Parallel ICA Methods for EEG Neuroimaging

Dan B. Keith, Christian C. Hoge, Robert M. Frank, and Allen D. Malony

Neuroinformatics Center
University of Oregon
Eugene, Oregon, USA
{dkeith, rmfrank, hoge, malony}@cs.uoregon.edu

Abstract

*HiPerSAT, a C++ library and tools, processes EEG data sets with ICA (Independent Component Analysis) methods. HiPerSAT uses **BLAS**, **LAPACK**, **MPI** and **OpenMP** to achieve a high performance solution that exploits parallel hardware. ICA is a class of methods for analyzing a large set of data samples and extracting independent components that explain the observed data. ICA is used in EEG research for data cleaning and separation of spatiotemporal patterns that may reflect different underlying neural processes. We present two ICA implementations (FastICA and Infomax) that exploit parallelism to provide an EEG component decomposition solution of higher performance and data capacity than current MATLAB-based implementations. Experimental results and the methodology used to obtain them are presented. Integrating HiPerSAT with **EEGLAB** [4] is described, as well as future plans for this research.*

1. Introduction

EEG (Electroencephalography)[1] measures electrical potentials on the scalp surface that occur as a result of dynamic brain function. The procedure involves placing multiple *sensors* on the scalp. These sensors have the ability to measure potential changes with microvolt sensitivity. Analog-to-digital conversion hardware and software discretizes the time-varying potential changes on the scalp into a digital form.

It is well-established [7] that electrochemical events within the brain can manifest as surface potential changes on the scalp. There are both clinical and research procedures which use this EEG data to infer physiological phenomenon, trauma and mental states. For example, EEG is used in the diagnosis and treatment of epilepsy, as well as to understand the cognitive basis of language.

Dense-array EEG ([18]) is a technique of using a fine-grained spherical mesh of sensors on the scalp, face and neck in order to provide greater resolution of brain dynamics. Dense-array EEG uses anywhere from 64 to 256 sensors, each of which is sampled many times per second (250hz is typical) producing a time-based stream of data that describes the voltage at each sensor. 15 minutes of data from a 128-channel sensor mesh will be more than 100Mb. Advances in EEG will increase both sensor density and sampling rate, resulting in even larger data sizes.

One challenge of using scalp-based data is that each sensor is actually measuring surface potential changes caused by a *superposition* of underlying signals from various sources within the brain, as well as extra-brain sources. These signals are transmitted to the scalp via *volume conduction* through various tissue and bone structures. Each sensor is actually receiving a mixture of different signals, and a given signal (e.g., from a firing neuron bundle) may be received by several or all of the sensors, and at different intensities. The signal-to-noise ratio is potentially low; sources of noise include electrical equipment and physiological phenomenon such as eyeblinks, and heartbeats.

The signal mixtures at each sensor can be separated into several independent components. Some of these will correspond to *artifacts* such as eyeblinks, heartbeats and in some cases, the experimental apparatus. Independent Component Analysis (ICA) is a mathematical technique for extracting components, where the extracted components describe temporally independent activities from spatially fixed overlapping

[1] Most of the techniques described here for EEG apply equally well to the sibling technique of MEG (Magnetoencephalography), which measures magnetic rather than electrical fields.

1-4244-0054-6/06/$20.00

sources. Sources corresponding to artifacts can then be removed from the signal mixtures to facilitate further analysis of the EEG data.

By combining the high-resolution data provided by dense-array EEG with sophisticated data analysis techniques such as ICA, researchers are developing ways to 'see' into the neurophysiological and cognitive processes within the brain. Research in the EEG/MEG community is constrained by sequential execution of algorithms within computational frameworks (e.g., MATLAB) with memory limitations and other overheads.

Our work overcomes these limitations by providing *HiPerSAT* (High Performance Signal Analysis Toolkit). *HiPerSAT* is a C++ library that facilitates the separation of EEG data via both the *FastICA* and *Infomax* techniques. In addition, we have a tool, `hipersat`, which allows easy access to this functionality from a command line. Finally, we have integrated HiPerSAT into the EEGLAB [5] framework, a MATLAB-based application that is used by neuroscientists to analyze and visualize EEG data.

HiPerSAT's implementation of *Infomax* and *FastICA* provides a significant advantage over the existing MATLAB-based algorithms in two ways: execution time and data size. The reduction in wall-clock time provided by HiPerSAT was one of the primary motivators for this research. However, an equally significant feature of HiPerSAT is its ability to efficiently handle much larger data sets than MATLAB.

2. Background and Mathematical Foundations

We can formally describe the data output of an EEG device with n sensors as a time-ordered series of vectors $\mathbf{x}_t$, each of which has length n, where $x_t[j]$ is the voltage at sensor j at time t. This is best viewed as a rectangular data matrix, where each row (also known as a *channel*) represents a particular sensor's voltage over time, and each column corresponds to a particular time point. This data matrix is the primary input for ICA methods.

ICA assumes that the potentials measured in $\mathbf{x}$ are mixtures of one or more underlying fundamental signal components. The goal of ICA as applied to EEG data is to explain the observed matrix $\mathbf{x}$ of signal mixtures in terms of two other quantities:

- $\mathbf{s}$ - A matrix of time-ordered values corresponding to posited independent *source signal* components
- $\mathbf{A}$ - A *mixing* matrix that accounts for how the independent signal components in $\mathbf{s}$ are mixed into the observed scalp *signal mixtures* $\mathbf{x}$.

Formally, we wish to solve the Equation 1 for both $\mathbf{A}$ and $\mathbf{s}$, given that we have a measured set of mixtures $\mathbf{x}$:

$$\mathbf{x} = \mathbf{A}\mathbf{s} \tag{1}$$

In order to derive $\mathbf{A}$, it is sufficient to derive $\mathbf{W}$, also known as the *unmixing matrix*, such that:

$$\mathbf{A} = \mathbf{W}^{-1} \tag{2}$$

$$\mathbf{s} = \mathbf{W}\mathbf{x} \tag{3}$$

Most ICA algorithms are based upon finding the *unmixing* matrix $\mathbf{W}$, as it is this matrix that allows the independent components $\mathbf{s}$ to be extracted from the set of signal mixtures $\mathbf{x}$.

ICA makes a few assumptions that restrict the set of solutions to a small set of possible solutions, ideally with a single *most-likely* solution. ICA assumes that the input data are a mixture of temporally independent components whose sources are spatially fixed over time. This means that knowledge of $\mathbf{s}_t[i]$ for a given sample $\mathbf{s}_t$ provides no information about $\mathbf{s}_t[j]$.

Because a signal mixture is actually a sum of independent components, we would expect that the *pdf* (probability distribution function) of a mixture is more gaussian than that of a components that contribute to it (*Central Limit Theorem of Statistics*).

ICA uses these assumptions as constraints to determine $\mathbf{A}$ and $\mathbf{s}$, given a sufficiently large set of samples $\mathbf{x}$. Most ICA methods require that the number of samples exceeds several times the square of the number of channels. Detailed information on the mathematics underlying ICA is available in Hyvärinen [1] and Stone [16].

An illustration of independent component extraction is shown in Figure 1. The left figure shows a signal mixture of four independent components (in this case, distinct synthetic sinusoidal signals). The right figure shows the resulting independent components as extracted by FastICA.

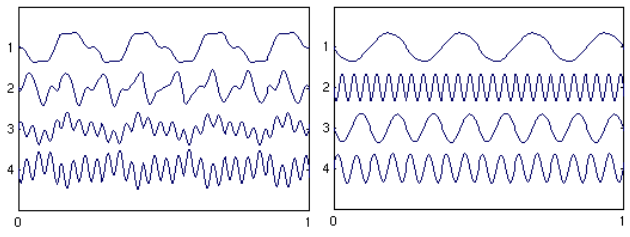

Figure 1. EEG Mixture and Components

2.1. Data Whitening

Both of the ICA algorithms that we have chosen to implement make the assumption that the input data $\mathbf{x}$ has been *whitened*. *Whitening* or *sphering* data is a process whereby the original mixture data $\mathbf{x}$ is multiplied by a matrix $\mathbf{S}_{ph}$ (the sphering matrix) to produce a set of *whitened* data that is uncorrelated with 0 mean and unity variance.

In Equation 2 above, the unmixing matrix $\mathbf{W}$ is applied to the mixture data $\mathbf{x}$ to generate the independent components. Because the ICA algorithms assume that $\mathbf{x}$ has been whitened, it is necessary to preprocess the data $\mathbf{x}$ by applying the sphering matrix $\mathbf{S}_{ph}$ to $\mathbf{x}$. This produces a set of whitened data $\mathbf{x}' = \mathbf{S}_{ph}\mathbf{x}$ which is amenable to the ICA algorithms (Equation 4).

$$\begin{aligned} \mathbf{s} &= \mathbf{W}\mathbf{x} && (4)\\ &= \mathbf{W}\mathbf{S}_{\mathbf{ph}}^{-\mathbf{1}}\mathbf{S}_{\mathbf{ph}}\mathbf{x} && (5)\\ &= \mathbf{W}_{\mathbf{gt}}\mathbf{x}' && (6) \end{aligned}$$

where $\mathbf{W}_{\mathbf{gt}}$ is known as the *weight matrix* and $\mathbf{x}'$ is the whitened data. Because $\mathbf{W}_{\mathbf{gt}}$ is an orthogonal matrix, it acts as an additional constraint on the constrained optimization problem that is the heart of ICA, thus speeding convergence to a solution.

2.2. FastICA

One form of ICA is encapsulated in the FastICA method, first described in Hyvärinen [12] and implemented in MATLAB as the function `fastica()` [11]. FastICA works by searching for a weight matrix that maximizes the non-gaussianity of the resulting components. Any mixture of non-gaussian random variables will be more gaussian than the variables themselves (*Central Limit Theorem of Statistics*). Therefore, it is possible to use non-gaussianity as a measure of statistical independence. FastICA uses this fact to build a weight matrix column-by-column, where each column maximizes the non-gaussianity of the corresponding component.

Non-gaussianity and therefore, independence, is maximized indirectly by computing and maximizing a *contrast function*. Different contrast functions can be used, although the use of *kurtosis* and *negentropy* have been shown to provide a good trade-off of speed and reliable convergence.

2.3. Infomax

One of the earliest ICA algorithms was described by Bell and Sejnowski [3] as an *information-maximization* or *infomax* algorithm. Infomax derives a weight matrix $\mathbf{W}_{\mathbf{gt}}$ that maximizes the statistical independence of the components by using an algorithm which minimizes the redundancy amongst outputs of a neural net. This is done by estimating the data's higher-order moments and ensuring that the resulting components are maximally independent and non-gaussian.

3. Related Work

Tucker [18] at the University of Oregon has pioneered and advocated the use of dense-array EEG measurements. Makeig at SCCN (*Swartz Center for Computational Neuroscience*) [7] has led the field in the use of ICA for discovering underlying components and for removing artifacts from such data. Makeig's group has developed the EEGLAB toolkit [5] that offers EEG analysis and visualization, including data analysis based on various ICA algorithms. In particular, the `runica()` function within EEGLAB is an improved implementation of the *infomax* algorithm as described by Bell and Sejnowski [3]. It is this MATLAB version of the algorithm that was used as a standard of correctness when implementing HiPerSAT's parallel C++ version.

ICA and the techniques known as *factor analysis* and *principal component analysis* (PCA) are both forms of *blind source separation*, which is the problem of determining the component sources in a set of mixtures, without having prior knowledge of how these sources are mixed. These algorithms solve essentially the same problem, but make different assumptions about the underlying data. ICA is different from PCA in that it minimizes the correlation of higher-order statistical moments. The blind source separation problem is also known as the *Cocktail Party Problem* [10] because of the way that a human brain can pick out the distinct conversations occurring simultaneously at a hypothetical cocktail party with multiple concurrent conversations.

4. Architecture of HiPerSAT

HiPerSAT is a C++ library framework that implements a variety of EEG analysis methods, including *FastICA* and *Infomax*. The design is based on a common set of utility code for I/O, data structures, and mathematical and statistical computations. In addition, it supports flexible parallelism models that can be reused depending on the analysis algorithm requirements.

Much of the matrix, vector, and linear algebra functionality needed by HiPerSAT is provided by two well-

known linear algebra libraries, BLAS (Basic Linear Algebra Subprograms) [6] and LAPACK (Linear Algebra PACKage) [2]. Modern platforms provide high-performance, C-accessible, threadsafe implementations of both LAPACK and BLAS.

HiPerSAT solves large problems using FastICA or Infomax running in sequential mode without requiring any parallel capabilities. However, the implementations achieve greater performance (see Section 10 below) when they are run in a parallel-capable environment. The current HiPerSAT implemention of FastICA relies upon MPI (running multiple processes) for its parallelism, whereas HiPerSAT's Infomax implementation uses OpenMP (single-process, shared memory, multiple processors).

There are several implementations of MPI available, including Intel's MPI, and the open-source MPICH [8] and LAM-MPI [9] [15]. The HiPerSAT performance results for FastICA we report here are based upon MPICH. OpenMP [14] is compiler-dependent and we make use of different OpenMP-compatible compilers in our work, including IBM's xlC and Intel's icc.

5. Overview of HiPerSAT Usage

The tasks of HiPerSAT are:

1. Obtain input data, in the form of a matrix $\mathbf{x}$.
2. Whiten data by computing sphering matrix $\mathbf{S}_{ph}$ such that $\mathbf{S}_{ph}\mathbf{x} = \mathbf{x}'$, where the rows in $\mathbf{x}'$ are uncorrelated with each other.
3. Search for a weight matrix $\mathbf{W}$ such that will unmix the whitened signal mixtures $\mathbf{x}'$ into a set of independent source signals $\mathbf{s}$.
4. Output the independent components and the computed weight, sphering, mixing matrices.

The HiPerSAT library is a set of C++ classes used to perform these steps. The `hipersat` program uses these classes in a utility accessible via a command line. Parameters to this command include the input and output files as well as various parameters such as algorithm (FastICA or Infomax), learning rate and convergence tolerance.

The `hipersat` program will initiate the sequential or parallel execution and then invoke the Whitening, FastICA, and Infomax methods in the correct order, based upon the command parameters.

6. Integration into EEGLab

EEGLAB is a widely-used neuroscience application providing visualization, filtering, and analysis of EEG data in a powerful, GUI-based environment [5]. EEGLAB can perform several ICA methods upon EEG data. By default, EEGLAB will execute a MATLAB-based version of Infomax called `runica()`.

We have extended EEGLAB's ICA calling interface to enable the convenient execution of HiPerSAT's Infomax and FastICA versions. If the user selects either `nic-fastica` or `nic-infomax` as the type of algorithm, then the EEG data will be exported to a disk file and the `hipersat` tool will be invoked. After `hipersat` completes, the resulting independent components and other output data are imported back into EEGLAB for subsequent display and analysis.

7. FastICA Execution

HiPerSAT implements two different algorithms, FastICA and Infomax, each using their own parallelism mechanism, MPI and OpenMP, respectively. The phases of execution are:

1. start processes on head and remote nodes
2. load parameters and replicate to workers
3. load data and distribute amongst workers
4. optionally whiten the data
5. compute weights using FastICA algorithm
6. output components, weights, sphering matrix

7.1. Process Creation

HiPerSAT's FastICA/MPI implementation assumes that there are one or more separate processes that run and communicate via MPI. The MPI execution model that we use in FastICA assumes that there is one machine or node (the *head node*) where the user will initiate execution of a FastICA run of HiPerSAT. The user initiates execution with the MPI-provided `mpiexec` command, which ensures that the `hipersat` command will be executed on the head node and on a set of specified additional *worker nodes*, separate machines where HiPerSAT is installed. After `mpiexec` has created processes on the head node and any worker nodes, execution of the `hipersat` main program will begin in each process.

7.2. Parameter Input and Distribution

Because only the head node is able to read the input parameter file (no shared filesystem is assumed), the relevant parameters must be distributed to all of the worker MPI processes so that the head node and workers have a common set of operating parameters. This is performed before the actual data file(s) are loaded

and processed. The head node uses the MPI function `MPI_BCast()` to transmit a copy of the parameter file to the workers. After this point, the head node and workers have a common set of operating parameters.

7.3. Data Input and Distribution

As above, the input EEG data file must be distributed to the workers from the head node. In the FastICA algorithm, if we have m samples in the input data set, and there are p total MPI processes, then each worker will get m/p samples from the original data file. If the number of samples is not evenly divisible by the number of MPI processes p, then the remainder of the samples will be given to a subset of the processes (i.e., some processes will have $m/p+1$ samples). The subset of samples allocated to a worker is called a *partition*.

This phase of FastICA processing ensures that the entire input file is loaded into an effectively *distributed memory*, which in this case spans multiple processes. No single process has the entire EEG data in its memory; instead, the set is partitioned amongst the head node and workers. Many of the subsequent stages in the algorithm are performed in parallel, with each `hipersat` process executing independently with periodic data exchange between the processes.

7.4. Data Whitening

The whitening of the input data is parallelized in the HiPerSAT FastICA algorithm. Here, each worker computes the covariance of its partition in parallel, and then combines the resulting matrix with a call to `MPI_Reduce`, which sums the various per-worker covariance matrices. The time for data whitening is neglible compared to the time required for either FastICA or Infomax; we do not report on the whitening time in this document.

7.5. FastICA

Each worker process will contain a whitened version of its partition. At this point, the FastICA algorithm properly begins with workers executing the method `searchForWeights()`. This method is responsible for computing a weight matrix $\mathbf{W}$ that properly separates the whitened mixtures into independent components. The method `searchForWeights()` is described in detail in a HiPerSAT technical report [13]; due to space considerations, we will summarize the performance complexity of the algorithm.

This method will compute $\mathbf{W_{gt}}$ one row at a time, with each row $\mathbf{w}$ corresponding to the weight vector that extracts one particular component from $\mathbf{x}'$. Let n be the number of input signal mixtures and the number of desired independent components; $\mathbf{W_{gt}}$ is therefore a $n \times n$ matrix and $\mathbf{w}$ is a vector of length n. The function `computeWeight()` performs this computation of a single component, and will be called $\mathbf{n}$ times.

`computeWeight()` performs a number of iterative calls to `improveWeight()`, which takes an existing weight vector and the entire data matrix and computes a better weight vector. This iterative process (based upon Newton's method) of weight vector improvement continues until a user-specifed number of iterations has passed, or a fixed-point has been met within some user-specified tolerance.

This means that the bulk of the performance cost is in the `improveWeight()` method. The mathematical basis for this method is derived in Hyvärinen [12]. Essentially, there is a fixed point convergence relation (Equations 7,8) that uses a gradient method to maximize the non-gaussianity of the product $\mathbf{w}'\mathbf{x}$, which is the projected independent component extracted by $\mathbf{w}'$.

$$\mathbf{w}' = E\{\mathbf{x}g(\mathbf{w}^\mathbf{T}\mathbf{x})\} - E\{g'(\mathbf{w}^\mathbf{T}\mathbf{x})\}\mathbf{w} \quad (7)$$

$$\mathbf{w}' = \mathbf{w}'/\|\mathbf{w}'\| \quad (8)$$

Because the expectation of these random variables is unknown *a priori*, the algorithm relies upon sampling the available data to get an estimate of the above expectations. This will require traversal of the data samples, and is one of the reasons why the algorithm takes longer as more samples are added. Because of the properties of whitened data and non-gaussian components, the generalized Equation 7 can be simplified into the actual formula that is used to compute new weights within the FastICA algorithm. This eliminates several matrix-matrix multiplies and matrix inversions, reducing some of these to rank-one updates (a much simpler matrix operation).

The function $g()$ is known as the *contrast* function and it and its derivative $g'()$ are used to evaluate the *gaussianity* of a signal component as generated by $\mathbf{w}'\mathbf{x}$. HiPerSAT supports three different contrast functions: *cubic*, *tanh*, and *gaussian*. The *cubic* contrast function is used to measure the *kurtosis* of the separated components, the *tanh* contrast function measures the *negentropy* of the components, and the *gaussian* function measures the *gaussianity* of the components.

The HiPerSAT implementation contains a version of `improveWeight()` for each of these contrast functions. Each version of `improveWeight()` shares the same basic loop structure, with subtle differences in how the contrast function and its derivative are used for each element in the loop.

7.6. Result Gathering and Output

After FastICA successfully converges on a weight matrix that satisfies the convergence criteria specified by the problem, HiPerSAT will export to disk the weight, sphering, mixing and unmixing matrices. HiPerSAT can optionally export the separated components as an EEG file. Because each worker has a separate partition of the input data, exporting the separated components is performed by the head node by gathering each worker's partition of separated component data and writing it to disk.

8. Infomax Execution

The Infomax implementation within HiPerSAT relies upon OpenMP for its parallelism. One of the potential advantages of an OpenMP approach is the ability for each separate thread to share the input EEG data, as well as to share data structures (e.g., the current weight matrix).

The phases of Infomax execution are:

1. start process
2. load parameters
3. load data
4. optionally whiten the data
5. create one or more worker threads
6. compute weights using Infomax algorithm
7. output components, weights, sphering matrix

8.1. Parameter and Data Input

The parallel workers of HiPerSAT/Infomax are implemented as threads within a single process, with each thread having access to the same shared data in memory, so there is no need to partition and distribute the parameters and data amongst the workers. Similarly, the exporting of the components does not require that the partitions are gathered prior to exporting. This makes reading in the parameters and input data a trivial operation.

8.2. Data Whitening

Data whitening in Infomax is identical to the standalone (non-parallel) version of data whitening used in FastICA. The whitening process is not yet parallelized in Infomax, but may be in future versions.

8.3. Infomax

In the Infomax algorithm, a single process contains the entire loaded and whitened EEG input data set. At this point, the Infomax algorithm properly begins with `searchForWeights()`, which is detailed in the HiPerSAT technical report [13].

After initializing the per-process data structures, `searchForWeights()` splits into several concurrent threads. These threads will iteratively call the method `TrainNN()`, which will adjust a provisional weight matrix to produce a better one. This iteration will continue until the stabilizes upon a solution within a given tolerance. It is this `TrainNN()` function which is measured when the Infomax iteration cost is reported in the graphs and tables below.

The `TrainNN()` function works by partitioning the input data into disjoint rectangular blocks consisting of the n channels for a small (approximately 35) number of samples. The order of the blocks is randomized to ensure that any temporal bias in the data is avoided.

As a block is processed, each OpenMP thread will operate on a fraction of the columns within a block. For example, if the block has 36 columns and there are 4 threads, then thread 1 gets columns $1, 5, 9, \ldots$, thread 2 might get columns $2, 6, 10, \ldots$ and so on. When processing a block $\mathbf{x}$, Infomax computes a new matrix $\mathbf{u}$ which is the same size as $\mathbf{x}$. This $\mathbf{u}$ matrix contains the projected components $\mathbf{Wx}$. The matrix multiplications within this parallel `for` loop will occur in parallel, affording a p-way speedup for this section of code if there are p processors available for OpenMP threads.

An *activation function* is applied to the block $\mathbf{u}$, resulting in a new matrix such that $\mathbf{y} \Leftarrow$ `activation(` $\mathbf{u}$ `)`. This activation function is used to to determine how the current weight matrix candidate $\mathbf{w}$ should be modified to make the projected components more independent. After all blocks have been processed, the weight matrix is updated and `TrainNN()` repeats the process until a suitable convergence is achieved.

8.4. Result Output

Writing out the separated components is currently implemented in the HiPerSAT version of Infomax is straightforward, since there is a single process containing the data matrix and the discovered weight matrix. The current implementation creates a temporary matrix to contain the sorted independent components. An improvement planned for HiPerSAT Infomax is to compute the separated components on the fly as they are output. This would reduce the memory requirements

during the output phase by 50% and would likely improve performance during this phase.

9. Experimental Methodology

We verified the validity and efficiency implementations of these algorithms with a series of tests. The validity tests ensured that we gave the correct answers, as defined by the EEGLAB implementations of these algorithms. We have validated that the results are virtually identical for a variety of both real and synthetic data sets. We have compared the results of the EEGLAB versions of Infomax and FastICA with the corresponding HiPerSAT results and they match.

To evaluate efficiency, we performed timed tests on a variety of hardware/software configurations, with a variety of data set sizes. The tested platforms are listed in Table 1.

Table 1. Tested HiPerSAT Platforms

Name	Processor(s)	Parallelism
Mac G4	PowerPC G4	sequential only
Neuronic	Xeon (2xCPU)	16xMPI/4xOpenMP
P655 (IBM)	8x1.5Ghz POWER4	8xOpenMP/8xMPI
P690 (IBM)	16x1.3Ghz POWER4	16xOpenMP/16xMPI

The primary data sets used were from a psychology study in which subjects were given various stimuli and asked to perform a reading task. The role of ICA in this study is to isolate and remove artifacts such as eyeblinks.

The basic metric of the input data size is its dimension, in terms of number of channels and number of samples. The expectation is that the problem complexity is proportional to the square of the number of channels and linearly proportional to the number of samples. The number of channels used in these experiments varied from 64 to 256, whereas the number of samples ranged from 100,000 to 10,000,000. As a point of reference, a complete ICA decomposition using the standard MATLAB implementations ranges in wall clock time from 1 hour to several days depending upon the dimension of the data set and the execution platform.

The two ICA algorithms (FastICA and Infomax) are both based upon an iterative method that achieves convergence. The number of iterations varies depending upon the content of the data, not its dimension. Therefore, different data sets of the same size could take vastly different amounts of time to converge to a solution. This makes the use of overall wall clock time a less useful metric for comparing performance between different algorithms and data sets.

In order to address the above deficiency, we use a per-iteration time cost metric that is invariant between different data of a given size. This per-iteration cost, C_{iter}, is used in the experimental results described here. This permits more experiments to be run, because the program is not run until convergence, allowing us to explore a broader evaluation space.

For FastICA, the iteration time measured corresponds to `improveWeight()` described in Section 7.5. The iteration time measured for Infomax corresponds to the time spent in `trainNN()` described in Section 8.3. In both cases, C_{iter} is computed as $C_{iter} = totalTime_{iter}/numberOfIters$, where $totalTime_{iter}$ is the measured time spent in the iterated functions.

In both FastICA and Infomax, we used the parallel performance analysis software TAU [17]. The TAU software allowed us to instrument the HiPerSAT source code to facilitate the gathering of fine-grained performance information for both MPI and OpenMP.

10. Experimental Results

Testing HiPerSAT proved to be challenging because of the potential for an overwhelming set of combinations of data file size, hardware (see Table 1), algorithm (Infomax vs. FastICA), parallel processing type (OpenMP vs. MPI vs. standalone), and processing parameters (learning rate, convergence threshold, etc). For the purposes of this paper, we will focus on those results most salient to the issue of performance gains possible via parallelism. We have restricted our analysis here to a standard number of samples (111,000 samples) in order to eliminate variability due to the number of samples.

Because the FastICA/MPI and Infomax/OpenMP algorithms use different parallelization approaches, we treat these sets of measurements as incommensurate and describe each separately. In the graphs that describe *Speedup*, the speedup associated with n processors is the ratio of the cost-per-iteration for n processors to the cost-per-iteration for 1 processor. Table 2 summarizes the results displayed in these graphs. The speedup shown is maximum speedup obtained on $69 \times 111{,}000$ data.

The performance graphs demonstrate the speedup of the per-iteration cost for a given dataset when different numbers of processors are applied to the problem. Each curve corresponds to a different number of channels (the number of samples is fixed at 111,000 samples). The speedup for an n-processor computation is computed by dividing the per-iteration cost by the per-iteration cost for a single processor. The diagonal line represents linear speedup.

Table 2. Sequential Cost and Speedup

Hardware/Algorithm	C_{iter} (sec)	Speedup
G4 MATLAB FastICA	1.21	n/a
G4 FastICA	0.3089	n/a
Neuronic MATLAB FastICA	0.1974	n/a
Neuronic FastICA	0.1319	14.6296
P655 FastICA	0.5449	7.9768
P690 FastICA	0.6286	15.9566
G4 MATLAB Infomax	11.21	n/a
G4 Infomax	4.36	n/a
Neuronic MATLAB Infomax	5.22	n/a
Neuronic Infomax	1.0892	1
P655 Infomax	3.4366	3.26
P690 Infomax	3.9838	3.2519

11. Expected vs. Observed Results

The FastICA algorithm equally partitions the samples among the available processors. An examination of the FastICA algorithm reveals only two significant points where synchronization and data exchange will occur between the various worker and head processes. Once for every desired component, where the initial weight vector for that component is exchanged; and once for every iteration of weight improvement, where the workers sum their weight estimates. All of the other work is executed in parallel by the workers, operating upon their own partition of the data. The ratio of computation to communication time is high for FastICA, resulting in a high level of parallel performance. The FastICA performance graphs show near-linear speedup for the different platforms.

In contrast, the performance of the parallel Infomax algorithm is more constrained. Our analysis of the parallel Infomax algorithm shows several points where parallelism is occurring, during neural network training and the first part of updating the weights. We should see performance scaling for these sections. Unfortunately, the performance scaling curves indicate much poorer performance than FastICA. One reason is that the amount of parallel work available in OpenMP parallel regions is not large in relation to the sequential computation.

One possible reason is that the algorithm suffers caching effects due to the composition of blocks from random selection of samples. These blocks are then separately processed by OpenMP threads, but reference memory locations across the dataset. We see in the performance curves that 64-channel speedup improves better generally than 128- and 256-channel experiments, especially on the IBM p655. This is unexpected, but after analysis is explained by the fact that more of the data set can be contained in the Level 3

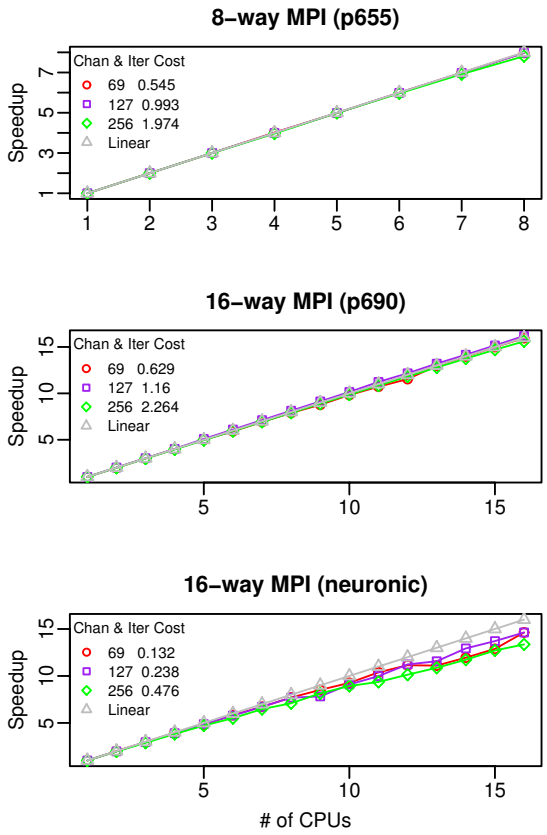

Figure 2. FastICA: Iteration cost vs # of procs

cache memory. For 64 channels and 110,000 samples, we suspect that the entire dataset fits in the 32 Mbyte Level 3 cache on the IBM p655 machine. As the number of samples increase, we should see better speedups for larger numbers of channels.

12. Future Directions for HiPerSAT

The HiPerSAT library and tools currently provide ICA decomposition via the FastICA and Infomax algorithms. We plan to enhance HiPerSAT by increasing the performance of the existing algorithms, as well as by implementing additional ICA methods that have shown promise in EEG analysis. In addition, we are building grid-based interfaces to permit the remote initiation and monitoring of HiPerSAT tasks.

12.1. Performance

Both FastICA and Infomax are very processor-intensive, especially with large data sets. Our early results are promising and show the potential of exploiting

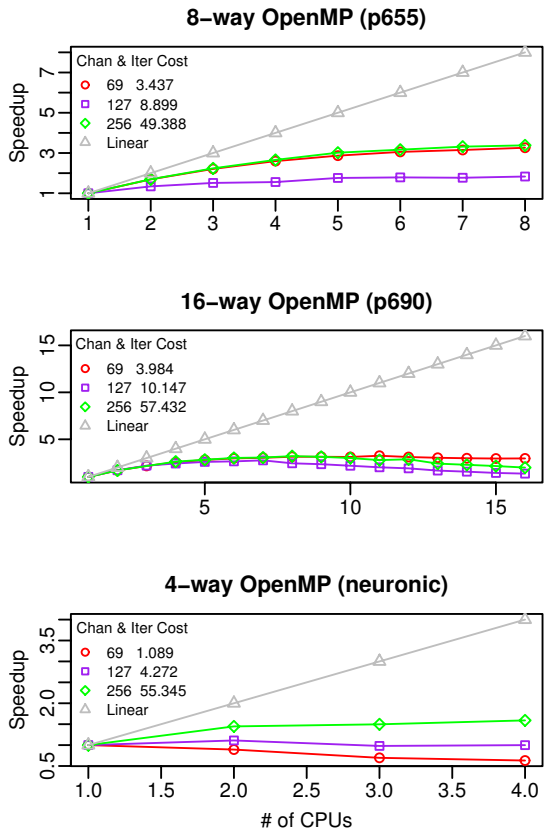

Figure 3. Infomax: Iteration cost vs # of procs

parallelism for these tasks. However, there are several potential performance enhancements worth investigating.

Although FastICA scales linearly with the number of processors, we can further improve overall performance by reading and writing the EEG data in parallel. This can be done directly during the FastICA execution by using MPI's asynchronous sends and receives to parallelize data distribution.

Clearly, the Infomax algorithm has room for performance improvement. Our scaling results show a best-case speedup of 3 running on 8 processors, and poorer performance overall. Although OpenMP performance is highly dependent on the platform and compiler, we need to further investigate ways to increase the portion of the algorithm that can operate in parallel. This includes minor changes such as adjusting the block size used during the training of a weight vector, and major changes such as allowing each worker thread to work on its own block in parallel, merging the learned weights after each step. The mathematical legitimacy of these optimizations must be analyzed.

We have come up with a more memory-efficient way to sort and export the independent components from Infomax in a windowed fashion which does not require duplication of the data before output.

Finally, we want to consider alternative parallelism approaches, including the design of an MPI-based Infomax, an OpenMP-based FastICA, and a hybrid MPI/OpenMP version of both algorithms, assuming these versions offer opportunities for improved performance.

12.2. New ICA Features and Algorithms

The versions of FastICA and Infomax currently implemented in HiPerSAT lack some features that are present in the EEGLAB-based versions of these algorithms (`runica` and `fastica`). We plan on implementing these optional features within HiPerSAT. They include:

- Performing PCA (Principal Component Analysis) reduction during preprocessing
- Random restarts to address a lack of convergence in FastICA.
- *symmetric-mode* FastICA instead of *deflationary-mode*
- *extended-mode* Infomax to detect subgaussian components.

There are a variety of other ICA algorithms that have been implemented in MATLAB. These algorithms differ in their assumptions about the data, the way they measure independence, and whether they account for temporal ordering of the data. They include SOBI (Second-Order Blind Identification), JADE (Joint Approximate Diagonalization of Eigen matrices), and ERICA (Equivariant Robust ICA). We will be examining these algorithms to determine their utility in the neuroimaging domain and their suitability for high-performance implementation.

12.3. Grid Execution

HiPerSAT's highest performance requires that it executes on on a parallel system. However, a researcher may use EEGLAB on their laptop or workstation to do much of their interactive analysis. We have extended the EEGLAB integration to allow remote execution of HiPerSAT tasks on computational servers. We are also building a grid-enabled interface to make HiPerSAT accessible as a grid service. One important advantage that will come from this will be the ability to use HiPerSAT on multiple EEG data sets concurrently.

13. Conclusions

We described the ICA class of techniques and parallel implementations of two ICA algorithms, FastICA and Infomax. We showed that for the large data sets typically found in dense-array EEG research, the performance improvement provided by HiPerSAT was significant compared to the sequential implementations in EEGLAB. The speedup for FastICA was impressive while only modest for Infomax. The ability to handle dataset sizes larger than the MATLAB-based versions is very important for the EEG neuroimaging community at large.

The FastICA algorithm was shown to scale quite nicely, with near-linear speedup as additional processors are added. However, the Infomax algorithm scaled sublinearly. This lack of parallelism and speedup in HiPerSAT Infomax is a consequence of the current implementation and warrants further research.

Although FastICA is faster and parallelizes better than Infomax, it has been shown that it can fail to converge on a solution. This appears to be especially problematic for larger numbers of channels (e.g., more than 127). Restarting the algorithm with new random weight vectors can reduce, but not eliminate, this convergence problem.

The choice of Infomax versus FastICA is also one of perceived correctness and quality of results. Neuroscientists will select among the various decomposition algorithms for reasons other than performance. Indeed, one active area of neuroscience research is the comparison of different ICA algorithms as applied to EEG. Different algorithms are appropriate for different assumptions about the EEG data and the underlying signals.

We are collaborating with neuroscience researchers at the University of Pittsburgh in the application of HiPerSAT in cognitive language processing studies focusing on word learning and reading assessment. The EEGLAB-based ICA implementation was insufficient for this work because of the large datasets involved. HiPerSAT has also been successfully used for artifact detection and cleaning in this and other neuroscience work. In general, we see the HiPerSAT library and tools as a complement to MATLAB-based ICA implementations, providing the EEG/MEG community with a higher performing, parallel solution.

14. Acknowledgment

The authors would like to thank Virat Agarwal, who provided valuable assistance in the development of HiPerSAT and the measurement of its performance.

This research was partially funded by a grant from the National Science Foundation, Major Research Instrumentation program, and a contract from the Department of Defense, Telemedicine Advanced Technology Research Center, for the University of Oregon's Neuroinformatics Center (Brain, Biology, and Machine Initiative).

References

[1] J. K. Aapo Hyvärinen and E. Oja. *Independent Component Analysis*. Wiley, 2001.

[2] E. e. a. Anderson. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[3] A. J. Bell and T. J. Sejnowski. An information-maximisation approach to blind separation and blind deconvolution. *Neural Computation*, pages 1129–1159, 1995.

[4] A. Delorme and S. Makeig. EEGLAB: an open source toolbox for analysis of single-trial EEG. *Journal of Neuroscience Methods*, 134:9–21, 2004.

[5] EEGLAB home page, 2005.

[6] L. S. B. et al. An updated set of basic linear algebra subprograms (blas), 2002.

[7] S. M. et al. Mining event-related brain dynamics. *Trends in Cognitive Science*, pages 204–210, 2004.

[8] W. G. et al. A high-performance, portable implementation of the mpi message passing interface standard.

[9] R. D. Greg Burns and J. Vaigl. LAM: An Open Cluster Environment for MPI. In *Proceedings of Supercomputing Symposium*, pages 379–386, 1994.

[10] Cocktail party demo, 2005.

[11] HUT - CIS: The FastICA package for MATLAB, 2005.

[12] A. Hyvärinen. Fast and robust fixed-point algorithms for ica. *IEEE Transactions on Neural Networks*, pages 626–634, 1999.

[13] D. B. Keith, C. C. Hoge, R. M. Frank, and A. D. Malony. NIC Technical Report - HiPerSAT. Technical report, Neuroinformatics Center, University of Oregon, 2005.

[14] Openmp application program interface.

[15] J. M. Squyres and A. Lumsdaine. A Component Architecture for LAM/MPI. In *Proceedings, 10th European PVM/MPI Users' Group Meeting*, pages 379–387, 2003.

[16] J. V. Stone. *Independent Component Analysis : A Tutorial Introduction*. MIT Press, 2004.

[17] TAU home page, 2005.

[18] D. Tucker. Spatial sampling of head electrical fields: the geodesic sensor net. *Electroencephalography and Clinical Neurophysiology*, pages 145–163, 1993.