Parallelism in Structured Newton Computations

Thomas F. Coleman and Wei Xu

Department of Combinatorics and Optimization
University of Waterloo
Waterloo, Ontario, Canada. N2L 3G1
E-mail: tfcoleman@uwaterloo.ca
E-mail: wdxu@math.uwaterloo.ca

Many vector-valued functions, representing expensive computation, are also structured computations. A structured Newton step computation can expose useful parallelism in many cases. This parallelism can be used to further speed up the overall computation of the Newton step.

1 Introduction

A fundamental computational procedure in practically all areas of scientific computing is the calculation of the Newton step (in n-dimensions). In many cases this computation represents the dominant cost in the overall computing task. Typically the Newton step computation breaks down into two separable subtasks: calculation of the Jacobian (or Hessian) matrix along with the right-hand-side, and then the solution of a linear system (which, in turn, may involve a matrix factorization). Both subtasks can be expensive though in many problems it is the first, calculation of the function and derivative matrices, that dominates.

In most cases when the Newton step computation is relatively expensive, the function that yields the Newton system is itself a 'structured' computation. A structured computation is one that breaks down into a (partially ordered) straight-line sequence of (accessible) macro computational subtasks. For example, if F is a function that is computed by evaluating the sequence F_1, F_2, F_3, in order, then F is a structured computation. The general structured situation can be described as follows: F is a structured computation, $z = F(x)$, if F is evaluated by computing a (partially-ordered) sequence of intermediate vectors y defined below:

$$\left.\begin{array}{rl} \text{Solve for } y_1 & : F_1^E(x, y_1) = 0 \\ \text{Solve for } y_2 & : F_2^E(x, y_1, y_2) = 0 \\ \vdots & \vdots \\ \text{Solve for } y_p & : F_p^E(x, y_1, y_2, \cdots, y_p) = 0 \\ \text{“Solve” for output } z & : z - F_{p+1}^E(x, y_1, y_2, \cdots, y_p) = 0 \end{array}\right\}. \tag{1}$$

For convenience define

$$F^E(x, y_1, \cdots, y_p) = \begin{pmatrix} F_1^E(x, y_1) \\ F_2^E(x, y_1, y_2) \\ \vdots \\ F_p^E(x, y_1, \cdots, y_p) \\ F_{p+1}^E(x, y_1, \cdots, y_p) \end{pmatrix}.$$

The Newton process for (1) can be written,

$$J^E \begin{pmatrix} \delta x \\ \delta y_1 \\ \delta y_2 \\ \vdots \\ \delta y_p \end{pmatrix} = -F^E = \begin{pmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ -F \end{pmatrix}, \tag{2}$$

where the square Jacobian matrix J^E is a block lower-Hessenberg matrix:

$$J^E = \left(\begin{array}{c|cccc} \frac{\partial F_1}{\partial x} & \frac{\partial F_1}{\partial y_1} & & & \\ \frac{\partial F_2}{\partial x} & \frac{\partial F_2}{\partial y_1} & \frac{\partial F_2}{\partial y_2} & & \\ \vdots & \vdots & \vdots & \ddots & \\ \frac{\partial F_p}{\partial x} & \frac{\partial F_p}{\partial y_1} & \cdots & \cdots & \frac{\partial F_p}{\partial y_p} \\ \hline \frac{\partial F_{p+1}}{\partial x} & \frac{\partial F_{p+1}}{\partial y_1} & \cdots & \cdots & \frac{\partial F_{p+1}}{\partial y_p} \end{array} \right). \tag{3}$$

It has been illustrated[2,3] that by exploiting the structure illustrated in (1), it is possible to compute the Newton step, at any point x, significantly faster than by following the standard 2-step procedure: form the Jacobian (Hessian) matrix of F, and then solve a linear system. The key insight is that the Jacobian matrix of the larger system illustrated in (1) is typically sparse and *thus can be computed much more cheaply than the (possibly) dense Jacobian matrix of* F. Moreover, given that the Jacobian matrix has been computed, it is possible to compute the Newton step to the original system $F(x) = 0$, by working directly with the large (sparse) matrix, and possibly avoiding the formulation of $J(x)$, the Jacobian matrix of F.

In this paper we show that these structural ideas also expose parallelism which can be used to further speed up the computation of the Newton step. The rest of the paper is organized as follows. Two extremal examples for exposing parallelism are studied in Section 2. One is the generalized partially separable problem, which is the best case for the parallelism. The other example is the composite function of a dynamic system, which is the worst case. Generally, most problems are somewhere between the above two cases. In Section 3, we construct an example for the structured general case and expose the parallelism to speed up the Newton step. Finally, conclusions are given in Section 4.

2 Two extremal cases

Many practical problems can be covered by the structural notions presented in Section 1. Here, we describe two examples. These two examples represent the two extreme cases. The first example, a generalized partially separable problem, yields a set of decoupled computations, which is the best for parallelism. The second example is a composite function of a dynamic system, representing a recursive computation.

A square generalized partial separable (GPS) function is a vector mapping $F : \mathbb{R}^n \to \mathbb{R}^n$. The evaluation of $z = F(x)$ may involve the following steps:

$$\left. \begin{array}{l} \text{Solve for } y_i : \ y_i - T_i(x) = 0 \quad i = 1, \cdots, p \\ \text{Solve for } z : \ z - \bar{F}(y_1, \cdots, y_p) = 0. \end{array} \right\}, \tag{4}$$

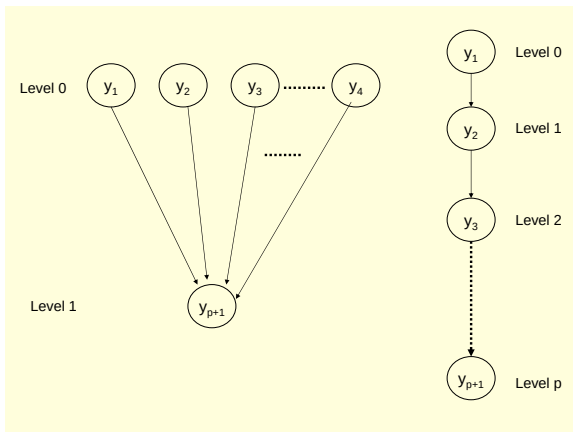

Figure 1. Directed acyclic graphs corresponding to the generalized partially separable function and the composite dynamic system.

where $T_i(i = 1, \cdots, p)$ and $\bar{F}$ are nonlinear functions. Clearly, the GPS case is the best one can hope for from the point of view of (macro-) parallelism since each computation of intermediate variable y_i is independent. The structured computation approach often allows for the identification of less obvious (macro-) parallelism. However, the worst case is the composite function of a dynamic system which involves heavy recursion:

$$\left.\begin{array}{l} \text{Solve for } y_i: \ y_i - T_i(y_{i-1}) = 0, i = 1, \cdots, p \\ \text{Solve for } z: \ z - \bar{F}(y_p) = 0, \end{array}\right\}. \tag{5}$$

In this case there is no obvious (macro) parallelism. The component functions T_i and corresponding Jacobian matrices J_i must be computed sequentially. The expanded Jacobian matrices of generalized partially separable function in (4) and the composite dynamic system (5) are, respectively,

$$J^E_{GPS} = \begin{pmatrix} -J_1 & I & & & \\ -J_2 & & I & & \\ \vdots & & & \ddots & \\ -J_p & & & & I \\ 0 & \bar{J}_1 & \bar{J}_2 & \cdots & \bar{J}_p \end{pmatrix} \quad \text{and} \quad J^E_{DS} = \begin{pmatrix} -J_1 & I & & & \\ & -J_2 & I & & \\ & & \ddots & \ddots & \\ & & & -J_p & I \\ & & & & \bar{J} \end{pmatrix}.$$

Typically, as illustrated above, many of the block entries of J^E in (3) are zero-blocks, and we can associate a directed acyclic graph, $\overrightarrow{G}(J^E)$, to represent this block structure. Specifically $\overrightarrow{G}(J^E)$ has $p+1$ nodes, $y_1, \cdots, y_p, y_{p+1} = z$ and there is a directed edge y_j from y_i iff $\frac{\partial F_i}{\partial y_j} \neq 0 (i = 1 : p+1, j = 1 : p, i \neq j)$. Thus, the corresponding directed acyclic graphs for the GPS and composite functions are shown in Figure 1. It illustrates that the generalized partially separable case is the most 'parallelism-friendly' since there are only two levels and p of total $p+1$ nodes can be computed concurrently. Figure 1 also illustrates that the composite dynamic system case is the worst with respect to parallelism

since there are p levels for a total of $p+1$ nodes and this sequence of nodes has to be computed sequentially [a].

To illustrate the benefit of parallelism, consider the following experiment on a GPS problem. We define a composite function $T(x) = \hat{F}(A^{-1}\tilde{F}(x))$, where $\hat{F}$ and $\tilde{F}$ are Broyden functions[1], $y = B(x)$, (their Jacobian matrices are tridiagonal), which is in the following form

$$\begin{aligned} &y_1 = (3-2x_1)x_1 - 2x_2 + 1, \\ &y_i = (3-2x_i)x_i - x_{i-1} - 2x_{i+1} + 1, \quad i = 2, 3, \cdots, n-1 \\ &y_n = (3-2x_n)x_n - x_{n-1} + 1, \end{aligned}$$

where n is the size of the vector variable x. The structure of A is based on 5-point Laplacian defined on a square $(\sqrt{n}+2)$-by-$(\sqrt{n}+2)$ grid. For each nonzero element of A, A_{ij} is defined as the function of x, that is $A_{ij} = r_j x_j$ where r_j is a random variable, e.g. $r_j = N(1,0)$, that is r_j is normally distributed around 1 with variance 0. So, the GPS function can be defined as follows

$$G(x) = \frac{MB^{-1}}{K}[T_1(x) + T_2(x) + \cdots + T_K(x)],$$

where $T_i(x)$ is same as $T(x)$ except the uncertainties in A, B is sparse symmetric positive definite tridiagonal, M is tridiagonal and K is a scalar. The explicit form of Jacobian matrix of $G(x)$ is

$$J = \frac{MB^{-1}}{K}(J_1 + J_2 + \cdots + J_K),$$

where J_i is the Jacobian matrix of $T_i(x)$. It is clear that parallelism can be used to concurrently evaluate each pair $(y_i,\ J_i),\ i = 1, \cdots, p$. A simple master-slave scheduling mechanism can be used to assign tasks to processors and collect results. The MatlabMPI package[5] developed by Kepner at MIT is used to implement the parallel computation. The experiments were carried out on a SGI Altix 3700 system with 64 1.4GHz Itanium2 processors and 192 GB RAM running under SuSE Linux Enterprise System (SLES) 10 with SGI's ProPack 5 added on. Matlab 7.0.1 (R14) does not support the 64-bit Itanium architecture, so we use "Matlab -glnx86" to run Matlab in 32-bit emulation mode. Twenty-four processors, one for master and the other twenty three for slaves, were employed to solve nonlinear equations of generalized partially separable problems with vector variable sizes ranged from 625 to 2500 and $K = 240$. In the parallel computation of the Newton method exploiting the structure, each processor computes the summation of 10 J_i's independently. Then, the master collects the summation from slaves to compute the Jacobian J, and solves the dense Newton system by '\' in Matlab. We do not construct the expanded Jacobian matrix, J^E_{GPS}, in this experiment because forming the explicit form of J is quite efficient. In the parallel computation of the Newton method without exploiting the structure, we treat the Jacobian matrix J as full and use the forward mode AD to compute derivatives . Each processor computes the same number of columns of Jacobian J independently. After that, the master collects the columns from slaves to construct J and solves the dense Newton system using '\'. The package ADMAT-2.0, a new version of ADMAT[4], is employed to

[a] In this paper, we talk a worst case view since some of the computations in node i depend on results from node $i-1$. We assume, for simplicity, that there is no concurrency between these nodes.

implement the structured and the forward mode AD. Table 1 displays the results of the running time of the standard Newton method exploiting and without exploiting the structure in sequential and parallel computation, respectively.

	Newton method exploiting the structure			Newton method without exploiting the structure		
n	Sequential computation	Parallel computation	Speedup	Sequential computation	Parallel computation	Speedup
625	121.92	8.52	14.31	268.87	73.97	3.63
900	310.47	19.41	16.00	564.07	172.11	3.28
1024	469.37	26.71	17.57	735.09	266.47	2.83
1600	1526.17	81.16	18.80	2817.22	824.97	3.41
2500	5556.49	265.46	20.93	27685.07	8556.37	3.19

Table 1. Running times of the Newton method for solving a GPS problem in sequential and parallel computation implemented on 24 processors in seconds.

As the problem size increases, the computation of the Jacobian matrices J_i becomes increasingly expensive, and this dominates the computation time of a single Newton step. The speedup due to parallelism approaches 20 exploiting the structure and is less than 4 without exploiting it, using 24 processors. In the computation of Jacobian matrix J, the product of A^{-1} with some matrix is required. To compute the product, we use '\' in Matlab to solve a multiple right-hand sides linear system, instead of computing the inverse of A. However, the running time of solving a multiple right-hand sides linear system is not proportional to the size of the right-hand sides. For example, A and B are matrices of 625-by-625 and C is a matrix of 625-by-25. The time for computing $A^{-1}B$ is only 3 to 4 times longer than computing $A^{-1}C$ although C is 25 times smaller than B. In other words, the product with A^{-1} restricts the speedup of parallelism. It explains that the speedup is less than 4 without exploiting the structure.

MatlabMPI was implemented through the file system in Matlab, rather than using "real message passing ". In other words, message passing between the master and slave processors can be quite time-consuming. However, in our program, we minimize the communications among processors. In a single Newton step, only two communications are required. One is sending the result back to the master from slaves. The other is distributing the updated x from the master to slaves for the next Newton step. Thus, the message passing in MatlabMPI does not slow down the parallel computation on the generalized partially separable problem. The time spent on communication is less than 5% running time in our experiments.

The other example is a dynamic system computation. Consider the autonomous ODE,

$$y' = F(y),$$

where $F(\cdot)$ is a Broyden function and suppose $y(0) = x^0$, we use an explicit one-step Euler method to compute an approximation y_k to a desired final state $y(T)$. Thus, we

n	Newton method exploiting the structure	Newton method without exploiting the structure	Speedup
200	0.3120	0.0940	3.3191
400	3.0620	0.2970	10.3098
800	26.6100	1.0790	24.6846
1000	52.3460	1.6090	32.6576

Table 2. Running times of one Newton step through two approaches and the speedup of exploiting the structure AD on a dynamical system.

obtain a recursive function in following form,

$$\begin{array}{l} y_0 = x \\ \text{for } i = 1, \cdots, p \\ \qquad \text{Solve for } y_i : \ y_i - F(y_{i-1}) = 0 \\ \text{Solve for } z : \ z - y_p = 0, \end{array}$$

where we take $p = 5$ in the experiment. This experiment was carried out on a laptop with Intel Duo 1.66GHz processor and 1GB RAM running Matlab 6.5. The Jacobian matrix J is treated as full when the structure is ignored. Table 2 records the running times and the speedup in sequential computation since there is no apparent parallelism in this example. Subsequently, Table 2 illustrates that the cost of Newton step with structured AD is linear while the cost of the unstructured is cubic. In other words, exploiting the structure accelerates the computation of the Newton step.

3 General case

Of course the generalized partially separable case represents the best situation with respect to parallelism whereas the right-hand of Figure 1, the composite function, represents the worst - there is no apparent parallelism at this level. In general, this structured approach will reveal some easy parallelism which can be used to further accelerate the Newton step computation. In this section, we will look at an "in-between" example, to illustrate the more general case.

We consider the evaluation of $z = F(x)$ in following steps:

$$\left.\begin{array}{l} \text{Solve for } y_i : \ y_i - T_i(x) = 0 \quad i = 1, \cdots, 6 \\ \text{Solve for } y_7 : \ y_7 - T_7((y_1 + y_2)/2) = 0 \\ \text{Solve for } y_8 : \ y_8 - T_8((y_2 + y_3 + y_4)/3) = 0 \\ \text{Solve for } y_9 : \ y_9 - T_9((y_5 + y_6)/2) = 0 \\ \text{Solve for } y_{10} : \ y_{10} - T_{10}((y_7 + y_8)/2) = 0 \\ \text{Solve for } y_{11} : \ y_{11} - T_{11}((y_8 + y_9)/2) = 0 \\ \text{Solve for } z : \ z - 0.4(y_{10} + y_{11}) - 0.2y_5 = 0. \end{array}\right\}, \tag{6}$$

where $T_i(x)(i = 1, \cdots, 6)$ is same as the function $G(x)$ in Section 2 except have $K = 3000$, $T_j(x)(j = 7, \cdots, 11)$ is the same as the function $T(x)$ in Section 2, but with different uncertainties. It is clear that the computation of $T_i(x)$ $(i = 1, \cdots, 6)$ dominates

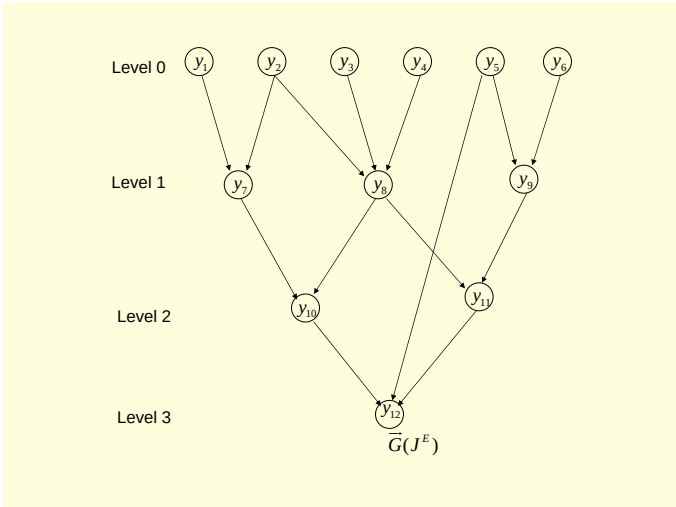

Figure 2. Directed acyclic graph corresponding to 12-by-12 Jacobian matrix J^E.

the running time of evaluating $z = F(x)$. The structure of the corresponding expanded Jacobian matrix is illustrated as follows,

$$J^E = \begin{bmatrix} X & X & & & & & & & & & & \\ X & & X & & & & & & & & & \\ X & & & X & & & & & & & & \\ X & & & & X & & & & & & & \\ X & & & & & X & & & & & & \\ X & & & & & & X & & & & & \\ & X & X & & & & & X & & & & \\ & & X & X & X & & & & X & & & \\ & & & & & X & X & & & X & & \\ & & & & & & & X & X & & X & \\ & & & & & & & & X & X & & X \\ & & & & & X & & & & & X & X \end{bmatrix},$$

and the corresponding directed acyclic graph $\overrightarrow{G}(J^E)$ is given in Figure 2. The level sets in $\overrightarrow{G}(J^E)$ (Figure 2.) can be used to identify the (macro-) parallelism in the structured Newton computation. All the nodes on level i can be computed concurrently. Then, after all the nodes on level i are computed, we can start to compute all the nodes on level $i+1$. In the experiment for exploiting the structure, we divide the six nodes on level zero into three groups. Each group employed 8 CPUs, one for master and others for slaves. The master processor is in charge of collecting results from 'slaves' in its own group and sending the results to the master processors in other groups if necessary. There are only three nodes on level 1, so only master processors do the computation and communication. On level two, one of the master processor is idled while only one master processor is working on level

	Newton method exploiting the structure			Newton method without exploiting the structure		
n	Sequential computation	Parallel computation	Speedup	Sequential computation	Parallel computation	Speedup
100	401.60	19.24	20.87	428.77	201.32	2.13
169	807.31	38.41	21.02	1002.07	240.45	4.17
225	1286.73	61.25	21.01	1694.07	335.18	5.05
400	3795.80	180.21	20.93	5373.92	602.02	3.19
625	8878.24	481.71	18.43	12954.22	1017.71	12.73
900	19681.72	1045.37	18.83	26883.98	2105.77	13.34

Table 3. Running times of the Newton method in sequential and parallel computation implemented on 24 processors.

3. Table 3 records the results of the Newton method exploiting and without exploiting the structure. When exploiting the structure, the time spent on communication is about 10% of running time much more than the percentage without exploiting it, which is 2%. In Figure 2, it shows that communications are required between slaves and master in each group on level zero and among master processors on other levels. Thus, the structured AD approach spent more time on communication than the forward mode, but it is still significantly faster than the forward mode case.

4 Conclusions

In this paper, we have illustrated how a standard Newton step also exposes useful parallelism in most cases. This parallelism can be used to further speed up the computation of the Newton step. We studied two extreme cases. One is the general partially separable case, the best case for (macro) parallelism. The other is the composite function of a dynamic system as shown in (5), where there is no exposed parallelism (though in this case the structural techniques proposed in (4) conveniently work particularly well). Generally, most cases are somewhere between these two extreme cases.

References

1. C.G. Broyden, *A class of methods for solving nonlinear simulations equations*, Mathematics of Computations **19**, 577-593 (1965).
2. T. F. Coleman and W. Xu, *Fast Newton computations*, in progress.
3. T. F. Coleman and A. Verma, *Structured and efficient Jacobian calculation*, In M. Berz, C.Bischof, G. Corliss and A. Griewank, editors, Computational Differentiation: Techniques, Applications and Tools , 149–159 (1996), Philadelphia, SIAM.
4. T. F. Coleman and A. Verma, *ADMIT-1: Automatic differentiation and MATLAB interface Toolbox*, ACM Transactions on Mathematical Software **16**, 2000 (150-175).
5. J. Kepner, *Parallel programming with MatlabMPI*, `http://www.ll.mit.edu/MatlabMPI`.