

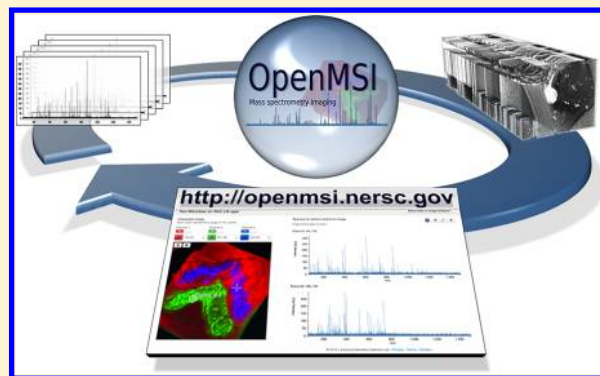
OpenMSI: A High-Performance Web-Based Platform for Mass Spectrometry Imaging

Oliver Rübél,* Annette Greiner, Shreyas Cholia, Katherine Louie, E. Wes Bethel, Trent R. Northen, and Benjamin P. Bowen*

Lawrence Berkeley National Laboratory, One Cyclotron Road, Berkeley, California, 94720, United States

Supporting Information

ABSTRACT: Mass spectrometry imaging (MSI) enables researchers to directly probe endogenous molecules directly within the architecture of the biological matrix. Unfortunately, efficient access, management, and analysis of the data generated by MSI approaches remain major challenges to this rapidly developing field. Despite the availability of numerous dedicated file formats and software packages, it is a widely held viewpoint that the biggest challenge is simply opening, sharing, and analyzing a file without loss of information. Here we present OpenMSI, a software framework and platform that addresses these challenges via an advanced, high-performance, extensible file format and Web API for remote data access (<http://openmsi.nersc.gov>). The OpenMSI file format supports storage of raw MSI data, metadata, and derived analyses in a single, self-describing format based on HDF5 and is supported by a large range of analysis software (e.g., Matlab and R) and programming languages (e.g., C++, Fortran, and Python). Careful optimization of the storage layout of MSI data sets using chunking, compression, and data replication accelerates common, selective data access operations while minimizing data storage requirements and are critical enablers of rapid data I/O. The OpenMSI file format has shown to provide >2000-fold improvement for image access operations, enabling spectrum and image retrieval in less than 0.3 s across the Internet even for 50 GB MSI data sets. To make remote high-performance compute resources accessible for analysis and to facilitate data sharing and collaboration, we describe an easy-to-use yet powerful Web API, enabling fast and convenient access to MSI data, metadata, and derived analysis results stored remotely to facilitate high-performance data analysis and enable implementation of Web based data sharing, visualization, and analysis.



Mass spectrometry imaging (MSI) as an analytical technique is rapidly finding widespread application in life sciences,^{1–3} bioengineering, medicine,⁴ drug development,^{5,6} and studies of metabolic processes and promises to enable transformative medical diagnostics and large-scale scientific experiments.^{7,8} In recent years, sample preparation methodologies, desorption-ionization techniques, and MSI instrumentation have advanced to a point where standard practices can be followed to yield high-quality data, enabling direct interrogation of the spatial distribution of metabolites and proteins within cells and tissues.

In MSI, many spatially defined mass spectra are acquired across a sample. In the raw form, the data for each position is represented as a profile of intensity values over a corresponding range of mass-to-charge (m/z) values. Modern mass spectrometers are capable of accurately measuring the m/z to approximately the mass of a single electron, generating massive and highly complex data sets.⁹

Despite numerous advances in analysis of MSI data sets, widespread adoption of MSI is hindered by a lack of fast and easy-to-use approaches for sharing, management, access, and provenance of raw MSI data and derived analyses.^{10–18} While

numerous open standards have been proposed for storage of MSI data, e.g. imzML or mzML,^{19,20} none of the current formats efficiently support standard data access patterns, such as reading of ion images, and they often introduce large storage overheads. This lack in performance already at the file-format level unnecessarily hinders visual data exploration and high-performance, complex data analysis. These formats came about due to the urgent need for a standardized way to store data and have been adopted by many laboratories as their preferred file format. However, as Web-based technologies and high-performance, parallel data analysis and computing become mainstream in today's laboratories, it is essential that beyond standardized data storage that data formats support efficient parallel I/O for fast read and write, compact data storage, and storage and management of metadata and data provenance information to facilitate complex analysis workflows.

Storage and management of MSI data is challenging; the data is extremely large, shows large (3–4 orders of magnitude)

Received: July 25, 2013

Accepted: October 2, 2013

Published: October 2, 2013

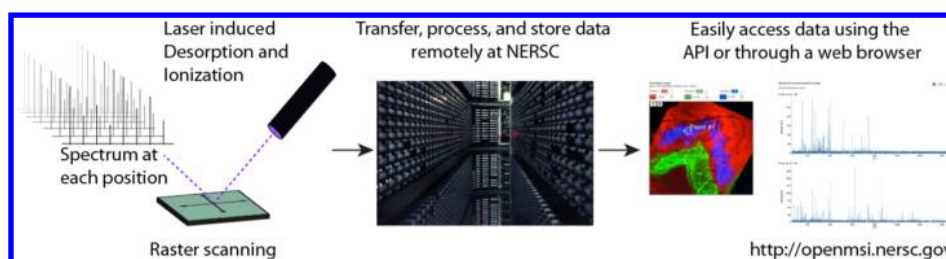


Figure 1. Illustration showing the main processing paradigm of OpenMSI. MSI data is acquired at the lab and transferred to NERSC for processing and storage. This enables us to take advantage of large-scale, high-performance compute resources to perform more complex analyses than possible using limited local compute capabilities. Raw MSI data and derived analyses results are then visualized and analyzed via the Web using the interactive OpenMSI online viewer providing fast feedback to the user.

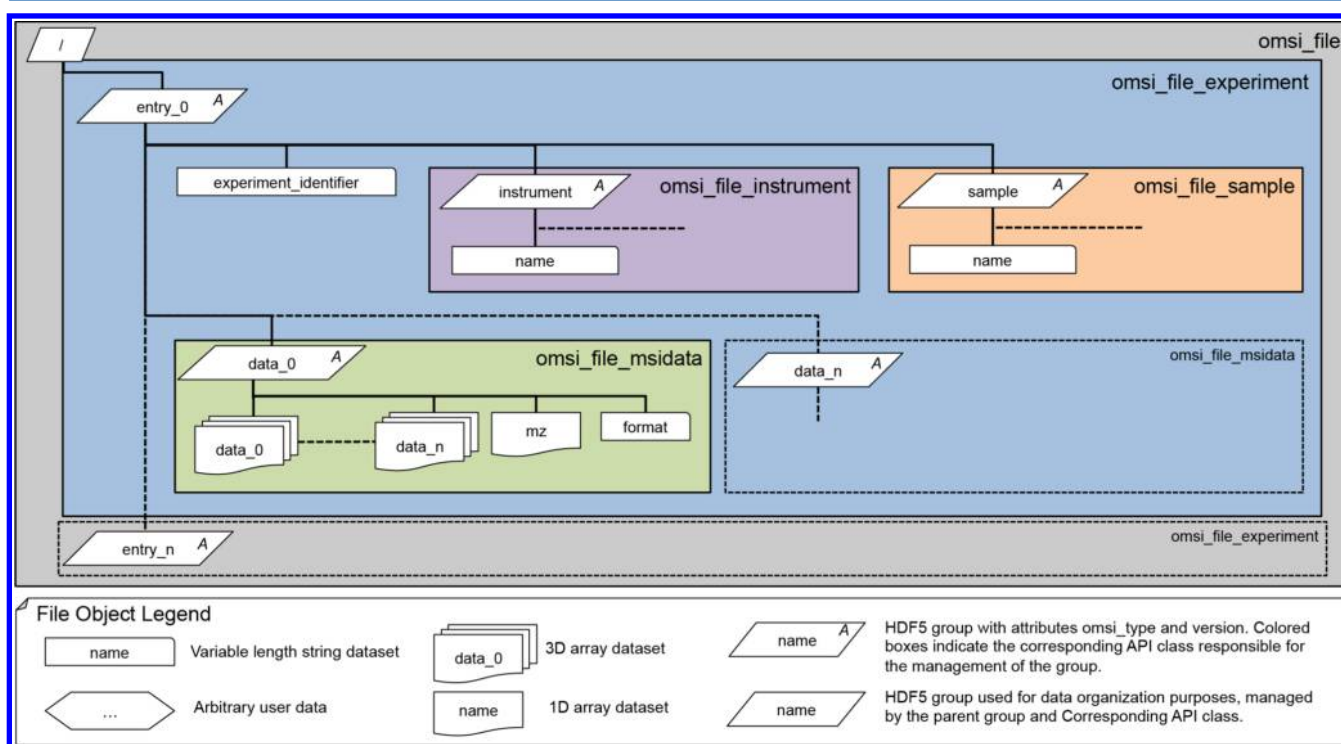


Figure 2. Overview of the hierarchy of the OpenMSI file format. The omisi file format and API follow the same semantic hierarchy. Each main HDF5 group is managed in the omisi file format API by a corresponding class (colored boxes) responsible for providing access to and creating the direct content of the group in the file.

differences between spatial and mass resolution and requires fast orthogonal access to spectra and ion images. A 2D MSI data set can be described as a three-dimensional cube of $(x,y,m/z)$ typically containing hundreds of thousands of positions (x,y) , with each position containing one or more spectra. Each spectrum describes the distribution of masses (m/z) at a given image location (x,y) and typically consists of 10^5 to 10^7 integer intensity values. See Part 1 in the Supporting Information for a detailed discussion of the MSI data requirements.

Here we describe a new paradigm in MSI data storage and processing based on a central data repository with Web-based processing and standardization of data formats. This approach takes advantage of advanced computing to make MSI data analysis rapid and accessible such that researchers can easily share and compare data. The OpenMSI file format, file API, and Web API described here build the foundation of the OpenMSI science resource (<http://openmsi.nersc.gov>). Overall the OpenMSI platform addresses many of the data challenges to MSI by making advanced, high-performance data analysis and computing easily accessible to MSI scientists, by enabling

fast sharing and access to raw MSI data and derived analyses via the Web (see Figure 1). All software has been implemented in Python using the hSpy library to interact with HDF5 and using the Django Web application framework for Web-related tasks.^{21,22}

■ OPENMSI FILE FORMAT AND API

In recent years, there have been significant advancements in methods for management, access, and storage of big data sets. One of the most widespread file formats for storing and accessing scientific data on massive parallel file systems is the HDF5 format.²³ HDF5 is a suite of technologies, consisting of a versatile data model, portable data format, and a widely accessible software library and API, which includes a rich set of integrated features for optimization of I/O performance and tools for managing, viewing, and manipulating HDF5 data collections.²³ We here describe the newly developed OpenMSI file format, a novel, extensible, portable, self-describing, parallel-aware MSI file format based on HDF5.

Specifically, what does this mean for the end user? *Extensibility* of the format means that the object-oriented design of the file structure and API enables flexible extension of the file format to handle custom application use-cases that may not yet be fully supported. *Portability* of the format means that OpenMSI files can be manipulated and viewed using standard HDF5 APIs and tools and used directly without change on all architectures and operating systems for which HDF5 is available. HDF5 is available for Windows and Unix-based systems, including Linux and MacOS, and well-supported HDF5 APIs exist for common programming languages, e.g., C, C++, Fortran, or Python. Also, many advanced visualization and analysis systems, e.g., Matlab, R, and VisIt, support HDF5 natively. *Self-describing* of the format means that all information about the data hierarchy, data types, etc. are directly encoded in the HDF5 files so that a user can, without prior knowledge about the file, explore the file hierarchy and load data similar to how one browses files and directories on a file system. Finally, in being *parallel-aware*, the HDF5 file format has been designed with parallel applications in mind. HDF5 provides many optimizations to enable and accelerate parallel I/O to single HDF5 files. This is fundamentally important to enable parallel algorithms to scale well on modern computing platforms, with I/O being one of the main bottlenecks in many parallel analyses. The OpenMSI file format addresses in this way many of the shortcomings of current MSI data formats. In the following we first describe how data is organized in the OpenMSI data format and then the optimization of the data layout.

In HDF5, data is stored as multidimensional data arrays. HDF5 supports a large range of standard data types as well as complex user-defined compound data types. Similar to directories in file systems, data sets can be organized via so-called groups in HDF5. In addition, groups and data sets may be assigned additional attributes, defining small metadata objects that can be used, e.g., to describe the nature and/or intended usage of a primary data object, e.g., data set or group.

Figure 2 describes the organization of raw MSI data and metadata via groups and data sets in OpenMSI HDF5-based data files. The file format also supports storage and management of derived analyses results, data provenance information, and data from other imaging modalities; this, however is beyond the scope of this manuscript.

All OpenMSI files contain a root group, /. Data associated with a particular imaging experiment is then stored in a corresponding /entry_# group. This allows for convenient storage of data from multiple related experiments in a single OpenMSI data file. Each /entry_# group contains a simple string data set /entry_#/experiment_identifier used to name and uniquely identify the experiment. Metadata describing the instrument and sample associated with the experiment are then stored in separate /entry_#/instrument and /entry_#/sample groups. Since the HDF5 format is self-describing, custom metadata may be added to the sample and instrument groups without violating the OpenMSI file format. Raw MSI data and derived analysis results are then stored in dedicated /entry_#/data_# groups. This data organization allows us to store an arbitrary number of raw MSI data sets and derived analyses for each experiment represented by a /entry_# group.

Storing raw MSI data in a /entry_#/data_# group, rather than directly in an HDF5 data set, provides us with great flexibility with respect to the data layout and storage of additional associated data sets. In the basic case where the raw

data defines a complete 3D MSI data cube, each data_# group contains (i) a string data set data_#/format indicating the data format used, (ii) a 1D floating-point array data_#/mz with the m/z values for the spectrum dimension, and (iii) a 3D array data_#/data_# with the 3D MSI data cube. With this design it is simple to organize MSI data in different formats (indicated by the data_#/format string) each optimized for different practical use cases. For example, to avoid possibly large storage overheads by storing an uncompressed, full 3D cube even if only a small region of interest has been imaged, the current implementation of the OpenMSI file API supports storage of mass spectra as a 2D data set of spectra along with additional small index data sets to record the relationship between spatial (x,y) locations and spectra. The design of the file format and API enables us to flexibly extend the file format to accommodate other optimized MSI storage formats as well as to integrate data from other imaging modalities, such as light microscopy. Organizing raw MSI in an HDF5 group allows us to also store multiple copies of the same data as numbered instances of data_#/data_# data sets. As described later, storing multiple copies of the same data using different data layouts can significantly accelerate orthogonal selective data accesses.

The OpenMSI file format API then follows an object-oriented design that models the group hierarchy of the file format. Each main HDF5 group (here called a managed group) is represented in the API by a corresponding class responsible for creation, management, and access of the corresponding HDF5 group type. The type of a managed group is uniquely determined by the naming scheme described above. For increased flexibility and extensibility of the file format, optional HDF5 attributes are associated with all main groups to indicate the interface class and version. In particular, the omsi_file_msidata interface class is designed to provide convenient access to raw MSI data stored in data_#/ groups independent of the data format used. The class provides a convenient array-based interface which allows a user to interact with the data as a 3D data cube independent of whether the MSI data set is stored as a full 3D cube or in a reduced data format. In the case where multiple copies of the same data set are available, the interface also automatically determines the data copy that is best suited to resolve a given data request. Providing a consistent data interface, independent of the underlying storage format, significantly simplifies the access to the data and eases development of data analysis algorithms. At the same time, the omsi_file_msidata interface allows developers to directly access all HDF5 data objects associated with the corresponding data_# group, enabling development of algorithms that are optimized to take advantage of different data organizations.

Data Layout Optimization. While HDF5 natively supports multidimensional arrays, on disk the data must be linearized to a one-dimensional data stream. The data layout describes the strategy by which the data is linearized. Traditional binary formats typically flatten MSI data into a single monolithic block on disk by storing the MSI data one spectrum at a time. These types of data layouts, in which the entire data is serialized into a monolithic block on disk that maps directly to a memory buffer of the size of the data sets, are referred to as contiguous data layouts. The traditional, one-spectrum-at-a-time continuous data layout is well suited to access single full spectra but shows very poor performance for access of ion images (see part 3 in the Supporting Information). To achieve optimal performance for the typical selective read operations on MSI data, in particular read of (i) spectra, (ii) ion

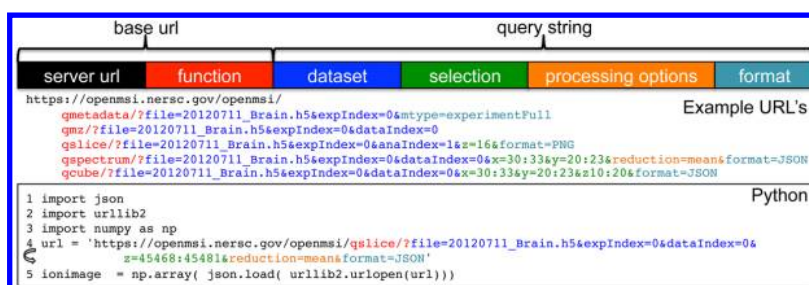


Figure 3. Illustration of the design of OpenMSI URL data requests (top), example URLs for data retrieval (middle), and a simple Python application example (bottom). Query string parameters may appear in arbitrary order.

images, and (iii) subcubes, our file format and API supports a number of data layout optimizations, including chunking, compression, and data replication, described in detail in the following. All data layout optimizations are implemented transparently for the user directly by HDF5 (chunking and compression) and the `omsi_file_msidata` API (data replication), allowing the user to interact with the data in a consistent manner independent of the data layout used to store the data on disk.

Accelerating Selective Data Access Operations Using Chunking. Chunked data layouts are an important alternative to contiguous data layouts. Chunking splits the data into multiple independent subparts, so-called, chunks, which are stored separately in the file. In HDF5, chunks may be stored in any order and at any position in the file, allowing chunks to be written and read independently, enabling efficient parallel read/write and improved I/O performance when operating on subsets of the data. Using chunking allows us to optimize the data layout to enable fast access to select data portions by improving data locality, hence, reducing the number of I/O operations needed and the size of the data that has to be traversed. Which chunked data layout is best depends greatly on the data access patterns to be optimized. Considering the most common data access patterns in MSI, we focus on the following main chunking strategies: (i) spectra aligned chunking, i.e., store a single full or partial spectrum per chunk; (ii) image-aligned chunking, i.e., store a single full or partial ion image per chunk; and (iii) hybrid chunking, i.e., store a 3D subcube describing a subset of multiple spectra and ion images (see part 5 in the Supporting Information).

Reducing Storage Cost and Accelerating I/O Using Compression. For chunked data layouts, HDF5 allows the data, i.e., the individual chunks, to pass through a series of user-defined I/O filters while being written to or read from disk. I/O filters are applied transparently by HDF5 whenever necessary, allowing the user to interact with the data in a consistent manner independent of the I/O filters used. Here, we focus on the use of compression filters with the goal to reduce storage cost and to accelerate data read operations by reducing the amount of data that needs to be transferred via the system bus and network. To ensure broad applicability of the OpenMSI file format, we focus on the use of gzip compression, which is (in contrast to, e.g., szip and LZFP) available by default as part of HDF5. Gzip defines a lossless compression scheme, i.e., no information is lost in the compression process.

Accelerating Orthogonal Data Accesses Using Data Replication. Linearization of the data on disk makes it impossible to achieve optimal performance for orthogonal data access operations, here access to spectra and ion images. Data layouts that are optimal for access of spectra are worst for

access to ion images and vice versa. While it may seem undesirable at first sight, replicated storage of MSI data using different optimized data layouts can significantly improve selective read performance, improve responsiveness of interactive applications, and substantially reduce the compute cost for parallel data analyses. Support for replicated data storage is implemented transparently in OpenMSI via the `omsi_file_msidata` API class, which allows users to interact with the data as a single, regular MSI data set. In the case where multiple copies of a MSI data set are available, the API automatically selects the data set that is most efficient to resolve a given data request and retrieves the data. In the context of OpenMSI, we often store two copies of the data, one optimized for access of spectra and one optimized for access of ion images with the chunking strategies automatically determined by the API. Even when storing the data twice, the resulting compressed MSI HDF5 files are in practice still substantially smaller (typically half the size or less) than the original raw binary data.

■ WEB API

The primary goals during the design of the OpenMSI Web API have been simplicity and usability. One primary objective has been to efficiently support exploratory analyses of the data via the Web while computationally intensive analyses are executed on high-performance compute resources at NERSC. We observe that most data analyses and visualization tasks are based on the following three data access pattern: (i) read spectra; (ii) read ion images; and (iii) read arbitrary subcubes of the data. We furthermore observe that while MSI data sets are large, the data required during individual data requests for data exploration are typically small.

The OpenMSI Web API consists of just five simple functions, `qmetadata`, `qmz`, `qslice`, `qspectrum`, and `qcube`, which together provide full access to the data, including metadata and raw MSI and derived analysis data. The basic methods are simple and can be effectively encoded in URL patterns (see Figure 3).

The `qmetadata` call is used to retrieve metadata information about which files are available on the server and which information is available in the files. The `qmz` call is used to retrieve information about the m/z data axis. Information about the m/z axis is in practice frequently reused. To avoid large overheads due to repeated transfer of the m/z data, we separate this into an independent call that is usually executed once at the beginning of any analysis.

The `qslice`, `qspectrum`, and `qcube` patterns are designed to provide easy-to-use support for the three most common selective access patterns, i.e., read ion image slices, read m/z spectra and read arbitrary subcubes of the data. To minimize

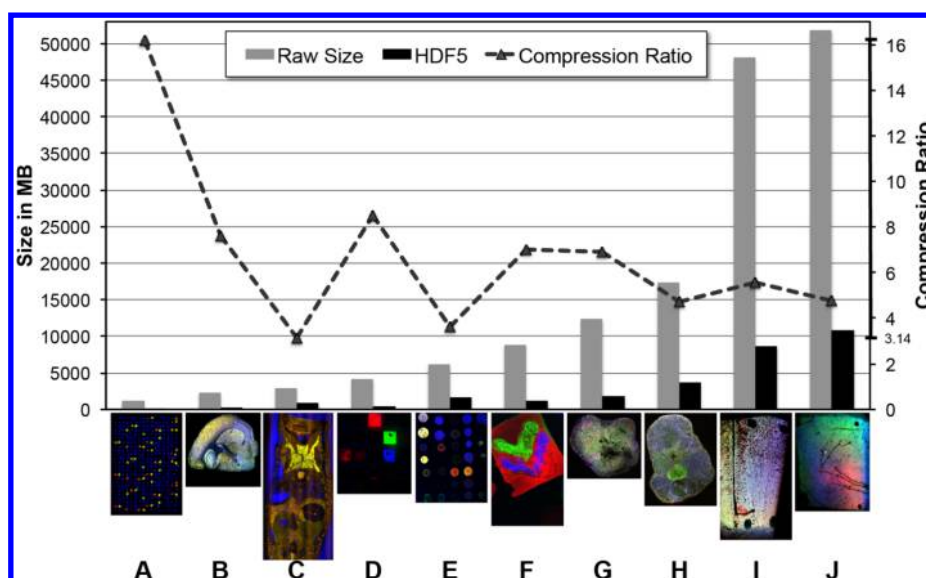


Figure 4. Size of original img data (gray bars) compared to the same data stored using the OpenMSI HDF5 data format (black bars) using gzip compression and a hybrid chunking of $4 \times 4 \times 2048$. We observe very good compression ratios (curve, right axis) in all cases. See also Figure 6 for more details about the data sets.

the amount of data that needs to be transferred via the Web, all access patterns support common data reduction operations, including maximum, minimum, average, standard deviation, variance, etc., which are applied on the server side prior to transfer of the data. This allows one to conveniently access, e.g., maximum projection ion images of selected m/z ranges or mean spectra for arbitrary sets of spectra, while only the final image or spectrum needs to be transferred via the Web. Further detailed descriptions of the five URL patterns are provided in part 4 in the Supporting Information.

We implemented the Web API in Python using the Django Web application framework, and like the file format, the Web service is cross platform compatible. In the current implementation of the Web API, we typically transfer all data either as easy-to-use JSON objects or as images (e.g., ion images of curve plots of spectra). To also support efficient retrieval of larger subsets of MSI data sets, we plan in the near future to also support retrieval of data directly in binary HDF5 format.

WEB-BASED DATA EXPLORATION

To demonstrate the applicability of the OpenMSI platform, we have developed an interactive Web-based, HTML5 data viewer based on the OpenMSI Web API. Using this viewer, a user can interactively define ion images and spectra to be displayed. Ion images and spectra are directly retrieved from the file from the complete, raw MSI data during each data request. As we show next, the OpenMSI data format and Web API can resolve these data requests in less than ~ 0.25 s via the Web even for large 50 GB MSI data sets. The viewer uses the standard URL patterns without any knowledge about the specific names of data sets or organization of the data in the HDF5 files. The Web client is in this way isolated from any specific implementation details on the server end and can flexibly display images and spectra for raw data and all derived analyses and their dependencies. The OpenMSI viewer is available online at <http://openmsi.nersc.gov/> (see also part 8 in the Supporting Information).

EVALUATION AND RESULTS

Data Layout Optimization and Performance. The goal of this study is to evaluate the effectiveness of the various data layout optimizations available as part of the OpenMSI format and API. Although text-based formats (e.g., XML) are very common in MSI,^{19,20} such formats are optimized for ease of use, not efficient data storage and fast data access. We, therefore, compare the various optimized data layouts to the common and much more efficient continuous binary data layout. However, we would like to note that due to the large storage and data read overheads of XML-based formats, the improvements in read performance and storage requirements would be in practice one or several orders of magnitude greater, if we were to take XML-based formats into account.

Identifying a Suitable Hybrid Chunked Data Layout. In practice, we expect spectra-aligned chunking to provide optimal performance for access of single, complete spectra while providing poor performance for access of ion images and vice versa for image-aligned chunked data (see parts 2, 3, and 5 in the Supporting Information). Hybrid chunked data layouts promise to provide fast read performance to arbitrary subcubes of the data while providing a compromise in performance for access to ion images and spectra. However, the large differences in resolution in physical space (x,y) and the spectra (m/z) in MSI data, make finding a well-performing hybrid chunking challenging. To identify a good hybrid chunked data layout we performed a large-scale autotuning-type experiment in which we explored: (i, ii, iii) the read performance of spectra, ion images, and subcubes; (iv) the data write performance; and (v) the storage requirements of all $k \times k \times l$ hybrid chunked data layouts with $k \in [1, 2, 4, 8, 16, 32]$ and $l \in [128, 256, 512, 1024, 2048, 4096, 8192]$ using a $100 \times 100 \times 100\,000$ sized data set as reference. These experiments have shown that a chunked layout of $4 \times 4 \times 2048$ may provide good performance in all above-mentioned evaluation criteria, and we use this configuration in the following to exemplify the performance characteristics of a hybrid chunked data layout. A detailed discussion of these experiments is available in part 5 in the Supporting Information.

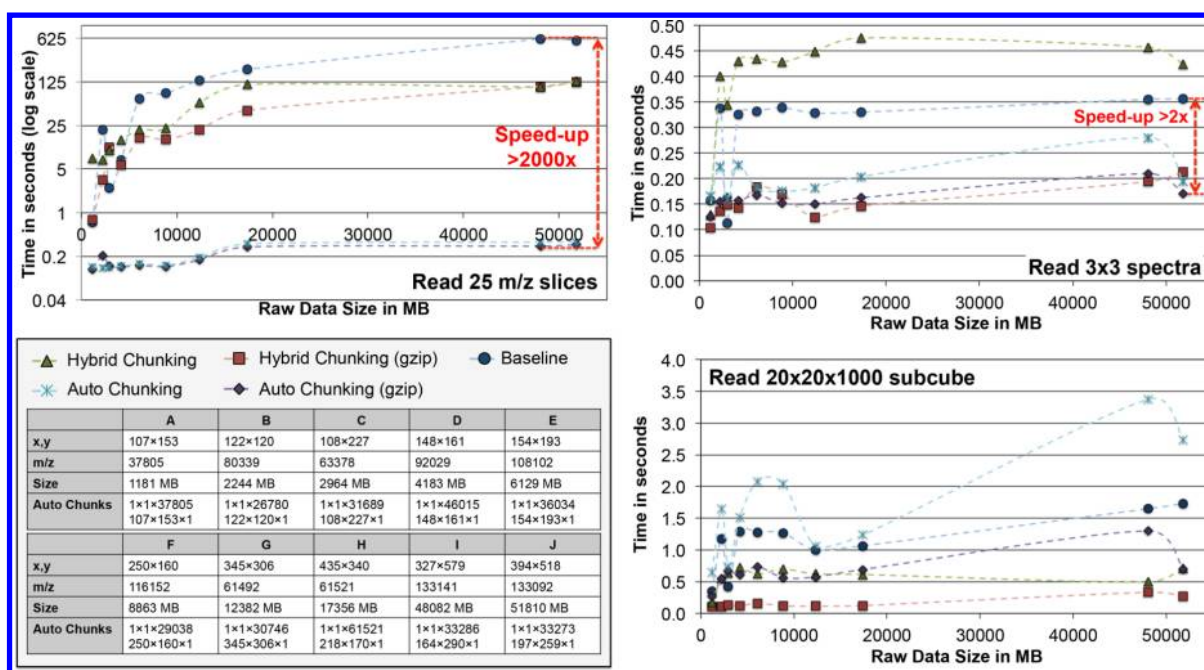


Figure 5. Serial read performance for the three most common data access patterns using the following data layouts (i) default monolithic layout (baseline), (ii,iii) hybrid ($4 \times 4 \times 2048$) chunking with and without compression, and (iv, v) auto chunking with and without compression. The auto chunking option uses two data copies optimized for retrieval of ion images and full spectra (see *Auto Chunks* row table above). The results were generated using a desktop workstation equipped with two Intel Xeon CPU, E5630 @ 2.53 GHz (only one core used here) and 18 GB of RAM.

Data Compression. Next we examined the ability of compression to reduce file size. Figure 4 shows a comparison of the size of a diverse set of MSI data sets stored using the OpenMSI file format compared to the standard raw binary data. We achieved 3–16 times compression without loss of data, e.g., a 3 GB image could be compressed to only 0.5 GB. This means, even when replicating the MSI data to accelerate data access, the resulting OpenMSI files are still much smaller than the raw binary data.

The combination of chunking and compression has also shown itself to be a viable solution for efficient storage of partial MSI data cubes and processed spectra. In this case, the data is still described as a complete MSI data cube. However, chunks are allocated by HDF5 during the first write, i.e., empty data chunks are never allocated by HDF5, while missing data values are automatically completed with zero values upon read. Furthermore, partial chunks are completed with zeros, which can be compressed very efficiently with very little overhead. To illustrate the effectiveness of this approach, we chose as an example an MSI data set of a lung with a resolution of $132 \times 149 \times 300\,000$. In the data set, an arbitrary region of interest consisting of 12 654 spectra has been imaged, and the spectra were preprocessed to remove background noise. From the total 5 900 400 000 data values (i.e., $\sim 11\,800$ MB) only 107 007 401 values (i.e., ~ 214 MB) are nonzero. Using a hybrid chunked layout of $4 \times 4 \times 2048$ in combination with compression, we require only ~ 196 MB to store the complete $132 \times 149 \times 300\,000$ data cube while allowing the user to seamlessly interact with the data as if it were a complete MSI data cube.

Optimizing Data Read Performance. The goal of the following tests has been to evaluate the performance of our file format and to identify the best suited data layouts. To evaluate the performance of different data layouts for the most common selective read patterns, we defined the following three representative test cases: (i) read 25 consecutive m/z slices,

(ii) read a 3×3 subset of complete spectra, and (iii) read a $20 \times 20 \times 1000$ subcube of the data. We here compare the performance of the following five data layouts: (i) the default monolithic layout (baseline), (ii) a hybrid ($4 \times 4 \times 2048$) chunking with compression, (iii) a hybrid ($4 \times 4 \times 2048$) chunking without compression, (iv) an autochunked data layout with compression, and (v) the same autochunked layout without compression. The autochunked data layout uses data replication in addition to chunking and compression to further optimize data read performance. Here, the data is stored twice using a spectrum-aligned and an image-aligned data chunking (see Figure 5), while the OpenMSI file API automatically chooses the best-suited data layout for a given data read.

To demonstrate the performance across a broad range of MSI data sets, we have chosen 10 MSI data sets that show varying spatial and m/z resolution and range in size between 1 GB up to 50 GB (see Figure 5). All tests were performed on a local desktop workstation equipped with two quad-core Intel Xeon E5630 CPUs running at 2.53 GHz and 18 GB of RAM. All data was stored on a local 1 TB regular spinning-disk hard drive. The tests were performed in serial, i.e., only one of the available compute cores was used in the tests. The tests were implemented in Python, and the source code of the tests is available in part 6 in the Supporting Information. We performed 50 random read operations for each of the 150 test cases, while we randomized (i) the m/z value for the image read, (ii) the (x/y) location for spectra read, and (iii) the $(x,y,m/z)$ origin for the subcube read. We report the 95th percentile of all measurements to demonstrate the expected sustained read performance for the different data layouts.

Figure 5 summarizes the results from all selective read performance tests. We observed that the baseline data layout shows particularly poor performance for the read of ion images (Figure 5, top left), requiring more than 600 s to retrieve just 25 consecutive images for data set I. Even though 25 ion images

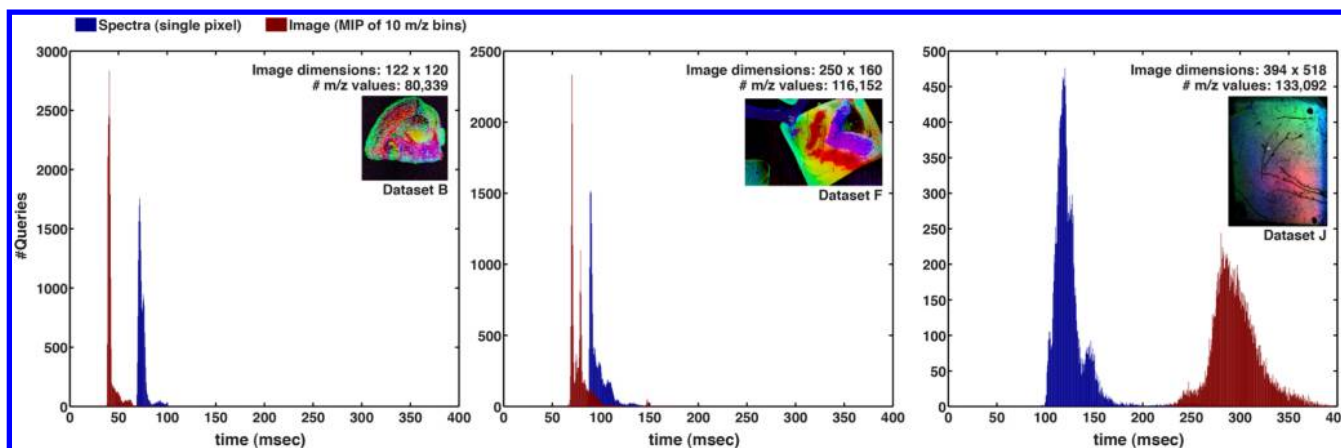


Figure 6. Web-based read performance. For each file, we evaluated 20 000 random single spectrum and 20 000 random ion-image data requests returning a maximum intensity projection over 10 m/z bins. The remote data requests were performed using Matlab (see also part 6 in the Supporting Information for further details).

constitute only ~ 9.5 MB of binary data, the entire ~ 48 GB data volume needs to be traversed to retrieve the data using the baseline layout. For the hybrid-chunked data layout (with compression) we observed speed-ups of up to $\sim 6.3\times$ for the image read, $\sim 2.6\times$ for the spectrum read, and $\sim 11.2\times$ for the subcube read compared to the baseline data layout. While this improvement in performance is significant, the read performance of the hybrid-chunked data layout is still insufficient for many time-critical analysis tasks and interactive data applications on large MSI data sets. This is due to the compromise the hybrid chunking is making in terms of performance to support orthogonal data access patterns.

Using the autochunked data layout (with compression) of the OpenMSI platform yielded speed-ups of more than $2000\times$ for the image read, enabling data read of ion images and spectra in less than 0.3 s even for the largest test data set. We also found that the performance for reading ion images using the autochunked approach depends mainly on the spatial resolution (i.e., number of pixels) of the images and is mostly independent of the resolution of the data in m/z (i.e., the total number of images). Similarly, the read performance of spectra is largely independent of the spatial resolution of the data in the autochunked case. These results suggest that this approach is scalable to meet the needs of data at scales higher than what is typically generated today. Using this approach enables for the first time fast retrieval of both spectra and ion image directly from the file without requiring caching of the data in memory. The performance we observe is sufficient to support interactive data exploration tasks even for very large MSI data sets.

We observe that the compressed data layouts perform significantly better even for reads from local disk when compared to the corresponding uncompressed data layouts. In cases where the data is stored on external storage systems, we would expect this behavior to be further amplified due to the reduced amount of data that needs to be transferred via the network when the data is compressed.

Performance of Web-Based Data Access Operations.

To test the performance of the OpenMSI platform's ability to access data across the Internet, images and spectra were programmatically retrieved from the server to a laptop computer. With the expectation that the size of a particular MSI file would significantly affect the time required to transfer results, we chose data sets J, F, and B (sizes are shown in Figure

6). We retrieved for each file 20 000 images and 20 000 spectra from the server at random spatial coordinates and m/z ranges and using the `qslice` and `qspectrum` commands, respectively. These requests were implemented in Matlab using the `urldread` command to retrieve the data as JSON structured text (the source code is provided in part 8 in the Supporting Information). The computer requesting the data was a MacBook Pro laptop with a 2.2 GHz Intel Core i7 processor and 8 GB of 1333 MHz RAM. All requests were made to a server at NERSC <https://openmsi.nersc.gov>. All of the files were stored on the physical, regular spinning disk of that server. The laptop was connected to a standard (1 Gb/s) office Ethernet connection in Berkeley, California.

The tests showed that the OpenMSI platform reliably supports subsecond data retrieval times for a wide range of MSI file sizes. For these data sets, the average time to retrieve a spectrum ranged between 74 and 126 ms. The average time to retrieve an image ranged between 43 and 294 ms. These results are consistent with the performance we have observed in the previous section for read performance directly from file. Figure 6 shows the histogram of the 20 000 requests for each of the six test cases. We observed very reliable read performance in all cases, indicated by the compact distribution of response times in the histograms.

URL-Based Data Analysis Sharing. Using the OpenMSI Web viewer prototype, a URL can be shared that presents the user with an interactive view based on specified visualization parameters, e.g., http://openmsi.nersc.gov/openmsi/client/viewer/?file=20120711_Brain.h5&expIndex=0&dataIndex=0&redValue=868.6&greenValue=840.6&blueValue=824.6&rangeValue=0.2&cursorX1=40&cursorY1=40&cursorX2=80&cursorY2=80

In this example, the m/z values and range are specified for creating an RGB image of three distinct ions and the spatial locations of two cursors are defined, selecting two spectra of interest plotted separately (see part 8 in the Supporting Information, OpenMSI Viewer).

CONCLUSION

We have described the OpenMSI platform and shown that it addresses many of the data challenges to MSI by making advanced, high-performance data analysis and computing easily accessible via the Web (<http://openmsi.nersc.gov>). The use of

the OpenMSI HDF5-based file format was found to be highly suited for this application. Optimization of data layouts using chunking, compression, and data replication were found to be critical enablers of rapid data access and resulted in >2000-fold improvement in image access. The Web-based API design enables easy-to-implement data access patterns with data retrieval speeds of less than 0.3 s across the Internet even for large 50 GB MSI data sets. By making MSI easily accessible, without the need for advanced knowledge in high-performance data analysis and computing, OpenMSI promises to transform how MSI is used in practice and promotes the widespread adoption of MSI as a novel imaging approach. While this has been beyond the scope of this manuscript, we would like to note that OpenMSI also supports management, provenance, and visualization of derived data analyses. In addition, the format is well suited to three-dimensional imaging, SIMS, and other mass spectrometry imaging approaches.

■ ASSOCIATED CONTENT

📄 Supporting Information

Additional information as noted in text. This material is available free of charge via the Internet at <http://pubs.acs.org>.

■ AUTHOR INFORMATION

Corresponding Authors

*E-mail: oruebel@lbl.gov.

*E-mail: bpbowen@lbl.gov

Author Contributions

O. Rübél and B. Bowen are the project leads of OpenMSI and are responsible for conception, overall design, and writing of the manuscript. A. Greiner and K. Louie contributed to design, development, and testing of the Web-based viewer and reviewed the manuscript. S. Cholia contributed to the design, development, and setup of the OpenMSI Web services and server and reviewed the manuscript. T.R.N. and E.W.B. contributed to the design and manuscript editing.

Notes

The authors declare no competing financial interest.

■ ACKNOWLEDGMENTS

This work was supported by and used resources of the National Energy Research Scientific Computing Center (NERSC), Ecosystems and Networks Integrated with Genes and Molecular Assemblies (ENIGMA), and the Low Dose Radiation Programs, which are supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. We thank K. Yelick, G. Karpen, and M. Maxon for valuable discussions, guidance, and support. We thank D. Skinner and the Outreach, Software and Programming Group at NERSC for their ongoing efforts and support to help deliver scientific data and high-performance computing to science communities. We thank R. E. Lance-Rübél for her patience, support, and advice.

■ REFERENCES

- (1) Garden, R. W.; Sweedler, J. V. *Anal. Chem.* **2000**, *72* (1), 30–36.
- (2) Caprioli, R. M.; Farmer, T. B.; Gile, J. *Anal. Chem.* **1997**, *69* (23), 4751–4760.
- (3) Stoeckli, M.; Chaurand, P.; Hallahan, D. E.; Caprioli, R. M. *Nat. Med.* **2001**, *7* (4), 493–496.
- (4) Monroe, E. B.; Annangudi, S. P.; Hatcher, N. G.; Gutstein, H. B.; Rubakhin, S. S.; Sweedler, J. V. *Proteomics* **2008**, *8* (18), 3746–3754.

- (5) Wiseman, J. M.; Ifa, D. R.; Zhu, Y.; Kissinger, C. B.; Manicke, N. E.; Kissinger, P. T.; Cooks, R. G. *Proc. Natl. Acad. Sci. U.S.A.* **2008**, *105* (47), 18120–18125.
- (6) McLean, J. A.; Ridenour, W. B.; Caprioli, R. M. *J. Mass Spectrom.* **2007**, *42* (8), 1099–1105.
- (7) Louie, K. B.; Bowen, B. P.; McAlhany, S.; Huang, Y.; Price, J. C.; Mao, J. H.; Hellerstein, M.; Northen, T. R. *Sci. Rep.-U.K.* **2013**, *3*, 1656.
- (8) Reindl, W.; Bowen, B. P.; Balamotis, M. A.; Green, J. E.; Northen, T. R. *Integr. Biol.* **2011**, *3* (4), 460–467.
- (9) Rompp, A.; Guenther, S.; Takats, Z.; Spengler, B. *Anal. Bioanal. Chem.* **2011**, *401* (1), 65–73.
- (10) Suits, F.; Fehniger, T. E.; Vegvari, A.; Marko-Varga, G.; Horvatovich, P. *Anal. Chem.* **2013**, *85* (9), 4398–4404.
- (11) Parry, R. M.; Galhena, A. S.; Gamage, C. M.; Bennett, R. V.; Wang, M. D.; Fernández, F. M. *J. Am. Soc. Mass Spectrom.* **2013**, *24* (4), 646–649.
- (12) Oetjen, J.; Aichler, M.; Trede, D.; Strehlow, J.; Berger, J.; Heldmann, S.; Becker, M.; Gottschalk, M.; Kobarg, J. H.; Wirtz, S.; Schiffler, S.; Thiele, H.; Walch, A.; Maass, P.; Alexandrov, T. *J. Proteomics* **2013**, *90*, 52–60.
- (13) Alexandrov, T.; Becker, M.; Guntinas-Lichius, O.; Ernst, G.; von Eggeling, F. *J. Cancer Res. Clin. Oncol.* **2013**, *139* (1), 85–95.
- (14) Trede, D.; Schiffler, S.; Becker, M.; Wirtz, S.; Steinhorst, K.; Strehlow, J.; Aichler, M.; Kobarg, J. H.; Oetjen, J.; Dyatlov, A.; Heldmann, S.; Walch, A.; Thiele, H.; Maass, P.; Alexandrov, T. *Anal. Chem.* **2012**, *84* (14), 6079–6087.
- (15) Trede, D.; Kobarg, J. H.; Oetjen, J.; Thiele, H.; Maass, P.; Alexandrov, T. *J. Integr. Bioinf.* **2012**, *9* (1), 189.
- (16) Moree, W. J.; Phelan, V. V.; Wu, C.-H.; Bandeira, N.; Cornett, D. S.; Duggan, B. M.; Dorresteijn, P. C. *Proc. Natl. Acad. Sci. U.S.A.* **2012**, *109* (34), 13811–13816.
- (17) Alexandrov, T. *BMC Bioinf.* **2012**, *13* (Suppl 16), S11.
- (18) Bruand, J.; Alexandrov, T.; Sistla, S.; Wisztorski, M.; Meriaux, C.; Becker, M.; Salzet, M.; Fournier, I.; Macagno, E.; Bafna, V. *J. Proteome Res.* **2011**, *10* (10), 4734–4743.
- (19) Römpp, A.; Schramm, T.; Hester, A.; Klinkert, I.; Both, J.-P.; Heeren, R. M. A.; Stöckli, M.; Spengler, B. *Methods Mol. Biol.* **2011**, *696*, 205–224.
- (20) Martens, L.; Chambers, M.; Sturm, M.; Kessner, D.; Levander, F.; Shofstahl, J.; Tang, W. H.; Römpp, A.; Neumann, S.; Pizarro, A. D.; Montecchi-Palazzi, L.; Tasman, N.; Coleman, M.; Reisinger, F.; Souda, P.; Hermjakob, H.; Binz, P. A.; Deutsch, E. W. *Mol. Cell. Proteomics* **2010**, *10* (1), R110.000133–R110.000133.
- (21) McKinney, W. *Python for Data Analysis*; O'Reilly Media, Inc.: Sebastopol, CA, 2012; p 452.
- (22) Forcier, J.; Bissex, P.; Chun, W. *Python Web Development with Django*; Pearson Education, Inc.: Boston, MA, 2009; p 408.
- (23) The HDF Group. *HDF5 User's Guide*; November 2011.