Parallel Maximum-Likelihood Inversion for Estimating Wavenumber-Ordered Spectra in Emission Spectroscopy

Hoda El-Sayed

Marc Salit
marc.salit@nist.gov

John Travis
john.travis@nist.gov

Judith Devaney
judith.devaney@nist.gov

William George
william.george@nist.gov

National Institute of Standards and Technology
Gaithersburg, Maryland USA

Abstract

We introduce a parallelization of the maximum-likelihood cosine transform. This transform consists of a computationally intensive iterative fitting process, but is readily decomposed for parallel processing. The parallel implementation is not only scalable, but has also brought the execution time of this previously intractable problem to feasible levels using contemporary and cost-efficient high-performance computers, including an SGI Origin 2000, an SGI Onyx, and a cluster of Intel-based PCs.

Key words : parallel processing, emission spectroscopy, cosine transform, maximum-likelihood inversion, performance evaluation, DParLib, MPI.

1 Introduction

Michelson interferometers used in optical spectroscopy produce interferograms that can be considered a superposition of cosine functions. The optical spectrum is estimated from this interferogram using some transform method, typically the Fourier transform, a least-squares procedure, which presumes normally-distributed noise. A more general maximum-likelihood cosine transform has been demonstrated to have several advantages over the Fourier transform when estimating the spectrum for typically sparse emission spectra. In particular, a maximum-likelihood inversion derived for a Poisson noise distributed signal has been demonstrated to eliminate the often troublesome noise redistribution associated with the Fourier transform of such signals [2]. The Fourier transform distributes all noise in the signal as white (the spectral character of the *noise*) while the maximum-likelihood cosine transform distributes the signal-carried noise to the signal, preventing the masking of small spectral features by the signal-carried noise from the large spectral features. The spectral estimates obtained using maximum-likelihood inversion have another potentially useful property—a line-shape which is burdened with a less distorting transform-function than the sinc function of the Fourier transform.

In this paper we present a parallel implementation of the maximum-likelihood inversion method. It will be shown that the parallel implementation is not only scalable, but has also brought the execution time of this problem to feasible levels using contemporary and cost-efficient high-performance computers including Origin 2000, SGI Onyx, and PC clusters. By parallelizing this application, we were able to reduce the running time of the program to seconds rather than hours. The rest of this paper is organized as follows: Section 2 describes the maximum-likelihood inversion and the sequential code implementation, Section 3 describes the computational methodology, where it outlines the sequential and the parallel algorithms, as well as their run-time cost, Section 4 shows the experimental results obtained on the different machines used, and Section 5 contains discussion and future work.

2 Maximum-Likelihood Inversion

The method of maximum-likelihood (ML) was first described in a 1922 paper by Fisher [4]. It is a method for estimating parameters for a model given a set of observed data. The essential feature of ML is to find a set of values for the model parameters for which the probability of observing the given data is highest. In this application, we model the data as the superposition of cosine functions and the parameters to be estimated by ML are the coefficients of those cosine functions.

The ML inversion method has been proposed for use in processing interferograms by Bialkowski [2]. The expectation-maximization (EM) algorithm used to compute the maximum likelihood is described by Shepp, Vardi, and Kaufman [11, 12] and also by Lange and Carson [10]. The reader is referred to those papers for more details on the development of the EM algorithm. The basic algorithm consists of iterating the following two equations:

$$\hat{F}(d_j) = \sum_{i=1}^{I} g(d_j; \hat{v}_i)\hat{a}(\hat{v}_i) \tag{1}$$

$$\hat{a}^{k+1}(\hat{v}_i) = \hat{a}^{(k)}(\hat{v}_i)\left[\frac{\sum_{j=1}^{J} \frac{g(d_j;\hat{v}_i)F(d_j)}{\hat{F}^{(k)}(d_j)}}{\sum_{j=1}^{J} g(d_j;\hat{v}_i)}\right] \tag{2}$$

where $\hat{F}(d_j)$ is the estimated interferogram, the $\hat{a}(\hat{v}_i)$ are estimates of optical power, $g(d_j; \hat{v}_i)$ are the basis set functions, $\hat{a}^{(k)}$ and $\hat{a}^{(k+1)}$ are parameter estimates after the *k* and *k+1* iterations and the $g(d_j; \hat{v}_i)$ are normalized on the d_j interval. $\hat{F}^{(k)}(d_j)$ is the expectation of the interferogram for iteration *k*. Initial parameter values, $\hat{a}^{(1)}$, are typically set to unity in the computations here. Convergence occurs when the sums to the right of $\hat{a}^{(k)}$ approach 1.

3 Computational Methodology

The algorithm was first implemented sequentially to get an initial baseline of the properties with respect to speed and to look at the results in terms of the physics. Figure 1 gives an outline of the serial algorithm. There are N_{pts} in the interferogram which is denoted $F(d_j)$, where the d_j are the points. There are N_{fr} values of frequency which are denoted $\hat{v}_i$. CL and FDR are constants. The maximum-likelihood inversion method uses two input files, one contains the frequencies of interest and the other contains the interferogram points.

In each iteration, we perform a computation based on all data points (N_{pts}) for each of the frequencies of interest (N_{fr}) leading to a sequential time complexity of $\mathcal{O}(N_{fr}N_{pts}N_{iter})$. Following the sequential implementation, we created a parallel algorithm. Since the number of points in the interferogram is much larger than the number of frequencies of interest, we decided to parallelize across the interferogram. This turns this problem into a data-parallel SPMD computation.

Two libraries supporting the data-parallel programming model, using the Message Passing Interface (MPI), have been developed at the National Institute of Standards and Technology. One was written for Fortran 90 programs, known as "F90-DParLib" [3, 7], and the other is written for C programs, known as "C-DParLib" [5]. Both of these libraries use MPI for all inter-process communication. The use of this industry standard library, MPI, ensures portability of these libraries with respect to message passing operations.

N_{pts} = (program parameter)
N_{fr} = (program parameter)

read $\hat{v}_i(N_{fr})$
read $RawInterf_j(N_{pts})$

$CosSpectrum_i(N_{fr}) = 1.0$
$d_j(N_{pts}) = (j - 0.5 * N_{pts})\frac{CL}{FDR}$

$cbasis_{i,j}(N_{fr}, N_{pts}) = 0.25 * (1 + \cos(2\pi * \hat{v}_i * d_j))$
$CosSum_i(N_{fr}) = \sum_j^{N_{pts}} cbasis_{i,j}$

For each iteration
BEGIN
  $OldCosSpectrum_i(N_{fr}) = CosSpectrum_i$

  For all points j
  BEGIN
    $CosInterf_j(N_{pts}) = \sum_i^{N_{fr}} cbasis_{i,j} * OldCosSpectrum_i$
  END

  For all frequencies i
  BEGIN
    $CosProduct_i(N_{fr}) = \sum_j^{N_{pts}} cbasis_{i,j} \frac{RawInterf_j}{CosInterf_j}$

    $CosSpectrum_i(N_{fr}) = \frac{OldCosSpectrum_i}{CosSum_i} CosProduct_i$
  END
END

print $CosSpectrum_i(N_{fr})$
print $CosInterf_j(N_{pts})$

Figure 1. Pseudo-code for the sequential maximum-likelihood inversion algorithm.

Since the serial version of the maximum-likelihood algorithm was implemented in C, C-DParLib was used to support the parallel implementation. The data-parallel programming paradigm supported by C-DParLib allows programmers to specify operations on arrays rather than operations on individual array elements, thus producing a simple SPMD (single program-multiple data) style of program. The most important functions provided by C-DParLib allow the programmer to specify the general distribution scheme for dividing the data arrays among the processors, such as block and block-cyclic data distribution along one or more array axes, without requiring the specification of the machine size. All specific distribution parameters, such as how many array elements are assigned to each processor are determined at runtime by the library. This frees the programmer to concentrate on the algorithm and not the details of array distribution. For many parallel algorithms, this data distribution service alone greatly simplifies the programming task.

In addition to handling all of the details of data distributions, other commonly needed data-parallel operations are also supported by C-DParLib such as array shifting, elemental operations, and array reductions. For operations that require communications, the library caches information for later reuse so that some of the overhead cost can be amortized over multiple operations. For operations not directly supported by the library, the programmer has access to all of the array distribution details needed to correctly access the distributed arrays and may use any of the MPI routines directly as needed.

We used this library to simplify the parallelization of the maximum-likelihood algorithm across interferogram points. Each processor receives a copy of all of the frequencies, and one section, of size N_{pts}/P, of the interferogram where P is the number of processors used. This parallelization resulted in a single communication step per iteration. In this step, the interferogram is summed across all frequencies. The sums are performed locally, and then a global reduction is performed. Thus the communication cost stays constant as the size of the interferogram grows, but grows as the number of frequencies is increased. Figure 2 gives an outline of the parallel implementation.

Time complexity depicts how the time requirement of the algorithm grows as a function of the problem size. Parallel time complexity depends upon the data size and the number of processors used. In parallel computing the overall time depends on the time to perform the computation as well as the time for the processors to communicate. This communication time is considered a pure overhead and could become a major problem for scalability if communication requirements grow rapidly with either an increase in the problem size or an increase in the number of processors. Using the previous definitions, the computational and communication costs are estimated respectively as:

$$T_{comp} = \mathcal{O}(N_{iter} N_{fr} \frac{N_{pts}}{P}) \quad (3)$$

$$T_{comm} = \mathcal{O}(N_{iter} N_{fr} P) \quad (4)$$

Hence, the total estimated time complexity, T, of this algorithm, which is defined by $T = T_{comp} + T_{comm}$ is

$$\begin{aligned} T &= \mathcal{O}\left(t_{comp} N_{iter} N_{fr} \frac{N_{pts}}{P} + t_{comm} N_{iter} N_{fr} P\right) \\ &= \mathcal{O}\left(N_{iter} N_{fr} (t_{comp} \frac{N_{pts}}{P} + t_{comm} P)\right) \quad (5) \end{aligned}$$

Where t_{comp} is the time for computing one floating point result, and t_{comm} is the time for one communication (sending or receiving one floating point data element). This communication complexity does not take into account the startup cost for sending and receiving each message. Although for very small problems this startup cost can be significant, for production size problems it should be insignificant when compared to the total communications cost.

If the number of frequencies is much less than the number of points in the interferogram, and assuming $P \ll N_{pts}$, the parallel time complexity should reduce to the computa-

N_{pts} = (program parameter)
N_{fr} = (program parameter)
P = (number of processors)

Proc 0: read $\hat{v}_i(N_{fr})$
Proc 0: read $RawInterf_j(N_{pts})$
// Communications
Distribute array sections N_{pts}/P
Copy arrays of N_{fr}

// Local operations on each proc.
$CosSpectrum_i(N_{fr}) = 1.0$
$d_j(N_{pts}/P) = (j_P - 0.5 * N_{pts})\frac{CL}{FDR}$
$cbasis_{i,j_P}(N_{fr}, N_{pts}/P) = 0.25 * (1 + \cos(2\pi * \hat{v}_i * d_{j_P}))$

// Communications
$CosSum_i(N_{fr}) = \sum_j^{N_{pts}} cbasis_{i,j_P}$

For each iteration
BEGIN
  // Local operations
  $OldCosSpectrum_i(N_{fr}) = CosSpectrum_i$
  For all local points j
  BEGIN
    $CosInterf_j(N_{pts}) = \sum_i^{N_{fr}} cbasis_{i,j} * OldCosSpectrum_i$
  END

  For all frequencies i (Local and Communications)
  BEGIN
    $CosProduct_i(N_{fr}) = \sum_j^{N_{pts}} cbasis_{i,j} \frac{RawInterf_j}{CosInterf_j}$
    // Local operation
    $CosSpectrum_i(N_{fr}) = \frac{OldCosSpectrum_i}{CosSum_i} CosProduct_i$
  END
END
// Communication
$CosInterf(Npts, P_0) = Gather(CosInterf_j(N_{pts}/P))$
// Local prints to Processor 0
print $CosSpectrum_i(N_{fr})$
print $CosInterf(N_{pts})$

Figure 2. Pseudo-code for the parallel maximum-likelihood inversion algorithm.

tion cost only, $\mathcal{O}(\frac{N_{iter}N_{fr}N_{pts}}{P})$. At some point, as we decrease the size of the problem on each processor, either by decreasing N_{pts} or increasing P, the communications cost relative to the computation will increase and the program will no longer scale. As we show in the next section, for the problem sizes we anticipate and the number of processors we have available, this has not yet become a problem.

4 Experimental Results

The parallel maximum-likelihood algorithm was executed on three different machines: an Intel-based PC cluster, an SGI Onyx, and an SGI Origin 2000. The PC cluster consisted of 16 333 MHz Intel Pentium II processors running Linux (kernel version 2.0.35). Communication between the PC nodes was over a 100 MB/s Ethernet. Each node had a 256 MB RAM and a 4 GB local disk. The SGI Origin 2000 consisted of 24 250Mhz MIPS R10000 CPUs, with 24 GB of memory and 96 GB of disk space total. The SGI Onyx consisted of 12 194-MHz MIPS R10000 CPUs, 3 GB of memory and 18 GB of scratch disk space. Both SGI systems were running the IRIX 6.5 operating system.

The parallel code was executed on different number of processors on the three different machines. Following are graphs of run times and speedup, where each plotted point is the median of eleven different timing runs on each of the three different machines.

Figures 3 and 4 show the median time and speedup for the PC cluster. Figures 5 and 6 show the median time and the speedup for the SGI Origin 2000. The SGI Onyx median time and speedup are shown in figures 7 and 8. For all tests, the interferogram consisted of 4096 points and computations were performed for 66 frequencies. In each case speedup is nearly linear. The speedup was computed by dividing the speed evaluated for one processor by the speed evaluated by multi processors. The SGIs exhibited super linear speedup relative to the 1 and 2 processor performance. We attribute this to the more effective use of the SGI's cache memories once the per-processor problem size has been reduced sufficiently.

5 Discussion and Future Work

We have implemented the maximum-likelihood algorithm on parallel processors and have shown how it can benefit from parallel processing. In addition, we have shown that the parallel implementation is not only scalable and portable, but has also brought the execution time of the maximum-likelihood computations to feasible levels using contemporary and cost-efficient high-performance computers including machines such as the SGI Origin 2000, SGI Onyx, and PC clusters. The implementation was studied on interferograms with sizes up to 4096.

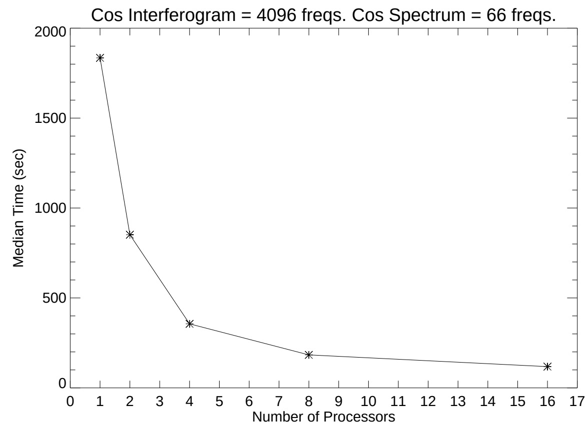

Figure 3. Median Timing for varying numbers of processors on an Intel-based PC cluster.

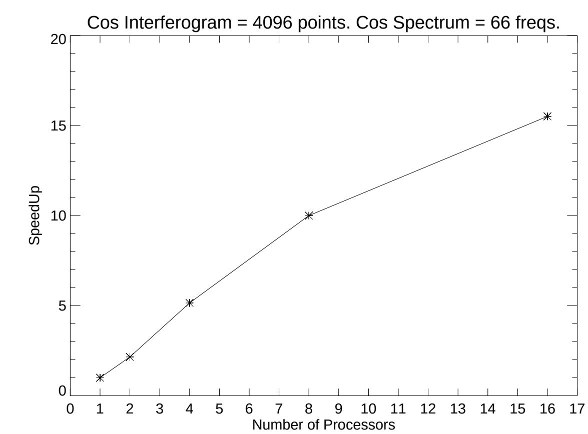

Figure 4. Speedup for varying numbers of processors on an Intel-based PC cluster.

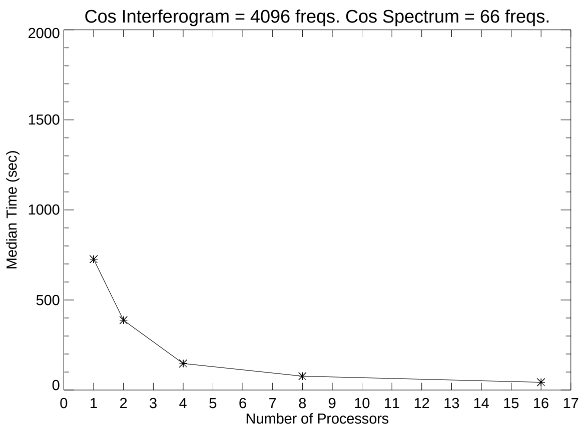

Figure 5. Median Timing for varying numbers of processors on an SGI Origin 2000.

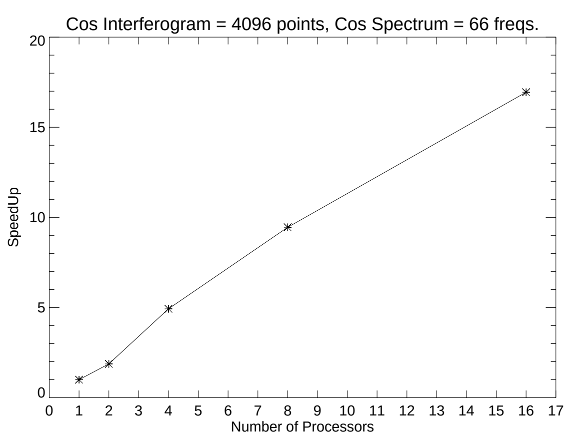

Figure 6. Speedup for varying numbers of processors on an SGI Origin 2000.

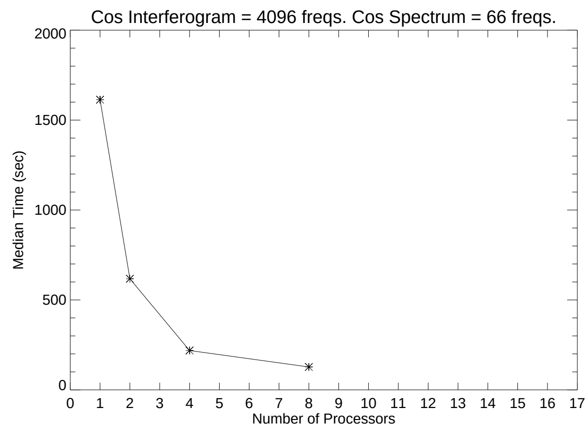

Figure 7. Median Timing for varying numbers of processors on an SGI Onyx.

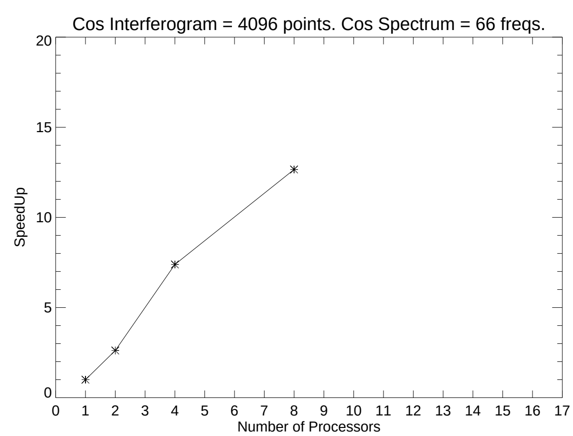

Figure 8. Speedup for varying numbers of processors on an SGI Onyx.

Future work will study the properties of the problem on additional test cases with the goal of improving the overall performance, with a focus on increasing the rate of convergence. Initially we intend to pursue two possible approaches. First, as an alternative to the expectation-maximization algorithm for calculating the maximum-likelihood as described here, we will investigate the use of an Iteratively Reweighted Least Squares (IRLS) algorithm [8]. The maximum-likelihood technique described here is effectively non-parametric in that it estimates parameters for millions of peaks. In this application, it is known that only around 10,000 peaks are possible. So, in the second approach we propose to take advantage of knowledge of the chemistry and physics of the sample being measured to derive a parametric Poisson model.

Since beginning this work we have become aware of several similar projects that have also parallelized the EM algorithm, such as Gyulai et al. [6], Jones and Mitra [9], and Bastiaens and Lemahieu [1], with the later 2 developing parallel algorithms for the imaging of positron emission tomography (PET) data. Although this algorithm has been shown to always converge [12], in most of the reports on implementations, the speed of convergence has been cited as an issue.

6 Disclaimer

Certain commercial equipment and software may be identified in order to adequately specify or describe the subject matter of this work. In no case does such identification imply recommendation or endorsement by the National Institute of Standards Technology, nor does it imply that the equipment or software is necessarily the best available for the purpose.

References

[1] K. Bastiaens and I. Lemahieu. Parallel implementation of the maximum likelihood-expectation maximization (ML-EM) reconstruction algorithm for positron emission tomography (PET) images in a visual language. In *Proc. Hybrid Image and Signal Processing IV*, pages 176–183. SPIE, Bellingham, WA, April 1994.

[2] S. E. Bialkowski. Overcoming the multiplex disadvantage by using maximum-likelihood inversion. *Society for Applied Spectroscopy*, 52:591–598, 1998.

[3] J. Devaney and J. Hagedorn. Internal paper. May 7, 1996.

[4] R. A. Fisher. On the mathematical foundations of theoretical statistics. In *Contributions to Mathematical Statistics*. J. Wiley & Sons, New York, 1950.

[5] W. George. Dynamic load balancing for data-parallel MPI programs. In *Message Passing Interface Developer's and User's Conference*, pages 95–100, March 1999. URL:`http://www.itl.nist.gov/div895/savg/papers/LoadBalmpidc99.ps`.

[6] C. Gyulai, S. Bialkowski, G. S. Stiles, and L. S. Powers. A comparison of three multi-platform message-passing interfaces on an expectation maximization algorithm. In *Proc. 1993 World Transputing Conference*, IOS Press, Amsterdam, Sept 1993.

[7] J. Hagedorn and A. Heckert. Dparlib user's guide. Draft of NIST software library documentation, March 18, 1997.

[8] R. M. Heiberger and R. Becker. Design of an S function for robust regression using iteratively reweighted least squares. *Journal of Computation and Graphical Statistics*, 1(3), 1992.

[9] H. M. Jones and G. Mitra. A parallel implementation of the expectation-maximization (EM) algorithm in positron emission tomography (PET). In *Parallel Optimization Colloquium - POC96*, pages 25–27, Versailles, France, March 1996.

[10] K. Lange and R. Carson. EM reconstruction algorithms for emission and transmission tomography. *J. Comput. Assist. Tomog.*, 8:306–316, 1984.

[11] L. A. Shepp and Y. Vardi. Maximum likelihood reconstruction for emission tomography. *IEEE Trans. Med. Imaging*, MI-1:113–121, 1982.

[12] Y. Vardi, L. A. Shepp, and L. Kaufman. A statistical model for positron emission tomography. *J. Amer. Statis. Soc.*, 80(389):8–20, 1982.