Introduction to High-Performance Computing

Exercise/5

Giorgio Amati

Corso di dottorato in Ingegneria Aeronautica e Spaziale 2024

g.amati@cineca.it / g.amaticode@gmail.com

Agenda

- ✓ **Exercise**
  - OpenMP
  - OpenACC
  - OpenMP-Offload
  - Cuda-Fortran
  - Do concurrent
  - OpenCL
  - Matmul
  - MPI
  - MPI+OpenMP

Code Structure/1

- ✓ Each directory contains source code and a script for compiling and one script for submitting, plus a README file
  - Source code in Fortran & in C (when possible)

```
├── LESSON_1 (openMP)
├── LESSON_2 (openACC)
├── LESSON_3 (openMP offload)
├── LESSON_4 (CUDA)
├── LESSON_5 (do concurrent)
├── LESSON_6 (OpenCL)
├── LESSON_7 (matmul)
├── LESSON_8 (MPI)
└── LESSON_9 (MPI+OpenMP)
```

Code Structure/2

✓ Example: `LESSON_3`

```
clean.sh
compile.C.sh
compile.fortran.sh
inc_precision.h
mm.c
mm.F90
mod_tools.F90
README
submit.c.slurm
submit.fortran.slurm
```

Code Structure/3

✓ The original code is the same so you can see the differences between different parallel paradigm

```
$ diff LESSON_3/mm.c LESSON_2/mm.c
53c53
<   #pragma omp target teams distribute parallel for
collapse(2) map(to:a,b) map(from:c)
---
>   #pragma acc kernels
55,56c55,56
<       for (j = 0; j < nn; j++) {
<           for (k = 0; k < nn; k++) {
---
>       for (k = 0; k < nn; k++) {
>           for (j = 0; j < nn; j++) {
```

Step to do

- ✓ Clean all (if necessary)
- ✓ Compile
- ✓ launch Job

```
[gamati01@login14 LESSON_3]$ ./clean.sh
cleaning directory....
.... all done
[gamati01@login14 LESSON_3]$ ./compile.C.sh
Currently Loaded Modulefiles:
 1) profile/base   2) nvhpc/21.9

Key:
default-version
compiling with  nvc -Minfo=acc -O2 -mp=gpu
main:
     54, Generating map(to:a[:][:])
         Generating map(from:c[:][:])
         Generating map(to:b[:][:])
That's all folks!!!
```

Check results

- ✓ Check reports a single value of the result.
  - Because it is a product on n random-number between 0 and 1 it should be, from a statistical point of view around <n/4>

```
================================
double precision
size 2048
================================
Initialization
Elapsed time for initialization
Total time -----------------> 0.066232
================================
Time  -----------------> 3.200976
Mflops ----------------> 5367.072163
Check -------------> 518.636322
```

Why Matrix-Matrix Multiplication?

- ✓ It is simple and, for size big enough, able to exploit CPU & GPU performance
- ✓ You can easily apply many HPC tricks & features
  - Three nested loops
  - 1 instruction
  - 2*n^3 Flops
  - n^2 Memory
  - 3 load + 1 store per iteration

```
do j = 1, n
   do k = 1, n
      do i = 1, n
         c(i,j) = c(i,j) + a(i,k)*b(k,j)
      enddo
   enddo
enddo
```

LESSON_1 (cpu)

- ✓ `nvc -Minfo=mp -O2 -mp`
- ✓ Try to measure performance (Mflops) increasing the number of threads
- ✓ Does the Speed-up depends form the size of the problem?
- ✓ Number of thread controlled via environmental variable
  - `export OMP_NUM_THREADS=2`

```
#pragma omp parallel shared(a,b,c),private(i,j,k)
#pragma omp for
for (i = 0; i < nn; i++) {
   for (k = 0; k < nn; k++) {
       for (j = 0; j < nn; j++) {
              c[i][j] = c[i][j] + a[i][k]*b[k][j];
…
```

LESSON_2 (GPU)

- ✓ **nvfortran/nvc -Minfo=acc -O2 -acc**
- ✓ Verify performance for version **mm.0.F90** and version **mm.1.F90**
- ✓ Does performance changes with different size?
- ✓ Does **C** code present the same behaviour?

```
!$acc data copyin(a,b) copy(c)
!$acc parallel
     do j = 1, n
        do k = 1, n
          do i = 1, n
          c(i,j) = c(i,j) + a(i,k)*b(k,j)
          enddo
        enddo
     enddo
!$acc end parallel
!$acc end data
```

```
!$acc data copyin(a,b) copy(c)
!$acc parallel collapse(2)
     do j = 1, n
        do i = 1, n
!$acc loop seq
          do k = 1, n
          c(i,j) = c(i,j) + a(i,k)*b(k,j)
          enddo
        enddo
     enddo
!$acc end parallel
!$acc end data
```

LESSON_3 (GPU)

- ✓ Does the OMP Offload version behave as OpenACC (in performance)?
  - ○ **nvfortran -Minfo=acc -O2 -mp=gpu**

```
!$OMP target teams distribute parallel do map(to:a,b) map(from:c)
    do j = 1, n
       do i = 1, n
          do k = 1, n
             c(i,j) = c(i,j) + a(i,k)*b(k,j)
          enddo
       enddo
    enddo
```

LESSON_4:CUDA Fortran (GPU)

- ✓ Again: what about performances?
  - ○ **nvfortran -Minfo=acc -O2 -Mcuda**

```
real(my_kind), dimension(:,:), allocatable:: a ! matrix (origin)
real(my_kind), dimension(:,:), allocatable:: b ! matrix (origin)
real(my_kind), dimension(:,:), allocatable:: c ! matrix
real(my_kind), dimension(:,:), device, allocatable:: a_gpu ! matrix (origin)
real(my_kind), dimension(:,:), device, allocatable:: b_gpu ! matrix (origin)
real(my_kind), dimension(:,:), device, allocatable:: c_gpu ! matrix

a_gpu = a
b_gpu = b
c_gpu = c
!$cuf kernel  do(2) <<<*,*>>>
do j = 1, n
     do j = 1, n
        do k = 1, n
             c_gpu(i,j) = c_gpu(i,j) + a_gpu(i,k)*b_gpu(k,j)
        enddo
     enddo
enddo
```

LESSON_4:CUDA C (GPU)

✓ Again: what about performances?

:x

```
// Allocate memory space on the device
 REAL *d_a, *d_b, *d_c;
 cudaMalloc((void **) &d_a, sizeof(REAL)*nn*nn);
 cudaMalloc((void **) &d_b, sizeof(REAL)*nn*nn);
 cudaMalloc((void **) &d_c, sizeof(REAL)*nn*nn);

 dim3 dimGrid(N_BLOCK, N_BLOCK);
 dim3 dimBlock(nn/N_BLOCK, nn/N_BLOCK);

// copy matrix A and B from host to device
 cudaMemcpy(d_a, a, sizeof(REAL)*nn*nn, cudaMemcpyHostToDevice);
 cudaMemcpy(d_b, b, sizeof(REAL)*nn*nn, cudaMemcpyHostToDevice);

 gpu_mm <<< dimGrid, dimBlock >>> (d_a, d_b, d_c, nn);

 // Transfer results from device to host
 cudaMemcpy(c, d_c, sizeof(REAL)*nn*nn, cudaMemcpyDeviceToHost);
```

LESSON_5 (CPU,GPU)

- ✓ https://www.youtube.com/watch?v=I40-p9MIG6k&t=3s
- ✓ Again what about the performance?
  - ○ **nvfortran -Minfo=acc -O2 -stdpar=gpu**

```
do concurrent(j=1:n)
   do k = 1, n
      do i = 1, n
         c(i,j) = c(i,j) + a(i,k)*b(k,j)
      enddo
   enddo
enddo
```

```
do concurrent(j=1:n,i=1:n)
   do k = 1, n
      c(i,j) = c(i,j) + a(i,k)*b(k,j)
   enddo
enddo
```

LESSON_6/1

- ✓ From https://github.com/ProjectPhysX/OpenCL-Wrapper
- ✓ https://www.youtube.com/watch?v=w4HEwdpdTns

```
int main() {
	Device device(select_device_with_most_flops()); // compile OpenCL C code for the fastest available device

	const uint N = 1024u; // size of vectors
	Memory<float> A(device, N); // allocate memory on both host and device
	Memory<float> B(device, N);
	Memory<float> C(device, N);

	Kernel add_kernel(device, N, "add_kernel", A, B, C); // kernel that runs on the device

	for(uint n=0u; n<N; n++) {
		A[n] = 3.0f; // initialize memory
		B[n] = 2.0f;
		C[n] = 1.0f;
	}

	print_info("Value before kernel execution: C[0] = "+to_string(C[0]));
```

LESSON_6/2

```
…
    A.write_to_device(); // copy data from host memory to device memory
    B.write_to_device();
    add_kernel.run(); // run add_kernel on the device
    C.read_from_device(); // copy data from device memory to host memory

    print_info("Value after kernel execution: C[0] = "+to_string(C[0]));
    return 0;
}
```

```
#include "kernel.hpp"
string opencl_c_container() {
kernel void add_kernel(global float* A, global float* B, global float* C) { //
equivalent to "for(uint n=0u; n<N; n++) {", but executed in parallel
        const uint n = get_global_id(0);
        C[n] = A[n]+B[n];
}
```

LESSON_7 (GPU)

- ✓ Some intrinsic functions, like matmul, has been offloaded to GPU.
  - `nvfortran -Minfo=acc -O2 -acc -gpu=managed -cuda -cudalib`
- ✓ How increases performance with size?

```
#ifdef _OPENACC
    use cutensorex
#endif
……
    real(my_kind), dimension(:,:), allocatable:: a ! matrix (origin)
    real(my_kind), dimension(:,:), allocatable:: b ! matrix (origin)
    real(my_kind), dimension(:,:), allocatable:: c ! matrix
…
    c =  matmul(a,b)
```

LESSON_8(CPU)/1

- ✓ MPI program (works only for 2 tasks)
  - **mpif90 -O3 mm_mpi.F90**

```
!mpi stuff (setup)
   call MPI_INIT(ierr)
   call MPI_COMM_SIZE(MPI_COMM_WORLD,nprocs,ierr)
   call MPI_COMM_RANK(MPI_COMM_WORLD,myrank,ierr)
```

LESSON_8(CPU)/2

- ✓ MPI program (works only for 2 tasks)
  - **mpif90 -O3 mm_mpi.F90**

```
!sending a and b elements
   if(myrank.eq.1) then
    call mpi_recv(a(1,1), n*n, MPI_DOUBLE_PRECISION,0,1,              &
                          MPI_COMM_WORLD, status,ierr)
    call mpi_recv(b(1,1), n*n, MPI_DOUBLE_PRECISION,0,2,              &
                          MPI_COMM_WORLD, status,ierr)
   endif
!
   if(myrank.eq.0) then
    call mpi_send(a(1,1), n*n, MPI_DOUBLE_PRECISION,1,1,              &
                          MPI_COMM_WORLD, ierr)
    call mpi_send(b(1,1), n*n, MPI_DOUBLE_PRECISION,1,2,              &
                          MPI_COMM_WORLD, ierr)
   endif
   call mpi_barrier(MPI_COMM_WORLD,ierr)
```

LESSON_8(CPU)/3

- ✓ MPI program (works only for 2 tasks)
  - ○ **mpif90 -O3 mm_mpi.F90**

```
   if (myrank == 0) then
    jstart = 1; jend = n/2
   endif
!
   if (myrank == 1) then
    jstart = n/2+1; jend = n
   endif
!
   do j=jstart, jend
      do k=1, n
         do i=1, n
        c(i,j) = c(i,j) + a(i,k)*b(k,j)
         enddo
      enddo
   enddo
```

LESSON_8(CPU)/3

- ✓ MPI program (works only for 2 tasks)
  - ○ **mpif90 -O3 mm_mpi.F90**

```
!  collecting c elements
   if(myrank == 0) then
    call mpi_recv(c(1,n/2+1), n*n/2, MPI_DOUBLE_PRECISION,1,4,&
                         MPI_COMM_WORLD, status,ierr)
   endif
!
   if(myrank == 1) then
    call mpi_send(c(1,n/2+1), n*n/2, MPI_DOUBLE_PRECISION,0,4,&
                         MPI_COMM_WORLD, ierr)
   endif
   call mpi_barrier(MPI_COMM_WORLD,ierr)
…
   call MPI_FINALIZE(ierr)
```

- ✓ Some intrinsic functions, like matmul, has been offloaded to GPU.
  - ○ `mpif90 -O3 -mp mm_mpi.F90`

```
!$OMP PARALLEL DO &
!$OMP DEFAULT(NONE) &
!$OMP PRIVATE(i,j,k)  &
!$OMP SHARED(a,b,c,n,jstart,jend)
   do j=jstart, jend
    do k=1, n
        do i=1, n
        c(i,j) = c(i,j) + a(i,k)*b(k,j)
        end do
    end do
   end do
!$OMP END PARALLEL DO
```

Recap/1

✓ Performance in Mflops,

	1024	2048	4096	8192	16384
Lesson_1	.	.	.	.	-
Lesson_2	.	.	.	.	
Lesson_3	.	.	.	.	-
Lesson_4	.	.	.	.	-
Lesson_5	.	.	.	.	-
Lesson_7	.	.	.	.	.
Lesson_8					
Lesson_9					

Recap/2

- ✓ Size Matters: GPUs and CPUs gives best performance for “relatively” big size.
- ✓ Independently from the paradigm used (OpenACC, OpenMP, CUDA) programmer has to take care and optimize the code (unrolling, blocking)
- ✓ (personal) The best solution doesn’t exists. Depends on you experience, knowledge, time devoted to programming, etc. etc.
- ✓ Paradigm can born, change or die in few year. Be ready to change if needed… :-(
- ✓
- ✓ Play a lot, and don’t be afraid to make a lot of mistake/errors