Elastic wave modeling with portable GPU offloading

B. Arntsen, NTNU

December 12, 2025

Introduction

Simulation of elastic wavefields are at the core of modern algorithms for inversion and imaging of the Earth at scales ranging from meters to thousands of kilometers. Traditional modeling algorithms are computationally expensive and up to a few years ago required the use of large Linus clusters for realistic simulations.

The rise of Graphical Processing Unit (GPU) technology has significantly increased the speed of numerical algorithms for modeling elastic waves and have made it possible to simulate realistic cases on small laptop systems. However programming languages for GPU systems are less portable and more complicated to use than traditional languages designed for Central Processing Units (CPU). For small research groups or individual scientists this increases the burden of developing software often requiring different versions for different GPU systems.

A strategy to reduce development time for GPU codes is to use available libraries to avoid time consuming low level coding. Python has a large number of available libraries and has in recent years become a popular choice for scientific computing. Although this has proven to be a successful approach, in some cases libraries leads to complicated and inefficient code.

A better approach is to add the capability to excising languages to offload parts of the code to a GPU. The main advantage is that the same source code can be compiled for different GPU systems, reducing the burden of maintaining several versions and removing the need for detailed low level knowledge of GPU systems. Some compilers already exist, but they have taken time to appear and given the pace of hardware development for GPU systems, they are likely to lag behind in the future.

For a small research group a possible way to solve the problem of portability and simplicity of programming is to use the old idea of a "small" language, i.e. a domain specific language sufficiently general to be used for many scientific problems and with language constructs which makes it easy to exploit the speed of GPU systems. To achieve portability, code for different excising and future GPU systems could be generated by the language implementation using the same source code. If the language is used in conjunction with an existing language (f.ex python), it can be specialized to implement time critical numerical kernels. For ease-of-use the syntax of this language should be close to the language it is used together with. A small compiler (or transpiler)

```
# Matrix add
int saxpy(float [*] a, float [*] b, float [*] c):
  int nx,ny

  nx=len(a,0)

  parallel(i=0:nx) :
    a[i] = b[i] + c[i]

  return(1)
```

Figure 1: The ε parallel statement

for such a language should be possible to implement without much effort if the output is restricted to languages suitable for vendor specific platforms, like CUDA C++, Hip C++ or Metal Objective C.

There are a large number of domain specific languges targeted at scientific computing and capable of generating code for GPUs.

In the sections below we show how a small python-like language (appropriately named ε) specialized to scientific computation can be implemented with modest effort and used to write code for simulation of elastic waves. The source code for the simulation can be compiled into code for multiprocessor CPUs or different types of GPU systems.

Elastic wave modeling

Small domain specific language

ε is a small language designed to express grid-based computations of time critical numerical algorithms. The syntax is close to Python, but semantics close to languages such as C.

There are only three basic dataypes, int, float and char which represents integers, floating point numbers and bytes. In addition to the basic scalar data types the language supports dynamic structures and multidimensional arrays. There are no static structures or arrys. ε does not have pointers, but references to arrays and structures are supported. A module system offers the use of an import statement for better structuring of source code.

To support parallel processing a parallel statement is used as shown in listing **??**.

Results

Discussion and conclusions

Figure 2: Run time for Acoustic wave simulation, red line is handcoded cuda with manual memory managment, orange line is hand coded cuda with automatic memory managment using unified memory and black line is code generated by ε.

Figure 3: Speed (gflops) for Acoustic wave simulation, red line is handcoded cuda with manual memory managment, orange line is hand coded cuda with automatic memory managment using unified memory and black line is code generated by ε.