Accelerating an Iterative Helmholtz Solver Using Reconfigurable Hardware

Art Petrenko
M.Sc. Defence, April 9, 2014
Seismic Laboratory for Imaging and Modelling
Department of Earth, Ocean and Atmospheric Sciences, UBC
Oh by the way: I have a stutter.
Seismic Wave Simulation
Seismic Exploration for Oil and Gas
Full-waveform Inversion

Seismic Wavefield ($u$)  
Earth model ($m$)
Full-waveform inversion is SLOW
The Accelerators Have Arrived

Top 10 of “Top 500” Supercomputers

<table>
<thead>
<tr>
<th>Rank</th>
<th>Site</th>
<th>System</th>
<th>Corres</th>
<th>Rmax (TFlop/s)</th>
<th>Rpeak (TFlop/s)</th>
<th>Power (KW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>National Super Computer Center in Guangzhou, China</td>
<td>Tianhe-2 (MilkyWay-2) - TH-IBF-FEP Cluster, Intel Xeon E5-2692 12C 2.20GHz, Intel Express-2, Intel Xeon Phi 31S1P, NVIDIA K20x</td>
<td>3,120,000</td>
<td>33,952.2</td>
<td>54,920.4</td>
<td>17,808</td>
</tr>
<tr>
<td>2</td>
<td>DOE/SC/Oak Ridge National Laboratory, United States</td>
<td>Titan - Cray XK7, Opteron 6274 16C 2.20GHz, Cray Gemini interconnected, NVIDIA K20x</td>
<td>560,840</td>
<td>17,596.0</td>
<td>27,112.5</td>
<td>8,208</td>
</tr>
<tr>
<td>3</td>
<td>DOE/NSA/LNL, United States</td>
<td>Sequoia - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM</td>
<td>1,072,864</td>
<td>17,117.3</td>
<td>20,112.7</td>
<td>7,990</td>
</tr>
<tr>
<td>4</td>
<td>RIKEN Advanced Institute for Computational Science (AICS), Japan</td>
<td>K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnected, Fujitsu</td>
<td>705,024</td>
<td>10,510.0</td>
<td>11,280.4</td>
<td>12,680</td>
</tr>
<tr>
<td>5</td>
<td>DOE/SC/Acorn National Laboratory, United States</td>
<td>Mira - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM</td>
<td>786,432</td>
<td>8,596.8</td>
<td>10,066.3</td>
<td>3,945</td>
</tr>
<tr>
<td>6</td>
<td>Swiss National Supercomputing Centre (CSCS), Switzerland</td>
<td>Par Diatom - Cray XC30, Xeon E5-2670 8C 2.60GHz, Aries interconnected, NVIDIA K20x, Cray Inc.</td>
<td>115,984</td>
<td>6,271.0</td>
<td>7,788.9</td>
<td>2,325</td>
</tr>
<tr>
<td>7</td>
<td>Texas Advanced Computing Center/Univ. of Texas, United States</td>
<td>Stampede - PowerEdge C8220, Xeon E5-2680 8C 2.70GHz, Intel Sandy Bridge, Intel Xeon Phi 5110P Dell</td>
<td>462,482</td>
<td>5,108.1</td>
<td>6,025.1</td>
<td>4,510</td>
</tr>
<tr>
<td>8</td>
<td>Forschungszentrum Juelich (FZJ), Germany</td>
<td>Juqueen - BlueGene/Q, Power BQC 16C 1.60GHz, Custom interconnected, IBM</td>
<td>458,752</td>
<td>5,006.9</td>
<td>6,872.0</td>
<td>2,301</td>
</tr>
<tr>
<td>9</td>
<td>DOE/NSA/LNL, United States</td>
<td>Vulcan - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom interconnected, IBM</td>
<td>392,248</td>
<td>5,000.0</td>
<td>6,023.2</td>
<td>1,972</td>
</tr>
<tr>
<td>10</td>
<td>Leibniz Rechenzentrum, Germany</td>
<td>SuperMUC - iDataPlex DX8600M4, Xeon E5-2680 8C 2.70GHz, Infiniband FDR, IBM</td>
<td>147,456</td>
<td>2,697.0</td>
<td>3,165.1</td>
<td>3,423</td>
</tr>
</tbody>
</table>
FPGAs: Reconfigurable Hardware Accelerators
The Punchline
Modelling Seismic Waves
Mathematical Formulation
Modelling Seismic Waves: The Wave Equation

\[
\left( \omega^2 m + \Delta \right) u = q
\]

- Frequency
- Laplacian
- Seismic Source (right-hand side)

Earth Model
Wavefield (iterate)

The Helmholtz system: Discretizing the wave equation

When simulating waves in the frequency domain, the PDE that describes the motion of the wave through a heterogeneous medium can be written as

\[
\frac{1}{v^2} \nabla^2 u = q,
\]

(2.1)

and is known as the Helmholtz equation. As written above, the Helmholtz equation represents the special case of a constant density isotropic medium which only supports acoustic waves. Damping effects of viscosity are modelled heuristically by allowing \( m \) to be complex-valued. I ignore the case of elastic and anisotropic media to keep the resulting implementation relatively simple. The symbol \( \nabla^2 \) represents the Laplacian operator. I take the subsurface earth model to be the slowness squared, \( m = \frac{1}{v^2} \), where \( v \) is the sound speed of the medium. \( u \) is the (complex) Fourier transform of the pressure with respect to time, and \( q \) is the amplitude of the source at angular frequency \( \omega \). The Laplacian operator here also implements perfectly matched layer (PML) boundary conditions that eliminate reflection artefacts from the boundaries of the domain by setting a damping layer consisting of complex velocities. (See Equation (2) in the work by Operto et al. [42].) In the frequency domain, Equation 2.1 must be solved for each frequency \( \omega \) that is to contribute to the final wavefield \( u \) for a given source.
Modelling Seismic Waves: Discretization
[Operto, 2007]

\[ A(m, \omega)u = q \]
Solving the Helmholtz System
Despite the advantages mentioned in the last section, the Kaczmarz algorithm (SSOR-NE) converges slowly, thus it is not suitable for direct application to the Helmholtz system (Equation 2.2). Instead, Björck and Elfving [4] showed that it can be used to accelerate the method of conjugate gradients, calling the resulting algorithm CGMN. Recent studies of how CGMN fares in solving the Helmholtz equation include work by van Leeuwen [57] and Gordon and Gordon [18]. In the latter case, CGMN is equivalent to the sequential (non-parallel, running on only one processor core) version of the algorithm CARP-CG, introduced by Gordon and Gordon [17]. I now describe the CGMN algorithm.

First, it is useful to represent the Kaczmarz sweeps in matrix notation. Following Tanabe [55], let 

\[ Q_i = I - \frac{a_i a_i^T}{a_i^T a_i} \]  

The double sweep can then be written as

\[ DKSWP(A, u, q, \lambda) = Q_1 \cdots Q_N Q_N \cdots Q_1 u + Rq = Qu + Rq. \]  

(2.5)

Since \( A \) is invertible, SSOR-NE will converge to the solution of Equation 2.2, as mentioned by Björck and Elfving [4]. At that point, the iterate \( u \) will be a fixed point of the relation 2.4, which means that Equation 2.5 can be re-written as a linear system:

\[ (I - Q)u = Rq, \]  

(2.6)

where \( I \) is the identity matrix. As mentioned by Björck and Elfving [4] and proved by, for example, Gordon and Gordon [17], the system in Equation 2.6 is consistent, symmetric and positive semi-definite. Björck and Elfving [4] show in their Lemma 5.1 that this is sufficient for CG to converge to the pseudoinverse (minimum \( \lambda \)-norm) solution of Equation 2.6, which is the same as the solution of the original system (Equation 2.2). Note that the matrices \( Q \) and \( R \) do not have to be formed explicitly, as their action on a vector is calculated using a double Kaczmarz sweep, as in Equation 2.5.

Thus, CGMN is the use of the method of conjugate gradients to solve the SSOR-NE iteration system (Equation 2.6) for the fixed point of that iteration. SSOR-NE is implemented efficiently using Kaczmarz row projections. Björck and Elfving [4] also note that it is possible to view CGMN as solving a variant of the Helmholtz system (Equation 2.2), preconditioned from the left by a matrix derived from a decomposition of \( AA^T \).

Pseudo-code for the CGMN algorithm is given below. Note that the double Kaczmarz sweep on line 1 of the algorithm is performed with an initial guess of zero because only the action
The Kaczmarz Algorithm: Equivalent to SSOR-NE

[Björck and Elfving, 1979]

Double Kaczmarz sweep on the original system:

\[ Au = q \]

One iteration of SSOR on the normal equations:

\[ AA^*y = q \]
\[ A^*y = u \]

Both are computed as:

\[ u_{k+1} = u_k + \lambda (b_i - \langle a_i, u_k \rangle) \frac{a_i^*}{\|a_i\|^2} \]

\[ k : 1 \rightarrow 2N \]
\[ i : 1 \rightarrow N, N \rightarrow 1 \]
Kaczmarz + CG = CGMN
[Björck & Elfving 1979]
CGMN: Solves for Fixed Point of Kaczmarz Row Projections

\[
\text{DKSWP}(A, u, q, \lambda) = Q_1 \cdots Q_N Q_N \cdots Q_1 u + Rq \\
= Qu + Rq.
\]

Assume \(u\) is a solution and re-arrange:

\[
(I - Q)u = Rq
\]
Contribution of This Work
Remark that it is possible to perform more than one double Kaczmarz sweep, one after the other, in CGMN (line 5 of Algorithm 1). If the Kaczmarz sweeps are viewed as a preconditioner for CG, as mentioned above, multiple sweeps correspond to a more exact preconditioner. Fewer outer CGMN iterations would have to be performed, at the cost of several invocations of $\text{DKSWP}$ in the main while loop. Because the Kaczmarz algorithm takes more sweeps to converge to a solution than CG takes iterations, such a trade-off might be viewed as exchanging a precise tool for a less precise one, and hence undesirable. However as I show in Section 4.1, the characteristics of the Kaczmarz algorithm as implemented on an FPGA-based accelerator might encourage such an adjustment.

In this chapter I have described the mathematical formulation of the approach I take to solving the seismic wave simulation problem. The reader will recall that this problem is relevant because of the role it plays in full-waveform inversion, a key algorithm in seismic exploration. The next chapter will focus on a description of the implementation of this problem to an FPGA-based hardware accelerator.

Algorithm 1 CGMN (Björck and Elfving [4])

<table>
<thead>
<tr>
<th>Input: $A, u, q, \lambda$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: $Rq \leftarrow \text{DKSWP}(A, 0, q, \lambda)$</td>
</tr>
<tr>
<td>2: $r \leftarrow Rq - u + \text{DKSWP}(A, u, 0, \lambda)$</td>
</tr>
<tr>
<td>3: $p \leftarrow r$</td>
</tr>
<tr>
<td>4: while $|r|^2 &gt; tol$ do</td>
</tr>
<tr>
<td>5: $s \leftarrow (I - Q)p = p - \text{DKSWP}(A, p, 0, \lambda)$</td>
</tr>
<tr>
<td>6: $\alpha \leftarrow |r|^2 / \langle p, s \rangle$</td>
</tr>
<tr>
<td>7: $u \leftarrow u + \alpha p$</td>
</tr>
<tr>
<td>8: $r \leftarrow r - \alpha s$</td>
</tr>
<tr>
<td>9: $\beta \leftarrow |r|^2_{\text{curr}} / |r|^2_{\text{prev}}$</td>
</tr>
<tr>
<td>10: $|r|^2_{\text{prev}} \leftarrow |r|^2_{\text{curr}}$</td>
</tr>
<tr>
<td>11: $p \leftarrow r + \beta p$</td>
</tr>
<tr>
<td>12: end while</td>
</tr>
</tbody>
</table>

Output: $u$
Low levels of abstraction are scary
Design at high level of abstraction

```java
 x[0]^R[26]};
DFEVar relaxationFactor = io.scalarInput("relaxationFactor", KaczmarzEngineCode.KaczmarzWriteMemKernel.TruncatedFloatingPoint);
DFEComplex kaczmarz_numerator = computation_stage ? relaxationFactor*(b - dot_product) : 0;
DFEComplex[] R_conj = new DFEComplex[KaczmarzEngineCode.KaczmarzManager.array_size];
for(int j=0; j<kaczmarzEngineCode.KaczmarzManager.array_size; j++){
    R_conj[j] = kaczmarzEngineCode.KaczmarzWriteMemKernel.ComplexTruncatedFloatingPoint.newInstance(this);
    R_conj[j].setReal(R[j].getReal());
    R_conj[j].setImaginary(-R[j].getImaginary());
}
DFEComplex[] R_scaled = new DFEComplex[KaczmarzEngineCode.KaczmarzManager.array_size];
for(int j=0; j<kaczmarzEngineCode.KaczmarzManager.array_size; j++){
    R_scaled[j] = kaczmarzEngineCode.KaczmarzWriteMemKernel.ComplexTruncatedFloatingPoint.newInstance(this);
    R_scaled[j] = kaczmarz_numerator*R_conj[j];
}
//DFEComplex[] x_updated = new DFEComplex[KaczmarzEngineCode.KaczmarzManager.array_size];
for(int j=0; j<kaczmarzEngineCode.KaczmarzManager.array_size; j++){
    x_updated[j] <= x[j] + R_scaled[j];
}
```
Implementation Details
Layout of 3D Wavefields in 1D Memory

3D layout

Linear layout (for 5 x 5 x 5 system)

increasing memory addresses
Buffering: Overcoming Latency of Memory Access

Memory (24 GB)

On-chip Buffers (BRAM)
4.6 MB maximum
Pipelining: Overcoming Latency of Computation

slow | medium

fast

1
2
3
4
Pipelining: Overcoming Latency of Computation

- Fast
  - 5
  - 6
  - 7
  - 8

- Slow
  - medium
# Pipelining: Overcoming Latency of Computation

The diagram illustrates the concept of pipelining to overcome latency in computation. It shows three stages: fast, medium, and slow, with different levels of latency represented by numbers 9, 10, 11, and 12. The stages are connected by arrows indicating the flow of computation, with the fast stage leading to the medium stage, and then to the slow stage.
Memory Access: 384 bytes / burst

<table>
<thead>
<tr>
<th>Number of bits in a real number</th>
<th>Number of bits in a complex number</th>
<th>Complex numbers per burst</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>48</td>
<td>64</td>
</tr>
<tr>
<td>32 (single precision)</td>
<td>64</td>
<td>48</td>
</tr>
<tr>
<td>48</td>
<td>96</td>
<td>32</td>
</tr>
<tr>
<td>64 (double precision)</td>
<td>128</td>
<td>24</td>
</tr>
</tbody>
</table>

The Kaczmarz algorithm also requires one complex element from the iterate and right-hand side vectors at every clock tick, which means that the vector elements must be an integer fraction of the burst size. Possible options for the bitwidth of the vector element datatype are shown in Table 3.1. As a compromise between the need for an adequately precise number representation and a design that fits onto the FPGA, I settle on 32 bits for a real number. Although it is not necessary to store the individual non-zero elements of a row at the same precision as the elements of the iterate and right-hand side vectors, this is being done in the current implementation for simplicity. I note that this is not optimal in terms of matrix storage efficiency.
Backward Sweep: Double Buffering

First 48 ticks

burst i-1  
WRITE →
burst i  
← READ

Second 48 ticks

burst i-1  
← READ
burst i-2  
WRITE →
Number Representation

Matrix row storage efficiency vs. Number of bits in a real number.

32 bits used in this work.
Results
End-to-end Execution Time

![Graph showing the comparison between one Intel Xeon E5-2670 core and an Intel core + FPGA-based accelerator.]

- One Intel Xeon E5-2670 core
- Intel core + FPGA-based accelerator
Kaczmarz Sweeps: No Longer the Bottleneck

other CGMN operations
(inner products, vector addition, etc.)

Reference Implementation

Accelerated Implementation

Kaczmarz sweeps
Effect of matrix row ordering on CGMN convergence

Sequential (1 to $N$)

“Accelerator ordering”
FPGA Resource Usage

inner product: \( \langle a, x \rangle \)

row projection component: \( \times a^* \)

updating the iterate: \( x_k + \ldots \)

kernel (other)

LUTs

FFs

BRAMs

DSPs

automatically generated low-level components (manager)

buffers

memory controller

Fraction of available resources
Recent Work: Multiple Kaczmarz Sweeps / CGMN Iteration
(432 x 240 x 25 system)
Avoiding Future Communication Bottlenecks

![Graph showing FPGA frequency vs. memory and PCIe bandwidth](image)

- **38.4 GB/s**: memory bandwidth limit
- **2 GB/s**: PCIe bandwidth limit
The Next Step

**Problem:** On-chip memory (4 MB) limits block size to $300 \times 300$ in the two faster dimensions.

**Solution:** Implement **domain decomposition** for larger systems.
Straight-forward Extension

Goal: Systematically use all 4 accelerators.
Solution: Solve several forward problems at once.
Future Work

**Problem:** Kaczmarz sweeps now account for only approximately 10% of CGMN time.

**Solution:** Port all of CGMN to the DFE.
Future Work

**Fact:** Reading $A$ from memory limits optimizations like increasing FPGA frequency.

**Result:** Read only earth model $m$ and generate $A$ on the DFE.
Future Work

**Problem:** Domain size limited by memory size: 24 GB.

**Solution:** **Parallelize** CGMN to CARP-CG [Gordon & Gordon, 2010].
Conclusion

Have implemented frequency-domain wave simulation using reconfigurable hardware. A speed-up of 2 x 1 Intel Xeon core results from a dataflow computing paradigm.
Acknowledgements

Thank you to:

- Felix Herrmann, Henryk Modzelewski, Diego Oriato, Simon Tilbury, Tristan van Leeuwen, Eddie Hung, Lina Miao, Rafael Lago, my Master’s committee members: Michael Friedlander, Christian Schoof, my external examiner: Steve Wilton, Maxeler Technologies, and everyone in the SLIM group!
References


F. Grüll, M. Kunz, M. Hausmann, and U. Kebschull. An implementation of 3D electron tomography on FPGAs. In Reconfigurable Computing and FPGAs (ReConFig), 2012 International Conference on, pages 1–5, 2012. doi: 10.1109/ReConFig.2012.6416732.


