Chapter 8
G-MSR: A GPU-Based Dimensionality Reduction Algorithm

Fahad Saeed and Muhammad Haseeb

In our previous chapters, we have introduced a generalized strategy has been devised that can be used for processing of MS-based omics data sets on a CPU-GPU architecture. Pre-processing of this data is an essential element for proteomics pipelines but the scalability of these pre-processing workflows has not been the focus of research in this domain. Hence many of the existing pipelines may take multiple hours or days to complete the processing [1].

The number of spectra produced by a single experiment can vary from few thousand to billions depending on the objective of the experiment, and species that are being considered. However, this is just for single run of experiments. The plethora of experimental spectra now available from different laboratories facilitates an enormous amount of data that can be used, reused, or reevaluated for systems biology researchers. Each spectrum consists of 2 columns of data where the first column consists of mass-to-charge ratio (m/z), and the second column consists of the corresponding intensities [2, 3]. In this chapter, we showcase how a generalized GPU-DAEMON strategy can be used for a noise reduction workflow for MS-based omics data using CPU-GPU architectures. Our strategy is called G-MSR and was first introduced in [3]. Note that this GPU-based noise reduction algorithm tries closely follows the processing patterns of MS-REDUCE algorithm [4].[1]

8.1 G-MSR Algorithm

Similar to the MS-REDUCE algorithm, G-MSR need to perform three steps: (1) Spectral classification, (2) Quantization, and (3) Weighted random sampling. The input parameters consist of a reduction factor R which is then applied to the given spectra s which will result in a spectrum that is equal to $R * |s|$. The classification stage allows the spectra to be classified into four classes. The spectra which are

[1] Some parts of this chapter may have appeared in [3].

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2022

F. Saeed and M. Haseeb, *High-Performance Algorithms for Mass Spectrometry-Based Omics*, Computational Biology 34,
https://doi.org/10.1007/978-3-031-01960-9_8

classified in the same class are quantized such that each spectrum is grouped based on the peaks that are significant. Then there is a weighted random sampling in used in each class such that peaks are randomly sampled from each quantum. The weighted sampling allows that the most significant peaks make it to the final reduced spectra.

The weighted random sampling step can be formulated as the following equation where x_i is the sampling weight for the ith quantum, q_i is the number of peaks in the quanta i, p' is the total peaks in the spectra, and n is the number of quanta.

$$\sum_{i=0}^{n} \frac{x_i}{100} = p' \tag{8.1}$$

Figure 8.2 shows the design of G-MSR overlapped on the GPU-DAEMON template and Fig. 8.1 shows a comparison in the workflows of MS-REDUCE and G-MSR.

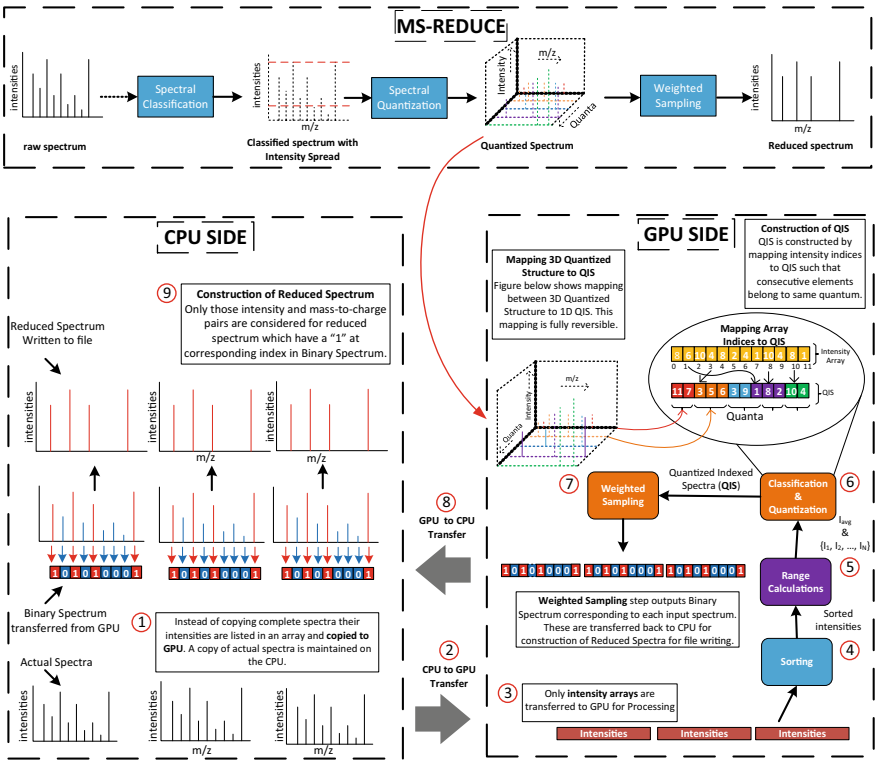

Fig. 8.1 **a** Shows the work flow of MS-REDUCE. **b** Construction of QIS from 3-D quantized spectrum from MS-REDUCE. **c** Work flow of G-MSR, blocks with same color represent processing in same kernel. A copy of actual spectra is maintained on the CPU for the construction of reduced spectra

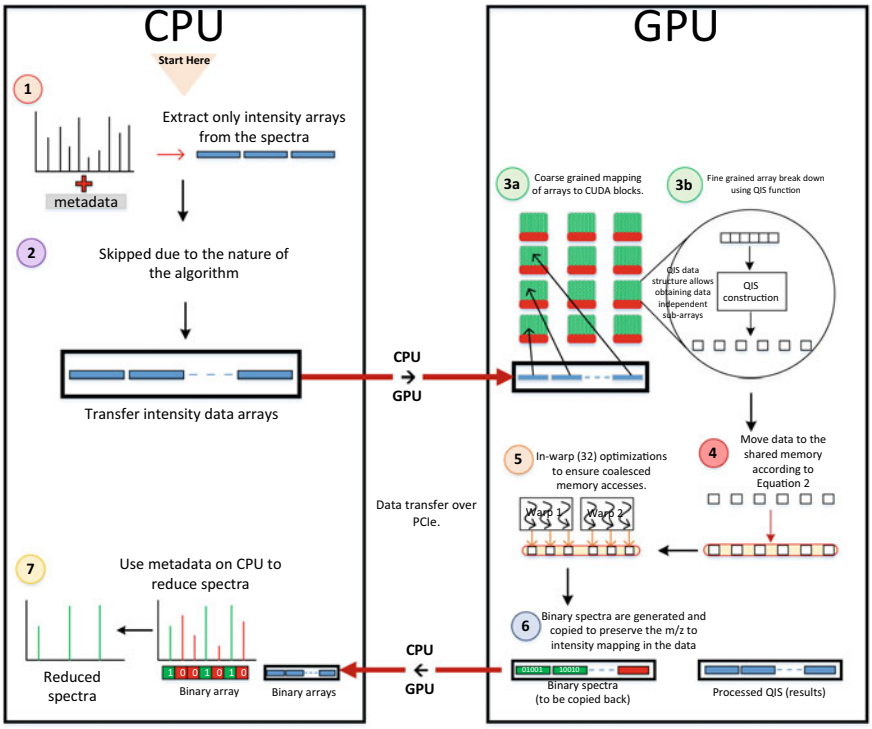

Fig. 8.2 Design of G-MSR overlaid on GPU-DAEMON template

8.1.1 Simplifying Complex Data Structures

As discussed before, the mass spectra obtained from MS consist of mass-to-charge ratios and their corresponding intensities. In a naive method complete spectra along with their meta-data would be transferred over the PCIe cable to GPU for processing. But following the GPU-DAEMON template we separate the intensities from the larger data structure in the forms of multiple arrays (one array for each spectrum) and only transfer these over to GPU memory. This cuts down the amount of data being transferred by more than 50%. The actual spectra are kept on the host for book-keeping and post-processing phase.

8.1.2 Simplifying Complex Computations

Since intensities are floating point numbers, we round them off to nearest integer before transferring them to GPU. This converts all the floating point computations

to integer computations thus simplifying the computations. As shown at the end of this chapter, this approximation does not affect the algorithm's performance.

8.1.3 Efficient Array Management

The quantization stage of MS-REDUCE reduction algorithm discussed in Chap. 3 transforms the spectra into 3-Dimensional data structures. Managing this 3-D data structure is challenging for data processing on a GPU architecture [5], also in-order for GPU-DAEMON's array management technique to work we need to map the data into a 1-Dimensional array. To achieve this, we introduced a novel data structure called Quantized Index Spectrum (QIS) which maps a 3-D quantized spectrum onto a 1-D array which can then be easily managed using the techniques discussed in Sect. 6.3.3. The QIS data structure serves a dual purpose of transforming 3-D quantized spectra to 1-D array while performing the step of quantization. As discussed before, the quantization step basically groups together the peaks of a spectrum. In a QIS data structure, these groups of peaks are present in contiguous memory locations, with a separate array of pointers keeping track of starting and ending points. Each of this group can be considered as a sub-array, since these sub-arrays are independent of each other we can use the strategy of Sect. 6.3.3 for exploiting fine-grained parallelism. For G-MSR algorithm, we replace F_{sub} by QIS construction in GPU-DAEMON template. In order to construct a QIS, instead of clustering peaks together as in MS-REDUCE, we clustered together with the indices of the peaks which make up a quantum. For each spectrum, a QIS is an array containing peak indices clustered together at computed distances. We refer to this structure as quantized-indexed-spectrum (QIS). We can formally define QIS for a spectrum s_i as: *Definition:* Q_i where $Q_i = \{q_1, q_2, q_3, \ldots, q_m\}$ and each $q_t = \{l_1, l_2, l_3, \ldots, l_n\}$ is quantum t, and l represents index for a peak in s_i. In QIS structure, quanta are sorted in their increasing order. Figure 8.1b shows the construction of QIS from intensity array. The QIS then overwrites the spectrum to conserve space.

8.1.4 Exploiting Shared Memory

To better exploit the shared memory, sub-arrays are then moved to the shared memory for further processing if the Eq. 6.2 is satisfied.

8.1.5 In-Warp Optimizations

The sub-arrays created by the QIS are a part of a larger array, with their beginning and end pointers listed separately. So, all the sub-arrays created by QIS are in contiguous

memory locations. This feature of QIS helps ensure that when consecutive sub-arrays are processed by consecutive threads of a warp, memory coalescing takes place.

8.1.6 Result Sifting

In the first step, rather than transferring complete spectra we transferred only the part which was needed for GPU-processing, and because of the random sampling which takes place in the third phase of dimensionality reduction algorithm [4], it becomes difficult to maintain which intensities are eliminated on the GPU-side. To tackle this problem, we used an additional property of QIS data structure, i.e. the indices of peaks which are eliminated on the GPU-side are retained with a place-holder. These place-holders help in constructing a binary spectrum indicating the indices of intensities to be retained in the reduced spectrum. We define Binary Spectra as *Definition:* Given a spectrum $s_i = \{p_1, p_2, p_3, \ldots, p_n\}$ a Binary Spectrum B_i for the corresponding reduced spectrum s_i' is defined as, $B_i = \{e_j = 1 | p_j \in s_i'\} \cup \{e_j = 0 | p_j \notin s_i'\}$. In other words, if a peak at index j in s_i is included in the reduced spectrum then there will be a 1 at index j of B_i; otherwise it will be zero. For each spectrum, a binary spectrum is generated and only these binary spectra are then copied back to CPU. Binary spectra are memory efficient and helpful in quick reconstruction of reduced spectra on the CPU side. Introduction of QIS and Binary Spectra thus enabled G-MSR to copy back just the bare minimum and resolve the GPU-CPU bottleneck.

8.1.7 Post Processing Results

The Binary Spectra copied back in the previous phase are then used for constructing the reduced spectra on the CPU side as shown in Fig. 8.2. Figure 8.3 shows the difference in the amount of data handled by MS-REDUCE and G-MSR.

8.2 Results and Experiments

8.2.1 Time Complexity Model

To compute the time complexity of G-MSR we replace $T(f_{sub}) = O(\frac{N}{B}) + O(\frac{N*n^2}{B*p}) + O(\frac{N*n}{B})$ and $T(f_{proc}) = O(\frac{s*N}{B})$ in Eq. 6.3. Here the f_{sub} time includes sorting, classification and construction of QIS data structure while the f_{proc} time consists of weighted random sampling phase. Replacing the values in Eq. 6.3 and simplifying leaves us with:

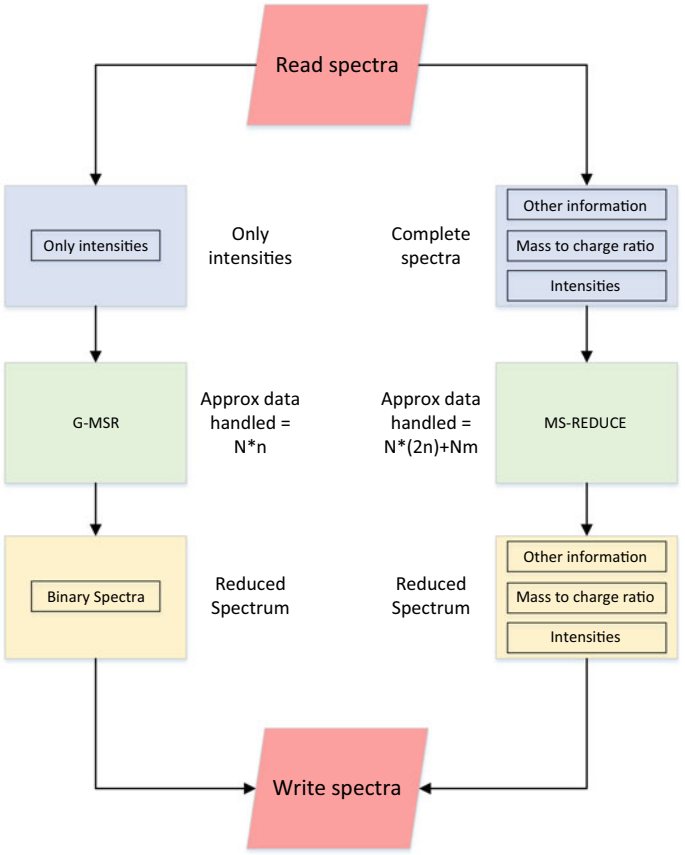

Fig. 8.3 Here N denotes the number of spectra, n the size of largest spectrum, and m the size of other information. Transferring only intensities to GPU for processing can conserve more than 50% of scarce in-core memory

$$O\left(\frac{N * (n^2 + l)}{B * p}\right) \tag{8.2}$$

where $l = p * (2 + n + n * s)$ and s is the sampling rate.

8.2.2 *Experiment Setup*

For all the experiments we made use of a Linux server running Ubuntu Operating System, version 14.01. The server houses two Intel Xeon E5-2620 Processors, clocked at 2.40 GHz with a total RAM of 48 GBs. The system has an NVIDIA Tesla

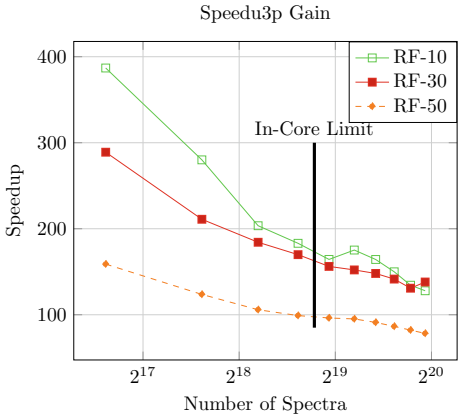

Fig. 8.4 Figure showing the speed up gained by G-MSR over MS-REDUCE while operating at reduction factor (RF) of 10, 30, and 50. The vertical line represents the point where in-core memory is filled

K-40c GPU with a total of 2880 CUDA Cores and 12 GBs of RAM. CUDA version 7.5 and GCC version 4.8.4 were used for compilation.

8.2.3 Scalability and Time Analysis

For this experiment, we used the appended UPS2 dataset which had over a million spectra. Timing experiments were performed with progressively increasing datasizes to cover the cases where data fits in the GPU's memory and when it doesn't. Our experiments showed peak speedups of 386, 288, and 158 for the three Reduction Factors (RF) of 10%, 30%, and 50%, respectively. In accordance with the Eq. 8.2 we get smaller speedup for larger RF and we observe a decrease in speedup with increasing number of spectra in Fig. 8.4. Also with higher RF, amount of data being processed is increased. This increased data leads to more memory being used per warp and thus minimizes the number of concurrent threads leading to increased execution time shown in Fig. 8.5.

8.2.4 Quality Assessment

We used the same method of quality assessment as shown in Fig. 5.6. For our experiments we set the FDR value of interest to 5%, i.e. any PSM having FDR value below 5% is an acceptable match, we call them effective matches. Figure 8.7 shows percentage of effective matches with varying reduction factors for both algorithms. G-MSR and MS-REDUCE gave almost same percentages of effective matches.

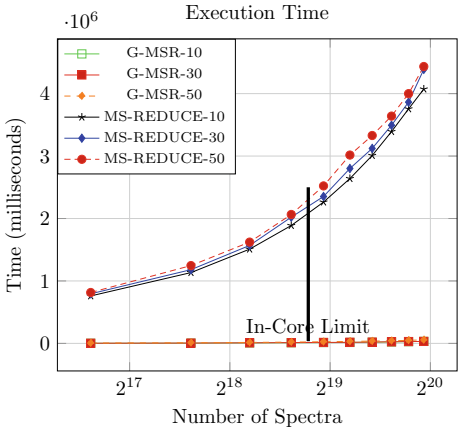

Fig. 8.5 Figure showing the execution times for G-MSR and MS-REDUCE for varying reduction factors. In the legend, numbers following the algorithm names are reduction factors. The vertical line represents the point where in-core memory is filled

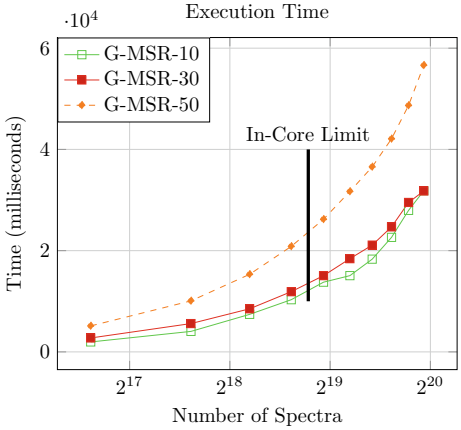

Fig. 8.6 Figure showing the execution times for G-MSR operating at reduction factors of 10, 30, and 50. The vertical line represents the point where in-core memory is filled

8.2.5 Reductive Proteomics for high-resolution instruments

In high-resolution proteomics, the x-axis resolution. i.e. number of bins can lead to large data processing times. Pre-processing of spectra with G-MSR and MS-REDUCE will reduce the dataset size and hence the processing times. We performed peptide deductions for UPS2 dataset after preprocessing it with G-MSR at different reduction factors (RF). We used Tide integrated with hiXcorr was used for peptide deduction in this experiment. Fig. 8.8 shows that the performance of peptide deduction becomes more scalable with smaller reduction factors even with increasing resolution.

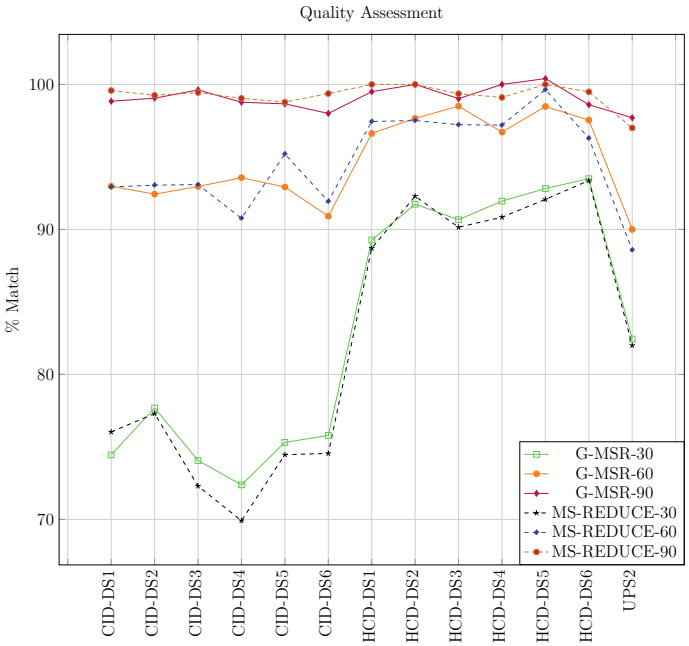

Fig. 8.7 A comparison of quality assessment plots of MS-REDUCE and G-MSR. In the legend, numerical value following name of the algorithm represents its reduction factor. X-axis contains the labels for the experimental datasets while Y-axis represents the percentage of peptide matches with FDR of greater than 5%

8.2.6 Comparison with Unified Memory

To assess the performance of G-MSR (a GPU-DAEMON-based version of MS-REDUCE algorithm), we compared it against a unified memory-based GPU implementation of MS-REDUCE. The unified memory technique enables quick and easy development of GPU-based algorithms. For our purpose, we simply took the sequential version of G-MSR [4] and modified the code following rules of GPU algorithm development using CUDA unified memory [6]. For scalability study, we appended the UPS2 dataset multiple times to get progressively larger datasets. Figures 8.9 and 8.11 shows that GPU-DAEMON based implementation consistently out-performs the naive implementation. It can be observed in Fig. 8.11 that CUDA unified-memory-based implementation reaches its in-core memory limit at only 14,000 spectra, while G-MSR as shown in Fig. 8.6 reaches its in-core memory limit at 400,000 spectra. Along with better speed, GPU-DAEMON helps conserve limited in-core memory so that more throughput can be achieved. Figure 8.10 shows that GPU-DAEMON version uses a very small amount of in-core memory in comparison to the unified-memory-based implementation.

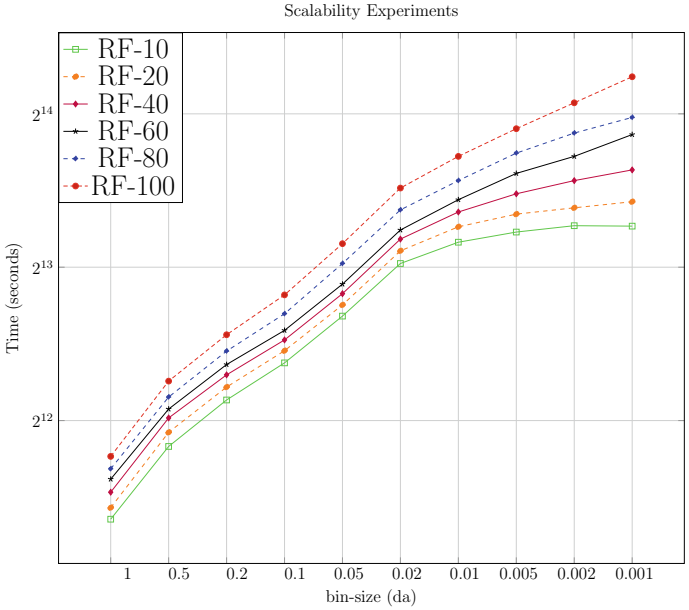

Fig. 8.8 Timing plots of peptide deduction process using Tide with hiXcorr algorithm. Here RF is the reduction factor. An increasing RF makes the process more scalable

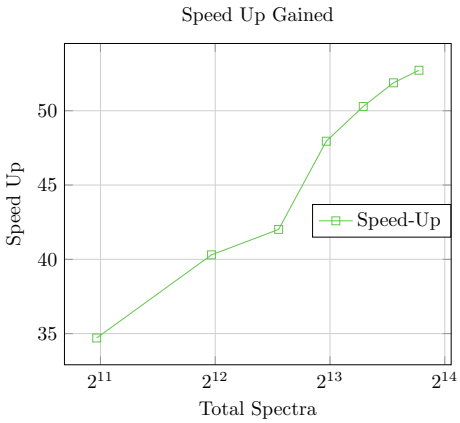

Fig. 8.9 Total speed up achieved by GPU-DAEMON implementation over CUDA unified memory-based implementation

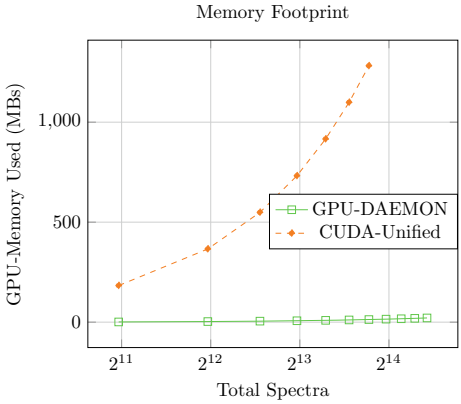

Fig. 8.10 Figure shows that GPU-DAEMON based implementation of MS-REDUCE uses only a fraction of memory as used by the CUDA unified memory implementation

Fig. 8.11 Figure shows that GPU-DAEMON-based implementation of MS-REDUCE scales better with increasing spectra. It should be noticed that CUDA unified memory-based version reaches in-core limit earlier and cannot process more than 14,000 spectra in a single pass while GPU-DAEMON implementation can process about 400,000 spectra before that limit is reached Fig. 8.6

References

1. Mujezinovic N, Schneider G, Wildpaner M, Mechtler K, Eisenhaber F (2010) Reducing the haystack to find the needle: improved protein identification after fast elimination of non-interpretable peptide ms/ms spectra and noise reduction. BMC Genomics 11(1):S13
2. Kong AT, Leprevost FV, Avtonomov DM, Mellacheruvu D, Nesvizhskii AI (2017) Msfragger: ultrafast and comprehensive peptide identification in mass spectrometry-based proteomics. Nature Methods 14(5):513–520
3. Awan MG, Saeed F (2017) An out-of-core GPU based dimensionality reduction algorithm for big mass spectrometry data and its application in bottom-up proteomics. In: Proceedings of the 8th ACM international conference on bioinformatics, computational biology, and health informatics. ACM, pp 550–555
4. Awan MG, Saeed F (2016) Ms-reduce: an ultrafast technique for reduction of big mass spectrometry data for high-throughput processing. Bioinformatics 32(10):1518–1526
5. Baskaran MM, Bordawekar R Optimizing sparse matrix-vector multiplication on GPUS using compile-time and run-time strategies, IBM Research Report, RC24704 (W0812-047)
6. Nvidia (2018) CUDA toolkit documentation. https://docs.nvidia.com/cuda/cuda-c-programming-guide/