Accelerating Non-negative Matrix Factorization for Audio Source Separation using OpenMP and CUDA

Eric Battenberg ericb@eecs.berkeley.edu

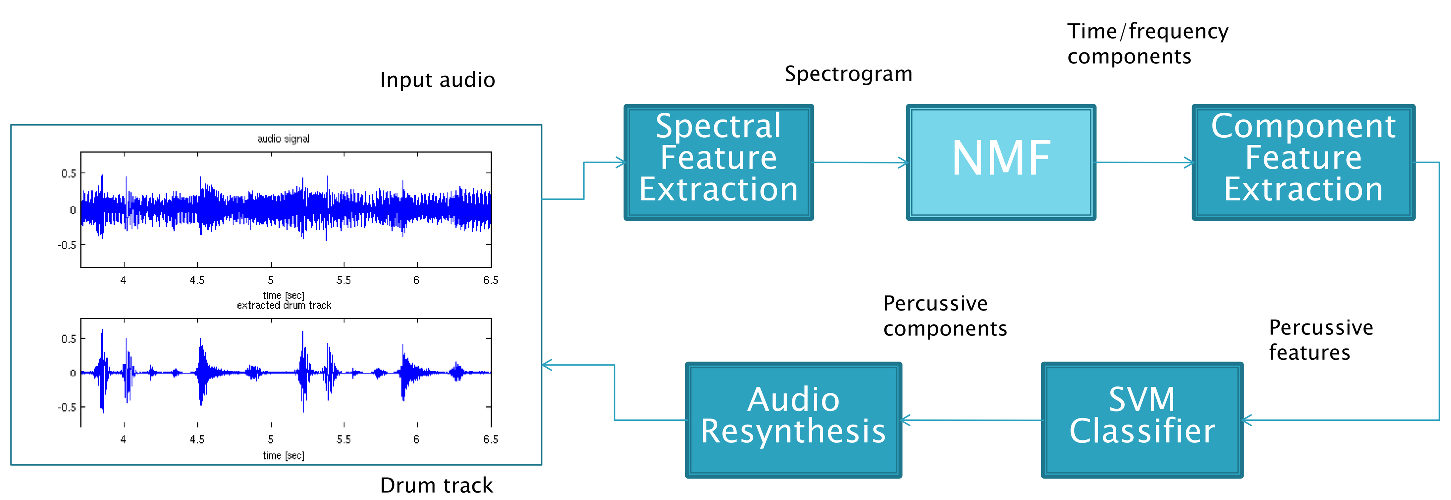
The Application

•Audio source separation is an important part of Music Information Retrieval

•Drum track extraction is a specific example of source separation and is useful in rhythm summarization, drum transcription, and beat tracking.

•We use Non-negative Matrix Factorization (NMF) as the source separation technique.

•The process:



•For a 512x3500 spectrogram representing 20 seconds of audio and 30-source NMF:

•NMF takes 80% of the compute time (18.5 of 23.1 sec) in the Matlab implementation.

•We will parallelize NMF using OpenMP for multi-core CPUs and CUDA for Nvidia GPUs.

Non-Negative Matrix Factorization

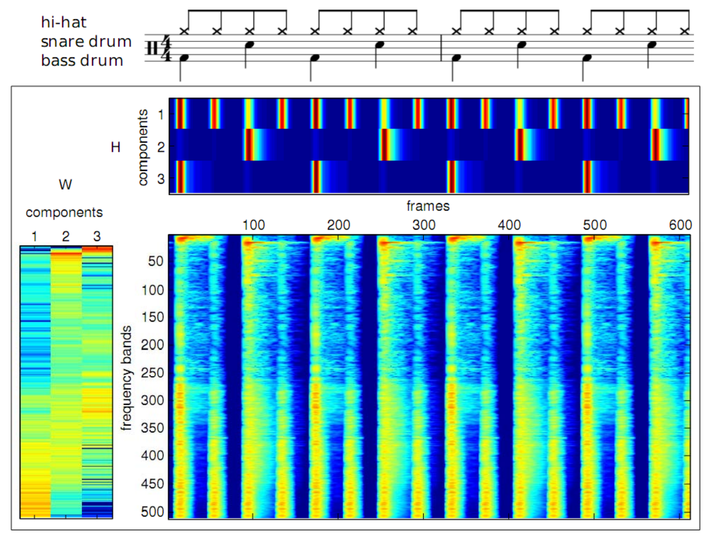
•NMF is an optimization problem, and for music a divergence cost function works well:

Given an $M \times N$ non-negative matrix $\mathbf{X} \in \mathbb{R}_+^{M \times N}$, find matrices $\mathbf{W} \in \mathbb{R}_+^{M \times K}$ and $\mathbf{H} \in \mathbb{R}_+^{K \times N}$ that minimize the cost function $f(\mathbf{X}, \mathbf{WH})$.

$$D(\mathbf{X}\|\mathbf{WH}) = \sum_{ij}\left(\mathbf{X}_{ij}\log\frac{\mathbf{X}_{ij}}{(\mathbf{WH})_{ij}} - \mathbf{X}_{ij} + (\mathbf{WH})_{ij}\right)$$

•We use multiplicative gradient-based updates:

$$\mathbf{H} \leftarrow \mathbf{H}.*\frac{\mathbf{W}^T\frac{\mathbf{X}}{\mathbf{WH}}}{\mathbf{W}^T\mathbf{1}}, \qquad \mathbf{W} \leftarrow \mathbf{W}.*\frac{\frac{\mathbf{X}}{\mathbf{WH}}\mathbf{H}^T}{\mathbf{1}\mathbf{H}^T}$$

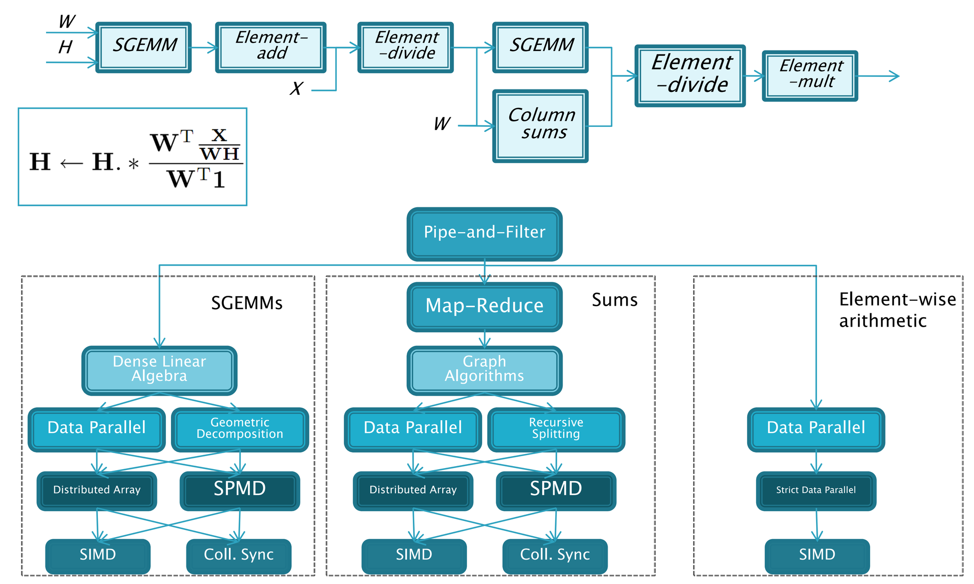


3-source NMF results aligned with the input audio's score

Organizing with Design Patterns

•Example of a design pattern decomposition for one update step on CUDA

•This helps us organize our code and communicate our computational needs.



•SGEMMs require ~400 Mflops per iteration, while other steps require less than 10 Mflops.

•But sums require inter-thread communication, and divides are slow.

OpenMP Results

```
    s = 0;
#pragma omp parallel num_threads(nt)
#pragma omp for reduction(+:s)
    for(i=0;i<N;i++)
        s += a[i];
```

for pragma with *reduction* clause

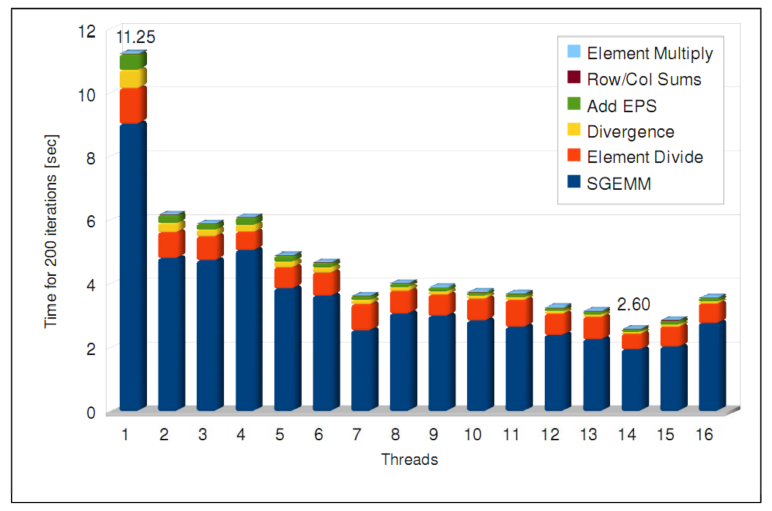
•Intel's MKL is used for SGEMMs

•OpenMP *for* and *reduction* clauses are used for sums and element-wise arithmetic

•Scaling on dual-socket Nehalem show:

•4x speedup over sequential C

•7x speedup over Matlab



Tuning CUDA

•We use SGEMM from CUBLAS 2.1

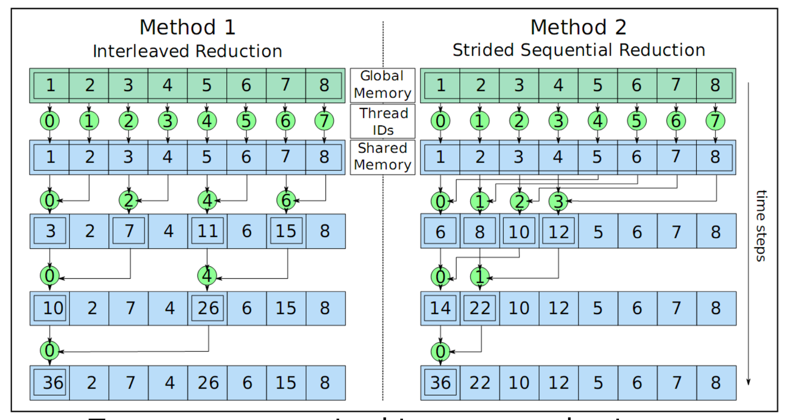
•**SGEMMs** run 26% faster if matrices are padded to multiples of 32

```
// kernel definition
__global__ void vecAdd(float* a,
                       float* b, float* c)
{
    int i = threadIdx.x+blockIdx.x*blockDim.x;
    c[i] = a[i] + b[i];
}

int main()
{
    . . .
    // kernel invocation
    vecAdd<<<B,N>>>(a,b,c);
}
```

Example of element-wise addition

•**Element-wise arithmetic** is accomplished using a separate thread for each element.



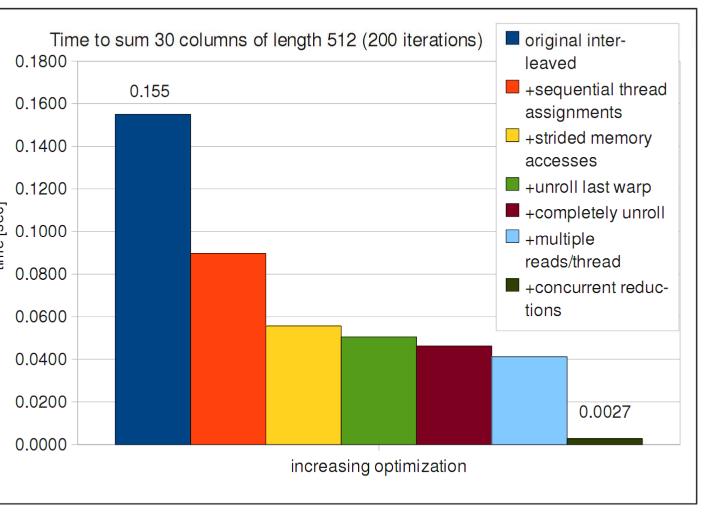
Two ways to organize binary tree reduction

•**Reductions** (sums) require most programming effort.

•Reorganize binary tree reduction to avoid divergent warps and memory bank conflicts (as in Method 2).

•Also, loop unrolling, and multiple global memory reads per thread.

•Most speedup comes from running the 30 sums concurrently.



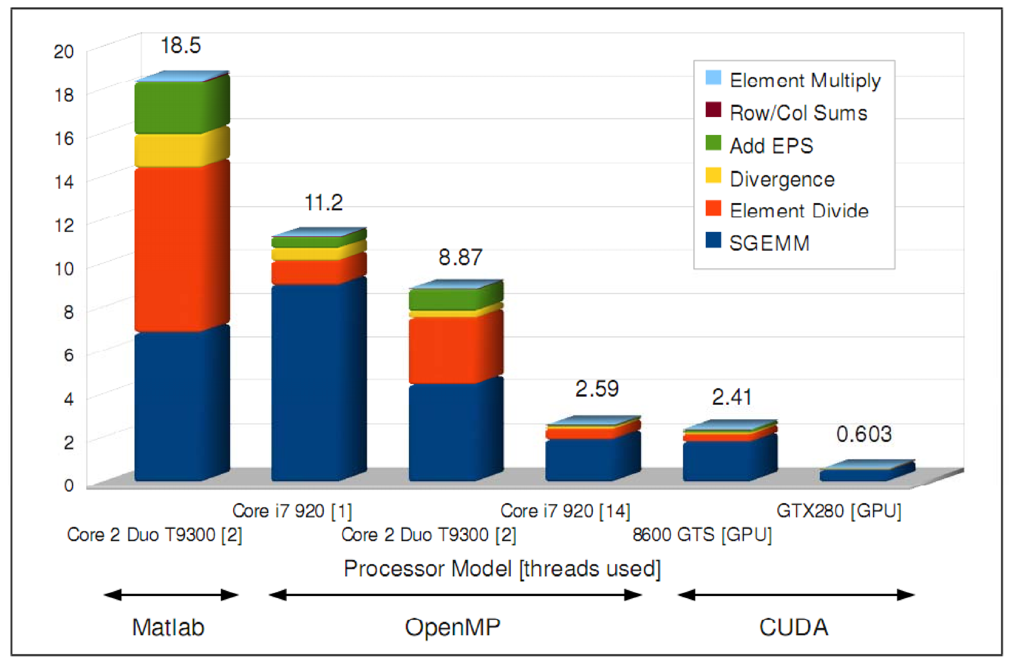
CUDA Results

•CUDA version runs over 30x faster than Matlab version.

•18.6x faster than OpenMP with 14 threads

•4.3x faster than sequential C

•Computation time down to 0.6 sec for 20 sec of audio which makes the app much more feasible.



•However, programming in CUDA requires much more effort than OpenMP and Matlab (especially when we need inter-thread communication).

•Programming in low-level CUDA is only worthwhile for important compute-intensive routines

Conclusions

•CUDA can achieve high performance for data-parallel music applications.

•Programmer effort in CUDA is much too great for music applications programmers.

Continued Work

•Developing Python modules of these implementations.

•Potential for Copperhead project to make CUDA more practical for writing music apps.

•Eventually building a DSL or framework to assist in constructing parallel music apps.