NASA Contractor Report 172178

ICASE

MINIMIZING INNER PRODUCT DATA DEPENDENCIES IN CONJUGATE GRADIENT ITERATION

John Van Rosendale

Contract Nos. NAS1-17070, NAS1-17130
July 1983

INSTITUTE FOR COMPUTER APPLICATIONS IN SCIENCE AND ENGINEERING
NASA Langley Research Center, Hampton, Virginia 23665

Operated by the Universities Space Research Association

National Aeronautics and Space Administration

Langley Research Center
Hampton, Virginia 23665

MINIMIZING INNER PRODUCT DATA DEPENDENCIES IN CONJUGATE GRADIENT ITERATION

John Van Rosendale
Institute for Computer Applications in Science and Engineering

ABSTRACT

The amount of concurrency available in conjugate gradient iteration is limited by the summations required in the inner product computations. The inner product of two vectors of length N requires time c*log(N), if N or more processors are available.

This paper describes an algebraic restructuring of the conjugate gradient algorithm which minimizes data dependencies due to inner product calculations. After an initial start up, the new algorithm can perform a conjugate gradient iteration in time c*log(log(N)).

Research supported by the National Aeronautics and Space Administration under NASA Contract Nos. NAS1-17070 and NAS1-17130 while the author was in residence at ICASE, NASA Langley Research Center, Hampton, VA 23665.

Introduction

Conjugate gradient iteration is a method of linear equation solution of great practical importance. See, for example, Hestenes and Steifel [4], Concus, Golub and O'Leary [3], or Chandra [2]. It can be used to solve any linear system

$$Au = b$$

where A is symmetric, positive definite, and can be quite efficient when coupled with various preconditioning techniques. However, CG (conjugate gradient) iteration involves the computation of inner products at every iteration. On parallel computers with large numbers of processors, the data dependencies inherent in these inner product calculations will limit the speed of conjugate gradient iteration for large sparse linear systems. This is pointed out in Schreiber[1] and Adams [1982]. In fact, given sufficiently many processors, the summation fan-ins in the inner product calculations will dominate the computation time on nearly all large sparse linear systems occurring in practice.

2. Conjugate Gradient Iteration

This paper presents a solution to this problem through an algebraic restructuring of the CG Algorithm. Consider first the standard CG iteration. One of a number of mathematically equivalent forms of it may be given as follows:

---

[1] Schrieber, R. 1983, Stanford University, California, personal communication; Schrieber, R., 1981, SIAM J. Sci. Statist. Comput., to be published.

$u^{(0)}$ arbitrary,

$$u^{(n+1)} = u^{(n)} + \lambda_n p^{(n)}, \qquad n=0,1,\cdots,$$

$$p^{(n)} = \begin{cases} r^{(n)}, & n=0, \\ r^{(n)} + \alpha_n p^{(n-1)}, & n=1,2,\cdots, \end{cases}$$

$$r^{(n)} = r^{(n-1)} - \lambda_{n-1} Ap^{(n-1)}, \qquad n=1,2,\cdots,$$

$$\alpha_n = \frac{(r^{(n)},r^{(n)})}{(r^{(n-1)},r^{(n-1)})}, \qquad n=1,2,\cdots,$$

$$\lambda_n = \frac{(r^{(n)},r^{(n)})}{(p^{(n)},Ap^{(n)})}, \qquad n=0,1,\cdots.$$

The data dependencies here are severe. One cannot generate $(r^{(n)},r^{(n)})$ until α_{n-1} and λ_{n-1} are known. But these quantities involve inner products dependent on $r^{(n-1)}$. As pointed out above, an inner product on vectors of length N requires time c*log(N). Thus it would seem that a CG iteration could not be done faster than in time c*log(N).

3. Idea of New Algorithm

This natural seeming idea, that a CG iteration on vectors of length N cannot be done faster than in time c*log(N), turns out to be incorrect. To see why, consider the computation of a typical inner product required,

$$(r^{(n)},r^{(n)}).$$

By the formulas above, $r^{(n)}$ is given as

$$r^{(n)} = r^{(n-1)} - \lambda_{n-1} Ap^{(n-1)}.$$

Now suppose we know $r^{(n-1)}$ and $p^{(n-1)}$ but not the value of λ_{n-1}. In this case we would be unable to evaluate $(r^{(n)},r^{(n)})$, but we could still perform most of the work involved in evaluating this inner product. Specifically, we can write the recurrence

$$(r^{(n)},r^{(n)}) = (r^{(n-1)},r^{(n-1)})$$

$$- 2\lambda_{n-1}(r^{(n-1)},Ap^{(n-1)})$$

$$+ \lambda_{n-1}^2(Ap^{(n-1)},Ap^{(n-1)}),$$

and can proceed to evaluate all inner products on the right here. If subsequently someone told us the value of λ_{n-1} we could compute the value of $(r^{(n)},r^{(n)})$ very rapidly, since only a few more real operations would then be needed to complete evaluation of the recurrence relation.

It is easy to see how this idea can be used to speed the computation of the CG algorithm on parallel computers. We have replaced an inner product computation requiring data not present until iteration n with inner products of vectors present at iteration n-1. Since these vectors are present sooner, we have that much longer to perform their inner products, to achieve the same parallel computation speed. Stated differently, assuming only the inner products limit the speed of the computation, the use of this recurrence relation for $(r^{(n)},r^{(n)})$ and the analogous relation for $(p^{(n)},Ap^{(n)})$ will approximately double the parallel speed of CG iteration, where it is assumed that sufficiently many processors are available and that communications cost can be neglected.

4. Recurrence Relations

The recurrence relation just described is one of a large class of such relations which can be exploited to speed up CG iteration. These relations will be given in detail in a future paper, but for now we consider only the general form of such recurrence relations. Consider the typical inner product:

$$(r^{(n)},r^{(n)})$$

The value of this inner product may be given in terms of the values of inner products of vectors occurring at any previous iteration together with the values of the real parameters

$$\alpha_{n-1},\alpha_{n-2},\cdots,$$

$$\lambda_{n-1},\lambda_{n-2},\cdots.$$

For example, for any $k > 0$, one can derive recurrence relations of the form

$$\begin{aligned}(r^{(n)},r^{(n)}) = &\sum_{i=0}^{2k} a_i(r^{(n-k)},A^i r^{(n-k)}) \\ &+ \sum_{i=0}^{2k} b_i(r^{(n-k)},A^i p^{(n-k)}) \qquad (*) \\ &+ \sum_{i=0}^{2k} c_i(p^{(n-k)},A^i p^{(n-k)}).\end{aligned}$$

The coefficients $\{a_i\}$, $\{b_i\}$, $\{c_i\}$ occurring here are polynomials in the parameters

$$\{\alpha_{n-1},\alpha_{n-2},\cdots,\alpha_{n-k},\lambda_{n-1},\lambda_{n-2},\cdots,\lambda_{n-k}\}.$$

Similar recurrence relations are available for the other type of inner product occurring in CG iteration, $(p^{(n)},Ap^{(n)})$.

5. New Algorithm

To construct a more parallel variant of CG iteration based on these recurrence relations, one begins by selecting a value for the constant k, which may be thought of as a look-ahead parameter. Then at iteration $n-k$, when vectors $r^{(n-k)}$ and $p^{(n-k)}$ become available one begins forming all of the inner products

$$(r^{(n-k)},A^i r^{(n-k)}), \quad i=0,1,\cdots,2k,$$

$$(r^{(n-k)},A^i p^{(n-k)}), \quad i=0,1,\cdots,2k,$$

$$(p^{(n-k)},A^i p^{(n-k)}), \quad i=0,1,\cdots,2k.$$

The values of these inner products are needed in the recurrence relations for the inner products

$$(r^{(n)},r^{(n)}),\ (p^{(n)},Ap^{(n)})$$

at iteration n. Thus we arrive at an algorithm whose data movements are sketched in Figure 1.

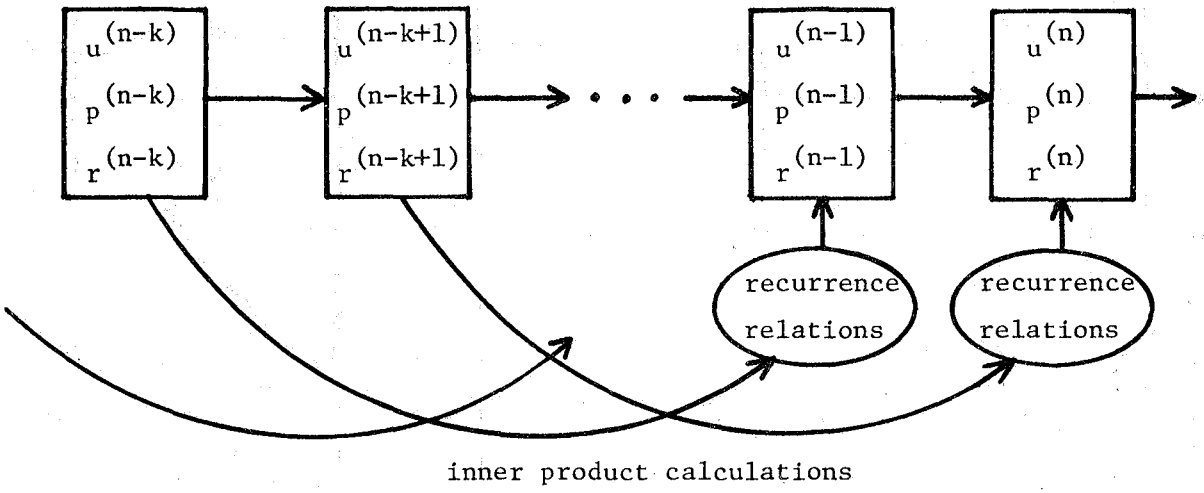

Figure 1. Principal Data Movement in New CG Algorithm.

Clearly the problems of the delays caused by the summations in the inner products is now solved. If we chose $k = \log(N)$, the inner product summation delays will have no impact on algorithm speed. However, two new issues now arise. First, we have not dealt with the way in which the parameters

$$\{\alpha_{n-1},\alpha_{n-2},\alpha_{n-k},\cdots\lambda_{n-1},\lambda_{n-2},\cdots\lambda_{n-k}\}$$

enter into the recurrence relations. In principle, there could be severe data dependencies here. Second, there seem to be a large number of inner products required now, most involving a relatively high power of the matrix A.

Neither of these problems is as serious as it first appears. For the first, it turns out the coefficients $\{a_i\}$ $\{\alpha_i\}$ $\{c_i\}$ in the recurrence relations above are polynomials in the parameters

$$\{\alpha_{n-1},\alpha_{n-2},\cdots\alpha_{n-k},\lambda_{n-1},\lambda_{n-2},\cdots\lambda_{n-k}\}$$

which are at most quadratic in each parameter separately. This fact, coupled with the observation that the parameters

$$\alpha_{n-k},\alpha_{n-k+1},\cdots,\lambda_{n-k},\lambda_{n-k+1},\cdots$$

gradually become available, enables us to effectively perform the coefficient evaluations in a pipelined fashion. Thus at iteration n, when we need the inner product $(r^{(n)},r^{(n)})$, we can have the recurrence relation (*) completely evaluated, except for performing the summations, or the analogous summations in the recurrence for $(p^{(n)},Ap^{(n)})$. This requires parallel time

$$\log(k) = \log(\log(N)).$$

The second problem mentioned above, the occurrence of high powers of the matrix A in the recurrence relation (*), can be resolved by the use of additional recurrence relations. First, observe that there is no need to compute powers of the matrix A, since we have the recurrences:

$$A^i r^{(n)} = A^i r^{(n-1)} - \lambda_{n-1} A^{i+1} p^{(n-1)},$$

$$A^i p^{(n)} = A^i r^{(n)} + \alpha_n A^i p^{(n-1)}.$$

Thus the set of vectors $\{A^i p^{(n)}\}_{i=0}^{k}$ and $\{A^i r^{(n)}\}_{i=0}^{k}$ can be updated with only one matrix vector product.

Next observe that nearly all of the inner products needed can also be obtained by recurrences. We have

$$\left(r^{(n)},A^i r^{(n)}\right) = \left(r^{(n-1)},A^i r^{(n-1)}\right)$$

$$- 2\lambda_{n-1}\left(r^{(n-1)},A^{i+1} p^{(n-1)}\right)$$

$$+ \lambda_{n-1}^2\left(p^{(n-1)},A^{i+2} p^{(n-1)}\right),$$

and similar recurrences for the other types of inner products occurring in relation (*). Given the values of the inner products

$$\left\{r^{(n)},A^i r^{(n)}\right\}_{i=0}^{2k},$$

$$\left\{r^{(n)},A^i p^{(n)}\right\}_{i=0}^{2k},$$

$$\left\{p^{(n)},A^i p^{(n)}\right\}_{i=0}^{2k},$$

at iteration n, we can obtain nearly all of the inner products needed at iteration n+1. Only two inner products need to be computed directly.

6. Computational Complexity

As pointed out above, the summations in the recurrence relations (*) require time

$$\log(k) = \log(\log(N)).$$

Thus if matrix A has at most d nonzeroes per row or column, this algorithm requires parallel time

$$\max\left(\log(d),\log(\log(N))\right).$$

The sequential complexity of this algorithm is essentially the same as that of the usual CG algorithm; we still need two inner products and a matrix vector product at every iteration.

REFERENCES

[1] Adams, L. [1982]. "Iterative Algorithms for Large Sparse Linear Systems on Parallel Computers," NASA Contractor Report 166027, NASA Langley Research Center.

[2] Chandra, R. [1978]. "Conjugate Gradient Methods for Partial Differential Equations," Ph.D. Thesis, Research Report #129, Department of Computer Science, Yale University.

[3] Concus, P., Golub, G. and O'Leary, D. [1976]. "A Generalized Conjugate Gradient Method for the Numerical Solution of Elliptic Partial Differential Equations," Sparse Matrix Computations, eds. J. Bunch, D. Rose, Academic Press, pp. 309-332.

[4] Hestenes, M., and Stiefel, E. [1952]. "Methods of Conjugate Gradients for Solving Linear Systems," J. Res. Nat. Bur. Std., pp. 409-426.

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-172178		

4. Title and Subtitle	5. Report Date
Minimizing Inner Product Data Dependencies in Conjugate Gradient Iteration	July 1983
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
John Van Rosendale	83-36
	10. Work Unit No.

9. Performing Organization Name and Address	11. Contract or Grant No.
Institute for Computer Applications in Science and Engineering Mail Stop 132C, NASA Langley Research Center Hampton, VA 23665	NAS1-17070 NAS1-17130
	13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address	Contractor report
National Aeronautics and Space Administration Washington, D.C. 20546	14. Sponsoring Agency Code

15. Supplementary Notes

Langley Technical Monitor: Robert H. Tolson
Final Report

16. Abstract

The amount of concurrency available in conjugate gradient iteration is limited by the summations required in the inner product computations. The inner product of two vectors of length N requires time c*log(N), if N or more processors are available.

This paper describes an algebraic restructuring of the conjugate gradient algorithm which minimizes data dependencies due to inner product calculations. After an initial start up, the new algorithm can perform a conjugate gradient iteration in time c*log(log(N)).

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Conjugate gradient parallel computation immer products	61 Computer Programming and Software Unclassified-Unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	12	A02

End of Document