD, &pid);

  MPI_Get_processor_name (pr_izena, &izenl);

  A = A + 1;
  printf (" >> P%2d proz. (%2d-(e)tik aktibatuta %s-(e)an, A = %d\n", pid, npr,pr_izena, A);

  MPI_Finalize ();
  return (0);
 }



/************************************************************************************
       zirku.c
       begizta baten paralelizazioa
************************************************************************************/

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

#define DECBIN(n,i) ((n&(1<<i))?1:0)

void test (int pid, int z)
{
  int v[16], i;

  for (i=0; i<16; i++) v[i] = DECBIN(z,i);

  if ((v[0] || v[1]) && (!v[1] || !v[3]) && (v[2] || v[3]) && (!v[3] || !v[4])
    && (v[4] || !v[5]) && (v[5] || !v[6]) && (v[5] || v[6]) && (v[6] || !v[15])
    && (v[7] || !v[8]) && (!v[7] || !v[13]) && (v[8] || v[9]) && (v[8] || !v[9])
    && (!v[9] || !v[10]) && (v[9] || v[11]) && (v[10] || v[11])
    && (v[12] || v[13]) && (v[13] || !v[14]) && (v[14] || v[15]))
  {
    printf(" %d) %d%d%d%d%d%d%d%d%d%d%d%d%d%d%d%d (%d)\n", pid, v[15],v[14],v[13],
        v[12],v[11],v[10],v[9],v[8],v[7],v[6],v[5],v[4],v[3],v[2],v[1],v[0], z);
    fflush(stdout);
  }
}


int main (int argc, char *argv[])
{
  int   i, pid, npr;

  MPI_Init (&argc, &argv);
  MPI_Comm_size (MPI_COMM_WORLD, &npr);
  MPI_Comm_rank (MPI_COMM_WORLD, &pid);

  for (i=pid; i<65536; i += npr)  test (pid, i);

  MPI_Finalize ();
  return (0);
}
```

```
/******************************************************************************
       bidali.c
       bektore bat bidaltzen da P0tik P1era
******************************************************************************/

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

#define N 10

int main (int argc, char **argv)
{
  int   pid, npr, jatorria, helburua, tag, ndat;
  int   VA[N], i;
  MPI_Status info;

  MPI_Init (&argc, &argv);
  MPI_Comm_rank (MPI_COMM_WORLD, &pid);
  MPI_Comm_size (MPI_COMM_WORLD, &npr);

  for (i=0; i<N; i++) VA[i] = 0;

  if (pid == 0)
  {
    for (i=0; i<N; i++) VA[i] = i;

    helburua = 1; tag = 0;
    MPI_Send (&VA[0], N, MPI_INT, helburua, tag, MPI_COMM_WORLD);
  }

  else if (pid == 1)
  {
    printf ("\n VA-ren balioa P1en, datuak jaso aurretik \n\n");
    for (i=0; i<N; i++) printf ("%4d", VA[i]);
    printf ("\n\n");

    jatorria = 0; tag = 0;
    MPI_Recv (&VA[0], N, MPI_INT, jatorria, tag, MPI_COMM_WORLD, &info);

    MPI_Get_count (&info, MPI_INT, &ndat);
    printf (" P1ek VA hartu du P%dtik: tag %d, datkop %d (osagai)\n\n",
            info.MPI_SOURCE, info.MPI_TAG, datkop);

    for (i=0; i<N; i++) printf ("%4d", VA[i]);
    printf ("\n\n");
  }

  MPI_Finalize ();
  return (0);
}
```

```
/*************************************************************
        probe.c
        probe funtzioaren erabilera
*************************************************************/

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


int main (int argc, char **argv)
{
  int   pid, npr, jatorria, helburua, tag;
  int   i, luzera, tam;
  int   *VA, *VB;
  MPI_Status info;

  MPI_Init (&argc, &argv);
  MPI_Comm_rank (MPI_COMM_WORLD, &pid);
  MPI_Comm_size (MPI_COMM_WORLD, &npr);

  if (pid == 0)
  {
    srand (time(NULL));
    luzera = rand () % 100;
    VA = (int *) malloc (luzera*sizeof(int));
    for (i=0; i<luzera; i++) VA[i] = i;

    printf ("\n VA-ren balioa P0n datuak bidali aurretik\n\n");
    for (i=0; i<luzera; i++) printf ("%4d", VA[i]);
    printf ("\n\n");

    helburua = 1; tag = 0;
    MPI_Send (&VA[0], luzera, MPI_INT, helburua, tag, MPI_COMM_WORLD);
    free (VA);
  }

  else if (pid == 1)
  {
    jatorria = 0;  tag = 0;
    MPI_Probe (jatorria, tag, MPI_COMM_WORLD, &info);
    MPI_Get_count (&info, MPI_INT, &tam);

    if (tam != MPI_UNDEFINED)
    {
      VB = (int *) malloc (tam*sizeof(int));
      MPI_Recv (&VB[0], tam, MPI_INT, jatorria, tag, MPI_COMM_WORLD, &info);
    }

    printf ("\n VB-ren balioa P1en, datuak jaso ondoren\n\n");
    for (i=0; i<tam; i++) printf ("%4d", VB[i]);
    printf ("\n\n");
    free (VB);
  }

  MPI_Finalize ();
  return (0);
}
```

▪ **Laborategiko ariketak** (hasiera-fasea)

Lehenengo saioetako osagarri gisa, bi ariketa hauek egin behar dituzu laborategian:

0.1. `bidali.c` programan, P0 prozesuak 10 osagaiko `VA` bektorea sortzen du, eta P1i bidaltzen dio. P1ek bektorea hartu eta inprimatzen du.

Aldatu programa honako hau egin dezan: bektorea hartu ondoren, P1ek batu egiten ditu bektorearen osagaiak eta emaitza P0ri itzultzen dio, berak inprima dezan.

0.2. `zirku.c` programak begizta baten iterazioak exekutatzen ditu paraleloan, zeinen bidez 16 aldagaiko funtzio logiko baten soluzioak bilatzen baitira. Iterazioen banaketa estatiko tartekatua da, eta, funtzioaren emaitza gisa, 1 ematen duten sarrera-aldagaien konbinazioak inprimatzen ditu.

Aldatu programa honako hau egin dezan: P0k inprimatu behar du zenbat soluzio aurkitu diren.

2.1.3. MPI (2): puntutik puntura mezuak bidaltzeko beste zenbait modu; elkarblokeoak.

Aplikazioaren 1. fasea bukatu eta lehen emaitzen analisia egin eta gero, puntutik punturako komunikaziorako MIPk eskaintzen dituen beste aukera batzuk aztertuko ditugu, hauek, hain zuzen ere:

- Komunikazio sinkronoa: `MPI_Ssend`. Komunikazio-blokeoen arazoa.
- Berehalako komunikazioa: kalkulua eta komunikazioa teilakatzea.

  `MPI_Isend`, `MPI_IRecv`, `MPI_Test`, `MPI_Wait`

Honako adibide hauek aztertuko ditugu laborategian: `dlock1.c`, `dlock2.c`, `dlock3.c`, `dlock3s.c`, `dlock4.c`, `dlock5.c`, `send-dead.c`, `ssend.c`, `isend.c`.

```
/******************************************************************************
       dlock1.c
      bi aldagairen trukaketa
******************************************************************************/

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

int main (int argc, char **argv)
{
  int   pid, iturburua, helburua, tag;
  int   A, B, C;
  MPI_Status  info;

  MPI_Init (&argc, &argv);
  MPI_Comm_rank (MPI_COMM_WORLD, &pid);

  if (pid == 0)
  {
    A = 5;

    printf ("\n    >> datuak jasotzen P1-etik   \n");
    iturburua = 1;  tag = 1;
    MPI_Recv (&B, 1, MPI_INT, iturburua, tag, MPI_COMM_WORLD, &info);

    printf ("\n    >> datuak bidaltzen P1-era  \n");
    helburua = 1; tag = 0;
    MPI_Send (&A, 1, MPI_INT, helburua, tag, MPI_COMM_WORLD);

    C = A + B;
    printf ("\n    C %d da 0 prozesuan \n\n", C);
  }

  else if (pid == 1)
  {
    B = 6;

    printf ("\n    >> datuak jasotzen P0-tik  \n");
    iturburua = 0;  tag = 1;
    MPI_Recv (&A, 1, MPI_INT, iturburua, tag, MPI_COMM_WORLD, &info);

    printf ("\n    >> datuak bidaltzen P0-ra  \n");
    helburua = 0; tag = 0;
    MPI_Send (&B, 1, MPI_INT, helburua, tag, MPI_COMM_WORLD);

    C = A + B;
    printf ("\n   C %d da 1 prozesuan \n\n", C);
  }

  MPI_Finalize ();
  return (0);
}
```

ARIKETAK: **`dlock`** Analizatu programa honen aldaerak, eta aztertu ondo edo gaizki dauden.

```
/*****************************************************************************
        send-dead.c
        send funtzioaren buffer-aren tamaina ikusteko
         programa blokeatzen da pakete handiago bat bidaltzerakoan
*****************************************************************************/

#include <stdio.h>
#include "mpi.h"
#define N 100000

int main (int argc, char** argv)
{
  int         pid, kont;                // Prozesuaren identifikadorea
  int         a[N], b[N], c[N], d[N];
  MPI_Status  status;

  MPI_Init (&argc, &argv);
  MPI_Comm_rank (MPI_COMM_WORLD, &pid);

  for (kont=100; kont<=N; kont=kont+100)
  {
    if (pid == 0)
    {
      MPI_Send (&a[0], kont, MPI_INT, 1, 0, MPI_COMM_WORLD);
      MPI_Recv (&b[0], kont, MPI_INT, 1, 0, MPI_COMM_WORLD, &status);
      printf ("igorlea %d \n", kont);
    }
    else
    {
      MPI_Send (&c[0], kont, MPI_INT, 0, 0, MPI_COMM_WORLD);
      MPI_Recv (&d[0], kont, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
      printf ("                   hartzailea %d \n", kont);
    }
  }

  MPI_Finalize ();
  return 0;
} /*  main  */
```

```
/*****************************************************************************
       ssend.c
      Ping-pong bi prozesadoreen artean. Ssend komunikazio sinkronoa
*****************************************************************************/

#include <stdio.h>
#include <mpi.h>
#include <math.h>
#include <unistd.h>

#define BIRAK 4

double kalkulu ()
{
  double aux;
  sleep (1);
  aux = rand () % 100;

  return (aux);
}


int main (int argc, char** argv)
{
  double      t0, t1, dat= 0.0, dat1, dat_rec;
  int         pid, i;
  MPI_Status  status;

  MPI_Init (&argc, &argv);
  MPI_Comm_rank (MPI_COMM_WORLD, &pid);

  if (pid == 0) t0 = MPI_Wtime ();

  for (i=0; i<BIRAK; i++)
  {
    if (pid == 0)
    {
      MPI_Ssend (&dat, 1, MPI_DOUBLE, 1, 0, MPI_COMM_WORLD);
      dat1 = kalkulu ();

      MPI_Recv (&dat_rec, 1, MPI_DOUBLE, 1, 0, MPI_COMM_WORLD, &status);
      dat = dat1 + dat_rec;
    }
    else
    {
      dat1 = kalkulu ();

      MPI_Recv (&dat_rec, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD, &status);
      dat = dat1 + dat_rec;
      MPI_Ssend (&dat, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
    }
  }

  if (pid == 0)
  {
    t1 = MPI_Wtime ();
    printf ("\n   Exekuzio-denborA = %f s \n", t1-t0);
    printf ("\n   Dat = %1.3f \n\n", dat);
  }

  MPI_Finalize ();
  return (0);
} /*  main  */
```

```
/**********************************************************************
        isend.c
        Ping-pong bi prozesadoreren artean.
        Isend berehalako komunikazioa
**********************************************************************/

  ...

  for (i=0; i<BIRAK; i++)
  {
    if (pid == 0)
    {
      MPI_Isend (&dat, 1, MPI_DOUBLE, 1, 0, MPI_COMM_WORLD, &request);

      dat1 = kalkulu ();

      MPI_Recv (&dat_rec, 1, MPI_DOUBLE, 1, 0, MPI_COMM_WORLD, &status);
      dat = dat1 + dat_rec;
      MPI_Wait (&request, &status);
    }
    else
    {
      dat1 = kalkulu ();

      MPI_Recv (&dat_rec, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD, &status);

      dat = dat1 + dat_rec;
      if (i != 0) MPI_Wait (&request, &status);
      MPI_Isend (&dat, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD,&request);
    }
  }

  ...
```

2.1.4. Jumpshot

`Jumpshot` MPICH-ekin banatzen den aplikazio grafiko bat da. MPI programak exekutatzerakoan sortutako aztarnen fitxategiak ("`log`" motakoak) grafikoki aztertzeko balio du, prozesuen arteko komunikazioa nola gauzatu den analizatzeko, eraginkortasun-arazoak detektatzeko eta abar. Exekuzioari buruzko informazio lortzeko, MPICH inguruneko MPE funtzioak gehitu behar zaizkie programei.

`log` fitxategia lortzeko, programaren leku estrategikoetan laginak hartzeko funtzioak sar ditzakegu (MPE funtzioak) edo, kasu sinpleetan, MPI funtzio guztietatik datuak hartu.

```
> mpicc -mpe=mpilog -o prog prog.c
> mpiexec –n xx prog
```

Konpilatu ondoren, programa exekutatu eta aztarna-fitxategi bat lortuko dugu, `clog2` motakoa. Ondoren, `jumpshot` exekutatuko dugu aztarnak analizatzeko:

```
> jumpshot
```

`Jumphot4`-k `slog2` motako aztarna-formatuarekin egiten du lan; beraz, aurretik formatu horretara bihurtu behar da fitxategia. Bihurketa aplikazioaren barnean egin daiteke, edo, bestela, honako hau exekutatuz:

```
> clog2TOslog2 prog.clog2
```

`Jumpshot` aplikazioaren hasierako leihoan, bistaratu nahi den fitxategia aukeratu daiteke. Leiho berri batean, exekuzioaren eskema grafiko bat ageriko da, non MPI funtzioak kolore ezberdinetan adierazten baitira. Puntutik punturako komunikazioetan, gezi batek lotzen du mezu bakoitzaren bidaltzea eta hartzea. Saguaren eskuineko botoiarekin, exekutatu diren funtzioei eta bidali diren mezuei buruzko informazioa lortzen da.

OHARRA: aplikazio grafikoa exekutatu ahal izateko:

-- Windows: >> exekutatu aurretik `X-Win32` (edo antzeko aplikazio bat)

-- Linux: >> makinan sartu `-X` adierazlea erabiliz: `ssh -X kontua@makina`

2.2. MPI: PROGRAMAZIO PARALELOARI BURUZKO PUZZLEA

Programa paralelo eraginkorrak nola diseinatu ikasteko, eta MPIri buruzko proiektuan garatu beharko duzun aplikazioaren diseinua bideratzeko, hiru ataletan banatu ditugu gai nagusiak, eta puzzle bat osatu haiekin.

Taldekide bakoitzak puzzlearen atal bat landu behar du, gainerako taldeen pareko kideekin bildu zalantzak eztabaidatu eta argitzeko, eta, azkenik, ikasitakoa partekatu taldekideekin. Puzzlearen atal bakoitzean, dagozkien gaiak ikasi, proposatutako programak egin, eta emaitzak egiaztatu behar dira.

Laborategian egin ditugun lehenengo saioetan, *cluster*-a nola erabili eta MPIren oinarrizko lehen funtzioak aztertu ditugu. Hortik abiatuta, hauek dira puzzlearen hiru atalak:

1. **Komunikazio kolektiboak**

2. **Datu-mota deribatuak eta komunikatzaileak**

3. **Lan-kargaren banaketa dinamikoa, eta mugen arazoa datuak banatzerakoan.**

2.2.1. Komunikazio kolektiboak

Komunikazioa atal garrantzitsuena da aplikazio paraleloak memoria banatuko sistemetan exekutatu ahal izateko. Dakizunez, prozesuen arteko komunikazioa mezu-ematearen bidez egiten da. Laborategiko hasierako saioetan, MPIk eskaintzen dizkigun puntutik punturako oinarrizko komunikazio-funtzioak aztertu ditugu (`MPI_Send` eta `MPI_Recv`), baina badago funtzio gehiago, komunikazio eraginkorragoa lortu eta programazio paraleloa sinplifikatzen dutenak: komunikazio kolektiboko funtzioak.

Jarduera honen bidez, komunikazio kolektiboa zertan datza ulertu beharko duzu, zein funtzio dauden horretarako MPIn, eta nola erabiltzen diren. Behean dituzu gaia lantzeko zenbait erreferentzia (askoz gehiago aurki dezakezu oso erraz), irakur dezazun eta zure taldekideentzat aurkezpen txiki bat presta dezazun.

Atal honetako kontzeptuak ondo ulertzeko, hurrengo hiru ariketak ebatzi behar dituzu, eta ebazpideak azaldu txosten labur batean (1 edo 2 orrialde gehienez). Programen oinarrizko kodeak `templates/puzzle` direktorioan dituzu, zure kontuan.

> Erreferentzia orokorrak

- www.mpi-forum.org/docs/docs.html
- computing.llnl.gov/tutorials/mpi/
- Pacheco P.: *Parallel Programming with MPI*. Morgan Kaufmann, 1997
- Gropp W., Lusk E., Skjellum A.: *Using MPI. Portable Parallel Programming with the Message-Passing Interface*. The MIT Press, 1999.

> Komunikazio kolektiboei buruzko erreferentziak

- Pacheco P.: *An Introduction to Parallel Programming*. Morgan Kaufmann, 2011. 3. kapitulua, 4. atala.
- Snir M., Otto S., Huss-Lederman S., Walker D., Dongarra J. *MPI: The Complete Reference, Volume 1, The MPI Core*. The MIT Press, 1999. 4. kapitulua

> Puzzlearen 1. atala: ariketak

P1.1. `N` osagaiko bektore bat banatu behar da `npr` prozesuren artean. Osatu `P11-distribute0.c` serieko programa, bektore-zati bakoitzaren tamaina eta hasieratik zati bakoitzaren hasierara dagoen distantziak kalkula ditzan, bi kasu hauetarako:

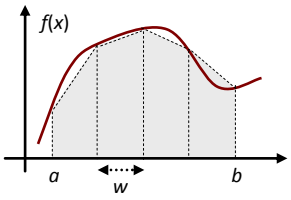
a. balizko hondarrak azken zatiari gehitzen zaizkio.

b. l balizko hondarrak banan-banan gehitzen dira zati desberdinetara.

Esaterako, exekutatzen baduzu programa datu hauekin —`N` = 17, `npr` = 5— honako hau inprimatu behar du:

```
Data to distribute: N and npr
17
5

FIRST DISTRIBUTION: the remainder for the last process
  3   0
  3   3
  3   6
  3   9
  5   12

SECOND DISTRIBUTION: the remainder distributed (+1) among the first
processes
  4   0
  4   4
  3   8
  3   11
  3   14
```

P1.2. `P12-inteser.c` programak integral baten balioa kalkulatzen du, oso ezaguna den metodoa erabiliz: funtzioaren kurbaren area estaltzen duten *n* trapezioren areak batuz. *n* hainbat eta handiagoa, hala izango da integralaren doitasuna ere.



Osatu `P12-intepar0.c` programa funtzio hori *P* prozesuren artean kalkulatzeko, komunikazio kolektiboko funtzioak erabiliz. Konparatu emaitza seriean lortzen duzunarekin. Esaterako, exekutatzen baduzu programa 4 prozesadoretan datu hauekin, emaitza honakoa hau izango da:

```
Introduce a, b (limits) and n (num. of trap.)
0
10
10000000

  Integral, from 0.0 to 10.0, 10000000 trap.) = 3.869022947101
  Execution time (4 proc.) = 47.474 ms
```

P1.3. Lau prozesutan exekutatu den aplikazio batean, P2k `B` bektorearen osagaiak (16 zenbaki oso) banatzen ditu, honela: P0ra: `B`[3], `B`[4], `B`[5]; P1era: `B`[7], `B`[8]; P2ra: `B`[10]; eta P3ra: `B`[12], `B`[13], `B`[14], `B`[15]. Banatu ondoren, prozesuek 100 gehitzen diote hartutako osagai bakoitzari; eta, azkenik, osagai guztiak `B` bektorean biltzen dira berriro P2n, dagozkien posizioetan.

Osatu `P13-scatter-gather0.c` programa eragiketa hori egin dezan. Hasieran, P2k hasieratu behar du bektorea `B[i] = i` balioetan, eta, bukaeran, `B` bektore berria inprimatu behar du. Honako hau inprimatu beharko luke:

```
B in pid=2 after the calculation
   0   1   2 103 104 105   6 107 108   9 110  11 112 113 114 115
```

```
/*********************************************************************
      P11-distribute0.c
      >>> TO DO: MODIFY AND COMPLETE <<<
*********************************************************************/

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

int main (int argc, char **argv)
{
  int    N, Nloc, npr, remainder, i;

  printf ("\n Data to distribute:  N and npr\n");
  scanf ("%d %d" , &N, &npr);

// FIRST DISTRIBUTION:  the remainder for the last process
//      >>> TO DO: CALCULATE SIZE AND SHIFTS FOR THE FIRST DISTRIBUTION <<<




  printf ("\n FIRST DISTRIBUTION: the remainder for the last process \n");
  for (i=0; i<npr; i++) printf ("\n  %d   %d", size[i], shift[i]);

// SECOND DISTRIBUTION: the remainder distributed (+1) among the first processes
//      >>> TO DO: CALCULATE SIZE AND SHIFTS FOR THE SECOND DISTRIBUTION <<<




  printf ("\n\n SECOND DISTRIBUTION: the remainder distributed (+1) among the first
              processes \n");
  for (i=0; i<npr; i++) printf ("\n  %d   %d", size[i], shift[i]);
  printf ("\n");
  return 0;
}
```

```
/*****************************************************************
        P12-inteser.c
        Integral of a function by sums of areas of trapezoids
*****************************************************************/

#include <stdio.h>
#include <sys/time.h>

struct timeval   t0, t1;
double texec;

void   Read_data (double* a_ptr, double* b_ptr, int* n_ptr);
double Integrate (double a, double b, int n, double w);
double f (double x);

int main (int argc, char** argv)
{
  double      a, b, w;
  int         n;
  double      resul;          // Result for the integral

  Read_data (&a, &b, &n);
  w = (b-a) / n;

// Integral calculation
  gettimeofday (&t0,0);
  resul = Integrate (a, b, n, w);

// Print results
  gettimeofday (&t1,0);
  texec = (t1.tv_sec - t0.tv_sec) + (t1.tv_usec-t0.tv_usec) / 1e6;
  printf ("\n  Integral (= ln x+1 + atan x), from %1.1f to %1.1f, %d trap. = %1.12f\n",
        a,b,n,resul);
  printf ("  Execution time (serie) = %1.3f ms \n\n", texec*1000);
  return (0);
} /*  main  */


// FUNCION Read_data
void Read_data (double* a_ptr, double* b_ptr, int* n_ptr)
{
  float a, b;

  printf ("\n  Introduce a, b (limits) and n (num. of trap.)  \n");
  scanf ("%f %f %d", &a, &b, n_ptr);

  (*a_ptr) = (double)(a);
  (*b_ptr) = (double)(b);
} /* Read_data */


// FUNCTION Integrate: local calculation of the integral
double Integrate (double a, double b, int n, double w)
{
  double resul, x;
  int    i;

  // Integral calculation
  resul = (f(a) + f(b)) / 2.0;
  x = a;

  for (i=1; i<n; i++) {
    x = x + w;
    resul = resul + f(x);
  }
  resul = resul * w;
  return (resul);
} /*  Integrate  */


// FUNCTION f: Function to integrate
double f (double x)
{
  double y;

  y = 1.0 / (x + 1.0) + 1.0 / (x*x + 1.0);
  return (y);
} /* f function */
```

```
/*****************************************************************
        P12-intepar0.c
        Integral of a function by sums of areas of trapezoids
        Using broadcast for data sending and Reduce for data receiving
        >>> TO DO: MODIFY AND COMPLETE <<<
*****************************************************************/

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

double   t0, t1;

void   Read_data (double* a_ptr, double* b_ptr, int* n_ptr);
double Integrate (double a_loc, double b_loc, int n_loc, double w);
double f (double x);

int main (int argc, char** argv)
{
  int         pid, npr;
  double      a, b, w, a_loc, b_loc;
  int         n, n_loc, remainder;
  double      resul, resul_loc;              // Result of the integral: global and local

// MPI Initializations
  MPI_Init (&argc, &argv);
  MPI_Comm_rank (MPI_COMM_WORLD, &pid);
  MPI_Comm_size (MPI_COMM_WORLD, &npr);

// Reading parameters and distributing them to all processes
//      >>> TO DO <<<

  read_data (&a, &b, &n);

// Dividing the calculation among the processes
  w = (b-a) / n;
  n_loc = n / npr;
  remainder = n % npr;
  if (pid < remainder) n_loc = n_loc + 1;
  a_loc = a + pid * n_loc * w;
  if (pid >= remainder) a_loc = a_loc + remainder * w;
  b_loc = a_loc + n_loc * w;

// Local calculation of the integral
  resul_loc = Integrate (a_loc, b_loc, n_loc, w);

// Adding the partial results
//      >>> TO DO: MERGE ALL THE PARTIAL RESULTS <<<


// Print results
  if (pid == 0)
  {
    t1 = MPI_Wtime();
    printf ("\n  Integral (= ln x+1 + atan x), from %1.1f to %1.1f, %d trap.) =
        %1.12f\n",a,b,n,resul);
    printf ("  Execution time (%d proc.) = %1.3f ms \n\n", npr, (t1-t0)*1000);
  }

  MPI_Finalize ();
  return (0);
} /*  main  */


// FUNCION Read_data
void Read_data (double* a_ptr, double* b_ptr, int* n_ptr)
{
  float a, b;

  printf ("\n  Introduce a, b (limits) and n (num. of trap.)  \n");
  scanf ("%f %f %d", &a, &b, n_ptr);

  (*a_ptr) = (double)(a);
  (*b_ptr) = (double)(b);
} /* Read_data */
```

```
// FUNCTION Integrate: local calculation of the integral
double Integrate (double a_loc, double b_loc, int n_loc, double w)
{
  double resul_loc, x;
  int    i;

  // Integral calculation
  resul_loc = (f(a_loc) + f(b_loc)) / 2.0;
  x = a_loc;

  for (i=1; i<n_loc; i++)
  {
    x = x + w;
    resul_loc = resul_loc + f(x);
  }
  resul_loc = resul_loc * w;

  return (resul_loc);
} /*  Integrate  */


// FUNCTION f: function to integrate
double f (double x)
{
  double y;

  y = 1.0 / (x + 1.0) + 1.0 / (x*x + 1.0);
  return (y);
} /* f function */
```

```
/*********************************************
        P13-scatter-gather0.c
        [4 processes]
        >>> TO DO: MODIFY AND COMPLETE <<<
*********************************************/

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

int main (int argc, char** argv)
{
  int  pid, npr, i;
  int  B[16],  Bloc[16];

  MPI_Init (&argc, &argv);
  MPI_Comm_rank (MPI_COMM_WORLD, &pid);
  MPI_Comm_size (MPI_COMM_WORLD, &npr);

  if (pid == 2) for (i=0; i<16; i++) B[i] = i;

// Scattering of B from pid=2
//      >>> TO DO <<<



// Local calculation
//      >>> TO DO: INCREMENT WITH 100 <<<



// Gathering of Bloc in pid=2
//      >>> TO DO <<<



// Print results
  if (pid == 2)
  {
     printf ("\n B in pid=2 after the calculation \n");
     for (i=0; i<16; i++) printf ("%4d", B[i]);
     printf ("\n\n");
  }

  MPI_Finalize ();
  return (0);
} /*  main  */
```

2.2.2. Datu-mota deribatuak / Komunikatzaileak

Laborategian egin ditugun adibideetan oso egitura sinpleko datuak bidali eta hartu ditugu (osoak, koma higikorrekoak… denak ondoz ondokoak), baina askotan malgutasun gehiago behar da bidali/hartu behar diren datuen egitura definitzeko (adibidez, memoriako ondoz ondoko posizioetan ez dauden datuak, edo mota desberdinetakoak…). Gainera, datuak hainbat mezutan bidaltzearen kostua dezente altuagoa da datu horiek guztiak mezu bakar batean bidaltzearena baino. Hori dela eta, MPIk hainbat aukera eskaintzen ditu datuak "formatu" desberdinetan bidali ahal izateko, beti ere prozesuen arteko komunikazioaren latentzia laburtzeko asmoz.

Bestalde, prozesuen arteko komunikazioa "egituratzeko", askotan erabilgarria da prozesuak hainbat komunikatzailetan banatzea, hasierako `MPI_COMM_WORLD` komunikatzaileaz gain (non prozesu guztiak biltzen diren). Hori egiten denean, prozesuek identifikatzaile desberdinak erabiliko dituzte komunikazioan, komunikatzailearen arabera.

Jarduera honen bidez, MPIren mezuetan doazen datuei egitura bat esleitzeko dauden aukerak eta nola erabiltzen diren ulertu beharko dituzu, bai eta prozesu-taldeak kudeatzeko estrategiak ere. Behean dituzu gaia lantzeko zenbait erreferentzia (askoz gehiago aurki dezakezu oso erraz), irakur dezazun eta zure taldekideentzat aurkezpen txiki bat presta dezazun.

Atal honetako kontzeptuak ondo ulertzeko, hurrengo hiru ariketak ebatzi behar dituzu, eta ebazpideak azaldu txosten labur batean (1 edo 2 orrialde gehienez). Programen oinarrizko kodea `templates/puzzle` direktorioan dituzu, zure kontuan.

> Erreferentzia orokorrak

- www.mpi-forum.org/docs/docs.html
- computing.llnl.gov/tutorials/mpi/
- Pacheco P.: *An Introduction to Parallel Programming*. Morgan Kaufmann, 2011.
- Gropp W., Lusk E., Skjellum A.: *Using MPI. Portable Parallel Programming with the Message-Passing Interface*. The MIT Press, 1999.

> Datu-mota deribatuei eta komunikatzaileei buruzko erreferentziak

- Pacheco P.: *Parallel Programming with MPI*. Morgan Kaufmann, 1997. 6. kapitulua, 2, 3 eta 5 atalak; 7. kapitulua, 3-5 atalak.
- Snir M., Otto S., Huss-Lederman S., Walker D., Dongarra J.: *MPI: The Complete Reference, Volume 1, The MPI Core*. The MIT Press, 1999. 3. kapitulua, 4, 5 eta 12 atalak; 5. kapitulua, 1, 2 eta 4 atalak.

> Puzzlearen 2. atala: ariketak

P2.1. MPI programa batean, P3 prozesuak matrize bat dauka —`MAT`, 5×5 oso—, eta haren diagonala bidali behar die gainerako prozesuei. Osatu `P21-diagonal0.c` programa funtzio hori egin dezan, bi kasu hauetan:

- diagonala bektore soil bat gisa hartzen da gainerako prozesuetan, eta osagaien batura kalkulatu eta inprimatzen dute.
- diagonala bertako `MAT` matrizeen diagonaletan hartzen da gainerako prozesuetan.

Exekutatu programa 4 prozesadoretan. Honako hau inprimatu behar du:

```
The sum of the received data in P1 is: 40
The new diagonal of MAT in P0 is:  0  4   8  12  16
```

P2.2. Osatu `P22-pack0.c` programa, non P0k bidali behar dizkio P1i, mezu bakar batean, hiru datu hauek: `A` matrizea (100×100 oso), `B` bektorea (200 *float*), eta `C` (*double* bat). Horretarako, P0k datuak paketatzen ditu eta paketea P1i bidaltzen dio; P1ek mezua hartzen du eta datuak despaketatzen ditu.

Exekutatu programa 2 prozesadoretan; honakoa inprimatu behar du:

```
Received in P1: A[10][10] = 100   B[33] = 13.2   C = 2.2
```

P2.3. Aplikazio paralelo jakin bat 8 prozesutan exekutatzen da. Une jakin batean, 4 prozesuko bi talde sortu behar ditugu: 0-3 prozesuak batean, eta 4-7 prozesuak bestean. Talde bakoitzean, prozesuek beste identifikadore bat izango dute.

Ondoren, datu-bilketa bat exekutatzen da talde bakoitzean: hasieran, prozesuetan 5 osagaiko `V` bektoreak daude (hasieratuta prozesuen `pid` balioekin); eta, bukaeran, prozesu guztiek 20 osagaiko `W` bektorea izan behar dute, talde bakoitzeko `V` bektoreak lotuta.

Osatu `P23-groups0.c` programa funtzio hori egin dezan. Exekutatu programa 8 prozesutan; honako hau inprimatu behar du:

```
W(0,5,10,15) data in pid 1 [pid2 1, group 0]: 0 1 2 3
W(0,5,10,15) data in pid 5 [pid2 1, group 1]: 4 5 6 7
```

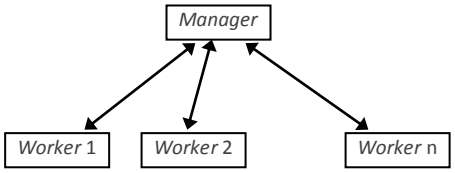
```
/*************************************************************
        P21-diagonal0.c
        MPI program for sending a diagonal of a matrix
        [4 processes]
        >>> TO DO: MODIFY AND COMPLETE <<<
*************************************************************/

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

#define N 5

int main (int argc, char **argv)
{
  int   i, j, sum;
  int   MAT[N][N], buf[N];
  int   pid;


  MPI_Init (&argc, &argv);
  MPI_Comm_rank (MPI_COMM_WORLD, &pid);

// Initialisation of the matrices in all the processes
  for (i=0; i<N; i++)
  for (j=0; j<N; j++)  MAT[i][j] = pid*i + j;

// Defining the diagonal type
//      >>> TO DO <<<



// 1. Sending the diagonal of the matrix MAT in P3 to P0, P1 and P2
// It is received as a vector in a buffer
//      >>> TO DO <<<


// In order to check, P1 adds the received data and print the result


   sleep (2);


// 2. Sending the diagonal of the matrix MAT in P3 to P0, P1 and P2
// It is received in the MAT matrix in each process
//      >>> TO DO <<<



// In order to check, P0 prints the new diagonal of the MAT matrix



  MPI_Finalize ();
  return 0;
}
```

```
/*************************************************************
        P22-pack0.c
        MPI program using pack/unpack
        [2 processes]
        >>> TO DO: MODIFY AND COMPLETE <<<
*****************************************************************/

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

#define sizeA 100
#define sizeB 2000

#define sizebuf 50000

int main (int argc, char **argv)
{
  int   pid, i, j;
  int   A[sizeA][sizeA];
  float B[sizeB];
  double C;

  char buf[sizebuf];
  int pos = 0;
  MPI_Status info;


  MPI_Init (&argc, &argv);
  MPI_Comm_rank (MPI_COMM_WORLD, &pid);


// P1 initialises the data
//    >>> TO DO <<<

    for (i=0; i<sizeA; i++)
    for (j=0; j<sizeA; j++) A[i][j] = i*j;

    for (i=0; i<sizeB; i++) B[i] = (float)i*0.4;

    C = 2.2;



// Packing the data in P1 and sending the packet to P2
//      >>> TO DO <<<



// Receiving the packet and unpacking the data in P2
//      >>> TO DO <<<



    printf ("\n Received in P2: A[10][10] = %d  B[33] = %3.1f  C = %3.1f\n\n",
            A[10][10], B[33], C);

  MPI_Finalize ();
  return 0;
}
```

```
/*****************************************************************
       P23-groups0.c
       MPI program using communicators (Split function)
       [8 processes]
       >>> TO DO: MODIFY AND COMPLETE <<<
*****************************************************************/

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

#define N 20

int main (int argc, char* argv[])
{
  int       npr, pid, i;
  int       V[N/4], W[N];
  MPI_Status info;


  MPI_Init (&argc, &argv);
  MPI_Comm_size (MPI_COMM_WORLD, &npr);
  MPI_Comm_rank (MPI_COMM_WORLD, &pid);

  for (i=0; i<N/4; i++) V[i] = pid;

// Groups and communicators creation
//      >>> TO DO <<<



// Sending from pid=0 in each group to the others processes
//      >>> TO DO <<<



// The process 0 in each group print  W(0,5,10,15) values and
// its pid in the global communicator and in the local one
//      >>> TO DO <<<



  MPI_Finalize ();
  return (0);
}  /* main */
```

2.2.3. Lan-banaketa dinamikoa / "Mugen" arazoa

Prozesuen artean banatzen den lan-karga orekatua izan dadin, batez ere atazen exekuzio-denbora aurretik ezagutzen ez denean, beharrezkoa da lan-banaketa dinamikoa izatea, hau da, banaketa exekuzioan zehar egitea.

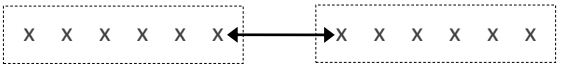


Kasu horietan, ohikoa da erabiltzea oso ezaguna den programazio-eredua: "*manager/worker*" eredua alegia. Eredu horretan, prozesuetako batek, *manager* prozesuak, batetik, atazak banatzen ditu lan-eskaerak heltzen direnean eta, bestetik, emaitzak biltzen ditu. Beste prozesuek, *worker*-ek, atazak eskatu, ataza horiek exekutatu eta emaitzak itzultzen dituzte, harik eta ataza guztiak denon artean exekutatu arte.

Hauxe izan daiteke algoritmo orokorra:

```
baldin manager
    lana dagoen bitartean
        itxaron eskaerak
        bidali lana
        hartu emaitzak / bidali lana

    lana bukatu bada
        hartu emaitzak / bidali bukaera-kodea

baldin worker
    bukaera-kodea heldu bitartean
        eskatu lana
        hartu lana / exekutatu lana
        bidali emaitzak / eskatu lana
```

Bestalde, aplikazio paraleloetan datuen banaketa egiteko maiz agertzen den beste arazo bat datuen mugena da. Esaterako, bektore bat banatzen dugunean, zati baten azken osagaia eta hurrengo zatiaren aurreneko osagaia prozesadore desberdinetara banatuko dira. Hasieran memoriako ondoz ondoko posizioetan baziren ere, orain guztiz urrun daude.



Beraz, haien artean erlazioren bat baldin bazegoen kalkulurako, orain komunikazio-funtzioen bidez gauzatu beharko da erlazio hori.

Jarduera honen bidez ulertu beharko dituzu aipatutako bi arazoak: atazen lan-banaketa dinamikoa, *manager/worker* eredua erabiliz, eta datuen trukea banaketaren mugetan. Aurkezpen txiki bat prestatu behar duzu zure taldekideentzat.

Atal honetako kontzeptuak ondo ulertzeko, hurrengo bi ariketak ebatzi behar dituzu, eta ebazpideak azaldu txosten labur batean (1 edo 2 orrialde gehienez). Programen oinarrizko kodea `templates/puzzle` direktorioan dituzu, zure kontuan.

> Puzzlearen 3. ataleko ariketak

P3.1. `P31-collatzser.c` programan 1–320 tarteko zenbaki naturalei Collatz-en algoritmoan oinarritutako funtzio bat aplikatzen zaie, non zenbaki bakoitza prozesatzeko lan-karga proportzionala baita zenbakia 1era konbergitzeko behar diren iterazioen kopuruarekin.

Programaren bi bertsio paralelo egin behar dituzu. Aurrenekoan, atazak (zenbakien prozesamendua) estatikoki banatu behar dira prozesadoreen artean, kontsekutiboki; hau da, prozesadore bakoitzak ondoz ondoko 320/`npr` zenbaki prozesatuko ditu.

Bigarrenean, atazen banaketak dinamikoa izan behar du, eskaeren arabera. Prozesu batek (P0k adibidez) *manager* gisa funtzionatuko du, eta gainerakoek *worker* moduan. P0k banan-banan bidaliko dizkie zenbakiak *worker* prozesuei, haiek hala eskatzen dutenean. Zenbakia prozesatu eta konbergitzeko behar izan duen iterazio kopurua itzuliko dio *manager*-i. Zenbaki gehiago geratzen badira prozesatzeko, beste bat bidaliko zaio, hala denon artean zenbaki guztiak prozesatu arte. *Manager* prozesuak kontrolatu behar du zenbat datu prozesatu duen *worker* bakoitzak, bai eta zer zenbakik behar izan duen iterazio gehien konbergitzeko.

Bi programak egiaztatu ondoren, exekutatu bertsio estatikoa 2, 4, 8, 16, 32 eta 64 prozesurekin; eta bertsio dinamikoa 1+1, 1+2, 1+4, 1+8, 1+16, 1+32 eta 1+63 prozesurekin. Neurtu exekuzio-denborak, eta kalkulatu lortu dituzun azelerazio-faktoreak eta eraginkortasunak. Marraztu bi parametro horien portaera prozesu kopuruaren arabera eta azaldu emaitzak.

P3.2. Aplikazio jakin batean (`P32-convoser.c`), `N` = 1.002 osagaiko `A` bektorea prozesatzen da, zeinari konboluzioko eragiketa hau aplikatzen baitzaio modu iteratiboan:

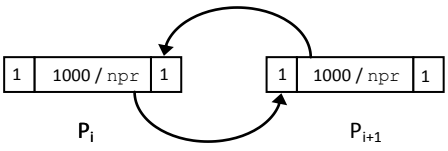
```
for (i=1; i<N-1; i++) Aux[i] = (A[i-1] + 2*A[i] + A[i+1]) / 4
for (i=1; i<N-1; i++) A[i] = Aux[i]
```

harik eta bektorearen balio maximoa hasierako maximoaren % 60ra heltzen den arte.

Paraleloan erabiliko den prozesu kopurua 1.000ren zatitzaile bat izango da. Idatz ezazu programaren bertsio paraleloa, non:

a. P0k, prozesatu behar diren bektorearen 1.000 osagaiak banatzen ditu 1.000/`npr` osagaiko zatitan: lehen zatia berak prozesatuko du, bigarren zatia P1ek…;
b. prozesu bakoitzak dagokion zatia prozesatu eta maximo lokala kalkulatzen du;
c. prozesuek maximo lokalak bidaltzen dizkiote P0ri, maximo globala kalkula dezan; ondorioz, P0k gainerako prozesuei abisatuko die jarraitu behar den edo eragiketa bukatu den;
d. bukatutakoan, prozesu guztiek `A` bektorearen zatiak bidaltzen dizkiete P0ri, hasierako bektorea berrosa dezan.

Adi: prozesadore bakoitzean bektore-zati bat prozesatzeko, dituen osagaiaz aparte, lehen osagaiaren aurrekoa eta azken osagaiaren hurrengoa behar dira, aurreko eta hurrengo



prozesadoreetan egongo direnak, hurrenez hurren. Beraz, prozesu bakoitzean 1 + 1000/`npr` + 1 osagaiko buffer bat erabiltzea iradokitzen dizugu, eta, konboluzioko iterazio bati ekin baino lehen, truke bat egitea `pid`-1 eta `pid`+1 prozesadoreekin bi datu horiek eskuratzeko (iterazioz iterazio aldatzen baitira haien prozesadoreetan).

```
/************************************************************
       P31-collatzser.c
************************************************************/

#include <stdio.h>
#include <sys/time.h>
#include <stdlib.h>

#define NUMBER 320


int collatz (int  n)
{
  int steps=0;

  while (n > 1)
  {
     if ((n % 2) == 1) n = 3*n + 1;
     else n = n/2;

     steps++;
  }
  return (steps);
}

void work (int steps)
{
  usleep (2000*steps);
}


main (int argc, char *argv[])
{
  int n, steps, total_steps=0, max_steps=0, n_max_steps;
  struct timeval t0,t1;
  double tex;

  printf ("\n COLLATZ (serial): 1 - %d\n\n", NUMBER);

  gettimeofday (&t0, 0);

  for (n=1; n<=NUMBER; n++)
  {
    steps = collatz(n);
    work (steps);
    total_steps += steps;
    if (steps > max_steps) {n_max_steps = n; max_steps = steps;}
  }

  gettimeofday (&t1, 0);

  tex = (t1.tv_sec - t0.tv_sec) + (t1.tv_usec - t0.tv_usec) / 1e6;
  printf (">>Total steps: %d >>Num_max_steps: %d (%d steps) >>Execution time:
        %1.3f ms\n\n", total_steps, n_max_steps, max_steps, tex*1000);
}
```

```
/*************************************************************
       P32-convoser.c
*************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <values.h>

#define NELEM 1002


main (int argc, char*argv[])
{
  int i, end, steps=0;
  float lim, max, max_ini;

  float A[NELEM], Aux[NELEM];


// initial values for A
  max_ini = MINFLOAT;
  A[0] = A[NELEM-1] = 150;
  for (i=1; i<NELEM-1; i++)
  {
    A[i] = (rand() % 1000) / 3.0;
    if (A[i] > max_ini) max_ini = A[i];
  }


// limit to converge
  lim = 0.6 * max_ini;

  printf("\n > Initial state (last 10 elements of A)\n");
  for (i=1; i<11; i++) printf("A[%d]: %.3f\n", NELEM-i, A[NELEM-i]);
  printf ("\n Initial max: %.3f\n", max_ini);


// convolution phase
  end = 0;
  while (!end)
  {
    max = MINFLOAT;

  // convolution
    for (i=1; i<NELEM-1; i++)
    {
      Aux[i] = (A[i-1] + 2*A[i] + A[i+1]) / 4;
      if (Aux[i] > max) max = Aux[i];
    }
    for (i=1; i<NELEM-1; i++) A[i] = Aux[i];

    steps++;

  // convergence control
    if (max < lim) end = 1;
  }

  printf ("\n\n Convolution steps: %d   Final max: %.3f\n\n", steps, max);
  printf (" > Final state (last 10 elements of A)\n");
  for (i=1; i<11; i++) printf("A[%d]: %.3f\n", NELEM-i, A[NELEM-i]);
}
```

2.3. ARIKETA OSAGARRIAK

Hemen dituzu beste ariketa batzuk, nahi baduzu, puzzlea denon artean partekatu eta gero probatzeko.

A1. Aplikazio paralelo jakin batean, `pid` = 0 prozesuak luzera desberdineko mezuak (osoak) hartzen ditu gainerako prozesuetatik, eta mezuetako datuak prozesatzen ditu `PROC(*dat,tam)` funtzioa exekutatuz, non `*dat` datuen hasiera-helbidea eta `tam` datu kopurua diren. Mezu bat hartutakoan, mezu motz batekin erantzuten da (oso bat) mezua hartu dela baieztatzeko eta, ondoren, datuak prozesatzen dira. Idatzi P0k exekutatuko duen kode zatia lan hori egiteko, baldin eta:

a. mezuak `pid`-ren ordena hertsian hartu eta prozesatu behar dira (aurrenik P1enak, gero P2renak…).

b. mezuak heldu diren ordenan hartu eta prozesatu behar dira.

A2. Aplikazio paralelo jakin batean, P0k `M[N][N]` matrizea bidaltzen dio P1i. P1ek ez du ezagutzen hartuko duen matrizearen tamaina; beraz, aurretik "postontzian begiratzen du", eta informazio horrekin memoria-espazioa erreserbatzen du eta mezua hartzen du. Idatzi eragiketa hori egiten duen MPI programa bat. Exekutatu kasu pare bat edo, eta egiaztatu emaitzak.

A3. Exekuzio paralelo jakin batean, P3 prozesuak mezu bat hartu behar du P1 prozesutik, baina ez ditu ezagutzen datu kopurua ezta mezuaren etiketa (`tag`) ere. `tag` = 0 bada, mezuak 100 osoko bektore bat ekarriko du; `tag` = 1 bada, mezuak *float* bat ekarriko du. Idatzi funtzio hori egiten duen programatxo bat. Exekutatu kasu pare bat edo, eta egiaztatu emaitzak.

A4. Aplikazio baten exekuzio paraleloko une jakin batean, prozesuek 10×10 osoko matrize bana dute, `A`, eta exekuzioarekin jarraitzeko denek behar dute matrize horien batura. Idatzi kodea eragiketa hori betetzeko, eta erabili komunikazio kolektiboko funtzioak. Adibide gisa, exekutatu programa 8 prozesutan, hasieratu matrizeak prozesuen `pid`-rekin, eta inprimatu emaitza bi prozesutan (P2 eta P4, adibidez).

A5. `matvecser.c` programan, eragiketa hauek exekutatzen dira matrize eta bektoreekin:

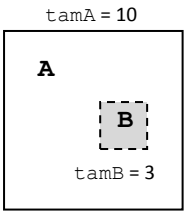
```
C[N] = A[N][N] * B[N]        (aldagai guztiak, double)
D[N] = A[N][N] * C[N]
PE = ∑(C[i]· D[i])
```

Hasieran, bektoreen tamaina eskatzen da, `N`, gero haiei memoria esleitzeko.

Idatzi eta exekutatu programaren bertsio paralelo bat. Kalkuluari ekin baino lehen, P0 prozesuak `A` eta `B` hasieratzen ditu; bukaeran, `C`, `D` eta `PE` P0ra bildu behar dira. Datuak banatzerakoan, onartu behar da zatien tamainak berdinak ez izatea (`_v` funtzioak), hau da, funtzionatu behar du edozein `N`-tarako eta edozein prozezu kopurutarako. Kodea idatzi baino lehen, analiza itzazu egin behar diren eragiketak eta erabaki zein daturekin lan egingo duen prozesu bakoitzak, beharko den memoria dinamikoa erreserbatu eta datu-banaketa egokia egin ahal izateko. Egiataztu emaitzak serieko bertsioarekin.

A6. MPI programa batean, P0 prozesuak 10×10 osoko matrize bat dauka (A), eta bidali behar dizkie gainerako prozesuei periferiako osagaiak (lehen eta azken lerroa, eta lehen eta azken zutabea). Prozesuek datuak hartu eta bertako matrizeen posizio baliokideetan kopiatuko dituzte datuak Idatzi funtzio hori egiten duen programa; defini itzazu behar diren datu-mota deribatuak, paketatu bi lerroak eta bi zutabeak, eta bidali paketea. Eragiketa egiaztatzeko, P1ek inprimatu behar du matrize berria.

A7. Exekuzio paralelo batean, P0 prozesuak A matrizea erabiltzen du (10×10 zenbaki oso), eta P1i bidali behar dio, maiz, mezu bakar batean, 3×3 tamainako B azpimatrizeak.



Idatzi eragiketa hori egingo duen programa. Horretarako: **(a)** definitu datu-egitura horri dagokion datu-mota deribatua; eta **(b)** bidali P1i Aren (2,5) osagaian hasten den 3×3 tamainako B matrizea.

Hasieratu A matrizea P0n balio hauekin:

```
for (i=0; i<tamA; i++)
for (j=0; j<tamA; j++)
    A[i][j] = i + j;
```

Inprimatu P1en hartu den matrizea. Honako hau inprimatu beharko luke:

```
7     8     9
8     9     10
9     10    11
```

A8. A matrizea bidali behar da P0tik P1era, baina P1en A matrize iraulia lortu behar da. Idatzi programa bat hori egiteko:

1. Definitu "zutabe" datu-mota deribatua.
2. P0k A matrizearen zutabeak bidaltzen dizkio P1i, bana-banan.
3. P1ek zutabeak hartzen ditu, baina matrize baten lerroetan. Zutabe guztiak hartutakoan, matrize hori Aren iraulia izango da.

Zure soluzioa egiaztatzeko, exekutatu programa matrize txiki batekin, 4×4koa; hasieratu matrizea P0n zorizko zenbakiekin, eta inprimatu matrizea P0n eta P1en.

Egin daiteke eragiketa osoa mezu bakar batekin? Nola?

```
/*************************************************************************
        matvecser.c
        C() = A()() x B()
        D() = A()() x C()
        PE = sum (C() . D())
*************************************************************************/

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


int main(int argc, char **argv)
{
  int    N, i, j;
  double PE = 0.0;
  double *Am, *B, *C, *D;

  printf ("\n Vector lengths  (<1000) = ");
  scanf ("%d" , &N);

/*   memory allocation */
  Am = (double *) malloc (N*N * sizeof(double));

  B = (double *) malloc (N * sizeof(double));
  C = (double *) malloc (N * sizeof(double));
  D = (double *) malloc (N * sizeof(double));

/* initial values  */
  for (i=0; i<N; i++)
  {
    for (j=0; j<N; j++) Am[i*N+j] = (double) (N-i)*0.1/N;
    B[i] = (double) i*0.05/N;
  }

/* calculus */
  for (i=0; i<N; i++)
  {
    C[i] = 0.0;
    for (j=0; j<N; j++) C[i] = C[i] + Am[i*N+j] * B[j];
  }

  PE = 0.0;
  for (i=0; i<N; i++)
  {
    D[i] = 0.0;
    for (j=0; j<N; j++) D[i] = D[i] + Am[i*N+j] * C[j];
    PE = PE + D[i]*C[i];
  }

/* results */
  printf ("\n\n   PE = %1.3f \n", PE);
  printf ("   D[0] = %1.3f, D[N/2] = %1.3f, D[N-1)] = %1.3f\n\n", D[0],D[N/2],D[N-1]);

  free (Am);
  free (B);
  free (C);
  free (D);
  return (0);
}
```

3. PARALELIZATU BEHAR DEN APLIKAZIOA

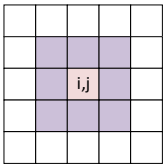
Puzzle moduan egindako ikasketa-fasearen ondoren, baditugu jada aplikazioak paralelizatzen hasteko behar diren tresna eta estrategiak. Proiektua aurkeztean azaldu genuen moduan, paralelizatu behar den aplikazioak zirkuitu inprimatuko txartelen portaera termikoa simulatzen du, txipak txartelean kokatzeko posiziorik egokienen bila, txartelaren batez besteko tenperatura minimizatzen dituzten posizioak, alegia.

3.1. APLIKAZIOAREN AZALPENA

`heat_s.c` programa aplikazioaren serieko bertsioa da. Diferentzia partzialetako ekuazioak ebazteko kasu jakin bat baino ez da (Poisson motakoa), oso ohikoak hainbat eta hainbat aplikazio tekniko/ zientifikotan.

Adibide honetan, abiapuntua txartel bat da, non beroa sortzen duten hainbat txip kokatu behar baitira. Beroa txartel osoan zehar barreiatuko da, harik eta egoera egonkor batera heldu arte. Batez besteko tenperatura kalkulatzeko, txartela bi dimentsioko puntu-sare batean banatzen da, eta sareko puntu bakoitzaren tenperatura aldatzen da, iterazioz iterazio, beraren eta auzokoen tenperaturen arabera, adierazpen honen arabera:

$$T^1_{i,j} = T^0_{i,j} + 0{,}1 \times [8 \text{ auzokoen } \Sigma T^0 - 8 \times T^0_{i,j}]$$



Adierazpen horren araberako tenperatura berria saretxoko (`i,j`) puntu bakoitzean kalkulatu ondoren, tenperatura igotzen da txipek beroa txertatzen duten puntuetan, eta tenperatura jaisten da aireztatuta dauden txarteleko posizioetan. Tenperatura gaurkotzeko eta barreiatzeko prozesuak errepikatu egiten dira batez besteko tenperatura egonkortu arte, edo, bestela, ezarritako iterazio kopuru maximoa gainditu arte. Bi eragiketak `diffusion` eta `thermal_update` funtzioetan egiten dira, hurrenez hurren.

>> thermal_update

```
// heat injection at chip positions

 for (i=1; i<NROW-1; i++)
 for (j=1; j<NCOL-1; j++)
   if (grid_chips[i*NCOL+j] > grid[i*NCOL+j])
     grid[i*NCOL+j] += 0.05 * (grid_chips[i*NCOL+j] - grid[i*NCOL+j]);


 // air cooling at the middle of the card

 for (i=1; i<NROW-1; i++)
 for (j=0.45*(NCOL-2)+1; j<0.55*(NCOL-2)+1; j++)
     grid[i*NCOL+j] -= 0.01 * (grid[i*NCOL+j] - param.t_ext);
```

>> difussion

```
while (end == 0)
{
  niter++;
  Tmean = 0.0;

  // heat injection and air cooling
  thermal_update (param, grid, grid_chips);

  // thermal difussion
  for (i=1; i<NROW-1; i++)
  for (j=1; j<NCOL-1; j++)
  {
    T = grid[i*NCOL+j] +
      0.10 * (grid[(i+1)*NCOL+j] + grid[(i-1)*NCOL+j] + grid[i*NCOL+(j+1)] +
          grid[i*NCOL+(j-1)] + grid[(i+1)*NCOL+j+1] + grid[(i-1)*NCOL+j+1] +
          grid[(i+1)*NCOL+(j-1)] + grid[(i-1)*NCOL+(j-1)]
          - 8*grid[i*NCOL+j]);

    grid_aux[i*NCOL+j] = T;
    Tmean += T;
  }

  //new values for the grid
  for (i=1; i<NROW-1; i++)
  for (j=1; j<NCOL-1; j++)
    grid[i*NCOL+j] = grid_aux[i*NCOL+j];

  // convergence every 10 iterations
  if (niter % 10 == 0)
  {
    Tmean = Tmean / ((NCOL-2)*(NROW-2));
    if ((fabs(Tmean - Tmean0) < param.t_delta) || (niter > param.max_iter))
         end = 1;
    else Tmean0 = Tmean;
  }
} // end while
```

Txartela irudikatzen duten 2D saretxoko puntuak (`grid`) tenperatura jakin batera hasieratzen dira, eta hortik eboluzionatzen da haien tenperatura, txipen tenperaturen arabera. Dena den, txartelaren goi/beheko eta ezker/eskuineko ertzetako puntuak ez dira prozesatzen, eta, ondorioz, haien tenperatura hasierakoa da beti.

Tamaina bereko beste matrize batek (`grid_chips`) txipek txartelean okupatzen dituzten puntuen tenperatura adierazten du, puntuak non tenperatura igo behar den iterazioz iterazio. Matrize hori sarrera-fitxategi bateko datuen arabera hasieratzen da, non adierazten baitira txipen tamainak, posizioak, tenperaturak eta abar. Bestalde, txartelaren zerrenda bertikal bat aireztatuta dago txartela freskatzeko, eta ondorioz, puntu horietan tenperatura jaitsi behar da iterazio bakoitzean. Bi eragiketa horiek, beroa txertatzea eta aireztatzea, `thermal_update` funtzioan betetzen dira.

Beroa barreiatzeko algoritmoak bi matrize erabiltzen ditu, bata saretxoko puntuetako uneko tenperaturekin (`grid`), eta bestea iterazio horretan kalkulatzen ari diren tenperatura berriekin (`grid_aux`). Iterazioa bukatzerakoan, matrize bat bestean kopiatzen da.

`Tmean`, txartelaren batez besteko tenperaturaren eguneratzea iterazioz iterazio egin daiteke, edo bestela, hainbat iterazio exekutatu eta gero; kasu honetan, 10 iteraziotan behin egiten da.

`Tmean0` aldagaia "aurreko" batez besteko tenperatura da (hastean, hasierako tenperatura); baldin eta aurreko eta oraingo batez besteko tenperaturen diferentzia txikiagoa bada ezarritako balio jakin bat baino, simulazioa bukatu egiten da.

Programa nagusia sinplea da:

```
{
  ...
  read_data (argv[1], &param, &chips, &chip_coord);
  ...

  // loop to process chip configurations
  for (conf=0; conf<param.nconf; conf++)
  {
    gettimeofday (&t0, 0);

    // initial values for grids
    init_grid_chips (conf, param, chips, chip_coord, grid_chips);
    init_grids (param, grid, grid_aux);

    // main loop: thermal injection/disipation until convergence (t_delta or max_iter)
    diffusion (param, grid, grid_chips, grid_aux);

    // writing configuration results
    gettimeofday (&t1, 0);
    tej[conf] = (t1.tv_sec - t0.tv_sec) + (t1.tv_usec - t0.tv_usec)/1e6;
    results_conf (conf, param, grid, grid_chips, &BT);
  }

  // writing best configuration results
  results (param, &BT, argv[1]);
  for (conf=0; conf<param.nconf; conf++) tsim += tej[conf];
  printf ("   > Time (serial): %1.3f s \n\n", tsim);
}
```

`read_data` funtzioak sarrera-fitxategia irakurtzen du, non simulatu behar diren txipen konfigurazioak baitaude. Datuak aldagaian hauetan gordetzen dira:

>> `param`: `info_param` motako *struct* bat, simulazioaren parametro orokorrak dituena (sarrera-fitxategiaren aurreneko lerroa)

```
struct info_param {
  int    nconf, nchip, max_iter, scale;
  float  t_ext, tmax_chip, t_delta;
};
```

`nconf`: simulatu behar diren konfigurazioen kopurua; konfigurazio bakoitzak txip berberak ditu, baina posizio desberdinetan.

`nchip`: txarteleko txip kopurua.

`max_iter`: simulazioa bukatzeko irizpidea: iterazio kopuru maximoa.

`scale`: simulazioaren eskala-faktorea (1etik 12ra); 1: oinarrizko txartela, 200×100 puntu, probak egiteko erabiliko duguna; 10: 2000×1000 puntuko txartela, benetako emaitzak ateratzeko erabiliko duguna, aplikazioa garatu eta egiaztatu ondoren.

`t_ext`: saretxoko puntuen hasierako tenperatura.

`tmax_chip`: txipen tenperatura maximoa.

`t_delta`: simulazioa bukatzeko irizpidea: tenperatura-diferentzia balio hori baino txikiagoa da.

>> `chips`: `info_chips` motako *struct* bat, txarteleko txipen definizioekin, tamaina (`h`, `w`) eta tenperatura (`t_chip`).

```
struct info_chips {
  int    h, w;
  float  t_chip;
};
```

>> `chip_coord`: (`nconf` × 2×`nchip`) tamainako matrize bat, lerro bat simulatu behar den konfigurazio bakoitzeko, txipen posizioak adierazteko (ezker aldeko goiko erpinaren `x` eta `y` koordenatuak).

▪ Txartelaren definizioa

Txartela definitzen duen **sarrera-fitxategia**, hasierako datuak dituena, hiru bloketan banatuta dago (hau adibide bat baino ez da:

3 4 20.0 160.0 0.01 10000 10	**1. blokea** (`param`): simulazioaren parametro orokorrak: simulatzeko konfigurazio kopurua (3); txip kopurua (4); hasierako tenperatura (20.0); txipen tenperatura maximoa (160.0); konbergentzia-irizpideak: tenperatura (0.01) eta iterazio kopuru maximoa (10000); eta eskala-faktorea (10).
40 40 100.0 50 20 160.0 30 60 120.0 20 20 80.0	**2. blokea** (`chips`), txipen definizioa, lerroz lerro; adibidez, 4 txip; lehenbizikoa: tamaina (40×40) eta tenperatura maximoa (100.0); bigarrena.
86 15 135 49 21 27 90 59	**3. blokea** (`chip_coord`): simulatu behar diren konfigurazioetako txipen (`x`,`y`) posizioak. Adib., 1. txipa: 86, 15; tamaina 40, 40 izanik, honako posizio hauek okupatuko ditu 200×100 puntuko oinarrizko txartelean: (86-125, 15-54).
126 40 26 72 168 29 62 23	2. konfigurazioa
67 35 129 2 22 11 119 84	3. konfigurazioa

`card` fitxategiak simulatu behar diren konfigurazioen deskribapena dauka. 4 txipeko 20 konfigurazio dira, 2000×1000 puntuko saretxo batean (eskala-faktorea = 10). Exekuzio-denbora handia denez, probetarako `card0` fitxategia erabiliko dugu, non bakarrik 4 konfigurazio ageri baitira, saretxoaren oinarrizko tamainan (200×100 puntu).

Programa exekutatzeko, exekutagarria eta txartela adierazi behar dira. Adibide:

```
> heat_s card0
```

▪ Emaitzak

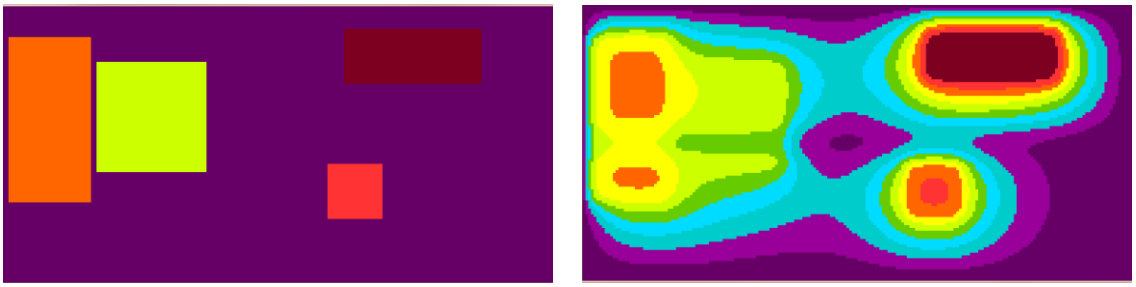
Simulazioaren emaitza gisa, batez besteko tenperatura baxuena lortzen duen konfigurazioaren datuak gordetzen dira, *struct* batean (`BT`): konfigurazio-zenbakia, batez besteko tenperatura,

hasierako txip-matrizea, eta azkeneko tenperaturen matrizea. Bi fitxategi sortzen dira datu horiekin: `card_ser.chips` (txipen matrizea) eta `card_ser.res` (azken tenperaturen matrizea).

Bisualizazioko aplikazio sinple batekin, matrizeak irudikatu daitezke pantailan, txartel txikienaren kasuan bakarrik (`card0`, 200×100 puntuko matrizeak); nahikoa da exekutatzea:

```
> vfinder card0_ser.res
```

Leiho berri batean, tenperaturen banaketa ageri da, kolore desberdinetan (eta konparatu dezakezu txipen tenperaturekin, `card0_ser.chips` fitxategikoak).



Hasierako tenperaturen banaketa `card0` txartelean (4 txip) eta azken emaitza difusio termikoaren prozesua simulatu ondoren.

▪ Programaren egitura

Serieko programa 3 modulutan banatuta dago: `heat_s.c` (programa nagusia), `diffusion.c` (`diffusion` eta `thermal_update` funtzioak), eta `faux.c` (datuak irakurtzeko eta emaitzak idazteko funtzio laguntzaileak). Exekutagarria sortzeko, hiru programak konpilatu behar dira; adibidez:

```
> icc -o heat_s heat_s.c diffusion.c faux.c
```

Fitxategi guztiak `templates/aplikazioa` karpetan daude (`.c` iturburuak, `.h` buruak, eta txartelen definizioak).

3.2. EGIN BEHAR DEN LANA

Serieko programa paralelizatu behar da, *cluster*-ean exekutatzeko. Bi bertsio egin behar dira.

▪ 1. fasea

Txipen konfigurazio bakoitza prozesu guztien artean exekutatu behar da; lehenbiziko konfigurazioa simulatu ondoren, bigarrenari ekingo diete prozesu guztiek, gero hirugarrenari… harik eta konfigurazio guztiak simulatu arte.

Txarteleko puntu-saretxoa zerrenda horizontaletan banatu behar da prozesuen artean. Puzzleko 3.2. ariketa ebazteko egin duzuen moduan, kontuan hartu prozesuek erabili beharko dituztela, iterazio bakoitzean, "mugetako" datuak, `pid`+1 eta `pid`–1 prozesuetan daudenak, alegia

Egiazta dezakezu emaitza zuzena dela, adibidez, serieko eta paraleloko fitxategiak konparatuz `diff` komandoaren bidez:

```
> diff card0_ser.res card0_par.res
```

eta tenperaturen banaketa ikusi honela:

```
> vfinder card0_par.res        (edo card0_par.chips)
```

Programa zuzena dela egiaztatu eta gero, exekuta ezazu benetako konfigurazio-fitxategiarekin (`card`), seriean eta 2, 4, 8, 16, 24, eta 32 prozesurekin. Neurtu exekuzio-denborak, eta kalkulatu lortutako azelerazio-faktoreak eta eraginkortasunak. Irudikatu datu horiek, eta ondorioak atera. Emaitzetan oinarrituta, balioetsi zein den prozesu kopuru egokiena (*P*) problema honetarako *cluster* honetan.

▪ 2. fasea

1. fasea bukatuta, programaren bigarren bertsio paraleloa egin behar da, beste estrategia bat erabiliz. Prozesu guztiak konfigurazio jakin bat simulatzen aritu beharrean, banatu behar ditugu prozesuak modu honetan: *manager* bat eta *P worker*-eko taldeak (aurreko fasean balioetsitako *P* bera). Hala, atazen banaketa dinamikoa egin daiteke, puzzleko 3.1 ariketan egin duzun modu bertsuan.

Manager-ak, beraz, simulatzeko konfigurazioak banatzen dizkie *worker*-eko taldeei, haiek hala eskatuta. Talde bakoitzak konfigurazio jakin bat simulatuko du eta emaitzak *manager* prozesuari bidaliko dizkio, beste konfigurazio bat bidali diezaion, talde guztien artean konfigurazio guztiak simulatu arte.

Bertsio honetarako, definitu eta erabili behar dituzu prozesu-taldeak, talde bakoitzeko prozesuen artean informazioa truka dezaten. *P* prozesuko talde bakoitzean, prozesu jakin bat komunikatuko da *manager*-arekin, atazak eskatu eta emaitzak bidaltzeko. Talde bakoitzaren barruan, simulazioa lehen faseko bera da.

Programa egiaztatu ondoren (`card0` fitxategia erabiliz), exekutatu, lehen bezala, `card` fitxategiarekin. Neurtu exekuzio-denborak kasu hauetan: 1 + 1×*P*, 1 + 2×*P*, 1 + 3×*P*, 1 + 4×*P*… prozesu. Kalkula itzazu azelerazio-faktoreak eta eraginkortasunak.

Konparatu bi bertsioen emaitzak eta atera ondorioak.

▪ **Txosten teknikoa**

Proiektuaren emaitza gisa, txosten teknikoa idatzi behar da: hasierako problema, nola ebatsi den, lortutako emaitzak, ondorioak, eta abar (bi faseetan). Txostenak, problema eta ebazpidea ondo interpretatzeko materialak izan behar ditu: grafikoak, datu-taulak, kode-zatiak komentatuta…, txostenak idazteko banatu den dokumentuaren arabera. Gehigarri gisa, aplikazioaren kodea sartu behar da.

3.3. ENTREGATZEKOAK ETA EPEAK

E6.1 Aplikazioaren 1. faseko oinarrizko emaitzak: zenbakizko emaitzak, grafikoak… (2 orri edo). Data: **apirilaren 26a**

E6.2 Behin betiko txosten teknikoa. Data: **maiatzaren 20a**.

E6.3/E7 Egindako lanaren ahozko aurkezpena, eta proiektuaren karpeta. Data: **maiatzaren 24a**.

3.4. SERIEKO BERTSIOAREN KODEA

/* File: defines.h */

```
// minimal card and maximun size
#define RSIZE 200
#define CSIZE 100
#define MAX_GRID_POINTS 3000000

#define NROW (RSIZE*param.scale + 2)  // extended row number
#define NCOL (CSIZE*param.scale + 2)  // extended column number

struct info_param {
  int    nconf, nchip, max_iter, scale; // n. config., n. chips, max. n. iter., card size scale
  float  t_ext, tmax_chip, t_delta; // external t., max. t. of a chip, t. incr. for convergence
};

struct info_chip {
  int    h, w;         // size (h, w)
  float  t_chip;       // temperatura
};

struct info_results {
  double Tmean;                     // mean temp.
  int    conf;                      // configuration number
  float  bgrid[MAX_GRID_POINTS];    // final grid
  float  cgrid[MAX_GRID_POINTS];    // initial grd (chips)
};
```

/* File: heat_s.c */

```
#include <stdio.h>
#include <values.h>
#include <sys/time.h>

#include "defines.h"
#include "faux.h"
#include "diffusion.h"

// global variables
float grid_chips[MAX_GRID_POINTS], grid[MAX_GRID_POINTS], grid_aux[MAX_GRID_POINTS];
struct info_results BT;


/*****************************************************************************/
void init_grid_chips (int conf, struct info_param param, struct info_chips *chips,
                      int **chip_coord, float *grid_chips)
{
  int i, j, n;

  for (i=0; i<NROW; i++)
  for (j=0; j<NCOL; j++)
    grid_chips[i*NCOL+j] = param.t_ext;

  for (n=0; n<param.nchip; n++)
  for (i=chip_coord[conf][2*n]*param.scale; i<(chip_coord[conf][2*n]+chips[n].h)*param.scale; i++)
  for (j=chip_coord[conf][2*n+1]*param.scale; j<(chip_coord[conf][2*n+1]+chips[n].w)*param.scale; j++)
    grid_chips[(i+1)*NCOL+(j+1)] = chips[n].tchip;
}


/*****************************************************************************/
void init_grids (struct info_param param, float *grid, float *grid_aux)
{
  int i, j;

  for (i=0; i<NROW; i++)
  for (j=0; j<NCOL; j++)
    grid[i*NCOL+j] = grid_aux[i*NCOL+j] = param.t_ext;
}
```

```
/*****************************************************************************/
/*****************************************************************************/
int main (int argc, char *argv[])
{
  struct info_param param;
  struct info_chips *chips;
  int    **chip_coord;

  int    conf, i;

  struct timeval t0, t1;
  double *tej, tsim = 0.0;


 // reading initial data file
  if (argc != 2) {
    printf ("\n\nERROR: needs a card description file \n\n");
    exit (-1);
  }

  read_data (argv[1], &param, &chips, &chip_coord);

  printf ("\n  ======================================================================");
  printf ("\n    Thermal diffusion - SERIAL version ");
  printf ("\n    %d x %d points, %d chips", RSIZE*param.scale, CSIZE*param.scale, param.nchip);
  printf ("\n    T_ext = %1.1f, Tmax_chip = %1.1f, T_delta: %1.3f, Max_iter: %d", param.t_ext,
              param.tmax_chip, param.t_delta, param.max_iter);
  printf ("\n  ======================================================================\n\n");

  BT.Tmean = MAXDOUBLE;
  tej = (double *) malloc(param.nconf * sizeof(double));


  // loop to process chip configurations
  for (conf=0; conf<param.nconf; conf++)
  {
    gettimeofday (&t0, 0);

    // inintial values for grids
    init_grid_chips (conf, param, chips, chip_coord, grid_chips);
    init_grids (param, grid, grid_aux);

    // main loop: thermal injection/disipation until convergence (t_delta or max_iter)
    diffusion (param, grid, grid_chips, grid_aux);

    // processing configuration results
    gettimeofday (&t1, 0);
    tej[conf] = (t1.tv_sec - t0.tv_sec) + (t1.tv_usec - t0.tv_usec)/1e6;
    results_conf (conf, param, grid, grid_chips, &BT);
  }


  // writing best configuration results
  results (param, &BT, argv[1]);
  for (conf=0; conf<param.nconf; conf++) tsim += tej[conf];
  printf ("   > Time (serial): %1.3f s \n\n", tsim);

  free (tej);
  free (chips);
  for (i=0; i<param.nconf; i++) free (chip_coord[i]);
  free (chip_coord);
}
```

```
/* File: difussion.c */

#include "defines.h"


/******************************************************************************/
void thermal_update (struct info_param param, float *grid, float *grid_chips)
{
  int i, j;

  // heat injection at chip positions
  for (i=1; i<NROW-1; i++)
  for (j=1; j<NCOL-1; j++)
    if (grid_chips[i*NCOL+j] > grid[i*NCOL+j])
      grid[i*NCOL+j] += 0.05 * (grid_chips[i*NCOL+j] - grid[i*NCOL+j]);

  // air cooling at the middle of the card
  int a = 0.45*(NCOL-2) + 1;
  int b = 0.55*(NCOL-2) + 1;

  for (i=1; i<NROW-1; i++)
  for (j=a; j<b; j++)
      grid[i*NCOL+j] -= 0.01 * (grid[i*NCOL+j] - param.t_ext);
}


/******************************************************************************/
void diffusion (struct info_param param, float *grid, float *grid_chips, float *grid_aux)
{
  int    i, j, end, niter;
  float  T;
  double Tmean, Tmean0 = param.t_ext;

  end = 0; niter = 0;

  while (end == 0)
  {
    niter++;
    Tmean = 0.0;

    // heat injection and air cooling
    thermal_update (param, grid, grid_chips);

    // thermal diffusion
    for (i=1; i<NROW-1; i++)
    for (j=1; j<NCOL-1; j++)
    {
      T = grid[i*NCOL+j] +
          0.10 * (grid[(i+1)*NCOL+j] + grid[(i-1)*NCOL+j] + grid[i*NCOL+(j+1)] +
                  grid[i*NCOL+(j-1)] + grid[(i+1)*NCOL+j+1] + grid[(i-1)*NCOL+j+1] +
                  grid[(i+1)*NCOL+(j-1)] + grid[(i-1)*NCOL+(j-1)]
                 - 8*grid[i*NCOL+j]);

      grid_aux[i*NCOL+j] = T;
      Tmean += T;
    }

    //new values for the grid
    for (i=1; i<NROW-1; i++)
    for (j=1; j<NCOL-1; j++)
      grid[i*NCOL+j] = grid_aux[i*NCOL+j];

    // convergence every 10 iterations
    if (niter % 10 == 0)
    {
      Tmean = Tmean / ((NCOL-2)*(NROW-2));
      if ((fabs(Tmean - Tmean0) < param.t_delta) || (niter > param.max_iter))
           end = 1;
      else Tmean0 = Tmean;
    }
  } // end while
  printf ("Iter: %d\t", niter);
}
```

```
/* File: faux.c */

#include <stdio.h>
#include <values.h>
#include "defines.h"


/*****************************************************************************/
void read_data (char *file_name, struct info_param *param, struct info_chips **chips,
                        int ***chip_coord)
{
  int   i, j, h, w;
  float tchip;
  FILE  *fdin;

  fdin = fopen (file_name, "r");

  // simulation parameters (param)
  fscanf (fdin, "%d %d %f %f %f %d %d", &param->nconf, &param->nchip, &param->t_ext,
       &param->tmax_chip, &param->t_delta, &param->max_iter, &param->scale);
  if (param->scale > 12) {
    printf("\n\nERROR: maximum scale factor is 12 \n\n");
    exit (-1);
  }

  // chip sizes and temperatures
  *chips = (struct info_chips *) malloc (param->nchip * sizeof(struct info_chips));

  for (i=0; i<param->nchip; i++) {
    fscanf (fdin, "%d %d %f", &h, &w, &tchip);
    (*chips)[i].h = h;
    (*chips)[i].w = w;
    (*chips)[i].tchip = tchip;
  }

  // chip positions
  *chip_coord = (int **) malloc (param->nconf * sizeof(int*));
  for (i=0; i<param->nconf; i++)
     (*chip_coord)[i] = (int*) malloc (2 * param->nchip * sizeof(int));

  for (i=0; i<param->nconf; i++)
  for (j=0; j<param->nchip; j++)
    fscanf (fdin, "%d %d", &(*chip_coord)[i][2*j], &(*chip_coord)[i][2*j+1]);

  fclose (fdin);
}


/*****************************************************************************/
void results_conf (int conf, struct info_param param, float *grid, float *grid_chips,
                        struct info_results *BT)
{
  int    i, j;
  float  Tmax = MINFLOAT, Tmin = MAXFLOAT;
  double Tmean = 0.0;

  for (i=1; i<NROW-1; i++)
  for (j=1; j<NCOL-1; j++)
    Tmean += grid[i*NCOL+j];

  Tmean = Tmean / ((NROW-2)*(NCOL-2));

  if (BT->Tmean > Tmean)
  {
    BT->Tmean = Tmean;
    BT->conf = conf;
    for (i=1; i<NROW-1; i++)
    for (j=1; j<NCOL-1; j++) {
      BT->bgrid[i*NCOL+j] = grid[i*NCOL+j];
      BT->cgrid[i*NCOL+j] = grid_chips[i*NCOL+j];
    }
  }
  printf ("Config: %2d \t Tmean: %1.2f\n", conf+1, Tmean);
}
```

```
/*****************************************************************************/
void fprint_grid (FILE *fd, float *grid, struct info_param param)
{
  int i, j;

  // j - i order for better visualitation
  for (j=NCOL-2; j>0; j--)
  {
    for (i=1; i<NROW-1; i++) fprintf (fd, "%1.2f ", grid[i*NCOL+j]);
    fprintf (fd, "\n");
  }
  fprintf (fd, "\n");
}


/*****************************************************************************/
void results (struct info_param param, struct info_results *BT, char *finput)
{
  FILE  *fd;
  char  name[100];

  printf ("\n\n >>> BEST CONFIGURATION: %2d\t Tmean: %1.2f\n\n", BT->conf+1, BT->Tmean);

  sprintf (name, "%s_ser.res", finput);
  fd = fopen (name, "w");
  fprintf (fd, "Tmin_ini %1.1f  Tmax_ini %1.1f  \n", param.t_ext, param.tmax_chip);
  fprintf (fd, "%d\t  %d \n", NCOL-2, NROW-2);

  fprint_grid (fd, BT->bgrid, param);

  fprintf (fd, "\n\n >>> BEST CONFIGURATION: %d\t Tmean: %1.2f\n\n", BT->conf+1, BT->Tmean);
  fclose (fd);

  sprintf (name, "%s_ser.chips", finput);
  fd = fopen (name, "w");
  fprintf (fd, "Tmin_chip %1.1f  Tmax_chip %1.1f  \n", param.t_ext, param.tmax_chip);
  fprintf (fd, "%d\t  %d \n", NCOL-2, NROW-2);

  fprint_grid (fd, BT->cgrid, param);

  fclose (fd);
}
```

ERANSKINA: Agiriak eta dokumentuak

Eranskin honetan proiektuan zehar ikasleek bete behar dituzten dokumentuak bildu ditugu: taldea osatzeko agiria eta konpromisoen dokumentua, lan-saioetako agiriak, eskoletatik kanpoko lan-denboren balioespena, eta inkestak. Eta horiekin batera, ahozko aurkezpenak ebaluatzeko errubrika eta agiria, emaitzen txosten teknikoak idazteko ildoak eta proiektuaren karpetaren egiturari eta edukiari buruzko iradokizunak.

E1. TALDEA OSATZEKO AGIRIA - KONPROMISOEN DOKUMENTUA

▪ Partaideak

	Izena eta abizenak	Sinadura
1.	_____	.....
2.	_____	.....
3.	_____	_____

▪ Konpromisoak

- Taldeak antolatutako bileretara joatea, eskola orduetan (irakaslearen aurreko saioetan), zein eskola orduetatik kanpo.
- Taldekide bakoitzari esleitutako lanak egitea zehaztutako epean.
- Bileretara prest eramatea nork bere gain hartutako lanak.
- Ziurtatzea taldekide guztiek ulertu dutela egindako lana osotasunean.
- Taldearen funtzionamendu ona lortzeko ahalegintzea.
- Gatazkarik sortuz gero, irekitasunez eta begirunez eztabaidatzea, irtenbidea aurkitzeko xedez.
- Taldearen funtzionamendu onerako adostutako betebeharrak bete ezean, balizko ondorioak onartzea: talde aldaketa, taldetik kanpo uztea …

_____ .....

Data Irakaslea

E2. TALDEAREN LAN-SAIOKO AGIRIA

▪ Bertaratuak

	Izena eta abizenak	Sinadura
1.		
2.		
3.		

▪ Jorratutako gaiak eta hartutako erabakiak

1.

▪ Hurrengo bilerarako gaiak eta atazen banaketa

1.

Data

Hasiera-ordua — Bukaera-ordua

E3. PROIEKTUAN EGINDAKO ESKOLETATIK KANPOKO LAN-DENBORAREN BALIOESPENA

(15 minutuko tarteak; adibidez: 2 h 15 m, 3 h 30 m, 45 m, eta abar.)

Jarduera	**Denbora**
Hasierako lanak	
MPI ikastea	
hasierakoa	
Puzzlea	
informazioa bilatzea eta irakurtzea	
ariketak ebaztea	
ikasitakoa taldean partekatzea	
aurkezpena prestatzea	
puzzlea	
Aplikazioa	
informazioa bilatzea eta irakurtzea	
garapena: diseinua, programazioa, emaitzen/denboren analisia	
txosten teknikoa idaztea	
aurkezpena prestatzea	
aplikazioa	
Azterketa	
azterketa prestatzea	
azterketa	
Proiektua	

Eskerrik asko zure lankidetzagatik

E4. AHOZKO AURKEZPENAK EBALUATZEKO ERRUBRIKA

Ebaluatzeko atala	**Bikain** [10 - 9]	**Ongi** [8 - 7]	**Hobetzeko** [6 - 4]	**Eskasa** [3 - 0]
Edukia Zuzena, egokia, nahikoa	Edukia zuzena eta egokia izan da. Oso ongi menperatzen du gaia.	Edukietan akatsen bat izan da, edo ez da nahikoa izan zenbait arlotan. Dena den, ematen du ongi menperatzen duela gaia.	Akats eta hutsune dezente izan dira. Garrantzi gehiegi eman zaio bigarren mailako informazioari. Gaiaren alderdi batzuk soilik ulertu ditu.	Edukiak okerrak eta oso eskasak izan dira. Ez du gaia menperatzen.
Antolaketa Argitasuna, logika, egituraketa, arrazoiketa.	Aurkezpena argia, logikoa eta ongi egituratua izan da. Entzuleek ongi jarraitu diete arrazoibideei.	Oro har, aurkezpena argia izan da eta ongi egituratua. Hala ere, alderdi batzuk ilun geratu dira.	Alderdi batzuk nahasi geratu dira. Kontzeptutik kontzeptura ibili da ordenarik gabe. Diskurtsoaren logikari jarraitzea zaila izan da.	Aurkezpena ez da batere argia izan, ezta logikoa ere. Ez du izan egiturarik. Ezin izan da ia ezer ulertu.
Estiloa Ideien azalpena, erritmoa, jarrera, bolumena, tonua, isiluneak/etenak	Aurkezpena egokia izan da entzuleentzat, eta erritmoa egokia. Une jakinetan izan ezik, ez du haren notak irakurri, eroso egon da publikoaren aurrean, eta entzuleek ongi entzun diote. Etenak egin ditu une egokietan.	Oro har, aurkezpena egokia izan da entzuleentzat. Erritmoa aldakorra izan da. Batzuetan, haren notetara jo behar izan du. Ez da eroso sentitu eta entzuleek arazoren bat izan dute ongi entzuteko. Eten gutxi egin ditu.	Alderdi batzuk agerikoak, eta beste batzuk oso maila altukoak. Erritmoa ez da egokia izan, oso azkarra edo oso motela. Askotan irakurri ditu haren notak, deseroso agertu da, eta arazoak izan dira ongi entzuteko. Oso eten gutxi egin ditu.	Aurkezpena guztiz desegokia izan da entzuleentzat. Ez da erritmorik izan. Notak irakurri ditu uneoro, oso deseroso egon da, eta entzuleek adi-adi egon behar izan dute ongi entzuteko. Ez du etenik egin.
Hizkuntza, hiztegia Tekniko-zientifikoa, zuzena.	Hiztegia zuzena eta egokia izan da.	Hiztegia nahiko zuzena eta egokia izan da.	Hiztegia desegokia izan da.	Hiztegia guztiz desegokia izan da.
Ikus-entzunezko baliabideak Homogeneotasuna, letraren tamaina, argitasuna.	Ikus-entzunezko materiala homogeneo eta kalitatezkoa izan da: letraren tamaina egokia, Informazioa ongi egituratua, alderdi nagusiak azpimarratuta. Gaia ulertzen lagundu du.	Erabilitako materiala ona izan da: letraren tamaina egokia, eta Informazioa ongi egituratua aurkeztu da. Hala ere, zenbait alderdi ez dira garbi ageri.	Aurkezpena nahasia eta iluna geratu da. Letraren tamaina txikiegia izan da, eta informazio gehiegi sartu da gardenkietan. Zaila jarraitzeko.	Aurkezpenerako prestatutako materialak oso desegokiak izan dira.
Denbora Esleitutako denbora erabili da, laburbiltzeko ahalmena	Denboraren banaketa orekatua izan da, edukien araberakoa. Ondo egokitu da esleitutako denborara.	Denboraren banaketa zertxobait desorekatua izan da: denbora soberan geratu da, edo azkar joan behar izan du bukaeran eduki guztiak azaltzeko.	Denboraren banaketa desorekatua. Kontzeptuak askoz lasaiago eta sakonago azaldu zitezkeen, edo bigarren mailako alderdi batzuk alboratu nagusiak lasaiago azaltzeko.	Denboraren banaketa oso desegokia (motzegia edo luzeegia). Ez da gauza izan dagokion denboran kontzeptuak xehetasunez azaltzeko, edo nagusietan zentratzeko.

E5. AHOZKO AURKEZPENAK EBALUATZEKO AGIRIA

Talde bakoitzaren ahozko aurkezpenaren kalifikazioa, atalez atal eta 0tik 10era, dagokion errubrikaren arabera:

10 - 9 [bikain] **8 - 7** [ongi] **6 - 4** [hobetzeko] **3 - 0** [eskasa]

Azken kalifikazioa lortzeko, ebaluatutako atalen pisua kontuan hartu behar da.

▪ **Ebaluatu duen taldea:**

	%	**Taldea:**	**Taldea:**	**Taldea:**	**Taldea:**	**Taldea:**	**Taldea:**	**Taldea:**
Edukiak	30							
Antolaketa	25							
Estiloa	15							
Hizkuntza	10							
Baliabideak	15							
Denbora	5							
Azken kalifikazioa (ez bete)								

Data:

Sinadurak:

_____ Ebaluatzailea

_____ Ebaluatzailea

_____ Ebaluatzailea

E6. PROIEKTUAREN KARPETAREN EGITURARI ETA EDUKIARI BURUZKO IRADOKIZUNAK

Karpetaren edukia ondo antolatuta mantendu behar da, ezin da dokumentu multzo soila izan. Zaindu edukiaren egitura eta formatua, dokumentu guztiek itxura homogeneoa izan dezaten. Horrez gain, jasotzen den informazioa beren testuinguruan kokatu behar da (adibidez, taula bat sartzen bada, azaldu edo analizatu, motz bada ere, taulak adierazten duen informazioa).

Proiektuaren karpetaren edukia ez da estatikoa, dinamikoa baizik, hau da, hobetu daiteke. Adibidez, eskolan landu eta gero, puzzleko ariketak zuzenduta gehitu daitezke. Horrela, proiektuaren bukaeran, karpetaren edukia zuzena eta osoa izango da. Hori dela eta, karpetaren bertsio berrirako, komenigarria da adieraztea, orri batean, egin diren hobekuntzak.

Karpetak bi formatu izango ditu: paperean eta digitala. Agiriak izan ezik, gainerako dokumentuak, paperaz gain, bertsio digitalean ere aurkeztuko dira; esaterako, programen iturburu-kodeak, aurkezpenetan erabilitako gardenkiak, aplikazioaren txosten teknikoa eta abar.

Karpetaren lehen bertsioa (paperean) puzzlea aurkezteko egunean entregatuko da, irakasleak errebisa dezan. Azken bertsioa maiatzaren 25ean entregatuko da, proiektuaren aurkezpenarekin betera: paperezko bertsioa, irakasleari; eta digitala, eGelara igoz.

Erreferentzia gisa, honela antola daiteke proiektuaren karpeta (aukera bat baino ez da):

- Azala: proiektuaren izenburua eta taldearen identifikazioa (kideak, lan-kontua…).
- Karpetaren edukiaren aurkibidea.
- Proiektuaren enuntziatua eta kontsultatu den informazioa (informazioa bera edo erabilitako erreferentziak).
- Proiektuan zehar egin diren jarduerak:
  - Hasierako posterra.
  - Puzzlea: (a) Gaien aurkezpena eta aurkitu diren soluziobideen azalpenak; (b) Ariketak ebatzita: proposatutako ebazpenaren azalpena, programen iturburu-kodea komentatuta, eta lortu diren emaitzen analisia.
  - Aplikazioa. Txosten teknikoa, berau idazteko eman diren irizpideen arabera.
- 1. eranskina. Taldea osatzeko agiria eta taldearen lan-saioko agiriak.
- 2. eranskina. Proiektua garatzeko taldekide bakoitzak eskolaz kanpo dedikatutako ordu kopurua (banatu den txantiloian).

E7. EMAITZEN TXOSTEN TEKNIKOAK IDAZTEKO ILDOAK

Proiektuan egindako lana **txosten tekniko batean** laburbildu behar da, non azaldu behar baitira ebatzitzako problemak, proposatutako ebazpideak, lorturiko emaitzak, ondorioak, eta abar. Txostena **proiektuaren azken emaitza** eta egindako lanaren isla da, eta, beraz, **merezi du arretaz egitea**, edukiaz zein formaz egokia eta kalitatekoa izan dadin. Lan bikainena (edukia), bikain ere azaldu behar da (itxura).

Adibidez, hauxe izan daiteke txostenaren **egitura** (egokitu zure kasurako):

- Aurkibidea
- Sarrera. Proiektuaren helburuak, eta edukien eta emaitzen nondik norakoak. Erabilitako tresnen ezau-garri nagusiak —hardwarea (arkitektura, prozesadoreak, memoria, erloju-maiztasuna...) eta softwarea (S.E., konpiladoreak, optimizazio-mailak, bertsioak, software-tresnak…), emaitzak testuinguru jakin batean kokatu, birsortu, beste batzuekin konparatu… ahal izateko.
- Oinarri teorikoak. Txosten motaren arabera, egindako lanaren oinarri teorikoen laburpena.
- Aplikazioa. Txostenaren atal nagusia da, eta azaldu behar dira: (a) ebatzi den problema; (b) hartutako ebazpideak (baztertutakoak), eta kodearen zati garrantzitsuenak, ondo komentatuta; (c) lortutako emaitzak, azalpenak, justifikazioak; (d) egokia bada, beste aukera batzuk, hobekuntzak, egiteke gelditu diren kontuak…
- Ondorio orokorrak, bibliografia eta eranskinak.

Iradokizun hauek lagungarri izan ditzakezu txosten hauetan ohikoak diren akats batzuk ekiditeko.

> Azalpenak

- Lortu dituzun emaitzen azalpenek argiak eta zehatzak izan behar dira; ez da nahikoa zer gertatzen den adieraztea ("kurbak gora egiten du, eta gero behera"), zergatik gertatzen den hori baizik. Portaera jakin baterako azalpenik ez baduzu, hala adierazi eta saiatu hipotesi bat ematen. Maiz, hipotesi batek esperimentu gehiago egitea eskatzen du, hipotesi horren araberako emaitzak eta benetan lortzen direnak alderatzeko. Aurreikusitakoa baino harantzago doazen hipotesiak eta probak egitea —sortzaile izatea— ohitura gomendagarria eta baloratua da, nahiz eta denbora ere kontuan hartu behar duzun.
- Proiektua garatzeko dokumentazioa erabili baduzu, aipatu behar duzu, beti. Erreferentzia guztiek ez dute "kalitate edo balio" tekniko/zientifiko bera; erabili beti informazio-iturburu egokiak. Dena den, ez kopiatu/itzuli testuak zuzenean; berridatzi edukiak zure hitzekin, ulertu eta asimilatu ondoren.

> Datuak

- Esperimentuak errepikatu eta exekuzio-denborak esanguratsuki desberdinak badira, batez besteko balioaz gain, maximoa, minimoa eta desbideratze estandarra ere adierazi behar dira. Lau balioek esperimentuari buruzko informazio osagarria eskaini dezakete hainbat kasutan. Begizta bat gehitu diezaiokezu programari *n* aldiz exekutatzeko eta exekuzio bakoitzeko denborak eta estatistikak lortzeko. Ohikoa da lehenbiziko exekuzioaren debora baztertzea, baita nabarmenki desbideratzen direnak ere (jakina, testuinguruan azaldu daitezkeenak izan ezik).
- Grafikoetan eta datu-tauletan, neurrien unitateak adierazi behar dira (ms, byte, MB/s...)

> Datuen adierazpena

- Askotan, emaitzak adierazten dituen taula bat nahikoa da portaera jakin bat azaltzeko. Erabil itzazu grafikoak baldin eta emaitzak interpretatzeko lagungarriak badira.
- Datuen adierazpen grafikoa erabilgarria da, baldin eta ahalbidetzen badu sistemaren portaera begi kolpe batez antzematea, edo arreta atal jakinetan fokatzea. Hori dela eta, X eta Y ardatzen eskalak egoki aukeratu behar dira, irudikatzen den funtzioaren eremu osoa ongi bistaratzeko. Eskuarki, eskala lineala edo logaritmikoa izan daiteke. Ardatzak ondo etiketatu behar dira, eta neurriak eta unitateak adierazi behar dira.
- Kasu gehienetan, X ardatzak eskala jakin bati jarraitu behar dio, marraztutakoa zuzen interpretatu ahal izateko. Esaterako, programa baten exekuzio-denborak ezin dira X ardatzean homogeneoki banatu, baldin eta bektoreen tamainak 100, 200, 500, 600 eta 4000 osoak badira.
- Irudiek eta taulek oin bana eraman behar dute, honako hau bezalakoa: "3. irudia/taula. Pr1 programaren exekuzio-denbora (ms) matrizeen tamainaren arabera (double)".

> Kodea

- Askotan, kode-zatiak gehitu behar dira txostenean. Hala bada, gehitu soilik beharrezkoa dena egindakoa interpretatzeko, eta komentatu kodea. Koderako, erabil itzazu pauso konstanteko eta tamaina txikiagoko letra-mota bat —esaterako, `courier` 8— eta lerroarte bakuna. Indexatu kodea irakurgarritasuna errazteko.

  Dena den, sartu eranskin batean kode osoa (oso luzea ez bada). Kodea makina jakin batean geratu bada, adierazi non dauden fitxategiak (kontua, direktorioa) eta azaldu, labur, fitxategi bakoitza.

> Estiloa, formatua

- Txosten tekniko/zientifikoen edukia konplexua izan daiteke. Aukeran, beraz, erredakzioa ahalik eta sinpleena izan beharko litzateke —esaldi ez oso luzeak, errazak, irakurgarriak—, baina, jakina, konplexutasuna azaltzeari uko egin gabe. Testua nahasia, astuna, korapilatsua eta ulergaitza bada, ez du bere edukia "transmititzen". Aukera baduzu, utzi testua denbora-tarte batez eta irakurri gero berriz; zu zeu konturatuko zara testuaren kalitateaz. Testua idatzi ez duen beste batek irakurtzea ere oso ona da; testuak zer esan behar duen aurretik ez dakienez, benetan esaten duena (eta nola) baino ez du irakurriko. Gogoratu "urrezko araua": txostena beste pertsona batentzat idatzi duzu, ez zuretzat.
- Gaurko edizio-tresnek kalitate formal handiko testuak sortzeko aukera eskaintzen dute: maketazioa egokia, kalitateko grafikoak, ortografia-faltarik gabekoa, eta abar. "0 akats"-eko testuak sortzea zaila da, baina hurbil ibiltzea, ez.

  Zure gustuko formatua erabili, baina sinplea. Adibidez: "times", "calibri"… edo LaTeX-en letra-mota (testu-prozesadorearen arabera); 10/11 puntuko tamaina testurako, zertxobait handiago (12/14 puntu) goiburuetarako, eta txikixeago (8/9) taula- eta irudi-oinetarako; lerroarte bakuna; 2,5/3 cm-ko marjinak; bi aldeetatik inprimatua.
- Zenbakitu orrialdeak eta, dokumentua luzea bada, aurkibide bat gehitu. Dokumentuaren kalitatea ez da orri kopuruaren arabera neurtzen, baina eskasia ere ez da bidea. Koloreak aipatzen badituzu testuan (esaterako, grafiko baten portaera azaltzean), ez ahaztu gutxienez zati hori kolorez inprimatzea. Azkenik, grapa batzuk, karpeta bat edo koadernatze sinple bat orri solteak baino egokiagoak dira.

Hitz gutxitan: **emaitzen txosten teknikoak zehatza, osoa, argia eta irakurgarria izan behar du**, gaiari buruz ezagutza minimoa duen edonork (zure ikaskideek, kasu) erraz irakurtzeko eta ulertzeko modukoa.

E8. PROIEKTUARI BURUZKO INKESTA (ERAGIN)

JARRAITUTAKO METODOLOGIARI BURUZKO IRITZI INKESTA

Irakasgaian jarraitu diren zenbait metodologia-aspektuei buruz duzun iritzia emateko eskatzen dizugu. Zure erantzunak aztertuak izango dira, eta etorkizunean irakasgaia hobetzen lagunduko digu. Horregatik, behar duzun denbora eskainiz, pentsatzen duzuna zintzoki adierazteko eskatzen dizugu. Eskerrik asko.

Landu diren metodologiaren aspektu guztiak kontutan hartuz, izandako esperientziaren planteamendu eta garapenari buruzko zure balorazio orokorra da:

□ batere pozik □ apur bat pozik □ nahiko pozik □ oso pozik

Justifikatu zure balorazioa:

Metodologia tradizionalagoekin konparatuz, balora ezazu zein neurritan uste duzun jarraitutako metodologiak lagundu zaituela ikasten:

□ gutxiago □ berdin □ gehiago □ askoz gehiago

Balora ezazu zein neurritan uste duzun jarraitutako metodologiak lagundu zaituela:

("1" oso gutxi, "2" gutxi, "3" nahiko, "4" asko)

Eduki teorikoak ulertzen	1	2	3	4
Teoria eta praktika artean erlazioak egiten	1	2	3	4
Irakasgaiaren edukiak elkar erlazionatzen ikuspegi integratu bat lortuz	1	2	3	4
Irakasgaiarekiko interesa eta motibazioa handitzen	1	2	3	4
Praktika profesionaleko egoerak aztertzen	1	2	3	4
Lanaren inguruan zure aldetik informazioa aurkitzen	1	2	3	4
Egoera erreal baten inguruan erabakiak hartzen	1	2	3	4
Arazoak ebazten edo egoera errealei soluzioak eskaintzen	1	2	3	4
Komunikazio gaitasunak garatzen (ahozkoa edo idatzizkoa)	1	2	3	4
Ikasteko autonomia garatzen	1	2	3	4
Zure ikasketarekiko jarrera parte-hartzailea hartzen	1	2	3	4
Zure talde lanerako gaitasunak garatzen	1	2	3	4
Praktika profesionalean behar diren gaitasunak garatzen	1	2	3	4
Ebaluazio-sistema egokia izan da metodologiarekiko	1	2	3	4

Irakasleak eman dizun orientazioak zure beharrak ase egin al ditu?

□ Gutxi □ Nahiko □ Asko □ Erabat

Aldatuko al zenuke zerbait? Hobetzeko proposamenen bat bururatzen al zaizu?

Hurrengo irakasgairen batean aukeran izango bazenu, metodologia hau aukeratuko al zenuke?

□ Bai □ Ez