CATEGORY: **DEVELOPMENT TOOLS & LIBRARIES**

POSTER
DT05

CONTACT NAME
Saber Feki: saber.feki@kaust.edu.sa

GPU TECHNOLOGY CONFERENCE

Towards Automatic Performance Tuning of OpenACC Accelerated Scientific Applications

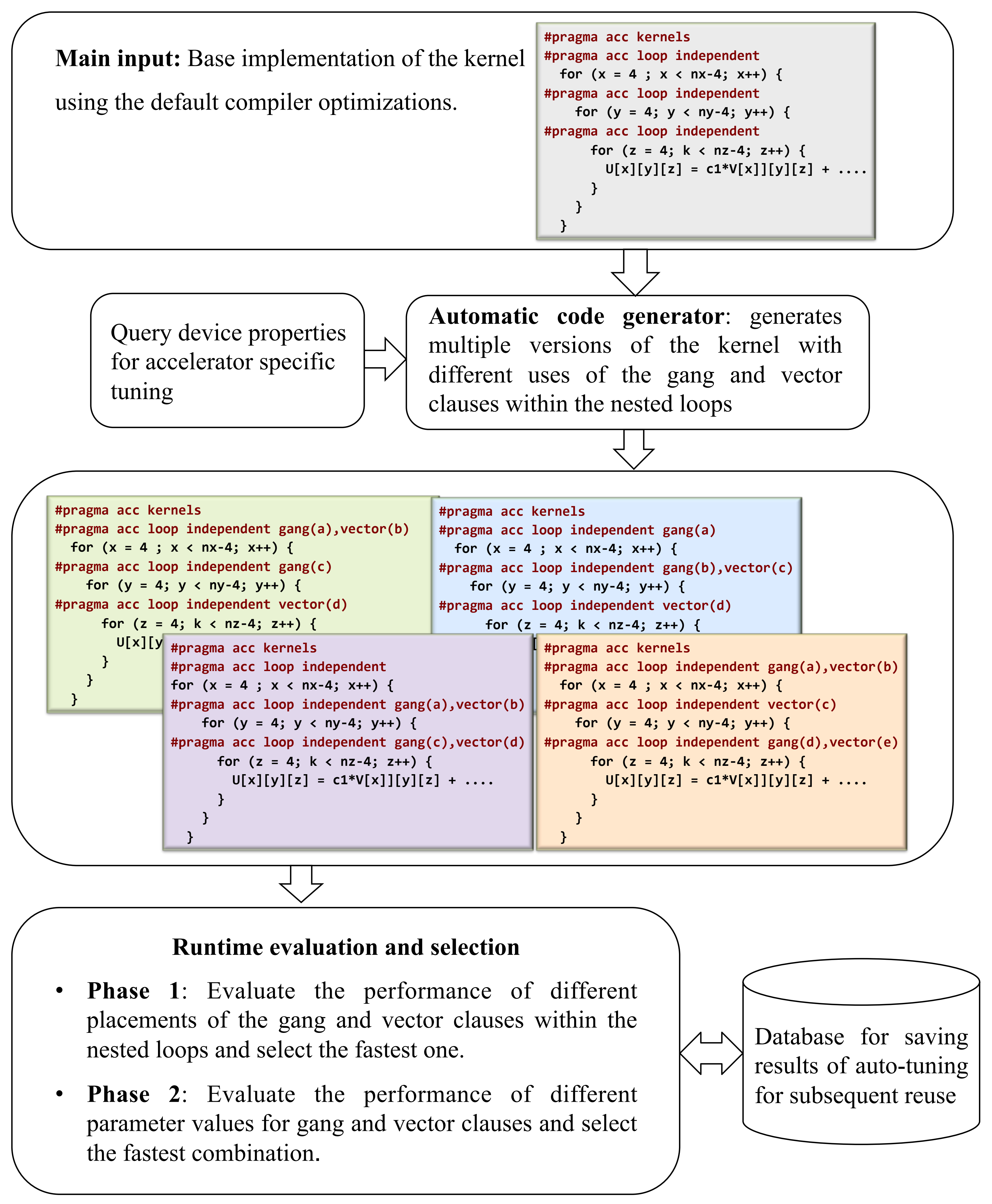
Saber Feki[1], Shahzeb Siddiqui[2]
KAUST Supercomputing Laboratory[1], Division of Computer, Electrical and Mathematical Sciences and Engineering[2]
King Abdullah University of Science and Technology, Kingdom of Saudi Arabia

Abstract

OpenACC was announced in Supercomputing 2011 as a new standard for parallel programming targeting hardware accelerators [1]. Although the goal of the standard is to increase programmer productivity by using compiler directives, getting the best performance of the target device is still tedious and requires performance tuning.

We propose a new methodology for empirical tuning of OpenACC accelerated scientific applications at runtime to relieve this burden from the end user. This strategy is already proven to work well on MPI communication operations in the Abstract Data and Communication Library [2] (ADCL). Indeed we are planning to partially use the framework of ADCL. We present the benefits of tuning OpenACC pragmas and clauses in an accelerated seismic imaging kernel. The application is compiled with the PGI compiler and runs on the NVIDIA K20c GPU. The performance results obtained are encouraging for future development of this methodology and its application to a larger spectrum of scientific applications.

Automatic Performance Tuning Methodology



Main input: Base implementation of the kernel using the default compiler optimizations.

```
#pragma acc kernels
#pragma acc loop independent
  for (x = 4 ; x < nx-4; x++) {
#pragma acc loop independent
    for (y = 4; y < ny-4; y++) {
#pragma acc loop independent
      for (z = 4; k < nz-4; z++) {
        U[x][y][z] = c1*V[x]][y][z] + ....
      }
    }
  }
```

Query device properties for accelerator specific tuning

Automatic code generator: generates multiple versions of the kernel with different uses of the gang and vector clauses within the nested loops

```
#pragma acc kernels
#pragma acc loop independent gang(a),vector(b)
  for (x = 4 ; x < nx-4; x++) {
#pragma acc loop independent gang(c)
    for (y = 4; y < ny-4; y++) {
#pragma acc loop independent vector(d)
      for (z = 4; k < nz-4; z++) {
        U[x][y
      }
    }
  }
```

```
#pragma acc kernels
#pragma acc loop independent gang(a)
  for (x = 4 ; x < nx-4; x++) {
#pragma acc loop independent gang(b),vector(c)
    for (y = 4; y < ny-4; y++) {
#pragma acc loop independent vector(d)
      for (z = 4; k < nz-4; z++) {
```

```
#pragma acc kernels
#pragma acc loop independent
for (x = 4 ; x < nx-4; x++) {
#pragma acc loop independent gang(a),vector(b)
    for (y = 4; y < ny-4; y++) {
#pragma acc loop independent gang(c),vector(d)
      for (z = 4; k < nz-4; z++) {
        U[x][y][z] = c1*V[x]][y][z] + ....
      }
    }
  }
```

```
#pragma acc kernels
#pragma acc loop independent gang(a),vector(b)
  for (x = 4 ; x < nx-4; x++) {
#pragma acc loop independent vector(c)
    for (y = 4; y < ny-4; y++) {
#pragma acc loop independent gang(d),vector(e)
      for (z = 4; k < nz-4; z++) {
        U[x][y][z] = c1*V[x]][y][z] + ....
      }
    }
  }
```

Runtime evaluation and selection

- **Phase 1**: Evaluate the performance of different placements of the gang and vector clauses within the nested loops and select the fastest one.
- **Phase 2**: Evaluate the performance of different parameter values for gang and vector clauses and select the fastest combination.

Database for saving results of auto-tuning for subsequent reuse

Fig. 1: Schematic of the automatic performance tuning procedure

Application: Seismic Imaging

We used in our experiments the isotropic finite difference kernel which constitutes the building block for the Reverse Time Migration (RTM) application and Full Waveform Inversion (FWI), extensively used by the oil and gas exploration industry for velocity model building and seismic imaging of the sub-surface. The Reverse Time Migration application uses forward modeling and backward migration using a finite difference kernel that solves the acoustic wave equation.

$$\frac{1}{c^2}\frac{\partial^2 P}{\partial t^2} = \frac{\partial^2 P}{\partial x^2} + \frac{\partial^2 P}{\partial y^2} + \frac{\partial^2 P}{\partial z^2}$$

where c is the velocity of the propagated wave and P is the wavefield pressure.

The 3D finite difference stencil scheme is 8th order in space and 2nd order in time. We plan in the future to extend this study to different orders in space to explore the impact of computation intensity on the parameter choices of the auto-tuner.

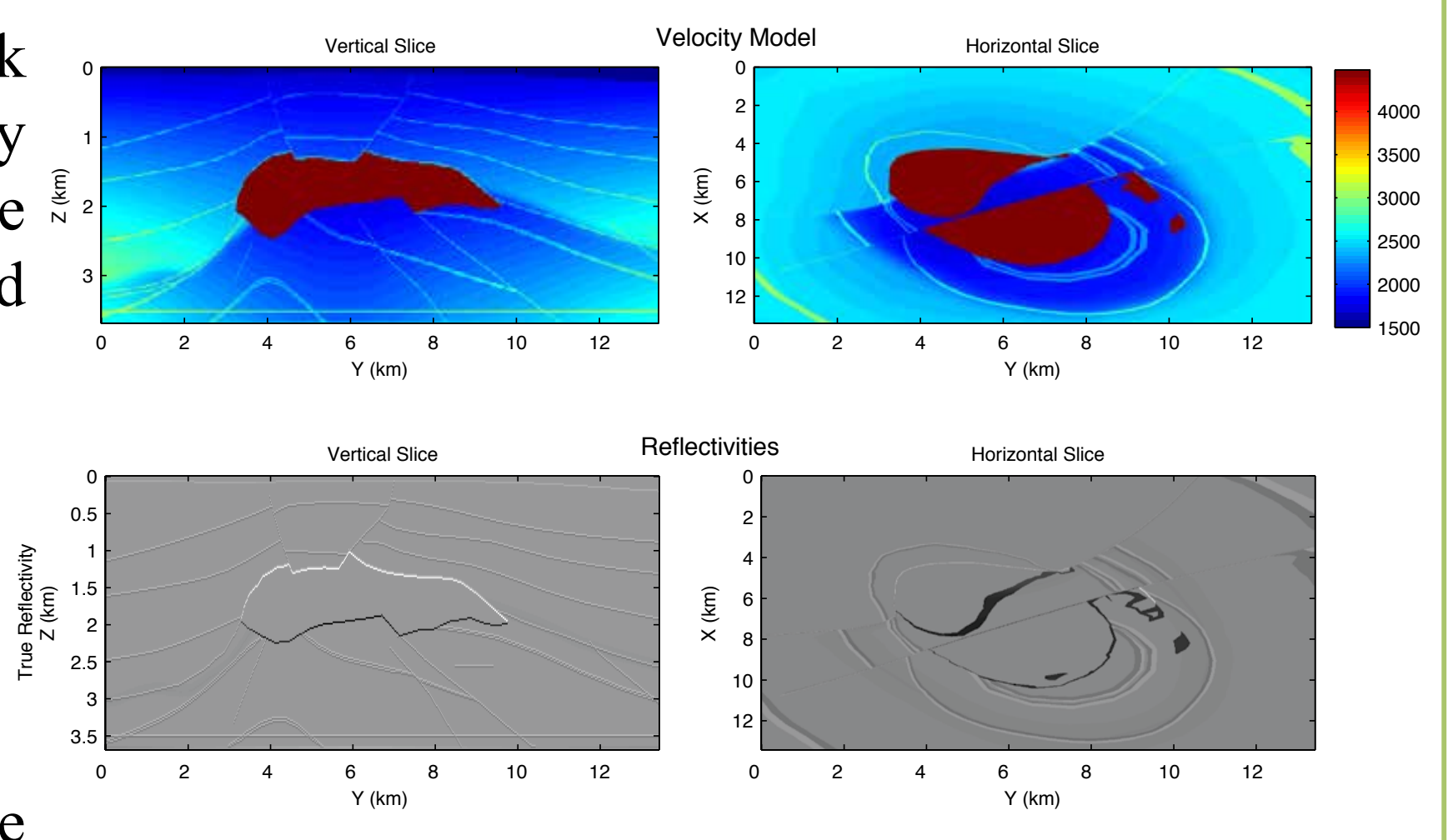


Fig. 2: 3D SEG/EAGE salt velocity model and corresponding reflectivity models obtained by using RTM [3]

We present here the performance results of manually applying the suggested tuning methodology on the RTM isotropic modeling kernel. As shown in Figure 3, the tuning procedure that we suggest, when applied to different 3D domain sizes, leads in many cases to a significant performance improvement (up to 30%) against the input code with the default compiler optimizations. By default, the compiler implicitly uses gang and vector clauses with heuristically determined parameter values. This proves that by tuning OpenACC kernels, we can achieve good performance improvement while still programming at a higher level of abstraction. Moreover, the placement of the gang and vector clauses within the nested loops (distinguished by different colors in figure 1 and table 1) as well as the grid and block sizes leading to the best performance vary for different domain sizes. This reinforces our tendency towards runtime performance tuning, especially when the problem size for instance is a runtime parameter, undefined at compile time.

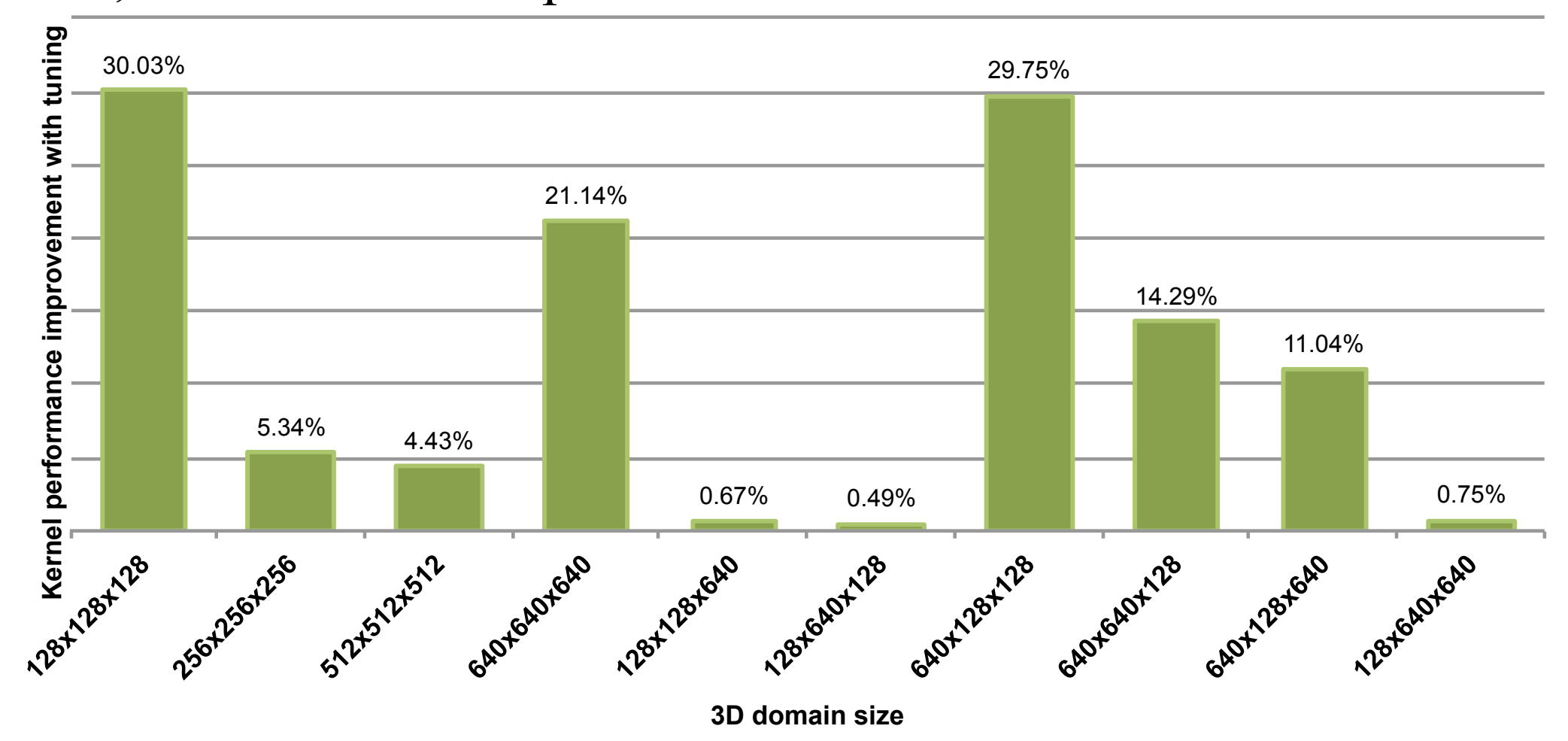


Fig. 3: Kernel performance improvement using the tuning methodology on different domain sizes

3D Domain Size	Grid/block sizes chosen by compiler	Tuned grid/block sizes
128x128x128	grid: [2x30] block: [64x4]	grid: [30x120] block: [64x4]
256x256x256	grid: [4x62] block: [64x4]	grid: [248x6] block: [64x6]
512x512x512	grid: [8x126] block: [64x4]	grid[504x63] block: [32x8]
640x640x640	grid: [10x158] block: [64x4]	grid: [10x316] block: [64x4x2]
128x128x640	grid: [10x30] block: [64x4]	grid: [10x64] block: [64x4]
128x640x128	grid: [2x158] block: [64x4]	grid: [4x256] block: [64x4]
640x128x128	grid: [2x30] block: [64x4]	grid: [2x316] block: [64x4x2]
640x640x128	grid: [2x158] block: [64x4]	grid: [2x316] block: [64x4x2]
640x128x640	grid: [10x30] block: [64x4]	grid: [10x316] block: [64x4x2]
128x640x640	grid: [10x158] block: [64x4]	grid: [10x256] block: [128x4]

Table 1: Compiler choice versus tuning result for gang and vector placement and values for different domain sizes

Although inserting OpenACC annotations is intended to be more productive than writing native CUDA, the best possible OpenACC performance can not be obtained unless the developer goes through a tedious tuning exercise. We suggested a new framework to perform runtime performance tuning automatically for OpenACC accelerated applications. The performance results obtained by hand tuning a finite difference kernel justify future development of this methodology and its application to a larger spectrum of scientific applications.

Future Work

References

[1] OpenACC standard *www.openacc-standard.org/*

[2] Saber Feki, Edgar Gabriel. ***A Historic Knowledge Based Approach for Dynamic Optimization*** , in proceedings of the International Conference on Parallel Computing, 2009, P. 389 - 396

[3] Yunsong Huang, Gerard T. Schuster. ***Multisource least-squares migration of marine streamer and land data with frequency-division encoding***, Geophysical Prospecting, 2012, Vol. 60, P. 663–680

Acknowledgments

- Michael Young (KAUST IT) for system support
- Tareq Males (KAUST CEMSE) for providing sequential C code
- NVIDIA for donating the hardware