BIOINFORMATICS ORIGINAL PAPER Vol. 25 no. 15 2009, pages 1937–1943
doi:10.1093/bioinformatics/btp294

Data and text mining

Highly accelerated feature detection in proteomics data sets using modern graphics processing units

Rene Hussong[1,*], Barbara Gregorius[1,2], Andreas Tholey[2] and Andreas Hildebrandt[1,*]

[1]Center for Bioinformatics and Computer Science Department, Saarland University, 66041 Saarbrücken and [2]Division Systematic Proteome Research, Institute for Experimental Medicine, Christian-Albrechts University, 24105 Kiel, Germany

Received on January 14, 2009; revised on April 9, 2009; accepted on April 27, 2009

Advance Access publication May 14, 2009

Associate Editor: John Quackenbush

ABSTRACT

Motivation: Mass spectrometry (MS) is one of the most important techniques for high-throughput analysis in proteomics research. Due to the large number of different proteins and their post-translationally modified variants, the amount of data generated by a single wet-lab MS experiment can easily exceed several gigabytes. Hence, the time necessary to analyze and interpret the measured data is often significantly larger than the time spent on sample preparation and the wet-lab experiment itself. Since the automated analysis of this data is hampered by noise and baseline artifacts, more sophisticated computational techniques are required to handle the recorded mass spectra. Obviously, there is a clear tradeoff between performance and quality of the analysis, which is currently one of the most challenging problems in computational proteomics.

Results: Using modern graphics processing units (GPUs), we implemented a feature finding algorithm based on a hand-tailored adaptive wavelet transform that drastically reduces the computation time. A further speedup can be achieved exploiting the multi-core architecture of current computing devices, which leads to up to an approximately 200-fold speedup in our computational experiments. In addition, we will demonstrate that several approximations necessary on the CPU to keep run times bearable, become obsolete on the GPU, yielding not only faster, but also improved results.

Availability: An open source implementation of the CUDA-based algorithm is available via the software framework OpenMS (http://www.openms.de).

Contact: rene@bioinf.uni-sb.de; anhi@bioinf.uni-sb.de

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 INTRODUCTION

Recent years have seen a surge of interest in proteomic data with applications ranging from basic pharmaceutical research over medical diagnostics and therapy to biotechnology and engineering (see, e.g. Diamandis, 2004; Jeffery and Bogyo, 2003; Ryan *et al.*, 2002). In contrast to the (rather static) genome, the proteome is highly dynamic. Experts estimate the number of human genes to about 30 000–40 000, while the number of possible proteins may lie between a few hundred thousands up to several millions, due to alternative splicing, post-translational modifications and different subunit assembly (Wikberg *et al.*, 2004). In order to determine the quantitative protein composition of some sample under study, mass spectrometry (MS) is often the method of choice. In a typical MS experimental setting, proteins are cut into smaller pieces by some digestive enzyme like trypsin and are guided into a chromatographic column that separates the peptides according to their physico-chemical properties. At discrete time points, the so-called *retention time rt*, peptides will leave the column in small groups, which enter a mass spectrometer in the following. Rather than measuring the *mass m* directly, one records the *mass over charge ratio m/z*. Finally, the experiment results in a 2D map $M(rt, m/z)$[1], which maps the retention time and the mass over charge ratio to the number of detected molecules, for short the *intensity i*.

To identify and quantify the proteins in the sample, regions of interest, the so called *features*, must be separated from parasitics like noise or baseline artifacts. Every subsequent step in the proteomic analysis pipeline depends necessarily on this basic feature identification stage, which is therefore crucial for the success of nearly every proteomic application.

Since real-world experiments can easily lead to several gigabytes of data, automated analysis is indispensable. Although recent years have seen a constant progress on the efficient and accurate analysis of high-throughput MS data (cf. Andreev *et al.*, 2003; Cox and Mann, 2008; Du *et al.*, 2006; Horn *et al.*, 1999; Leptos *et al.*, 2006, to mention just a few of them), the problem is still challenging. Due to the huge computational load, one is often driven to simpler, heuristic algorithms. Of course, this strategy can also spoil the quality of the analysis since low abundant peptides may be missed or many false positive regions of interest may be identified. Besides improvements on the software side, there have been attempts to accelerate the processing of MS scans by field programmable gate arrays (FPGAs) as presented in (Bogdan *et al.*, 2007).

Using the compute capabilities of modern graphics cards and an integral transform—the so-called *isotope wavelet transform* (Hussong *et al.*, 2007)—the work presented here aims at high-quality results that can be computed within minutes on today's standard

*To whom correspondence should be addressed.

[1]For the sake of simplicity, we will use the notations $M(rt, m/z) = M(rt, t) = M(s, t) = M(s, m/z)$ interchangeably, where t denotes the t-th data point in the s-th scan of M with retention time rt and mass-over-charge value m/z.

scientific workstations. Our approach is generic, in the sense that the wavelet kernel function can easily be exchanged. Thus, alternative wavelet methods (like those using the popular Mexican Hat wavelet, e.g. (Lange *et al*., 2006) and (Du *et al*., 2006)) can also benefit from this software framework.

The article is organized as follows: first, we will formally introduce the isotope wavelet as a theoretically justified method for feature detection in proteomic data sets and shortly recapitulate the most important advantages of the algorithm. Next, we will summarize the memory model of the compute unified device architecture (CUDA) as introduced by NVIDIA (NVIDIA, 2008) for modern graphics processing units (GPUs). We will present two different strategies to port the wavelet transform onto the GPU, speeding up the algorithm by at least an order of magnitude. The theoretical part of the article is followed by a detailed analysis of running times on real-world data sets, as well as a qualitative comparison of the CPU and the GPU results, which can differ especially in regions of low signal-to-noise (S/N) ratios. We will argue that porting the algorithm to the GPU can even increase the quality of the data, since the wavelet can be sampled exactly without spoiling the performance. This is in contrast to the original CPU implementation, which is based on several approximations to manage the high computational load. The article concludes with a discussion of the results and an outlook.

2 APPROACH

Signal processing in MS can be seen as a pipeline consisting of several subsequent steps, each of them removing a specific type of artifact from the recorded spectrum. A common combination of filters starts with a resampling of the data, since MS signals often feature an irregular spacing between neighboring m/z data points. This step seems so trivial that it is hardly ever mentioned, although it seems clear that a 'simple' resampling of the data might significantly shift the position of monoisotopic peaks, which have to be determined within an error range of several ppm or ppb, depending on the task and instrument in use. The same problem appears in the following steps that often consist of a baseline filter, like a white top hat (cf., e.g. Sauve and Speed, 2004), and finally some kind of smoothing operation to eliminate high-frequency noise. Usually, altering effects increase rapidly with the number of different filters applied on the data. Hence, it would be desirable to design a method that removes most noise components[2] as well as baseline artifacts, but needs no explicit resampling of the original data. The isotope wavelet as introduced in Hussong *et al*. (2007) fulfills most of these properties.

2.1 The isotope wavelet

As suggested by their name, wavelets can be considered as small wave-like functions. These functions are by construction robust against the most common parasitics present in any kind of experimental signal. With respect to the Fourier domain, wavelets are located between high-frequency noise and low-frequency baseline artifacts[3] and therefore automatically ignore 'unwanted' characteristics of the signal and concentrate on the frequency band in which the 'real' signal should appear. The design of a wavelet which in addition respects the specific characteristics of peptidic mass signals renders the application of additional filters to mass spectrometric signals unnecessary, as the wavelet is robust with respect to chemical noise artifacts due to its bandpass characteristics.

In general, the *continuous wavelet transform* $W_{\psi_{a,b}}$ (for short W_ψ) of a signal $s \in L^2(\mathbb{R})$ with respect to the wavelet ψ is given by

$$W_\psi[s](a,b) = \frac{1}{\sqrt{c_\psi |a|}} \int_{-\infty}^{\infty} s(x)\psi\left(\frac{x-b}{a}\right) \mathrm{d}x \tag{1}$$

$$c_\psi := 2\pi \int_{-\infty}^{\infty} \frac{\left|\hat{\psi}(\omega)\right|^2}{|\omega|} \mathrm{d}\omega \tag{2}$$

where $a \in \mathbb{R}\setminus\{0\}$ and $b \in \mathbb{R}$ (Louis *et al*., 1997). a and b are called the scale and the translation parameter, respectively, and $\hat{\psi}(\omega)$ denotes the Fourier transform of $\psi(x)$. Hence, a wavelet transform is just a series of convolutions $\psi_{a,b} *_b s$ depending on a. In the case of the isotope wavelet, we assign to every charge state z a separate wavelet scale a, i.e. every charge state z requires its own convolution with a dilated or compressed isotope wavelet, respectively. The translational parameter b can then be considered the m/z position within the mass spectrometric scan under study. Since the isotopic envelope follows approximately a Poisson distribution depending on the mass m, the convolution kernel must be adapted over the mass range. Hence, the isotope wavelet transform is not a wavelet transform in a strict mathematical sense, but rather a so-called *adaptive wavelet transform*, where the convolution kernel is slowly changing over the mass range. In the following, we will often write 'ψ_m' instead of 'ψ' to stress this specific mass dependency.

For every charge state z, we define the isotope wavelet ψ_m as

$$\psi_m(x) := H(x)\left(\phi_m(x) - \frac{1}{\xi}\int_0^{\xi} \phi_m(x)\,\mathrm{d}x\right)$$

$$\phi_m(x) := \sin\left(\frac{2\pi z x}{m_n}\right) \underbrace{\frac{e^{-\lambda(m)}\lambda(m)^{xz}}{\Gamma(1+xz)}}_{(*)} \tag{3}$$

Since $\Gamma(1+xz) = (xz)!$ for $xz \in \mathbb{N}$, the part of Equation (3) marked as $(*)$ can be interpreted as a continuous version of the discrete Poisson distribution. $H(x) := \theta_0(x)(1-\theta_\xi(x))$, where θ is the Heaviside step function and ξ denotes a cutoff parameter that ensures compact support and therefore guarantees that ψ_m is a wavelet for fixed m. λ is a linear function based on the coefficients of the *averagine* model introduced in Gay *et al*. (1999) and m_n depicts the isotopic peak distance, adopting values around the mass of a single neutron ($m_n \approx 1.008664$).[4] Note that the first maximum position p_1 (the first

[2]Today, there is no generally accepted universal model for noise in all kinds of MS experiments; progress has been made in particular situations (Du *et al*., 2008; Krutchinsky and Chait, 2002), but as of yet not all causes and distributions of noise in MS are fully understood.

[3]This property necessarily depends on a reasonably chosen scaling parameter [see Equation (1)].

[4]Slightly different values for m_n have been proposed in the literature see, e.g. Horn *et al*. (1999). For high-resolution data, as produced by modern orbitrap

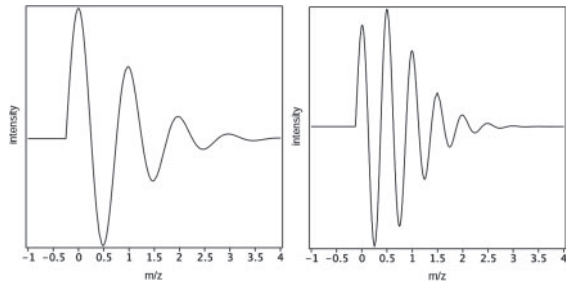

Fig. 1. The isotope wavelet at $m=1000$ Da and $z=1$ (on the left), as well as $m=2000$ Da and $z=2$ (on the right). The intensities of the 'hills' and 'valleys' follow the averagine peptide model.

'peak') of $\phi_m(x)$ according to Equation 3) is close to $x \approx 0.25 \cdot m_n/z$. For the sake of simplicity, we will assume that the wavelet is always shifted such that p_1 coincides with $x=0$. Figure 1 depicts the isotope wavelet for two different charge states and masses. For more information about the isotope wavelet, refer to Hussong *et al*. (2007).

2.2 Feature detection in MS scans

Having computed the wavelet transforms $W_{\psi_m}[s]$ of scan s for all wanted charge states $z \in 1, \ldots, z_{\max}$, every signal originating from a peptide will show a chirp-like structure as depicted in Figure 2. Since we know the locations where minima and maxima must occur if the underlying signal has indeed been triggered by a peptide, we can use the intensities at these positions of the transform for scoring a region of interest. If the score exceeds a user-defined threshold τ that is coupled to the average and the SD of the transform's intensity distribution, the position of the local maximum is returned as monoisotopic position with charge state z. Therefore, the algorithm directly identifies isotopic patterns. This is in contrast to common practice, where single peaks are identified first and afterwards an additional clustering step, the so-called *deconvolution*, has to be performed. Note that the algorithm for this 1D case has only a single parameter (the threshold τ) to be adjusted by the user.

2.3 CUDA

CUDA is a programming environment for leveraging the massive parallelization of graphics processing units (GPUs). CUDA extends the C language by additional commands that allow the programmer to implement computations directly on the GPU in a familiar C programming style. In contrast to the multi-purpose CPU, the GPU strongly focuses on data processing, i.e. it is a highly parallel, multi-threaded architecture tailored to compute-intensive integer and floating point operations that show a data parallel nature (NVIDIA, 2008). CUDA has its own memory model and therefore, data to be processed by the GPU must be loaded onto/from the device. In general, one can distinguish four different types of memory spaces: the global memory, as well as the shared memory, which are both read- and writable, and the constant as well as the texture memory that are both read-only. In principle, one can load the required data from the host onto the device's global memory and perform every necessary operation without using any other memory model.

machines e.g., the exact value of m_n might be an additional critical parameter that should be adopted to the individual machine and experimental setting.

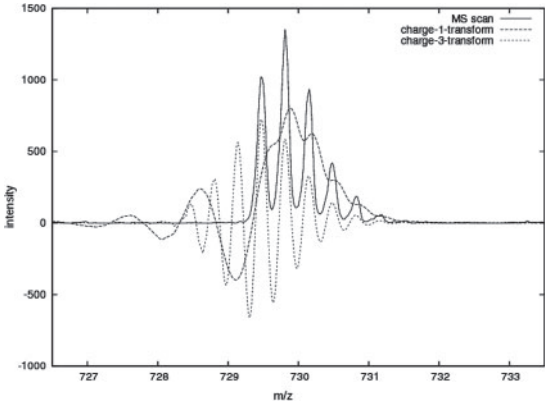

Fig. 2. Isotopic pattern of charge state 3 (solid line). Wavelet transforms W_{ψ_m} for $z \neq 3$ do not result in the typical chirp-like structure (for the sake of clarity only the charge-1- and the charge-3-transforms have been plotted).

However, texture memory is cached and shared memory is located directly on the chip, such that these memory models are much faster. In absence of so-called *bank conflicts*, accessing shared memory is even as fast as accessing a register. Consequently, if data points have to be accessed several times during a single device computation (as it is obviously the case if we convolve parts of a MS signal with the isotope wavelet as kernel), it usually pays off to load the data from global memory into the shared memory of the device (see Section 3.1).

For starting a computation on the graphics device, the user has to invoke a specific type of function, called the *kernel*, which is then executed n times in parallel on the GPU. The execution on the GPU is organized in the form of a *grid* that features a number of so-called *blocks* (B), which themselves contain a specific number of threads (T). All threads of the same block have read and write access to the same shared memory, whereas the global and the texture memory can—as the name already implies—be accessed from each thread across all thread-blocks. The Supplementary Material includes a figure providing a short overview of CUDA's hardware implementation. Since the complete description of the CUDA architecture is out of scope for this work, the interested reader is referred to (NVIDIA, 2008) for additional information.

3 METHODS

As the isotope wavelet transform is based on a simple convolution with a kernel function showing compact support on a comparably small interval of a MS scan, the algorithm is perfectly suited for parallelization. As the scoring function that has to be applied on the transformed data is a simple sum of intensity values at known positions, parallelizing the scoring function is also possible. We implemented two different parallel versions of the algorithm described in Section 2.1 that differ in their realization of the integral transform, while the implementation of the scoring function stays the same. The first version simply loads the spectrum into the cached texture memory which already leads to a drastic speedup. The second version, instead, loads the required data into shared memory, further accelerating the method, but notably complicating the algorithm. Since the implementation of the scoring function and the texture memory version of the wavelet transform are straightforward, we will omit a discussion and concentrate on the shared memory implementation.

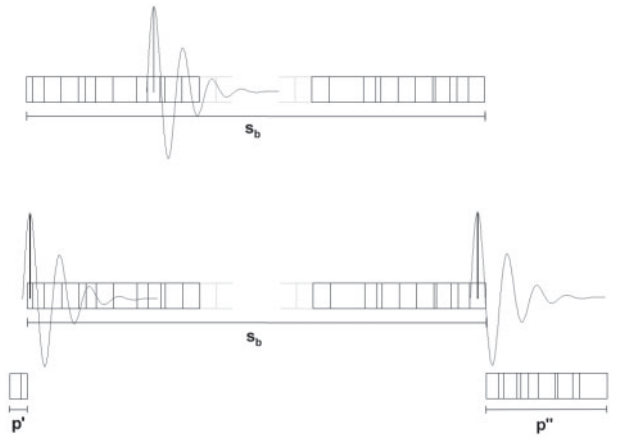

Fig. 3. Shared memory situation for computing the convolution. Each of the irregular-spaced boxes represents a data point of a MS scan. At the boundaries, additional point sets p' and p'' with $p := p' \cup p''$ must be loaded into the shared memory.

3.1 Parallelization with a shared memory model

The overall strategy is as follows: given a data map M, we compute the isotope wavelet transform $W_{\psi_m}[s]$ of every scan $s \in M$ for every $z \in 1, \ldots, z_{\max}$, by splitting s into equally sized parts s_b of length $|s_b| \leq 512$, which is the maximum number of allowed threads per block.[5] Since $|s|/|s_b|$ is usually not integral, we have to pad the signal s by some additional points p. Every part s_b is now assigned to a single block b and every thread $t \in b$ computes the convolution at the data point $W_{\psi_m}[s_b](t)$. As every convolution is based on the product of several intensities from a point's neighborhood, we usually need to load more than $|s_b|$ data points into shared memory (cf. Fig. 3). As apparent from Figure 1 the isotope wavelet is compactly supported, but $\psi_m(x) \neq 0$ also for $x < 0$. Hence, in order to compute the convolution $W_{\psi_m}[s]$ we need to 'extend' the region s_b by loading additional data points not only on the right, but also on the left end (cf. Fig. 3). Since MS data are mostly irregularly spaced, we cannot determine the necessary number of additional loads a priori. Thus, we implemented a 'worst case' strategy in that the signal s is traversed once before the wavelet transform and the minimal spacing $\Delta_{\min}$ between two neighboring points is determined. Knowing the extensions of the wavelet to the left $(\approx -0.25\, m_n/z)$ and to the right (depending on the maximum mass $m_{\max}$), we can reserve the maximum amount of memory necessary to compute all transforms on s. Unfortunately, shared memory per block is limited to 16 KB. Since we have to load a m/z value and an intensity value i for each data point, it might happen that due to the additional data points p, even a single wavelet would not fit into the available memory. Fortunately, we can detect these cases before the computation is started and hence automatically switch to the slower GPU implementation using texture instead of shared memory.

In order to balance the load between all threads of the same block, every thread loads the data point at which it computes the convolution. But, as there are usually much more data points to load than available threads, we require every thread t to load all points q such that $q = t + |s_b| \cdot j$, where $j \geq 0$ and integral. Pseudo-code for the described kernel function is given in Algorithm 1.

3.2 Multi-GPU architectures

By now, we used a single GPU to parallelize the isotope wavelet transform. If there are several CUDA compatible graphics cards available, we can expand the parallelization such that every device computes the transform(s) and scores of its own MS scan. Following this idea, we used Intel Threading Building Blocks (TBB) (Intel, 2005–2008) to combine the multi-core architecture of modern computers with high-performance CUDA devices. Although it would be possible to use any other kind of multi-threading software framework like Pthreads (http://www.opengroup.org/austin/papers/posix_faq.html) or OpenMP (http://www.openmp.org), e.g. the authors preferred TBB due to its simplicity and task-based structure, which is in clear contrast to the raw-thread programming paradigms of similar architectures. A parallel processing of several scans s_i of some data map M is straightforward: we split the data set into d equally sized parts, where d is the minimum out of the number of available CPU cores and the number of available GPU cards. A simple `parallel-for` statement can now be used to parallelize the processing of the map. Of course, the splitting of M requires a subsequent merging procedure to recombine the results of the different threads. Usually, the set of features affected by cutting the map into smaller pieces is comparably small such that the overhead caused by the parallelization is nearly negligible and the final gain in speed is approximately linear in d (see Section 4.2).

[5]The number of threads per block realizable in practice is limited by the shared memory and the number of registers in use.

Algorithm 1 GPU kernel function—isotope wavelet transform

Input: vector s_b and integers $|p'_{max}|$, $|p''_{max}|$, $|s|$, z
Output: $W_{\psi_m}[s_b]$, the transform of s_b w.r.t. z, for short w_b

```
__shared__ pB[], iB[]
/*block-shared memory for the positions and intensities*/
gp ← |p'_max| + |s_b| · blockID + threadID
/*the global position (in s) for the current thread*/
lp ← |p'_max| + threadID
/*the local position (in s_b) for the current thread*/
if threadID < |p'_max| then /*|p'_max| threads load p'*/
    pB[threadID] ⟵ s_b[gp − |p'_max|].getMZ()
    iB[threadID] ⟵ s_b[gp − |p'_max|].getIntensity()
end if
a = 0
while lp + a · |s_b| < |p'_max| + |s_b| + |p''_max| do /*load the rest*/
    pB[lp + a · |s_b|] ⟵ s_b[gp + a · |s_b|].getMZ()
    iB[lp + a · |s_b|] ⟵ s_b[gp + a · |s_b|].getIntensity()
    a ⟵ a + 1
end while
__syncthreads /*wait until all data is loaded*/
if gp − |p'_max| ≥ |s| then /*global position already in p''*/
    return
end if
v ⟵ s_b convolved with ψ_m at gp
w_b[gp].setIntensity(v)
```

4 RESULTS

Since the isotope wavelet transform as a method for feature finding in MS datasets has already been evaluated by peptide mass fingerprints (Hussong *et al*., 2007) and a myoglobin quantification study (Schulz-Trieglaff *et al*., 2007, 2008), this publication is aimed at the algorithmic approach and the speedups achievable through vectorization. At this point, we will not discuss the quality of the results and the merits of the isotope wavelet when compared with

alternative methods. Such a discussion and comparison can be found in the Supplementary Material, where we demonstrate that the GPU-based implementation of the isotope wavelet is as least as accurate as, but substantially faster than two popular, state-of-the-art feature detection schemes (Katajamaa *et al*., 2005, 2006).

Furthermore, we will demonstrate that even the quality of the transform can be increased on the GPU, as many approximations necessary on a CPU implementation can be replaced by exact computation of sine, exponential and reciprocal gamma functions [cf. Equation (3)].

4.1 Data sets, experimental and computational setup

In the following, we will base our evaluation on several datasets. The first one is an artificial mix consisting of eight peptides (see also, Lange *et al*. (2006)) with charge states one and two, encompassing about 45 MB of raw data (in mzData format). With the help of data set 1, we will demonstrate that the exact evaluation of Equation (3) is more robust with respect to artifacts such that the threshold parameter can be increased and potential false positive responses suppressed. For analyzing the performance of the algorithm, we will use several simulated electrospray MS data sets, a dilution series of a tryptic digest measured by MALDI-TOF MS and subsequent annotation by Mascot queries, as well as a real-world electrospray Q-TOF MS data set, containing peptides with sequences relevant for immuno-proteomics (MHC peptides) Schulz-Trieglaff *et al*. (2008). Here, we show the analysis of the immuno-proteomics data set, while all other results are attached as Supplementary Material.

The mass range of the MHC data set extends from 400 Th to 1000 Th, the average resolution per scan is about 0.03 Th, while the set contains peptides up to charge state four. To estimate speedup factors for the GPU-based implementation, we resampled each scan several times to contain 20 000 up to 100 000 measurement points. For each of the resulting sets a runtime analysis has been performed on a machine featuring 16 GB RAM and eight 2.3 GHz CPU cores. The machine contains two NVIDA Tesla C870 devices, each of them consisting of 128 computing kernels and 1.5 GB total memory. Table 1 lists the resampled data sets, their corresponding number of data points per scan and their raw data sizes in megabyte. The listing also summarizes the results of the computational experiments.

4.2 Discussion

Time measurements have been performed for four different types of implementation: an exact single CPU version, which evaluates Equation (3) with double precision (`CPU-E`), a single CPU-based implementation using several approximations and a fine-grained presampling on elementary functions where possible (`CPU-A`) and two different versions making use of a single (`GPU-1`) or two Tesla devices (`GPU-2`), respectively. Note that time measurements for `CPU-E` have just been listed for completeness, since the resultant computational load on the central processing unit is usually too costly for real-word scenarios.

At first glance, one notices a drastic speedup from a pure CPU-based implementation to the multi-GPU version using the benefits offered by shared memory. For the data set with highest resolution, the speedup factor from `CPU-A` to the high performance implementation even exceeds the 200-fold (cf. Fig. 4). Hence, even high-quality electrospray or MALDI MS data sets, e.g. can be processed within minutes using the isotope wavelet transform. In order to demonstrate that also users without access to the high-performance Tesla devices can benefit from 'standard scientific' NVIDIA graphics cards, often shipped with scientific workstations, we performed a second test series[6] using a single NVIDIA Quadro NVS 290. This card is of course significantly slower than the Tesla C870 device, but still in our experiments up to 10 times faster than the CPU. Corresponding speedup factors are also shown in Figure 4, where the test is listed as `GPU-X`.

By investigating the data depicted in Table 1 in more detail, we see that neither runtimes from `CPU-A` nor runtimes from the GPU-based implementations scale linearly with the number of involved data points. Admittedly, at first glance a quite linear effect is present for `GPU-1` and `GPU-2` up to 60 000 data points. Using Callgrind (Weidendorfer *et al*., 2004), a callgraph-based profiling tool, we identified significant differences for the CPU and GPU codings. Both CPU implementations spent most of their time in sampling the isotope wavelet at the irregular-spaced grid points. Hence, the speed of the—in principle—fast, $\mathcal{O}(n)$ computation of a single scale of the wavelet transform is limited by the computationally

Table 1. Runtime analysis

No. of points (per scan)	ca. size (MB)	CPU-E (exact)	CPU-A (approximating)	GPU-1 (exact)	GPU-2 (exact)
20 000	90	13 m 54 s	4 m 24 s	8 s	4 s
40 000	179	55 m 23 s	16 m 23 s	16 s	8 s
60 000	267	2 h 3 m 56 s	37 m 43 s	26 s	13 s
80 000	356	3 h 43 m 49 s	1 h 10 m 02 s	44 s	22 s
100 000	445	5 h 43 m 57 s	1 h 50 m 02 s	1 m 0 s	29 s

The table lists the resampled data sets originating from data set 2 (see text) with their corresponding m/z resolution per scan and their size in megabytes with respect to the mzData format. Runtimes for four different isotope wavelet implementations are given. These measurements exclude the time necessary to load the data file into memory and the time needed to write the results back to disk. Hence, real runtimes exceed the given time values by roughly 5–25 s, depending on the size of the data set. All experiments have been performed three times; time values have been averaged and rounded to seconds.

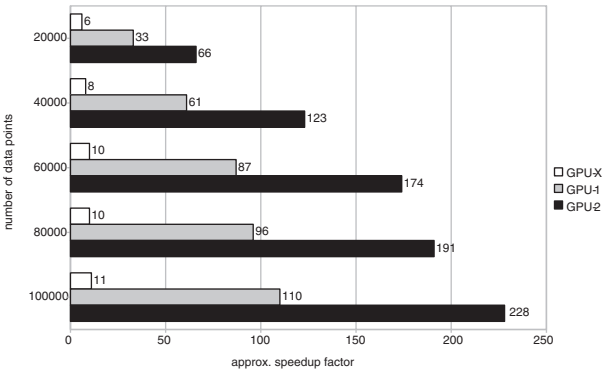

Fig. 4. Speedup factors (with respect to `CPU-A`) for `GPU-1` and `GPU-2`. In addition factors for `GPU-X`, a 'native' graphics card (see text), are reported.

[6]This analysis has been performed on a different machine: Sun Microsystems Ultra 24, 4 CPU cores, each 3.00 GHz, 4 GB RAM.

expensive adaptive kernel function. In the GPU case, however, the most time-determining part is the post-processing of potential seed regions. Besides the scoring, the post-processing also includes the so-called *sweep-line algorithm* as introduced in Schulz-Trieglaff *et al*. (2007), which can be considered a clustering of features over neighboring scans, such that potential peptide signals occurring in less than five scans, e.g. are rejected. It is obvious that for the most part of this algorithm a serial processing cannot be avoided. In addition to the relatively expensive post-processing procedure, also the pre-processing stage plays a significant role for the overall runtime. As spectra increase in size, more time is lost by determining the maximal extensions for p' and p'', as well as by memory allocation and related operations. Since the proportion of these pre- and post-processing procedures on the overall computation time increase stronger than the linear runtime behavior of the wavelet transform, the overall scaling is slightly worse than the theoretically possible one.

Porting the isotope wavelet transform onto the GPU results not only in a significant acceleration, it simultaneously increases the quality of the transform and therefore the receiver operating characteristics (ROC) of the results. We analyzed data set 1 with three different implementations: `CPU-E`, `CPU-A` and `GPU-1`. While both exact implementations (`CPU-E` and `GPU-1`) found all relevant peptide signals, the approximative implementation had difficulties in the identification of some low-intense peptides. An example is given in Figure 5. When setting the thresholding parameter τ to 5, `CPU-A` completely misses the feature, while the exact implementations correctly detect the signal. Hence, an approximative evaluation of Equation (3) can decrease the sensitivity of the method. Consequently, the user not only profits by the significant speed gain of the parallel GPU-based version, but additionally avoids artifacts from computational approximations.

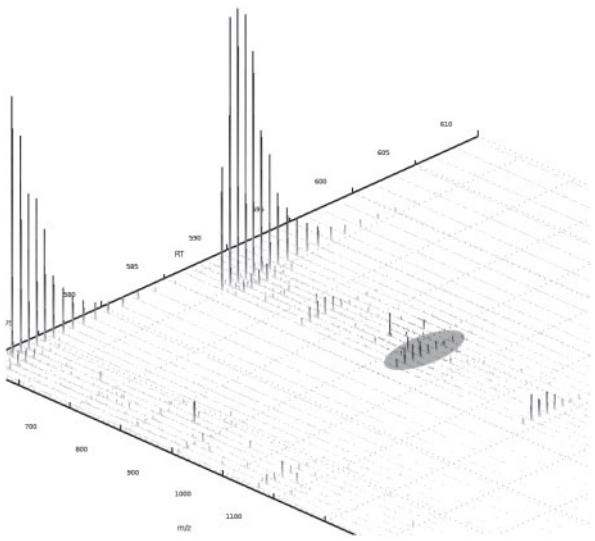

Fig. 5. Relatively low-intense peptide part of data set 1. Completely missed by `CPU-A`, but correctly detected by all exactly evaluated transforms. Image produced with the help of TOPPView (Kohlbacher *et al*., 2007).

5 OUTLOOK

As demonstrated in Hussong *et al*. (2007), the isotope wavelet transform is theoretically well founded and works well in a number of practical applications (Schulz-Trieglaff *et al*., 2007, 2008). The vectorization described here greatly improves its applicability and sometimes even the quality of the results. The method scales very well, and no significant improvements in parallelization are to be expected. Thus, we can hope that alternative feature detection techniques using integral transforms (e.g. wavelets) will adapt the proposed algorithm. In addition, the enormous speedups achieved allow us to consider a large number of future improvements for the wavelet technique used in this article. In the following, we will describe some of the ideas that can be addressed now that efficiency is no longer an issue (Sturm *et al*., 2008).

We designed the isotope wavelet as a very general framework for analyzing MS records independent from the instruments' idiosyncrasies like peak widths, peak tailing effects, etc. However, in many cases one has additional knowledge about the expected shape of the recorded signals that can help to ease the analysis a priori. Unfortunately, in order to design a matched filter [as seen in Andreev *et al*. (2003) for the chromatographic time domain], one has to know the underlying noise distribution, which usually significantly differs between types of machines (Du *et al*., 2008). Thus, the authors are currently developing a wavelet function that adapts automatically to the data at hand and can therefore fit much better to high-resolution data often featuring very narrow peaks.

Another problem also related to the specific characteristics of individual machines are shifts in the monoisotopic position caused by peak tailing effects, as frequently observed especially in time-of-flight (TOF) instruments. The mass deviation caused by this peak tailing can become critical, in particular for low-intensity signals; here additionally, mass signals are often smeared out due to poor ion statistics. The extend of the mass deviation is described in more detail in the analysis of the myoglobin dilution series, presented in the Supplementary Material. The adaptive kernel currently under development will significantly reduce or even completely eliminate this problem that is not unique to the isotope wavelet transform, but also present in many other methods like fitting procedures. Another hurdle to be taken is a suitable handling of non-isotope resolved patterns, where single peaks within the isotopic cluster can hardly be distinguished from each other. In the case that these peaks overlap completely, the convolution with the wavelet kernel yields zero, since the wavelet necessarily has a vanishing moment [cf. Equation (3)]. This problem is currently also addressed.

The most important shortcoming at the moment is the insufficient handling of overlapping peptide signals, which depending on the data often occur in real-world applications. In principle, wavelet techniques are intrinsically well suited for separating overlapping patterns [see, e.g. the algorithm presented in Wu *et al*. (2001) that is currently adapted to work with different kinds of kernel functions like the isotope wavelet]. In addition, the authors consider techniques as presented in Du and Angeletti (2006) to be combined with the transform, in that potential *seed* regions limit the set of required basis functions significantly.

Finally, the software interface has to be easily operable by non-computer scientists working in the wet-lab. The authors consider especially a simple-to-use and intuitive graphical user interface (GUI) as crucial for the progress in computational proteomics.

Consequently, a web server implementation of the algorithm is under development, which will complement the source code that is available via OpenMS (Sturm *et al*., 2008).

6 CONCLUSION

We presented a highly accelerated implementation of an adaptive wavelet transform that has been hand-tailored to identify potential signals in mass spectrometric data sets in proteomics. The algorithm used for parallelization is general enough that it can be easily adopted by other signal processing methods in computational proteomics. By porting the wavelet transform onto high-performance graphics cards using the CUDA programming environment, we obtained up to 200-fold acceleration on real world data sets. We demonstrated that even the quality of the results can be improved, since the massive parallelization power of modern GPUs enables an exact evaluation of the associated kernel function. Despite currently unsolved problems like a missing technique to separate overlapping signals, the work demonstrates that one of the most critical bottlenecks in high-throughput (computational) proteomics can be eased by high-end graphics devices. In the near future, additional steps in the proteomic pipeline that help to boost the quality of the computational analysis will be ported to CUDA to finally create a fast and easy-to-use software framework for MS.

ACKNOWLEDGEMENTS

The authors would like to express their thanks to Alan Kaatz, whose parallel sorting algorithm (Kaatz, 2007) has been used in a modified version; to Christian Huber (Analytical Chemistry, Salzburg, Austria), who provided some of the MS data sets. Many thanks to Dirk Neumann (Drug Transport, Saarland University, Germany) and Tom in der Rieden (Technical Coordinator of the Cluster of Excellence, Saarland University, Germany) for their support regarding the GPUs.

Funding: Clusters of Excellence 'Multimodal Computing and Interaction' (to R.H. and B.G.), 'Inflammation@Interfaces' (to A.T.) within the Excellence Initiative of the German Federal Government; DFG (grants BIZ4:1-4 to R.H. and A.H.); NVIDIA Professor Partnership Program.

Conflict of Interest: none declared.

REFERENCES

Andreev,V.P. *et al*. (2003) A universal denoising and peak picking algorithm for LC-MS based on matched filtration in the chromatographic time domain. *Anal. Chem.*, **75**, 6314–6326.

Bogdan,I. *et al*. (2007) Hardware acceleration of processing of mass spectrometric data for proteomics. *Bioinformatics*, **23**, 724–731.

Cox,J. and Mann,M. (2008) MaxQuant enables high peptide identification rates, individualized p.p.b.-range mass accuracies and proteome-wide protein quantification. *Nat. Biotechnol.*, **26**, 1367–1372.

Diamandis,E.P. (2004) Mass spectrometry as a diagnostic and a cancer biomarker discovery tool. *Mol. Cell. Proteomics*, **3**, 367–378.

Du,P. and Angeletti,R.H. (2006b) Automatic deconvolution of isotope-resolved mass spectra using variable selection and quantized peptide mass distribution. *Anal. Chem.*, **78**, 3385–3392.

Du,P. *et al*. (2006a) Improved peak detection in mass spectrum by incorporating continuous wavelet transform-based pattern matching. *Bioinformatics*, **22**, 2059–2065.

Du,P. *et al* (2008) A noise model for mass spectrometry based proteomics. *Bioinformatics*, **24**, 1070–1077.

Gay,S. *et al*. (1999) Modeling peptide mass fingerprinting data using the atomic composition of peptides. *Electrophoresis*, **20**, 3527–3534.

Horn,D.M. *et al*. (1999) Automated reduction and interpretation of high resolution electrospray mass spectra of large molecules. *J. Am. Soc. Mass Spectrom.*, **11**, 320–332.

Hussong,R. *et al*. (2007) Efficient analysis of mass spectrometry data using the isotope wavelet. In *COMPLIFE 2007: The Third International Symposium on Computational Life Science*, *AIP Proceedings*, vol. 940, American Institute of Physics, Utrecht, The Netherlands, pp. 139–149.

Intel Threading Building Blocks (TBB) Getting started guide (2005–2008) Available at http://www.threadingbuildingblocks.org/documentation.php. (last accessed date September 26, 2008)

Jeffery,D.A. and Bogyo,M. (2003) Chemical proteomics and its application to drug discovery. *Curr. Opin. Biotechnol.*, **14**, 87–95.

Kaatz,A. (2007) Parallel sorting code for floating points. Available at http://courses.ece.uiuc.edu/ece498/al1/HallOfFame.html (last accessed date October 17, 2008)

Katajamaa,M. *et al.* (2005) Processing methods for differential analysis of LC/MS profile data. *BMC Bioinformatics,* **6**, [doi:10.1186/1471-2105-6-179].

Katajamaa,M. *et al.* (2006) MZmine: toolbox for processing and visualization of mass spectrometry based molecular profile data. *Bioinformatics,* **22**, 634–636.

Kohlbacher,O. *et al*. (2007) TOPP – the OpenMS proteomics pipeline. *Bioinformatics*, **23**, e191–e197.

Krutchinsky,A.N. and Chait,B.T. (2002) On the nature of the chemical noise in MALDI mass spectra. *J. Am. Soc. Mass Spectrom.*, **13**, 129–134.

Lange,E. *et al*. (2006) High-accuracy peak picking of proteomics data using wavelet techniques. In *Pacific Symposium on Biocomputing*, World Scientific, The Big Island of Hawaii, pp. 243–254.

Leptos,K.C. *et al*. (2006) MapQuant: open-source software for large-scale protein quantification. *Proteomics*, **6**, 1770–1782.

Louis,A.K. *et al*. (1997) Wavelets: theory and applications. John Wiley & Sons, West Sussex, England.

NVIDIA CUDA Compute Unified Device Architecture (CUDA) Programming Guide, Version 2.0 (2008) Available at http://www.nvidia.com/object/cuda_develop.html (last accessed date June 24, 2008)

Ryan,T.E. and Patterson,S.D. (2002) Proteomics: drug target discovery on an industrial scale. *Trends Biotechnol.*, **20**, s45–s51.

Sauve,A.C. and Speed,T.P. (2004) Normalization, baseline correction and alignment of high-throughput mass spectrometry data. In *Proceedings of the Genomic Signal Processing and Statistics workshop*. Baltimore, MO, USA.

Schulz-Trieglaff,O. *et al*. (2007) A fast and accurate algorithm for the quantification of peptides from mass spectrometry data. In *Proceedings of the 11th Annual International Conference on Research in Computational Molecular Biology (RECOMB)*, Vol. 4453, *Lecture Notes in Bioinformatics (LNBI)*, Springer, Oakland, CA, USA, pp. 473–487.

Schulz-Trieglaff,O. *et al*. (2008) Computational quantification of peptides from LC-MS data. *J. Comput. Biol.*, **15**, 685–704.

Sturm,M. *et al*. (2008) OpenMS - an open-source software framework for mass spectrometry. *BMC Bioinformatics*, **9**, [doi:10.1186/1471-2105-9-163].

Weidendorfer,J. *et al*. (2004) A tool suite for simulation based analysis of memory access behavior. In *Proceedings of the 4th International Conference on Computational Science (ICCS)*, Vol. 3038 of *Lecture Notes in Computer Science (LNCS)*, Springer, Krakow, Poland, pp. 440–447.

Wikberg,J.E.S. *et al*. (2004) Proteochemometrics: a tool for modeling the molecular interaction space. In Kubinyi,H. and Müller,G. (eds) *Chemogenomics in Drug Discovery: A Medicinal Chemistry Perspective.* WILEY-VCH, Weinheim, Germany.

Wu,S. *et al*. (2001) Flip shift subtraction method: a new tool for separating the overlapping voltammetric peaks on the basis of finding the peak positions through the continuous wavelet transform. *J. Electroanal. Chem.*, **508**, 11–27.