CUDA-Accelerated Acquisition of Spread Spectrum Signal in Satellite Communication

Ying Liu, Hongyuan Cui, Renliang Zhao

School of Computer and Control
University of Chinese Academy of Sciences

& Haixin Zheng

Academy of Equipment
Outline

- Motivation
- Acquisition Algorithm for Spread Spectrum (SS) Signal
- Parallel Acquisition Algorithm on CUDA
- CU-Acqss on Multiple GPUs
- Performance Evaluation
- Conclusion
Motivation

- Spread-spectrum techniques: a signal generated with a particular bandwidth is deliberately spread in the frequency domain, resulting in a signal with wider bandwidth.

-- Wikipedia

Applications
- GPS, Beidou
- Telemetry
Motivation

A typical working flow of SS in satellite communication

![Diagram of satellite communication process]

1. **Motivation**
   - A typical working flow of SS in satellite communication

![Diagram of satellite communication process](image-url)
Motivation

- Acquisition techniques for SS signal
  - Serial acquisition
    - Time cost is high
  - Parallel acquisition
    - Fast Acquisition, but expensive by using multiple FPGAs/DSPs
    - Difficult in programming
Motivation

- Serial acquisition
  - Serial acquisition on a single PC or workstation is slow, e.g. 19.552 seconds per 300 microseconds’ bit stream
  - Computational complexity is high
    - \( O(NL*NA*\text{pointCodeN}*\text{channels}) \)
    - E.g. \( NL = 40, NA = 5600, \text{pointCodeN} = 2810, \# \text{of channels}= 8 \), 16711.8164 million floats operations per second in 56M sample rate
    - A continuous data stream, possibly infinite

- Real-time acquisition of SS signal is critical
  - Real-time acquisition is in demand
Motivation

- Parallel acquisition is a promising solution
  - Data parallel fashion
  - Very high parallelism

- Software Radio —— New trend in satellite communication
  - Low cost
  - Easy to program
  - Easy to update
  - Scalable
  - Compatible

Suitable for CUDA parallelization
Motivation

- Accelerate the algorithm for acquisition of spread spectrum signal using CUDA-enabled GPUs
  - Identify the computation bottleneck, *sliding correlation*
  - Propose an efficient parallel scheme for sliding correlation kernel on CUDA
  - Apply various CUDA optimization techniques
  - Implement a CUDA-enabled acquisition algorithm on multiple GPUs
Motivation

- Experimental results present up to 473X speedup comparing with the execution time on CPU with IPP
- Real-time acquisition is achieved in all cases
- Good scalability is observed
for(int ch = 0; ch < # of channels; ch++){
    for(int i = 0; i < needLen; i++){
        localGold[i] = pnData[(int)floor(fmod(((float)pnRate*(1.0+(channels[ch])/ft*1)*i/fs),1023.0))];
        mix_I[i] = sin(2*pi*(f0+channels[ch])*i/fs+phase0)*sig[i];
        mix_Q[i] = cos(2*pi*(f0+channels[ch])*i/fs+phase0)*sig[i];
    }
    for(int j = 0; j < pointCodeN; j++){
        for(int t = 0; t < NL * NA; t++){
            mi_I[t] = localGold[t] * mix_I[pointCode*j + t];
            mi_Q[t] = localGold[t] * mix_Q[pointCode*j + t];
        }
        for(int k2 = 0; k2 < NL; k2++){
            for(int k1 = 0; k1 < N; k1++){
                tmpI = 0.0;
                tmpQ = 0.0;
                n = k1*sumL + k2*NA;
                for(int p = n; p < n + sumL; p++){
                    tmpI += mi_I[p];
                    tmpQ += mi_Q[p];
                }
                data[NL*N*j+k1+k2*N] = tmpI;
                dataQ[NL*N*j+k1+k2*N] = tmpQ;
            }
        }
    }
}

//… ….next page
Acquisition Algorithm (Contd.)

```c
for(int ch = 0; ch < # of channels; ch++){
    //…Continue from last page
    ippsFFTFwd_CToC_32f(dataI, dataQ, yR, yI, spec, NULL);
    for(int e = 0; e < Nfft*NL*pointCodeN; e++){
        yR[e] = sqrt(yR[e] * yR[e] + yI[e] * yI[e]);
    }
    for(int q = 0; q<pointCodeN;q++){
        for(int t = 0; t < Nfft; t ++){
            for(int j = 0; j < NL; j ++){
                Rout[q*Nfft+t] += yR[q*Nfft*NL + j*Nfft + t];
            }
        }
    }
    // select the max amplitude from Rout, then its index (position) is acquired.
}
```

\[O(NL*NA*pointCodeN*channels)\]
Parallel Acquisition Algorithm on CUDA

- High parallelism exists
  - Mixing: needLen
  - Sliding correlation: pointCodeN*N*NL
  - FFT-based acquisition: pointCodeN*Nfft

- Three CUDA kernels
  - Mixing: operations on vector elements are independent
  - Sliding correlation: massive number of independent dot products → hotspot!
  - FFT-based acquisition: amplitude calculations and selections are independent
Parallel Acquisition Algorithm on CUDA

- **Kernel 1: Mixing**
  - Each thread calculates three values in three vectors
    - `localGold`
    - I branch of *Mixing* signal
    - Q branch of *Mixing* signal
  - `needLen` threads in total

```c
for(int i = 0; i < needLen; i++){
    localGold[i] = pnData[(int)floor(fmod(((float)pnRate*(1.0+(channels[ch])/ft*1)*i/fs),1023.0))];
    mix_I[i] = sin(2*pi*(f0+channels[ch])*i/fs+phase0)*sig[i];
    mix_Q[i] = cos(2*pi*(f0+channels[ch])*i/fs+phase0)*sig[i];
}
```
Parallel Acquisition Algorithm on CUDA

Kernel 2: sliding correlation

- Each thread calculates a dot product of $sumL$ pairs of points, $pointCodeN*N*NL$ threads in total
- Different threads in a block take different segments of $mix_I/Q$
- Load the current segment into the shared memory in an SM
- All the threads in a block share a common part of the $localGold$, load the current part into the shared memory in an SM
Parallel Acquisition Algorithm on CUDA

Block 0

thread 0 thread 1 thread 2 …… thread n

mix_I/Q

dot product

localGold

for another block

dataI/Q
Parallel Acquisition Algorithm on CUDA

- **Kernel 3: FFT-based acquisition**
  - Perform FFT on \(data_I\) and \(data_Q\) by invoking CUFFT in batch mode
  - Each thread computes the modulus for \(NL\) points, and accumulates the moduli
  - \(Nfft*pointCodeN\) threads in total

```c
for(int e = 0; e < Nfft*NL*pointCodeN; e++){
    yR[e] = sqrt(yR[e] * yR[e] + yI[e] * yI[e]);
}
for( int q = 0; q<pointCodeN;q++){
    for(int t = 0; t < Nfft; t ++){
        for(int j = 0; j < NL; j ++){
            Rout[q*Nfft+t] += yR[q*Nfft*NL + j*Nfft + t];
        }
    }
}
```
Parallel Acquisition Algorithm on CUDA

- CUDA-enabled acquisition algorithm
  - For each channel
    - Call *mixing* kernel
    - Call *sliding correlation* kernel
    - Call CUFFT, calculate the amplitudes
    - Select the largest amplitude and the index of the corresponding reference sequence
  - Each block allocates shared memory for *branch I*, *branch Q* and *localGold*
  - If the shared memory allocated overwhelms the capacity, *blockSize* should be reduced

\[ \{ \text{GPU side} \rightarrow \text{CPU side} \]
CU-Acqss on a Single GPU

- In each channel
  - Three kernels are performed on the GPU successively
  - The maximum value is selected on the CPU

- The computation for different channels is performed on the GPU successively
CU-Acqss on Multiple GPUs

- Distribute the computation for different channels to different GPUs
- Workload partition schema on 4 GPUs
  - E.g. Round robin
Performance Evaluation

- Hardware platform
  - Xeon CPU E5-2620 v2 @ 2.10GHz (2 processors), 16.0GB RAM
  - NVIDIA’s Tesla K40 card with 2880 SPs and 12GB GDDR5

- Software
  - Windows Server 2008 R2 Standard
  - Visual Studio 2010, CUDA 7.5

- Datasets
  - 6 sets of data from a real spread spectrum telemetry system with 56M sampling rate
  - 300 microseconds, each set containing 8192*2048 floats
Performance Evaluation

Parameters of each data set

<table>
<thead>
<tr>
<th>Data set #</th>
<th># of channels</th>
<th>pointCodeN</th>
<th>NA</th>
<th>NL</th>
<th>N</th>
<th>sumL</th>
<th>pointCode</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
<td>2074</td>
<td>6222</td>
<td>40</td>
<td>64</td>
<td>97</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>2810</td>
<td>5600</td>
<td>40</td>
<td>64</td>
<td>87</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>2334</td>
<td>7000</td>
<td>50</td>
<td>64</td>
<td>109</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>2667</td>
<td>8000</td>
<td>40</td>
<td>64</td>
<td>125</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>2340</td>
<td>9333</td>
<td>40</td>
<td>64</td>
<td>145</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>15</td>
<td>2280</td>
<td>11200</td>
<td>30</td>
<td>64</td>
<td>175</td>
<td>5</td>
</tr>
</tbody>
</table>

Counterpart: Integrated Performance Primitives (IPP) was used in the implementation on CPU
## Performance Evaluation

### Speedups on a single K40

<table>
<thead>
<tr>
<th>Data Set #</th>
<th>CPU (ms)</th>
<th>CU-AcqSS (ms)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8,018</td>
<td>58.232</td>
<td>138</td>
</tr>
<tr>
<td>2</td>
<td>10,405</td>
<td>96.183</td>
<td>108</td>
</tr>
<tr>
<td>3</td>
<td>13,713</td>
<td>97.092</td>
<td>141</td>
</tr>
<tr>
<td>4</td>
<td>15,756</td>
<td>149.429</td>
<td>105</td>
</tr>
<tr>
<td>5</td>
<td>18,564</td>
<td>273.068</td>
<td>68</td>
</tr>
<tr>
<td>6</td>
<td>19,750</td>
<td>183.155</td>
<td>109</td>
</tr>
</tbody>
</table>

Real-time acquisition was achieved in all cases!
Performance Evaluation

- **Sliding correlation** kernel took more than 80% of the GPU time
- **GPU occupancy**

<table>
<thead>
<tr>
<th>Data set #</th>
<th>Theoretical</th>
<th>Achieved</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>75%</td>
<td>68.28%</td>
</tr>
<tr>
<td>2</td>
<td>100%</td>
<td>98.32%</td>
</tr>
<tr>
<td>3</td>
<td>75%</td>
<td>69.69%</td>
</tr>
<tr>
<td>4</td>
<td>75%</td>
<td>72.26%</td>
</tr>
<tr>
<td>5</td>
<td>50%</td>
<td>47.13%</td>
</tr>
<tr>
<td>6</td>
<td>37.5%</td>
<td>37.19%</td>
</tr>
</tbody>
</table>

GPU occupancy per SM in *sliding correlation* kernel!
Performance Evaluation

pointCodeN=2810, NL=40, NA=5600

<table>
<thead>
<tr>
<th># of channels</th>
<th>CPU (ms)</th>
<th>CU-AcqSS (ms)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2,902</td>
<td>23.804</td>
<td>122</td>
</tr>
<tr>
<td>4</td>
<td>5,382</td>
<td>47.529</td>
<td>113</td>
</tr>
<tr>
<td>6</td>
<td>7,940</td>
<td>71.131</td>
<td>112</td>
</tr>
<tr>
<td>8</td>
<td>10,265</td>
<td>94.797</td>
<td>108</td>
</tr>
<tr>
<td>10</td>
<td>12,964</td>
<td>120.256</td>
<td>108</td>
</tr>
<tr>
<td>12</td>
<td>15,335</td>
<td>142.137</td>
<td>108</td>
</tr>
</tbody>
</table>

Good scalability when varying # of channels!
## Performance Evaluation

#of channels=8, pointCodeN=2810, NA=5600

<table>
<thead>
<tr>
<th>NL</th>
<th>CPU (ms)</th>
<th>CU-AcqSS (ms)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>2,730</td>
<td>26.878</td>
<td>101</td>
</tr>
<tr>
<td>20</td>
<td>5,336</td>
<td>49.428</td>
<td>108</td>
</tr>
<tr>
<td>30</td>
<td>8,237</td>
<td>72.053</td>
<td>114</td>
</tr>
<tr>
<td>40</td>
<td>11,310</td>
<td>94.514</td>
<td>120</td>
</tr>
<tr>
<td>50</td>
<td>12,948</td>
<td>116.916</td>
<td>111</td>
</tr>
<tr>
<td>60</td>
<td>15,834</td>
<td>139.782</td>
<td>113</td>
</tr>
</tbody>
</table>

Good scalability when varying $NL$!
# Performance Evaluation

## #of channels=8, NL=40, NA=5600

<table>
<thead>
<tr>
<th>Point CodeN</th>
<th>CPU (ms)</th>
<th>CU-AcqSS (ms)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>700</td>
<td>3,495</td>
<td>27.152</td>
<td>129</td>
</tr>
<tr>
<td>1400</td>
<td>5,726</td>
<td>49.626</td>
<td>115</td>
</tr>
<tr>
<td>2100</td>
<td>8,237</td>
<td>74.188</td>
<td>111</td>
</tr>
<tr>
<td>2800</td>
<td>10,296</td>
<td>94.291</td>
<td>109</td>
</tr>
<tr>
<td>3500</td>
<td>12,842</td>
<td>116.822</td>
<td>110</td>
</tr>
<tr>
<td>4200</td>
<td>15,335</td>
<td>139.575</td>
<td>110</td>
</tr>
</tbody>
</table>

Good scalability when varying *pointCodeN*!
### Performance Evaluation

#### Speedups on four K40 GPUs

<table>
<thead>
<tr>
<th>Data Set #</th>
<th>CPU (ms)</th>
<th>CU-AcqSS (ms)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8,018</td>
<td>16.947</td>
<td>473</td>
</tr>
<tr>
<td>2</td>
<td>10,405</td>
<td>26.297</td>
<td>396</td>
</tr>
<tr>
<td>3</td>
<td>13,713</td>
<td>29.385</td>
<td>467</td>
</tr>
<tr>
<td>4</td>
<td>15,756</td>
<td>49.891</td>
<td>316</td>
</tr>
<tr>
<td>5</td>
<td>18,564</td>
<td>74.105</td>
<td>251</td>
</tr>
<tr>
<td>6</td>
<td>19,750</td>
<td>56.985</td>
<td>347</td>
</tr>
</tbody>
</table>

yingliu@ucas.ac.cn

2016/09
Performance Evaluation

- Speedups using 2, 3 and 4 GPUs

![Graph showing speedups using 2, 3 and 4 GPUs across different numbers of channels. The graph includes lines for 2 GPU, 3 GPU, and 4 GPU configurations, illustrating the speedup ratio for each scenario.]
Performance Evaluation

- Speedups with varying # of channels using multiple GPUs
Conclusion

- Identified the computational hotspot of acquisition algorithm for spread spectrum signal, *sliding correlation*
- Proposed an efficient scheme to accelerate *sliding correlation* kernel on CUDA
- Implemented the acquisition algorithm, CU-AcqSS, on CUDA
- Implemented CU-AcqSS on multiple GPUs
Conclusion

- Performance was evaluated on 6 sets of data
- Good speedups were observed for all data sets
- Good scalability was observed when varying the computation
- Real-time acquisition was achieved in all cases