Reconfigurable Computing - CORDIC algorithms

Oliver Sinnen
Electrical and Computer Engineering
The University of Auckland

Cray XD1 supercomputer with FPGA (Xilinx Virtex II Pro) acceleration
Trigonometric Functions

- Functions such as $\sin$, $\cos$, $\tan$, ... appear in many problems
  - Signal processors
    - Transforms, Filters, ...
    - for end applications such as Radar, ...
  - Robotics
    - Motion prediction, environment geometry calculation, ...
  - Linear systems
    - Control, ...

- Computation techniques
  - Taylor series
    - Requires floating point
    - Iterative
    - Slow
  - Lookup tables
    - Fast
    - Require memory or
    - Limited precision
CORDIC algorithms – idea

- CORDIC arithmetic
  - Coordinate Rotation by Digital Computer
  - Technique proposed by Volder, 1956

Main idea:
- Use rotation transform:
- Rotate a vector \((x,y)^T\) by angle \(\Phi\)
CORDIC algorithms – idea

- **CORDIC arithmetic**
  - Coordinate Rotation by Digital Computer
  - Technique proposed by Volder, 1956

**Main idea:**
- **Use rotation transform:**
- **Rotate a vector** \((x,y)^T\) **by angle** \(\Phi\)
CORDIC algorithms – idea

- CORDIC arithmetic
  - Coordinate Rotation by Digital Computer
  - Technique proposed by Volder, 1956

Main idea:
- Use rotation transform:
- Rotate a vector \((x, y)^T\) by angle \(\Phi\)

\[
\begin{align*}
    x_1 &= x \cos(\Phi) - y \sin(\Phi) \\
    y_1 &= x \sin(\Phi) + y \cos(\Phi)
\end{align*}
\]
CORDIC algorithms – idea

- How does a vector rotation help us?
  \[ x_1 = x \cos(\phi) - y \sin(\phi) \]
  \[ y_1 = x \sin(\phi) + y \cos(\phi) \]

- Choose unit vector \((1,0)^T\) as starting vector

=>

\[ x_1 = \cos(\phi) \]
\[ y_1 = \sin(\phi) \]

=> Technique: Calculate the trigonometric functions \(\sin, \cos\) by rotating unit vector \((1,0)^T\)!
CORDIC procedure

How do we rotate a unit vector using computer arithmetic?

=> Using series of rotation with *tabled* values

Procedure:

- Iteratively rotate a unit vector until angle is $\phi$

\[
\begin{align*}
    x^{(i+1)} &= x^{(i)} \cos(\phi^{(i)}) - y^{(i)} \sin(\phi^{(i)}) \\
    y^{(i+1)} &= x^{(i)} \sin(\phi^{(i)}) + y^{(i)} \cos(\phi^{(i)}) \\
    z^{(i+1)} &= z^{(i)} - \phi^{(i)}
\end{align*}
\]

- $z$ is variable to keep track of the total rotation

Example, to rotate by $30^\circ$:

\[
45.0 - 26.6 + 14.0 - 7.1 + 3.6 + 1.8 - 0.9 + 0.4 - 0.2 + 0.1 = 30.1 \approx 30
\]
**CORDIC procedure**

**Procedure:**

- **Iteratively rotate a unit vector until angle is $\phi$**

  \[
  \begin{align*}
  x^{(i+1)} &= x^{(i)} \cos(\phi^{(i)}) - y^{(i)} \sin(\phi^{(i)}) \\
  y^{(i+1)} &= x^{(i)} \sin(\phi^{(i)}) + y^{(i)} \cos(\phi^{(i)}) \\
  z^{(i+1)} &= z^{(i)} - \phi^{(i)}
  \end{align*}
  \]

- **After $m$ rotations, with $x^{(0)}=x$, $y^{(0)}=y$, $z^{(0)}=z$**:

  \[
  \begin{align*}
  x^{(m)} &= x \cos(\sum \phi^{(i)}) - y \sin(\sum \phi^{(i)}) \\
  y^{(m)} &= x \sin(\sum \phi^{(i)}) + y \cos(\sum \phi^{(i)}) \\
  z^{(m)} &= z - (\sum \phi^{(i)})
  \end{align*}
  \]
Pseudorotations

Let us make it simpler for hardware implementation

\[
\begin{align*}
  x^{(i+1)} &= x^{(i)} \cos(\phi^{(i)}) - y^{(i)} \sin(\phi^{(i)}) \\
  y^{(i+1)} &= x^{(i)} \sin(\phi^{(i)}) + y^{(i)} \cos(\phi^{(i)})
\end{align*}
\]

\[
\begin{align*}
  x^{(i+1)} &= \cos(\phi^{(i)}) (x^{(i)} - y^{(i)} \tan(\phi^{(i)})) \\
  y^{(i+1)} &= \cos(\phi^{(i)}) (y^{(i)} + x^{(i)} \tan(\phi^{(i)}))
\end{align*}
\]

Pseudorotation:

\[
\begin{align*}
  x^{(i+1)} &= x^{(i)} - y^{(i)} \tan(\phi^{(i)}) \\
  y^{(i+1)} &= y^{(i)} + x^{(i)} \tan(\phi^{(i)}) \\
  z^{(i+1)} &= z^{(i)} - \phi^{(i)}
\end{align*}
\]

without \textit{cos} term

- Pseudorotations are simpler
- \textit{cos} term is constant for fixed angles
- Can be easily compensated later
CORDIC iteration

Choosing $\Phi^{(i)}$

- To simplify, pick $\Phi^{(i)}$ such that
  $$\tan(\Phi^{(i)}) = d_i 2^{-i}$$
  with $d_i \in \{-1, 1\}$

- Then
  $$x^{(i+1)} = x^{(i)} - d_i y^{(i)} 2^{-i}$$
  $$y^{(i+1)} = y^{(i)} + d_i x^{(i)} 2^{-i}$$
  $$z^{(i+1)} = z^{(i)} - d_i \tan^{-1}(2^{-i})$$

Advantage:

- Computation of $x^{(i+1)}$ and $y^{(i+1)}$ requires only $i$-bit right shift and add/subtract
- $\tan^{-1}(2^{-i})$ precomputed and stored in table

=> one CORDIC iteration involves 2 shifts, 1 table lookup, 3 additions!
CORDIC algorithms – rotation mode

- **CORDIC iterations**

  \[
  x^{(i+1)} = x^{(i)} - d_i y^{(i)} 2^{-i} \\
  y^{(i+1)} = y^{(i)} + d_i x^{(i)} 2^{-i} \\
  z^{(i+1)} = z^{(i)} - d_i \tan^{-1}(2^{-i})
  \]

  Rule: Choose \( d_i \in \{-1, 1\} \) such that \( z \to 0 \)

- **CORDIC equations**
  - with \( z = \sum \phi^{(i)} \)

  \[
  x^{(m)} = K(x \cos(z) - y \sin(z)) \\
  y^{(m)} = K(x \sin(z) + y \cos(z)) \\
  z^{(m)} = 0
  \]

  \[K = \prod \frac{1}{\cos(\phi^{(i)})} = \prod \sqrt{1 + \tan^2(\phi^{(i)})}\]

  \( K \) is a constant and can be precomputed (if always the same rotation angles are used)
Rotation mode – example

- **CORDIC iterations**

\[
\begin{align*}
x^{(i+1)} &= x^{(i)} - d_i y^{(i)} 2^{-i} \\
y^{(i+1)} &= y^{(i)} + d_i x^{(i)} 2^{-i} \\
z^{(i+1)} &= z^{(i)} - d_i \tan^{-1}(2^{-i})
\end{align*}
\]

Rule: Choose \( d_i \in \{-1, 1\} \) such that \( z \to 0 \)

- **From the rule it follows:** \( d_i = \text{sign}(z^{(i)}) \)

**Example:**

Rotate by 30°, i.e. \( z = 30° \):

\[-45.0 + 26.6 - 14.0 + 7.1 - 3.6 - 1.8 + 0.9 - 0.4 + 0.2 - 0.1\]

\[= -0.1 \approx 0\]

- **For \( n \)-bit precision of result, \( n \) CORDIC iterations!**

\[
\begin{array}{c|c|c}
 i & \tan^{-1} 2^{-i} \text{ (in degree)} \\
0 & 45.0 \\
1 & 26.6 \\
2 & 14.0 \\
3 & 7.1 \\
4 & 3.6 \\
5 & 1.8 \\
6 & 0.9 \\
7 & 0.4 \\
8 & 0.2 \\
9 & 0.1 \\
\end{array}
\]

\( K = 1.646760258121... \)
Rotation mode – calculating of $\cos \Phi, \sin \Phi$

- **CORDIC equations**

  \[
  x^{(m)} = K (x \cos(z) - y \sin(z)) \\
  y^{(m)} = K (x \sin(z) + y \cos(z)) \\
  z^{(m)} = 0
  \]

**$\cos \Phi, \sin \Phi$**

- $z = \Phi, x = 1/K, y = 0$ \hspace{1cm} $1/K$ can be precomputed

  => $x^{(m)} = \cos \phi$ \\
  $y^{(m)} = \sin \phi$

Domain of convergence:

- **Usually** $-A < z < A$, with $A > \pi/2$ (90°), where $A$ is the sum of all predefined angles $A = \sum \tan^{-1}(2^{-i})$

- **Outside this range:**
  - use trigonometric identities, e.g. $\cos(\phi-\pi) = -\cos \phi$
CORDIC algorithms – vector mode

- **In rotation mode** \( z \rightarrow 0 \)

Now
- **In vector mode** \( y \rightarrow 0 \)
- **After** \( m \) rotations, with \( x^{(0)}=x \), \( y^{(0)}=y \), \( z^{(0)}=z \) :

\[
\begin{align*}
x^{(m)} &= K \left( x \cos \left( \sum \phi^{(i)} \right) - y \sin \left( \sum \phi^{(i)} \right) \right) \\
y^{(m)} &= K \left( x \sin \left( \sum \phi^{(i)} \right) + y \cos \left( \sum \phi^{(i)} \right) \right) \\
z^{(m)} &= z - \left( \sum \phi^{(i)} \right)
\end{align*}
\]

\[
y^{(m)} = 0 = K \left( x \sin \left( \sum \phi^{(i)} \right) + y \cos \left( \sum \phi^{(i)} \right) \right) = K \sin \left( \sum \phi^{(i)} \right) \left( x + \frac{y}{\tan \left( \sum \phi^{(i)} \right)} \right)
\]

\[
\Rightarrow \quad \frac{-y}{x} = \tan \left( \sum \phi^{(i)} \right)
\]
Vector mode

- **with** \( \frac{-y}{x} = \tan(\sum \phi^{(i)}) \)

\[
\Rightarrow \quad x^{(m)} = K \left( x \cos(\sum \phi^{(i)}) - y \sin(\sum \phi^{(i)}) \right)
= K \cos(\sum \phi^{(i)})(x - y \tan(\sum \phi^{(i)}))
= K \cos(\sum \phi^{(i)})(x + y^2/x)
= K \frac{x + y^2/x}{\sqrt{1 + \tan^2(\phi^{(i)})}} = K \frac{x + y^2/x}{\sqrt{1 + y^2/x^2}}
= K \sqrt{x^2 + y^2}
\]

\[
z^{(m)} = z - (\sum \phi^{(i)})
= z \pm \tan^{-1}(y/x)
\]

\[
\cos(\phi^{(i)}) = \frac{1}{\sqrt{1 + \tan^2(\phi^{(i)})}}
\]
Vector mode

- **CORDIC iterations**

\[
x^{(i+1)} = x^{(i)} - d_i \cdot y^{(i)} \cdot 2^{-i} \\
y^{(i+1)} = y^{(i)} + d_i \cdot x^{(i)} \cdot 2^{-i} \\
z^{(i+1)} = z^{(i)} - d_i \cdot \tan^{-1}(2^{-i})
\]

**Rule:** Choose \(d_i \in \{-1, 1\}\) such that \(y \to 0\)

- **CORDIC equations**

\[
x^{(m)} = K \sqrt{x^2 + y^2} \\
y^{(m)} = 0 \\
z^{(m)} = z + \tan^{-1}(y/x)
\]

- **From the rule it follows:** \(d_i = -\text{sign}(x^{(i)} y^{(i)})\)
  - Within the usual domain of convergence, \(x^{(i)}\) is always positive
  \[\Rightarrow d_i = -\text{sign}(y^{(i)})\]
Vector mode – calculation of $\tan^{-1} \Phi$

- **CORDIC equations**

  \[
  x^{(m)} = K \sqrt{x^2 + y^2} \\
  y^{(m)} = 0 \\
  z^{(m)} = z + \tan^{-1}(y/x)
  \]

\[\tan^{-1} \Phi\]

- $z = 0$, $x = 1$, $y = \Phi$

$\Rightarrow z^{(m)} = \tan^{-1} \phi$
CORDIC algorithms – summary

Summary:

- By appropriately choosing the start values for $x, y, z$ and the rule for $d_i$ calculation, many functions can be computed
  - $\sin, \cos, \tan^{-1}$, vector rotation, vector magnitude
  - even more complex functions, e.g. $\tan^{-1}(y/x)$
- For $n$-bit precision of result, $n$ CORDIC iterations
- One CORDIC iteration involves 2 shifts, 1 table lookup, 3 additions

=> complexity similar to sequential multiplication!

Virtually all functions of common interest can be computed with the (generalised) CORDIC method

- $\cos^{-1}, \sin^{-1}$, hyperbolic functions (e.g. $\sinh$) ...
- even division and multiplication!
- $\ln w, e^z$ ...
CORDIC implementations

Implementation

- Angles in radians, range \([-\pi/2, \pi/2]\) (i.e. \([-90^\circ, 90^\circ]\))
  - For other angles: use trig. identities and convert to angle in \([-\pi/2, \pi/2]\)
  - Angles are multiples of \(\pi\) => possible to work in \([-\frac{1}{2}, \frac{1}{2}]\), with \(\pi\) implicit
- Values are real numbers
  - Range is limited
    
    \(eg\) all numbers are \(0 \leq x < 1\)

=> well suited to fixed-point arithmetic

- Choose an appropriate representation
  - Use 2’s complement representation
  - Place the virtual point after the first binary bit
- Fixed point representations work well with data from sensors
  - It comes from fixed-point A to D converters!

- Precision
  - Determine the necessary precision, i.e. the number of bits

=> that determines the necessary number of CORDIC iterations
CORDIC implementations

- Assume
  - Application demands \( n \)-bits of accuracy on \( m \)-bit inputs

- One CORDIC iteration involves 2 shifts, 1 table lookup, 3 additions

- Design a basic block with these operations
CORDIC basic block – rotation mode

- CORDIC iteration

\[
x^{(i+1)} = x^{(i)} - d_i y^{(i)} 2^{-i} \\
y^{(i+1)} = y^{(i)} + d_i x^{(i)} 2^{-i} \\
z^{(i+1)} = z^{(i)} - d_i \tan^{-1}(2^{-i})
\]

\[d_i = \text{sign}(z^{(i)})\]
CORDIC implementation

- Combine \( n \) basic blocks
  - \( n \) blocks, each with \( m \)-bit add/subtract/shift units
  - \( O(m \times n) \) space complexity
  - \( O(m \times n) \) time complexity
    - \( O(n \log(m)) \) with efficient adders
  - Latency – long – \( n \cdot t_{block} \)
  - Throughput – \( f = 1/(n \cdot t_{block}) \)

Note
- LUT in each block is only one value
- Propagation of \( i \) not necessary
- Shift can be “wired”
CORDIC implementation – pipeline

- **Pipeline $n$ basic blocks**
  - Add a register to each block
  - Complexity remains the same
  - Latency – increased by register overhead
  - Throughput – increases by $p' < n$
Pipeline implementation – $k$ iterations/stage

- **Wrap** $k$ instances of the basic model in a stage model
  - pipeline register after each stage
  - Complexity remains the same
  - Latency – less register overhead
  - **Stage delay can be adjusted to match that for slowest stage in the remainder of the system**
  - Throughput – increases by $p' < \frac{n}{k}$
CORDIC implementation – sequential

- Wrap the basic model in a recycling block
  - $O(m)$ space complexity
  - $O(m \cdot n)$ time complexity
    - $O(n \log(m))$ with efficient adders
  - Latency – $n \cdot (t_{\text{block}} + t_{\text{reg}})$
  - Good compromise between space and speed
    - Can use fast carry chain adders!
Sequential implementation – $k$ iterations/stage

- Wrap $k$ instances of the basic model in a recycling block
  - $O(m \frac{n}{k})$ space complexity
  - $O(m \cdot n)$ time complexity
    - $O(n \log(m))$ with efficient adders
  - Latency – $(n/k) \times (k \cdot t_{\text{block}} + t_{\text{reg}})$
  - Adjust $k$ to use all available space to gain speed