Introducing OpenMP to ScalSALE

Shared-Memory Parallelism: CPUs, GPUs and in-between (Final Project)

Submission date: 14.3.23 (submission in individuals)

Abstract

This project is about a scientific computing code simulating some physics (ScalSALE). The code is written in Modern Fortran, in object-oriented programming (OOP) fashion, and supports the Message Passing Interface (MPI) parallel scheme to distribute the calculation domain among multiple processes. The code is a framework for various benchmarks of different hydrodynamic simulations. Specifically, in this work we will focus on the 3D Sedov-Taylor simulation, as will be described later. The given framework currently does not support shared-memory parallelism. In this project we will implement OpenMP parallelization under MPI ranks (processes) to support shared-memory parallelization, including GPU offloading.

ScalSALE

ScalSALE [i] is a scientific benchmark for supercomputers that is suggested to incorporate multi-physical schemes while maintaining scalable and efficient execution times. ScalSALE is an OOP code, written in Modern Fortran, and supports the OpenMPI paradigm (and currently no other shared-memory parallelism). MPI enables the scalability of the code beyond a single compute node, as processors can run on separate nodes and communicate via the network.

ScalSALE is based on the well-known LULESH [ii] benchmark. LULESH is written in Fortran (also, there are versions in C/C++). LULESH supports multiple parallel paradigms (OpenMPI, OpenMP, OpenCL, OpenACC, etc.) to benchmark various physics. However, in comparison to LULESH, ScaleSALE looks more like a real code project. One of the problems that ScalSALE (and also LULESH) can simulate is the well-known 3D Sedov-Taylor. ScalSALE supports either Lagrangian rezoning or Eulerian rezoning of the grid. In this work we will focus on a **3D Sedov-Taylor** problem simulated with **Lagrangian** rezoning.

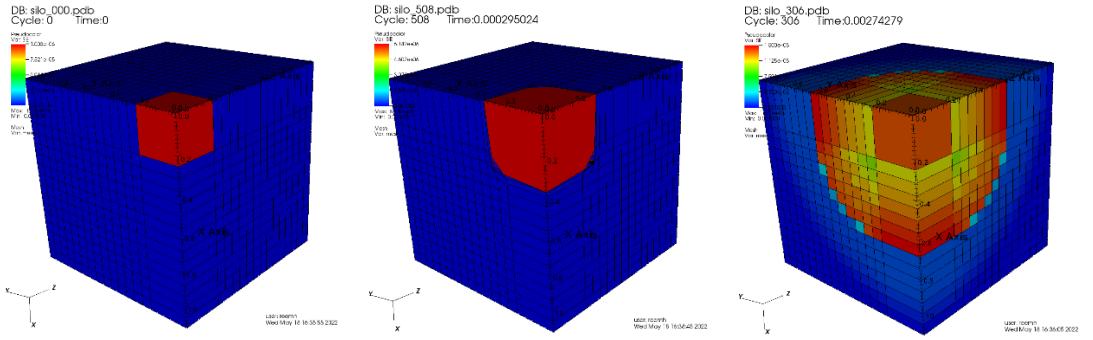

Figure 1: From left to right: (1) A sedov-Taylor simulation in t=0, (2) progress over time with Lagrangian rezoning, (3) progress over time with Eulerian rezoning.

Do not bother yourself with the physics in ScalSALE. It is sufficient to read the code and understand the numerics. In addition, the benchmark supports various types of simulations, and the code is long and contains multiple files and modules. In this project you are provided with a single datafile (that refers to a 3D Sedov-Taylor problem with Lagrangian rezoning), and you should consider working only on the parts of the code that are relevant to that problem – in order to speed up the given execution. Therefore, you are encouraged to use code profiling to find out what parts of the code are relevant.

Our goal: adding OpenMP to ScalSALE

Currently, ScalSALE does not support shared-memory parallel paradigms, such as OpenMP. It supports only the MPI parallel paradigm. In this project we will add support in shared-memory parallelization under each MPI process. This is called MPI+X, where X in our work is OpenMP [iii]. Do not bother yourself with the MPI implementation (how it is coded, and how ranks communicate with each other). The MPI level is encapsulated and implemented under *the Parallel* directory. In this work we simulate 3D Sedov-Taylor only with 1 MPI rank, i.e., only one process.

In this project you will add support for OpenMP in places you find most critical via code profiling for the given datafile. **Your goal is to speed up the calculation** of the given problem in comparison to a serial execution.

In this project you have 2 goals:

- **(Goal #1):**
  You first will insert **CPU shared-memory parallelization** with OpenMP to speed up the entire execution of the given problem.

- **(Goal #2):**
  Secondly, you will target intensive kernel computations in the code that can be **accelerated on a GPU**, and then show what the gained speedup for such kernels when offloading to GPU.

Modern Fortran and OpenMP

In many aspects, Modern Fortran is a comparable language to C++ (see this guide). OpenMP syntax in Fortran is a bit different than OpenMP in C/C++. However, you can quickly learn the gaps! Try the following links:

(1) https://www.openmp.org/wp-content/uploads/OpenMPRefCard-5-2-web.pdf.

(2) https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5-2.pdf.

Datafile:

In this work we provide you a datafile for the 3D Sedov-Taylor problem (with Lagrange rezoning). The size of the grid in is 150X150X150 cells. You can find the datafile in *ScalSALE/src/Datafiles* under the name ***datafile_sedov_taylor_3d_1p_lagrange.json***. Your goal is to optimize the execution of this given problem.

Other examples for datafiles are available under ScalSALE/src/Datafiles but we will not use them. Pay attention that the input datafile for the code should be on path ScalSALE/src/Datafiles/datafile.json. So, for your execution, edit this path to contain the relevant datafile. You can read more about how a datafile is written in the README file.

Validation:

You can validate (or at least get indication) that your implementation is correct if you make sure the resulting values of the grid are repeatable. The code is deterministic, so you would expect the same result for each serial or correct parallel execution. For that purpose, we print the total pressure, the velocity, cell mass and sie values of the grid at the end of the program (in *problem.f90* file). We use a stride of 10 in each axis to reduce storage and I/O overheads.

```
open (unit=411, file='total_pressure_result.txt', status = 'replace')
write(411,*) this%total_pressure%data(1)%values(1:this%nx:10, 1:this%ny:10, 1:this%nz:10)
close (411)
open (unit=412, file='velocity_result.txt', status = 'replace')
write(412,*) this%velocity%data(1)%values(1:this%nx:10, 1:this%ny:10, 1:this%nz:10)
close (412)
open (unit=413, file='total_cell_mass_result.txt', status = 'replace')
write(413,*) this%total_cell_mass%data(1)%values(1:this%nx:10, 1:this%ny:10, 1:this%nz:10)
close (413)
open (unit=414, file='total_sie_result.txt', status = 'replace')
write(414,*) this%total_sie%data(1)%values(1:this%nx:10, 1:this%ny:10, 1:this%nz:10)
close (414)
```

Source code:

The source code of ScalSALE is available on GitHub:

https://github.com/Scientific-Computing-Lab-NRCN/ScalSALE.

Compile and run:

Please read the README carefully, and see what the prerequisites are, and how to compile and execute ScalSALE on your environment. **In addition, use the following guideline to compile and run easily:**

To compile ScalSALE in your environment, you need an mpif90 compiler. mpif90 is used to compile and link MPI programs written in Fortran 90). mpif90 is a wrapper of another Fortran compiler (ifort, ifx, gfortran, etc.). You also need to install json-fortran with the same compiler.

How to install json-fortran easily?

```
$ cd ~
$ git clone https://github.com/jacobwilliams/json-fortran.git
$ cd json-fortran
$ ./build.sh
$ mkdir build
$ cd build
$ FC=mpif90 cmake ../
$ make
```

(make sure mpif90 is loaded with the same compiler you want to compile ScalSALE).

Then edit the file *src/CMakeLists.txt* in lines 14 or 17, according to the compiler you want to use (Intel/GNU) and update the installation path of json-fortran (for example */home/u166450/json-fortran/build*).

Note: The first line of the file ScalSALE/src/CMakeLists.txt is:

```
cmake_minimum_required (VERSION 3.5.2)
```

However, you might have CMake available in lower version (on DevCloud for example the default is CMake 3.16). This is sometimes fine as well, so edit the line:

`cmake_minimum_required (VERSION 3.16)` to be able to run cmake.

Now you are ready to build ScalSALE:

- Run `cmake` to configure the Makefile according to the paths set in CMakeLists.txt. Alternatively, you can run `./clean.sh` (in *src/Scripts*) to configure and compile the project. If you run `"./clean.sh GNU"` it will use the GNU compiler loaded. If you just run `./clean.sh` it will compile with the available Intel compiler (by default, and if available).

- Then enter directory *src/Scripts* and run `./make.sh` to compile the project.
- After you compile the code, run "`./run.sh 1`" to execute the code locally.

Note:

- The `./make.sh` script in *src/Scripts* is used to recompile your code (do not need to configure the Makefile again if you only change the source code).
- The `./clean.sh` script in *src/Scripts* is used to reconfigure the Makefile and compile your code again. Use when you change the compiler/the compilation line/other paths in CMakeLists.txt, etc.
- In the first time, after you clone the files, you might be required to change the scripts' permissions with "`chmod +x`".

Environments:

You can access both Intel DevCloud and NegevHPC for this project. We suggest that you use Intel DevCloud to implement **goal #1** (CPU parallelism) and use NegevHPC to implement **goal #2** (GPU offloading).

Note: We have only a single compute node that is allocated to you all in NegevHPC (this node contains NVIDIA A100 GPU). Therefore, it is very recommended to use the sources available on Intel DevCloud for goal #1. However, you might face difficulties adapting the code to Intel GPUs in the DevCloud, so you are encouraged to tackle goal #2 on NegevHPC with NVIDIA A100 GPU.

Intel DevCloud:

Load OneAPI sources and tools to use one of the MPI compilers for Fortran available to you on Intel DevCloud.

```
source /opt/intel/inteloneapi/setvars.sh > /dev/null 2>&1.
```

then see mpif90 is available to you.

If you run on Intel DevCloud, you can submit the job to a remote compute node (with the q file, as we already know) or connect interactively with "`qsub -I …`" to a desired node and then execute locally.

Hardware: For **goal #1** we recommend running on a CPU node with 2 sockets of intel® xeon® gold 6128 processor. Such a node contains 12 physical cores and 192GB DRAM. You are already experienced with submitting a job to such a node.

Use Intel VTune on DevCloud for profiling.

NegevHPC:

To access NegevHPC, read the material **"Accessing NVIDIA A100 GPU on NegevHPC"** on course website.

When you work on NegevHPC for **goal #2**, you would like to compile ScalSALE with GNU version 9.1.0, which is configured in NegevHPC to support offloading to NVIDIA

GPUs. Use "`module load gnu/9.1.0 openmpi/4.1.3-gnu`" to load the compilers before you compile the code.

Do not forget to compile with `./clean.sh GNU` (otherwise the makefile seeks for an Intel compiler).

If you run on NegevHPC, you already have an example in your home directory how to submit the DAXPY computation with SLURM to the GPU node. You can submit ScalSALE as well via SLURM sbatch in similar way.

Note: You are not provided with graphics on NegevHPC. You can only interact via the terminal. Please do not exceed 200MB usage in your home directory.

General Notes:

- Pay attention that the main function is in *ScalSALE/src/Main/problem.f90*.

```
712         else if (this%mesh%dimension == 3) then
713             do while (this%time%Should_continue() .and. ncyc < max_ncyc)
714                 reem_start = omp_get_wtime()
715                 call this%hydro%do_time_step_3d(this%time)
716                 call this%time%Update_time()
717                 !  call this%Write_to_files()
718                 counter = counter + 1
719                 ncyc = ncyc + 1
720                 write(*,*) "Cycle time: ", omp_get_wtime()-reem_start
721             !       call this%cr%Checkpoint(ckpt_name)
722             end do
723         end if
```

- In the main function in *problem.f90*, the execution is wrapped with a timer, that prints the total execution time at the end of the execution. Also, for each cycle the code prints its execution time.
- You are encouraged to add more timers during the execution to monitor performance of specific parts of the code and the efficiency of OpenMP constructs you may insert.
- We run only 10 cycles of the computation (as you can see in *problem.f90*), because we do not want to waste your time and wait for convergence…
- **You are allowed to make changes to the code** if it helps you with the OpenMP implementation, i.e., it is not always simply adding a pragma and that's it.
- **You are encouraged to change the compiler flags to enhance vectorization and other compiler optimizations to gain more performance.** The compilation line is in *CMakeLists.txt* file. Do not forget to run `cmake` or the `./clean.sh` script again to reconfigure the Makefile (`./clean.sh` also compile your code).
- If you just change the code itself during your work, you can run `./make.sh` to recompile (do not need to configure the Makefile again…).

Submission instructions:

- Submit your solution to **goal #1** and your solution to **goal #2** in separate directories.
- All the files should be submitted in a single zip on the course website (option 1).
- Another option is to push your project to a git repo on GitHub and send us by email the link (Option 2). You can open a private repo and share it with us.
- Submit all the source files of the code, after you implemented your OpenMP version.
- Besides the code, **submit/include a PDF** that explains your work and includes:
  - What did you learn from the profiling?
  - Report of your achievements (you can also add print screens from the terminal of time measurements).
  - Describe where did you include OpenMP pragmas, and what changed you have made to the sources.
  - Describe what compilers you chose to use, and any further optimizations you applied.
  - Describe any interesting issues or insights you faced during the work.

Bonus (up to 10 points to final grade):

When you are done and satisfied with the speed-up you gained for the Lagrangian rezoning simulation, try to speed up the Sedov-Taylor simulation with the Eulerian rezoning (i.e., add parallelism to the *advect.f90* file). The datafile for this problem is *datafile_sedov_taylor_3d_1p_euler.json.* The only difference between this datafile and the previous (Lagrange) one is the rezoning ("Lagrange" --> "Euler").

[i] Harel, Re'em, et al. "ScalSALE: Scalable SALE Benchmark Framework for Supercomputers." arXiv preprint arXiv:2209.01983 (2022)

[ii] LULESH | Advanced Simulation and Computing (llnl.gov)

[iii] https://drive.google.com/file/d/1gzaLG8SgoKhSSWvyU8Q-LnG9zL1YxqNl/view?usp=share_link