An Investigation of Decoupling Single Element Solutions from the Mesh in the Finite Element Method on Nvidia GPUs

MSc. High Performance Computing

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Author: Alex Keating
Student ID: 17307230
Supervisor: Jose Refojo & Prof. Kirk Soodhalter

Department of Mathematics
Faculty of Engineering, Mathematics and Science
Trinity College Dublin, University of Dublin

August 18, 2019

Declaration

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at `http://www.tcd.ie/calendar`.

I have completed the Online Tutorial in avoiding plagiarism 'Ready, Steady, Write', located at `http://tcd-ie.libguides.com/plagiarism/ready-steady-write`.

Alex Keating

August 18, 2019

Acknowledgements

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne die Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Schriften entnommen wurden, sind als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden. Ich versichere, dass die eingereichte elektronische Fassung der eingereichten Druckfassung vollständig entspricht.

Abstract

[Abstract goes here (max. 1 page)]

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Nowadays, with the advent of modern computing, there has been a huge push towards taking advantage of these resources. In the scientific world, there is a constant drive towards building optimised libraries for performing operations, spanning areas such as, basic linear algebra [CITE], fast Fourier transformations [CITE] or genetic algorithms for machine learning [CITE]. All of these libraries have been written with the intentions of exhausting as much processing power, memory and parallelisation as possible. These libraries are available across all kinds of architectures, Intel's MKL for example built for Intel CPUs, Nvidia's CUDA SDK written for their own GPUs or MAGMA, a 3rd party library written for heterogeneous architectures. These advancements in scientific computing have not come from nowhere, but rather are becoming duly necessary as most modern problems in science become impossible to solve without taking advantage of modern computing resources. A clear example of this was seen last year when the first image of a black hole was rendered, using a novel image cleaning machine learning algorithm and 5 petabytes of data - clearly something that cannot be done without some form of distributed memory architecture.

Nvidia have been among the leaders of this charge from both a hardware and software point...

The need for taking advantage of parallelism in modern science now clear, one problem that crops up in a vast array of areas is solving partial differential equations, or PDEs. These are relationships between variables and their partial derivatives and can be seen in areas such as fluid dynamics with the Navier-Stokes equations, in finance with the Black-Scholes equation or even in engineering when modelling stress of a structure - an important one for the topic at hand in this paper. Unfortunately, while analytic solutions exist for theoretical problems of this nature, studied in undergraduate courses, in the real world most of these PDEs are not of this convenient solvable nature and thus require numerical methods to solve. There are many variants of numerical methods which one can use to solve PDEs, some more simple than others, such as the finite difference method which decomposes the domain into a simple grid and approximates the derivatives, and some more complex but robust, like meshfree methods which create a collection of Voronoi cells on the domain instead of a basic grid - allowing for more complex structures. Certain methods land somewhere in a middle ground of both complexity and adaptability such as the finite volume method and the finite element method - the later of which will form the basis of this study.

The finite element method [CITE STRANG] is a numerical method developed originally for use in engineering for modelling stress on structures and has since quickly expanded to use in all branches of science such as electrostatics, something and something. Its engineering foundations will become clear throughout the paper as much of the terminology has remained unchanged. This paper will investigate current approaches

to applying the finite element method on Nvidia GPUs and attempt at isolating and pinpointing certain bottlenecks for parallelism upon which may be improved. The importance of this is clear, as PDE-related problems get larger and more complex, the need for more computing power is evident to get approximate solutions in reasonable amounts of time.

1.1 Preliminaries

Maybe ???

Chapter 2

Finite Element Method

This paper focuses its mathematics on implementing the finite element method to solve PDEs. This chapter will go through the necessary mathematics behind each of the steps behind the finite element method such as variational calculus, approximation theory and the finite elements themselves. The chapter also goes through the actual approach itself, alongside the worked example for the case of this study of the Laplace equation.

2.1 General Problem

Before the nuts and bolts of the finite element method are discussed, let us first consider an n-dimensional, general n^{th} order PDE of the form,

$$f\left(\mathbf{x}; u(\mathbf{x}), \frac{\partial u}{\partial x_1}, \ldots, \frac{\partial u}{\partial x_n}; \frac{\partial^2 u}{\partial x_1 \partial x_1}, \ldots, \frac{\partial^2 u}{\partial x_1 \partial x_n}; \ldots\right) = 0, \tag{2.1}$$

where $\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$. For a case of a 2-dimensional, 2^{nd} order PDE, we end up with the operator,

$$\mathcal{L} = a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2} + F\left(x, y; u; \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}\right). \tag{2.2}$$

This can be used, applied to a function to leave a PDE,

$$\mathcal{L}u = 0, \tag{2.3}$$

which is going to be our entire basis for this study - attempting to approximate the function u. This problem can then be bounded by three types of non-homogeneous boundary conditions in order to be properly posed and have a unique/non-trivial solution:

1. Dirichlet condition: $u = g(s)$.
2. Neumann condition: $\frac{\partial u}{\partial \mathbf{n}} = h(s)$.
3. Robin condition: $\frac{\partial u}{\partial \mathbf{n}} + \sigma(s)u = k(s)$,

where s is he arc length of the boundary C and $\mathbf{n}$ is a vector, externally normal to C. How these conditions are imposed and handled will be seen later in the report.

2.2 Approximation Theory

Suppose there exists some function $u(x)$ that we wish to approximate. The most common way to is to estimate the value of the function using a collection of *basis functions*

$\psi_i(x)$, and unknown coefficients, c_i, giving,

$$u(x) \approx \sum_{i=0}^{N} c_i \psi_i(x). \tag{2.4}$$

There are a collection of ways to construct your basis functions and obtain solutions to the approximation (REFERENCE EQ) and this paper looks at three in particular:

- least squares.
- Galerkin.
- weighted residuals.

There are other methods of approximation such as collocation and regression but they are not discussed here. Before these approaches are explained, two things must first be defined. Firstly, consider a function space V defined by the span of set of basis functions,

$$V = \text{span}\{\psi_0, \ldots, \psi_N\}, \tag{2.5}$$

then it can be said the any function $u \in V$ can be written as a linear combination of the basis functions,

$$u(x) = \sum_i c_i \psi_i. \tag{2.6}$$

Consider now, functions f, g - the squared norm or inner-product of these two functions is defined as,

$$\langle f, g \rangle = \int f(x) g(x) dx \tag{2.7}$$

2.2.1 Least Squares

Suppose we are given a function $f(x)$ which needs to be approximated by a function $u(x) \in V$ as defined above. The most obvious way to approximate this function would be to minimise the differential between the two, $f - u$. Subbing this into the inner product we are left with,

$$e = \langle f - u, f - u \rangle = \langle f - \sum_i c_i \psi_i, f - \sum_i c_i \psi_i \rangle, \tag{2.8}$$

$$e = \langle f, f \rangle - 2 \sum_i c_i \langle f, \psi_i \rangle + \sum_{i,j} c_i c_j \langle \psi_i, \psi_j \rangle. \tag{2.9}$$

Of course, now as is well known from optimisation, to minimise this residual function, its derivative must be taken at each of the N points, $\frac{\partial e}{\partial c_i}$. Evaluating and setting $\frac{\partial e}{\partial c_i}$, results in the equation,

$$-\langle f, \psi_i \rangle + \sum_j c_j \langle \psi_i, \psi_j \rangle = 0, \quad i \in \{0, \ldots, N\}, \tag{2.10}$$

which can equally be written as,

$$\sum_j A_{i,j} c_j = b_i, \tag{2.11}$$

where,

$$A_{i,j} = \langle \psi_i, \psi_j \rangle, \tag{2.12}$$

$$b_i = \langle f, \psi_i \rangle. \tag{2.13}$$

Now there is a system of linear equations which can be solved by usual means to obtain the approximation $u(x)$ of the function $f(x)$. Mathematically, it is equivalent to say that the method of least squares utilises the inner-product of the residuals, and solves by minimising it,

$$\min_{c_0,\ldots,c_N} \langle e, e \rangle, \tag{2.14}$$

giving the system of $N+1$ equations,

$$\left\langle e, \frac{\partial e}{\partial c_i} \right\rangle = 0, \quad i \in \{1, \ldots, N\}. \tag{2.15}$$

2.2.2 Galerkin

In the previous subsection, it was seen that the least squares method operates by minimising the error term between the two functions, or alternatively, forcing the error to be orthogonal to the function space V. However, in reality, we do not actually know the true error as f is not explicitly known and so instead a residual R is used. Take for example, Eq. (REFERENCE HERE), if we sub in an approximation $\hat{u} = \sum_i c_i \psi_i$, we get,

$$R = \mathcal{L}\left(\sum_i c_i \psi_i\right) \neq 0. \tag{2.16}$$

Now, the residual R can be made orthogonal to the space V by imposing,

$$\langle R, v \rangle = 0, \forall v \in V, \tag{2.17}$$

and since any function in V can be approximated using (REFERENCE), it can be said that,

$$\langle R, \psi_i \rangle = 0, \quad i \in \{1, \ldots, N\}, \tag{2.18}$$

thus leaving another system of linear equations to solve. This is also known as projecting R onto V.

2.2.3 Weighted Residuals

The method of weighted residuals is a relatively simple generalisation of the standard Galerkin method. Rather than imposing that the residual is orthogonal to the space V known as the trial space, it is chosen to be orthogonal to some other space W, also known as the test space. This leaves the equation,

$$\langle R, v \rangle = 0, \forall v \in W, \tag{2.19}$$

, where,

$$W = \text{span}\{w_0, \ldots, w_n\} \tag{2.20}$$

and so this leaves,

$$\langle R, w_i \rangle = 0, \quad i \in \{1, \ldots, N\}, \tag{2.21}$$

again, leaving a system of $N+1$ linear equations.

2.3 Variational Calculus

2.3.1 Weak Formulation

As it should be clear by now, the overall aim of the finite element method is the minimise a residual in order to get an approximation of the function u as seen in (REFERENCE). What may not seem immediately obvious at a first glance, but an important thing to factor in, (REFERENCE) is not how the PDE problem is usually posed when trying to apply FEM. In fact, usually, the problem is posed in its *weak formulation*. The weak form of this equation is one which contains at most first-order derivative, as opposed to its strong form containing second-order derivatives. Usually, if only approximating a function, this issue wouldn't crop up. However, when dealing with calculus of variations, one must remember that boundary conditions must be imposed and reducing the order of derivatives weakens the demands on the test and trial functions, allowing them to not need be continuous in their second derivative.

Take for example a general PDE defined,

$$\mathbf{v} \cdot \nabla u + \beta u = \nabla \cdot (\alpha \nabla u) + f, \quad \mathbf{x} \in \Omega \tag{2.22}$$

$$u = u_D, \quad \mathbf{x} \in \Gamma_D \tag{2.23}$$

$$-\alpha \frac{\partial u}{\partial \mathbf{n}} = g, \quad \mathbf{x} \in \Gamma_N, \tag{2.24}$$

multiplying this by a test function v and getting the inner product over the domain Ω, just like was seen in the method of weighted residuals leaves the equation,

$$\int_\Omega (\mathbf{v} \cdot \nabla u + \beta u) v \; d\mathbf{x} = \int_\Omega \nabla \cdot (\alpha \nabla u) v \; d\mathbf{x} + \int_\Omega f v \; d\mathbf{x}. \tag{2.25}$$

Applying Green's lemma to the second-order term then results in the following equation,

$$\int_\Omega \nabla \cdot (\alpha \nabla u) v \; d\mathbf{x} = -\int_\Omega \alpha \nabla u \cdot \nabla v \; d\mathbf{x} + \oint_\Gamma \alpha \frac{\partial u}{\partial \mathbf{n}} v \; ds + \int f v \; d\mathbf{x}. \tag{2.26}$$

Since the boundary integral is 0 on Γ_D and g on Γ_N, it can be rewritten as,

$$\int_\Omega \nabla \cdot (\alpha \nabla u) v \; d\mathbf{x} = -\int_\Omega \alpha \nabla u \cdot \nabla v \; d\mathbf{x} + \oint_{\Gamma_N} g v \; ds + \int f v \; d\mathbf{x}, \tag{2.27}$$

and the entire PDE can be shown in terms of the inner product defined in (REFERENCE),

$$\langle \mathbf{v} \cdot \nabla u, v \rangle + \langle \beta u, v \rangle = -\langle \alpha \nabla u, \nabla v \rangle + \langle g, v \rangle_N + \langle f, v \rangle . \tag{2.28}$$

This PDE has now been transformed into its weak formulation. Looking at the (REF SECTION), it's quite obvious now that u can be approximated by subbing in an estimate, applying the method of weighted residuals and achieving a system of linear equations. Thus the solution space has been reduced to a finite dimension.

2.3.2 Functionals

While regular calculus deals with changes in ordinary variables, calculus of variations by contrast deals with changes in functions - handling special functions known as functionals which take functions as inputs compared to just variables. These are beneficial for the FEM as if we consider the problem posed in the previous section, much of what is trying to be accomplished is finding the function which minimises an integral i.e. minimising a functional. Take for example, the functional

$$F[y(x)] = \int_\Omega f(x, y(x), y'(x)) \; dx, \tag{2.29}$$

in this instance, one would search for what function $y(x)$ would minimise the integral.

Before discussing how to derive these functionals, first look at an abstract notation for variational forms we saw in the previous sections. Consider a function with trial functions u, and test functions v, supported on V, then define the problem as,

$$a(u, v) = L(v), \quad v \in V, \tag{2.30}$$

where $a(u, v)$ is in bilinear form and contains all terms which have both test and trial functions, whereas, $L(v)$ is linear, containing only test functions. This is equivalent to the integrals seen in (REFERENCE) which we wish to minimise. This equation is equivalent to minimising the functional,

$$I[v] = \frac{1}{2}a(u, v) - L(v). \tag{2.31}$$

NEED TO CITE THIS.... In a multidimensional case, consider the PDE,

$$-\nabla(\alpha\nabla u) + \beta u = f, \quad \mathbf{x} \in \Omega \tag{2.32}$$

$$u = u_D, \quad \mathbf{x} \in \Gamma_D \tag{2.33}$$

$$-\alpha\frac{\partial u}{\partial \mathbf{n}} + \sigma(s)u = k, \quad \mathbf{x} \in \Gamma_R \tag{2.34}$$

ADD DERIVATION HERE. The resulting functional is,

$$I[u] = \int_\Omega (\nabla u \cdot \alpha\nabla u) + \beta u^2 - 2uf)\ d\mathbf{x} + \oint_{\Gamma_R} (\sigma u^2 - 2uk)\ ds \tag{2.35}$$

These functionals are an alternative and mathematically equivalent variant on attempting to minimising the problem at hand and a commonly seen in papers discussing the FEM. PAGE 234!!

2.4 Finite Elements

Up to this point, the basis functions that have been used have all been defined across the entire domain Ω. In this section, piecewise polynomial basis functions will be used, defined with compact support across what are known as elements or cells. These cells will contribute their own weighted value to an assembly process, generating an overall linear system $Lu = b$, where L is known as the global stiffness matrix and b the stress vector. This system will result in the solution to the PDE problem at hand.

2.4.1 Cells & Basis Functions

For illustrative purposes, a single dimensional problem is used here to demonstrate the actual concept of the finite elements or cells. Consider now a domain,

$$\Omega = \Omega^{(0)} \cup \Omega^{(1)} \cdots \Omega^{(N_e)}, \tag{2.36}$$

where

$$\Omega^{(i)} \cap \Omega^{(j)} = \forall i, j \in \{0, \ldots, N_e\}. \tag{2.37}$$

Within this domain, define N_n nodes, equally spaced out. Suppose a node has a *global index* $i \in \{0, \ldots, d\}$ and are laid out in no particular order. We define the node's *local index* in cell e as $r \in \{0, \ldots, d\}$, again these do not need to necessarily be in order or equally spaced. No that the local and global numbering of the nodes are defined, we define the function,

$$i = q(e, r), \tag{2.38}$$

where $q(e, r)$ is a mapping function from local index r in cell e, back to the node's global index across the domain. Figure (REFERENCE) demonstrates these definitions. Now, basis functions must be defined across the domain in order to make an approximation for u. Consider a node, globally numbered i, locally numbered r, on cell e, with $d+1$ nodes in said cell, we define its corresponding basis function $\psi_i(x)$ as a Lagrange polynomial of degree d, defined,

$$l_d(x) = \prod_{\substack{0 \leq m \leq k \\ m \neq j}} \frac{x - x_m}{x_j - x_m} \tag{2.39}$$

which is 1 at node r and 0 everywhere else. If the node is internal, than the function is simply defined as is, if the node is shared with a neighbouring cell, then the basis function is a piecewise combination of the Lagrange polynomial from both cells which share the node. The value d is known as the degree of freedom and is related to the order of polynomial used as the basis function - the higher the order, the more nodes per cell. Figures BLAH BLAH illustrate these piecewise polynomials in both 1D and 2D. Now it can be said that

$$u(x) \approx \sum_{\mathcal{I}_s} c_i \psi_i(x), \tag{2.40}$$

where $\mathcal{I}_s$ is the set of indices for which $u(x_i)$ is unknown.

2.4.2 Assembling the Stiffness Matrix

The next and most obvious step to do is a linear system must be assembled from these basis functions in order to solve for these unknowns $\{c)i\}_{\mathcal{I}_s}$ and achieve an approximation for u. It was shown in REFERENCE SECTION, that a variational form can be written in an abstract means form as,

$$a(u, v) = L(v), \quad v \in V. \tag{2.41}$$

With that in mind, consider the PDE (REFERENCE), moving all the terms with both test and trial function to the left-hand side, this can be rewritten as,

$$\langle \mathbf{v} \cdot \nabla u, v \rangle + \langle \beta u, v \rangle + \langle \alpha \nabla u, \nabla v \rangle = \langle g, v \rangle_N + \langle f, v \rangle \,, \tag{2.42}$$

or subbing in (REFERENCE), leaves,

$$\sum_{\mathcal{I}_s} \left(\langle \mathbf{v} \cdot \nabla \psi_j, \psi_i \rangle + \langle \beta \psi_j, \psi_i \rangle + \langle \alpha \nabla \psi_j, \nabla \psi_i \rangle \right) c_j = \langle g, \psi_i \rangle_N + \langle f, \psi_i \rangle \,, \tag{2.43}$$

which clearly demonstrates a linear system,

$$\sum_{\mathcal{I}_s} A_{i,j} c_j = b_i, \tag{2.44}$$

where,

$$A_{i,j} = \langle \mathbf{v} \cdot \nabla \psi_j, \psi_i \rangle + \langle \beta \psi_j, \psi_i \rangle + \langle \alpha \nabla \psi_j, \nabla \psi_i \rangle \tag{2.45}$$

$$b_i = \langle g, \psi_i \rangle_N + \langle f, \psi_i \rangle \,. \tag{2.46}$$

Equation (REFERENCE) should clearly demonstrate now why it was important to convert the PDE into its weak formulation as now we have first-order derivatives of the basis and trial functions which need to be continuous. Had it been left in strong form these would have been second-order.

This has now shown what the overall stiffness matrix will be comprised of but how does one achieve this from the elements and their basis functions? Remembering that these basis functions are compactly supported, it can be seen that $A_{i,j}$ can be assembled by incrementing the integral result from all cells which contain nodes i, j i.e.

$$A_{i,j} \coloneqq A_{q(e,r),q(e,s)} + \tilde{A}^{(e)}_{r,s} \tag{2.47}$$

$$b_i \coloneqq b_{q(e,r)} + \tilde{b}^{(e)}_{r}, \tag{2.48}$$

where $\tilde{A}^{(e)}_{r,s}$ and $\tilde{b}^{(e)}_{r}$ are known as the element matrix and element vector, respectively.

2.4.3 Enforcing Boundary Conditions

2.5 Implemented Example

Chapter 3

GPU Implementations

3.1 Current Approaches

3.2 FEM-SES

3.3 Issues???

In this research we follow

PowerTAC is an example of a multiagent competitive gaming platform

Chapter 4

Results

4.1 Test-bed Architecture

Chapter 5

Conclusion

Appendix A

Appendix

...