GPU Acceleration

Release 0.0.1

Ben Johnson

Dec 16, 2021

MACHINES

This documents our attempt accelerate DART subroutines during the NERSC GPU Hackathon.

CHAPTER

ONE

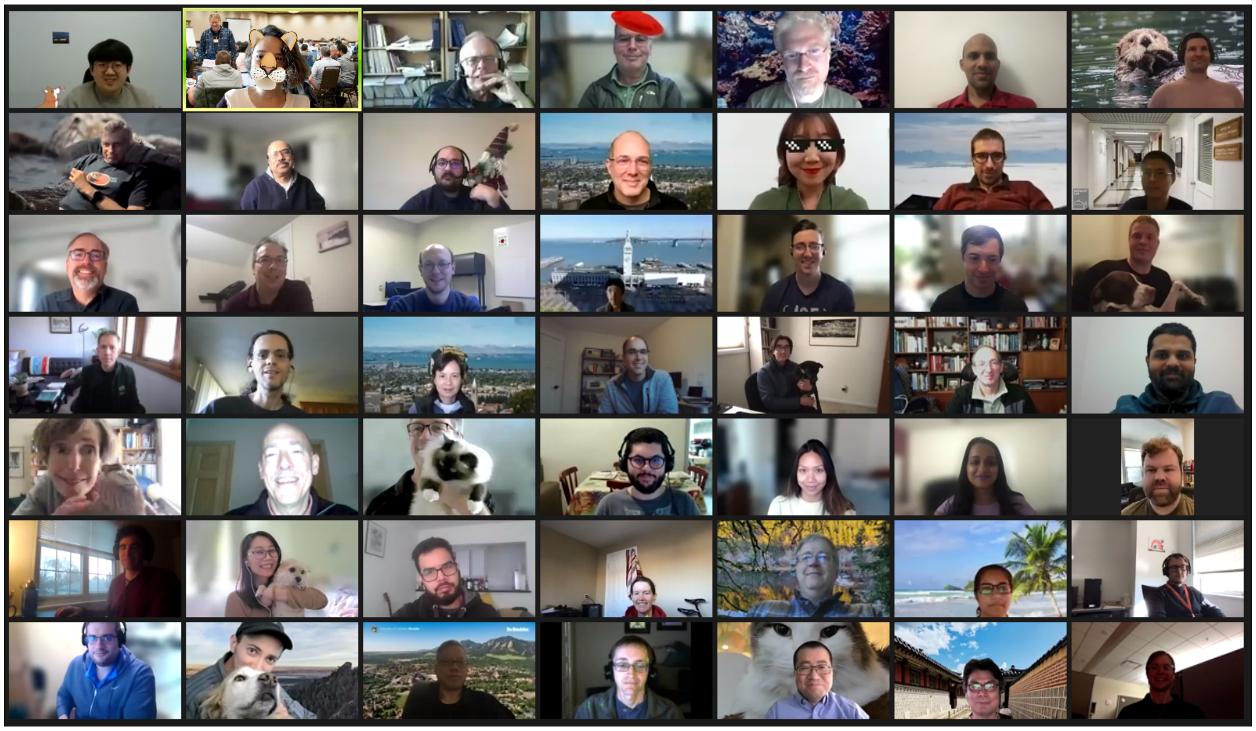
INTRODUCTION SLIDES FOR TEAM-DART

We delivered a PowerPoint presentation to teach the participants and mentors what DART does.

CHAPTER

TWO

GROUP PHOTOS



2.1 Casper

2.1.1 Working set of modules

Chris and the team were able to configure the modules correctly to build the `get_close` kernel on Casper. The working build script is here:

```
/glade/scratch/criedel/HACKATHON/DART-hackathon21/hackathon/get_close_obs/work/build_
↪testcode.sh
```

The working modules commands are:

```
module purge
module load ncarenv/1.3
module load nvhpc/21.9
module load ncarcompilers/0.5.0
module load openmpi/4.1.1
module load netcdf
module list
```

Note: The default nvhpc module on Casper is `20.11`. It doesn't support NVTX, so use `21.9` instead.

2.1.2 Interactive job

This command works for requesting an interactive job on Casper:

```
execcasper -A P86850054 -q gpudev -l select=1:ncpus=8:ngpus=4:mpiprocs=8:mem=200GB -l␣
↪walltime=00:30:00
```

2.1.3 Interactive build

```
execcasper -A P86850054 -q gpudev -l select=1:ncpus=8:ngpus=4:mpiprocs=8:mem=200GB -l␣
↪walltime=00:30:00
cd /glade/work/johnsonb/git/DART-hackathon21/hackathon/get_close_obs/work/
./build_testcode.sh
./test_get_close_obs
```

2.1.4 Job script

The working job script is here:

```
/glade/scratch/criedel/HACKATHON/DART-hackathon21/hackathon/get_close_obs/work/casper_
↪submit.sh
```

2.2 Ascent

2.2.1 MPI issue

`test_get_close_obs` fails to compile on Ascent because the `test_get_close_obs_nml` namelist input statement doesn't exactly match the namelist in `DART-hackathon21/hackathon/get_close_obs/work/input.nml`.

Since `input.nml` was altered to include a tolerance:

```
&test_get_close_obs_nml
   ...
   tolerance = 0.00000001
   ...
/
```

Lines 95-96 in `test_get_close_obs.f90` bust be altered to:

```
namelist /test_get_close_obs_nml/ my_num_obs, obs_to_assimilate, num_repeats, lon_start,␣
↪lon_end, &
                                lat_start, lat_end,cutoff,compare_to_correct,tolerance
```

2.3 Perlmutter

2.3.1 Directory

Helen's scratch directory for the hackathon is:

```
/gpfs/wolf/gen170/scratch/hkershaw/DART-hackathon/hackathon/get_close_obs/work
```

2.3.2 Nsight profiling

To get Nsight systems to profile the compiled program:

```
jsrun -g 1 -n 1 nsys profile ./test_get_close_obs
```

2.4 Compiling

2.4.1 Programming environment

In order to get Nvtx working, the `nvhpc/21.9` module must be available and loaded. In the `mkmf.template` files, we have been using an additional variable, `ACCFLAGS` to set options for the `nvfortran` compiler.

2.4.2 Compiler flags

To run the DART `get_close` kernel, these are the additional compiler flags:

```
ACCFLAGS = -acc -ta=tesla:cc70,deepcopy,pinned -Minfo=accel -Mnofma -r8
```

- `deepcopy` is of particular concern for us. DART has a lot of nested derived types: `type%type%type`. The compiler was not reliably able to determine that the nested types needed copying to the GPU. The deepcopy flag forces this, but ideally you would not force a deep copy on everything. Improvements to the compiler would be needed to fix this. There is a workaround for forcing the correct copy in the code, which is adding a loop around the openACC directives. However, this is not good for code readability as it looks like a pointless loop.
- `Mnofma` was to force less optimization while debugging.
- `r8` was to force double precision type conversions. This was a sanity check while debugging memory problems. It was not needed in the end.
- `Minfo=accel` prints out at compile time what the compiler was able to parallelize. It is similar to the old intel `-vec-report` flag.
- `cc70` is the compute capability, so this depends on the graphics card. Ascent (Oak Ridge's machine) and Casper are V100 gpus so you use `cc70`. Perlmutter is A100 (same as Derecho) so you use `cc80`. This is not intuitive at all for users.

2.4.3 General performance results

`ACCFLAGS = -acc -ta=tesla:cc80,deepcopy` (3x)

`ACCFLAGS = -acc -ta=tesla:cc80,deepcopy,pinned` (15x)

2.5 Zero to GPU hero with OpenACC

2.5.1 Overview

Jeff Larkin of Nvidia produced a lecture Zero to GPU hero with Open ACC.

He notes that there are basically three approaches to writing parallelized code:

1. Parallel syntax in a standard language
2. Directives in OpenACC
3. CUDA

Standard languages

C++ and FORTRAN (2008) have GPU accelerated features in the languages already. Here is an example from FORTRAN:

```
do concurrent (i = 1:n)
   y(i) = y(i) + a*x(i)
enddo
```

Directives

Compiler directives are a middle ground between standard facets of a given language and CUDA. The OpenACC directives annotate your existing code to give a specialized compiler additional information (in comments) that can allow for parallelization.

Such compiler directives enhance the language and enable the code to run well on GPUs when it is compiled on specific compilers, such as the Nvidia HPC compilers.

```
!$acc data copy(x,y)

...

do concurrent (i = 1:n)
   y(i) = y(i) + a*x(i)
enddo

...

!$acc end data
```

CUDA

Maximum level of performance but at the cost of portability. This code runs well on GPUs but isn't available anywhere else – e.g. it can't run on CPUs.

```
attribute(global)
subroutine saxpy(n, a, x, y) {

  int i = blockId%x*blockDim%x + threadIdx%x;

  if (i < n) y(i) += a*x(i)
}

program main
  real         :: x(:), y(:)
  real,devince :: d_x(:), d_y(:)
  d_x = x
  d_y = y

  call saxpy
    <<<(N+255)/256,256>>>(...)

 y = d_y
```

2.5.2 Amdahl's Law

The performance gains of your application made available by parallelization are constrained by what serial steps such as data movement (both I/O and to and from the GPU) and atomic operations.

2.5.3 Types of directives

1. Initializing parallel execution
2. Managing data movement (if the CPU and GPU have distinct memory)
3. Optimization, such as loop mapping

2.5.4 Directive syntax

```
!$acc directive clauses
```

Think of the directive like a function call and the clauses as arguments passed to the function.

2.5.5 CUDA managed memory (aka CUDA unified virtual memory)

Fundamentally, the CPU has access to a large block of relatively slow system memory, while the GPU has access to a smaller block of faster GPU memory.

Passing data between these two memory devices is a serial process that is bandwidth limited. The link between the two memories is PCI-Express or NVLink.

With CUDA managed memory, the programmer does not need to decide where the data resides. This distinction is handled by the operating system and the GPU driver.

2.5.6 miniWeather

This tutorial uses the miniWeather fluid dynamics application.

2.5.7 General steps

First step

Gather the application's initial performance profile. There are many tools for doing this, such as NVIDIA Nsight Systems, gprof, Tau, Vampir, or HPCToolkit.

Adding directives

By adding an "acc parallel loop" directive, the compiler is made aware that the programmer wants to parallelize the loop and that it is safe to do so.

This example is in C++:

```
#pragma acc parallel loop collapse(2)
private(ll,s,inds,stencil,vals,d3_vals,r,u,w,t,p)
```

The `collapse(2)` clause tells the compiler to parallelize the first *and* second loops in the code. **GPUs excel when there are more opportunities for parallelization.** It's a good idea to aggressively use the collapse clause to enable as much parallelism as possible by the compiler.

The `private` clause tells the compiler that each iteration in the parallel computation needs its own copy of the private variables in order to prevent a race condition.

Keep in mind, the `parallel` and `loop` directives are two distinct directives. The `parallel` directive creates "gangs" or multiple memory blocks, and the `loop` directive specifies which loops to parallelize. Most often, however, the directives are written together as `parallel loop`.

Data optimization

Managed memory (unified virtual memory) suffices much of the time for optimizing the flow of memory between the CPU and GPU. However, there are `data` directives for explicit data management. This is an advanced topic that is beyond the scope of Larkin's talk.

Reduction directive

The last directive to add is the reductions directive. It contributes very little to performance, but it is necessary to add the entire time step to the GPU. When every iteration of a loop is doing an operation onto a particular variable, such as a tendency variable, the reduction clause is needed to prevent a data race.

The reduction clause is told which operation is being performed (such as addition) and which variables the operation is being performed upon.

```
#pragma acc parallel loop collapse(2) reduction(+:mass_loc, te_loc)
```

The compiler itself is good at detecting reductions. If the programmer doesn't add the reduction directive, the compiler might print an `Minfo` message notifying the user that it added an implicit reduction clause to a loop. When `Minfo` does this, go back and add the clause explicitly.

2.5.8 Compiling

Nvidia's HPC software development toolkit, which contains compilers among other tools, is known as NVHPC. The command to compile fortran is `nvfortran`. The compilers support GPU and CPUs as well, from x86, to ARM to Power.

Compiler options:

- `-acc` enable OpenACC support
- `-gpu` provides the option to use managed memory: `-gpu=managed` makes data visible on both the CPU and GPU
- `-Minfo=accel` prints compiler feedback on how your code was accelerated (very useful)

The `Minfo` reporting also tells you when the compiler **couldn't** accelerate the code, giving you an opportunity to go back and fix any problems.

Sometimes performance actually decreases because of the serial movement of data between the CPU and the GPU. Jeff Larkin says, "Don't worry, it will go away." When we use a performance profiler, we'll see when data movement occurs, and we can start optimizing the code to minimize data movement. Our goal is to make the computations run on the GPU for the entire time step.

2.5.9 Nvidia Nsight systems

This program provides a GUI to show what exactly is happening with the GPU kernels and the movement of data between CPU and GPU.

Note: The NVTX section of the Nvidia Nsight window shows the calls to the function. If there are gaps between function calls, you should determine why the gaps exist and fill in the time with processing on the GPU.

Look at which function calls are preceded and succeeded by gaps and examine the source code. If there is another function that does computation on the CPU then the code might be triggering page faults. Ensure that the data stays on the GPU and doesn't get moved back and forth between the CPU and the GPU.

In the case of `compute_tendencies_z` there is a loop after the function call that adds the tendencies to the fluid state:

```
else if (dir == DIR_Z) {
  compute_tendencies_z(state_forcing, flux, tend);
}
```

(continues on next page)

(continued from previous page)

```
...

state_out[inds] = state_init[inds} + dt * tend[indt];
```

This reference to `tend[indt]` is causing the page fault.

Final profiling

If done correctly, the GPU acceleration will dramatically speed up serial code. It no longer makes sense to compare the serial code to the GPU-enhanced code. Instead, the full-socket MPI performance should be compared against the GPU-enhanced performance.

If an application is threaded, then the threaded versus GPU-enhanced performance would be the fair comparison.

2.6 Nsight

Nvdia's general-purpose profiling tool is called Nsight Systems. Bob Knight, John Stone and Daniel Horowitz deliver a lecture on Nvidia Nsight Systems.

The tutorial posted in the hackathon's #announcement Slack channel is delivered by Max Katz of Nvidia.

There are two additional tools within the Nsight product family:

- *Nsight Compute*, which is used for CUDA
- *Nsight Graphics*, which is used for graphics shading tools

The latter are multipass tuners that are great for specific applications.

2.6.1 Two ways to use Nsight systems

Graphical user interface

There is a GUI that can be used via a host-target set up.

Command line interface

When the `nvhpc` compiler is loaded, nsight systems can be called from the command line using:

```
nsys [command_switch][optional command_switch_options][application] [optional␣
↪application_options]
```

2.7 NVTX

The Nvidia tools extension SDK (NVTX) is an API that allows for annotating code for performance evaluation with Nsight. NVTX has a Fortran interface.

The `-lnvhpcwrapnvtx` flag must be added to `mkmf.template` in order for it to work properly:

```
ACCFLAGS = -acc -ta=tesla:cc70,deepcopy -Minfo=accel -I${NVHPC_ROOT_PATH}/include -
↪lnvhpcwrapnvtx -Minstrument
```

The `-Minstrument` flag is optional, however it supposedly inserts NVTX ranges at subprogram entries and exits.

Within Fortran source code, the `nvtx` module must be used:

```
use nvtx
```

and ranges can be started and ended as demonstrated below.

```
call nvtxStartRange("First label")

!$acc parallel loop reduction(+:tmp1,tmp2,tmp3)

GLOBAL_OBS: do obs = 1, num_obs_to_assimilate

...

enddo GLOBAL_OBS

!$acc end parallel

call nvtxEndRange
```