Introduction

The following material is the description of a suggested parallel programming project CS442/EECE432 in lieu of a final exam. For those students in the class who wish to work on their own programming project and have had me approve it, the section “Requirements And Documents For The Assignment” of this document also outlines the requirements and the material I expect to see in order to give you a full grade on the project.

Second Order Blind Identification Algorithm For Computing Independent Components of Magnetoencephalography

This project is based upon a request to the High Performance Computing Center to improve the speed of extracting information from data representing signals from the brain; the data for a complete study is massive in size and even for the test data it is nearly a gigabyte. The problem presented to us boiled down to a request to apply a second-order blind identification algorithm to a collection of modified data that took over 5 hours of computation on a 1.5 GHz AMD Athlon processor executing serial code (actually written in MATLAB) to process and reduce that computation time down to 10 minutes or so. And indeed, the statement from the scientists making the request was that the real problems they want to work on had datasets 10 or even 100 times bigger.

The improvement to 10 minutes was very ambitious. Certainly parallelism could help but of course there were other considerations such as considerably more efficient implementations for the second-order blind identification algorithm and indeed the use of different numerical linear algebra techniques to reduce the amount of computation. The project for you is to take the simplified description of the algorithm below and make in run in parallel. The project requires a write-up by you, as outlined below, explaining what tasks are concurrent and how you are taking advantage of the parallelism (there are many ways to implement this in parallel). Please justify the choices you made and provide an analysis of improvements made, including predictions on how the parallelism will scale as the parameters of the problem increase. Also provide a description of ways you think your implementation could be improved and why the changed algorithm may be an improvement.

In essence, the computation is as follows: First, a collection of 32 symmetric matrices of size $122 \times 122$ is computed as described below. The eigensystem of the first matrix, say $A_1$, is computed, giving an eigenvector matrix $Q$ (that is, $A_1 Q = Q \Lambda$ where $\Lambda$ is a diagonal matrix with real elements on the diagonal) and which is orthogonal ($Q$ is orthogonal if and only if $Q^T = Q^{-1}$). Then, included in this first step, each of the remaining 31 matrices, say, $A_2, A_3, \ldots, A_{32}$ is transformed by $Q$ as follows creating a collection of 31 matrices $B_1, B_2, \ldots, B_{31}$: $B_k = Q^T A_k Q$ for $k = 1, 2, \ldots, 31$. Finally, as the second step, the joint diagonalization transformation $V$ of the 31 $B$ matrices is computed such that $V^T B_k V$, for all $k$, are approximately diagonal matrices. $V$ is the desired result which is further analyzed by the application code. The further manipulation of $V$ is not part of this assignment.

Undoubtedly, you do not know how to compute any of these quantities but we are going to learn, creating a parallel program that performs this computation in a reasonable time. The description below is of the initial test data set to give you some idea of the size of the problem. We will discuss below how to reduce the problem down to something we can easily code and debug on Blackbear; the original description is given here to give you some idea of the sizes of parameters in the real problem.
The first step is to compute the 32 \( A \) matrices. These are computed as follows, assuming one has a data set \( \text{ALL\_DATA} \) of 550,000 columns of length 122:

1. Read a vector \( \tau \) of 32 integers, where each element is in the range 1 to 100 with no duplicates. (You can create your own vector of say 32 random integers – they do not need to be ordered but should not be duplicated in the final test.)
2. Initialize a set of 32 matrices \( A_1, A_2, \ldots, A_{32} \) of size 122 \( \times \) 122 to zero
3. For \( k = 1, 2, \ldots, 500 \)
   a. Read in an array \( \text{DATA} \) of size 122 \( \times \) 1100 from the data set \( \text{ALL\_DATA} \); that is, read in the next 1100 columns of length 122 from the given data set.
   b. Form the mean of the rows of the array \( \text{DATA} \) and subtract the mean of each row from the elements in each row.
   c. For each integer \( j \) from 1 to 32
      i. Form the product \( R \) of two matrices of size 122\( \times \)1000, the first matrix is the first 1000 columns of \( \text{DATA} \) and the second matrix is the transpose of the matrix made up from 1000 consecutive columns of \( \text{DATA} \) starting from column \( 1+\tau(j) \). In matrix notation,
         \[
         R = \text{DATA}(1:122,1:1000) \cdot \text{trans} \{\text{DATA}(1:122,1+\tau(j):1+\tau(j)+999)\}
         \]
      ii. Form the matrix sum \( 0.5 \cdot (R + R^T) \), add it to \( A_j \) and assign to \( A_j \).
4. Scale all \( A \) matrices by 1100*500; that is, compute \( A_k = A_k/550000 \) for all \( k = 1, 2, \ldots, 32 \)
5. Compute the eigenvector matrix \( Q \) of the first symmetric matrix \( A_1 \) (using the appropriate LAPACK routine \( ?\text{SYEV} \))
6. Form the matrices \( B_k = Q^T A_{k+1} Q \) (this is two matrix products) for \( k = 1, 2, \ldots, 31 \).

The second and final step is the joint diagonalization of the 31 \( B \) matrices; we describe Cardoso’s algorithm for performing joint diagonalization of such a set of matrices. A description of this algorithm can be found in the CS442 handout directory as cardoso_jd.ps, a published paper by Cardoso; also, I have placed a modified version of Cardoso’s MATLAB code (serial code for complex matrices, placed side by side in a row, forming a row-block of matrices) as the file joint_diag.m in this directory as well. Finally, I describe below Cardoso’s algorithm at a high level. (Although you may not know MATLAB, the code is straightforward; the paper should make clear some of the mysteries and finally the description of the algorithm below should fill in the remaining gaps – of course, ask questions where you do not understand this code)

The overall structure of Cardoso’s algorithm is as follows:

1. \[
D = (B_1 \ B_2 \ \ldots \ B_{31}) \ (122 \times 122 \times 31 \text{ array}), \quad V = I \ (122 \times 122 \text{ identity matrix}) \quad \text{and} \quad \text{THRESH} = \sqrt{\text{machine precision}} \quad \text{(this is approximately 1.0E-4 for single precision or 1.0E-8 for double precision)}
\]
2. Perform the following iterations until convergence is reached (in your test code, I highly recommend you limit the number of iterations to say 100 or 1000 until you are convinced you have implemented the code correctly and your test matrices have a joint diagonalizing matrix – if all the matrices are the same, for example, it will have a joint diagonalizer; otherwise, you code may loop forever).
   a. Set a variable to indicate the iteration has converged (see step (b ii 3) below)
   b. For every off-diagonal element \( (p,q) \) in turn of the first 122 \( \times \) 122 block of \( D \) of size 122\( \times \)122 (there are 122*121/2 such elements, with \( p < q \)):
      i. Compute the Given’s rotation \( G \) (a 2\( \times \)2 matrix)
         \[
         G = \begin{pmatrix}
         c & s \\
         -s & c
         \end{pmatrix}
         \]
         as follows:
         1. Form the following vectors
            a. \( \text{dpp} \) = a vector of the \( (p,p) \) elements of each \( B_k \), \( k = 1, 2, \ldots, 31 \)
            b. \( \text{dqq} \) = a vector of the \( (q,q) \) elements of each \( B_k \), \( k = 1, 2, \ldots, 31 \)
            c. \( \text{dpq} \) = a vector of the \( (p,q) \) elements of each \( B_k \), \( k = 1, 2, \ldots, 31 \)
d. $d_{qp}$ = a vector of the $(q,p)$ elements of each $B_k$, $k = 1, 2, \ldots, 31$

e. $f = d_{pp} - d_{qq}$

f. $g = d_{pq} + d_{qp}$

g. $h = d_{pq} - d_{qp}$

2. From these vectors, form the $3 \times 3$ symmetric matrix $S$:

$$
S = \begin{pmatrix}
    f^T f & f^T g & 0 \\
    f^T g & g^T g & 0 \\
    0 & 0 & h^T h
\end{pmatrix}
$$

3. Compute the largest (real) eigenvalue of $S$ and its corresponding eigenvector $v$ using the LAPACK routine ?SYEVX (? is S for the single precision and ? is D for double precision using a threshold of THRESH)

4. Normalize $v$ so that its first component is positive (that is, if $v(1) < 0$, set $v = -v$)

5. Compute $c$ as $\sqrt{(1 + v(1)/2)}$ and $s = v(2)/(2c)$ (which has formed $G$)

i. If $G$ is significant enough ($|s| > \text{THRESH}$), then

1. Apply the Given’s rotation as follows: $D = G^T DG$ (The product $G^T D$ is applied to rows $p$ and $q$ of $D$ giving the result $D'$ and the product $D'G$ is applied to 31 pairs of columns $p$ and $q$ of each subblock of $D'$).

2. Accumulate the Given’s rotation in $V$ as follows: $V = V^* G$ (This product is performed to the $p$-th and $q$-th columns of $V$)

3. Set the variable to indicate the iteration has NOT converged.

c. If the iteration has converged, quit else repeat from step a above

As you can see, Cardoso’s joint diagonalization algorithm is not trivial but has been considerably simplified by using real data (not complex data). However, the project is a real computation that is challenging to run in parallel, and shows many of the real challenges in the parallelization of code.

**Hints And Suggestions**

Of course, the issue immediately is what kind of decomposition into tasks should be used. There are several reasonably decompositions to use but to minimize your effort, use the parameter value 31 and/or 32 as the number of natural tasks to divide this problem into (500 is a possible candidate for the first part of the problem, that is forming the $A$ matrices, but does not apply well to the joint diagonalization part). However, even with this hint, there are further ways to parallelize this, using 16, 8, 4, or 2 processors.

Your attempts to parallelize this algorithm should be general relative to the natural parameters of the problem. For example, in the real application, the 500 parameter could be much larger; the parameter 122 could be larger but 128 is about the maximum. The parameter 32 (or 31 = 32-1) is also possibly larger but is unlikely to be larger than 128.

As stated, this project should not be attempted in its full size, at least initially. Write serial code first to debug the formation of the $A$ matrices, the $B$ matrices, and the joint diagonalization matrices. In the real problem, we were not given any small test data to try out the parallel or serial code, but you should generate your own; for example, initially test the problem with generated matrices that are all the same and say of size 2 or 3. Then, the joint diagonalization result is simple, namely the eigenvectors of the first matrix $A_1$. Then, try matrices that are all the same values everywhere, except for a few elements in which THRESH/100 or THRESH has been added to a few elements of some of the matrices (make sure these perturbations result in symmetric matrices).

Do not try to write your own eigensolvers. Use LAPACK (these are serial Fortran codes and are implemented on Backbear in a library /usr/local/lib/liblapack.a – the documentation can be found on the web or I can make the source available which has all of the documentation in comments in the routines you need. This library is callable from Fortran, C, and C++ with care but I am considering installing a C version of LAPACK for C users – the ambitious amongst you
may want to download the C version from the web and install a *.a library for yourself and your fellow C users.) Do not use the parallel version of LAPACK (called SCALAPACK) as you do not need SCALAPACK to obtain reasonable parallelism and unfortunately it does not support directly the joint diagonalization computation.

Finally, ask questions in class and discuss the issues and problems you see with myself and classmates. The parallelization of the formation of the matrices scales well, with a straightforward approach up to 16 processors for problems sizes with 500 and 32 for the values of the two key parameters, and that for the joint diagonalization does not scale easily or so well. It is conceivable that an algorithm that scales for the joint diagonalization portion uses a different method than Given’s rotations but you are not expected to find or use a better algorithm. The purpose of the project is not to achieve perfect scaling but to generate a correct implementation with an analysis that explains the performance you observe.

**Requirements and Documents For The Assignment**

I require the following documentation and testing for your project to obtain full marks. For those doing their own project, I request a description of the problem, such as appears above in addition to the following items.

1. A write-up for your parallel algorithm, describing the task decomposition (the concurrent tasks), supported by printouts of your code.
2. Describe the interaction (MPI communication) you used to coordinate the concurrent tasks and processes you developed.
3. Evaluate your own implementation by timing the interactions and from the timing results, indicate the degree of scaling of your implementation. Analyze your code explaining the scaling you have observed and conjecture ways to fix or improve it. Provide mathematical estimates of the speedup, efficiency, and scalability of your implementation.
4. Predict how your parallel program will scale when the 500 parameter is doubled several times, and when the 32 parameter is doubled several times. To obtain some indication on how your program is scaling, try and run it with a series of runs, trying different parameters. For parameters of the size of this problem (500 and 32), the problem should run in 30 minutes on 16 processors or so for a good implementation and so test your code on smaller problems that take in the range of a few minutes. This can easily be done by reducing the size 500 to 100 or reducing 122 to 10 to 20.
5. Convince me your program is correct. For example, for the test cases you use, the products $D_k \equiv V^T B_k V$ for the output $V$ and inputs $B_k$ should be nearly diagonal, or the final $V D V^T$ where $D$ is the final $D$ matrix in Cordoso’s algorithm should be the sequence of $B$ matrices $(B_1, B_2 \ldots B_{31})$. 