

A Parallel Algorithm for Matched Field Processing Using MPI and VSIPL

Randall R Judd

SPAWAR Systems Center, San Diego

Abstract

BACKGROUND

This describes a demonstration of a port of a single thread algorithm to a parallel environment using open standards for a portable solution. A single thread matched field algorithm was implemented using the VSIPL library standard. The algorithm was ported to a parallel environment using the MPI standard.

Matched Field Processing (MFP). MFP is a beamforming method, where an estimated field derived from array data is correlated with a reference field. The reference field is calculated using a normal mode model. A higher correlation between the estimated field and the reference field indicates a higher probability that the data target location is given by the reference source origin. By correlating the reference field with the estimated field, for a grid of reference sources, a target location is estimated.

Vector/Signal/Image Processing Library (VSIPL). VSIPL is an open standard for a math library for vector, signal, and image processing. VSIPL is a relatively new standard with version 1.0 completed in March of 2000; however, several vendors have completed partial implementations and an open source (generic) version is available for general use at the VSIPL web site (www.vsipl.org). The generic VSIPL implementation was used for this study.

Message Passing Interface (MPI). MPI is a mature open standard for message passing on a parallel computer. Commercial and free open source versions are available. The MPICH implementation of MPI was used on a Linux cluster and a vendor-supplied implementation of MPI was used on a Hewlett Packard V2500 SMP.

PRESENTATION

Although there are several technical goals to the MFP algorithm study, the primary subject for this talk is to explore the VSIPL standard in a parallel environment.

First, the matched field processing algorithm and its conversion to a parallel environment will be briefly covered. One of the primary purposes of the study is to determine if MFP is ready for transition to operational systems. MFP has been studied for over 20 years in the research community but has been too compute intensive for use in operational systems. With the advent of faster, inexpensive processors and parallel computers, the computational limitations may no longer be valid.

Second, the communication of VSIPL with external libraries will be described. In particular, communication between VSIPL and MPI will be described. Although VSIPL objects and data are opaque, a method has been included in the VSIPL library which allows VSIPL to associate a standard data array in memory with a VSIPL block. Since a pointer is available to the data array, the pointer may be used to communicate with functions external to VSIPL.

Third, methods for creating and using VSIPL views and updating the views on a parallel process will be discussed. VSIPL allows the creation of VSIPL views on blocks associated with a NULL data pointer. When the VSIPL block is rebound to a valid data pointer and the data is admitted to VSIPL, all the views become valid. Using this method and a message passing library, it is possible to stage the necessary VSIPL objects at the beginning of the algorithm and then update and use the objects on a parallel system as data becomes available.

Finally, portability of the application will be discussed. The application code was developed on a simple cluster of four dual processor SMP 450-MHz Pentiums (8 total processors) with Linux as the operating system and using 100 BaseT Ethernet for communication between SMPs. The completed application was then ported to a Hewlett Packard V2500 sixteen processor SMP running HPUX.

A Parallel Algorithm for Matched Field Processing using MPI and VSIPL

Randall Judd
SSC-SD D857
judd@spawar.navy.mil
(619) 553-3086

Sponsors
Clair Guthrie - NAVSEA PMS 411
John Tague - ONR 321US

Goals

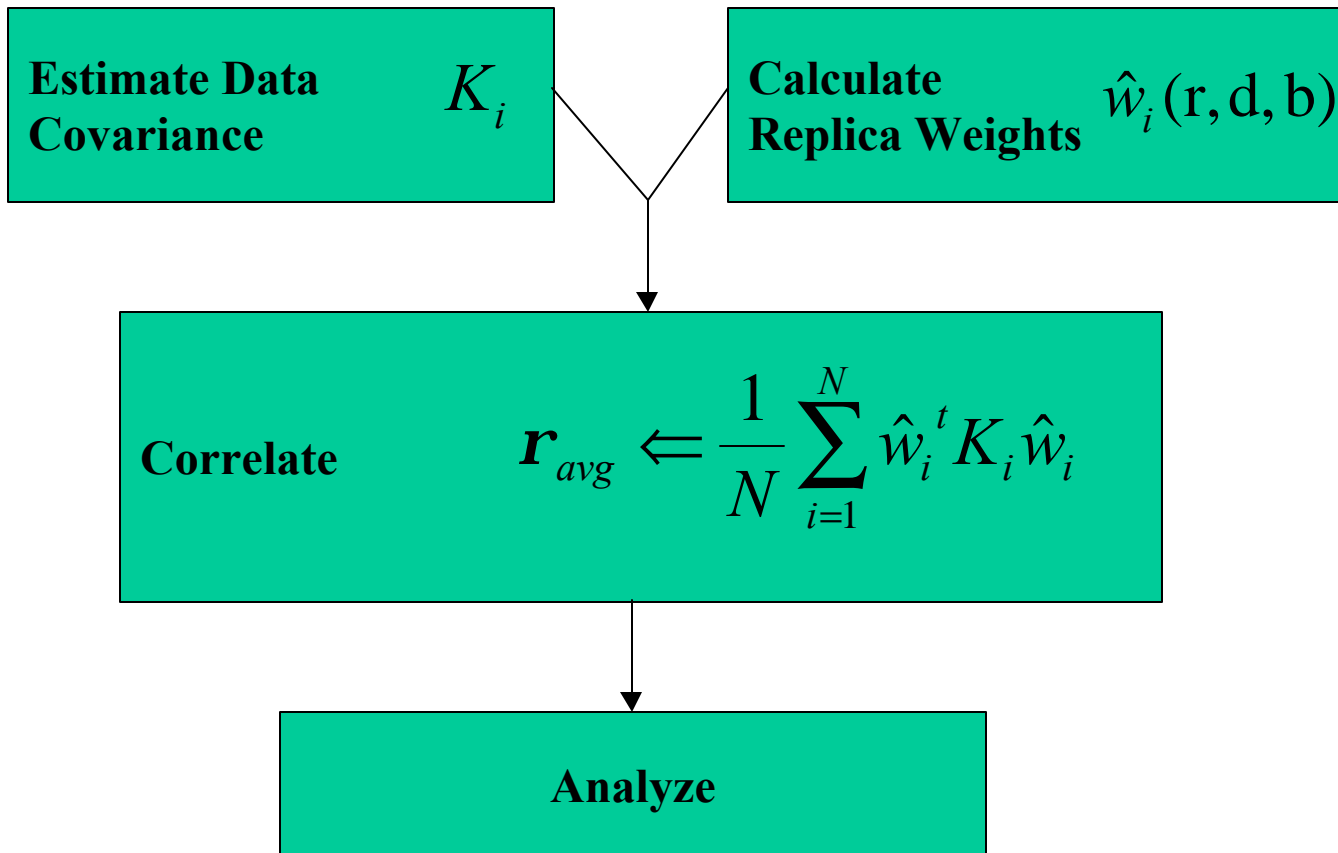
- **Demonstrate Elements of the TASP COE in a substantial algorithm**
- **Produce an easy to use MFP application which inputs time series data at one end and outputs matched field correlation data at the other**
- **Investigate the computational power required to produce real time MFP output in an operational system**
 - **How do we transition MFP to the fleet?**

Matched Field Processing

- **What is it?**
 - It is a generalization of conventional beamforming
- **How is it done?**
 - The acoustic pressure field is estimated at each sensor by measuring the sensor response
 - The pressure field is calculated at each sensor using the solution of the wave equation in a waveguide for a grid of assumed source locations
 - The estimated field is correlated with the calculated field for each grid points
 - The maximum correlation on the grid usually corresponds to the target location
- **What are the advantages over conventional beamforming?**
 - It provides a range, depth, and bearing to a target
 - It works in the near field

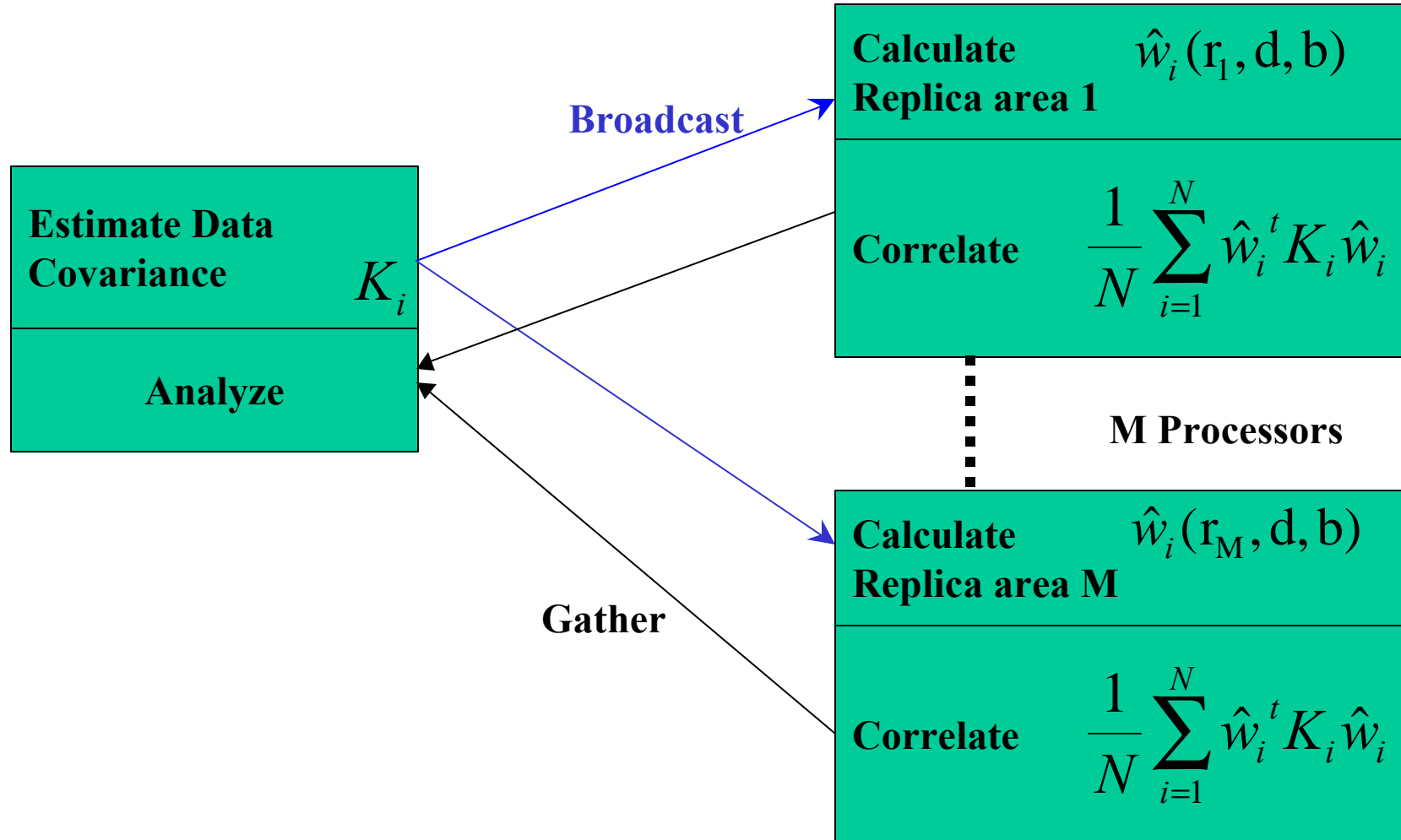
Single Processor Algorithm

for $i=1$ to N frequencies



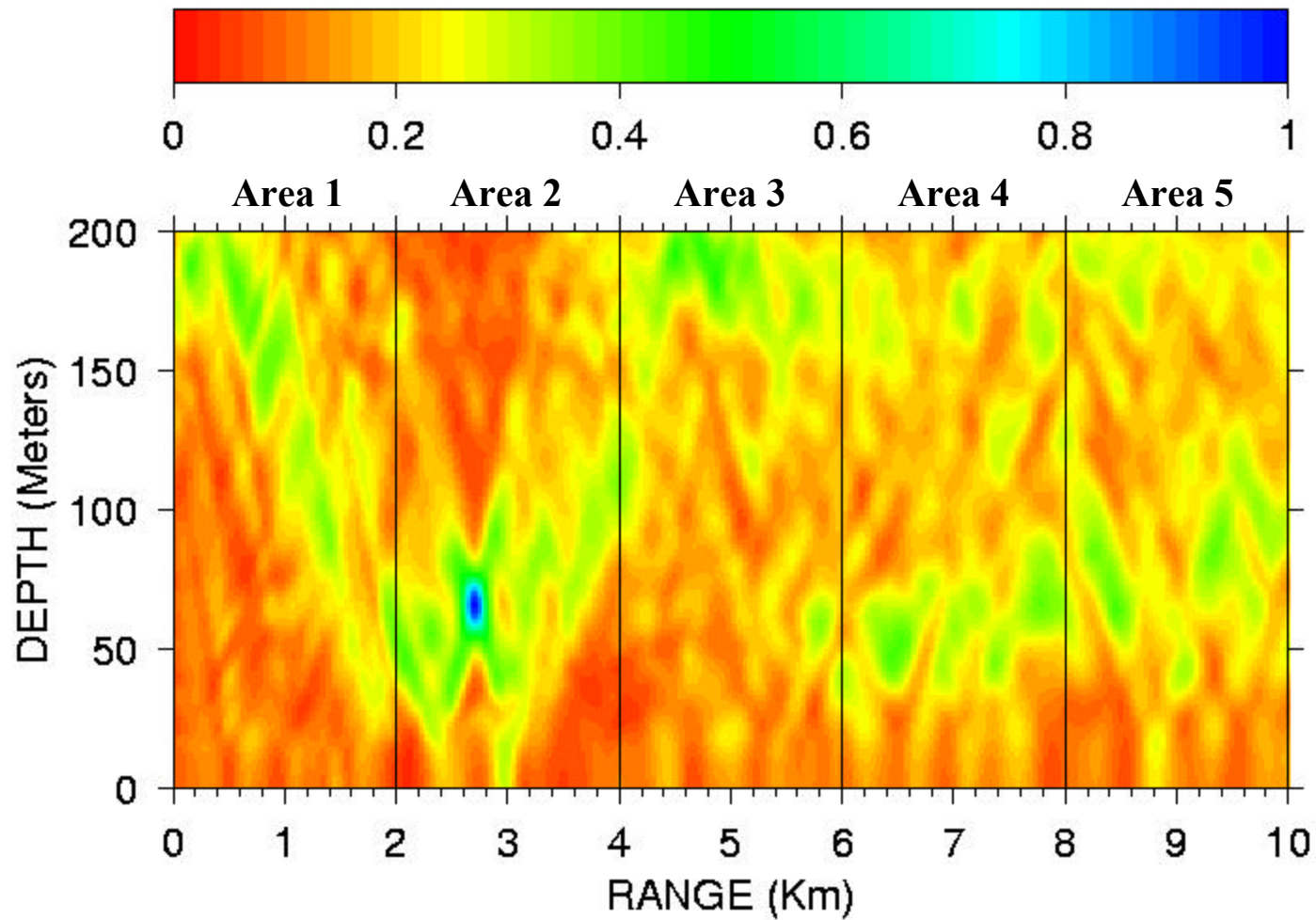
Parallel Processor Algorithm

for $i=1$ to N frequencies



Search areas for 5 process search

Normalized MFP Surface



MPI Functionality

- There are only six MPI calls used in the program.
 - MPI_Init and MPI_Finalize
 - MPI_Comm_size and MPI_Comm_rank
 - MPI_Bcast
 - To broadcast the covariance matrix
 - MPI_Gatherv
 - To collect the correlation values for analysis.
- Broadcast and Vector gather require a pointer to a memory buffer and a buffer length.
 - VSIPL blocks, where data are stored, are opaque and don't work with MPI calls directly
 - VSIPL supplies mechanisms which allow for I/O from the library using standard C buffers

Input Methodology for VSIPL

- **Memory is allocated by the user using some standard method**
 - VSIPL calls this “user memory”
- **Data layout in memory is defined in the VSIPL specification**
- **The memory is associated with a VSIPL *block***
- **An *admit* function gives ownership of the data to VSIPL and (if requested) forces the data in the block to be the same as the data in the associated memory**
- **All views (matrix, vector) of the block are valid after the block is admitted**

Output Methodology for VSIPL

- A VSIPL *block* which has previously been associated with user memory and admitted is now full of data for output.
- The *block* is *released* and (if requested) the data in user memory is forced to agree with the data in the block.

VSIPL I/O



Data Input to VSIPL

- Variables for time series data input
 - `tsdata` = program specific structure for time series input
 - `data` = pointer to series data array in proper format for VSIPL
 - `data_block` = VSIPL block, setup for use with user data
- Input the data
 - ```
float *data = mfp_tsdatap_getdata_f(tsdata);
vsip_blockrebind_f(data_block,data);
vsip_blockadmit_f(data_block,VSIP_TRUE);
```

# Spectral Estimate

- **Variables for spectral estimate**
  - **window** = real vector view
  - ts\_data** = real matrix view of data\_block (channels by length)
  - cfft\_data** = complex matrix view (channels by length/2+1)
  - rcfftm** = VSIPL multiple FFT object (VSIP\_ROW)
- **Calculate the spectrum**
  - `vsip_vmmul_f(window, ts_data, VSIP_ROW, ts_data);`
  - `vsip_rcfftmop_f(rcfftm, ts_data, cfft_data);`
- **Give the time series data array back to the program**
  - `vsip_blockrelease_f(data_block, VSIP_FALSE);`

# Covariance Matrix Distribution

- Create a user data array of proper size ( $2 * \text{nfreq} * \text{nchan}^2$ ) on each process.
  - There is one covariance matrix for each frequency
  - The covariance matrix is square of size nchan.
- On each process associate a block with the user data then bind matrix views of size nchan with offsets so the matrix view are side by side in the block.
- Admit the block on process zero, estimate and average the covariance matrices as required, and release the block.
- Broadcast the user data array. Each process has an identical set of covariance matrices and they are valid when the block is admitted.
- On process zero estimate the next covariance matrix. For processes greater than zero estimate the correlations.

# Correlation Calculation

- Create an array containing search information for each process
  - Index the array by the processor number
  - Each process uses the array to determine its own search area
- Create an array (`int *ncounts`) with the number of correlation values estimated by each process.
  - Index the array by the process number
  - Process zero allocates space for the entire correlation array but does not do correlation estimates
- After the correlation array for each node is complete, vector gather all the nodes into the node zero array for analysis
  - `MPI_Gatherv(mfp_get_corr_f(param), ncounts[myid], MPI_FLOAT, mfp_get_corr_f(param), ncounts, ndspls, MPI_FLOAT, 0, MPI_COMM_WORLD);`

# Portability

- **This particular algorithm is portable for workstation clusters and SMP's**
  - Network of PC's (4 Nodes, 8 CPU's) running Linux, MPICH, and TASP VSIPL
  - Eight processor Sun Enterprise 4000 running Solaris, MPICH and TASP VSIPL
  - Hewlett Packard V250 with 16 processors running HPUX, HP's MPI and TASP VSIPL
- **But**
  - Need to demonstrate portability between workstation cluster and embedded
  - Need to demonstrate portability between multiple implementations of VSIPL

# Conclusion

---

- **VS IPL works well in a message passing environment such as MPI**
  - as long as you are only passing data associated with VS IPL Blocks
  - VS IPL does not currently support passing more complicated objects such as QRD or SVD objects