Supplementary Material

Table of Contents

Algorithm descriptions

The library provides implementations of 4 different algorithms, 1 designed to compress first order smooth data like retention time or M/Z arrays, 1 designed to transform first order smooth data for more efficient zlib compression, and 2 for compressing non-smooth data with lower requirements on precision like ion count arrays.

Implementations and unit test are provided in `C++` and `java`.

MS Numpress positive integer compression (numPic)

Intended for ion count data, this compression simply rounds values to the nearest integer, and stores

these integers in a truncated form which is effective for values close to zero.

MS Numpress short logged float compression (numSlof)

Also targeting ion count data, this compression takes the natural logarithm of values, multiplies by a scaling factor and rounds to the nearest integer. For typical ion count dynamic range these values fits into two byte integers, so only the two least significant bytes of the integer are stored.

The scaling factor can be chosen manually, but the library also contains a function for retrieving the optimal numSlof scaling factor for a given data array. In this case optimal refers to the greatest precision storable for this data in 2 bytes. Since the scaling factor is variable, it is stored as a regular double precision float first in the encoding, and automatically parsed during decoding.

MS Numpress linear prediction compression (numLin)

This compression uses a fixed point representation, achieved by multiplication by a scaling factor and rounding to the nearest integer. To exploit the frequent linearity of the data, linear prediction is then used in the following way.

The first two values are stored without compression as 4 byte integers. For each following value a linear prediction is made from the two previous values:

```
Xpred   = (X(n) - X(n-1)) + X(n)
Xres    = Xpred - X(n+1)
```

The residual `Xres` is then stored, using the same truncated integer representation as in Numpress Pic.

The scaling factor can be chosen manually, but the library also contains a function for retrieving the optimal numLin scaling factor for a given data array. Again, optimal here refers to the greatest precision for the fixed point byte size. Since the scaling factor is variable, it is stored as a regular double precision float first in the encoding, and automatically parsed during decoding.

MS Numpress safe linear prediction (numSafe)

This transformation uses the same linear prediction as numLin, but without the fixed point representation or integer truncation. This means that no compression is achieved and the resulting binary array will be exactly the same size as the input array. Note that even so some minimal degradation will occur due to the double operation rounding errors, but as sequential compression and decompression is hardly performed the transformation should still be practically lossless.

Truncated integer representation

This encoding works on a 4 byte integer, by truncating initial zeros or ones. If the initial (most significant) half byte is 0x0 or 0xf, the number of such halfbytes starting from the most significant is stored in a count halfbyte. This initial count is then followed by the rest of the int's halfbytes, in little-endian order. A count halfbyte c of

```
0 <= c <= 8         is interpreted as an initial c      0x0 halfbytes
9 <= c <= 15        is interpreted as an initial (c-8)  0xf halfbytes
```

Examples:

```
int     c       rest
0   =>  0x8
-1  =>  0xf     0xf
23  =>  0x6     0x7 0x1
```

Implementation notes

We recommend to simply embed the numpress library source files in your source when implementing numpress support in new tools. At the point of writing, implementations so far are all open source, which means there are many reference implementations, especially for C++. For the time being, we also recommend numpress writer implementations to produce mzML 1.1 compliant files, meaning amongst other things that only one compression per binary should be allowed (uncompressed, zlib, numPic, numSlof or numLin), and the 32/64-bit tag written out even though it's unnecessary with numpress compressions.

During this project we also implemented support for reading and writing imzML with ProteoWizard, in an attempt to investigate eventual gains with not having to encode all data in base64 binary data. While there is the obvious gain in file size from the reduced redundancy, we did not see any conclusive improvements in handling speed using our implementation. Nevertheless, we are in contact with the ProteoWizard team to eventually include this imzML-support in ProteoWizard. It should also be noted that imzML was simply used as a means for storing binary data in an external binary file, and we have not added any handling of imaging relevant information.

Tools and scripts used for testing

For testing we have extended ProteoWizards msconvert. Added abilities include supporting numpress compressions, allowing both numpress and zlib compression on the same binary data array, writing and reading imzML, as well as explicitly setting the write buffer size. At the time of writing, numpress support and double compression is already included in the official msconvert, and inclusion of other amendments is discussed. Access to exact binaries and scripts used for testing will be provided upon request.

Missing values

We were unable to achieve a few measurements. Because of their proprietary nature we cannot write custom vendor files. Reading the largest SWATH DIA file in the mzML.gz format crashed on all computers in a failure to allocate memory, even on the 24 GB machine. Reading of the extracted chromatograms (file 6) crashed for unknown reasons when trying to read the mz5 and mz5zlib formats. Unpromising and likely incorrect preliminary results for the imzML-based formats stopped completion of read and write timing for these formats.

Supplementary Table 1: MS data files

id	resolution	type	original size (MB)	vendor	peak picked	instrument
1	high	DDA	724	Agilent	no	QTOF
2	low	SRM	13.8	Thermo	no	TSQ Vantage
3	low	DDA	16.2	Thermo	no	LCQ
4	high	SWATH DIA	430[1]	ABI Sciex	no	Triple TOF
5	high	SWATH DIA	2500[1]	ABI Sciex	no	Triple TOF
6	high	Chrom from 5[2]	219.7	ABI Sciex	no	Triple TOF
7	high	DDA	520.8	Thermo	no	Orbitrap XL
8	high	MS+MS/MS	538	Waters	no	QTOF Ultima
9	high	DDA	202[3]	Thermo	yes	Orbitrap XL
10	high	DDA	577	Thermo	no	Orbitrap Velos

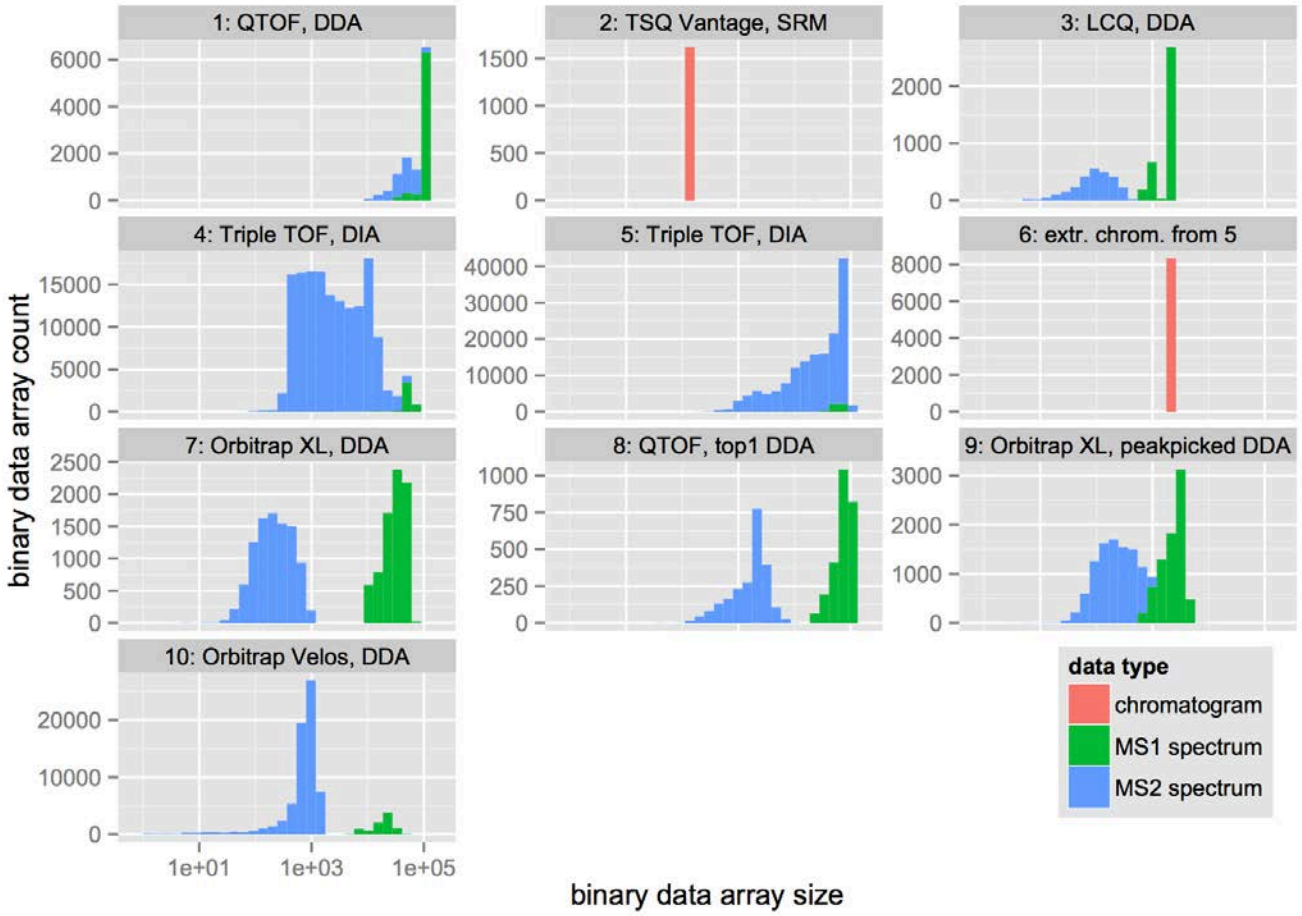
[1)] The large size difference comes from the very different samples, were 4) is an information sparse dilution of stable isotope peptides in water, and 5) is a information dense yeast extract (Suppl. Fig. 1).

[2)] Chromatograms were extracted using a custom extraction tool, by deconvolution using a top-hat filter of 10 ppm total width. This means peaks within 10 ppm of a known fragment m/z, in MS2 spectra of the swath corresponding to the known precursor m/z, were summed with weights decreasing linearly with distance from the exact fragment mass.

[3)] This is in mzML format since we cannot write custom Thermo raw files.

Supplementary Figure 1: Binary data array length distributions

Distributions of binary data array lengths for the 10 MS data files used for benchmarking. Depending on instrument speed, a varying amount of MS2 spectra can be seem in relation to the number of MS1 spectra. Also, the more complex MS1 spectra tend to give long binary data arrays compared to the simpler MS2 spectra, especially for high-resolution and/or high sampling frequency instruments. File 4 and 5 are acquired with the same method on the same instrument, but on hundreds of synthetic peptides in file 4, and a yeast lysate in file 5.



Supplementary Table 2: All tested compression schemes

Table showing the tested compression schemes

scheme	binary data type			
	m/z	ion count	retention time	whole file
mzML	-	-	-	-
mzML.gz	-	-	-	gzip
zlib	zlib	zlib	zlib	-
mz5	-	-	-	-
mz5zlib	zlib	zlib	zlib	-
numAll	numLin	numSlof	numLin	-
numAll.gz	numLin	numSlof	numLin	gzip
numAllzlib	numLin & zlib	numSlof & zlib	numLin & zlib	gzip
numLin	numLin	-	numLin	-
numPic	-	numPic	-	-
numSlof	-	numSlof	-	-
numLinZlib	numLin & zlib	-	-	-
numSafeZlib	numSafe & zlib	-	-	-
imzML	-	-	-	-
imzMLnumAll	numLin	numSlof	numLin	-
imzMLnumAllzlib	numLin & zlib	numSlof & zlib	numLin & zlib	-
imzMLzlib	zlib	zlib	zlib	-
vendor	-	-	-	-

Supplementary Table 3: Max relative error for numSlof on 10 test set files

file	negative	positive
06May2010_TestSuperHirn_BSA30fmol_01.d.ms1.clean.diff	-7.43E-005	7.59E-005
110620_fract_scxB05.raw.ms1.clean.diff	-0.000151466	0.000176515
110620_fract_scxB05.raw.ms2.clean.diff	-0.000127385	0.000134896
120224_006_SW.wiff.ms1.clean.diff	-6.63E-005	6.58E-005
120224_006_SW.wiff.ms2.clean.diff	-5.44E-005	5.41E-005
120302_006_SW.wiff.ms1.clean.diff	-6.73E-005	6.79E-005
120302_006_SW.wiff.ms2.clean.diff	-5.71E-005	5.71E-005
121213_PhosphoMRM_TiO2_discovery.RAW.ms1.clean.diff	-0.000160543	0.000149204
121213_PhosphoMRM_TiO2_discovery.RAW.ms2.clean.diff	-1.03E-005	1.81E-005
ADH_100126_mix.raw.ms1.clean.diff	-4.45E-005	4.32E-005
Velos_120905_09.raw.ms1.clean.diff	-0.000141388	9.60E-005
Velos_120905_09.raw.ms2.clean.diff	-6.37E-005	5.94E-005
peakPicked.121213_PhosphoMRM_TiO2_discovery.ms1.clean.diff	-9.86E-005	9.84E-005
peakPicked.121213_PhosphoMRM_TiO2_discovery.ms2.clean.diff	-1.03E-005	1.81E-005

Supplementary Table 4: Max relative error for numLin on 10 test set files

file	negative	positive
06May2010_TestSuperHirn_BSA30fmol_01.d.ms1.clean.diff	-2.33E-010	2.33E-010
110620_fract_scxB05.raw.ms1.clean.diff	-2.34E-010	2.35E-010
110620_fract_scxB05.raw.ms2.clean.diff	-2.67E-010	3.14E-010
120224_006_SW.wiff.ms1.clean.diff	-2.33E-010	2.32E-010
120224_006_SW.wiff.ms2.clean.diff	-1.61E-009	1.59E-009
120302_006_SW.wiff.ms1.clean.diff	-2.33E-010	2.30E-010
120302_006_SW.wiff.ms2.clean.diff	-1.57E-009	1.63E-009
121213_PhosphoMRM_TiO2_discovery.RAW.ms1.clean.diff	-2.33E-010	2.33E-010
121213_PhosphoMRM_TiO2_discovery.RAW.ms2.clean.diff	-1.22E-010	2.79E-010
ADH_100126_mix.raw.ms1.clean.diff	-2.33E-010	2.33E-010
Velos_120905_09.raw.ms1.clean.diff	-2.32E-010	2.32E-010
Velos_120905_09.raw.ms2.clean.diff	-2.39E-010	2.69E-010
peakPicked.121213_PhosphoMRM_TiO2_discovery.ms1.clean.diff	-2.33E-010	2.31E-010
peakPicked.121213_PhosphoMRM_TiO2_discovery.ms2.clean.diff	-1.22E-010	2.79E-010

Supplementary Table 5: Max relative error for numPic on 10 test set files

file	negative	positive
06May2010_TestSuperHirn_BSA30fmol_01.d.ms1.clean.diff	0	0
110620_fract_scxB05.raw.ms1.clean.diff	0	0
110620_fract_scxB05.raw.ms2.clean.diff	0	0
120224_006_SW.wiff.ms1.clean.diff	0	0
120224_006_SW.wiff.ms2.clean.diff	0	0
120302_006_SW.wiff.ms1.clean.diff	0	0
120302_006_SW.wiff.ms2.clean.diff	0	0
121213_PhosphoMRM_TiO2_discovery.RAW.ms1.clean.diff	-0.66481	0.390022
121213_PhosphoMRM_TiO2_discovery.RAW.ms2.clean.diff	-0.14251	0.0750765
ADH_100126_mix.raw.ms1.clean.diff	0	0
Velos_120905_09.raw.ms1.clean.diff	-0.52203	0.198162
Velos_120905_09.raw.ms2.clean.diff	-0.13293	0.114401
peakPicked.121213_PhosphoMRM_TiO2_discovery.ms1.clean.diff	-0.00113	0.0011207
peakPicked.121213_PhosphoMRM_TiO2_discovery.ms2.clean.diff	-0.14251	0.0750765

Supplementary Table 6: Max error for numPic on 10 test set files

file	negative	positive
06May2010_TestSuperHirn_BSA30fmol_01.d.ms1.clean.diff	0	0
110620_fract_scxB05.raw.ms1.clean.diff	0	0
110620_fract_scxB05.raw.ms2.clean.diff	0	0
120224_006_SW.wiff.ms1.clean.diff	0	0
120224_006_SW.wiff.ms2.clean.diff	0	0
120302_006_SW.wiff.ms1.clean.diff	0	0
120302_006_SW.wiff.ms2.clean.diff	0	0
121213_PhosphoMRM_TiO2_discovery.RAW.ms1.clean.diff	-0.499985	0.5
121213_PhosphoMRM_TiO2_discovery.RAW.ms2.clean.diff	-0.42815	0.325621
ADH_100126_mix.raw.ms1.clean.diff	0	0
Velos_120905_09.raw.ms1.clean.diff	-0.499939	0.5
Velos_120905_09.raw.ms2.clean.diff	-0.49959	0.499481
peakPicked.121213_PhosphoMRM_TiO2_discovery.ms1.clean.diff	-0.499939	0.5
peakPicked.121213_PhosphoMRM_TiO2_discovery.ms2.clean.diff	-0.42815	0.325621

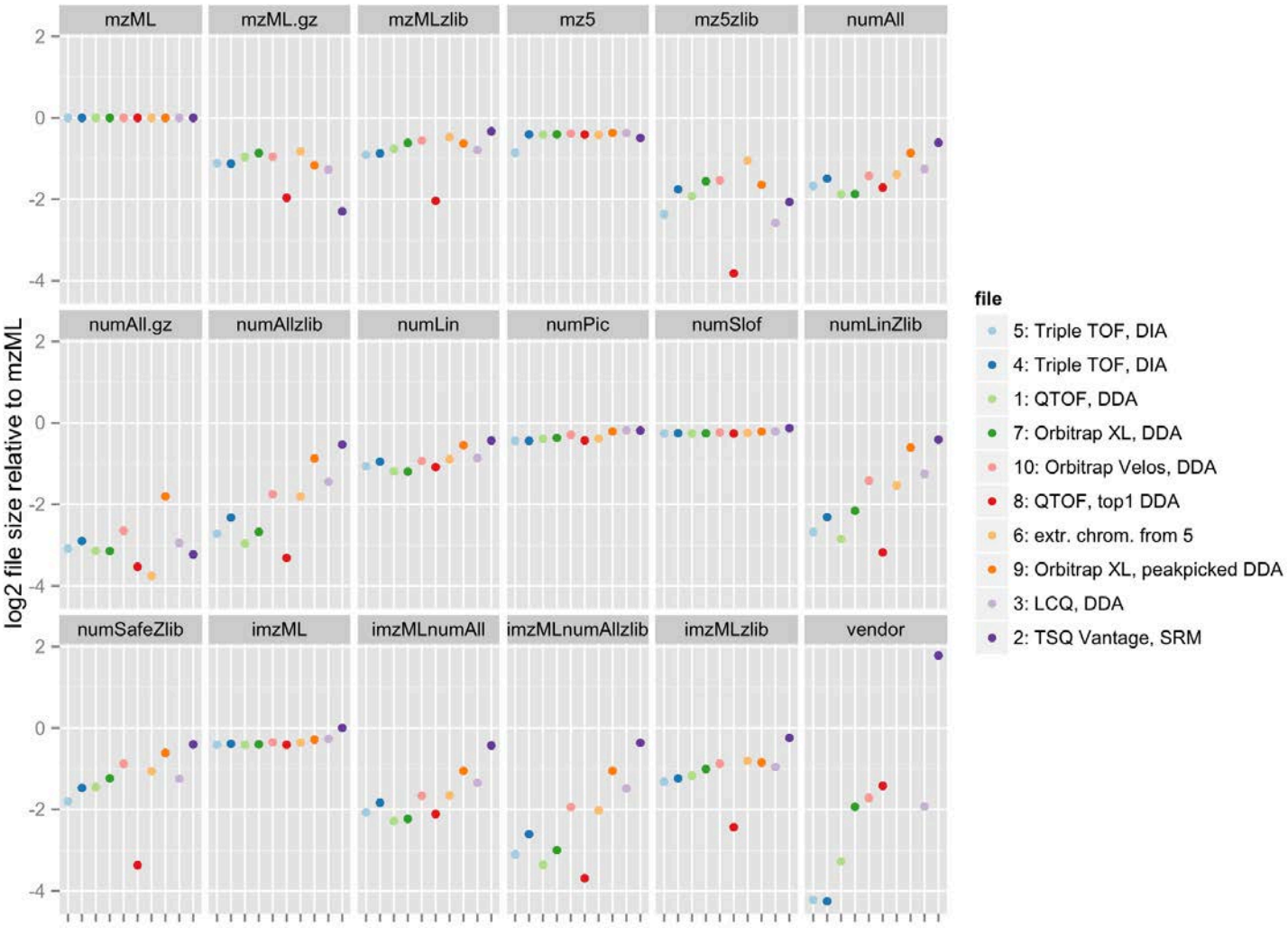
Supplementary Table 7: Size and timing statistics

color legend:	20%	50%	100%	200%	500%

	size		high performance computers				all computers			
			read time		write time		read time		write time	
	mean	sd	mean	sd	mean	sd	mean	sd	mean	sd
imzML	79.9%	7.3%	30.7%	31.0%	79.8%	16.6%	32.1%	31.6%	198.5%	581.1%
imzMLnumAll	34.2%	16.1%	26.3%	31.4%	106.2%	37.3%	31.0%	34.1%	288.3%	784.3%
imzMLnumAllzlib	27.1%	21.4%	27.4%	33.1%	144.6%	64.5%	29.4%	33.2%	140.6%	71.9%
imzMLzlib	49.8%	16.1%	30.5%	31.0%	237.4%	107.6%	34.2%	33.4%	257.4%	176.6%
mz5	73.3%	6.4%	23.3%	14.1%	69.7%	15.6%	28.8%	21.2%	135.6%	220.1%
mz5zlib	27.2%	11.1%	39.4%	7.1%	101.1%	22.5%	39.3%	13.6%	118.2%	103.3%
mzML	100.0%	0.0%	100.0%	0.0%	100.0%	0.0%	100.0%	0.0%	100.0%	0.0%
mzML.gz	43.9%	11.7%	179.2%	53.0%	836.7%	776.5%	153.7%	56.5%	705.3%	713.1%
mzMLzlib	59.9%	14.5%	113.8%	15.0%	257.3%	111.1%	102.5%	35.1%	218.6%	110.8%
numAll	39.0%	12.1%	63.8%	11.2%	97.0%	8.5%	62.3%	22.2%	117.5%	129.3%
numAll.gz	13.2%	5.8%	87.0%	19.8%	281.0%	129.2%	78.9%	29.1%	237.7%	125.1%
numAllzlib	29.3%	18.9%	71.2%	17.8%	136.2%	30.3%	68.6%	28.5%	121.9%	39.9%
numLin	53.8%	9.8%	70.7%	9.8%	92.7%	6.2%	65.8%	20.6%	89.7%	18.5%
numLinZlib	33.8%	21.5%	76.8%	22.4%	157.2%	54.3%	71.8%	31.2%	136.9%	58.3%
numPic	79.7%	5.8%	82.1%	14.9%	111.5%	9.7%	75.4%	24.9%	107.7%	31.0%
numSafeZlib	43.9%	18.1%	102.8%	18.5%	288.5%	137.5%	93.6%	35.1%	240.2%	128.4%
numSlof	85.2%	2.5%	92.7%	1.4%	114.2%	49.6%	85.3%	22.9%	109.7%	41.9%
vendor	60.5%	110.9%	247.4%	499.6%	NA	NA	208.9%	414.3%	NA	NA

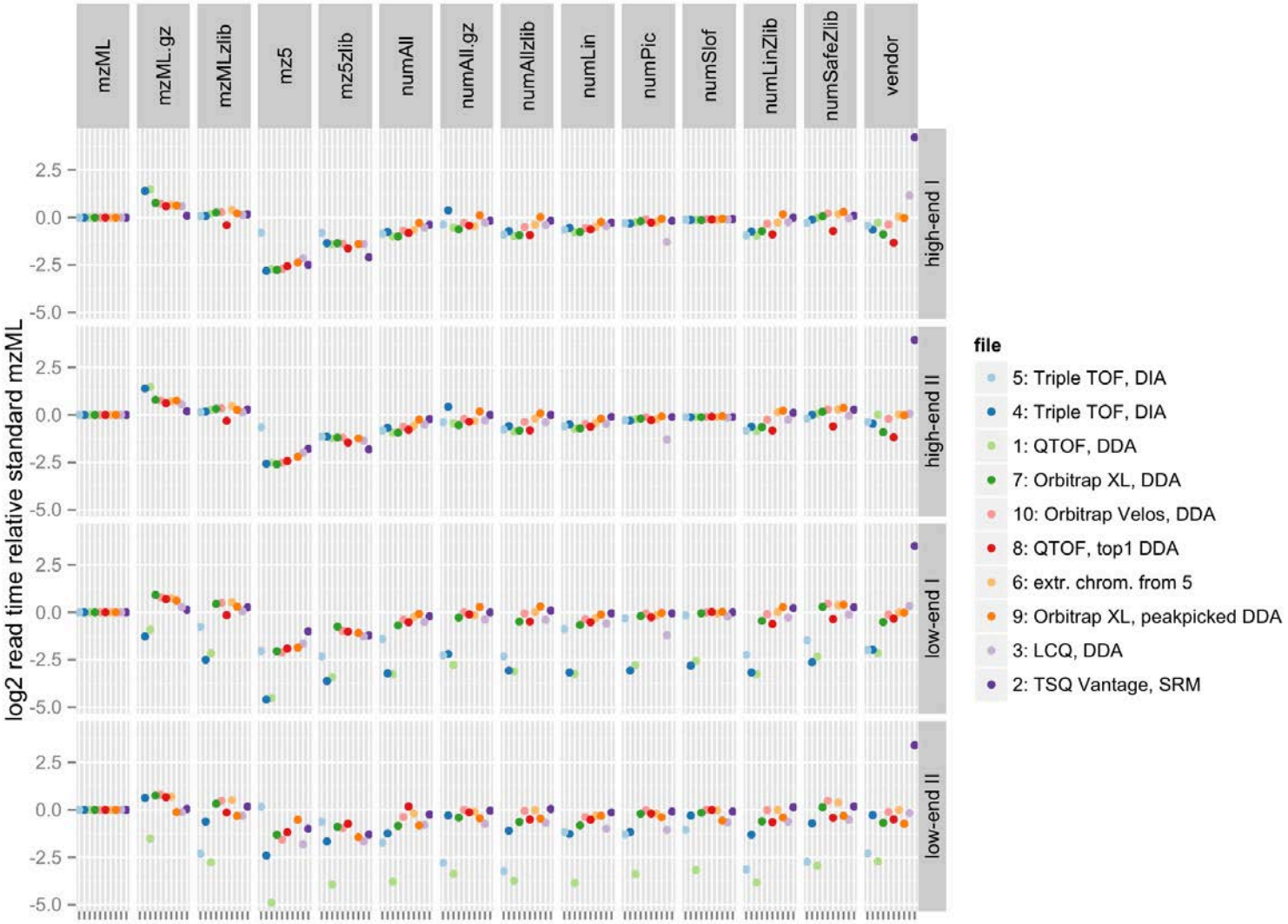
Supplementary Figure 2: Relative file size

Log_2 file size subtracted by log_2 mzML file size for all file formats and files. Files are sorted in descending mzML-size order (same as in Fig 1c). Note that file 6 and 9 do not exists in vendor format, since they are computer derivates on other files.



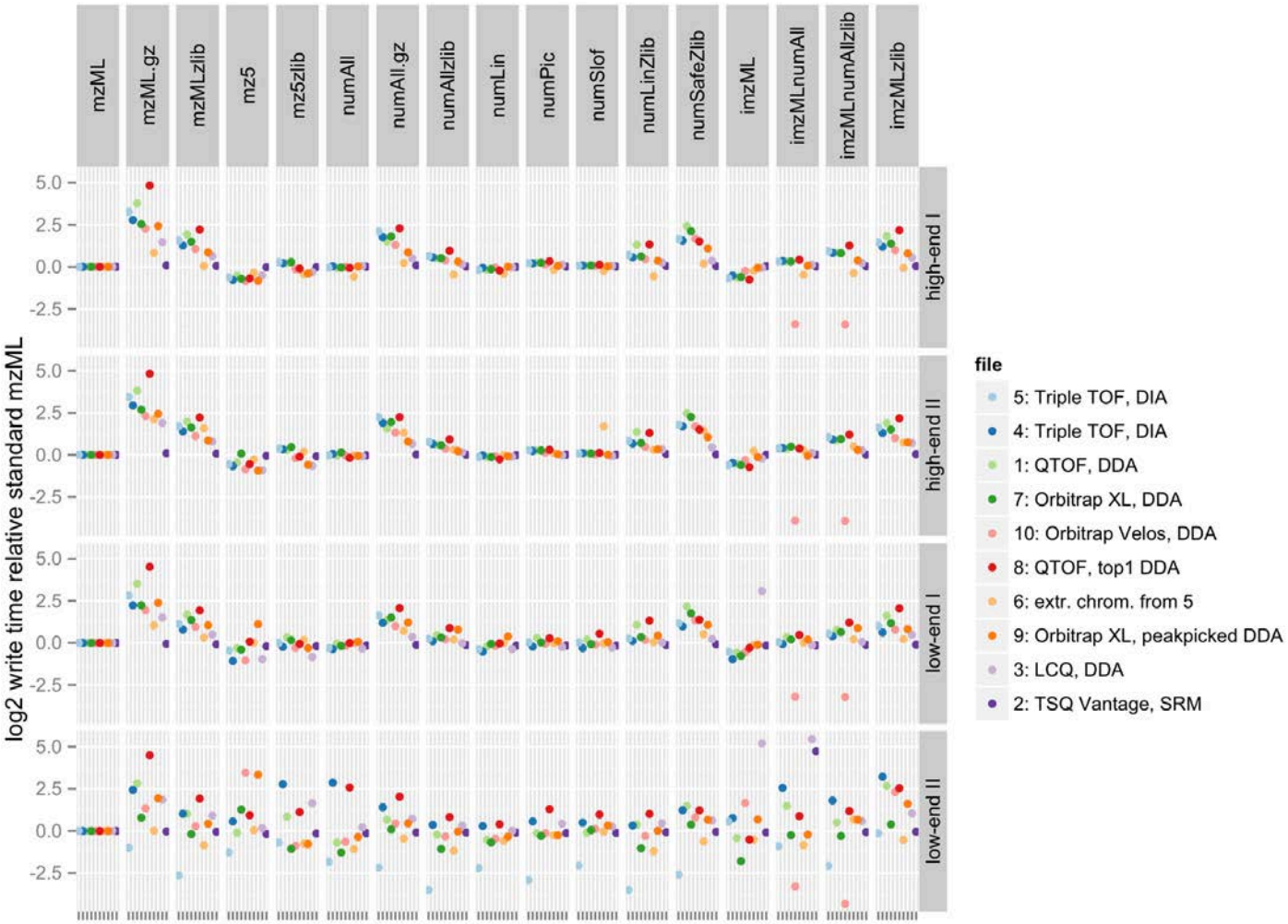
Supplementary Figure 3: Relative read time

Log_2 read times subtracted by mzML log_2 read time for all file formats, computers and files. Files are sorted in descending mzML-size order (same as in Fig 1c).



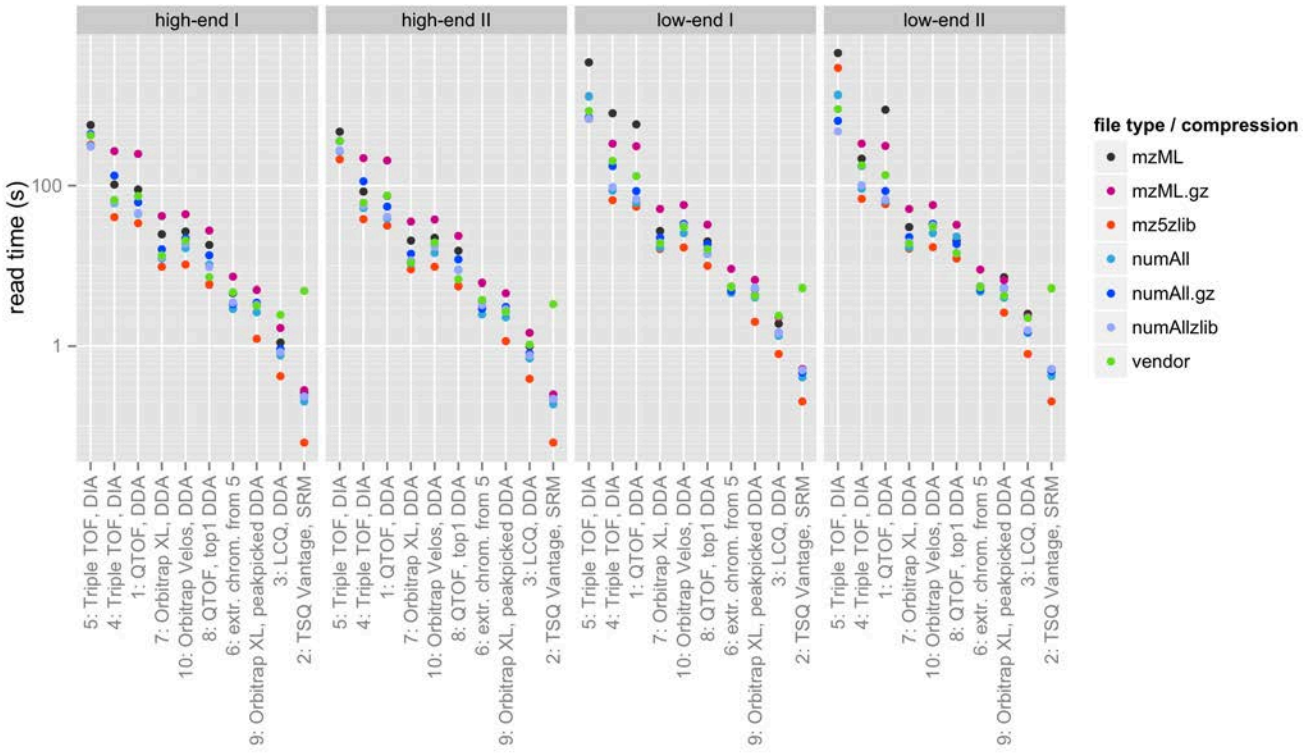
Supplementary Figure 4: Relative write time

Log_2 write times subtracted by mzML log_2 write time for all file formats, computers and files. Files are sorted in descending mzML-size order (same as in Fig 1c). Note that for us it is not possible to write in the vendor formats.



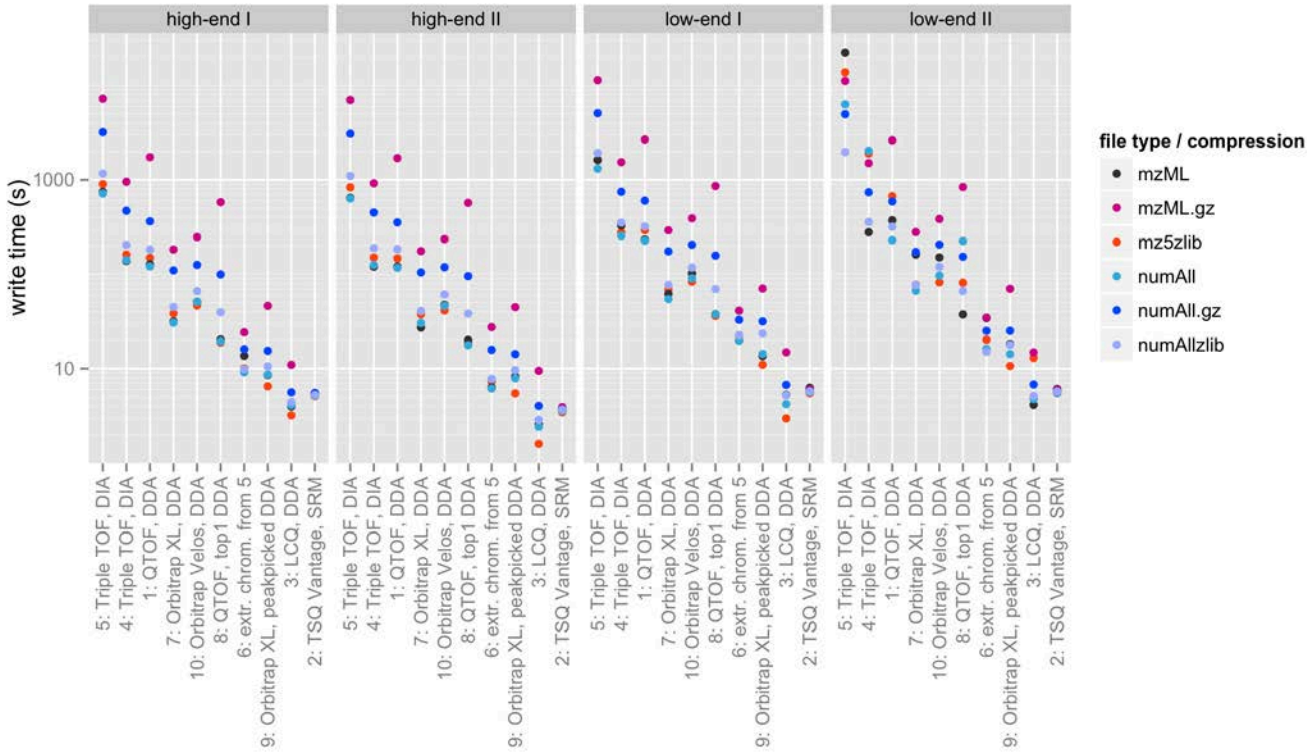
Supplementary Figure 5: Absolute read time

The read time in seconds for selected file formats and all files, spanning over 3 orders of magnitude. Files are sorted in descending mzML-size order (same as in Fig 1c).



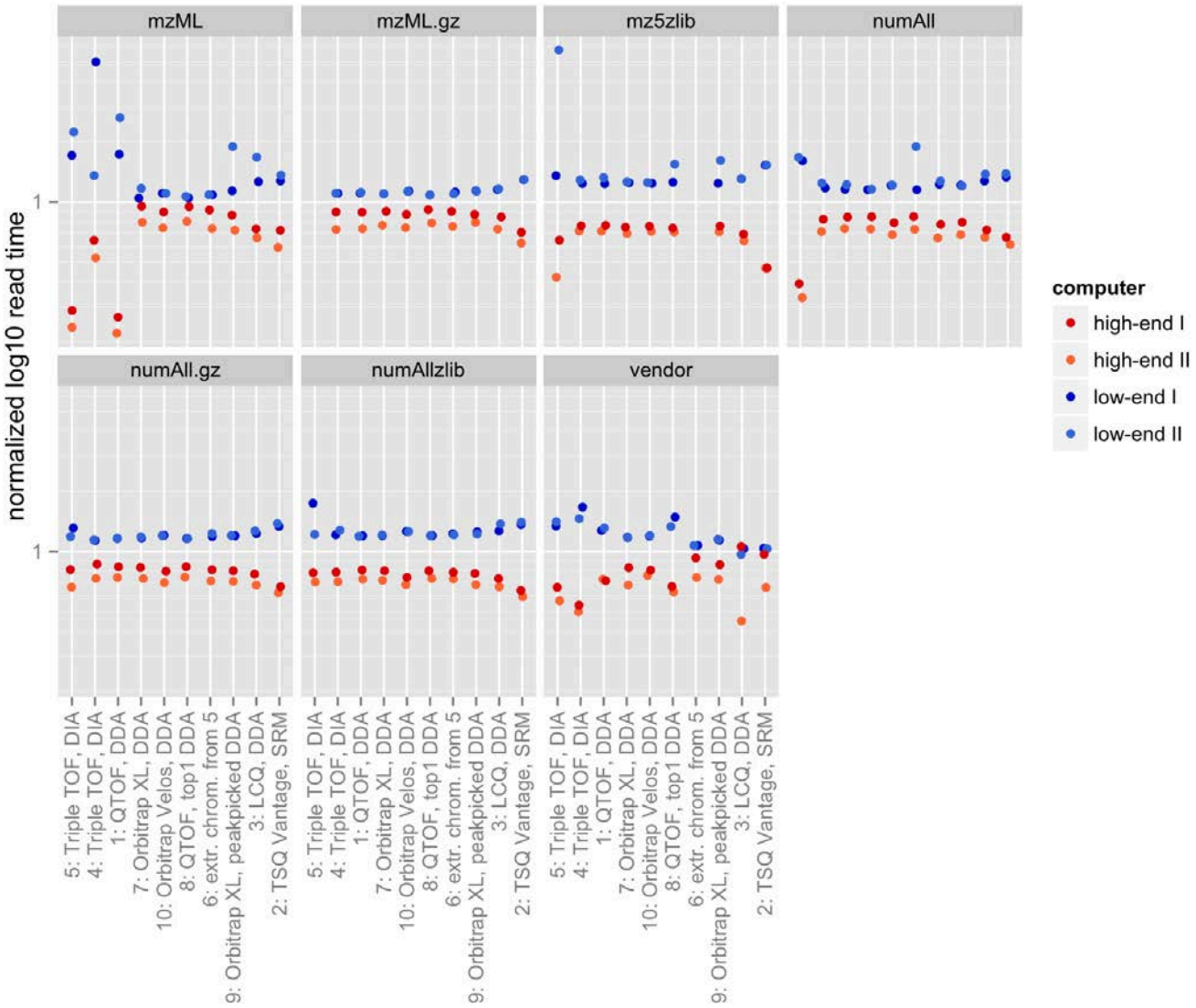
Supplementary Figure 6: Absolute write time

The write time in seconds for selected file formats and all files, spanning over 4 orders of magnitude. Files are sorted in descending mzML-size order (same as in Fig 1c).



Supplementary Figure 7: Read time for different computers

Comparison of read times on different computer hardware for different file formats. Files are sorted in descending mzML-size order (same as in Fig 1c). Read times were normalized by the median read time for each file and format.



Supplementary Figure 8: Write time for different computers

Comparison of write times on different computer hardware for different file formats. Files are sorted in descending mzML-size order (same as in Fig 1c). Write times were normalized by the median write time for each file and format.