Lab 7:
OpenACC and GPU

Jan Izquierdo
jan.izquierdo@

Laura LLorente
laura.llorente@

17/05/2024

Q1: Answer the following questions:
- How many GPUs are available in this node?
1
- How many SMs has this GPU?
68
- What's the size of GPU's main memory?
10495.52 Mb

Q2: Modify the gpu.slurm file so that the job runs only on a node with a GeForceGTX1080Ti GPU. You have to add a #SBATCH --gres option with the appropriate information. Verify that your solution works correctly.

```
#!/bin/bash
#
#SBATCH --job-name=gpu_check
#SBATCH -N 1 # number of nodes
#SBATCH -n 1 # number of tasks
#SBATCH --partition=cuda-ext.q
#SBATCH --gres=gpu:GeForceGTX1080Ti:1
```

Q3: According to the information generated by the compiler,
Which loops were parallelized? How do you know it?
What are the reasons that prevented parallelization of the other loops?

We compile the loops.c with the makefile, compiler will then inform about the parallelization

Data movements:
Loop information (dependences, loop is or not paralelized):
LOOP 1 → NO PARALLEL 'Complex loop carried dependence of x-> prevents parallelization Loop carried dependence of y-> prevents parallelization'
LOOP 2 → NO PARALLEL Complex loop carried dependence of x-> prevents parallelization Loop carried dependence of y-> prevents parallelization''
LOOP 3 → PARALLELIZABLE 'Loop is parallelizable'

Q4: Look for the sections related to CUDA kernel statistics and CUDA Memory Operation Statistics.
- What's the total time spent by each loop executed at the GPU?

Loop1: 14048 ns (7,8%)
Loop2: 1622022 ns (90,5%)
Loop3: 2944 ns (1,6%)

- What's the total time and data size moved from the CPU (host) to the GPU (device) and viceversa? Can you explain these values according to the operations required by each loop?

Host to GPU: 19.531. It has executed 5 operations
GPU to host: 11.719. It has executed 3 operations

Q6: Modify the loops.c application to use now the parallel loop directive. Execute the new version and check if there are any differences in performance compared to the kernels version. Is this execution correct? Why? Modify the program to get a proof that supports your answers.

We changed

```
#pragma acc kernels
```

for

```
#pragma acc parallel loop private(i)
```

as parallelizing the loop should shorten the execution time and adding the private clause avoids rewriting position i on the arrays.

Q7: Implicit data movement actions are taken by the compiler if no explicit OpenACC data clauses are used. Take a look again at the compilation output of the loops.c program (pg. 10 an 11) and identify which data movement operations were implicitly generated by the compiler.

In the **first loop** the compiler has done the next data movement actions:

- **Generating implicit allocate(x[:1000]) [if not already present]**
  data allocation
- **Generating implicit copyin(x[1:999]) [if not already present]**
  data transfer to device (HtoD)
- **Generating implicit copyout(x[:999]) [if not already present]**
  data transfer back to host (DtoH)

In the **second loop** the compiler has done the next data movement actions:

- **Generating implicit copyin(x[:1000]) [if not already present]**
  data transfer to device (HtoD)
- **Generating implicit copy(y[:1000]) [if not already present]**
  data transfer back to host (DtoH)

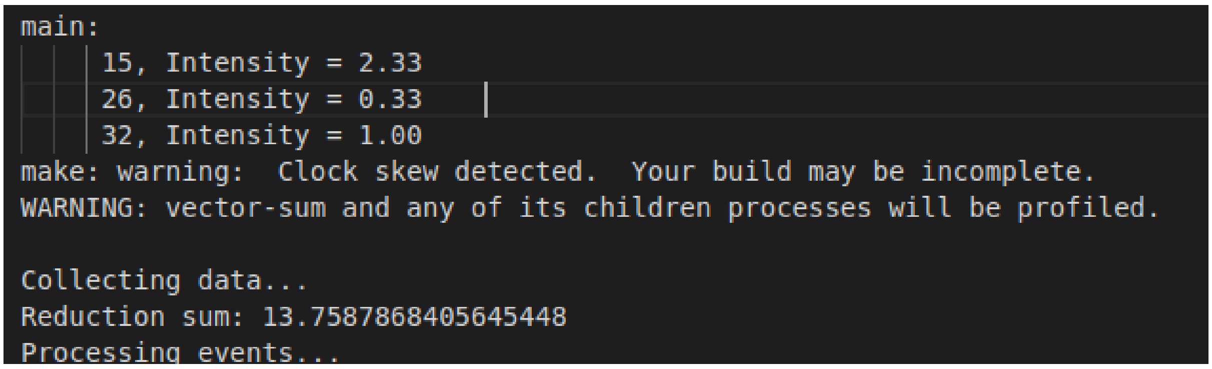
In the **third loop** the compiler has done the next data movement actions:

- **Generating implicit copyin(x[:1000]) [if not already present]**
  data transfer to device (HtoD)
- **Generating implicit copy(y[:1000]) [if not already present]**
  data transfer back to host (DtoH)

PART 6.5

1. Vector addition Calculate the sum of two vectors (C = A + B) in parallel using OpenACC. A skeleton <code is provided in vector-sum.c (a TODO comment shows the place where your code should go). The main computation loop should be parallelised using OpenACC. Try both acc parallel and acc kernels and check the compiler diagnostics output. Run the programs and compare the results.

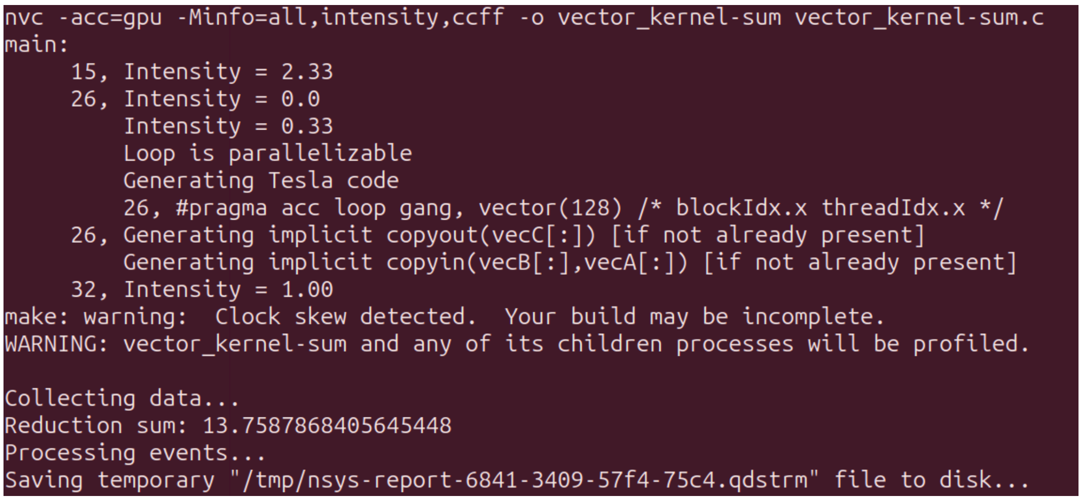
#pragma omp parallel for private(i)
→ INTENSITY 2.33, 0.33 1.00
→ Creates a report in Nsight Systems and database format



```
main:
        15, Intensity = 2.33
        26, Intensity = 0.33
        32, Intensity = 1.00
make: warning:  Clock skew detected.  Your build may be incomplete.
WARNING: vector-sum and any of its children processes will be profiled.

Collecting data...
Reduction sum: 13.7587868405645448
Processing events...
```

#pragma acc kernels
→ INTENSITY 2.33, 0.0, 0.33
→ Info about parallelization of the loops
→ CUDA STATISTICS:
- API
- kernel
- Memory operation
- trace



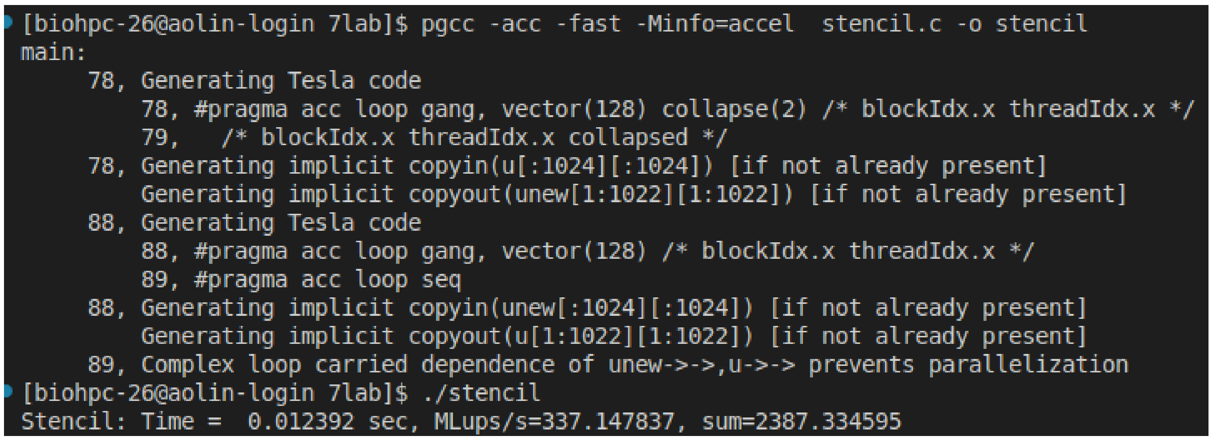
```
nvc -acc=gpu -Minfo=all,intensity,ccff -o vector_kernel-sum vector_kernel-sum.c
main:
     15, Intensity = 2.33
     26, Intensity = 0.0
         Intensity = 0.33
         Loop is parallelizable
         Generating Tesla code
         26, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
     26, Generating implicit copyout(vecC[:]) [if not already present]
         Generating implicit copyin(vecB[:],vecA[:]) [if not already present]
     32, Intensity = 1.00
make: warning:  Clock skew detected.  Your build may be incomplete.
WARNING: vector_kernel-sum and any of its children processes will be profiled.

Collecting data...
Reduction sum: 13.7587868405645448
Processing events...
Saving temporary "/tmp/nsys-report-6841-3409-57f4-75c4.qdstrm" file to disk...
```

2. Stencil Parallelise a simple stencil update kernel with OpenACC parallel or kernels pragmas. The file stencil-loop.c implements a simple stencil update kernel. Search for a TODO tag and try to parallelize the given loop nest with OpenACC parallel or kernels pragmas. Regard that the outer most loop (iter) simulates different time iterations that should not be parallelized. Your ACC parallelization clauses (parallel, kernels, loop),.. should affect only the i and j loops. Data movement clauses should be placed in the right place to avoid excessive data movements. Pay attention to the compiler diagnostics output. Initially, you can set the number of iterations (niter variable) to 4, so that you can quickly check that your parallel code is working properly. Write a first version using only parallel or kernel directives. Write a second version adding explicit data movement clauses. Execute and profile both versions and compare the differences in data movement and execution times. Compile a reference version without OpenACC setting number of iterations (niter variable) to 2000 and compare the results with your best parallel version. What speedup are you getting with your gpu versions of the program?

1: Use kernel or parallel directives

#pragma acc parallel loop:



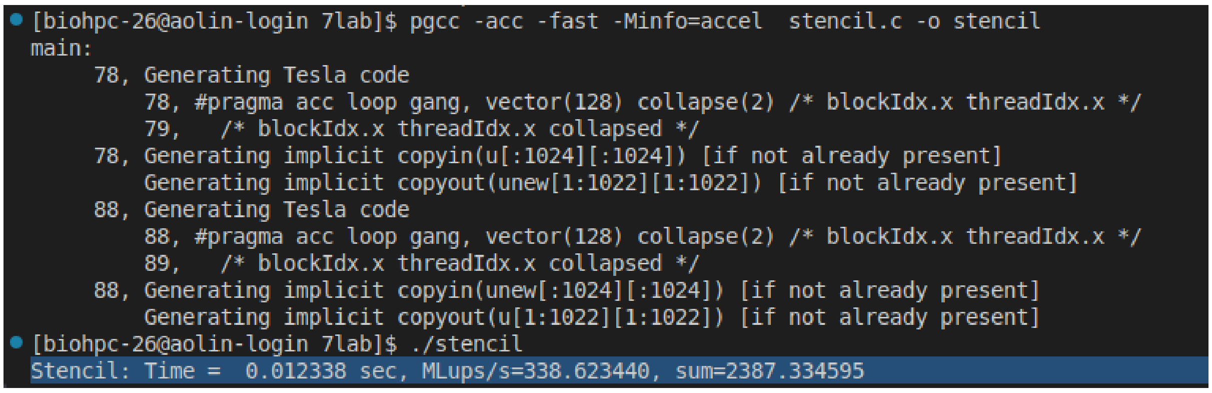
```
[biohpc-26@aolin-login 7lab]$ pgcc -acc -fast -Minfo=accel  stencil.c -o stencil
main:
     78, Generating Tesla code
         78, #pragma acc loop gang, vector(128) collapse(2) /* blockIdx.x threadIdx.x */
         79,   /* blockIdx.x threadIdx.x collapsed */
     78, Generating implicit copyin(u[:1024][:1024]) [if not already present]
         Generating implicit copyout(unew[1:1022][1:1022]) [if not already present]
     88, Generating Tesla code
         88, #pragma acc loop gang, vector(128) /* blockIdx.x threadIdx.x */
         89, #pragma acc loop seq
     88, Generating implicit copyin(unew[:1024][:1024]) [if not already present]
         Generating implicit copyout(u[1:1022][1:1022]) [if not already present]
     89, Complex loop carried dependence of unew->->,u->-> prevents parallelization
[biohpc-26@aolin-login 7lab]$ ./stencil
Stencil: Time =  0.012392 sec, MLups/s=337.147837, sum=2387.334595
```

Stencil: Time = 0.012392 sec, MLups/s=337.147837, **sum=2387.334595**

We could use the collapse(2) clause to indicate to the program that there are 2 loops to be parallelized, so that it handles the parallelization better.

2: Use more explicit clauses

#pragma acc parallel loop private(i) collapse(2)



```
[biohpc-26@aolin-login 7lab]$ pgcc -acc -fast -Minfo=accel  stencil.c -o stencil
main:
     78, Generating Tesla code
         78, #pragma acc loop gang, vector(128) collapse(2) /* blockIdx.x threadIdx.x */
         79,   /* blockIdx.x threadIdx.x collapsed */
     78, Generating implicit copyin(u[:1024][:1024]) [if not already present]
         Generating implicit copyout(unew[1:1022][1:1022]) [if not already present]
     88, Generating Tesla code
         88, #pragma acc loop gang, vector(128) collapse(2) /* blockIdx.x threadIdx.x */
         89,   /* blockIdx.x threadIdx.x collapsed */
     88, Generating implicit copyin(unew[:1024][:1024]) [if not already present]
         Generating implicit copyout(u[1:1022][1:1022]) [if not already present]
[biohpc-26@aolin-login 7lab]$ ./stencil
Stencil: Time =  0.012338 sec, MLups/s=338.623440, sum=2387.334595
```

Stencil: Time = 0.012338 sec, MLups/s=338.623440, sum=2387.334595

For niter=2000:

We first tried executing the code using the gpu.slurm commands:
sbatch gpu.slurm
Contains:
touch stencil.c
make stencil
nsys nvprof --print-gpu-trace stencil
However the output time was massive and inconsistent, so we changed methods
V1: Stencil: Time =**1118.768207 sec,** MLups/s= 1.867204, **sum=69443.905510**

V2: Stencil: Time = **169.119460 sec**, MLups/s= 12.352026, **sum=69443.905510**

Control: Stencil: Time = **8.440643 sec**, MLups/s= 247.489202, **sum=69443.905510**

We used this method instead:
module add cuda/11.2
module load nvhpc/21.2
pgcc -acc -fast -Minfo=accel stencil.c -o stencil
./stencil
This method yielded better results, even though they are not what we expected:
V1: Stencil: Time = 2.024384 sec, MLups/s=1031.903038, sum=69443.905510

V2: Stencil: Time = 2.011776 sec, MLups/s=1038.370077, sum=69443.905510

Control: Stencil: Time = 2.014513 sec, MLups/s=1036.959305, sum=69443.905510

All tests present similar times, we find no speedup, it may be because niter=2000 is too low and if we were to significantly increase niter we should see a difference in the execution times of the parallel and control programs.
There are differences in data movement times, it seems that the higher the time, the lower MLups/s is, this is because time is dividing MLups, so for the same amount of MLups, a larger time reduces MLups/s.