Finite-choice algorithm optimization in Conjugate Gradients*

Jack Dongarra and Victor Eijkhout†

January 2003

Abstract

We present computational aspects of mathematically equivalent implementations of the Conjugate Gradient method. Since the implementations have different performance characteristics, but compute the same quantities, an adaptive implementation can at run time pick the optimal choice.

1 Introduction

Recent times have seen the emergence of software that dynamically adapts itself to its computational environment. One example is ATLAS [9], where upon installation a generator program produces and times many equivalent implementations of Blas level 3 routines, keeping the most efficient one. This process consumes time – up to several hours – but only once per installation; linking to the generated library can give several factors improvement in the performance of a code.

Here we investigate the issue of optimizing Conjugate Gradient methods. The choice between different but mathematically equivalent implementations can be made at run-time rather than at installation-time as in the the ATLAS case. Another important difference between our case and ATLAS is that the search space in Atlas is multi-dimensional, and with such a degree of freedom along each dimension, that we can essentially consider it infinite. The choices among different implementations of CG, by contrast, are very finite. They can be realized by simple switches in one overarching implementation. What is more, since all implementations compute the same quantities, switching between one implementation and another can be done during a run, without restarting.

The variants of CG we consider are all geared towards optimization in the case of parallel execution. Several authors [1, 2, 4, 6] have proposed numerically equivalent implementations of the Conjugate Gradient method that rearrange the inner product

*Lapack Working Note 159, University of Tennessee Computer Science report ut-cs-03-502

†Innovative Computing Lab, Department of Computer Science, University of Tennessee, Knoxville TN 37996

computations, usually at the expense of some extra computation. The new arrangements of inner products are such that all inner products in a single iteration can combine their communication stages. On parallel architectures with a large latency this can give appreciably savings in overall execution time. We also consider an implementation of IC-preconditioned CG by van der Vorst [3] which overlaps one inner product with half of the preconditioner application, effectively hiding its communication time.

Since all these methods compute the same scalar and vector quantities in each iteration, a dynamic implementation can evaluate performance characteristics of the architecture during the first few iterations, and finish the process with what is perceived to be the optimal method for the given input parameters.

Our choice of CG is no intrinsic limitation of the techniques described; the inner product elimination strategies of [1, 2, 4, 6] can be applied immediately to BiCG (and hence to QMR) and with slightly more effort to such methods as BiCGstab.

2 Variants of Conjugate Gradients

In this section we consider several rearrangement strategies for the Conjugate Gradient method.

The Conjugate Gradient method in its original form [5] has two inter-dependent inner products: each inner product is used to compute a vector, which in turn is needed for the other inner product. In a parallel context, each inner product can potentially take a time disproportionate to its scalar operation count, and the thought of rearranging the method to combine the inner products is then a naturaly one.

The Saad/Meurant, Chronopoulos/Gear, and Eijkhout variants of CG below proceed by combining inner products; they involve more scalar operations than the original method. Since all methods have the same convergence speed in terms of numbers of iterations, on a single processor these variants clearly offer no advantage over the classical formulation. On a large number of processors and a high-latency network they will probably offer an advantage, offsetting the extra operations with decreased communication time.

2.1 The Saad/Meurant method

A first proposed variant by Saad [7] uses

$$r_{i+1}^t M^{-1} r_{i+1} + r_i^t M^{-1} r_i = \alpha_i^2 (Ap_i)^t M^{-1} (Ap_i), \tag{1}$$

so the term $r_{i+1}^t M^{-1} r_{i+1}$ can be computed recursively without the need for an inner product, and before the vector r_{i+1} is actually constructed. Instead of computing $r_i^t M^{-1} r_i$, we compute $(Ap_i)^t M^{-1} (Ap_i)$ at the same time we compute $p_i^t Ap_i$. The vector $M^{-1} r_i$ is now computed by an extra recurrence

$$M^{-1} r_{i+1} = M^{-1} r_i - \alpha_i M^{-1} Ap_i.$$

However, this method illustrates the potential dangers of rearranging operations: the variant methods are not guaranteed to be as stable as the original. Indeed, this method is unstable in many cases.

The first variant we actually consider here for implementation is a modification by Meurant [6] of Saad's method, and this has in practice been shown to be as stable as the original method. Meurant proposed to use equation (1) only as a predictor and used in β_i; after r_{i+1} and $M^{-1}r_{i+1}$ have been constructed, the value of $r_{i+1}^t M^{-1} r_{i+1}$ is computed as an explicit inner product – to be used in α_{i+1} – combined with the other two inner products. We give the structure of the algorithm in figure 1.

- From p_i form Ap_i and $M^{-1}Ap_i$.
- Now compute simultaneously the inner products

$$p_i^t Ap_i, \qquad r_i^t M^{-1} r_i, \qquad \text{and} \qquad (Ap_i)^t M^{-1}(Ap_i);$$

- with $r_i^t M^{-1} r_i$ compute $\alpha_i = p_i^t Ap_i / r_i^t M^{-1} r_i$, and the value of the inner product $r_{i+1}^t M^{-1} r_{i+1} = -r_i M^{-1} r_i + \alpha_i^2 (Ap_i)^t M^{-1}(Ap_i)$. With this, also compute $\beta_i = r_{i+1} M^{-1} r_{i+1} / r_i M^{-1} r_i$. Save the vector $M^{-1}Ap_i$.
- Update $r_{i+1} = r_i - \alpha_i Ap_i$ and $M^{-1}r_{i+1} = M^{-1}r_i - \alpha_i M^{-1}Ap_i$.
- Update $p_{i+1} = M^{-1}r_{i+1} - \beta_i p_i$.

Figure 1: Meurant's modification of Saad's CG method.

2.2 The Chronopoulos and Gear method

The second variant we consider was published by Chronopoulos and Gear [1] and eliminates the other, $p^t Ap$, inner product. This method was later independently rediscovered by D'Azevedo, Eijkhout, and Romine[2], who proved that it is as stable as the original CG method.

The basic relation for this variant,

$$p_i^t Ap_i = r_i^t M^{-t} A M^{-1} r_i - \beta_i^2 p_{i-1}^t Ap_{i-1}, \tag{2}$$

replaces the $p^t Ap$ inner product by $r^t M^{-t} A M^{-1} r$, which can be computed combined with the $r^t M^{-1} r$ inner product. Also, we now construct Ap_i recursively from

$$Ap_{i+1} = AM^{-1}r_{i+1} + \sum_{k=1}^{i} Ap_k u_{ki+1}$$

rather than explicitly by a matrix-vector product. We give the structure of the algorithm in figure 2.

2.3 The Eijkhout variant

Eijkhout has proposed [2, 4] a method that is very similar to the Chronopoulos and Gear one. It is also based on recursive calculation of $p^t Ap$ from an actually constructed inner

- From r_i and $M^{-1}r_i$ compute simultaneously

$$r_i^t M^{-1} r_i \qquad \text{and} \qquad (M^{-1}r_i)^t A M^{-1} r_i;$$

- with this, compute $\beta_i = r_{i+1}M^{-1}r_{i+1}/r_i M^{-1} r_i$.
- Recursively compute $p_i^t A p_i$ from $r_i^t M^{-t} A M^{-1} r_i$ and $p_{i-1}^t A p_{i-1}$ from

$$p_i^t A p_i = r_i^t M^{-t} A M^{-1} r_i - \beta_i^2 p_{i-1}^t A p_{i-1},$$

- With $p_i^t A p_i$ compute $\alpha_i = p_i^t A p_i / r_i^t M^{-1} r_i$, and use this to update x_i and r_i to x_{i+1} and r_{i+1}.
- Apply the preconditioner to form $M^{-1}r_{i+1}$, then the matrix to form $AM^{-1}r_{i+1}$.
- Now construct Ap_{i+1} recursively by

$$Ap_{i+1} = AM^{-1}r_{i+1} + Ap_i\beta_i$$

Figure 2: Chronopoulos and Gear's method

product $(M^{-1}r)^t A(M^{-1}r)$. Equation (2) is then replaced by

$$p_i^t A p_i = r_i^t M^{-t} A M^{-1} r_i + \beta_i (M^{-1}r_{i-1})^t (Ap_{i-1}), \tag{3}$$

This requires the recursive construction of the Ap_i vectors from the actual matrix-vector product $AM^{-1}r_i$. Additionally, there is now a third inner product $(M^{-1}r_i)^t(Ap_i)$ in each iteration. This inner product can be combined with the already existing ones, so there is only an increase in the scalar work per iteration, not in the communication behaviour. There is no stability analysis for this method, but in tests it has appeared as stable – or better – as the Chronopoulos/Gear variant.

2.4 Van der Vorst's method

Finally, for the special case of preconditioning by a Block Jacobi method with an Incomplete Cholesky solve of the blocks we consider a method by van der Vorst [3] which overlaps one inner product computation with the second half of the Cholesky solve.

The crucial observation here is that, in the presence of a Cholesky preconditioner $M = LL^t$, the inner product $r^t M^{-1} r$ can be computed as $(L^{-1}r)^t(L^{-1}r)$, that is, before the preconditioned residual $M^{-1}r$ has been formed completely. The method then is

construct $s = L^{-1}r$
calculate $z = L^{-t}s$, and simultaneously

$z^t r = s^t s.$

This method is by heuristic reasoning exactly as stable as the original method; in fact, since it uses symmetry to a larger extent than the original method it may in fact be more stable. While the other methods presented here can be generalized beyond CG, for instance to the BiCG method, this method essentially relies on the symmetry of the preconditioner and on Arnoldi orthogonalization, as opposed to Lanczos orthogonalization in the BiCG method. Thus, no generalization to other methods than CG offers itself. However, by combining van der Vorst's overlapping trick with the Saad/Meurant method, we find a variant of CG where *all* inner product communications can be hidden.

2.5 Norm calculation

For all methods except van der Vorst's and Meurant's – note that this include the original formulation – communication latency can be eliminated in the norm calculation that is performed for the convergence test. These CG methods feature the following lines:

calculate $\|r\|$;
and possibly terminate the iteration
calculate $z = M^{-1}r$
calculate $z^t r$

Rearranging this as

calculate $z = M^{-1}r$
calculate combined $\|r\|$ and $z^t r$;
and possibly terminate the iteration

combines the norm and inner product calculations, but in the final iteration one superfluous inner product is computed. This variant may be an improvement if the savings in the inner product computation outweighs the loss because of the extra inner product. This will often be the case, but not when

- there is no savings in the inner products, because the communications cost are negligible, or
- the preconditioner is relatively expensive and the number of iterations low.

Of course, if the stopping test is on $z^t r$ there is no separate norm calculation; with a test on $\|z\|$ the norm calculation can be combined with the inner product.

2.6 Block orthogonalization of residuals

The orthogonal residuals of the CG method are linear combinations of a Krylov sequence based on the first residual. Thus, communication latency could be lessened by computing explicitly a number s of Krylov vectors, and orthogonalizing them *en*

bloc [1, 8]. Its clearest disadvantage is instability that grows with the number of Krylov vectors computed.

Computational benefits are a mixed story:

- If all inner products are computed simultaneously, in essence using a classic Gram-Schmidt method for orthogonalization, the influence of latency is greatly diminished.
- The QR factorization used for the Gram-Schmidt orthogonalization is a Blas Level 3 kernel, hence more efficient in execution than the inner products in the original method.
- On the other hand, while in CG a residual is only orthogonalized against the previous two, in this method it has to be orthogonalized against all $s-1$ other vectors. Hence, the scalar computation cost is higher than of the original method: $O(s^2)$ inner products as opposed to $O(s)$.

We expect that this method may be advantageous on machines that have a large difference between Blas levels 1 and 3 performance.

References

[1] CHRONOPOULOS, A., AND GEAR, C. s-step iterative methods for symmetric linear systems. *Journal of Computational and Applied Mathematics 25* (1989), 153–168.

[2] D'AZEVEDO, E., EIJKHOUT, V., AND ROMINE, C. Lapack working note 56: Reducing communication costs in the conjugate gradient algorithm on distributed memory multiprocessor. Tech. Rep. CS-93-185, Computer Science Department, University of Tennessee, Knoxville, 1993.

[3] DEMMEL, J., HEATH, M., AND VAN DER VORST, H. Parallel numerical linear algebra. In *Acta Numerica 1993*. Cambridge University Press, Cambridge, 1993.

[4] EIJKHOUT, V. Lapack working note 51: Qualitative properties of the conjugate gradient and lanczos methods in a matrix framework. Tech. Rep. CS 92-170, Computer Science Department, University of Tennessee, 1992.

[5] HESTENES, M., AND STIEFEL, E. Methods of conjugate gradients for solving linear systems. *Nat. Bur. Stand. J. Res. 49* (1952), 409–436.

[6] MEURANT, G. Multitasking the conjugate gradient method on the CRAY X-MP/48. *Parallel Computing 5* (1987), 267–280.

[7] SAAD, Y. Practical use of polynomial preconditionings for the conjugate gradient method. *SIAM J. Sci. Stat. Comput. 6* (1985), 865–881.

[8] VAN ROSENDALE, J. Minimizing inner product data dependencies in conjugate gradient iteration. Tech. Rep. 172178, ICASE, NASA Langley Research Center, Hampton, Virginia, 1983.

Classical	Saad/Meurant	Chronopoulos/Gear	Eijkhout
Norm calculation:			
$\boxed{error = \sqrt{r^t r}}$			
Preconditioner application:			
$z \leftarrow M^{-1} r$	$z \leftarrow z - \alpha q$	$z \leftarrow M^{-1} r$	id
Matrix-vector product:			
		$az \leftarrow A \times z$	id
Inner products 1:			
$\boxed{\rho \leftarrow z^t r}$	$\rho_{\rm predict} \leftarrow -\rho_{\rm true} + \alpha^2 \mu$	$\boxed{\begin{array}{c} error = \sqrt{r^t r} \\ \rho \leftarrow z^t r \\ \zeta \leftarrow z^t az \end{array}}$	$\boxed{\begin{array}{c} error = \sqrt{r^t r} \\ \rho \leftarrow z^t r \\ \zeta \leftarrow z^t az \\ \epsilon \leftarrow (M^{-1} r)^t (Ap) \end{array}}$
$\beta \leftarrow \rho / \rho_{\rm old}$	$\beta = \rho_{\rm predict} / \rho_{\rm old}$	$\beta \leftarrow \rho / \rho_{\rm old}$	id
Search direction update:			
$p \leftarrow z + \beta p$	id	id	id
Matrix-vector product:			
$ap \leftarrow A \times p$	id	$ap \leftarrow az + \beta ap$	id
Preconditioner application:			
	$q \leftarrow M^{-1} ap$		
Inner products 2:			
$\boxed{\pi \leftarrow p^t ap}$	$\boxed{\begin{array}{c} \pi \leftarrow p^t ap \\ \mu \leftarrow ap^t q \\ error = \sqrt{r^t r} \\ \rho_{\rm true} = z^t r \end{array}}$	$\pi \leftarrow \zeta - \beta^2 \pi$	$\pi \leftarrow \zeta + \beta \epsilon$
$\alpha = \rho / \pi$	$\ldots \rho_{\rm true} \ldots$	$\alpha = \rho / \pi$	
Residual update:			
$r \leftarrow r - \alpha Ap$	id	id	id
3 separate inner products	4 combined	3 combined	4 combined
	1 extra vector update	id	id

Figure 3: Structure of the inner loop of the CG variants

[9] WHALEY, R. C., PETITET, A., AND DONGARRA, J. J. Automated empirical optimization of software and the ATLAS project. *To appear in Parallel Computing* (2001). Also available as University of Tennessee LAPACK Working Note #147, UT-CS-00-448, 2000 (www.netlib.org/lapack/lawns/lawn147.ps).