Full Covariance Gaussian Mixture Models Evaluation on GPU

Jan Vaněk, Jan Trmal, Josef V. Psutka, Josef Psutka
Department of Cybernetics, University of West Bohemia
Univerzitni 8, 306 14 Plzen, Czech Republic
vanekyj@kky.zcu.cz, jtrmal@kky.zcu.cz psutka_j@kky.zcu.cz, psutka@kky.zcu.cz

***Abstract*— *Gaussian mixture models (GMMs) are often used in various data processing and classification tasks to model a continuous probability density in a multi-dimensional space. In cases, where the dimension of the feature space is relatively high (e.g. in the automatic speech recognition (ASR)), GMM with a higher number of Gaussians with diagonal covariances (DC) instead of full covariances (FC) is used from the two reasons. The first reason is a problem how to estimate robust FC matrices with a limited training data set. The second reason is a much higher computational cost during the GMM evaluation. The first reason was addressed in many recent publications. In contrast, this paper describes an efficient implementation on Graphic Processing Unit (GPU) of the FC-GMM evaluation, which addresses the second reason. The performance was tested on acoustic models for ASR, and it is shown that even a low-end laptop GPU is capable to evaluate a large acoustic model in a fraction of the real speech time. Three variants of the algorithm were implemented and compared on various GPUs: NVIDIA CUDA, NVIDIA OpenCL, and ATI/AMD OpenCL.*

***Keywords*— *Gaussian Mixture Models, Full Covariance, Automatic Speech Recognition, GPU, CUDA, OpenCL.*

I. Introduction

Using GPU as a coprocessor becomes quite popular in a few recent years. GPUs are well suited for highly parallel and arithmetically intensive tasks. The number of GPU-enabled applications and libraries grows fast. The reason is not only the GPU performance. Also the development of high-level programming languages and other useful tools play a key role. CUDA [1] is the most favorite development platform in the scientific community, but it is usable for NVIDIA GPUs only. An unrestricted open language standard - OpenCL - is more general [2]. It is now supported by NVIDIA and ATI/AMD GPUs. However, the last generation of Intel GPUs (Ivy Bridge) supports the OpenCL as well. It opens a nice scenario that all computers will support the GPU coprocessing in near future.

The evaluation of GMMs probabilities belongs to well parallelizable algorithms, especially in cases, where many GMMs and data can be processed in a parallel (or at least in a block-parallel) manner. One of these cases is the evaluation of the acoustic model in ASR. A few recent publications [3],[4],[5], and [6] describe the GPU implementation of the model evaluation. However, all the publications deal with DC-GMMs and NVIDIA GPUs only. Last year we published our NVIDIA GPU implementation of DC-GMMs [7]. Our implementation was significantly faster than previously mentioned ones. This year we added also ATI/AMD OpenCL variant and described the kernels in more detail in [8]. This paper is a continuation where the efficient implementation of the FC-GMMs evaluation is described in three variants: NVIDIA CUDA, NVIDIA OpenCL, and ATI/AMD OpenCL. The FC-GMMs evaluation on GPU was also mentioned in several recent papers but with a lack of the implementation details [9] or with only the moderate performance ([10] and [11]) that do not meet the ASR needs.

This paper is organized as follows: At the beginning, the estimation of full covariance matrices is introduced in Section II. Then, in Section III, the use of GPU as a coprocessor is generally described. The description of our implementation is presented in Section IV. A result Section V shows, compares, and discusses the performance of the implementation. Finally, Section VI concludes the paper.

II. Full covariance modeling for automatic speech recognition

For many years, acoustic models based on DC-GMMs represent the standard in the ASR domain. The feature extraction part in the ASR produces almost decorrelated feature vectors and that is the reason why diagonal covariances work fine. However, a part of a dependency between the dimensions still remains. A higher number of Gaussians can help to capture these dependencies, but not completely. In contrast, applying full covariance matrices increases the number of parameters to be estimated heavily. With a limited amount of training data, some of the covariance matrices may become ill-conditioned. The final acoustic model is not robust and its evaluation is numerically unstable. However, several methods were developed to make the estimated covariances more robust :

- **Smoothing, shrinkage** [12], [13] The absolute value of the off-diagonal elements is decreased via multiplication by positive constant lesser than one. The constant can be equal for the entire model or it can be set for each Gaussian/state individually. It may be predetermined or dependent on amount of training data gathered by individual Gaussian.
- **Covariance matrices sharing** [14] To reduce the number of estimated parameters, a smaller set of covariance matrices can be selected and shared between Gaussians.

This work was supported by the Technology Agency of the Czech Republic, project No. TE01020197 and by the European Regional Development Fund (ERDF), project "New Technologies for Information Society" (NTIS), European Centre of Excellence, ED1.1.00/02.0090.

The entire matrix or only the off-diagonal part can be shared.

- **Variable number of Gaussians** Instead of set of GMMs with uniform number of Gaussians, a variable number of Gaussians for individual GMM can be trained. The number of Gaussians is set to keep robustness of the matrices.

III. Using GPU as coprocessor

GPU is a very powerful processor but it is not designed for general computing. It is designed for data-parallel processing, where hundreds or thousands data-parts are processed independently in the parallel manner. GPU is equipped with many cores (therefore GPU is also called many-core processor). These cores are grouped into multiprocessors, which execute the same code with different data. It leads to programming model based on a double-hierarchical parallelization. In first coarse-grained level, entire data are split between larger blocks. These blocks are processed by single multiprocessor. In fine-grained level, the data-part (block) is handled by many threads in a single-instruction-multiple-data manner. Threads inside the block can share some limited amount of data and do local synchronization. The global synchronization is achieved at the end of the GPU program (kernel). Therefore, splitting the algorithm into more kernels may be necessary. Atomic instructions can be also used for synchronization in local or global domain. The programming model fit to NVIDIA and ATI/AMD GPUs (and other many-core architectures) and therefore programming languages CUDA and OpenCL are practically equivalent. They differ in terminology but the programming style is identical and transition from CUDA to OpenCL or vice versa is relatively easy. NVIDIA and AMD provide many very good guides, tutorials, documents, and examples [1], [15]. Therefore, broader knowledge about GPU computing can be easily gathered.

Good knowledge about programming model is necessary to implement an algorithm on GPU. But, making the program efficient on target architecture needs much more effort and experience. In addition, more code variants need to be prepared for individual GPU architectures or data kinds/sizes [16]. Usually, it is not enough to keep in mind only the recommendations from optimization guides provided by NVIDIA or AMD.

IV. An optimized kernel for likelihood calculation

Our implementation is developed for the evaluation of the acoustic model in the real-time ASR. However, the same optimization tips and tricks can be used also in other domains. The model consists of N tied-states, which are modeled as a GMM with M Gaussians. The evaluation of the model stands in evaluation of posterior probabilities of GMMs for given stream of the feature vectors $\boldsymbol{x_t}$ of dimensions D. The probabilities are computed according to

$$p(\boldsymbol{x_t}|\Theta_n) = \sum_{m=1}^{M} \frac{w_{nm}}{(2\pi)^{N/2}|\boldsymbol{C}_{nm}|^{1/2}} \exp\left(-\frac{1}{2}\left[(\boldsymbol{x_t}-\boldsymbol{\mu}_{nm})^T\boldsymbol{C}_{nm}^{-1}(\boldsymbol{x_t}-\boldsymbol{\mu}_{nm})\right]\right), \tag{1}$$

where $p(\boldsymbol{x_t}|\Theta_n)$ is the posterior probability of the feature vector $\boldsymbol{x_t}$ and given state n , w_{nm}, $\boldsymbol{\mu}_{nm}$, and $\boldsymbol{C}_{nm}$ are weight, mean vector, and covariance matrix of the Gaussian m in the sate n, respectively. The probability is computed in a logarithmic domain in order to bring a better numerical stability:

$$\log(p(\boldsymbol{x_t}|\Theta_n)) = \mathrm{LogSum}_{m=1}^{M}(K_{nm} - 1/2e(\boldsymbol{x_t}|\Theta_{nm})) \tag{2}$$

$$e(\boldsymbol{x_t}|\Theta_{nm}) = (\boldsymbol{x_t}-\boldsymbol{\mu}_{nm})^T\boldsymbol{C}_{nm}^{-1}(\boldsymbol{x_t}-\boldsymbol{\mu}_{nm}), \tag{3}$$

where $LogSum()$ is a function, which calculate the sum in a non-log domain while the input and the output is in the log-domain. K_{nm} is a pre-calculated constant from Gaussians parameters Θ_{nm} and it is a logarithm of the initial fraction from the Equation (1).

From the definition of the problem above, a large amount of parallelism is possible due to the independence of states and feature vectors. A partial independence of Gaussians and dimensions can be also utilized. Then multiple degrees of the parallelism offer high number of possible variants how to implement the algorithm to the target architecture. However, there are also some constraints and required data sizes, which need to be met. In the ASR, the typical sizes are thousands of states, tens or hundreds of Gaussians per state, and the dimension about 40. The number of feature vectors is also high, but the decoder in the ASR usually goes through the time-sequence successively. In tasks of real-time ASR, the number of the available feature vectors is very limited, because the larger feature vectors buffer increases the ASR delay.

Although, we have done three variants of the implementation, which share a common ground. The entire evaluation is handled by a single kernel. The work is split into blocks of 8 feature vectors and 64 or 128 of states. Threads are mapped to the states and each thread computes final probabilities for 8 feature vectors in the block. All input data are arranged in the memory to the order in which they are read by the threads. It ensures a fast coalesced memory access. In addition, a texture memory is used for all the read-only data. The dimension of feature vectors and model parameters is extended to be dividable by 4 and the additional dimensions are padded with zeros. It allows to read and to store all the data as a float4 data type. The loops inside the kernel can be partially unrolled by factor of 4 also. The kernel pseudo-code is in Algorithm 1. It consist of three loops. The main loop is for the Gaussians and the inner loops are going through the inverse covariance matrix. In fact, it goes only through the triangular matrix because of the covariance symmetry. The off-diagonal elements are doubled and only the triangular part of the matrix is stored in order to reduce memory size and bandwidth requirements. The $addLog()$ is implemented as $max(x,y) + log(1 + exp(min(x,y) - max(x,y)))$.

```
set 8 likelihood registers to "log-zero"
for each Gaussian do
    set 8 accumulators to zero
    compute address into model texture memory
    for i, all dimensions, 4 at once do
        fetch float4 mean(i)
        for $j \leq i$, 4 at once do
            if $i \neq j$, fetch float4 mean(j)
            fetch 4xfloat4 block of inverse covariances
            icov(i,j)
            compute updates for all 8 accumulators with
            4x4 submatrix
        end
    end
    fetch $K$ constants for actual Gaussians (a float per
    thread)
    add $K$ const. to all 8 accumulators and do addLog()
end
store final likelihoods
```

Algorithm 1: Kernel pseudo-code for FC-GMM likelihoods computation

A. NVIDIA variants

NVIDIA CUDA and NVIDIA OpenCL variants are equivalent. The kernel is the same, only the programming languages are different. In the CUDA variant, 1D textures are used for a slightly better performance. 2D textures need to be used in the OpenCL because large 1D textures are not supported. It is suitable to use the NVIDIA fast local memory. We use it to store the entire set of 8 feature vectors. The data are preloaded on the beginning of the kernel and than shared between all the threads. All threads access the same place in the local memory. Therefore, no bank conflict arises. All other variables (likelihoods, accumulators, temporary means and inverse covariances) are in the register memory. To reduce the number of used registers, the computation of 4x4 inverse covariances (icov) submatrix accumulators is split to 4 parts and the float4 icov variable is reused. 64 states/threads in the block was the optimal value for tested NVIDIA GPUs.

B. ATI/AMD variant

ATI/AMD GPUs differ from NVIDIA GPUs in the architecture, mainly in following features:

- Entire core has the same clocks (NVIDIA has higher clocks for the computing core, excluding new Kepler-based GPUs).
- Each stream core is equipped with five or four stream processors. This is why ATI/AMD GPUs have a higher raw computational performance than comparable NVIDIA GPUs.
- On chip local memory (32kB) is relatively slow in comparison with the register (note that HD 4000 and older have no local memory).
- Relatively large (256kB) register space is available.
- Best performance is obtained when using vectorized data types (as float4).

We use no local memory, because of it's limited bandwidth. The feature vectors are read through the texture cache and stored in float4 registers. All the needed data for new i, j coordinates are fetched at once at the beginning of the most inner loop. ATI/AMD performance is sensitive to interposing branches or memory fetches into computing block of code. Therefore, it is advisable to make the memory-related and the computing blocks of the code as large as possible. Also indexes needed to memory operations should be computed in advance because they may violate the memory-related block. ATI/AMD variant uses 128 threads/states in the block.

C. Variable number of Gaussians

If the number of Gaussians per state is constant, the algorithm and data preparation becomes simpler. However, only minor changes need to be done to support the variable number of Gaussians. Because 64-states (or 128-states) are computed together in the independent blocks, a constant number of Gaussians per state has to be ensured within individual blocks only. Therefore, states of the acoustic model are sorted in advance according to the number of Gaussians per state and divided into 64-state blocks. Usually, some virtual Gaussians need to be added, but the total number of virtual Gaussians is negligible. Only an additional memory-offset vector needs to be passed into the kernel because 64-states data-blocks vary in size size.

V. RESULTS

The performance of the GPU implementation is often presented as a speed-up of baseline CPU variant. But the speed-up value depends on both implementations. It is always simpler to use a slower baseline than really speed-up the GPU variant. In [17], a group of researchers from Intel optimized 14 popular algorithms for both platforms (Core i7 960 CPU and NVIDIA GTX 280 GPU) and they found out that the average speed-up was 2.5x only. Note that not all of the tested algorithms fit to the GPU architecture well. We use rather some absolute metrics to present the implementation performance.

We make a benchmark, which evaluates a FC-GMM based acoustic model. We also chose two metrics. The first one is an inverted real-time-factor (1/RTF), which compares the total length of processed speech with the total elapsed time including CPU-GPU memory transfers. It means that if we process a minute of speech during one second we get 1/RTF

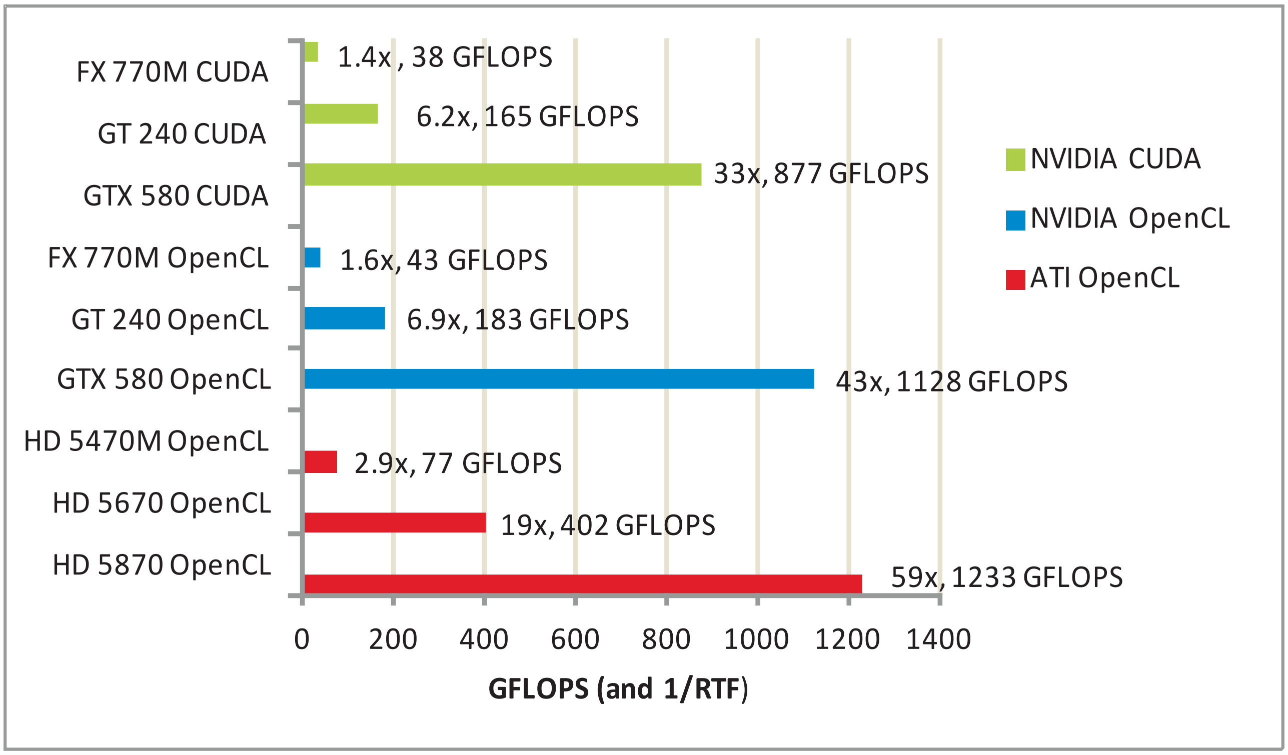

Fig. 1. *Performance result achieved with six tested GPUs. GFLOPS and inverted real-time factors (1/RTFs) of our three implementation variants. Tested on model with 80k FC-Gaussians in total, 36 dimensions. 1/RTF calculation is based on 100 vectors per second rate. GPU elapsed time includes CPU-GPU memory transfers.*

60x (with respect to 100 feature vectors per second). The second performance metrics is GFLOPS (billions of floating-point operations per second). The total number of needed floating point operations in the task is a product of the number of the feature vectors, the total number of the Gaussians, and the number of flops needed to evaluate one Gaussian and one vector, which we defined as 5 flops for each of $D(D-1)/2$ off-diagonal (triangular-half) covariance elements, plus 4 flops for diagonal D elements, and 9 flops for addLog function. Note that the potential virtual Gaussians are not included into the total flops count.

We tested three GPUs from each of both vendors:

- NVIDIA GTX 580 (512cores, 1,560 MHz)
- NVIDIA GT 240 (96 cores, 1,340 MHz)
- NVIDIA FX 770M laptop (32 cores, 1,250 MHz)
- ATI Radeon HD 5870 (1600 cores, 850 MHz)
- ATI Radeon HD 5670 (400 cores, 775 MHz)
- ATI Mobility Radeon HD 5470 laptop (80 cores, 750 MHz)

We used Windows 7 64bit with CUDA 4.0, NVIDIA 285.86 and ATI Catalyst 11.6 drivers.

We tested our implementation on an acoustic model with 80k Gaussians in total (5k tied states, 16 Gaussians each), dimension of feature vectors was 36. The total GPU elapsed time included CPU-GPU memory transfers of data and results. The initialization of GPU and the initial upload of model parameters were not included. GPU kernel was called for 256 feature long blocks of vectors. The achieved results are given in Fig. 1. The 1/RTF are in a range from 1.4x to 59x. It means up to 59-times faster than the real processed speech length. Even the slowest laptop GPU meets the real-time constraint. The ATI/AMD GPU architecture fits well to this algorithmically intensive task and the performance of ATI GPUs is better than the performance of the comparable NVIDIA GPUs. The difference between the NVIDIA CUDA and the OpenCL variant is also interesting. Although the kernels are identical the performance of OpenCL variant is significantly better. In our previous paper [7], we compared the NVIDIA CUDA and the OpenCL variant on diagonal covariance GMMs and there was no difference between them. We also tried cross-tests: NVIDIA OpenCL variant running on ATI hardware and vice versa. A significant performance drop between 15% and 50% was observed on the cross-tests.

A. ASR experiments

We have done also some preliminary results with our ASR system. In this case, the acoustic model evaluation is off–loaded to GPU and CPU does the decoding part. Double data/result buffering is used and GPU is called asynchronously. GPU evaluates the entire model - all the states, not only the active ones. The overhead of the active states selection is bigger than using our fast implementation to compute all the states. This experiments are not focused on the speed of the GMMs evaluation. Since the CPU decoder part is slower than GPU part, the GPU performance is hidden. We would like to show that the full covariance models can outperform the diagonal ones in word error rate (WER) of the entire ASR system without increasing the elapsed time.

For the training of acoustic models, a Czech read-speech database consisting of the speech of 800 speakers (384 males and 416 females) was used. The total length of training data was 220 hours. The front-end consisted of PLP features with delta and delta-delta coefficients followed by a cepstral mean normalization. The total dimension of feature vectors was 36. Am initial diagonal single Gaussian acoustic model was trained. It consisted of 4,922 tied-states. On the basis of the initial model, two models were trained: A diagonal model with 120k Gaussians, 24 Gaussians per each state, excluding non-speech states that had 128 Gaussians. A higher number of Gaussians did not reduce WER further, 24 Gaussians was the optimal number. The second - full covariance model - used the variable number of Gaussians and had 45k Gaussians in total. The training methodology is still under development and it is going to be published soon, this results are only preliminary.

The test consisted of 2 hours of speech from 12 speakers (6 males, 6 females). A trigram language model with 140k of words was enriched with all words from testing utterances to have no OOV (out of vocabulary) word. The diagonal model achieved WER 7.92%. The use of full covariance model reduced the WER to 6.65% without increasing the elapsed time. Also the CPU memory requirements stayed the same, since the model was off-loaded to the GPU memory, where 400 MB was occupied.

VI. Conclusions

In this paper we presented the GPU implementation of the FC-GMMs evaluation algorithm. We described and compared three variants: NVIDIA CUDA, NVIDIA OpenCL, and ATI/AMD OpenCL with six GPUs, three from each vendor. The results show that the implementation is close to the peak of the GPU performance and even the slow laptop GPUs meet real-time constraints. We have shown that the full covariance model can outperform a standard diagonal model. A 14% relative WER reduction was achieved in our preliminary ASR experiment. In addition, there is no increase of elapsed time of ASR.

References

[1] NVIDIA corp., "CUDA: NVIDIAs parallel computing architecture", http://www.nvidia.co.uk/object/what_is_cuda_new_uk.html.

[2] Khronos Group Std., "The OpenCL specification, version 1.1", http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf

[3] Cardinal P., Dumouchel P., Boulianne G., Comeau M., "GPU Accelerated Acoustic Likelihood Computations", in Proc. of INTERSPEECH 2008, pp. 964-967, Brisbane, Australia.

[4] Kveton P., Novak M., "Accelerating hierarchical acoustic likelihood computation on graphics processors", In Proc. of INTERSPEECH 2010, pp. 350-353. Makuhari, Japan.

[5] Dixon P. R., Oonishi T., Furui S, "Harnessing graphics processors for the fast computation of acoustic likelihoods in speech recognition", Computer, Speech and Language, Vol. 23, pp. 510-526. 2009.

[6] Chong J., Gonina E., Yi Y., Keutzer K., "A fully data parallel WFST-based large vocabulary continuous speech recognition on a graphics processing unit", In Proc. of INTERSPEECH 2009, pp. 1183-1186. Brighton, United Kingdom.

[7] Vanek J., Trmal J., Psutka J.V., "Optimization of the Gaussian Mixture Model Evaluation on GPU", In Proc. of INTERSPEECH 2011, pp. 1748-1751, Firenze, Italy.

[8] Vanek J., Trmal J., Psutka J.V., Psutka J., "Optimized Acoustic Likelihoods Computation for NVIDIA and ATI/AMD Graphics Processors", IEEE Trans. Speech and Audio Proc., 20(6), 2012.

[9] Cook H., Gonina E., Kamil S., Friedland G., Patterson D., Fox A., "CUDA-level performance with python-level productivity for Gaussian mixture model applications", In Proc. of the 3rd USENIX conference on Hot topic in parallelism (HotPar'11). USENIX Association, Berkeley, CA, USA.

[10] Kumar N., Satoor S., Buck I., "Fast parallel Expectation Maximization for Gaussian Mixture Models on GPUs using CUDA", 11th IEEE International Conference on High Performance Computing and Communications, 2009.

[11] Pangborn A. D., "Scalable data clustering using GPUs", Masters thesis, Rochester Institute of Technology, 2010.

[12] Diehl, F., Gales, M.J.F., Liu, X., Tomalin, M., Woodland, P.C., "Word Boundary Modelling and Full Covariance Gaussians for Arabic Speech-to-Text Systems", In Proc. INTERSPEECH 2011, p.p. 777-780.

[13] Bell, P., King, S., "A Shrinkage Estimator for Speech Recognition with Full Covariance HMMs", In Proc. Interspeech 2008, Brisbane, Australia.

[14] Gales, M.J.F., "Semi-Tied Covariance Matrices for Hidden Markov Models", IEEE Trans. Speech and Audio Proc.,7(3), May 1999.

[15] AMD Company, "AMD Accelerated Parallel Processing, OpenCL Programming Guide", http://developer.amd.com/gpu.

[16] Gonina E., Friedland G., Cook H., Keutzer K., "Fast Speaker Diarization Using a High-Level Scripting Language", In Proc. of ASRU Workshop, 2011.

[17] Lee V. et al., "Debunking the 100X GPU vs. CPU Myth: An Evaluation of Throughput Computing on CPU and GPU", In Proc. of ISCA 2010, Saint-Malo, France.