Implementing CUDA Audio Networks

Giancarlo Del Sordo
Acustica Audio
giancarlo@acusticaudio.com
Motivation

• Vintage gear processing in software domain
• High audio quality results
• Low cost and user-driven content approach
Outline

• Audio theory
• Basic engine solution
• Network implementation
• Results
Audio theory
Standard DSP approach

Example: simple 2 poles filter (source www.musicdsp.org)

c = pow(0.5, (128-cutoff) / 16.0);
r = pow(0.5, (resonance+24) / 16.0);

//Loop:

v0 = (1-r*c)*v0  -  (c)*v1  +  (c)*input;
v1 = (1-r*c)*v1  +  (c)*v0;

output = v1;

Drawbacks:

• time/costs required for achieving good quality algorithms
• complexity for taking advantage of GPU resources
Volterra Series

\[ y(n) = \sum_{i=0}^{M-1} h_1(i) \times x(n-i) + \sum_{i=0}^{M-1} h_2(i) \times x^2(n-i) + \sum_{i=0}^{M-1} h_3(i) \times x^3(n-i) + \ldots \]
Solution for dynamic processing (preamps, reverbs)

<table>
<thead>
<tr>
<th>Input Assessment</th>
<th>0 dB</th>
<th>-2 dB</th>
<th>-4 dB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><img src="h1(n)" alt="Diagram of preamplifiers" /></td>
<td><img src="h2(n)" alt="Diagram of reverbs" /></td>
<td><img src="h3(n)" alt="Diagram of reverbs" /></td>
</tr>
</tbody>
</table>

- **Envelope detector/follower**
  - Length: 2ms to 200ms
  - Refresh: 2ms to 50ms

- **Preamplifiers**
  - Length: 200ms to 10s
  - Refresh: up to 180ms
Solution for dynamic processing (comp/limiters)

- Envelope detector/follower
- Transfer curve
- 0 dB
- -2 dB
- -4 dB

Length: 2ms to 200ms
Refresh: 2ms to 50ms
Solution for equalizers/filters

Length: 10ms to 500ms

cnob 1

cnob 2

root

nodes

leafs
Solution for time-varying fx (chorus, flanger, ...)

Length: 2ms to 200ms
Refresh: 2ms to 50ms
Block diagram of the engine (1)

Kernel engine
- Realtime
- Advantages:
  - low costs required for achieving accurate algorithms (sampling approach)
  - easy CUDA implementation for Kernel engine

Vectorial engine
- Refresh: 2ms to 180ms
Block diagram of the engine (2)
Basic engine solution
Inner block (1)

Input

```c
memset (cVtemp, 0, cOp32->iVextendedbufferbytes);
for (iV = 0; iV < cOp32->iVldim; iV++) cVtemp [iV].x = (float) dX_ [iV];
cudaMemcpyAsync (cVtemp_, cVtemp, cOp32->iVextendedbufferbytes, cudaMemcpyHostToDevice, cOp32->iVstream);
```

FFT (input and kernel)

```c
if (!bUpdate) acfftExecC2C (cOp32->cVacplanx, cVtemp_, cVtemp_, CUFFT_FORWARD, cOp32->iVstream);
else {
    cudaMemcpyAsync (cOp32->fVirfftpartition_ [iIrchannel], cOp32->fVirfftpartition [iIrchannel], cOp32->iVcircularbufferbytes, cudaMemcpyHostToDevice, cOp32->iVstream);
    acfftExecC2C (cOp32->cVacplany [iIrchannel],
                 (cufftComplex*) cOp32->fVcoalescedirfftpartition_ [iIrchannel],
                 (cufftComplex*) cOp32->fVcoalescedirfftpartition_ [iIrchannel],
                 CUFFT_FORWARD, cOp32->iVstream);
}
```

Multiply and add

```c
muacp<<iCpar0, iCpar1, 0, cOp32->iVstream>>>(
    cVtemppointera_, (tCcomplex*) fVtemppointerb_, (tCcomplex*) fVtemppointerc_,
    cOp32->iVbins, cOp32->iVpartitions, cOp32->iVcircularpointer);
```
Inner block (2)

Output

```c
switch (cOp32->iVchainedposition) {
    case iCchainedfirst:
        cudaMemcpyAsync (&cOp32->fVchainedbuffer_ [iVtemp2],
                         fVtemppointerc_, cOp32->iVextendedbufferbytes,
                         cudaMemcpyDeviceToDevice, cOp32->iVstream);
        ...
        break;
    case iCchainedmiddle:
        addfl<<<iCpar0, iCpar1, 0, cOp32->iVstream>>>(fVtemppointerc_,
                                                   &cOp32->fVchainedbuffer_ [iVtemp2],
                                                   cOp32->iVextendedsection);
        ...
        break;
    case iCchainedlast__:
        addfl<<<iCpar0, iCpar1, 0, cOp32->iVstream>>>(&cOp32->fVchainedbuffer_ [iVtemp2],
                                                     fVtemppointerc_,
                                                     cOp32->iVextendedsection);
    case iCchainednone__:
        acfftExecC2C (cOp32->cVacplanxinv, (cufftComplex*) fVtemppointerc_, cVtemp_,
                      CUFFT_INVERSE, cOp32->iVstream);
        ...
        cudaMemcpyAsync (cOp32->fVextendedbuffer, cVtemp_, cOp32->iVextendedbufferbytes,
                         cudaMemcpyDeviceToHost, cOp32->iVstream);
```

Output IFFT $y$
Inner block (3)

In-depth analysis: multiply and add

\[
\text{muacp} \langle\langle\_\_\rangle\rangle (\ldots);
\]

```c
static __global__ void muacp ( tCcomplex* cA, tCcomplex* cB, tCcomplex* cC, int iSize,
    int iPartitions, int iCircularpointer) {
    const int iCnumthreads = blockDim.x * gridDim.x;
    const int iCthreadid = blockIdx.x * blockDim.x + threadIdx.x;
    int iVctemp;

    for (int iV = iCthreadid; iV < iSize; iV += iCnumthreads)
        for (int iVinner = 0; iVinner < iPartitions; iVinner++) {
            iVctemp = iSize * ((iCircularpointer + iVinner) % iPartitions) + iV;
            cC[iVctemp] = masce (cA[iV], cB[iSize * iVinner + iV], cC[iVctemp]);
        }
}
```

```c
static __device__ __host__ inline tCcomplex masce (tCcomplex cA, tCcomplex cB, tCcomplex cC) {
    tCcomplex cVreturn;

    cVreturn.x = cC.x + cA.x*cB.x - cA.y*cB.y;
    cVreturn.y = cC.y + cA.x*cB.y + cA.y*cB.x;
    return cVreturn;
}
```
**In-depth analysis: FFT**

```c
void acfftExecC2C ( ACFFTLOCHANDLE cPlan, float2* cInput, float2* cOutput,
    int iDirection, int iStream = 0) {

    ((ACFFTLOC*) cPlan)->cVexecutefun [iDirection <= 0 ? 0 : 1] (cInput, cOutput, cPlan, iStream);
}

(*cPlan)->cVexecutefun [0] = cVacfftfunfw [...];

int iVacfftsize [] = {8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384};
ACFFTFUN cVacfftfunfw [] =
(acfnv, acfnv, acfnv, acfnv, acfnv, acfnv, acfnv, acfnv, acf01, acf02, acf03, acf04, acf05, acf06);

void acf06 ( float2* cInput, float2* cOutput, ACFFTLOCHANDLE cAcfft, int iStream = 0) {
    ACFFTLOC* cVloc = (ACFFTLOC*) cAcfft;

    v2_16<< grid2D(cVloc->iVbatches*(16384/16)/64), 64, 0, iStream >>>(cInput, cOutput);
    v1024<< grid2D(cVloc->iVbatches*16), 64, 0, iStream >>>(cOutput, cOutput);
}
```

Vasily Volkov inner blocks:
1024 = 16*4*16
16384 = 16*1024
Buffer tuning and latency (1)

FFT buffer (iVextendedbuffersize)

CUDA acceleration

0 100 200 300 400 500 600 ms

0 2000 4000 6000 8000 10000 12000 14000 16000 18000

compressors
preamps
early reflections
long TMV
rooms
reverbs
Buffer tuning and latency (2)

Results:
- real-time resources are doubled adding GPU-powered instances

CPU: Core2 Duo T5800 2.0 Ghz
GPU: GeForce 9600M GT
Buffer tuning and latency (3)

CPU: Core2 Duo T5800 2.0 Ghz
GPU: GeForce 9600M GT

3 instances, 1 kernel, 5.5 seconds, FFT buf = 16384, 30 partitions

IPP (Intel Performance Primitives): CPU 40%
FFTW: CPU 44%
CUDA – cufft library: CPU 5%, GPU 90%
CUDA – custom fft implementation: CPU 5%, GPU 36%
Audio processing buffers (1)

- Input
- Processed
- Realtime headroom
- Latency compensation
- Output
- Sleeps
Audio processing buffers (2)

- FFT output
- Refresh period (kernel swap)
- Example: linear interpolation
- Result: per-sample smoothing
Audio processing buffers (3)

Example: preamplifier featuring 50 ms kernels

Without interpolation

Log interpolation

Custom interpolation
Threading model

Host/DSP thread

GUI thread

CUDA thread

loading command

alloc $H_1$ alloc $H_2$ chaining

proc $H_1$ proc $H_2$

critical section

critical section

bypass sending old data new data

CUDA RUNTIME

CUDA RUNTIME
Block diagram of a plug-in implementation
Network implementation
Network implementation
Inter-process communication (1)

Custom socket implementation

cVsocket = new CoreSckt (type);

- Reliable
- Slower
- Remote host

- Not reliable (only for media data)
- Remote host

- Reliable
- Fast (10x)
- Local host
Inter-process communication (2)

Example: set a 2 state variable bean

```
cVprocess->setbb(CorePrcs::iCbeanbypass, true);
```

Client

```
cVtransportopcode->setvl (0, iCopcodesetbb);
``` message

```
cVtransportopcode->send_ (cVlocalsocket);
``` opcode

```
cVbeanstructb->setvl (0, iObject);
``` subopcode/bean item

```
cVbeanstructb->setvl (1, bValue);
cVbeanstructb->send_ (cVlocalsocket);
cVlocalsocket->lflush ();
cVtransportack->recv (cVlocalsocket);
```

Server

```
case iCopcodesetbb:
  cVbeanstructb->recv (cVlocalsocket);
cVbeanstructb->getvl (0, iVsubopcode);
cVbeanstructb->getvl (1, bVtemp);
cVbeanstructb->getvl (2, cVbeanstruct.iVobject);
cVbeanstructb->getvl (3, cVbeanstruct.iVsubobject);
cVprocess->setbb
  (iVsubopcode, bVtemp, &cVbeanstruct);
cVtransportack->send_ (cVlocalsocket);
```

Different threads on server side:

- Audio processing
- Editor: low priority beans
32 bits legacy hosts in 64 bits OS

32 bits:
- x64 version is not available
- plug-in is a dynamic library

64 bits:
- extended address space
- easy CUDA installation (driver, toolkit)
- very low overhead using named pipes/domain sockets
- lacking of memory fragmentation effect and fault tolerance
Multi-servers/Multi-clients block diagram

Achievements:

• single control point, shared resources, easy maintenance
• more OS are available at the same time (example: MAC OSX host connected to cheap windows servers)
Results
The library

- MAC OSX Tiger..Snow Leopard, Windows XP... 7
- Support for Microsoft, Borland/CodeGear, Apple compilers
- Support for IPP library
- Optional support for FFTW library
- Direct convolution implemented in ASM ia32 and x64
- Automatic application aimed to vintage gear sampling
- Huge number of emulations available and used daily in professional audio productions
On the road

- Big recording facility
- Client/Server as replacement of missing outboard
Next steps

- CUDA implementation of the vectorial engine
- Resource optimization (clients connected to the same server instance could share resources)
- New features based on client-to-client communication (ie cross-talk modeling)