SANDIA REPORT

SAND2000-1529
Unlimited Release
Printed June 2000

A Parallel Prediction-Augmented Classical Least Squares/Partial Least Squares Hybrid Algorithm: CPLS 1.0 Code

Celeste A. Drewien

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

A disclosure of invention relating to the subject of this publication has been filed with the U.S. Department of Energy.

Approved for public release; further dissemination unlimited.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865)576-8401
Facsimile: (865)576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800)553-6847
Facsimile: (703)605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/ordering.htm

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

SAND2000-1529
Unlimited Release
Printed June 2000

A Parallel Prediction-Augmented Classical Least Squares/Partial Least Squares Hybrid Algorithm: CPLS 1.0 Code

Celeste A. Drewien
P.O. Box 5800
Sandia National Laboratories
Albuquerque, NM 87185-0977

Abstract

A parallel code or CPLS 1.0, based upon the recently invented prediction-augmented Classical Least Squares/Partial Least Squares (PA-CLS/PLS) hybrid algorithm, was developed in order to perform quantitative analyses of spectral data. A parallel routine was desired for fast calculation of full calibration models, cross-validation verification, and true prediction results from large data collections, which are expected to result from two-dimensional infrared (IR) imaging spectroscopy. This report contains a summary of the parallel PA-CLS/PLS hybrid algorithm, the mathematical formulation, and software libraries used in the implementation of the parallel algorithm. An explanation of how to use the three routines in the CPLS 1.0 code--full calibration, cross-validation, and true prediction--is provided along with variable definitions, input file requirements, and output file data formats. In order to demonstrate parallel speed-up, the added advantage of multiple processors, and communication versus computation times, timing results for each of the three routines are reported for a representative (but small) sample set.

Acknowledgments

The author would like to thank the following people for their assistance: David Haaland, Bruce Hendrickson, David Melgaard, Sue Goudy, Eric Engquist, Doug Pannell, and Frederick Koehler of Sandia National Laboratories and L. Susan Blackford, Antoine Petitet, and Clint Whaley of University of Tennessee-Knoxville.

Contents

ABSTRACT **3**

ACKNOWLEDGMENTS **4**

SECTION 1: THE PARALLEL PREDICTION-AUGMENTED CLS/PLS HYBRID ALGORITHM AND UNDERLYING MATHEMATICAL DESCRIPTION **6**

PARALLEL PA-CLS/PLS ALGORITHM 6
PA-CLS/PLS MATHEMATICAL BACKGROUND 8
F-STATISTIC 12

SECTION 2: HOW TO USE THE CPLS 1.0 CODE **13**

ITEMS THE USER MUST SUPPLY 13
CPLS_input.file Input Parameters *14*
**_full_input.dat Input File* *15*
**_absorb.dat Input File* *16*
**_moreK.dat Input file* *16*
FURTHER EXPLANATION OF THE CPLS_INPUT.FILE INPUT DATA 16
Output files/Sub-directory Name *16*
Processor Grid Parameters *17*
Input/Output *20*
Mb: Block Sizes—Distributing the Matrices *21*
Msize, Nsize, Ksize, Nskipc, and Nskips--Data Sizes *21*
Mstep—Step size for cross-validation *22*
Irecl--Record Length *22*
OUTPUT DATA FORMATS 22
**_pred_input.dat* *23*
**_full_output.dat* *23*
**_cross_output.dat* *24*
**_F_test.dat* *24*
**_PREDICTIONS.dat* *24*
**_info* *25*
CPLS SOURCE AND MISCELLANEOUS FILES 25
Makefile *25*
Readme *28*

SECTION 3: THE TEST DATA SET AND TIMING RESULTS **29**

SUMMARY **39**

REFERENCES **40**

Section 1: The parallel prediction-augmented CLS/PLS hybrid algorithm and underlying mathematical description

Quantitative spectral analyses are performed by multivariate techniques, such as classical least squares (CLS), partial least squares (PLS), inverse least squares (ILS), and principal component regression (PCR) methods. Recently, Haaland[1] has developed a hybrid CLS and PLS approach that takes advantage of the strengths of each method and yields superior prediction capabilities over traditional multivariate methods. This new method is being referred to as the prediction-augmented classical least squares/partial least squares (PA-CLS/PLS) method.

With the advent of two-dimensional (2D) infrared (IR) imaging spectrometers, which are capable of collecting IR spectra at every point on a 2D camera, large sets of spectral data can now be obtained. With increased sizes of data sets, high demands are placed on accurate yet efficient information extraction routines and the need for massively parallel routines becomes highly important.

The PA-CLS/PLS hybrid algorithm has been developed into a parallel algorithm to accommodate large data sets and to reduce computation times. This parallel PA-CLS/PLS code has been named CPLS 1.0. It offers improved computation times for developing a calibration model, performing cross-validation on the calibration data set, and predicting concentrations from a large collection of spectra from (an) unknown sample(s).

The mathematics behind the PA-CLS/PLS hybrid algorithm, with a particular focus upon the parallel algorithm, and a description of the code are provided below.

Parallel PA-CLS/PLS Algorithm

A parallel prediction-augmented classical least squares/partial least squares hybrid algorithm, or the CPLS 1.0 code, was developed in Fortran 90 programming language taking advantage of the parallel routines provided in the Scalapack, or Scalable Linear Algebra Package[2]. Scalapack can be used on homogenous or heterogeneous computer platforms and should provide a high degree of portability to the code. The Scalapack software library aims to provide high performance linear algebra subroutines for distributed memory MIMD machines.

The CPLS 1.0 code provides three major functions:

1. Full calibration
   - A full calibration model is calculated using the spectral and concentration data from a known data set, supplied by the user as *_full_input.dat. The full calibration model is output in the file *_pred_input.dat.

- Prediction on every sample in the full calibration set is then performed using the newly calculated full calibration model. The code outputs the predicted concentration for each sample as a function of component and factor and provides a measure of the fit of the calculated spectral data to the actual data. This output is provided in *_full_output.dat.

2. Cross-validation
   - During cross-validation, a calibration model is calculated when a sample(s) is(are) excluded from the original data set. The routine then predicts the concentration of the sample(s) excluded from the data set as a function of component and factor; it provides a measure of the fit of the calculated spectral data to the actual data. The output is provided in the file *_cross_output.dat.
   - PRESS, prediction error variance, is computed from the summation over all samples of the square of the difference of the predicted concentration from the known concentration. This output is also provided in the file *_cross_output.dat.
   - The use of an F statistic for selecting the optimal number of factors is then performed. This output is provided in *_F_test.dat.
3. True Prediction
   - Through use of the model calculated during full calibration and of the optimal number of factors found during cross-validation, predicted concentrations can be calculated from a collection of spectral data for unknown samples; the unknown sample spectral data must be provided by the user in a file called *_absorb.dat. A fit of the calculated spectral data to the actual spectral data is also made. The output is all provided in *_PREDICTIONS.dat.

The PA-CLS/PLS hybrid algorithm has been implemented in a parallel format in order to reduce the computation time involved with large numbers of samples in the full calibration/cross validation routines and in the true prediction routine. Two options for parallel computation are offered by the CPLS 1.0 code:

> Option 1. Parallel computation around the numbers of samples in the data sets.
> Option 2. Parallel computation around the numbers of samples in the data sets when the data sets are so large as to exceed the space available on one node.

The user will find option 1 most applicable for the vast majority of situations and will find that option 2 becomes more appropriate for use when very large data sets are being used, especially for the true prediction routine.

The CPLS 1.0 code can be used on any computer system that provides a Fortran 90 compiler, the Scalapack, BLAS (Basic Linear Algebra Subprograms), LAPACK (Linear Algebra Package), and BLACS (Basic Linear Algebra Communication Subprograms) libraries, and MPI (Message Passing Interface). The user can download all of these libraries from the web site http://www.netlib.org. The user should assure that an optimized BLAS library exists on the computer system upon which the CPLS 1.0 code is to be used.

PA-CLS/PLS Mathematical Background

This section contains an explanation of some of the mathematics used in the parallel algorithm. The following nomenclature is used:

1. Upper case letters represent matrices
   A is the spectral matrix
   C is the concentration matrix
   E_A is the spectral residual matrix
   E_c is the concentration residual matrix
   K is the baseline plus pure component matrix
   P is the pure component matrix
   T is the matrix of scores

2. Lower case letters represent vectors
   a is a spectral vector
   c is the concentration vector for the nth component per factor
   b is the PLS loading vector for the nth component per factor
   e_c is the concentration residual for the nth component per factor
   t is the score vector for the nth component per factor
   v is the regression coefficient relating scores to concentrations for the nth component per factor (note v stores a collection of scalar quantities)
   w is the weight loading vector for the nth component per factor

Table 1a summarizes the mathematics behind the PA-CLS/PLS hybrid calibration algorithm.
Step 1 is centering/scaling of the data in order to reduce round-off errors during computation. The code contains an option for column-centering of the data through use of the Iscale parameter = 1.
Steps 2-5 of the calibration routine are CLS procedures.
In the CPLS 1.0 code, Step 2 is performed using a QR decomposition of C to find P. The concentration and spectral (A) matrices are provided in the file *_full_input.dat by the user.

Step 3 forms the K matrix by adding a baseline to the pure (P) matrix and optionally by adding additional information to the K matrix (Nadd > 0). Therefore, the K matrix has more rows than the P matrix.

A baseline for the pure component spectra is introduced as a linear, quadratic, or cubic fit of the data over the range of frequencies being investigated. A linear fit is performed by setting the lowest frequency to be equivalent to –1 and the highest frequency to be equivalent to 1. The remaining values are then found by calculation of the slope of the x-axis (frequency range over which spectral measurements were obtained) data across this domain. These values, ranging from –1 to 1 are placed into the first row of the K matrix. For a quadratic fit, these values are then squared and placed into the second row of the K matrix. For a cubic fit, the first row of K matrix is multiplied by the second row of K matrix and placed into the third row of the K matrix.

The user through the Nbase parameter in the CPLS_input.file determines the level of baseline fit.

The next set of rows added to the K matrix is the pure (component) matrix, calculated in Step 2.

Additional pure component spectra may be added to the K matrix in rows beneath the pure matrix portion of the K matrix. The number of additional spectra to be added are input by the user as Nadd in the CPLS_input.file. Data for additional input to K matrix must be supplied in the file *_moreK.dat.

Step 4 is the calculation of the matrix of scores, T. The equation in Step 4 can be rewritten in terms of transpose matrices such that a QR decomposition of K transpose is used to solve the linear algebra problem K'T'=A'. Once T' is found, another transpose is used to find T.

Step 5 is the calculation of the residuals of the spectral and concentration matrices. Note that T contains excess columns due to the increased size of the K matrix versus the P (pure) matrix; thus the excess columns have to be omitted from the calculation of the concentration residuals.

Steps 6-12 take advantage of PLS regression[3]. The PLS steps are performed iteratively for each component (herein referred to as n) over a given number of factors (referred to as h). The number of factors used by the full calibration and cross-validation codes is the Nend parameter supplied by the user in the CPLS_input.file. The number of components is the difference Nsize-Nskipc, supplied by the user in the CPLS_input.file.

Step 6 first requires the normalization of the c vector. Normalization can be performed by finding the 2 norm of the concentration vector for each component. The entire column in the concentration matrix is then multiplied by the inverse of its 2 norm. A matrix-vector multiply of the spectral or spectral residual matrix with the normalized concentration vector (A'|c|) yields the weight-loading vector, w.

Step 7 requires the weight-loading vector to be normalized. Then a matrix-vector multiply of A|w| yields the score vector, t.

Step 8 again requires a normalization procedure. A vector-vector multiple of |t'|c leads to a scalar regression coefficient, v.

Step 9 forms the PLS loading vector, b, by a matrix-vector multiply of A'|t|.

In Step 10, the residuals, E_A and e_c, are calculated by subtraction of the model values from the original values or upon the next iteration from the previous residual values.

TABLE 1a. Calibration Algorithm[3]

Step 1. Pretreatment of data
Center A and C
Scale A (optional)

CLS algorithm

Step 2. Find Pure Component Spectra P
CP=A
Step 3. Create K
Add baseline and additional spectral features as rows above and below P, respectively.
Step 4. Generate predicted concentrations or scores.
TK=A
Step 5. Find residuals.
E_A=A-TK
Strip extra columns of T.
E_c=C-T

PLS Algorithm

Set h=1.
Step 6. Form weight loading vectors for each component
w=A'c/c'c
Step 7. Form score vectors for each component
t=Aw/w'w
Step 8. Find scalar regression coefficients for each component.
v=t'c/t't
Step 9. Form PLS loading vectors for each component
b=A't/t't
Step 10. Calculate residuals in A and c
E_A =A-tb'
e_c =c-vt
Step 11. Substitute E_A for A and e_c for c
Step 12. Increment h and go to Step 6.

Table 1b summarizes the CLS/PLS augmented-prediction algorithm. The K matrix and b, v, and w data sets found as the calibration model in the full calibration routine are input data into the prediction algorithm.

Step 1, centering/scaling, is used during true prediction if this option was used during full calibration and cross-validation.

Steps 2-4 of the prediction algorithm utilize CLS procedures.
Step 2 allows for addition of new data into the K matrix as desired.
Step 3 is a calculation of the score vector, t, by a matrix-vector multiply of K'a or K' e_a.

Step 4 yields the spectral residual, e_a. Because the scalar regression coefficient (v) used in the CLS calibration model was effectively one, the t vector found in Step 3, minus the excess values for the baseline and additional data, is the predicted concentration. Here the result of a matrix-vector multiply tK is subtracted from the spectral vector a or residual vector e_a to yield the residual vector, e_a.

Thus, the PLS portion of the algorithm starts with this predicted concentration (c_0) and with e_a substituted for a. Iterations over the number of optimal factors, which are values automatically output from the cross-validation routine into the file *_F_test.dat, are performed for each component and for every sample spectra, respectively.

Step 6 finds the score vector (t) by a vector-vector multiply of w'a or w' e_a.
Step 7 uses the scalar regression coefficient (v) multiplied by the score vector (t) to update the predicted concentration.
The spectral residuals e_a are calculated in Step 8.

TABLE 1b. Prediction Algorithm[3]

Step 1. Center **a**, the unknown spectral data. Scale if A was scaled.

CLS Algorithm

Step 2. Develop K.
Use K from full calibration model and provide additional spectra as desired.
Step 3. Calculate the spectral intensities of the sample.
$t=K'a$
Step 4. Find residuals in spectrum.
$e_a=a-tK$
Strip extra rows of t.
Step 5. Substitute e_a for a.

PLS Algorithm

Set h=1.
Step 6. Find spectral intensities of sample.
$t=w'a$
Step 7. Relate the intensities to the concentration using the scalar regression coefficients.
$c_h=c_{h-1}+vt$
where c_0 is the average concentration from the calibration set.
Step 8. Determine residuals in calculated spectra.
$e_h=e_{h-1}-bt$
Step 9. Substitute e_a for a.
Step 10. Increment h and go to Step 6.

Using every sample in the data set except those excluded by the Nskips option, the full calibration routine forms a full calibration model via the calibration algorithm. The full calibration routine then determines the fit of the full calibration model to the data set by

predicting the concentration of every sample in the full calibration set via the prediction algorithm.

The cross-validation routine also utilizes both the calibration algorithm and the prediction algorithm. However, the cross-validation routine effectively performs the full calibration routine multiple times. It successively skips one or more samples in the data set (Mstep) and forms a calibration model for the remaining subset of data. It then predicts the concentration of the sample(s) skipped.

The true prediction routine utilizes only the prediction algorithm. The prediction algorithm is performed for every sample in a data set whose spectra are known but whose concentrations are unknown.

F-Statistic

The F-statistic used for selecting the optimal number of factors during cross-validation is performed as follows:

- Dimension an array to the size of the number of factors used during PLS iteration (Nend) and call it prob.
- Find the minimum value of the prediction error variance or PRESS. Call this value K=min(press).
- Divide each value in the PRESS array by the minimum value K: press(i)=press(i)/K
- For each factor and for each component, find

$$prob(i) = \frac{\left(\sqrt[3]{press(i)} - 1\right)\left(1 - \frac{2}{9m}\right)}{\left(\sqrt[3]{press(i)^2} + 1\right)\left(\frac{2}{9m}\right)}$$

$$prob(i) = 0.196854 * prob(i) + 0.115194 * prob(i)^2$$
$$+0.000344 * prob(i)^3 + 0.019527 * prob(i)^4 + 1$$

$$prob(i) = \frac{1}{prob(i)^4}$$

$$prob(i) = 1 - 0.5 * prob(i)$$

- The optimal factor for a component is the first prob(i) less than or equal to 0.75. The optimal factor number per component is output into the *_F_test.dat file.

Section 2: How to use the CPLS 1.0 code

As previously mentioned, the CPLS 1.0 code provides three routines to the user:

- Full calibration from a collection of known spectra and concentrations
- Cross-validation on the collection of known spectra and concentrations
- True prediction for a collection of known spectra but unknown concentrations

and, the code is designed to be used in that order. That is, the user should first run the code in the full calibration mode (Lswitch = 1). A full calibration model will be built based upon the collection of known spectral and concentration data. Once a full calibration model is established, the user should run the code in the cross-validation mode (Lswitch = 2); the same collection of spectral and concentration data must be used for cross-validation as for full calibration. It is not necessary to run cross-validation before full calibration; however, true prediction (Lswitch = 3) cannot be run without running a full calibration first. So that cross-validation is not mandatory for running true prediction, the user may enter Nend = 0 when running the true prediction routine. The true prediction routine will then set the number of factors (Nend) to the value from the *_pred_input.dat file. Optimal prediction results will only be obtained when the cross-validation routine is run prior to the true prediction routine, that is when the true prediction routine predicts on the optimal number of factors.

The full calibration and cross-validation routines require the *_full_input.dat file. By running the full calibration routine (Lswitch = 1), the full calibration model data in the proper format are automatically output into a file named *_pred_input.dat. By running the cross-validation code (Lswitch = 2), the optimal number of factors for each component is determined via an F-statistic, and the optimal number of factors is output into a file *_F_test.dat, which may be used as input by the true prediction routine. For true prediction (Lswitch = 3), the data collection of spectra from samples whose concentrations are to be predicted, is input into the code. This data set must be in a file called *_absorb.dat. The predictions and other information are output into a file named *_PREDICTIONS.dat.

The CPLS 1.0 code must be compiled and linked on your computer system. In order to perform this task, the user must alter the Makefile for machine dependent parameters (see the Makefile description below). As previously mentioned, the SCALAPACK, BLAS, BLACS, and LAPACK libraries and MPI must be available on the computer system, and high performance will only be obtained by having an optimized BLAS library for the computer system.

Items the User must Supply

The CPLS 1.0 code requires the following:

1. CPLS_input.file—user input options needed for processor grid layout, data size, record length, block sizes, etc. The CPLS_input.file file is supplied as part of the CPLS 1.0 code and only needs to be modified by the user. This file must reside in the CPLS directory.
2. *_full_input.dat—binary unformatted data file containing the number of samples, the number of components, and the number of frequencies to be analyzed followed by the concentration data, the spectral data, and the x-axis (frequency range) data all stored as 4 Byte reals
3. *_absorb.dat—binary unformatted data file containing the number of samples and the number of frequencies to be analyzed followed by the spectral data from unknown samples whose concentration is to be predicted, all stored as 4 Byte reals
4. *_moreK.dat—(Optional input of more K matrix information.) The file must contain the number of features to add (Nadd), the number of frequencies to be analyzed (Ksize), and the data all stored as 4 Byte reals in a binary unformatted data file

The * refers to an 8 character name that will be user supplied in the CPLS_input.file. The user must create a sub-directory by the name of * within the CPLS directory. The required input files with * in their names must reside within this sub-directory.

Items 1 and 2 above are required in order to utilize the full calibration and cross-validation routines. Because the true prediction routine relies upon the model obtained during full calibration and upon the optimal number of factors calculated from the F-test during cross-validation, the true prediction routine can only be utilized after full calibration (and cross-validation) have been performed. For true prediction, Items 1 and 3 are required. Optionally for each routine, the *_moreK.dat file may be used to enter more information into the K matrix.

CPLS_input.file Input Parameters

The CPLS 1.0 code relies upon information provided by the user through the CPLS_input.file file. The user must edit this file and save it as a text file. The input variables are:

1. Outfile—a character array that is eight or less characters in length is required for naming of output files and a sub-directory
2. Msize—m dimension of concentration and spectral matrices, i.e. the number of samples in the input data set, *_full_input.dat for full calibration and cross-validation or *_absorb.dat for true prediction
3. Nsize—n dimension of the concentration matrix, i.e. the number of components in the input data set, *_full_input.dat for full calibration and cross-validation or *_absorb.dat for true prediction
4. Ksize—n dimension of the spectral matrix, i.e. the number of spectral features in the input data set, *_full_input.dat for full calibration and cross-validation or *_absorb.dat for true prediction

5. Mb—number of data rows to separate the rows of the concentration matrix over during 2-D block-cyclic distribution
6. Nprow—number of processor rows
7. Npcol—number of processor columns
8. Nadd—number of additional spectral features to be added to the K matrix. Set to 0 if additional spectral features are not supplied in a file named *_moreK.dat.
9. Nbase—level of baseline correction:
   1 = linear fit
   2 = quadratic fit
   3 = cubic fit
10. Nend—number of iterations (factors) to be used. Set to 0 if cross-validation has not been performed and *_F_test.dat does not exist.
11. Nskipc—number of components to be ignored during calculations
12. Nskips—number of samples to be ignored during calculations
13. Mstep—number of samples to skip during cross-validation
14. Irecl—the record length for binary read and write statements of 4 Byte reals. See Fortran 90 language manual for computer in use.
15. Lswitch—a numerical switch that chooses which routine of the CPLS 1.0 code to run
    1 = full calibration
    2 = cross-validation
    3 = true prediction
16. Iscale—switch to indicate scaling option
    0 = no scaling
    1 = column/row centering of data

Finally, the column number(s) of component(s) to be ignored during calculation of the model need to be input, followed by the row number(s) of the sample(s) to be ignored. Further discussion of the above terms follows below.

*_full_input.dat Input File

Concentration, spectral data, and the frequencies over which the data were sampled are required input into the CPLS 1.0 code. The *_full_input.dat file needs to be an unformatted binary file of real values stored as 4 Byte reals in little endian storage. The information in the file must consist of the following in the order specified:

1. m—The number of samples in the data set. This value must be a real value. This value is Msize input into the CPLS_input.file file.
2. n—The number of components in the data set. This value must be a real value. It is the Nsize input into the CPLS_input.file file.
3. k—The number of frequencies in the data set. This value must be a real. Ksize in the CPLS_input.file set must agree with this value.
4. m*n real values that are the concentration data stored in column-oriented format.
5. m*k real values that are the spectral data stored in column-oriented format.
6. k real values that are the frequencies at which the data was sampled.

As an example of the input for the ct23sa.abs data set, which was used to validate the CPLS 1.0 code and perform timing tests, ct23sa_full_input.dat would contain the following collection of values as 4 Byte reals in little endian format:

34.00000 6.000000 456.0000
2257.000 2508.000 0.000000 2758.000.... Until end of concentration data
1.247896 1.236963 1.255083 1.251408... .Until end of spectral data
7497.879 7505.593 ... Until end of frequency data...11007.69

Here, Msize is 34; Nsize is 6; and, Ksize is 456.

* _absorb.dat Input File

For true prediction, the spectral data taken from a collection of unknown samples must be stored in the *_absorb.dat file. This file also must be a collection of real values stored as 4 Byte reals in little endian storage. The data file must consist of the following information in the order specified:

1. m—The number of samples in the data set. This value must be a real value. This value is the Msize input into the CPLS_input.file file. Note Msize for full calibration and cross-validation must be the same number; however, Msize for true prediction can differ from the Msize for the other two routines.
2. k—The number of frequencies in the data set. This value must be a real value. Ksize in the CPLS_input.file set must agree with this value. This k must agree with the Ksize value used in the full calibration and cross-validation routines.
3. m*k real values that are the spectral data stored in column-oriented format.

* _moreK.dat Input file

For each routine, more information can be added to the K matrix. As mentioned, a Nadd value greater than 0 will require the file *_moreK.dat to exist and be read by the routine. This file needs to contain reals values stored as 4 Byte reals in little endian storage. Again, the file is unformatted so all values must be 4 Byte reals. The first two values in the file are the Nadd value and the Ksize value; these are required by the code for error checking purposes. Starting at the third record, the values of the additional spectral features in a column-oriented sequence must exist.

Further Explanation of the CPLS_input.file Input Data

Output files/Sub-directory Name

The first line of the CPLS_input.file must contain a character string that is no more that eight characters in length. Examples such as 'ct23sa', 'sample23', and 'test' are valid. Once the choice of name is made, the user must create a sub-directory of that name within the CPLS directory. Using the name ct23sa, the user would type:

mkdir ct23sa

in order to create this sub-directory. Then the ct23sa_full_input.dat file (see *_full_input.dat explanation) and if desired the ct23sa_moreK.dat file (see *_moreK.dat explanation) must be placed in this sub-directory. All output files are directed to this sub-directory and contain this name as the first part of their entire name. This naming scheme has been adopted, so that the user may perform different runs from differing data sets and not overwrite previous results—if a differing sub-directory name is used. Note that the CPLS_input.file alone resides in the CPLS directory. The user may wish to copy this file to the sub-directory in order to save the input parameters used for a given run.

Processor Grid Parameters

Presently the code requires the user to specify input parameters about the number of processors and the processor grid layout; thus, the parallelism of the code is not hidden from the user. Future versions of the code could recommend an optimal number of processors and the optimal processor grid layout for specified data set sizes in order to make the parallelism of the code transparent to the user.

The processor grid is user defined by the input parameters of the CPLS_input.file file:

- Nprow—number of processor rows
- Npcol—number of processor columns

The command line for running the code will require input for the number of processors (Nprocs) desired. For instance on a Dec Alpha, the CPLS 1.0 code is run via the command line:

dmpirun –np 8 cpls

where 8 designates a request for 8 processors.

This Nprocs value requested on the command line may exceed the product of Nprow and Npcol (values input to the CPLS_input.file); but, it may not be less than that product.

In determining the number of processors and the processor sub-grid layout—that is, the number of processors in a row and the number of processors in a column, the following is recommended for option 1 in which memory is not an issue:

- Set Nprow = 1
- Provide a value of Npcol that, if possible, can be evenly divided into the number of samples to be analyzed.
- Request Nprocs=Nprow*Npcol on the command line.

When the data collections become very large (especially for true prediction), the user may wish to take advantage of the second parallel computation option. **It is not recommended to use option 2 (i.e. Nprow > 1) unless the code will not function due to the space/storage requirements.** The following explanation may be useful in helping the user understand how the data is distributed and used when more than one processor row is used in a context (sub-grid).

When space constraints become an issue, matrices may be distributed across sub-grids. That is, several processors comprise a sub-grid and each processor owns only a part of an

entire matrix. All mathematical operations involving the matrix are then performed by appropriate communication between processors in the sub-grid as no one processor owns all of the data values of a matrix. The time required for performing calculations is therefore increased by the time required for communicating information between processors in the sub-grid. Thus, the use of the option 2 (i.e., Nprow > 1) should be limited to the cases where memory constraints are encountered and hinder operation of the code/routine on one processor.

Otherwise, the formation of sub-grids via option 1 or option 2 has no effect on the parallelism of the code, because parallelism is derived from splitting the work amongst the sub-grids. And the work is split based upon the number of samples in the data set (Msize) and the number of sub-grids. The work is spread between several sub-grids, whose individual task is to perform the computation for only those samples that have been assigned to it.

An understanding of the separation of work requires a clear understanding of the separation of processors into sub-grids. The following explanation is provided.

The processors are linearly ordered from 0-(Nprocs-1). For the case of 8 processors, the nodes are referred to as 0 –7. An initial context (Icntxt) is formed in which each node is numbered between 0 and 7 as shown:

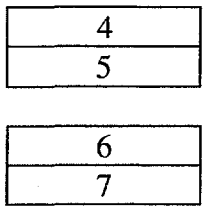
0	1	2	3	4	5	6	7

Then the processors are separated into new contexts (sub-grids) that are Nprow by Nlocalcol[1] in size. When Nprow=2 and Npcol=4, Nprocs = 8 processors, identified as 0 through 7 in the initial context (Icntxt), are split to form four new contexts designated as Localcntxt(k) :

0
1

2
3

[*] Presently, the value Nlocalcol exists within the codes but has been taken out of the CPLS_input.file file for user input. Nlocalcol defaults to 1 within the body of the code. This can be altered so that sub-grids with more than one column are formed. However, the need for multiple processor grids is derived from the large number of iterations around the number of samples, and the need for multiple processors in one sub-grid arises due to the amount of data, primarily that of a large number of samples. That is, by the time a user needs to invent multiple sub-grids that consist of more than one processor, the user will need to distribute the data due to the large number of samples in the data set--not due to the numbers of components or frequencies, which by comparison will at this point be small. Computation surrounding samples is largely performed on a row basis, while computation surrounding components or frequencies is largely performed on a column basis. Thus, defaulting the value of Nlocalcol to 1 seems valid.

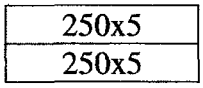


Nodes 0-1 form the first (sub-)context, or Localcntxt(1); nodes 2-3 form the second (sub-)context, or Localcntxt(2); nodes 4-5 form the third (sub-)context, or Localcntxt(3); and nodes 6-7 form the fourth (sub-)context, or Localcntxt(4). Finally, nodes 0, 2, 4, and 6 being the first node of each sub-grid or (sub-)context are joined in yet another context in order to communicate data between each sub-grid and node 0 (recall node 0 performs all of the input/output). This last context is the called Psumtxt for processor sum context and involves only nodes 0, 2, 4, and 6:

The effect upon the parallel processing can be understood by considering the work performed by one node in serial, two nodes in parallel, and eight nodes in parallel split into 2 sub-grids of four nodes. Examples are provided for a concentration matrix whose size is 500x5 or 500 samples by 5 components.

For an example using one (serial) node, the node would contain all of the concentration matrix; it would perform 5 repetitions (one per component) of the main computation loop 500 times (i.e. one for each sample).

For an example of two processors in an option 2 layout, a concentration matrix can be spread across a processor grid that is 2x1, or 2 processor rows by 1 processor column for a total of 2 processors (nodes) in the following manner:



Or, half of the matrix resides on Node 0 and the remaining half resides on Node 1.

For this choice of processor grid, 5 repetitions of the main computation loop will be performed 500 times by nodes 0 and 1—not simultaneously but serially. Note that this is not an improvement over a serial code that would perform 5 repetitions of the main computation loop 500 times; it should be less efficient because communication is required to send data between processor 0 and processor 1. The only reason this type of grid layout would be desirable is if the space of the concentration, spectral, and K matrices and other arrays required by the algorithm exceed the memory allocations of one processor.

For an example of two processors in an option 1 layout, node 0 contains all of the concentration matrix (and other matrices) as does node 1. But node 0 performs 5 repetitions of the main computation loop 250 times, while node 1 performs 5 repetitions

of the main computation loop 250 times. Node 0 performs calculations for samples 1-250; Node 1 performs calculations for samples 251-500. After completion of the computation, Node 0 outputs its calculations. Node 1 then sends its information to Node 0; Node 0 outputs Node 1's information. Some time is spent in transmitting initial and final information between nodes, but the total computation time has been effectively halved. The total time to run the code is then a combination of the time to perform the computation and the time involved in communication. If the communication time is small by comparison to the computation time, a total time savings is achieved.

The computational arrangement for the 8 processors shown above (option 2 layout) is to split Msize (or 500) by 4, since there are 4 contexts. Now, the first sub-grid (context) performs calculations for samples 1 to Msize/4, the second sub-grid performs calculations for samples Msize/4 to Msize/2, etc. Because all sub-grid computation requires communication within the local sub-grid (i.e. 2 nodes), this processor grid layout would be even less efficient than using 4 processors in an option 1 configuration of Nprow=1 and Npcol=4, if memory were not an issue. Thus, use of Nprow greater than 1 should be performed with caution.

Finally, the use of too many processors (even with option 1) can also lead to a decrease in performance time. If a decrease in computation time does not exceed an increase in communication time, no parallel speed-up will be achieved.

To restate this point: A greater number of processors may not provide any additional speed-up advantage due to communication used within the code. Examples of longer times resulting from the use of additional processors can be observed in the timing results provided in section 3.

Input/Output

All input/output (I/O) is performed via the lowest number processor. Node 0 needs to distribute and collect all information for input and output. Note that I/O is inherently serial.

Processor 0 is identified in a variety of ways:

1. On Icntxt:
   - Iam = 0
   - Mgr=0
   - Mgc=0
2. On localcontxt(k):
   - Myrow=0 & Mycol = 0 for k=1
   - Mlr=0 & mlc = 0 for k=1
3. On Psumtxt:
   - Ppr = 0
   - Ppc = 0

An example of the input process follows.

Processor 0 opens the data file and reads as instructed. For the binary unformatted data sets used by CPLS 1.0, the sizes of matrices are read first. Only processor 0 needs this information. A check on these values versus the matrix sizes reported in the CPLS_input.file file is made. If the numbers do not match, an error is reported by setting a flag in the code called Info to 1. The value of Info is broadcast to all other processors. If Info is 1, the code exits with an ERROR message; otherwise, if Info is 0, the code continues. Then processor 0 reads a column of the matrix and broadcasts the data to the other processors in Psumtxt. Psumtxt processors are the lowest number processors in the localcontxt(k), as described above. These processors receive the data, and within their sub-grid add the column of data to its matrix. (If option 2 is used, each sub-grid performs a communication operation to place the column of data across local processor rows.) This operation is repeated for every column of a matrix.

Output is performed through processor 0. If processor 0 does not contain all information needed for output, communication is performed in the opposite manner to input so that processor 0 is given the information prior to output.

Mb: Block Sizes—Distributing the Matrices

A matrix is distributed according to a two-dimensional block-cyclic distribution scheme for dense matrices. The above explanation implies that distributing the matrix across nodes 0 and 1 in the option 2 configuration places exactly the data for samples 1-250 on node 0 and for 251-500 on node 1. This is only the case when Mb = 250. The user is referred to Scalapack literature[2] for explanations of the 2-d block-cyclic distribution of dense matrices and of description arrays. The matrix is distributed in a block size (Mb) as defined by input from the user. Mb is the block sizes for Msize. Recommended values for Mb are 32 or 64 unless the data set is smaller than 32; in that instance, set Mb equal to Msize.

Msize, Nsize, Ksize, Nskipc, and Nskips--Data Sizes

The Msize value refers to the number of samples in the data collection. It is not the number of samples being analyzed—that value is Msize-Nskips. Thus, if your input data set from your spectrometer contains information from 40 samples, Msize is 40. If you wish to ignore 5 samples in the full calibration and cross-validation routines then Nskips is 5. You must then enter the 5 sample numbers to be ignored at the bottom of the CPLS_input.file. During computation, only 35 samples are used.

Likewise, Nsize is the number of components in the data collection. It is not the number of components being analyzed—that value is Nsize-Nskipc. If your input data collection contains information from 6 components and you wish to ignore 2 during the computation of a model, enter 6 for Nsize, 2 for Nskipc, and then enter the 2 component numbers you wish to ignore below Iscale in the CPLS_input.file. During computation only 4 components are used.

Ksize is the number of frequencies in the data collection. In reading in spectral data, the same samples must be skipped as were skipped during input of the concentration data, but no option for skipping frequencies is available.

The full calibration and cross-validation routines will use the same input values for Msize, Nsize, Ksize, Nskips, and Nskipc. The true prediction routine must use the same input values for (Nsize-Nskipc) and Ksize, but Msize-Nskips may differ as the prediction data collection may contain a differing number of samples than the calibration data set. (Note that the total Nsize-Nskipc must equal that from the full calibration and cross-validation routine—not Nsize and Nskipc, individually. These values may differ due to differing sampling techniques.) The true prediction routine can only provide predictions for Nsize-Nskipc components and Ksize frequencies, because these are the values the full calibration model and F-statistic are based upon.

Mstep—Step size for cross-validation

During cross-validation, a calibration model is built from the data without the use of all samples in the data collection. One or multiple samples (Mstep) may be withdrawn from the data collection. A calibration model is then built with Msize-Nskips-Mstep samples. If Mstep is two, the first two samples are removed from the data and a model is formed from the remaining samples. Then the next two samples are removed from the data and a model is formed from samples 1,2, and samples 5-Msize-Nskips.

Irecl--Record Length

The input value Irecl is machine dependent, and the user should consult the Fortran language reference manual for the particular machine being used in order to determine what value should be input into CPLS_input.file. The desired value to be input as Irecl is that which will enable reading of a 4 byte real. On a DEC Alpha, a record length of 1 is used to input or output a 4 byte real; thus, Irecl would be set to 1.

The user may design a short Fortran 90 code that uses the INQUIRE feature of the language in order to determine the record length for a particular computer system.

Note: It is the user's responsibility to provide little endian 4 Byte real data sets and make sure that the record length is compatible for the machine being used. Big endian storage and 8 Byte real values are not accommodated by the CPLS 1.0 code.

Output data formats

The user may choose to inspect the output data; therefore, the arrangement of each output file is provided below. In addition to the *_info that is an ASCI text file, there are two different types of output files—those containing real values and one containing double precision values. All calculated values pertinent to the user are supplied as 4 Byte reals in little endian storage. Real values are used because the original data collections had the precision level of real values. All calculated values for the full calibration model are output as double precision values, because the code promotes the reals to doubles and performs double precision operations. In order to continue computation in the true

prediction routine at a higher precision level, the *_pred_input.dat file contains double precision values.

In the following files, please note that Msize (or msize) is the value Msize-Nskips and Nsize (or nsize) is the value Nsize-Nskipc. That is, the ct23sa.abs file contained Msize = 34, but Nskips was set to 3. When the output files were written, the new Msize was 31.

*_pred_input.dat

The *_pred_input.dat file is an unformatted binary collection of double precision values:

Nend	rec=1
Nsize	rec=2
Newn	rec=3
Ksize	rec=4
Cvec(nsize)—concentration centering vector	rec=5 to 5+nsize-1
Avec(ksize)—spectral data centering vector	rec=5+nsize to 5+nsize+ksize-1
Kmatrix(newn*ksize)	rec=5+nsize+ksize to 5+nsize+ksize+ksize*newn-1
b(ksize*nsize*nend)	rec= 5+nsize+ksize+ksize*newn to 5+nsize+ksize+ksize*newn+ksize*nend*nsize-1
v(nsize*nend)	rec= 5+nsize+ksize+ksize*newn+ksize*nend*nsize to 5+nsize+ksize+ksize*newn+ksize*nend*nsize+nsize*nend-1
w(ksize*nsize*nend)	rec= 5+nsize+ksize+ksize*newn+ksize*nend*nsize+nsize*nend to end

*_full_output.dat

The *_full_output.dat file is an unformatted binary collection of real values:

Msize	rec=1
Nend	rec=2
Nsize	rec=3
Ksize	rec=4
Cpredc(msize*nsize)—concentration predicted during CLS	rec=5 to 5+msize*nsize-1
Cpredp(msize*nsize*nend)—concentration predicted during PLS	rec=5+msize*nsize to 5+msize*nsize*(1+nend)-1
fitc(msize)—spectral fit from CLS	rec=5+msize*nsize*(1+nend) to 5+msize*nsize*(1+nend)+msize-1
fitp(msize*nsize*nend)—spectral fit during PLS	rec=

5+msize*nsize*(1+nend)+msize to end

*_cross_output.dat

The *_cross_output.dat file is an unformatted binary collection of real values:

Msize rec=1
Nend rec=2
Nsize rec=3
Ksize rec=4
Cpredc(msize*nsize)—concentration predicted during CLS
rec=5 to 5+msize*nsize-1
Cpredp(msize*nsize*nend)—concentration predicted during PLS
rec=5+msize*nsize to
5+msize*nsize*(1+nend)-1
fitc(msize)—spectral fit from CLS rec=5+msize*nsize*(1+nend)
to 5+msize*nsize*(1+nend)+msize-1
fitp(msize*nsize*nend)—spectral fit during PLS rec=
5+msize*nsize*(1+nend)+msize
to 5+msize*nsize*(1+nend*2)+msize-1
pressc(nsize)—PRESS from CLS rec=
5+msize*nsize*(1+nend*2)+msize
to 5+msize*nsize*(1+nend*2)+msize+nsize-1
pressp(nsize*nend)—PRESS from PLS rec=
5+msize*nsize*(1+nend*2)+msize+nsize
to end

*_F_test.dat

The *_F_test.dat file is an unformatted binary collection of reals, which are 4 Bytes in size. The data has been output as reals, so that any difference in the record length between integers and reals on a computer system is not an issue. The first record contains the Nend value used during cross-validation; this value must match the Nend value used during true prediction if this file exists. The second record is the number of components or Nsize-Nskipc used during cross-validation. The remaining values, starting at record three, are the optimal factors for components 1 through Nsize-Nskipc.

*_PREDICTIONS.dat

The *_PREDICTIONS.dat file is an unformatted binary collection of real values:

Msize rec=1
Nend rec=2
Nsize rec=3
Ksize rec=4

Cpredc(msize*nsize)—concentration predicted during CLS
rec=5 to 5+msize*nsize-1
Cpredp(msize*nsize*nend)—concentration predicted during PLS
rec=5+msize*nsize to
5+msize*nsize*(1+nend)-1
fitc(msize)—spectral fit from CLS rec=5+msize*nsize*(1+nend)
to 5+msize*nsize*(1+nend)+msize-1
fitp(msize*nsize*nend)—spectral fit during PLS rec=
5+msize*nsize*(1+nend)+msize to end

* info

The time for running each routine is output into an ASCI text file named *_info. This file is first created by the full calibration routine; it must exist in order for the cross-validation and true prediction routines to function correctly. The user interested in recording times may wish to read the data reported in this file. Total timing starts above input and is completed after all output, thus I/O is incorporated into the total times reported. Calculation times are reported separately within each file; these times surround the main computation loop and do not include communication time (unless inherent to the computation as in the case of Nprow > 1).

CPLS Source and Miscellaneous Files

The CPLS 1.0 code is comprised of the following set of files:

1. CPLS_main.f
2. CPLS_calib.f
3. CPLS_cross.f
4. CPLS_predict.f
5. CPLS_test.f
6. Dsum.f
7. Dvsum.f
8. Pdsum_.c
9. Makefile
10. Readme

The input files are also part of the code. These files have been named and described above.

Makefile

The makefile for the CPLS 1.0 code on a DEC Alpha is provided below in order to identify necessary changes a user may need to make for use of the CPLS 1.0 code on another machine.

Compiler options such as optimization levels and debug, etc. are system dependent and must be supplied by the user. The user must also supply the path to the Scalapack directory in the variable SCALE. The PBLAS source code path is PBLAS_SRC. The archives for Scalapack, BLACS libraries must be listed under ARCHIVE. The math library has been given the variable MATH_ARCHIVE and is included in the archive list.

```
#=====================================================
#MAKEFILE FOR CPLS 1.0 CODE
#=====================================================
#User supplied parameters

#C Compiler
CC = cc

#F77 Compiler
F77 = f77

#F90 Compiler
F90 = f90

#compiler options
FFLAGS = -lmpi -lrt -pthread -g

#MPI directories
MPI_HOME = /usr/community/mpich
MPI_INC = -I$(MPI_HOME)/include
MPI_LIB = -L$(MPI_HOME)/lib/alpha/ch_p4 -lfmpi

#Scalapack directory
SCALE = /usr/community/SCALAPACK

#Pblas source directory
PB_INC = -I$(SCALE)/PBLAS/SRC

#BLACS library directory
BLAC = /usr/community/BLACS/LIB

#Machine type
MACH = ALPHA

#archives
MATH_ARCHIVE = /usr/lib/libdxml.a
ARCHIVE = $(SCALE)/scalapack_$(MACH).a \
     $(SCALE)/pblas_$(MACH).a \
```

```
        $(SCALE)/tools_$(MACH).a \
        $(SCALE)/redist_$(MACH).a \
        $(BLAC)/blacsF77init_MPI-$(MACH)-0.a \
        $(BLAC)/blacsCinit_MPI-$(MACH)-0.a \
        $(BLAC)/blacs_MPI-$(MACH)-0.a \
        $(BLAC)/blacsF77init_MPI-$(MACH)-0.a \
        $(BLAC)/blacsCinit_MPI-$(MACH)-0.a \
        $(SCALE)/pblas_$(MACH).a \
        $(MATH_ARCH)

#End of User Supplied Parameters
#====================================================
#INCLUDE
INCLUDE = $(MPI_INC) $(PB_INC)

#LIBRARIES
LIBS = -lm $(MPI_LIB)

# SOURCE FILES, OBJECT FILES, and HEADERS
FILES = CPLS_main.f CPLS_cross.f CPLS_calib.f CPLS_test.f \
        CPLS_predict.f dsum.f dvsum.f pdsum_.c

OBJECTS = CPLS_main.o  CPLS_cross.o CPLS_calib.o  CPLS_test.o \
        CPLS_predict.o  dsum.o dvsum.o pdsum_.o

HEADERS = header

EXECFILE = cpls

#HELP
help:
        @echo  "This makefile supports the following:"
        @echo  "make cpls:  creates executable parallel program"
        @echo  "make clean:  deletes *.o, core, and executable files"

#LINK and COMPILE
$(EXECFILE):  $(OBJECTS)
        $(F77)  $(OBJECTS)  $(FFLAGS)  $(INCLUDE)  $(LIBS)  $(ARCHIVE)  -o
$(EXECFILE)

#COMPILE
.c.o:  $$@.c $(HEADERS)
        $(CC) $(CFLAGS) $(INCLUDE) -c $*.c

.f.o:  $$@.f $(HEADERS)
        $(F90) $(FFLAGS)$(INCLUDE) -c $*.f
```

```
#UTILITY TARGETS
clean:
	- rm -f *.o core $(EXECFILE)
```

Readme

The Readme file provides a quick user's guide to the CPLS 1.0 code, particularly commenting upon the CPLS_input.file variables.

Section 3: The Test Data Set and Timing Results

An actual data set referred to as ct23sa.abs was used for timing of the three routines offered by the CPLS 1.0 code. The original IR spectroscopy data were collected from 34 samples for 6 components over 456 frequencies. The expected data collection should contain 4096 samples with 512 frequencies (future sets may contain up to 262,144 samples with 512 frequencies). The present data set is therefore relatively small in size, so that full advantage of parallel speed-up is difficult to demonstrate using this data set; however, it will be shown that speed-up was achieved.

To analyze this data set, the CPLS_input.file file in the CPLS directory was modified to read as follows:

'ct23sa'	sub-directory name
6	device out
34	value of msize
6	value of nsize
456	value of ksize
2	value of MB
1	value of NPROW
1	value of NPCOL
0	value of NADD
1	value of NBASE
2	value of NEND
1	value of NSKIPC
3	value of NSKIPS
1	value of MSTEP
1	value of IRECL
1	value of LSWITCH
1	value of ISCALE
5	column number(s) of components to skip in concentration
32	row number(s) of samples to skip
33	
34	

A sub-directory by the name of ct23sa must now exist within the CPLS directory. In that sub-directory, the file ct23sa_full_input.dat must exist. Because NADD is 0, the ct23sa_moreK.dat file need not be present.

Note that Msize is 34, Nsize is 6, and Ksize is 456. However, the routines will only perform calculations for Msize-Nskips or 31 samples and Nsize-Nskipc or 5 components.

Mb was set to values of 2, 8, and 16 in order to determine the effect of block size upon the computation time. No statistical difference was detected. The values for Iscale, Irecl,

Nadd, Nend, Nbase, Nskipc, and Nskips were not changed, but Lswitch was changed in order to run the three routines offered by the CPLS 1.0 code. While Nprow was held at 1, Npcol was varied between 1-6.

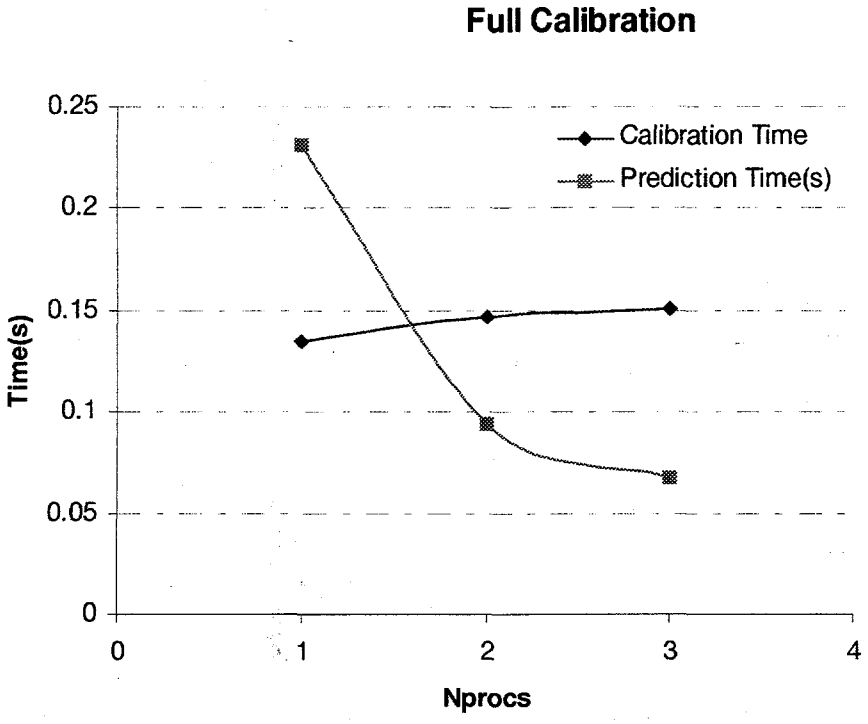
A DEC Alpha with 612 MHz Type EV56 CPU, 4.0 MB cache were used for the time tests. The machine clock granularity was tested and found to be ~1 ms. The operating system was Digital Unix V4.0D. An optimized BLAS library was used.

By running the full calibration code, the files ct23sa_pred_input.dat, ct23sa_calib_ooutput.dat and ct23sa_info were created in the ct23sa sub-directory. The time results for full calibration (Lswitch = 1) are reported in Table 2.

Table 2—Timing Results for full calibration

#Processors	Calibration Calculation Time(s)	Calibration Output Time(s)	Prediction Calculation Time(s)	Prediction Output Time(s)	Total Time(s)
1	0.1044	0.1864	0.2352	0.0566	1.1874
	0.1647	0.1903	0.2523	0.0663	1.3115
	0.1118	0.1912	0.2078	0.0585	1.0917
	0.0986	0.1959	0.2098	0.0575	1.1169
	0.1727	0.1903	0.2440	0.0556	1.3249
	0.1571	0.1869	0.2364	0.0556	1.3498
Average time	0.1349	0.1990	0.2309	0.0584	1.2304
2	0.1014	0.1887	0.0947	0.1561	1.2527
	0.1791	0.1893	0.0956	0.1385	1.2830
	0.1725	0.1970	0.0936	0.1229	1.2675
	0.1727	0.1864	0.0936	0.0810	1.1119
	0.1093	0.1922	0.0937	0.2061	1.6310
	0.1460	0.1893	0.0956	0.1385	1.2635
Average time	0.1468	0.1905	0.0945	0.1405	1.3016
3	0.1250	0.2174	0.0760	0.1677	1.3237
	0.1376	0.1894	0.0664	0.1647	1.2394
	0.1845	0.1895	0.0674	0.1172	1.2639
	0.1631	0.2139	0.0664	0.1259	1.2752
	0.1309	0.1902	0.0630	0.1195	1.1467

Average time	0.1647 0.1510	0.2137 0.2023	0.0683 0.0679	0.1342 0.1382	1.4234 1.2787



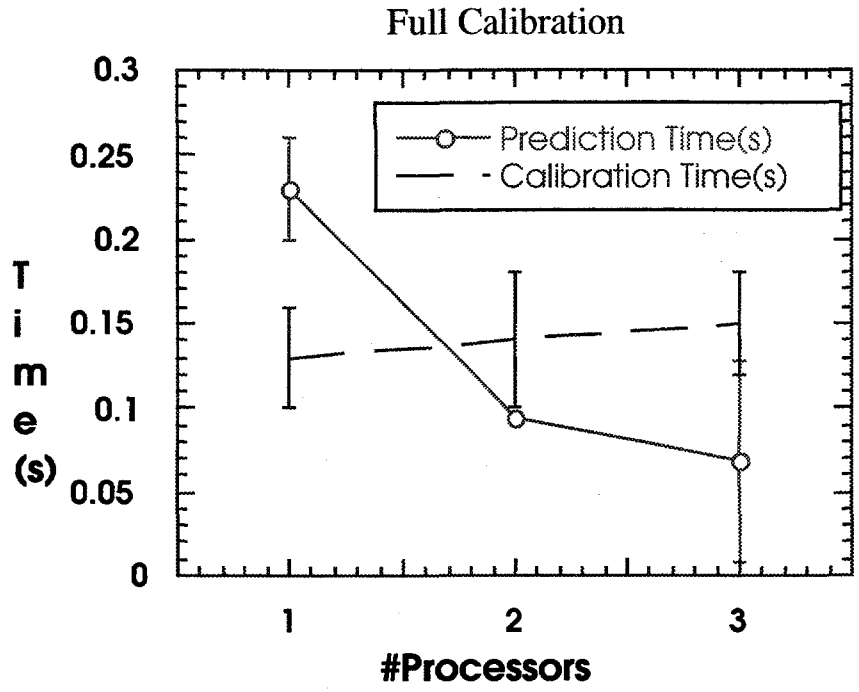


Figure 1—Timing remains constant for the calibration portion but decreases for the prediction portion of the full calibration routine as the number of processors is increased.

These results show that prediction calculations required the largest portion of total run time. By using two processors instead of one, the time for prediction calculations was effectively halved. But, the total time for running the code was not decreased—instead a (slight) increase in total time was observed. The data show that the time for the calibration part of the full calibration code did not change when two processors (i.e. nprow=1, Npcol = 2) are used versus one processor. This is because the work during the calibration part of the routine has not been distributed amongst processors. The prediction portion of the full calibration code has distributed the computation between the processors in use, and a reduction in the prediction calculation time is observed as the number of processors is increased. However, the reduced time for performing the parallel prediction calculations was offset by an increase in the output time due to communication of results between nodes. Thus, using more processors did not reduce the total time. Had the problem size been larger, total times should decrease (up to a point) as the number of processors increase.

A plot of calibration calculation time and prediction calculation time vs. number of processors (in option 1 layout) is shown in Figure 1. The constant calibration time is observed, as is the decreased prediction time with increased number of processors from 1 to 3.

After the cross-validation routine was run, the files ct23sa_cross_output.data and ct23sa_F_test.dat was created in the ct23sa sub-directory. The timing results for cross-validation were added to the end of the ct23sa_info file that already existed in the ct23sa sub-directory. The time results for the cross-validation routine, see Table 3, do show nearly 50% reduction in time to perform the calculations when two processors are used versus one processor (Lswitch = 2).

Table 3—Timing Results for cross-validation

#Processors	Calculation time(s)	Output time(s)	Total time(s)
1	1.4464	0.3124	2.2690
	1.4607	0.3162	2.2067
	1.4456	0.3152	2.1835
	1.4611	0.3291	2.2069
	1.4542	0.2706	2.1474
	1.4450	0.2953	2.1639
Average time	1.4522	0.3065	2.1962
2	0.7511	0.3093	1.5022
	0.7501	0.2615	1.4470
	0.7476	0.2045	1.3490
	0.7509	0.3316	1.5949
	0.7454	0.2574	1.4043
	0.7482	0.2498	1.4675
Average time	0.7489	0.2690	1.4608
3	0.5364		
	0.5348		
	0.5260		
	0.5338		
	0.5304		
	0.5309		
Average time	0.5320		
4	0.3884		
	0.3894		
	0.3855		
	0.4114		
	0.3919		

	0.3903		
Average time	0.3928		
5	0.3374		
	0.3373		
	0.3464		
	0.4179		
	0.3415		
	0.3416		
Average time	0.3537		
6	0.5651		
	0.2934		
	0.5774		
	0.5533		
	0.5040		
	0.5680		
Average time	0.5102		

Figure 2 shows that the calculation time on four processors was nearly the same as the calculation time recorded on five or six processors. The calculation time reported is from Node 0, and load imbalance is corrected first by the lowest node number (or lowest sub-grid number if option 2 is used) and then upward. Thus, when 31 samples are split across six nodes, Node 0 has to perform calculations for six samples while nodes 1-5 perform calculations for only five samples.

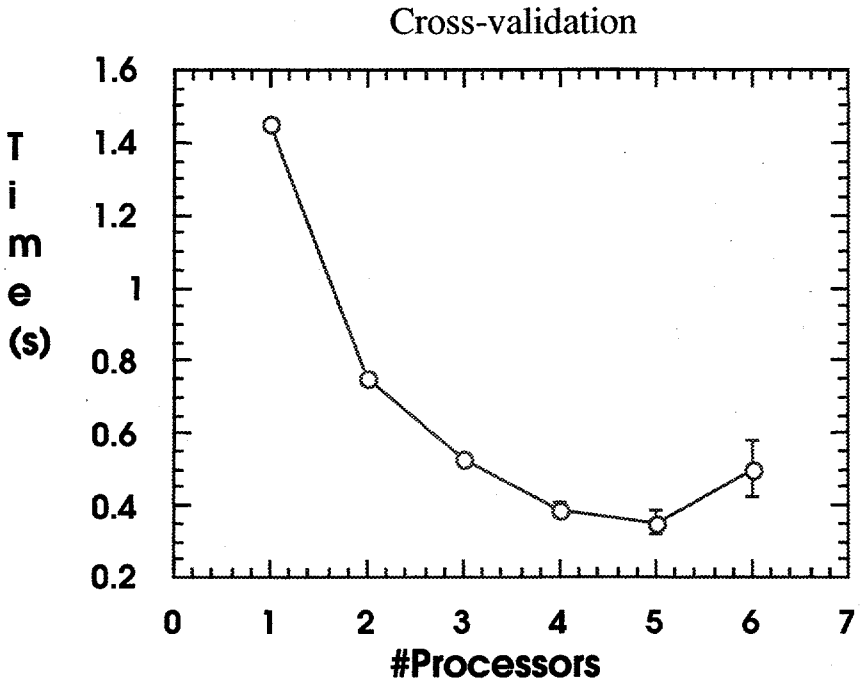


Figure 2—Computation time decreases during cross-validation until more than four processors are used.

The output time again adds significantly to the total time. In the case of three processors, the time increased significantly. This is a result of communication required to distribute the data to the three processors and again collect the data. It is shown that no advantage is obtained by increasing the number of processors beyond a certain limit because communication starts to swamp out the time gained during computation.

Without larger data sets, suggestions for optimal numbers of nodes and grid arrangements versus problem size can not presently be made.

After running the true prediction routine, the file ct23sa_PREDICTIONS.dat was created in the ct23sa sub-directory and timing results were appended to the existing ct23sa_info file.

Timing results from the true prediction routine are presented in Table 4. The average time for true prediction (Lswitch=3) calculations using one processor was 0.175 s, while the total time for the routine was ~1.1s. By increasing the number of processors, a decrease in true prediction calculation time was observed. Again, for the small problem size, the decreased calculation time was quickly offset by the increased communication time so that the savings in total time was no greater than ~10% when two processors were used instead of one. A plot of the true prediction time vs. number of processors in shown in Figure 3.

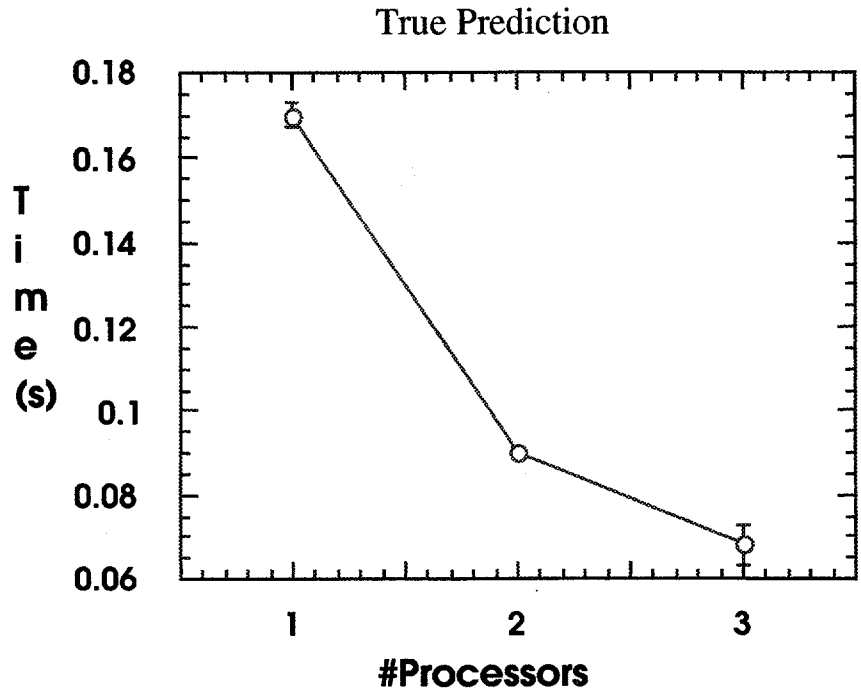


Figure 3—Nearly parallel speed-up is observed for true prediction computation.

Table 4—Timing Results for True Prediction

#Processors	Calculation time(s)	Output time(s)	Total time(s)
1	0.1737	0.2244	1.5530
	0.1756	0.1294	0.9423
	0.1764	0.1182	0.9765
	0.1738	0.1719	1.0423
	0.1752	0.1122	0.9258
	0.1766	0.1629	0.9800
Average time	0.1752	0.1532	1.0700
2	0.0928	0.1250	0.9235
	0.0945	0.1033	0.8255
	0.0936	0.1157	0.8506
	0.0927	0.1054	0.8584
	0.0956	0.2016	1.0125
	0.0926	0.1088	0.8587
Average time	0.0936	0.1266	0.8882
3	0.0672	0.1326	1.0021
	0.0673	0.1180	0.8726
	0.0673	0.1054	0.8173
	0.0683	0.1879	1.1913
	0.0683	0.1278	0.9190
	0.0663	0.1169	0.8340
Average time	0.0674	0.1314	0.9394

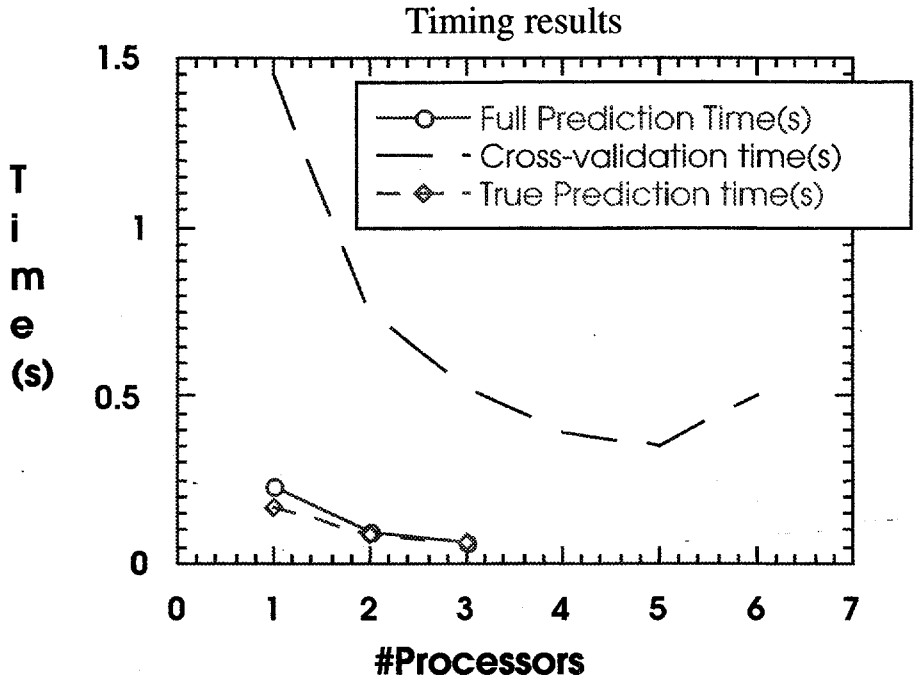


Figure 4—A comparison of timing results for the parallel computation portions of each routine in the CPLS 1.0 code shows that the cross-validation routine is the most time intensive routine.

Figure 4 contains a comparison of timing results for the parallel portions of each routine. Note that the cross-validation routine which calibrates and predicts on each sample in the data collection is the most costly routine time-wise. The full calibration routine is parallel in its prediction portion. Note that the times for the full calibration prediction routine virtually overlap those of the true prediction routine—this should occur as the two routines were run for the same data collection and effectively perform the same work.

Summary

The CPLS 1.0 code, developed for quantitative analyses of spectral data using the recently developed prediction-augmented CLS/PLS hybrid algorithm, is described. This parallel code offers the user three routines—full calibration, cross-validation, and true prediction—all necessary steps for performing full quantitative analyses. As demonstrated on a small but representative data set, the code achieves parallel speed-up for the parallel computation portions of the routines.

References

1. Private communication, David M. Haaland and David K. Melgaard (1999).
2. Scalapack Users' Guide, L. Susan Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, Society for Industrial and Applied Mathematics (1997) Philadelphia.
3. "Partial Least-Squares Methods for Spectral Analyses. 1. Relation to Other Quantitative Calibration Methods and the Extraction of Qualitative Information", David M. Haaland and Edward V. Thomas, *Analytical Chemistry* (1988) vol. 60, p. 1193

Distribution

1	MS 0149	LDRD Office, 4001
3	MS 0342	David Haaland, 1812
5	MS 0342	Frederick Koehler IV, 1812
3	MS 0342	Michael Keenan, 1812
1	MS 0342	Christine Wehlburg, 1812
5	MS 0977	Celeste A. Drewien, 6524
2	MS 1134	David Melgaard, 1835
1	MS 0612	Review and Approval Desk, 9612 for DOE/OSTI
2	MS 0899	Technical Library, 9616
1	MS 9018	Central Technical Files, 8940-2