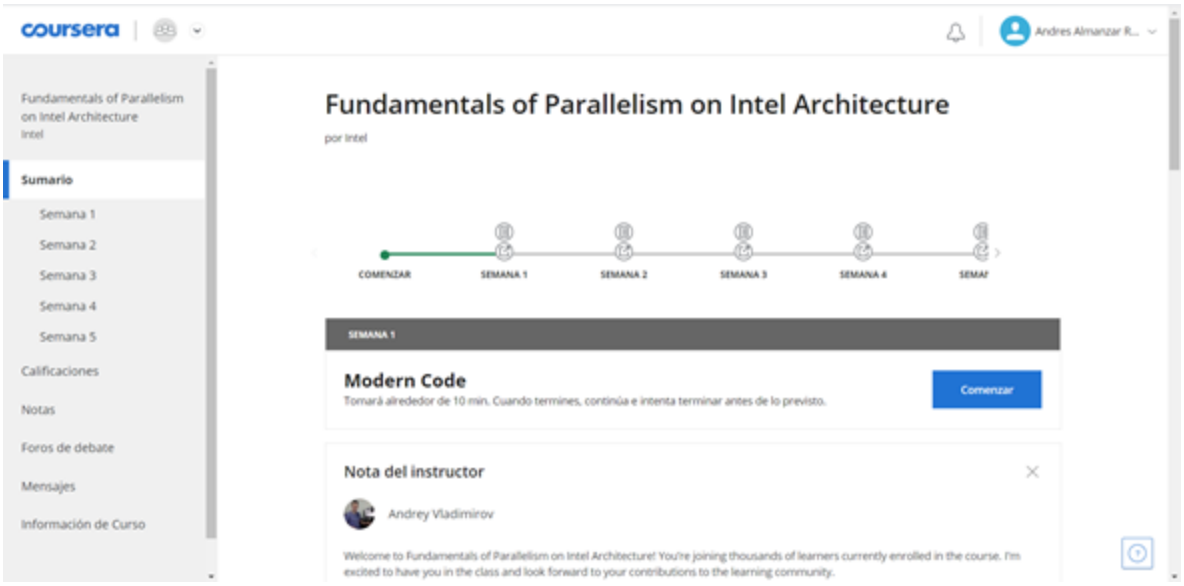
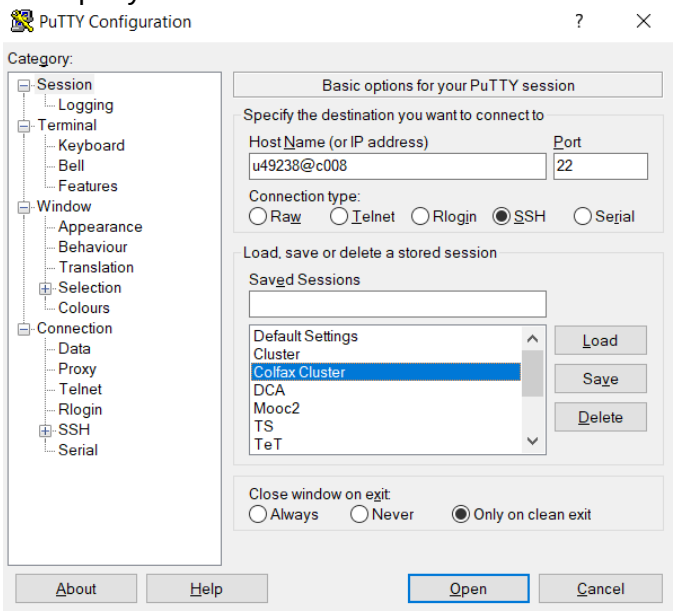
Desarrollo de Mooc2 – Andres Almanzar Restrepo

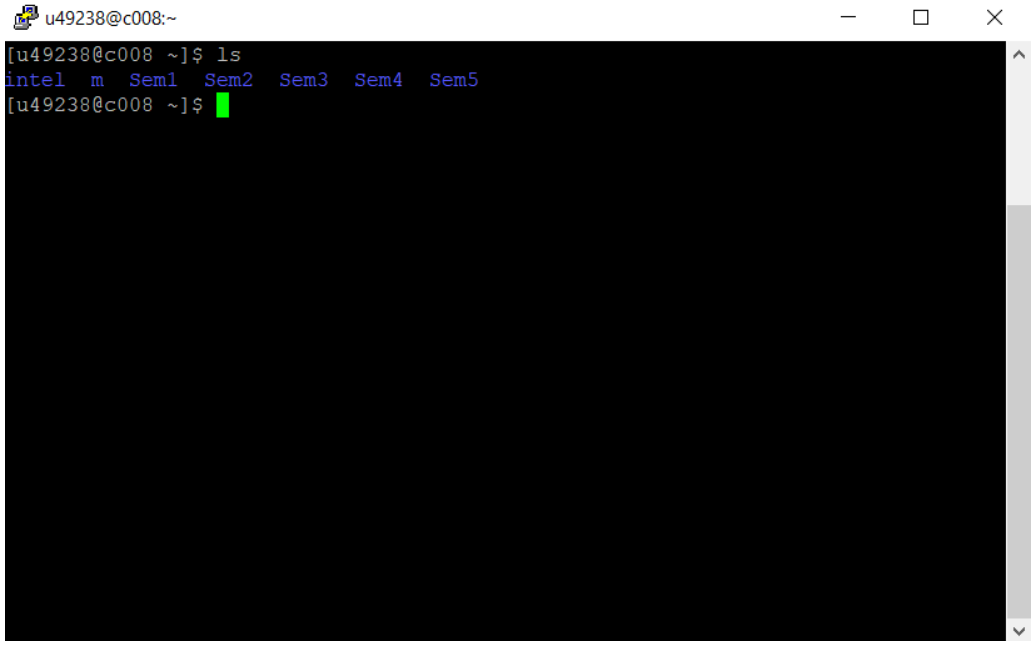


Inscripción al curso “Fundamentals of Parallelism on Intel Architecture” en coursera mediante vinculación con la universidad EAFIT

Adjunto en la carpeta del curso hay una carpeta llamada Mooc2, en la que se encuentra la llave para acceder al cluster, así como también se encuentra en el siguiente repositorio https://github.com/aalmanzarr/Mooc2TET

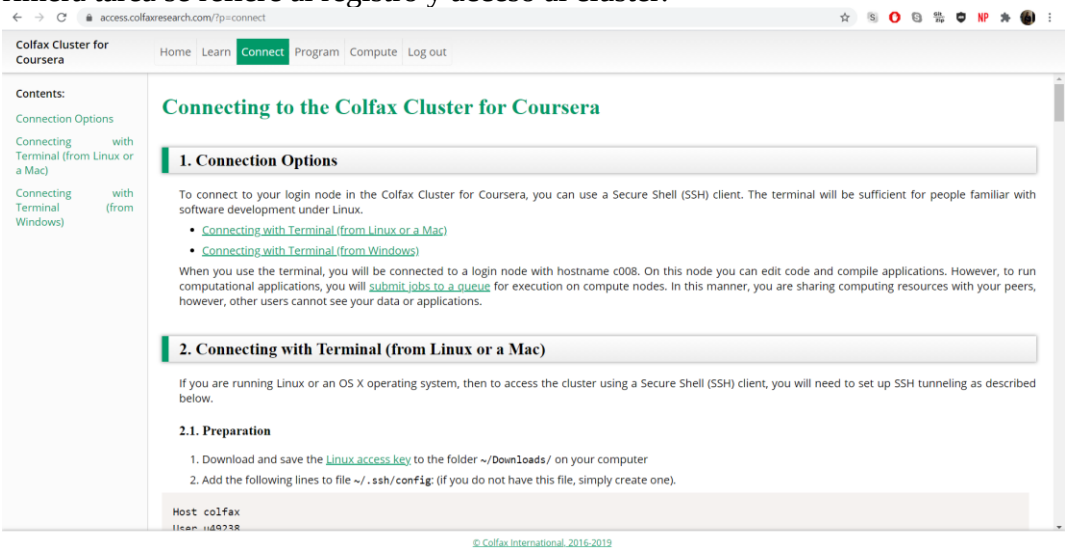
Una vez registrado en el curso y teniendo la llave para acceder al cluster, procedi a acceder a este mediante putty





Todo el manejo de datos hacia el cluster y de este a mi maquina local los realice mediante SCP
Una vez habiendo podido acceder al cluster procedi a continuar con el curso, sus quices y tareas calificables. A continuación, hablo de cada uno semana a semana.

Sem1
Para la primera semana se deben realizar dos tareas, una de ellas calificable y la otra no, además del quiz.
La primera tarea se refiere al registro y acceso al cluster.



En esta pagina se encuentran las instrucciones para conectarse al cluster, una vez conectado y habiendo leyendo los términos de uso se puede validar la terea como hecha.

Términos de uso
https://access.colfaxresearch.com/doc/Colfax_Cluster_Service_Terms.pdf

Tarea no calificable terminada

Last transaction TID: 26703
Your grade has been sent to the course platform.

This lab allows 1 submission in 30 days. You have exceeded this amount. You will be able to submit again in 24 days.

TID	Grade	Feedback	Submitted file	Submission time
26703	100 *	Your request for access is processing. In a few minutes, visit the access portal to view connection and usage instructions. If after 10 minutes you are still getting the 'No Access' error, please send us a message and indicate your request ID: 49238	none	2020/09/21 13:59:54

Legend:

* - your best grade for this lab so far

x - not yet submitted to the course platform. Grade transfer may take up to 5 minutes. To re-check the status, reresh this page

Habiendo realizado esta tarea y ya conectado al cluster procedí con el curso para finalmente llegar al quiz y a la tarea calificable de la Semana1

Quiz Sem1

1. Intel Xeon Phi processors are a... 1 punto

- ...computing platform for legacy applications
- ...more efficient computing platform than traditional CPUs for applications that have good thread and vector parallelism
- ...specialized family of processors for applications requiring high clock speeds

2. Multiple cores can share data in system memory 1 punto

- True
- False

3. In vector units, you can apply: 1 punto

- A single stream of instructions to multiple data elements in a short vector
- Multiple streams of instructions to multiple data elements in a vector
- A single stream of instructions to a single data element in a vector

4. There are no mainstream processors with clock speeds above 4 GHz. What is the reason for that? (pick all answers that apply) 1 punto

- [x] Adding parallel processing capabilities to CPUs gives better performance per watt than increasing clock speeds
- [x] Increasing the clock speed would increase the power requirement, which make the required cooling solutions expensive and impractical
- [ ] Clock speed above 4GHz would result in slower computational speed

5. Which of the following commands compiles a C++ application contained in a single file "code.cc" into an executable "myApp" using the Intel C++ compiler? 1 punto

- icpc code.cc -o myApp
- gcc code.cc -o myApp
- icpc code.cc myApp

Modern code

Envía tu tarea
FECHA DE ENTREGA 27 de sep. 23:59 PDT **INTENTOS** 3 cada 8 hours

Probar de nuevo

Recibe la calificación
PARA APROBAR 80 % o más

Calificación
100 %

Ver comentarios

Conservaremos tu puntaje más alto.

Tarea Sem1

Para esta tarea era necesario modificar el archivo Makefile para que este hiciese la compilación del main.cc tanto con el compilador GNU como con el compilador de Intel

```
TARGET=main.cc

default: app-icc app-gcc

app-icc : ${TARGET}
	# Insert Intel compiler compilation here

app-gcc : ${TARGET}
	g++ -o "$@" "$<"

clean :
	rm app-icc app-gcc
```

Archivo descargado del curso

```
TARGET=main.cc

default: app-icc app-gcc

app-icc : ${TARGET}
	# Insert Intel compiler compilation here
	icpc -o "$@" "$<"

app-gcc : ${TARGET}
	g++ -o "$@" "$<"

clean :
	rm app-icc app-gcc
```

Archivo editado para la tarea

Al ejecutar el compilador

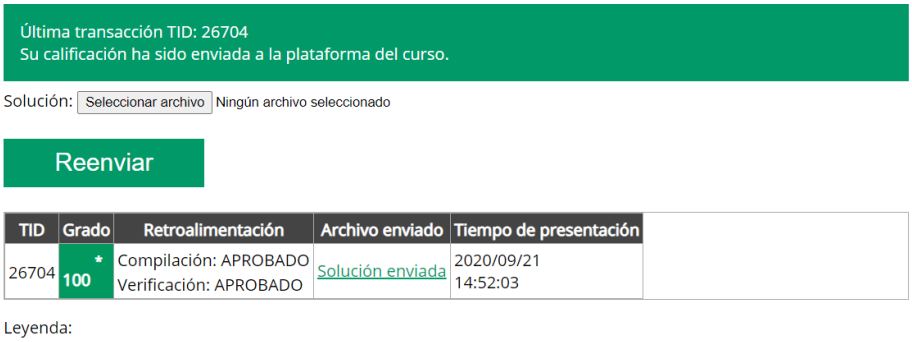
```
[u49238@c008 Sem1]$ make -B
# Insert Intel compiler compilation here
g++ -o "app-gcc" "main.cc"
```

Sin editar el archivo

```
[u49238@c008 Solved]$ make -B
# Insert Intel compiler compilation here
icpc -o "app-icc" "main.cc"
g++ -o "app-gcc" "main.cc"
```

Habiendo editado

Finalmente, después de haber logrado esto procedí a subir el archivo al curso para recibir la calificación



Sem2

La segunda semana del curso consta de un quiz y de una tarea calificable sobre vectorización.

Quiz Sem2

1. Which of the two loops is safe to be vectorized with AVX-512 vector instructions acting on 512-bit vector registers? By "safe" we mean that the vectorized loop will produce the same results as the scalar loop. Here A is an array of type "double". 1 punto

Code A:

```
1  for (int i = 0; i < n-1; i++)
2    A[i] += A[i+1];
```

Code B:

```
1  for (int i = 4; i < n; i++)
2    A[i] += A[i-4];
```

- ○ Code B
- ○ Both
- ○ Neither
- ◉ Code A

2. Which of the following cannot be automatically vectorized by the compiler? 1 punto

- ◉ While loop
- ○ For loop

3. What compiler argument do you have to use with the Intel C++ compiler to produce an optimization report in Linux? 1 punto

- ○ /Qopt-report
- ○ -opt-report
- ◉ -qopt-report

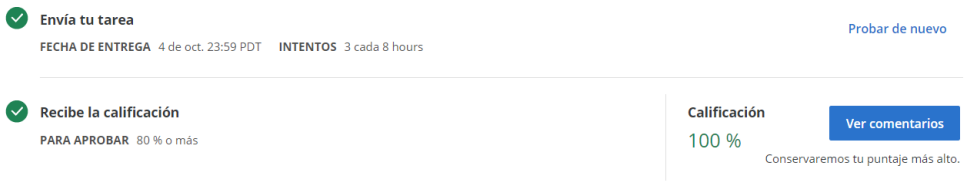
4. In order to target multiple architectures, which compiler argument to use?

- ○ -x[code]
- ◉ -ax[code]
- ○ -z[code]

5. Which compiler argument should be used to target the architecture on which the code is compiled?

- ○ -axMIC-AVX512
- ○ -xAVX
- ◉ -xhost
- ○ -axAVX

Vectorization



Tarea Sem2

Para esta semana era necesario modificar un código para poder paralelizarlo de la mejor manera y que se ejecutara en menor tiempo, para esto fue necesario modificar el archivo Makefile para ver mediante queue como se estaba ejecutando

```
CXX=icpc
CXXFLAGS=-xMIC-AVX512 -qopenmp-simd -mkl
OPTRPT=-qopt-report=5

default : app

distribution.o : distribution.cc distribution.h
	icpc -c ${OPTRPT} ${CXXFLAGS} -o "$@" "$<"

diffusion.o : diffusion.cc distribution.o
	icpc -c ${OPTRPT} ${CXXFLAGS} -o "$@" "$<" distribution.o

app : main.cc diffusion.o distribution.o
	icpc ${OPTRPT} ${CXXFLAGS} -o "$@" "$<" diffusion.o distribution.o

clean :
	rm app diffusion.o distribution.o *.optrpt
```

Makefile sin queue

```
CXX=icpc
CXXFLAGS=-xMIC-AVX512 -qopenmp-simd -mkl
OPTRPT=-qopt-report=5

default : app

distribution.o : distribution.cc distribution.h
	icpc -c ${OPTRPT} ${CXXFLAGS} -o "$@" "$<"

diffusion.o : diffusion.cc distribution.o
	icpc -c ${OPTRPT} ${CXXFLAGS} -o "$@" "$<" distribution.o

app : main.cc diffusion.o distribution.o
	icpc ${OPTRPT} ${CXXFLAGS} -o "$@" "$<" diffusion.o distribution.o

queue : default
	echo 'cd $$PBS_O_WORKDIR; ./app' | qsub -l nodes=4:flat -l walltime=00:03:00\
 -N MonteCarlo

clean :
	rm app diffusion.o distribution.o *.optrpt
```

Makefile con queue para ver cómo se estaba ejecutando el programa

Para poder reducir el tiempo de ejecución era necesario paralelizar el archivo, para esto era necesario modificar el archivo diffusion.cc distribution.cc y distribution.h, siendo el mas complejo y necesario el diffusionn.cc, a continuación, dejo una captura de pantalla los archivos ya editados

```
#include <mkl.h>
#include "distribution.h"


//vectorize this function based on instruction on the lab page
int diffusion(const int n_particles,
              const int n_steps,
              const float x_threshold,
              const float alpha,
              VSLStreamStatePtr rnStream) {
  int n_escaped=0;

  float xAcum[n_particles];
  for (int j = 0; j < n_steps; j++) {
    float rn[n_particles];
    vsRngUniform(VSL_RNG_METHOD_UNIFORM_STD,
                 rnStream, n_particles, rn, -1.0, 1.0);

    for (int i = 0; i < n_particles; i++) {
      xAcum[i] += dist_func(alpha, rn[i]);
    }
  }

  for (int i =0; i < n_particles; i++){
    if (xAcum[i] > x_threshold) n_escaped++;
  }
  return n_escaped;
}
```

```cpp
#include "distribution.h"

//distribution function definition
#pragma omp declare simd
float dist_func(const float alpha, const float rn) {
  return delta_max*sinf(alpha*rn)*expf(-rn*rn);
}
```

```cpp
#ifndef DISTRIBUTION_H
#define DISTRIBUTION_H
#include <cmath>
#include <omp.h>

const float delta_max = 1.0f;

#pragma omp declare simd

float dist_func(const float alpha, const float rn);

#endif
```

Tanto para distribution.cc como para distribution.h basto con poner una sentencia #pragma, mientras que para diffusion.cc fue necesario modificar los ciclos for, empezando por intercambiarlos y crear el buffer temporal para almacenar las posiciones

Last transaction TID: 26765
Your grade has been sent to the course platform.

Solution: Seleccionar archivo Ningún archivo seleccionado

Resubmit

TID	Grade	Feedback	Submitted file	Submission time
26765	100 *	Compilation: PASSED Verification: PASSED Vectorized dist_func(): PASSED Vectorized diffusion(): PASSED Performance: PASSED	Submitted solution	2020/09/22 17:19:08
26751	30	Compilation: PASSED Verification: PASSED Vectorized dist_func(): PASSED Vectorized diffusion(): FAILED diffusion() does not appear to be vectorized	Submitted solution	2020/09/22 15:57:31

Al principio no lograba completar la tarea, pero finalmente luego de entender que debía hacer logre que el programa se ejecutara de la manera esperada obteniendo el 100% de la calificación

Sem3

La tercera semana del curso consta de un quiz y de una tarea calificable sobre Filtrado multiproceso.

Quiz Sem3

1. OpenMP is a framework for... 1 punto

- ○ multiprocessing in distributed-memory systems (clusters)
- ◉ multithreading and vectorization in shared-memory systems

2. Which flag is required to compile your OpenMP application with Intel compiler? 1 punto

- ○ -fopenmp
- ○ -qopt-report
- ○ -xhost
- ◉ -qopenmp
- ○ -openmp

3. Based on this snippet, which of the following is correct about the number of copies of a variable in memory at runtime if we are using 4 threads? 1 punto

```
1  int A, B;
2  #pragma omp parallel private(B)
3  {
4    int C;
5    //code to be executed
6    }
```

- ◉ There is a single copy of A and there are 4 copies of both B and C
- ○ There is a single copy of A and B and there are 4 copies of C
- ○ There are 4 copies of A, B and C

4. Which of these mutexes has the highest performance penalty? 1 punto

- ◉ #pragma omp critical
- ○ #pragma omp atomic
- ○ #pragma omp parallel

5. What is the default number of OpenMP threads in an application running on an Intel Xeon Phi processor 7210 with 4-way hyper-threading? Feel free to look up the technical specifications of this processor model online

1 punto

- ○ 192
- ○ 1
- ◉ 256
- ○ 64
- ○ 128

6. Which of the following code snippets has a data race ("race condition"):

1 punto

Code A:

```
1  #pragma omp parallel for
2
3  for (int i = 0; i < n-1; i++)
4
5  A[i] += A[i+1];
```

Code B:

```
1  #pragma omp parallel for
2
3  for (int i = 1; i < n; i++)
4
5  A[i] += A[i-1];
```

- ○ Code A
- ○ Code B
- ◉ Both

7. Consider the following piece of code for computing (incorrectly?) the mean and standard deviation of values in an array: 1 punto

```
1  void ComputeStats(int n, double* x, double & mean, double & stdev) {
2    mean = 0.0;
3    stdev = 0.0;
4
5    #pragma omp parallel for
6
7    for (int i = 0; i < n; i++) {
8      mean += x[i];
9      stdev += x[i]*x[i];
10
11   }
12
13   mean /= double(n);
14   stdev = sqrt(stdev/double(n) - mean*mean);
15 }
```

What should you change in this code to make it correct and efficient?

- (●) Add a reduction clause in the parallel pragma
- ( ) Add a #pragma omp critical clause
- ( ) Add a #pragma omp atomic clause

OpenMP

Envía tu tarea — Probar de nuevo

FECHA DE ENTREGA 11 de oct. 23:59 PDT **INTENTOS** 3 cada 8 hours

Recibe la calificación

PARA APROBAR 80 % o más

Calificación

100 %

Ver comentarios

Conservaremos tu puntaje más alto.

Tarea Sem3

Para esta semana era necesario modificar el código del worker.cc para poder paralelizarlo de la mejor manera, para esto paralelice el primer ciclo y use mutex para la sección crítica y que se ejecutara en menor tiempo, si bien para esta semana no cree el queue en el Makefile es recomendable crearla para llevar mejor control del tiempo de ejecución

```cpp
#include <vector>
#include <algorithm>

//helper function , refer instructions on the lab page
void append_vec(std::vector<long> &v1, std::vector<long> &v2) {
  v1.insert(v1.end(),v2.begin(),v2.end());
}


void filter(const long n, const long m, float *data, const float threshold, std::vector<long> &result_row_ind) {

  //work on one row at a time
  for (long i = 0; i < n; i++){
    float sum = 0.0f;
    //compute sum of all the elements in a row
    for (long j = 0; j < m; j++) {
      sum += data[i*m + j];
    }

    //store the sum in an array(vector) only if it is valid (i.e if it is greater than threshold
    if (sum > threshold)
      result_row_ind.push_back(i);
  }

  //sort the values stored in the vector
  std::sort(result_row_ind.begin(),
            result_row_ind.end());
}
```

Worker.cc sin modificar

```cpp
#include <vector>
#include <algorithm>

//helper function , refer instructions on the lab page
void append_vec(std::vector<long> &v1, std::vector<long> &v2) {
  v1.insert(v1.end(),v2.begin(),v2.end());
}


void filter(const long n, const long m, float *data, const float threshold, std::vector<long> &result_row_ind) {

  //work on one row at a time
#pragma omp parallel for
  for (long i = 0; i < n; i++){
    float sum = 0.0f;
    //compute sum of all the elements in a row

    for (long j = 0; j < m; j++) {
      sum += data[i*m + j];
    }

    //store the sum in an array(vector) only if it is valid (i.e if it is greater than threshold
    if (sum > threshold) {
#pragma omp critical
{
      result_row_ind.push_back(i);
}
    }
  }

  //sort the values stored in the vector
  std::sort(result_row_ind.begin(),
            result_row_ind.end());
}
```

Worker.cc ya modificado y paralelizado

Finalmente al modificar los archivos y ver que aparentemente todo estaba paralizado según los archivos *.optrpt subí el archivo para recibir la calificación

Última transacción TID: 27016
Su calificación ha sido enviada a la plataforma del curso.

Solución: Seleccionar archivo Ningún archivo seleccionado

Reenviar

TID	Grado	Retroalimentación	Archivo enviado	Tiempo de presentación
27016	* 100	Compilación: APROBADO Verificación: APROBADO Rendimiento: APROBADO	Solución enviada	2020/09/26 12:51:36

Sem4

La cuarta semana del curso consta de un quiz y de una tarea calificable FFT en HBM.

Quiz Sem4

1. In KNL memory architecture, Cache mode for MCDRAM means: 1 punto

- (•) MCDRAM acts as a cache between L2 cache and DDR4 RAM
- ( ) MCDRAM acts as a cache between L1 cache and L2 cache
- ( ) DDR4 RAM acts as a cache between MCDRAM and L2 cache

2. Which memory has the lowest access latency in KNL memory organization? 1 punto

- (•) L1 cache
- ( ) L2 cache
- ( ) MCDRAM
- ( ) DDR4 RAM

3. What is the flat mode in KNL memory architecture? 1 punto

- (•) MCDRAM is treated as a separate NUMA node
- ( ) DDR4 is treated as LLC
- ( ) MCDRAM is treated as LLC

4. If your application is bandwidth-limited and requires less than 16 GB of memory, which is the easiest and the most efficient way of running it in high-bandwidth memory (HBM) of an Intel Xeon Phi processor? 1 punto

- (•) Run the whole program in the HBM using numactl
- ( ) Allocate bandwidth-critical data in HBM using memkind library
- ( ) Set the processor in the cache mode and run the application without modification

5. If your bandwidth-limited application requires more than 16 GB of memory, what are the options that can be used to run your application in the high-bandwidth memory of an Intel Xeon Phi processor? 1 punto

- [x] Use cache mode
- [x] Use memkind library
- [ ] Use numactl

6. How to make the Intel compiler implement streaming stores in a loop? 1 punto

- [x] -qopt-streaming-stores=always
- [ ] #pragma vector temporal
- [x] #pragma vector nontemporal

```
class AllBalls {
  // Coordinates of the centers of the balls:
  float x[1000], y[1000], z[1000];
  float R[1000];
  // Radii of the balls
};

AllBalls b;
```

Case C:

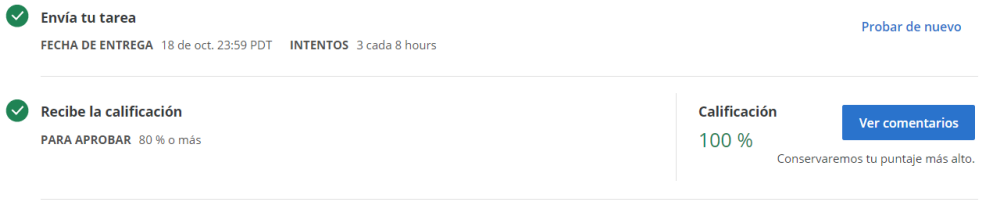
```
struct BallVector {
  float x[8], y[8], z[8];
  float R[8];
};

BallVector b[125];
```

- (•) B is the best, A is the worst
- ( ) A is the best, B is the worst
- ( ) C is the best, A is the worst

8. When an application reads and updates a single 32-bit floating-point number residing in the main memory, how much data is physically sent from the main memory to the core? 1 punto

- (•) 64 bytes
- ( ) 32 bytes
- ( ) 16 bytes

Memory traffic



Tarea Sem4

Para esta semana la tarea no era solo modificar una parte de código sino también añadirle código al programa para que se ejecutara de la mejor manera y en el menor tiempo, si bien para esta semana no cree el queue en el Makefile es recomendable crearla para llevar mejor control del tiempo de ejecución, para esto modifique el Makefile para ver mediante queue el comportamiento de la aplicación.

```
#include <mkl.h>


//implement scratch buffer on HBM and compute FFTs, refer instructions on Lab page
void runFFTs( const size_t fft_size, const size_t num_fft, MKL_Complex8 *data, DFTI_DESCRIPTOR_HANDLE *fftHandle) {
  for(size_t i = 0; i < num_fft; i++) {
    DftiComputeForward (*fftHandle, &data[i*fft_size]);
  }
}
```

Al descargar el programa, recibimos este worker.cc sobre el cual debemos trabajar

en el caso de esta aplicación, no podemos poner todos los datos de entrada en HBM porque la memoria de la aplicación requerida es mayor que el tamaño del HBM disponible (16GB). Por lo tanto, los datos de entrada deben almacenarse en la memoria del sistema DDR4 normal.

Para tamaños grandes de FFT como es el caso aquí, la diferencia de rendimiento entre el uso de HBM y el uso de memoria normal (DDR4) puede ser suficiente para que se beneficie de copiar el conjunto de datos de DDR4 a un búfer temporal en HBM, haciendo la FFT en el búfer temporal y copiando los resultados.

Se utiliza las libreria hbwmalloc, que es un contenedor de nivel superior alrededor de la biblioteca Memkind.

```
#include <mkl.h>
#include <hbwmalloc.h>
#include <iostream>
//implement scratch buffer on HBM and compute FFTs, refer instructions on Lab page
void runFFTs( const size_t fft_size, const size_t num_fft, MKL_Complex8 *data, DFTI_DESCRIPTOR_HANDLE *fftHandle) {

MKL_Complex8 *buff;
  hbw_posix_memalign((void**) &buff, 4096,
                     sizeof(MKL_Complex8)*fft_size);
  for(size_t i = 0; i < num_fft; i++) {
#pragma omp parallel for
    for(int j = 0; j < fft_size; j++){
      buff[j] = data[i*fft_size + j];
    }
    DftiComputeForward (*fftHandle, &buff[0]);
#pragma omp parallel for
    for(int j = 0; j < fft_size; j++){
      data[i*fft_size + j] = buff[j];
    }

  }
}
```

Código modificado, añadiéndole la librería hbwmalloc y buffer, además una copia multiproceso para obtener el máximo ancho de banda.

Última transacción TID: 27018
Su calificación ha sido enviada a la plataforma del curso.

Solución: Seleccionar archivo Ningún archivo seleccionado

Reenviar

TID	Grado	Retroalimentación	Archivo enviado	Tiempo de presentación
27018	* 100	Compilación: APROBADO Verificación: APROBADO Rendimiento: APROBADO	Solución enviada	2020/09/26 14:27:32

Sem5

La quinta semana del curso consta de un quiz y de una tarea calificable sobre MPI.

Quiz Sem5

1. In distributed-memory systems, the _____ framework can be used for communicating data between processes. 1 punto

- ○ OpenMP
- ○ Ethernet
- ○ Vectorization
- ◉ MPI

2. The function MPI_Comm_size returns the _____ 1 punto

- ◉ number of processes in a given communicator
- ○ number of communicators in the applications
- ○ number of cores in the processor
- ○ size of the last communicated message

3. The MPI function that sends data from a group of processes to one process is____ 1 punto

- (•) MPI_Gather
- ( ) MPI_Scatter
- ( ) MPI_Bcast
- ( ) MPI_Send

4. In a hybrid OpenMP+MPI application for a modern multicore processor, how do you manage the code of processes and threads? (choose the best answer) 1 punto

- ( ) MPI processes send and receive the code of their threads by way of passing message
- ( ) MPI processes replace multithreading with one process running per core
- ( ) Threads launch multiple MPI processes
- (◉) The code of MPI processes has multithreading

5. In this snippet, 2 processes are communicating, one is sending a message and the other is receiving it, in order to prevent this application from blocking, what are the correct values of DESTINATION and SOURCE ? 1 punto

```
1  if (myid == 1)
2  {
3    buffer="Hello world";
4    MPI_Send(&buffer, count, MPI_INT, DESTINATION, tag, MPI_COMM_WORLD);
5    printf("processor %d sent %d integers\n",myid,count);
6  }
7  if (myid == 2)
8  {
9    MPI_Recv(&buffer, count, MPI_INT, SOURCE, tag, MPI_COMM_WORLD, &status);
10   printf("processor %d received %d intgers\n",myid,count);
11 }
```

- (•) DESTINATION=2 and SOURCE=1
- ( ) DESTINATION=2 and SOURCE=1 or 2
- ( ) DESTINATION=1 and SOURCE=2
- ( ) DESTINATION=1 or 2 and SOURCE=1

MPI

✓ **Envía tu tarea** Probar de nuevo

FECHA DE ENTREGA 25 de oct. 23:59 PDT **INTENTOS** 3 cada 8 hours

✓ **Recibe la calificación**

PARA APROBAR 80 % o más

Calificación

100 %

Ver comentarios

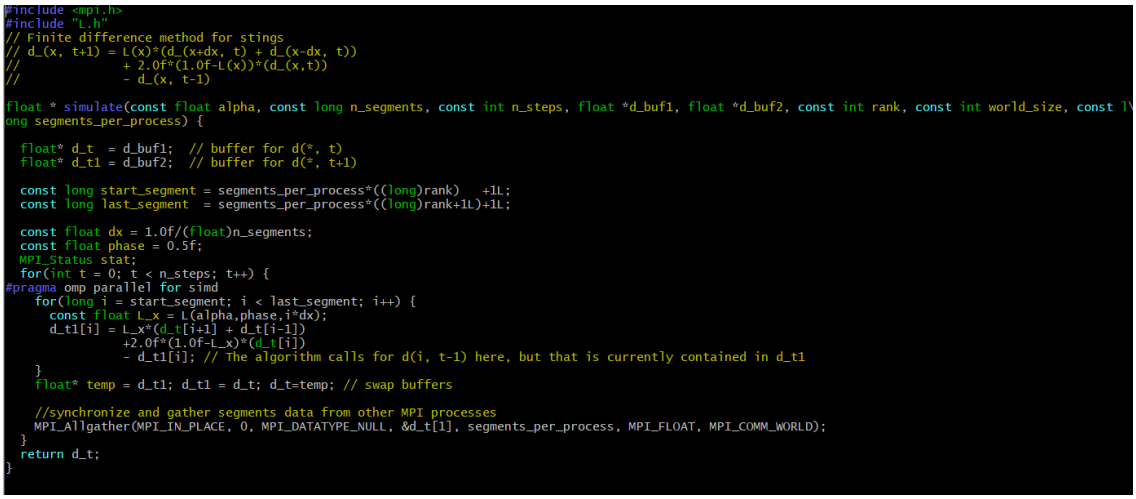
Conservaremos tu puntaje más alto.

Tarea Sem5

Para esta semana era necesario modificar el código del worker.cc para poder cumplir con los requerimientos necesarios para culminar la tarea y el curso de la mejor manera.

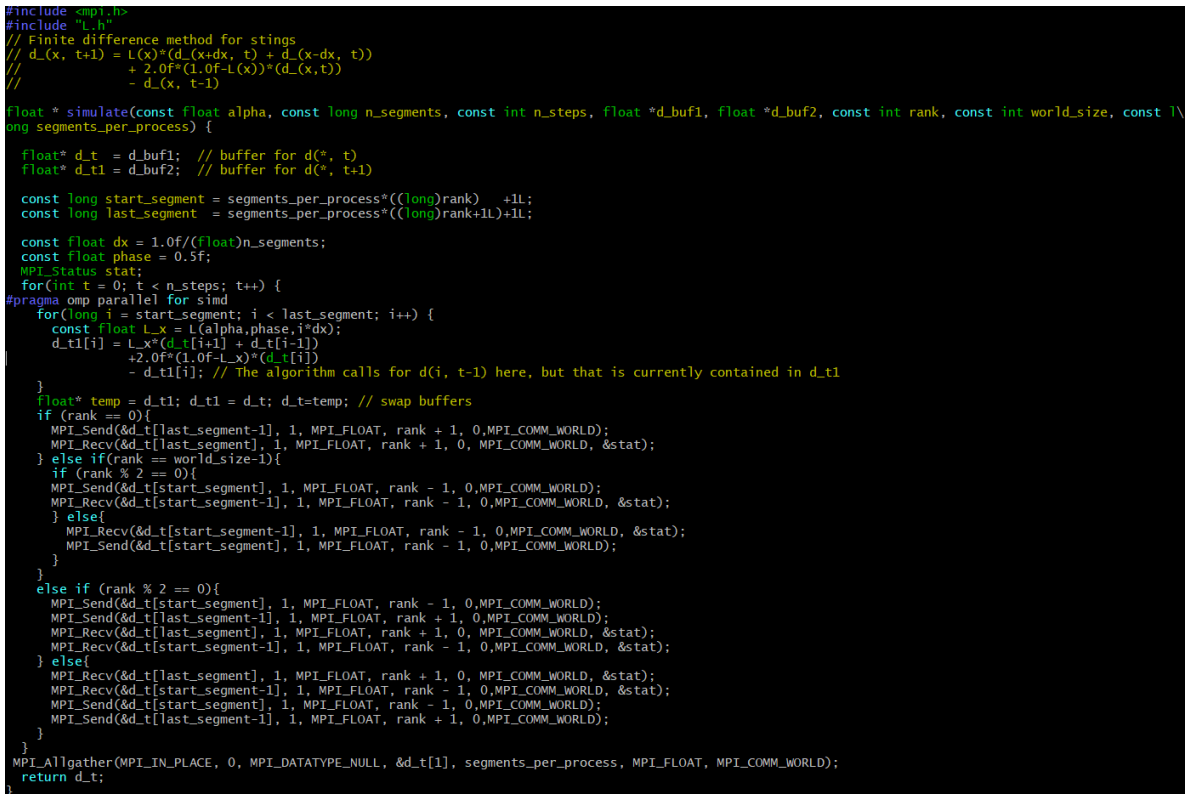
Modificar el worker.cc para que solo los vecinos inmediatos se transfieran después de cada paso de tiempo.

los comandos MPI están bloqueando por defecto. Si inicia una operación MPI_Send en cada proceso a la vez, la aplicación se bloqueará.



```
#include <mpi.h>
#include "L.h"
// Finite difference method for stings
// d_(x, t+1) = L(x)*(d_(x+dx, t) + d_(x-dx, t))
//              + 2.0f*(1.0f-L(x))*(d_(x,t))
//              - d_(x, t-1)

float * simulate(const float alpha, const long n_segments, const int n_steps, float *d_buf1, float *d_buf2, const int rank, const int world_size, const l\
ong segments_per_process) {

  float* d_t  = d_buf1;  // buffer for d(*, t)
  float* d_t1 = d_buf2;  // buffer for d(*, t+1)

  const long start_segment = segments_per_process*((long)rank)   +1L;
  const long last_segment  = segments_per_process*((long)rank+1L)+1L;

  const float dx = 1.0f/(float)n_segments;
  const float phase = 0.5f;
  MPI_Status stat;
  for(int t = 0; t < n_steps; t++) {
#pragma omp parallel for simd
    for(long i = start_segment; i < last_segment; i++) {
      const float L_x = L(alpha,phase,i*dx);
      d_t1[i] = L_x*(d_t[i+1] + d_t[i-1])
                +2.0f*(1.0f-L_x)*(d_t[i])
                - d_t1[i]; // The algorithm calls for d(i, t-1) here, but that is currently contained in d_t1
    }
    float* temp = d_t1; d_t1 = d_t; d_t=temp; // swap buffers

    //synchronize and gather segments data from other MPI processes
    MPI_Allgather(MPI_IN_PLACE, 0, MPI_DATATYPE_NULL, &d_t[1], segments_per_process, MPI_FLOAT, MPI_COMM_WORLD);
  }
  return d_t;
}
```

Worker.cc sin modificar



```
#include <mpi.h>
#include "L.h"
// Finite difference method for stings
// d_(x, t+1) = L(x)*(d_(x+dx, t) + d_(x-dx, t))
//              + 2.0f*(1.0f-L(x))*(d_(x,t))
//              - d_(x, t-1)

float * simulate(const float alpha, const long n_segments, const int n_steps, float *d_buf1, float *d_buf2, const int rank, const int world_size, const l\
ong segments_per_process) {

  float* d_t  = d_buf1;  // buffer for d(*, t)
  float* d_t1 = d_buf2;  // buffer for d(*, t+1)

  const long start_segment = segments_per_process*((long)rank)   +1L;
  const long last_segment  = segments_per_process*((long)rank+1L)+1L;

  const float dx = 1.0f/(float)n_segments;
  const float phase = 0.5f;
  MPI_Status stat;
  for(int t = 0; t < n_steps; t++) {
#pragma omp parallel for simd
    for(long i = start_segment; i < last_segment; i++) {
      const float L_x = L(alpha,phase,i*dx);
      d_t1[i] = L_x*(d_t[i+1] + d_t[i-1])
                +2.0f*(1.0f-L_x)*(d_t[i])
                - d_t1[i]; // The algorithm calls for d(i, t-1) here, but that is currently contained in d_t1
    }
    float* temp = d_t1; d_t1 = d_t; d_t=temp; // swap buffers
    if (rank == 0){
      MPI_Send(&d_t[last_segment-1], 1, MPI_FLOAT, rank + 1, 0,MPI_COMM_WORLD);
      MPI_Recv(&d_t[last_segment], 1, MPI_FLOAT, rank + 1, 0, MPI_COMM_WORLD, &stat);
    } else if(rank == world_size-1){
      if (rank % 2 == 0){
      MPI_Send(&d_t[start_segment], 1, MPI_FLOAT, rank - 1, 0,MPI_COMM_WORLD);
      MPI_Recv(&d_t[start_segment-1], 1, MPI_FLOAT, rank - 1, 0,MPI_COMM_WORLD, &stat);
      } else{
        MPI_Recv(&d_t[start_segment-1], 1, MPI_FLOAT, rank - 1, 0,MPI_COMM_WORLD, &stat);
        MPI_Send(&d_t[start_segment], 1, MPI_FLOAT, rank - 1, 0,MPI_COMM_WORLD);
      }
    }
    else if (rank % 2 == 0){
      MPI_Send(&d_t[start_segment], 1, MPI_FLOAT, rank - 1, 0,MPI_COMM_WORLD);
      MPI_Send(&d_t[last_segment-1], 1, MPI_FLOAT, rank + 1, 0,MPI_COMM_WORLD);
      MPI_Recv(&d_t[last_segment], 1, MPI_FLOAT, rank + 1, 0, MPI_COMM_WORLD, &stat);
      MPI_Recv(&d_t[start_segment-1], 1, MPI_FLOAT, rank - 1, 0,MPI_COMM_WORLD, &stat);
    } else{
      MPI_Recv(&d_t[last_segment], 1, MPI_FLOAT, rank + 1, 0, MPI_COMM_WORLD, &stat);
      MPI_Recv(&d_t[start_segment-1], 1, MPI_FLOAT, rank - 1, 0,MPI_COMM_WORLD, &stat);
      MPI_Send(&d_t[start_segment], 1, MPI_FLOAT, rank - 1, 0,MPI_COMM_WORLD);
      MPI_Send(&d_t[last_segment-1], 1, MPI_FLOAT, rank + 1, 0,MPI_COMM_WORLD);
    }
  }
 MPI_Allgather(MPI_IN_PLACE, 0, MPI_DATATYPE_NULL, &d_t[1], segments_per_process, MPI_FLOAT, MPI_COMM_WORLD);
  return d_t;
}
```

Worker.cc modificado habiendo añadido los comandos MPI para la transferencia de datos

Reenviar

TID	Grado	Retroalimentación	Archivo enviado	Tiempo de presentación
27019	* 100	Compilación: APROBADO Verificación: APROBADO Verificación de rendimiento: APROBADO Código enviado completado en 0.613611s	Solución enviada	2020/09/26 19:07:24

Es sumamente impresionante y gratificante ver como los códigos que una vez se ejecutaron en tiempo altos, algunos muy altos con trabajo y buena implementación pasaban a ejecutarse en tiempos muy bajos

Otros:

Los códigos ejemplo de cada semana se encuentran en el repositorio dentro de la carpeta otros, así mismo en el cluster.

https://github.com/aalmanzarr/Mooc2TET

Calificación general del curso:

¡Aprobaste este curso! Tu calificación es 100.00 %

Elemento	Estado	Fecha lím...	Peso	Calificac...
Código moderno Cuestionario	Aprobado	27 de sep. 23:59 PDT	5 %	**100 %**
Hola Mundo Herramienta externa con calificación	Aprobado	27 de sep. 23:59 PDT	15 %	**100 %**
Vectorización Cuestionario	Aprobado	4 de oct. 23:59 PDT	5 %	**100 %**
Vectorización de la difusión de Montecarlo Herramienta externa con calificación	Aprobado	4 de oct. 23:59 PDT	15 %	**100 %**
OpenMP Cuestionario	Aprobado	11 de oct. 23:59 PDT	5 %	**100 %**
Filtrado multiproceso Herramienta externa con calificación	Aprobado	11 de oct. 23:59 PDT	15 %	**100 %**
Tráfico de memoria Cuestionario	Aprobado	18 de oct. 23:59 PDT	5 %	**100 %**
Lote de FFT en HBM Herramienta externa con calificación	Aprobado	18 de oct. 23:59 PDT	15 %	**100 %**
MPI Cuestionario	Aprobado	25 de oct. 23:59 PDT	5 %	**100 %**
Vibración de cuerdas MPI Herramienta externa con calificación	Aprobado	25 de oct. 23:59 PDT	15 %	**100 %**

Certificado:

intel Software

COURSE CERTIFICATE

27.09.2020

Andrés Almánzar Restrepo

has successfully completed

Fundamentals of Parallelism on Intel Architecture

an online non-credit course authorized by Intel and offered through Coursera

Verify at coursera.org/verify/FLaJKF9NNRPT

Coursera has confirmed the identity of this individual and their participation in the course.