Computers & Geosciences 105 (2017) 148–157

Contents lists available at ScienceDirect

Computers & Geosciences

journal homepage: www.elsevier.com/locate/cageo

Performance prediction of finite-difference solvers for different computer architectures

Mathias Louboutin[a,*], Michael Lange[b], Felix J. Herrmann[a], Navjot Kukreja[b], Gerard Gorman[b]

[a] Seismic Laboratory for Imaging and Modeling (SLIM), The University of British Columbia, Canada
[b] Earth Science and Engineering Department, Imperial College, London, UK

ARTICLE INFO

Keywords:
Finite-differences
HPC
Modelling
Multi-physics
Performance
Wave-equation

ABSTRACT

The life-cycle of a partial differential equation (PDE) solver is often characterized by three development phases: the development of a stable numerical discretization; development of a correct (verified) implementation; and the optimization of the implementation for different computer architectures. Often it is only after significant time and effort has been invested that the performance bottlenecks of a PDE solver are fully understood, and the precise details varies between different computer architectures. One way to mitigate this issue is to establish a reliable performance model that allows a numerical analyst to make reliable predictions of how well a numerical method would perform on a given computer architecture, before embarking upon potentially long and expensive implementation and optimization phases. The availability of a reliable performance model also saves developer effort as it both informs the developer on what kind of optimisations are beneficial, and when the maximum expected performance has been reached and optimisation work should stop. We show how discretization of a wave-equation can be theoretically studied to understand the performance limitations of the method on modern computer architectures. We focus on the roofline model, now broadly used in the high-performance computing community, which considers the achievable performance in terms of the peak memory bandwidth and peak floating point performance of a computer with respect to algorithmic choices. A first principles analysis of operational intensity for key time-stepping finite-difference algorithms is presented. With this information available at the time of algorithm design, the expected performance on target computer systems can be used as a driver for algorithm design.

1. Introduction

The increasing complexity of modern computer architectures means that developers are having to work much harder at implementing and optimising scientific modelling codes for the software performance to keep pace with the increase in performance of the hardware. This trend is driving a further specialisation in skills such that the geophysicist, numerical analyst and software developer are increasingly unlikely to be the same person. One problem this creates is that the numerical analyst makes algorithmic choices at the mathematical level that define the scope of possible software implementations and optimisations available to the software developer. Additionally, even for an expert software developer it can be difficult to know what are the right kind of optimisations that should be considered, or even when an implementation is "good enough" and optimisation work should stop. It is common that performance results are presented relative to a previously existing implementation, but such a relative measure of performance is wholly inadequate as the reference implementation might well be truly terrible. One way to mitigate this issue is to establish a reliable performance model that allows a numerical analyst to make reliable predictions of how well a numerical method would perform on a given computer architecture, before embarking upon potentially long and expensive implementation and optimization phases. The availability of a reliable performance model also saves developer effort as it both informs the developer on what kind of optimisations are beneficial, and when the maximum expected performance has been reached and optimisation work should stop.

Performance models such as the roofline model by Williams et al. (2009) help establish statistics for best case performance — to evaluate the performance of a code in terms of hardware utilization (e.g. percentage of peak floating point performance) instead of a relative speed-up. Performance models that establish algorithmic optimality and provide a measure of hardware utilization are increasingly used to determine effective algorithmic changes that reliably increase performance across a wide variety of algorithms (Asanovic et al., 2006). However, for many scientific codes used in practice, wholesale algo-

* Corresponding author.
E-mail address: mloubout@eos.ubc.ca (M. Louboutin).

http://dx.doi.org/10.1016/j.cageo.2017.04.014
Received 16 September 2016; Received in revised form 18 April 2017; Accepted 26 April 2017
Available online 18 May 2017

rithmic changes, such as changing the spatial discretization or the governing equations themselves, are often highly invasive and require a costly software re-write. Establishing a detailed and predictive performance model for the various algorithmic choices is therefore imperative when designing the next-generation of industry scale codes.

We establish a theoretical performance model for explicit wave-equation solvers as used in full waveform inversion (FWI) and reverse time migration (RTM). We focus on a set of widely used equations and establish lower bounds on the degree of the spatial discretization required to achieve optimal hardware utilization on a set of well known modern computer architectures. Our theoretical prediction of performance limitations may then be used to inform algorithmic choice of future implementations and provides an absolute measure of realizable performance against which implementations may be compared to demonstrate their computational efficiency.

For the purpose of this paper we will only consider explicit time stepping algorithms based on a second order time discretization. Extension to higher order time stepping scheme will be briefly discussed at the end. The reason we only consider explicit time stepping is that it does not involve any matrix inversion, but only scalar product and additions making the theoretical computation of the performance bounds possible. The performance of other classical algorithm such as matrix vector products or FFT as described by (Patterson and Hennessy, 2007) has been included for illustrative purposes.

2. Introduction to stencil computation

A stencil algorithm is designed to update or compute the value of a field in one spatial location according to the neighbouring ones. In the context of wave-equation solver, the stencil is defined by the support (grid locations) and the coefficients of the finite-difference scheme. We illustrate the stencil for the Laplacian, defining the stencil of the acoustic wave-equation (Eq. (A.1)), and for the rotated Laplacian used in the anisotropic wave-equation Eqs. (A.3), (A.4) on Figs. 1–2. The points coloured in blue are the value loaded while the point coloured in red correspond to a written value.

The implementation of a time stepping algorithm for a wavefield u, solution of the acoustic wave-equation (Eq. (A.1)) is straightforward from the representation of the stencil. We do not include the absorbing boundary conditions (ABC) as depending on the choice of implementation it will either be part of the stencil or be decoupled and treated separately.

Algorithm 1. Time-stepping

for $t = 0$ **to** $t = n_t$ **do**
  for $(x, y, z) \in (X, Y, Z)$ **do**

$$u(t, x, y, z) = 2u(t-1, x, y, z) - u(t-2, x, y, z) + \sum_{i \in stencil} a_i u(t-1, x_i, y_i, z_i)$$

  end for
  Add Source: $u(t,.,.,.) = u(t,.,.,.) + q$
end for

In Algorithm 1, (X, Y, Z) is the set of all grid positions in the computational domain, (x, y, z) are the local indices, (x_i, y_i, z_i) are the indices of the stencil positions for the centre position (x, y, z) and n_t is the number of time steps and q is the source term decoupled from the stencil. In the following we will concentrate on the stencil itself, as the loops in space and time do not impact the theoretical performance model we introduce. The roofline model is solely based on the amount of input/output (blue/red in the stencils) and arithmetic operations (number of sums and multiplication) required to update one grid point, and we will prove that the optimal reference performance is independent of the size of the domain (number of grid points) and of the number of time steps.

Notes on parallelization:

Using a parallel framework to improve an existing code is one of the most used tool in the current stencil computation community. It is however crucial to understand that this is not an algorithmic improvement from the operational intensity. We will prove that the algorithmic efficiency of a stencil code is independent of the size of the model, and will therefore not be impacted by a domain-decomposition like parallelization via OpenMP or MPI. The results shown in the following are purely dedicated to help the design of a code from an algorithmic point of view, while parallelization will only impact the performance of the implemented code by improving the hardware usage.

3. Roofline performance analysis

The roofline model is a performance analysis framework designed to evaluate the floating point performance of an algorithm by relating it to memory bandwidth usage (Williams et al., 2009). It has proved to be very popular because it provides a readily comprehensible performance metric to interpret runtime performance of a particular implementation according to the achievable optimal hardware utilization for a given architecture (Williams and Patterson, 2008). This model has been applied to real-life codes in the past to analyze and report performance including oceanic climate models Epicoco et al. (2014), combustion modelling Chan et al. (2013) and even seismic imaging (Andreolli et al., 2014). It has also been used to evaluate the effectiveness of implementation-time optimisations like autotuning (Datta and Yelick, 2009), or cache-blocking on specific hardware platforms like vector processors (Sato et al., 2009) and GPUs (Kim et al., 2011). Tools are available to plot the machine-specific parameters of the roofline model automatically (Lo et al., 2014). When more information about the target hardware is available, it is possible to refine the roofline model into the cache-aware roofline model which gives more accurate predictions of performance (Ilic et al., 2014). The analysis presented here can be extended to the cache-aware roofline model but in order to keep it general, we restrict it to the general roofline model.

The roofline model has also been used to compare different types of basic numerical operations to predict their performance and feasibility on future systems (Barba and Yokota, 2013), quite similar to this paper. However, in this paper, instead of comparing stencil computation to other numerical methods, we carry out a similar comparison between numerical implementations using different stencil sizes. This provides an upper-bound of performance on any hardware platform at a purely conceptual stage, long before the implementation of the algorithm.

Other theoretical models to predict upper-bound performance of generic code on hypothetical hardware have been built (Lai and Seznec, 2013; Wahib and Maruyama, 2014; Hofmann et al., 2015a; Duplyakin et al., 2016) but being too broad in scope, can not be used to drive algorithmic choice like choosing the right discretization order. Some of these models have also been applied to stencil codes (Stengel et al., 2015; Datta et al., 2009), however the analysis was of a specific implementation and could not be applied in general. There are many tools to perform performance prediction at the code-level (Hammer et al., 2015; Narayanan et al., 2010; Unat et al., 2015; Rahman et al., 2011). However, any tool that predicts performance based on a code is analyzing the implementation and not the algorithm in general. Although performance modelling is a deep and mature field, most work is restricted to modelling the performance of specific implementations in code. Hofmann et al. (2015b) makes a comparison quite similar to the one we do here where two algorithmic choices for the same problem are being compared with a performance model.

In this section we demonstrate how one creates a roofline model for a given computer architecture, and derives the operational intensity for