C++/CUDA Implementation of the Weeks Method for Numerical Laplace Transform Inversion

Acunum Algorithms and Simulations, LLC
www.acunum.com
contact@acunum.com

Patrick Kano & Moysey Brio

May 17, 2011

Introduction

While tables of forward and inverse Laplace transforms are commonly used for textbook exercises, real world engineering and scientific problems often present Laplace space solutions \( F(s) \) that cannot be inverted by standard methods. The numerical inversion of the Laplace transform is an obvious approach to mitigating this problem. Numerical inversion is however known to be a paradigm of an exponentially ill-posed problem [2] and thus requires careful consideration. Many algorithms have been presented to perform the numerical inversion [3]. Implemented here is one of the more well known, the Weeks method [6]. This software has been written to take advantage of the parallel computations that are available on modern graphics processing units.

Description of the Weeks Method

The Weeks method involves a Laguerre polynomial expansion of the time domain function \( f(t) \).

\[
L_n(x) = \frac{e^x}{n!} \frac{d^n}{dx^n} (x^n e^{-x})
\]

\[
f(t) \approx \sum_{n=0}^{N-1} a_n e^{(\sigma-b)t} L_n(2bt)
\]

The expansions coefficients are dependent on the Laplace space function \( F(s) \) where \( s \) is a complex variable.

\[
a_n = a_n(F(s), \sigma, b)
\]

The coefficients are computed by performing an contour integration in the \( w \)-plane on a unit circle that encloses the singularities of the mapped \( F(w) \).

\[
a_n = \frac{1}{2\pi i} \int_{|w|=1} w^{-(n+1)} \frac{2b}{1-w} F\left(\sigma - b \frac{w+1}{w-1}\right) dw
\]

By sampling \( F(w) \) on the unit circle at a uniform angular distribution, the numerical integration via a midpoint rule is equivalent to a discrete Fourier Transform.

\[
w = e^{i\theta}
\]

\[
s = \sigma - b \frac{w+1}{w-1}
\]

The core computation of the Weeks method Laguerre expansion is thus to compute the coefficients via a Fast Fourier transform of samples of the Laplace space function \( F(s) \) that has been mapped with a Moebius transformation from the \( s \) to
the $w$ plane. The choice of the mapping parameters is obviously critical.

**Parameter Selection**

The Weeks method includes two free positive scaling parameters ($\sigma$, $b$). By appropriately choosing these parameters, an accurate inversion can be obtained. The parameter selection can be accomplished from a minimization of an absolute error estimate $E(t; \sigma, b)$ for the approximate $f(t)$. The calculation of the error estimate involves the determination of the expansion coefficients and thus a fast Fourier transform for each ($\sigma, b$) pair. This direct global error estimate minimization is a computationally intensive approach. It is however a robust method [5] and the reliability of the inversion is paramount.

**GPU Acceleration**

New in this software is the use of graphics processing units [GPU] to calculate in parallel the error estimate for each ($\sigma$, $b$) pair [4]. The code is fundamentally a standard C++ class called *WeeksNLAP*. However, for the coefficient computations, NVIDIA’s Compute Unified Device Architecture [CUDA] is used [1]. With CUDA, C kernel functions are included to implement the parallel sampling and Fast Fourier transforms on the GPU. This error estimation problem is well suited to GPU implementation in that it involves multiple executions of the same instructions. The graphics card contains many more cores than the central processing unit and is therefore able to compute in parallel far more of these similar commands. To be concise, the C-GPU functions are primarily used to perform in parallel the $F(s)$ sampling and FFT for each ($\sigma$, $b$).

From these, a minimum error estimate is obtained and an optimal ($\sigma$, $b$) pair used in the Weeks expansion.

**User Interface**

To assist the user, the interface to the code has been reduced to two files.

1. **UserFs.txt** Provides the definition of $F(s)$ to invert
2. **UserParameters.txt** Provides the user controlled computation parameters

Currently, the code must be recompiled for each $F(s)$ expression that the user defines in UserFs.txt. $F(s)$ must also be written using CUDA’s cuDoubleComplex variables.

The parameters file UserParameters.txt is read in the WeeksNLAP class constructor when an object is created. The format of the file can not be changed. The contents are described in this table:

<table>
<thead>
<tr>
<th>Type</th>
<th>Parameter</th>
<th>Description</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer</td>
<td>LengthTimes</td>
<td>Number of inversion times.</td>
<td>This vector must have a length equal to LengthTimes</td>
</tr>
<tr>
<td>double</td>
<td>*Timevec</td>
<td>The inversion times.</td>
<td></td>
</tr>
<tr>
<td>double</td>
<td>*RelErrorTol</td>
<td>Relative error tolerance for the inversions.</td>
<td>0.x is an x% relative error</td>
</tr>
<tr>
<td>integer</td>
<td>SearchSwitch</td>
<td>A switch to allow the user to directly specify the ($\sigma$, $b$) parameters or allow the code to perform an internal search. 1-search for parameters</td>
<td>The user can specify the scaling parameters directly without a search.</td>
</tr>
<tr>
<td>Type</td>
<td>Data</td>
<td>Description</td>
<td>Comment</td>
</tr>
<tr>
<td>----------</td>
<td>--------------</td>
<td>-------------------------------------------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>double</td>
<td>time</td>
<td>Inversion time supplied by the user.</td>
<td></td>
</tr>
<tr>
<td>double</td>
<td>Invertf</td>
<td>Approximate f(t)</td>
<td></td>
</tr>
<tr>
<td>double</td>
<td>sigmaP</td>
<td>(\sigma) used in the inversion</td>
<td>from the search or defined by the user</td>
</tr>
<tr>
<td>double</td>
<td>bP</td>
<td>(b) used in the inversion</td>
<td>from the search or defined by the user</td>
</tr>
<tr>
<td>double</td>
<td>AbsTotalError</td>
<td>Estimated absolute truncation and round-off error</td>
<td>0.x is an x% relative error</td>
</tr>
<tr>
<td>double</td>
<td>RelTotalError</td>
<td>Estimated relative truncation and round-off error using the approximate f(t)</td>
<td>0.x is an x% relative error</td>
</tr>
<tr>
<td>integer</td>
<td>ToleranceMetFlag</td>
<td>A flag to indicate if the estimated relative error is below the user specified tolerance.</td>
<td>1=Failure 0=Success</td>
</tr>
</tbody>
</table>
The estimated error is clearly at the limits of double precision accuracy.

The total execution time in seconds for the code to invert at sixteen times is shown in the following table as a function of the number of expansion coefficients and grid resolution. The code is run with \( \sigma \) from \([0,20)\) and \( b \) from \([0,40)\).

<table>
<thead>
<tr>
<th>( N ) coefficients</th>
<th>( b ) resolution</th>
<th>( \sigma ) resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td>16</td>
<td>2.05</td>
<td>3.28</td>
</tr>
<tr>
<td>0.5</td>
<td>3.28</td>
<td>5.75</td>
</tr>
<tr>
<td>0.25</td>
<td>5.77</td>
<td>10.69</td>
</tr>
<tr>
<td>32</td>
<td>1</td>
<td>3.3</td>
</tr>
<tr>
<td>1</td>
<td>3.3</td>
<td>5.77</td>
</tr>
<tr>
<td>0.5</td>
<td>5.77</td>
<td>10.72</td>
</tr>
<tr>
<td>0.25</td>
<td>10.73</td>
<td>20.61</td>
</tr>
<tr>
<td>64</td>
<td>1</td>
<td>5.79</td>
</tr>
<tr>
<td>0.5</td>
<td>10.75</td>
<td>20.66</td>
</tr>
<tr>
<td>0.25</td>
<td>20.67</td>
<td>40.48</td>
</tr>
</tbody>
</table>

The run time doubles with double resolution.

Focusing on the GPU implemented computations yields the run times in the next table. These numbers are in seconds and for the inversion at a single time. They were captured using the `cudaEvent` timing mechanism in CUDA. An observation from this data is that the parallel FFT on the GPU is only a fraction of the total run time. Rather, the F(s) sampling and coefficients computations are much more time consuming.
B) The second example involves a numerical inversion to obtain the zeroth Bessel function. The function’s value at $t=10$ is 
\[ F(s) = \frac{1}{\sqrt{s^2 + 1}} \leftrightarrow f(t) = J_0(t) \]

\[ F(s) = \frac{1}{\sqrt{s^2 + 1}} \leftrightarrow f(t) = J_0(t) \]

C) The third example can be analytically inverted using tables and partial fractions. The expression for $f(t)$ is purely harmonic and has a value at $t=1$ of 0.4851788474. Again, one sees in the screenshot that the code accurately captured the correct approximate $f(t)$.

\[ F(s) = \frac{1}{s(s^2 + 9/25)} \leftrightarrow f(t) = (25/9)(1 - \cos(3t/5)) \]
D) This fourth example involves fraction exponents. The time domain function includes a coefficient that is defined in terms of the Gamma function. At time \( t=2 \), the time domain function is 0.23202351899818. The code accurately estimates this value to within 0.6\% relative error.

\[
F(s) = \frac{1}{(s + 3/5)^{9/11}} \quad \leftrightarrow \quad f(t) = \frac{1}{t^{2/11}e^{30/5I\Gamma(9/11)}}
\]

Conclusion
This software demonstrates the use of GPU acceleration for optimal parameter selection in a numerical Laplace transform inversion algorithm. The Weeks method is straightforward to parallelize and thus is a good candidate for GPU computations. The required fast Fourier transforms are also well implemented by NVIDIA on the GPU. More importantly, the algorithm actually performs on the tested problems despite the well known inherent ill-posedness of the numerical inverse Laplace transform.

The software is sufficient for illustration purposes but could be expanded and refined into a more user-friendly form. The codes are intended for adaption and use in other programs and tools. If they are used, a reference and e-mail to Acunum would greatly appreciated. The files are provided 'as is' with no guarantees under a standard BSD license.
References

The Bad Truth about Laplace's Transform.

[3] Patrick Kano, Moysey Brio, and Jerome V. Moloney
Application of Weeks Method for the Numerical Inversion of the Laplace Transform to the Matrix Exponential
Communications in Mathematical Sciences
Volume 3, Number 3, pp. 335-372, 2005
www.math.arizona.edu/~brio/WEEKS_METHOD_PAGE/pkanoWeeksMethod.html

[4] Patrick Kano, Moysey Brio
Weeks' Method for Numerical Laplace transform inversion with GPU acceleration

Algorithms for Parameter Selection in the Weeks Method for Inverting the Laplace Transform
SIAM Journal on Scientific Computing
Volume 21 Issue 1, Aug.-Sept. 1999
http://dip.sun.ac.za/~weideman/research/weeks.html

Numerical Inversion of Laplace Transforms Using Laguerre Functions
Journal of the ACM (JACM)
Volume 13 Issue 3, July 1966

Appendix – CUDA Setup
Two sets of code are supplied; one is for Windows with Visual Studio and the other is intended for Linux-like environments. The core source and header files are identical in both sets. The two cases reflects the different ways that the codes have been compiled. They were written on a ASUS N53S series laptop with Windows 7, an Intel i7 central processor, and an NVIDIA GeForce GT 540M graphics processor. This GPU has an NVIDIA compute capability of 2.1 and 96 cores.

Visual Studio
The codes have been compiled and tested with Visual Studio 2008. Setting up Visual Studio and CUDA to interface correctly on the laptop has been a nontrivial task. A useful reference is here:
http://www.codeproject.com/KB/GPU-Programming/visualstudioCUDA.aspx

A few more helpful comments are:
• Download and install Visual Studio 2008 (not 2010). Our own personal experience has shown that the 2008 works well with CUDA 3.2; see the blog:

• Download and install the NVIDIA toolkit 3.2. A 4.0 release is available but was not used to develop this code.

• Download and install the NVIDIA software development kit 3.2. It is also from:

The included VS project has the CUDA configuration and properties set to work with a GPU with compute capability 2.1. It was however necessary to manually place some files and configure the visual studio project.
• add : C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\VCProjectDefaults\NvCudaDriverApi.v3.2

• add : C:\Program Files (x86)\Microsoft Visual Studio 9.0\VC\VCProjectDefaults\NvCudaRuntimeApi.v3.2

• Within the VS project add the build rules with: project -> Custom Build Rules-> CUDA Driver API Build Rule 3.2

• add manually : Tools->Options->Projects and Solutions->VC++ Project Settings ->
  C:\(the path)\NVIDIASOFTWARE\cudatoolkit_3.2_win_buildrules-patch

• add manually : Tools->Options->Projects and Solutions->VC++ Directories ->
  1. Executable files: C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v3.2\bin
  2. Include files:
     a. C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v3.2\include
     b. C:\ProgramData\NVIDIA Corporation\NVIDIA GPU Computing SDK 3.2\C\common\inc
  3. Library files
     a. C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v3.2\lib\Win32
     b. C:\ProgramData\NVIDIA Corporation\NVIDIA GPU Computing SDK 3.2\C\common\lib

**Cygwin**

To test compatibility, the codes were also compiled and run on the laptop using Cygwin

http://www.cygwin.com/

A MakeFile is provided for this purpose. It is simpler to use the MakeFile rather than Visual Studio. To create the executable, one simply uses on the command line the standard 'make'. 'make clean' is also provided to remove the object files. It is useful to 'make clean' when the 'UserFs.txt' file has been modified before one re-compiles for a new F(s). The MakeFile is currently setup to use the Windows 'cl' compiler.