Parallel Music Applications

Thoughts and Results

Eric Battenberg and David Wessel

Universal Parallel Computing Research Center
The Center for New Music and Audio Technologies
University of California, Berkeley

Microsoft Parallel Applications Workshop 28, 29 May 2009

CNMAT

Range of Apps

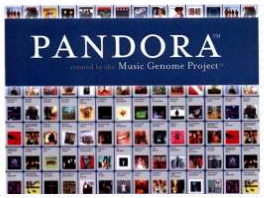
Hundreds of apps and plug-ins

Performance/Composition

Music Information Retrieval

gracenote

ircam
Centre
Pompidou



last·fm

Hearing Augmentation
for Music

Starkey

SHRC

□ Begin Experiment (press only once at beginning of experiment)

● Proceed to the Next Song

Song 0 Trial 0

□ Pause Music

□ Resume Music

3D Sound:
Speaker/Microphone Arrays

Meyer Sound

In this talk…

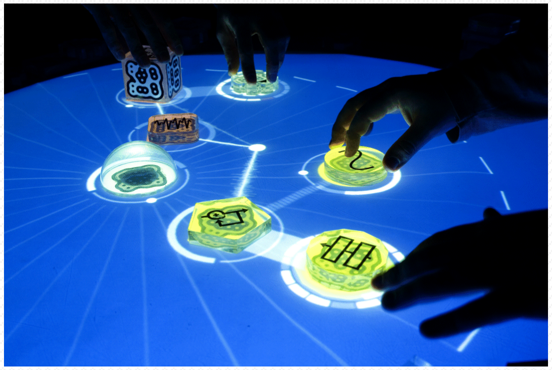
- Background on music applications
- Insights into music and parallel computing
- Organizing Apps with Parallel Design Patterns
- Case study
  - Parallelizing drum track extraction on OpenMP and CUDA
- Brainstorm
  - The future of performance and retrieval

Music Performance and Composition

- Novel musical interfaces allow for accessible and interesting performances.

Multi-Touch Array
Designed by David Wessel, Adrian Freed, Rimas Avizienis, and Matthew Wright

Tablo
Designed by Adrian Freed



Reactable
Designed by Sergi Jordà, Marcos Alonso, Martin Kaltenbrunner and Günter Geiger

Music Performance and Composition

- It is becoming common for amateur musicians to create professional-quality music in a “home studio” or Digital Audio Workstation
- DAW =

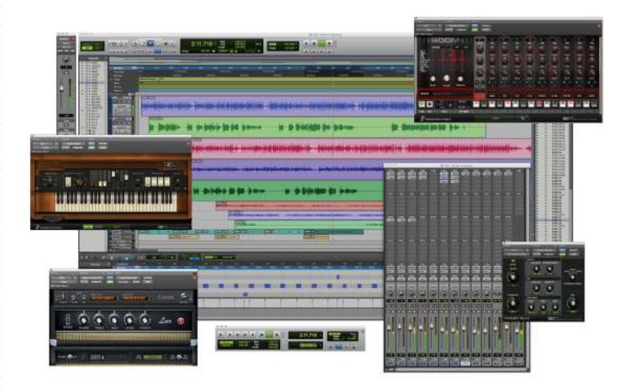
Personal computer

+

Sound card/mixer

+

Audio editing software



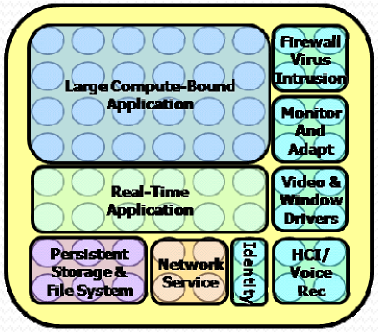
Music Performance and Composition

- The power of audio editing/processing software lies in its extensibility via plug-ins.
- In an audio processing chain, plug-ins can be composed in a task-parallel matter.
- When composed:
  - Are they thread safe?
  - Will they cause catastrophic performance conflicts?
  - Will they appropriately share hardware resources with other programs?

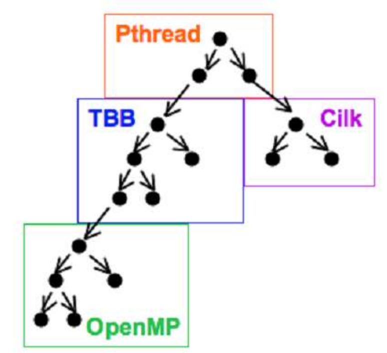
Audio plug-ins

Partitioning Hardware Resources

- What do we need from the OS?
  - *Tesselation*: low-level resource allocation
  - For music, we also need timing/deadline guarantees for real-time performance/processing

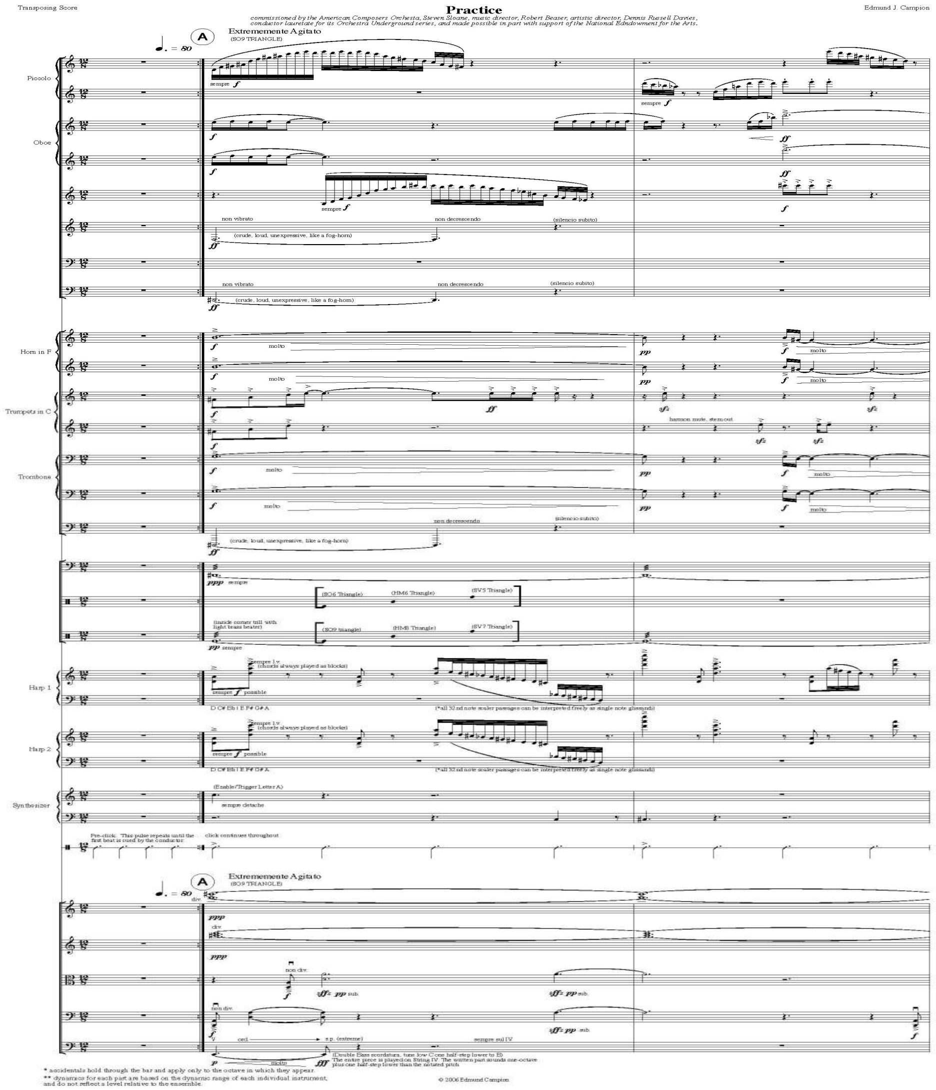


- What do we do with the allocated resources?
  - Naïve composition of computational kernels can destroy performance.
  - *Lithe*: Second-level application-aware low-level resource partitioning.



Music is inherently very parallel

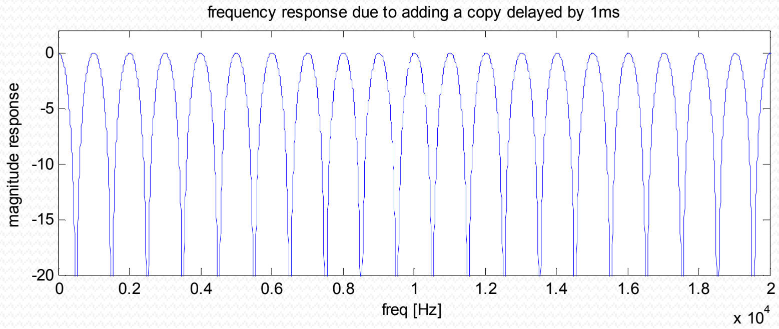
- Multiple tracks, lines, voices, parts, channels, etc.


Transposing Score
Practice
commissioned by the American Composers Orchestra, Steven Sloane, music director, Robert Beaser, artistic director, Dennis Russell Davies, conductor laureate for its Orchestra Underground series, and made possible in part with support of the National Edndowment for the Arts.
Edmund J. Campion
A
Extrememente Agitato
(SO9 TRIANGLE)
Piccolo
Oboe
Horn in F
Trumpets in C
Trombone
Harp 1
Harp 2
Synthesizer
© 2006 Edmund Campion

But audio synchronization and timing are very important in parallel music apps.

Audio Synchronization/Timing

- The ear is *very* sensitive to timing.
- If tasks are processed on separate cores, delays can be introduced.
- If these delays are not compensated for, the sound quality can be adversely affected.
- Examples:
  - Musical piece played without any delay
  - Same piece with a copy added that is delayed by 1ms.
    - We get a "combing" effect in the frequency domain.



No delay

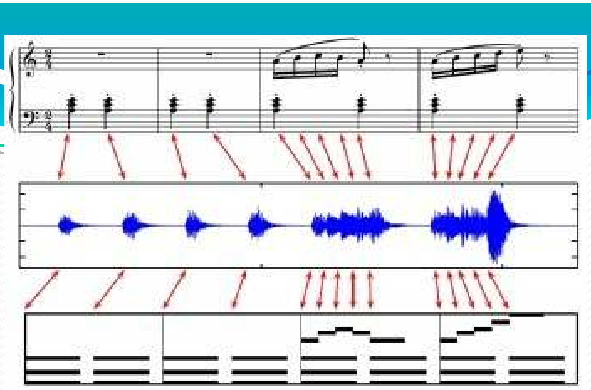
1ms delay

Open Sound Control (OSC)

a way to achieve synchronization

- Communication protocol to share musical data over a network.
- Symbolic and high-resolution numeric argument data
- Pattern matching language to specify multiple recipients of a single message
- High resolution time tags for sub-sample accurate synchronization
- "Bundles" of messages whose effects must occur simultaneously (atomic updates)

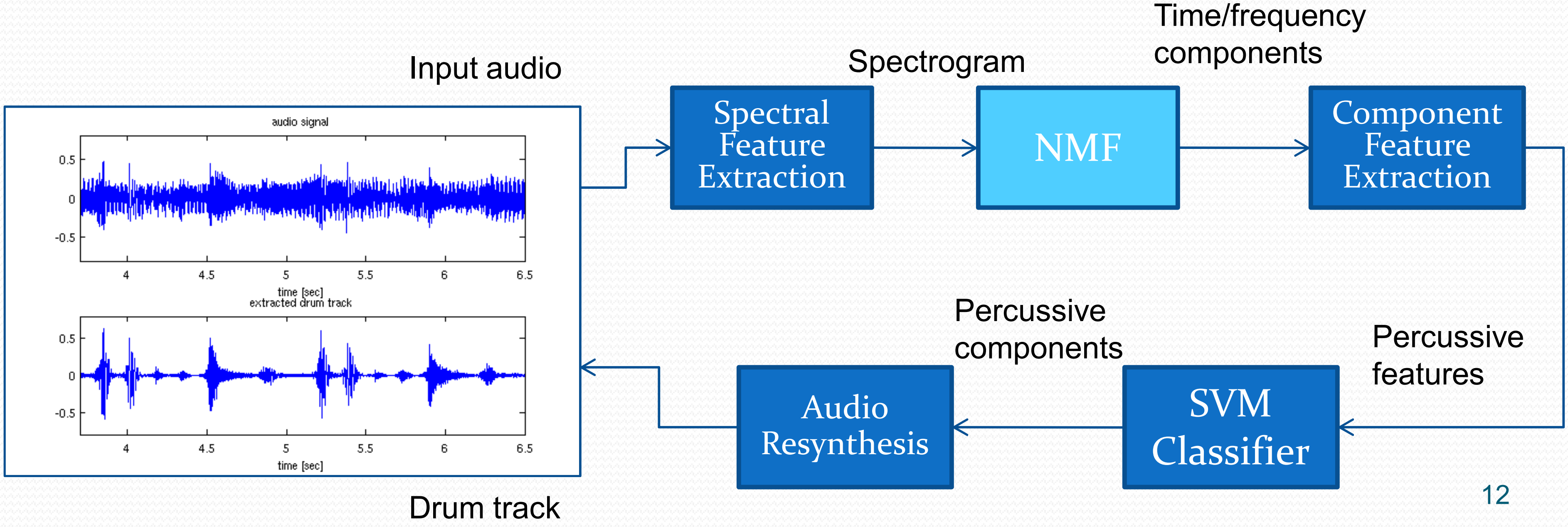
MIR Apps



- *Music Information Retrieval, "Machine Listening", "Music Understanding"*
- **Transcription** - Automatically generate a score or tablature from audio
- **Source separation** - Isolate certain instruments (including the singer)
- **Similarity, Playlist creation, content discovery**
  - Automatically generate a playlist to fit a mood or based on song similarity.
- **Artist, genre, mood classification or quantification**
  - Help organize a music archive
- **Score Following, lyrics sync, beat tracking**
  - Useful for DJs, karaoke, music education, and automated accompaniment.
- **Song Segmentation**
  - Partition song into discrete passages (verse, chorus, bridge) for individual analysis
- The hope is that someday you will be able to query for music like this:
- *"I like the drummer but can't stand the singer. Find me something in the same genre with drumming like this but with a singer that sounds more like John Lennon."*

Case Study: Drum Track Extraction

- An example of source separation where the drum track is isolated.
  - Useful in drum transcription, beat tracking, and rhythm analysis.
- Audio spectrogram is factorized into components using Non-negative Matrix Factorization (NMF).
- Components are classified using a Support Vector Machine (SVM).
- "Percussive" components are used to synthesize an audio drum track.
- **NMF step is most computationally intensive.**
  - 80% of time in Matlab (18.5 sec of 23.1 sec total for 20 sec of audio)
  - **We will parallelize NMF using OpenMP (for multi-core) and CUDA (for GPUs)**



Input audio

Spectrogram

Time/frequency components

Spectral Feature Extraction

NMF

Component Feature Extraction

Percussive features

SVM Classifier

Percussive components

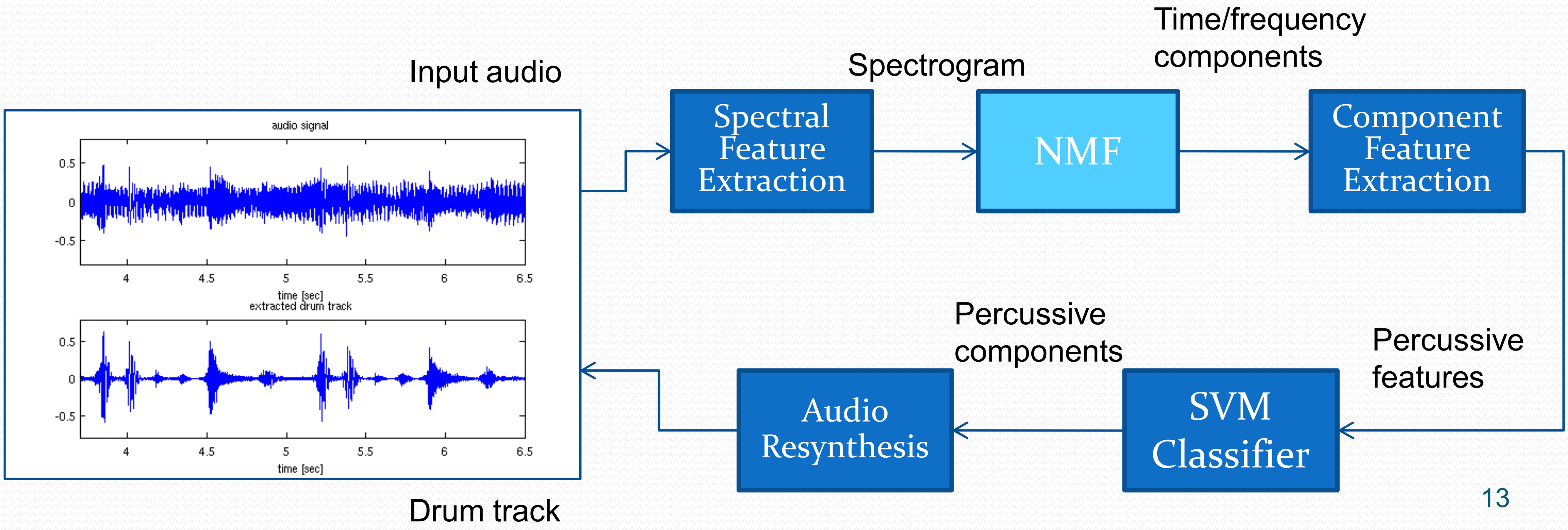
Audio Resynthesis

Drum track

Case Study: Drum Track Extraction

Audio examples (listen for drums in original)

	1	2	3
Original	🔊	🔊	🔊
Drum Track	🔊	🔊	🔊



Input audio

Spectral Feature Extraction

Spectrogram

NMF

Time/frequency components

Component Feature Extraction

Percussive features

SVM Classifier

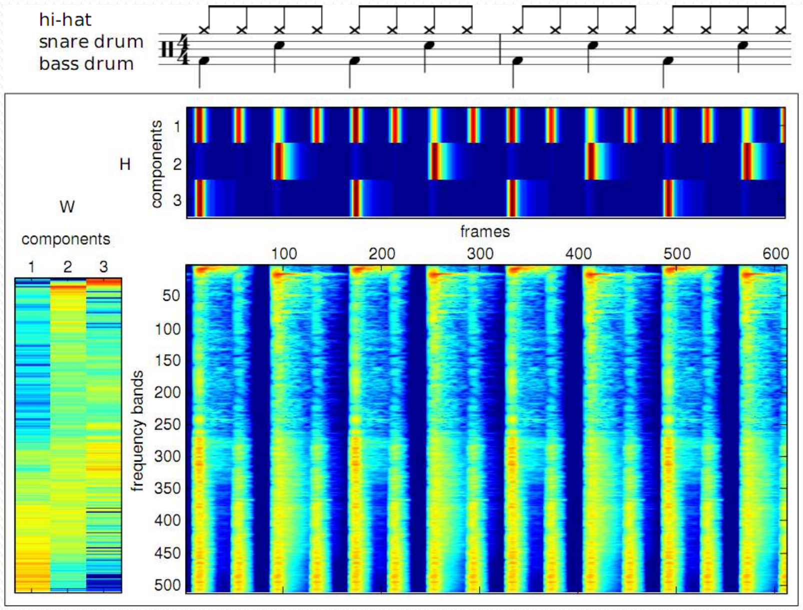
Percussive components

Audio Resynthesis

Drum track

Case Study: Drum Track Extraction

- Use Non-negative Matrix Factorization to separate an audio spectrogram into sources.
  ($X = W*H$)
  - Here we see a spectrogram surrounded by its time (*H)* and frequency *(W)* component matrices.
    (3 sources).
  - The time components in *H* are aligned with the corresponding drum score.



Case Study: Drum Track Extraction

- NMF is the optimization problem:

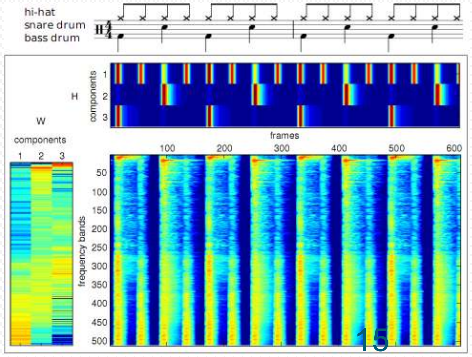
Given an $M \times N$ non-negative matrix $\mathbf{X} \in \mathbb{R}_+^{M \times N}$, find matrices $\mathbf{W} \in \mathbb{R}_+^{M \times K}$ and $\mathbf{H} \in \mathbb{R}_+^{K \times N}$ that minimize the cost function $f(\mathbf{X}, \mathbf{WH})$.

- A cost function that works well for music:
  - Similar to Kullback-Leibler divergence

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

- **Multiplicative gradient-based updates**

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



Case Study: Drum Track Extraction

- For [512 x 30 x 3445] NMF,
  - 512 frequency components, 30 sources, 3445 time frames (~20 sec)
- For each iteration we have:
  - 423 Mflops of SGEMMs (**S**ingle-precision **Ge**neral **M**atrix **M**ultiply)
  - 3.6 Mflops of element-divides (slow)
  - 0.1 Mflops element-multiplies
  - 0.1 Mflops sums (requires communication)

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

- Also:
  - Add a small constant to divisor matrices to prevent divide-by-zero. (Add *EPS,* 3.6 Mflops)
  - Compute log-based cost function every 25 iterations to check for convergence.

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

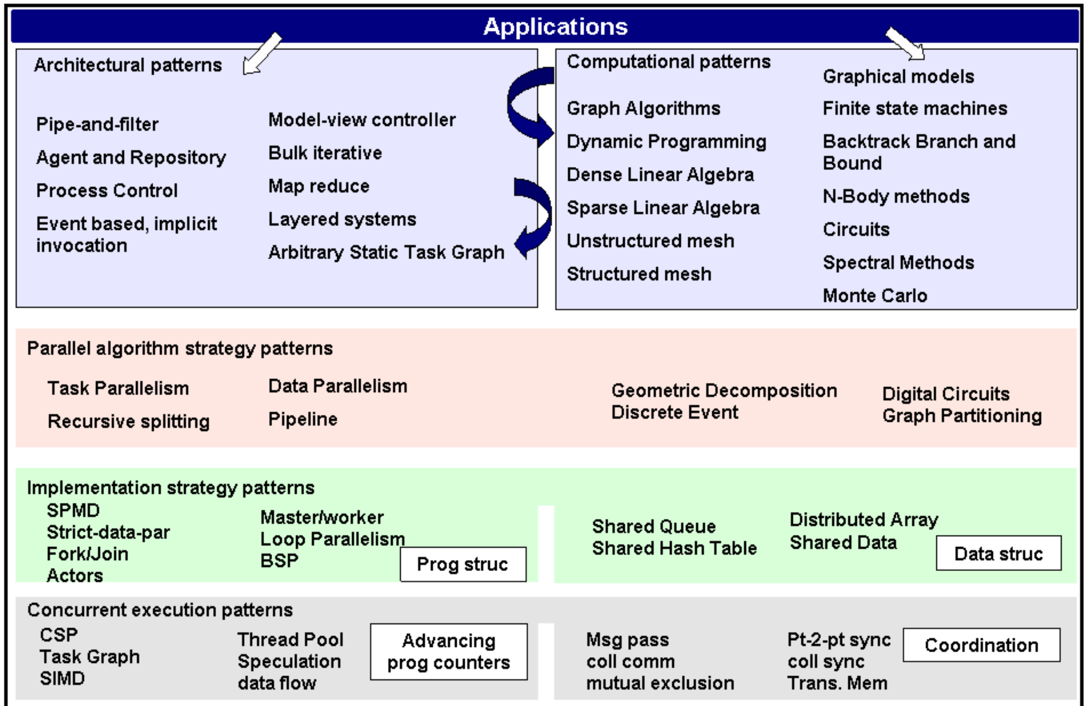
Organizing Parallel Apps

- How can we organize the design of our applications?
- How can we best communicate our development process and computing demands to other applications experts?

Parallel Design Patterns

- Application developers are starting to adopt HPC jargon since science has been using parallel computing for decades.
- The Par Lab, led by Tim Mattson and Kurt Keutzer, is developing a parallel pattern language, OPL.

- OPL is hierarchical
  - Higher-level patterns rely on the details contained in lower-level patterns
- Purpose of parallel pattern language.
  - Education about best practices
  - Common terminology
  - Guides the design process.



Applications

Architectural patterns

Pipe-and-filter
Agent and Repository
Process Control
Event based, implicit invocation

Model-view controller
Bulk iterative
Map reduce
Layered systems
Arbitrary Static Task Graph

Computational patterns

Graph Algorithms
Dynamic Programming
Dense Linear Algebra
Sparse Linear Algebra
Unstructured mesh
Structured mesh

Graphical models
Finite state machines
Backtrack Branch and Bound
N-Body methods
Circuits
Spectral Methods
Monte Carlo

Parallel algorithm strategy patterns

Task Parallelism
Recursive splitting
Data Parallelism
Pipeline
Geometric Decomposition
Discrete Event
Digital Circuits
Graph Partitioning

Implementation strategy patterns

SPMD
Strict-data-par
Fork/Join
Actors
Master/worker
Loop Parallelism
BSP
Prog struc

Shared Queue
Shared Hash Table
Distributed Array
Shared Data
Data struc

Concurrent execution patterns

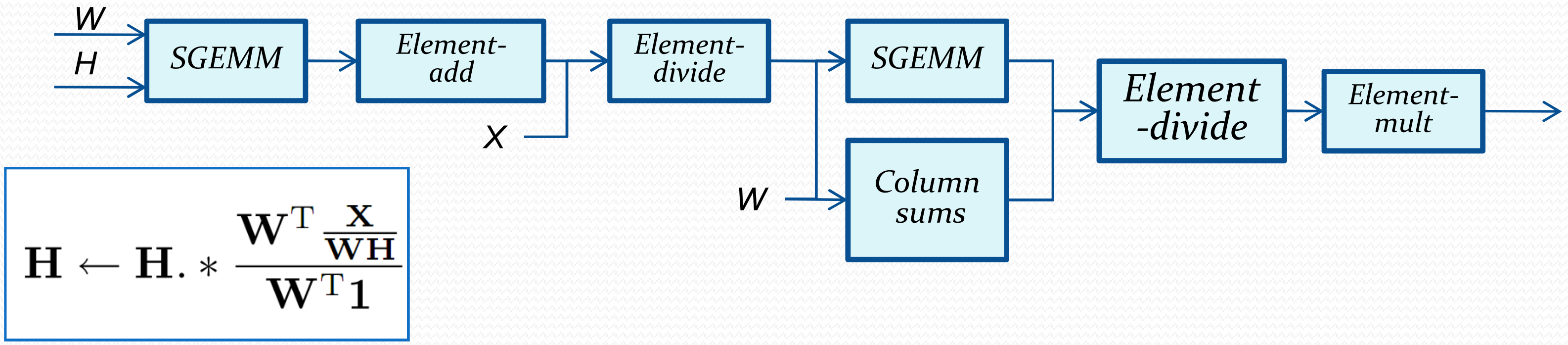
CSP
Task Graph
SIMD
Thread Pool
Speculation
data flow
Advancing prog counters

Msg pass
coll comm
mutual exclusion
Pt-2-pt sync
coll sync
Trans. Mem
Coordination

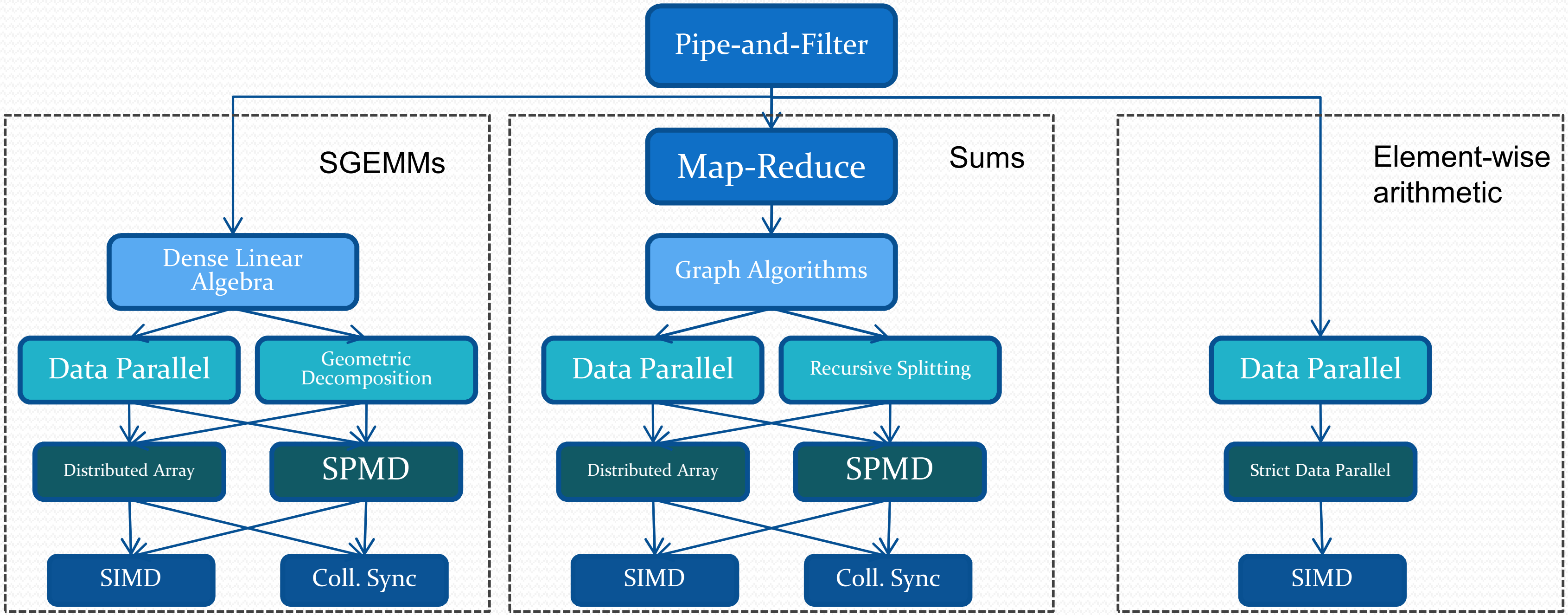
Figure 1: overall structure of OPL showing the five layer model. Implementation strategy patterns are divided into 2 sets; one describing a program's structure and the other data structures. The concurrent execution patterns are broken down into a set of patterns that "advance a program counter" and a set that coordinates the execution of parallel threads.

Parallel Design Patterns

- Example design pattern decomposition for CUDA implementation of NMF
- The pattern language helps us organize our code.
- Each design pattern is described in a document, outlining best practices and giving pointers to helpful resources.

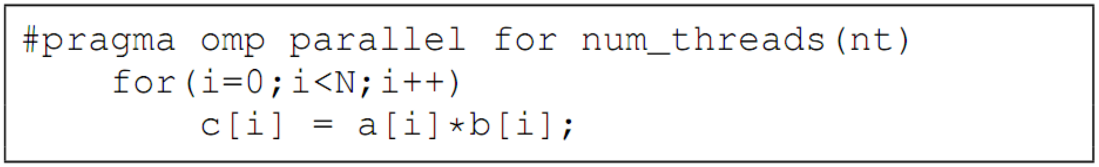


$$\mathbf{H} \leftarrow \mathbf{H}.*\frac{\mathbf{W}^{\mathrm{T}}\frac{\mathbf{X}}{\mathbf{WH}}}{\mathbf{W}^{\mathrm{T}}\mathbf{1}}$$



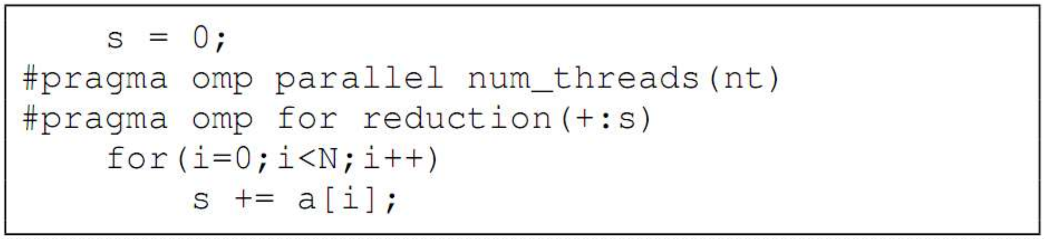
OpenMP (the easy stuff)

- Data-parallel *for* loop
  - To be used for **element-wise arithmetic**
  - Create team of *nt* threads to do independent chunks of work



```
#pragma omp parallel for num_threads(nt)
    for(i=0;i<N;i++)
        c[i] = a[i]*b[i];
```

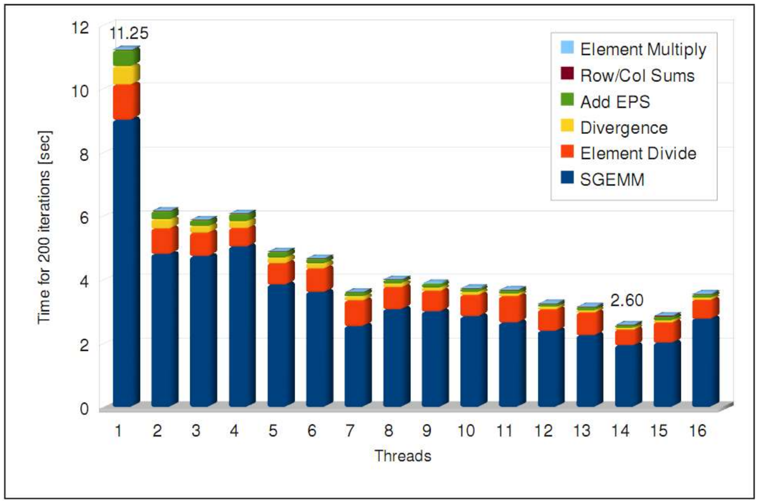
- Reduction
  - **For sums**
  - Create team of *nt* threads to compute partial sums
  - Then add the partial sums to final variable *s*



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

OpenMP (the easy stuff)

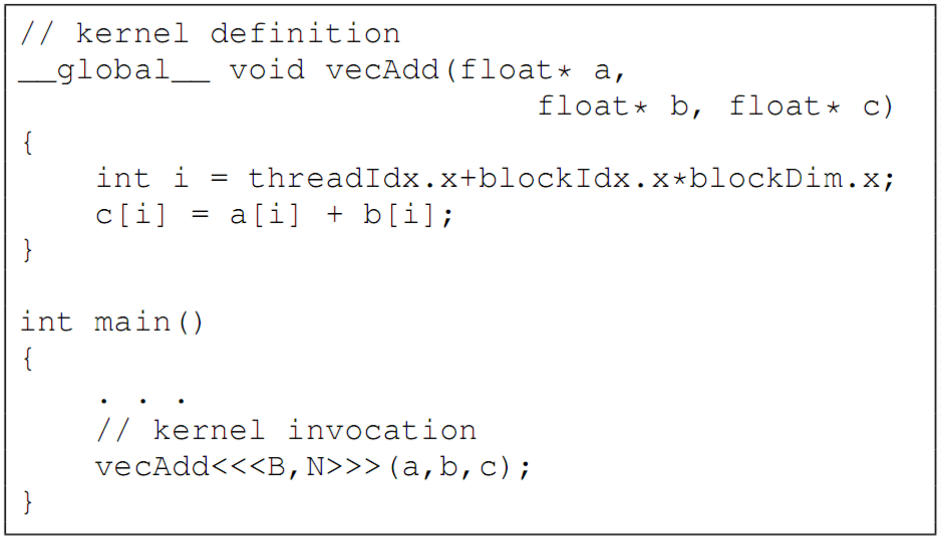
- We use MKL for SGEMMs
- Use OpenMP for other routines
- Performance scaling on dual-socket Core i7 920:



- SGEMMs show most significant speedup
  - Highest work to communication ratio
- Non-linear speedup suggests this won’t scale well to more cores using this architecture and programming model.
- However,
  - >7x speedup compared to Matlab
  - >4x speedup compared to sequential C

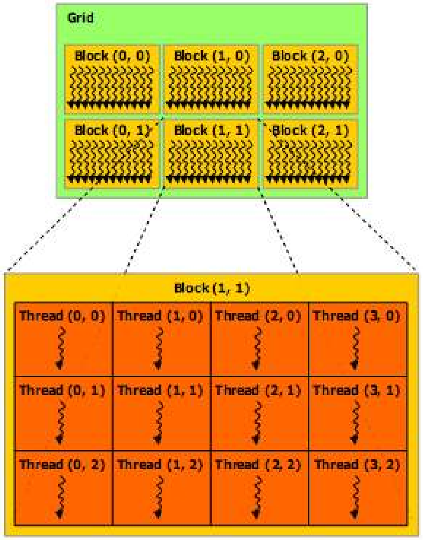
CUDA (some harder stuff)

- CUDA is used to program Nvidia GPUs for general computation.
- GPU code is executed by many threads independently in a SPMD manner.
- Threads grouped into a *thread block* can share memory.
- Threads are physically executed in groups of 32, called *warps*.
  - If all threads within a warp do the same thing, we get SIMD.
- Below we see a kernel definition and invocation for vector addition.
  - Kernel is invoked with *B* blocks of *N* threads.
  - Each thread operates on one element of each array.
  - The element index is computed from the thread ID, block ID, and block size corresponding to the running thread.



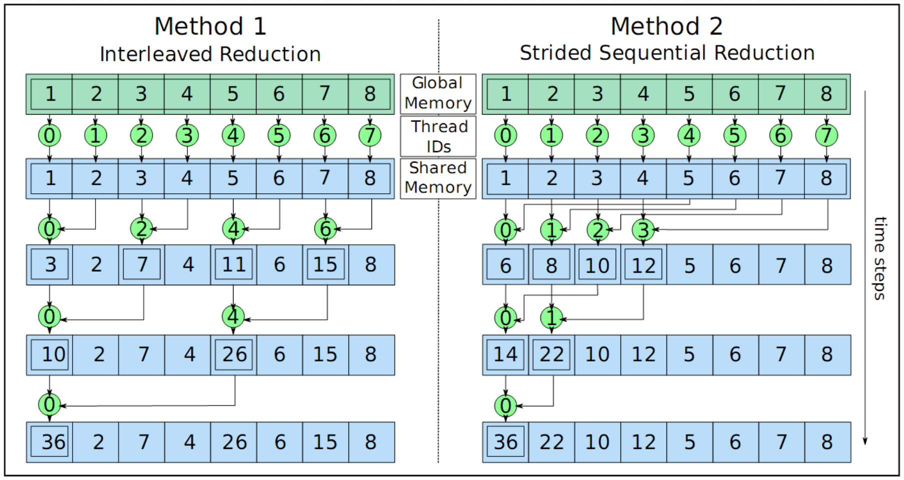
```
// kernel definition
__global__ void vecAdd(float* a,
                       float* b, float* c)
{
    int i = threadIdx.x+blockIdx.x*blockDim.x;
    c[i] = a[i] + b[i];
}

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

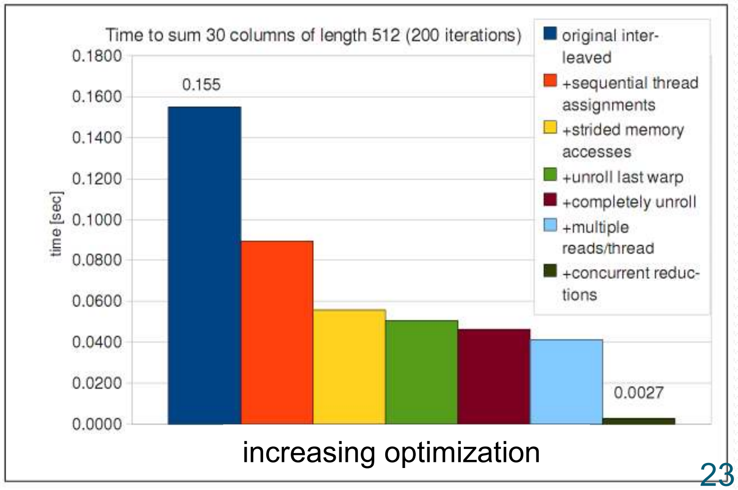


CUDA (some harder stuff)

- NMF Implementation in CUDA
  - SGEMMs – use CUBLAS 2.1, achieves 60% of peak (373 GFLOPS on GTX 280)
    - Padding matrices to multiples of 32 reduces SGEMM running time by 26%
  - Element-wise arithmetic – similar to example code
  - Reductions (sums) – a lot harder in CUDA than OpenMP
    - Use optimizations covered in CUDA SDK for shared memory reduction.
      - Reorganize binary tree traversal.
      - Loop unrolling, multiple reads per thread.
    - Run the 30 sums concurrently. An important optimization.



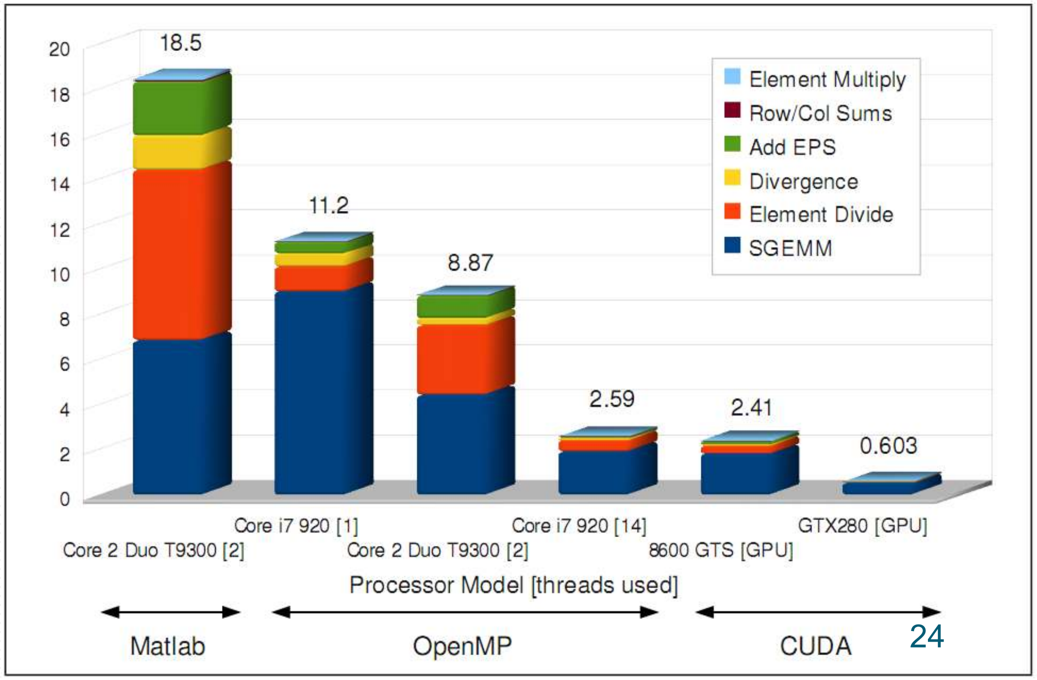
57x speedup overall



CUDA vs. OpenMP

- CUDA achieves much higher performance on current GPUs for highly data-parallel computations. (>30x speedup compared to Matlab, 4x faster than OpenMP+Nehalem)
- OpenMP can achieve multi-core speedup on data-parallel computations with very little programmer effort.
- If inter-thread communication is required, things become much more difficult.
  - OpenMP gets harder.
  - CUDA gets *a lot* harder.

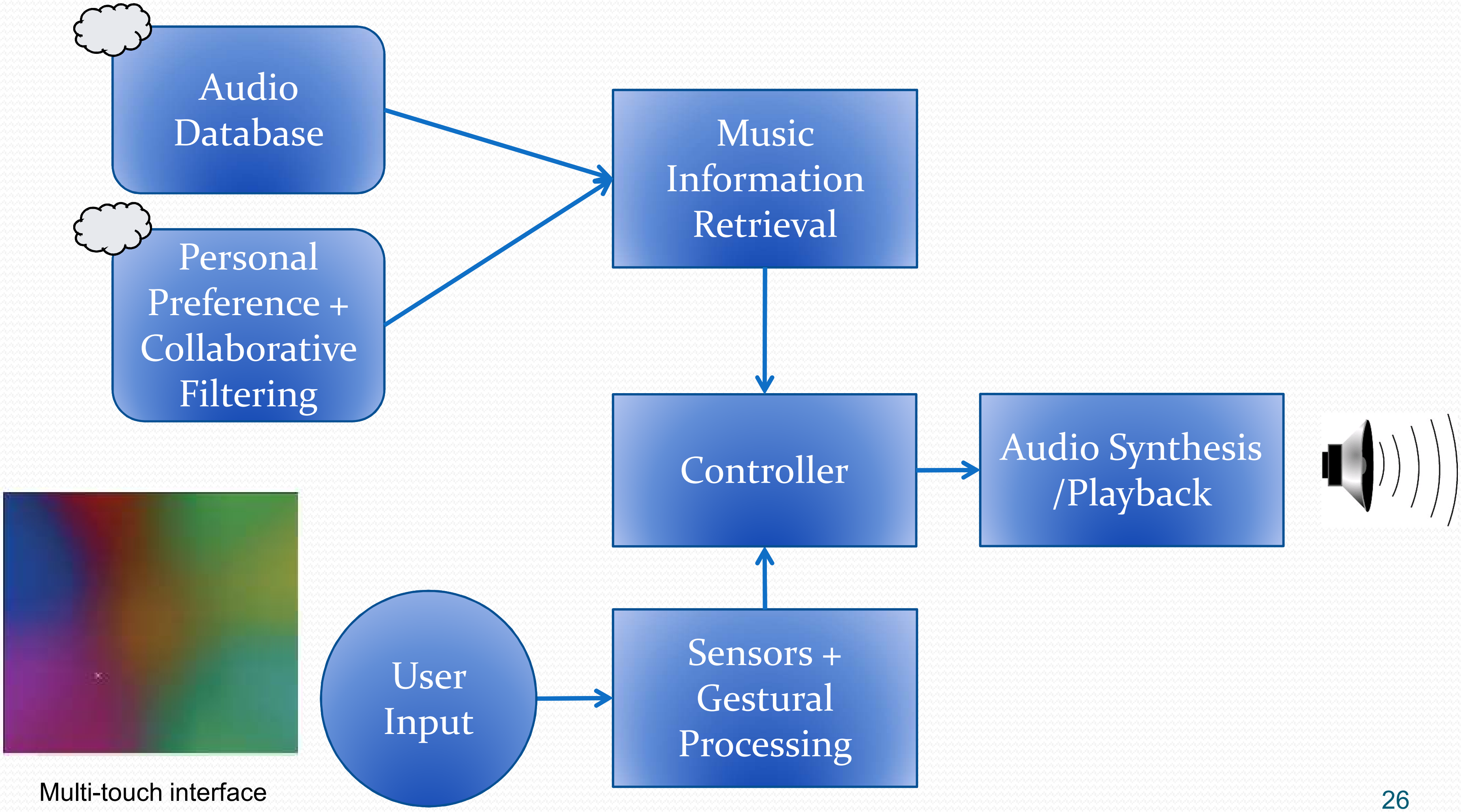
- For music application developers, CUDA is only feasible for computational kernels that require very high performance. What about latency of going to GPU and back?
- We will be releasing Python modules based on these implementations.
- Can be used for general NMF as well.



An idea for the future: Analysis/Performance Hybrid

- Combine MIR analysis on a database of music in the cloud with audio synthesis techniques to create custom music controlled by gestural processing and personal preferences.
- Automatic Mash-ups/Remixes.
- Gestural music selection (e.g. at a party)
- As little or as much interaction as desired.
- Can be used in music performance or just for interactive listening.

Brainstorm:
Interactive Musical Experience


Audio Database
Personal Preference + Collaborative Filtering
Music Information Retrieval
Controller
Audio Synthesis /Playback
User Input
Sensors + Gestural Processing

Multi-touch interface

Wrap

- There are tons of music applications.
  - For both music fans and musicians.
- Parallel computing enables new music applications
  - But synchronization and real-time are important.
- Parallel design patterns are useful for communicating ideas and organizing code.
- Questions?