PARALLEL CONJUGATE GRADIENTS ON FINITE ELEMENT MATRICES

RAYMOND VAN VENETIË AND JAN WESTERDIEP

1. Introduction

> "The conjugate gradient method (CG) is an algorithm for the numerical solutions of particular systems of linear equations, namely those whose matrix is symmetric and positive definite. The conjugate gradient method is often implemented as an iterative algorithm, applicable to sparse systems that are too large to be handled by a direct implementation or other direct methods such as the Cholesky decomposition. Large sparse systems often arise when numerically solving partial differential equations or optimization problems." [Wik14a]
>
> "The finite element method (FEM) is a numerical technique for finding approximate solutions to boundary value problems for partial differential equations. It uses subdivision of a whole problem domain into simpler parts, called finite elements, and variational methods from the calculus of variations to solve the problem by minimizing an associated error function. Analogous to the idea that connecting many tiny straight lines can approximate a larger circle, FEM encompasses methods for connecting many simple element equations over many small subdomains, named finite elements, to approximate a more complex equation over a larger domain." [Wik14b]

In this report, we will look at a parallel implementation of the conjuate gradient method. We will solve linear systems coming from the finite element method.

To test our implementation, it is useful to have a supply of matrices ready. The University of Florida has a huge collection of sparse matrices. [DH14] We chose to read these matrices in Matrix Market file format[1]. [NIS07]

2. CG

In its most basic (sequential, non-preconditioned) form, CG can be written down as in Algorithm 1. [Bis04, Alg. 4.8] We slightly adapted it from its original form to be able to use the different BLAS routines.

The iterates x_k obtained from the CG algorithm satisfy the following inequality [Sle14, Slide 23]:

$$\frac{\|x - x_k\|_A}{\|x - x_0\|_A} \leq 2\left(\frac{\sqrt{\kappa_2(A)} - 1}{\sqrt{\kappa_2(A)} + 1}\right)^k \leq 2\exp\left(-\frac{2k}{\sqrt{\kappa_2(A)}}\right)$$

where $\kappa_2(A)$ is the 2-condition number of A, which for symmetric positive definite matrices equates

$$\kappa_2(A) = \frac{\lambda_{max}}{\lambda_{min}}.$$

It is therefore of interest to create a condition number that is as low as possible.

[1]The reason for this is pure laziness: it is possible to download these matrices in this format and the creators of the Matrix Market supplies a I/O C library.

2.1. **Preconditioned CG.** A preconditioner P of a matrix A is a matrix such that $P^{-1}A$ has a smaller condition number than A. As the theoretical convergence rate is highly dependent on the condition number, we can improve this using such a preconditioner. Instead of solving $Ax = b$, we will solve $P^{-1}Ax = P^{-1}b$. Blablabla

3. FEM matrices

4. Parallellizing CG

5. Using sparsity of matrices in parallel CG

6. hoe heette het ook alweer als je na een kleine verandering in je matrix opnieuw ging optimizen

7. Future work

7.1. **Adaptive FEM.**

References

[Bis04] Rob H. Bisseling. *Parallel Scientific Computation: A Structured Approach using BSP and MPI*. Oxford University Press, Oxford, UK, March 2004.

[DH14] Tim Davis and Yifan Hu. The University of Florida Sparse Matrix Collection. `http://www.cise.ufl.edu/research/sparse/matrices`, 2014.

[NIS07] NIST. Matrix Market. `http://math.nist.gov/MatrixMarket`, 2007.

[Sle14] Gerard L.G. Sleijpen. Numerical linear algebra — krylov methods for hermitian systems. `http://www.staff.science.uu.nl/~sleij101/Opgaven/NumLinAlg/Transparancies/Lecture7.pdf`, 2014.

[Wik14a] Wikipedia. Conjugate gradient method — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Conjugate_gradient_method`, 2014.

[Wik14b] Wikipedia. Finite element method — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Finite_element_method`, 2014.

Algorithm 1 Sequential CG

```
1: function CG(k_max, ε, n, A[n][n], b[n], x_0[n], x[n])
2:     int k := 0;
3:     float r[n];
4:     float ρ;
5:     float ρ_old;
6:     float nbsq;
7:
8:     x ← x_0;
9:     r ← b;
10:    r ← r − Ax;
11:    ρ ← ⟨r, r⟩;
12:    nbsq ← ⟨b, b⟩;
13:
14:    while ρ > ε^2 · nbsq ∧ k < k_max do
15:        float p[n];
16:        float w[n];
17:        float α;
18:        float β;
19:        float γ;
20:
21:        if k == 0 then
22:            p ← r;
23:        else
24:            β ← ρ/ρ_old;
25:            p ← βp;
26:            p ← r + p;
27:        end if
28:
29:        w ← Ap;
30:        γ ← ⟨p, w⟩;
31:        α ← ρ/γ;
32:        x ← x + αp;
33:        r ← r − αw;
34:        ρ_old ← ρ;
35:        ρ ← ⟨r, r⟩;
36:
37:        k ← k + 1;
38:    end while
39: end function
```