Using the Vector, Signal and Image Processing Library with OpenMP and MPI

CS 425 Brice Lambi

Introduction

This paper is a review of two publications about using the Vector Signal and Image Processing Library (VSIP) with two other programming tools. The first tool is OpenMP, and the second is MPI. Using VSIP with parallel programming tools is non-trivial and this paper will review several MPI interoperability solutions proposed by Purushotham Bangalore, and an OpenMP solution implemented by Jeremy Kepner.

Vector Signal and Image Processing Library

VSIP is an open standard computational ANSI C API useful for signal and image processing. The VSIP API consists of a collection of data types and support functions. All memory storage used in the VSIP API is abstracted and only accessible through blocks. A block is a portable method used for accessing data. A programmer must access all data in a block using the VSIP accessor functions. Memory allocation using VSIP block functions is similar to malloc in the sense that it will allocate the appropriate amount of space but it will not initialize the data referenced by the block.

Before a programmer can make use of a block it must be bound to a view. A view is the object that is used to access data as a vector, matrix or
A view cannot change its block once it is created. Every view has an offset which is the index into the block of the first element the view will access. This is important because when using MPI, data must be accessed directly.

**Exploiting VSIPL and OpenMP for Parallel Image Processing**

OpenMP is an open standard C/Fortran API that allows programmers to write portable thread based parallelized code for shared memory systems. OpenMP and VSIPL can be used together to create image processing algorithms that will utilize the image processing capabilities of VSIPL and the thread based parallelism offered by OpenMP.

Image convolution follows this outline: for each pixel in the image, pixels nearby are multiplied by some function and summed together to get the value of a new pixel. Image convolution is used to reduce blur and remove noise. 2-D image convolution is achieved by using a Fast Fourier Transform algorithm whose inputs are an image and a set of functions used for the image convolution. This image convolution algorithm has been implemented in the Interpreted Data Language (IDL) environment, most image processing applications are written in IDL. IDL implements this algorithm using VSIPL for convolution processing and OpenMP for parallelism.

Each function passed to the image convolution algorithm must be convolved. So the simplest way to parallelize this algorithm would be to
convolve each function in a separate thread and use one thread to combine all the results. When the input is passed to the IDL routine, IDL will then pass pointers to an external C function where OpenMP will do the parallelization work. Each thread will execute its convolution using VSIP functions, the threads will be joined and the results will be collected. A pointer to the final image will be returned to the IDL environment.

This algorithm was developed and tested by Jeremy Kepner on a SGI Origin 2000, 64 300MHz MIPS 10000 processors with an aggregate memory of 16GB. The code was run using IDL version 5.3, SGI's OpenMP compiler and the TASP implementation of VSIP. Results of running this algorithm show linear speedups for 1-32 processors. Unfortunately the paper by Jeremy Kepner did not mention how many threads were used in the testing of this algorithm.

**Interoperability of VSIP with MPI**

The main issue when combining VSIP and MPI is memory access. A programmer cannot directly access data stored in VSIP blocks but the programmer needs direct access to that data before it can be used with MPI send and receive functions. To address this issue Purushotham Bangalore proposed several solutions.

The first solution is the simplest. Start by copying the data from a
VSIPL block using the VSIPL accessor functions into a public data array that can be accessed directly by the programmer. Broadcast the data in the new data array on the send side, on the receive side read the data into a data array and then copy that data array into a private VSIPL abstract data type. The problem with this approach is overhead. Copying of data from private to public storage twice means extra copies of the data and time spent doing the extra copying.

The second solution is to use admit and release functions provided by the VSIPL standard. Admit and release functions offer a means to access the private VSIPL data directly. The admit function locks the VSIPL block into VSIPL ownership and allows the data only to be accessed through the VSIPL API. The release function unlocks the block and allows the private data to be accessed only by user activities. Using this method the programmer will release data from an abstract data type to a public array, send/receive the array and then admit the public array to the abstract data type. This appears to be the most reasonable solution to the VSIPL/MPI interoperability problem.

A third solution proposed by Bangalore is to develop a separate data type language and library specification. MPI provides a sophisticated data type language and VSIPL provides a useful gather/scatter specification. A new specification would offer support for both. The disadvantage to this approach is complexity. Although this may be a convenient long term solution, it is not practical for immediate implementation.
Summary and Conclusion

VSIP is a C API that uses data abstraction to store and process user data. OpenMP is a thread based parallelization toolkit. MPI is message passing API that requires data be viewable publicly in order to send/receive that data. There are several uses for parallelizing VSIP programs using OpenMP or MPI.

OpenMP can be used to parallelize image convolution. Jeremy Kepner achieved linear speedup using OpenMP to parallelize image convolution using VSIP in the IDL environment.

It is necessary for a programmer to implement their own methods of making MPI function with VSIP. Using the admit and release functions, programmers can avoid the overhead of copying VSIP data from private to public data arrays. If a programmer would like a simpler solution they can explicitly copy data from private VSIP data arrays into public data arrays. When the data is in a public data array it can then be used for MPI sends and receives. Another, less feasible, solution is to implement a new library that will handle data gather/scatter as well as a data type language without requiring the programmer to worry about copying private data to public arrays and vice versa.
References

