Global J. of Engng. Educ., Vol.8, No.3
Published in Australia

CORDIC-Algorithmen Verbinden Mathematik, Computer-Architektur und Anwendungen

Thomas Risse
Hochschule Bremen, Institut für Informatik & Automation
Neustadtswall 30, D-28199 Bremen, Deutschland

Lehren und Lernen soll aktuell, Anwendungsbezogen, explorativ, Praxisnah und Problemorientiert sein, – auch und besonders in Mathematik. CORDIC-Algorithmen werten effizient die elementaren trigonometrischen und hyperbolischen Funktionen aus. Sie stellen damit einen, im Übrigen vernachlässigten Themen-Bereich dar, der die obigen wünschenswerten Eigenschaften in hohem Maß aufweist. Denn CORDIC-Algorithmen erlauben, eine Vielzahl von Problemen etwa der Signalverarbeitung durch exemplarische Implementierungen zu lösen. CORDIC-Algorithmen sind somit in beispielhafter Weise geeignet, Mathematik, Computer-Architektur mit im weitesten Sinn Signalverarbeitung und damit eben allgemein Grundlagen, Implementierung und Anwendung zu verbinden.

EINFÜHRUNG

Taylor-Polynome werden traditionell als das erste und häufig alleinige Mittel der Wahl dargestellt, elementare Funktionen auszuwerten. Dabei zeigen schon einfachste, ältere Taschenrechner, dass Potenzreihen-Entwicklung eher nicht Methode dieser Berechnungen sein kann. Eigenschaften von Intel x86 (Co-) Prozessoren deuten in dieselbe Richtung.

Grund genug, sich mit CORDIC-Algorithmen zu befassen. Diese erlauben, die elementaren trigonometrischen und hyperbolischen Funktionen und deren Inverse effizient auszuwerten. Dabei werden Überlegung zu Konvergenz, Effizienz, Zulässigkeit, Genauigkeit usw. notwendig.

Sollen CORDIC-Algorithmen implementiert werden, sind eine Reihe von Entwurfsentscheidungen zu treffen, die diverse Kriterien wie Geschwindigkeit, Genauigkeit, Chip-Fläche, Stromverbrauch oder auch Fehler-Toleranz mehr oder weniger gut bedienen.

CORDIC-Algorithmen spielen in vielen Anwendungen eine entscheidende Rolle, wie Beispiele aus etwa Signalverarbeitung, generativer Computer-Graphik oder Bildverarbeitung belegen.

Eine derartige ganzheitliche Betrachtung von CORDIC-Algorithmen bietet die Chance, Mathematik – und damit Theorie –, Implementierung – und damit Computer-Architektur – und etwa Signalverarbeitung – und damit Anwendungen – zu verbinden.

In der Mathematik-Ausbildung liefern CORDIC-Algorithmen vielfältige Beispiele etwa für elementare Funktionen, Approximation, Konvergenz, Numerik oder Algorithmik und Komplexität. Diese Bezüge zur Mathematik-Ausbildung werden im Folgenden jeweils durch >>>*Stichwort* aufgezeigt.

TRADITION

Ausgangspunkt ist die Frage, wie man effizient beispielsweise, exp(π), sin(1/3), arctan(1/7), cosh(0.7), sqrt(e) usw. usf. berechnet. >>> elementare Funktionen.

Die vermutlich typische, übliche wie gängige Antwort lautet:

- Per Polynom-Approximation (Potenz-, Taylor-Reihe) >>> polynomiale Approximation;
- Per look up table mit etwa linearer Interpolation, was immerhin das Prinzip trading space for time klassisch illustriert. >>> look up table >>> lineare Interpolation.

Selbstverständlich gibt es weitere Verfahren wie etwa:

- Das Newton (-Raphson) – natürlich nur für Funktionen ähnlich dem Beispiel sqrt(e) >>> Newton-Raphson Verfahren;

- Die Fourier-Approximation (Fourier-Reihe) – natürlich nur für periodische Funktionen >>> Fourier-Approximation.

BEISPIELE FÜR DIESE TRADITION IN DER LITERATUR

Einige beispielhafte Zitate aus gängigen, einschlägigen Lehrbüchern untermauern diese traditionelle Praxis:

- Hartmann: *Der Taschenrechner macht nichts anderes, als die ersten Glieder von* $1+x+x^2/2!+\ldots+x^n/n!+\ldots$ *auszuwerten* [1];
- Papula: *Näherungsweise Berechnung von Funktionswerten …* [2];
- Stingl: *Per Taylor-Polynom kann man den Wert beliebiger Funktionen näherungsweise, nur mit Hilfe der Grundrechnungsarten berechnen* [3];
- Stöcker: *Unter der Verwendung von Potenzreihen kann die Berechnung von Funktionswerten auf einfache Weise erfolgen …* [4];
- Brauch, Dreyer und Haacke, *Reihen von Zahlen werden i.A. dazu benutzt, Funktionswerte von transzendenten Funktionen zu berechnen. Zur Berechnung des Sinus eines Winkels mit dem Taschenrechner …* [5];
- Burg, Haf und Wille: *Durch Partialsummen der Potenzreihen sind Polynome gegeben, die die Funktionen mehr oder weniger gut approximieren und daher zur numerischen Berechnung herangezogen werden können*. Es folgt immerhin der Verweis auf Bernstein-Polynome, Tschebyscheff-Polynome und B-Spline-Approximation [6].

Alle Autoren behandeln selbstverständlich Potenz- und Fourier-Reihen, jedoch behandelt keiner der genannten Autoren CORDIC-Algorithmen. Dabei sind diese alles andere als unwichtig.

BEDEUTUNG DER FRAGESTELLUNG

Sicherlich fallen einem jeden mannigfache Situationen ein, in denen die Auswertung elementarer Funktionen unverzichtbar ist.

- sin(x) und cos(x) beispielsweise für Rotationen der generativen Computer Graphik, Roboter-Steuerung, Sinus-Generatoren, Fourier-Transformation und ihre mannigfachen Abkömmlinge usw;
- arctan(x) beispielsweise für Überführung Cartesischer Koordinaten in Polar-Koordinaten usw;
- sqrt(x) beispielsweise für Abstände von Punkten, für das Normieren von Vektoren usw;
- exp(x) beispielsweise für Wachstums- oder Abkling-Vorgänge, für Normalverteilung, Ausfallwahrscheinlichkeiten nicht alternder Bauteile, usw.

TAYLOR-APPROXIMATION KANN'S NICHT SEIN!

Einige Indizien, besonders etwa bei Verwendung leistungsschwacher Taschenrechner, lassen vermuten, dass die Taylor-Approximation vielen Anwendungen gerade nicht zugrunde liegen kann.

- Vermittels Taylor-Approximation dauert die Auswertung beispielsweise des Logarithmus wesentlich länger als die Auswertung der Exponential-Funktion. >>> Konvergenz-Geschwindigkeit;
- Die Konvergenz-Geschwindigkeit der Taylor-Approximation hängt stark vom Argument ab! >>> Konvergenz-Geschwindigkeit.

Weitere Indizien liefern die Prozessoren der Intel x86-Familie.

Seit 1980 implementiert der 8087, der mathematische Coprozessor des 8086, in Hardware eine Reihe von Konstanten, trigonometrischen und logarithmischen Funktionen, die zusammen mit ihrer Ausführungszeit (cycles) [7-9] für alle Mitglieder der x86-Prozessor-Familie aufgelistet seien (Tabelle 1).

FPATAN und FPTAN sind im Intel-Jargon sogenannte *partielle* Funktionen: FPATAN berechnet arctan für den Quotienten zweier Argumente und FPTAN berechnet zugleich sin und cos. Das Ergebnis tan liefert erst eine zusätzliche Division.

Seit 1985 sind im 80386 und seinen Nachfolgern als zusätzliche Funktionen sin und cos explizit implementiert (Tabelle 2) [7-9].

Dies bestätigt die Intel-Aussage zu FSINCOS: *This instruction is faster than executing the FSIN and FCOS instructions in succession* [10].

Die Indizien sind erdrückend: simple Taylor-Approximation allein kann diesen Implementierungen nicht zugrunde liegen! (Story et al zeigt, wie die Entwicklung extrem leistungsfähiger Gleitkomma-Einheiten die Implementierungen der elementaren Funktionen in aktuellen Intel-Prozessoren der x86-Familie beeinflusst hat und wie kompliziert die Berechnung tatsächlich ist! [11]).

Tabelle 1: Funktion und Instruktion.

Funktion	Instruktion	8087	80287	80387	80486	80586
Arctan	FPATAN	250-800	250-800	314-487	218-303	17-173
Tan	FPTAN	30-540	30-540	191-497	200-273	17-173
sqrt(x)	FSQRT	180-186	180-186	122-129	83-87	70
y*log_2(x)	FYL2X	900-1100	900-1100	120-538	196-329	22-111
y*log_2(x+1)	FYL2XP1	700-1100	700-1100	257-547	171-326	22-103

Note: FPATAN: arg=Y/X mit 8087/80287: $0 \leq Y < X < \infty$.
FPTAN: $0 \leq$ arg $\leq \pi/4$ auf 8087/287, berechnet Y=sin(arg) und X=cos(arg). Zusätzliche Division liefert das Ergebnis Y/X. 80387: falls $|arg|>\pi/4$ bis zu 76 zusätzliche cycles, 80487: falls $|arg|>\pi/4$ zusätzlich $arg/(\pi/4)$ cycles.

Tabelle 2: Funktionen sin und cos explizit implementiert.

Funktion	Instruktion	8087	80287	80387	80486	80586
cos	FCOS			123-772	257-354	18-124
Sin	FSIN			122-771	257-354	16-126
sin und cos	FSINCOS			194-809	292-365	17-137

Note: FCOS, FSIN, FSINCOS: 80387: falls $|arg|>\pi/4$ bis zu 76 zusätzliche cycles, 80487: falls $|arg|>\pi/4$ zusätzlich $arg/(\pi/4)$ cycles.

TAYLOR-APPROXIMATION VERSUS CORDIC

Die Unterschiede von Taylor-Approximation und CORDIC-Algorithmen seien am Beispiel der Berechnung von cos(1.45) illustriert (*Fehler aufgrund Rundung des Argumentes werden hier nicht betrachtet*):

- Taylor: Approximiere cos(x) durch $1-x^2/2!+x^4/4!-+\ldots+ (-1)^n\ x^{2n}/((2n)!)$. Die Reihe wird eben durch das Taylor-Polynom für festes Argument approximiert. Fehler entstehen aufgrund des Abbrechens der Taylor-Reihe, und der Arithmetik bei der Auswertung des Taylor-Polynoms.
- CORDIC Volder entdeckte 1959 (!) den COordinate Rotation DIgital Computer, CORDIC, der trigonometrische Funktionen auswertet und den Walther 1971 zur Auswertung beliebiger elementarer Funktionen verallgemeinerte [12][13]. Approximation des Funktionswertes durch Approximation des Argumentes. Unterstellt, dass die transzendente Funktion cos(x) für bestimmte Argumente korrekt berechnet werden kann, entstehen so Fehler aufgrund dieser Approximation des Argumentes und der Arithmetik bei der Auswertung von cos(x).

Abbildung 1 zeigt den Unterschied der Berechnung von cos(1.45), auf der einen Seite per Taylor-Approximation und auf der anderen Seite per CORDIC-Algorithmus.

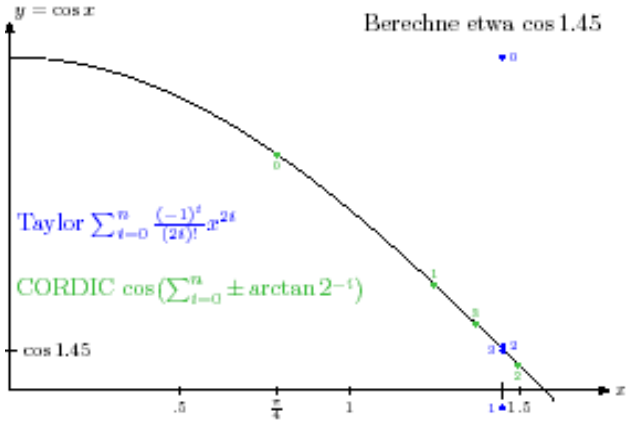

Abbildung 1: Unterschied der Berechnung von cos(1.45): Taylor-Approximation und CORDIC-Algorithmus.

CORDIC

CORDIC-Algorithmen berechnen etwa die trigonometrischen Funktionen sin(x) oder cos(x) gemeinsam (*Wie fangen Mathematiker einen Löwen? Sie fangen zwei und lassen einen wieder laufen!*).

In komplexer Schreibweise gilt: $\cos\varphi+j\sin\varphi = \exp(j\varphi) = \exp(j\varphi_{n-1})\exp(j\varphi_{n-2})\ldots\exp(j\varphi_1)\exp(j\varphi_o)$ falls $\varphi = \varphi_{n-1}+\varphi_{n-2}+\ldots+\varphi_1+\varphi_o$.
oder eben vektoriell geschrieben – wobei hier $\varphi_o=0$ gesetzt sei

$$\begin{pmatrix}\cos\varphi\\ \sin\varphi\end{pmatrix} = \begin{pmatrix}\cos\varphi & -\sin\varphi\\ \sin\varphi & \cos\varphi\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix}$$
$$= \begin{pmatrix}\cos\varphi_{n-1} & -\sin\varphi_{n-1}\\ \sin\varphi_{n-1} & \cos\varphi_{n-1}\end{pmatrix}\cdots\begin{pmatrix}\cos\varphi_1 & -\sin\varphi_1\\ \sin\varphi_1 & \cos\varphi_1\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix}$$

Noch ist nichts gewonnen, außer dass die Auswertung von cos(φ) und sin(φ) auf eine Rotation oder mehrere Rotationen zurückgeführt wurde.

CORDIC-ROTATIONEN

$$\begin{aligned}\begin{pmatrix}\cos\varphi\\ \sin\varphi\end{pmatrix} &= \prod_{i=0}^{n-1}\begin{pmatrix}\cos\varphi_i & -\sin\varphi_i\\ \sin\varphi_i & \cos\varphi_i\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix}\\ &= \prod_{i=0}^{n-1}\cos\varphi_i\begin{pmatrix}1 & -\tan\varphi_i\\ \tan\varphi_i & 1\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix}\\ &= \left(\prod_{i=0}^{n-1}\cos\varphi_i\right)\prod_{i=0}^{n-1}\begin{pmatrix}1 & -\tan\varphi_i\\ \tan\varphi_i & 1\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix}\end{aligned}$$

Wählt man nun $|\varphi_i|=\arctan(2^{-i})$, dann besteht eine CORDIC-Rotation

$$\begin{pmatrix}1 & \mp\tan\varphi_i\\ \pm\tan\varphi_i & 1\end{pmatrix}\begin{pmatrix}x\\y\end{pmatrix}=\begin{pmatrix}1 & \mp 2^{-i}\\ \pm 2^{-i} & 1\end{pmatrix}\begin{pmatrix}x\\y\end{pmatrix}=\begin{pmatrix}x\mp 2^{-i}y\\ 2^{-i}x\pm y\end{pmatrix}$$

aus zwei SHIFTs und zwei ADDs. CORDIC-Rotationen sind also mit minimalem (hardware-) Aufwand durchzuführen.

CORDIC-Rotationen beinhalten allerdings eine implizite Skalierung, die durch Multiplikation mit geeigneten Faktoren ausgeglichen werden muss, nämlich bei Prä- oder Post-Skalierung per Multiplikation mit

$$K=\lim_{n\to\infty}\prod_{i=0}^{n-1}\cos\arctan 2^{-i}=\lim_{n\to\infty}\prod_{i=0}^{n-1}\frac{1}{\sqrt{1+2^{-2i}}}\approx\frac{1}{1.64676}\approx 0.607253$$

>>>Norm-erhaltende Transformation

CORDIC-ALGORITHMUS (ROTATION FORWARD)

CORDIC-Algorithmen operieren auf Tripeln (x_i,y_i,z_i)

$$\begin{pmatrix}x_{i+1}\\ y_{i+1}\end{pmatrix}=\begin{pmatrix}1 & \mp 2^{-i}\\ \pm 2^{-i} & 1\end{pmatrix}\begin{pmatrix}x_i\\ y_i\end{pmatrix}=\begin{pmatrix}x_i\mp 2^{-i}y_i\\ 2^{-i}x_i\pm y_i\end{pmatrix}\quad\text{mit}\quad\begin{pmatrix}x_o\\ y_o\end{pmatrix}=\begin{pmatrix}1\\0\end{pmatrix}$$
$$0\leftarrow z_{i+1}=z_i-\alpha_i\,\mathrm{sgn}(z_i)\quad\text{mit }\alpha_i=\arctan 2^{-i}\qquad z_o=\varphi$$

Wie beispielsweise sin 30° bzw. cos 30° approximiert wird, zeigt in Abbildung 2.

Numerische Experimente mit CORDIC-Algorithmen erlaubt im Übrigen auch das interaktive Dokument [14]. Hier operiert der CORDIC im so genannten *circulären* Modus und zwar in der Richtung rotation oder forward, vgl. [15].

CORDIC-KONVERGENZ UND -KONVERGENZ-BEREICH

Konvergenz der CORDIC-Algorithmen garantiert der folgende Satz:
Gegeben $\alpha_o \geq \alpha_1 \geq \ldots \geq \alpha_n > 0$ mit $\alpha_k \geq \alpha_n+\alpha_{k+1}+\alpha_{k+2}+\ldots+\alpha_n$ für alle $0 \leq k \leq n$. Gegeben ein reelles r

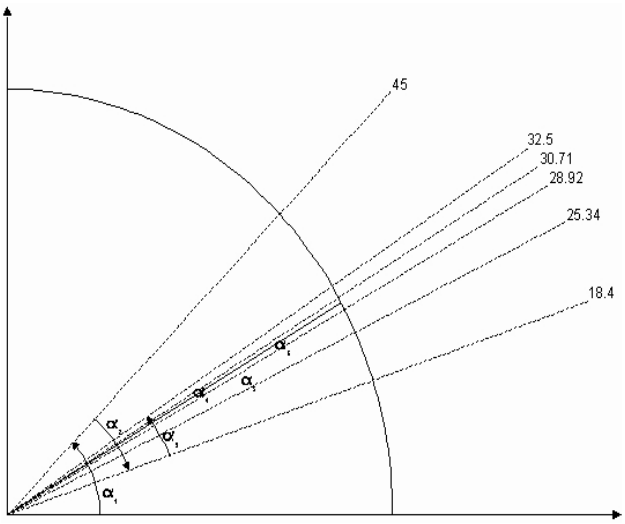

Abbildung 2: Wie sin 30° bzw. cos 30° approximiert wird.

mit $|r| \leq \alpha_1+\alpha_2+\ldots+\alpha_n$.

Definiere induktiv $z_{i+1} = z_i + \alpha_i\,\mathrm{sgn}(r-z_i)$ mit $z_o=0$. Dann gilt:

$$|r-z_k|\leq\alpha_n+\sum_{i=k}^{n}\alpha_i \text{ für alle } 0\leq k\leq n \text{ und speziell } |r-z_{n+1}|\leq\alpha_n$$
$$\text{Falls } \alpha_n\to 0 \text{ dann } z_n\to r$$

Beweis durch Fallunterscheidung s. z.B. [14].

Folgerung: Gesicherte Konvergenz für alle r mit

$$|r|\leq\sum_{i=0}^{n}\alpha_i\qquad\text{mit}\quad\alpha_i=\arctan 2^{-i}\quad\text{und}\quad\sum_{i=0}^{\infty}\alpha_i\approx 1.743$$

CORDIC-SCALING

Falls der Skalierungsfaktor K in sign-digit-representation $K=\kappa_0\,2^0+\kappa_1\,2^{-1}+\ldots+\kappa_n\,2^{-n}$ mit $\kappa_i=\pm 1$ gegeben ist, so lässt sich auch die Skalierung nur mit SHIFTs und ADDs durchführen. Darüber hinaus gibt es alternative Herangehensweisen, vgl. z.B. [16].

CORDIC-VECTORING (VECTORING, BACKWARD)

Der CORDIC-Algorithmus operiert im so genannten *circulären* Modus auch in der entgegen gesetzten Richtung *vectoring* oder *backward*, vgl. [15].

Um $\varphi = \arcsin(\text{arg})$ oder $\arccos(\text{arg}) = \pi/2-\arcsin(\text{arg})$ zu berechnen, wird der Einheitsvektor e_x rotiert, bis die y-Koordinate des gedrehten Vektors mit arg übereinstimmt: $y_i \longrightarrow \text{arg}$ (Abbildung 3).

$$\begin{pmatrix}x_{i+1}\\ y_{i+1}\end{pmatrix}=\begin{pmatrix}1 & \mp 2^{-i}\\ \pm 2^{-i} & 1\end{pmatrix}\begin{pmatrix}x_i\\ y_i\end{pmatrix}=\begin{pmatrix}x_i\mp 2^{-i}y_i\\ 2^{-i}x_i\pm y_i\end{pmatrix}\quad\text{mit}\quad\begin{pmatrix}x_o\\ y_o\end{pmatrix}=\begin{pmatrix}1\\0\end{pmatrix}$$
$$\varphi\leftarrow z_{i+1}=z_i+\alpha_i\,\mathrm{sgn}(arg-y_i)\text{ mit }\alpha_i=\arctan 2^{-i}\qquad z_o=0$$

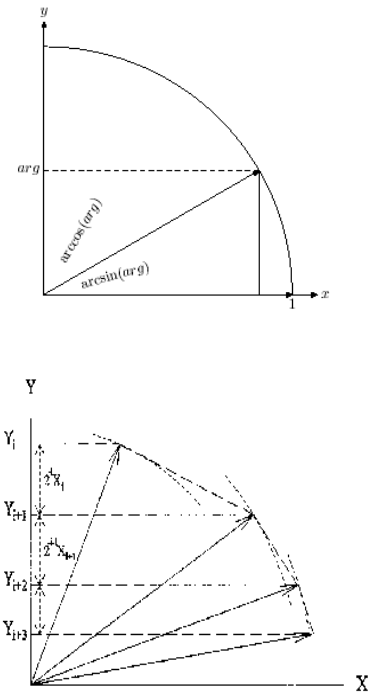

Abbildung 3: Einheitsvektor e_x rotiert.

KOORDINATION-TRANSFORMATIONEN

Offensichtlich lassen sich mit CORDIC-Algorithmen im circulären Modus Polar- in Cartesische Koordinaten und umgekehrt umrechnen.

rotating	$(r,\varphi) \to (x,y) = r(\cos\varphi, \sin\varphi)$
vectoring	berechne $\varphi = \arctan\frac{y}{x}$ zu $(x,y) \to (r,0) = \left(\sqrt{x^2+y^2}, 0\right)$

HYPERBOLIC CORDIC

Die enge Beziehung zwischen den trigonometrischen und den hyperbolischen Funktionen legt den hyperbolischen Modus des CORDIC nahe. >>>hyperbolische Additionstheoreme

$$\cos(x+y) = \cos x \cos y - \sin x \sin y \qquad \sin(x+y) = \sin x \cos y + \sin y \cos x$$
$$\updownarrow \qquad \cosh(j\,x) = \cos x \qquad \sinh(j\,x) = j\,\sin x \qquad \updownarrow$$
$$\cosh(x+y) = \cosh x \cosh y + \sinh x \sinh y \qquad \sinh(x+y) = \sinh x \cosh y + \sinh y \cosh x$$

Der *hyperbolic CORDIC* berechnet also

per rotating	sinh, cosh und so tanh, coth und exp = sinh + cosh
per vectoring	arsinh, arcosh, artanh und auch $\ln x = 2\,\text{artanh}\,\frac{x-1}{x+1}$

LINEAR CORDIC

Der CORDIC-Algorithmus im sogenannten *linearen* Modus multipliziert (*rotating*) oder dividiert (*vectoring*)! Wenn auch diese Implementierung den Standard-Implementierungen für Multiplizierer oder Dividierer nicht ebenbürtig ist, bietet sie doch den Vorteil, dass ein vorhandener CORDIC fast ohne zusätzlichen hardware-Aufwand eben zugleich auch Multiplikationen und Divisionen ausführen kann – eine attraktive Variante dann, wenn man durch Einsparen zusätzlicher funktionaler Einheiten etwa die Verlustleitung reduzieren kann [17].

CORDIC FUNKTIONALITÄT

Die Funktionalität von CORDIC-Implementierungen sei hier zusammengestellt:

mode	m	rotating
circular	1	$\sin, \cos \Rightarrow \tan, \cot, \begin{pmatrix} r \\ \varphi \end{pmatrix} \to r \begin{pmatrix} \cos\varphi \\ \sin\varphi \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix}$
linear	0	Multiplikation
hyperbolic	-1	$\sinh, \cosh \Rightarrow \tanh, \coth, \exp$

mode	m	vectoring
circular	1	$\arcsin, \arccos, \arctan, \text{arccot} \Rightarrow \begin{pmatrix} x \\ y \end{pmatrix} \to \begin{pmatrix} r \\ \varphi \end{pmatrix}$
linear	0	Division
hyperbolic	-1	$\text{artanh} \Rightarrow \text{arsinh}, \text{arcosh}, \ln, \text{sqrt}$

Die CORDIC-Algorithmen in den drei Modi stehen in Form von interaktiven Dokumenten [14] zum Ausprobieren zur Verfügung.

IMPLEMENTIERUNG

Eine Implementierung ist immer im Licht bestimmter Anwendungen und bestimmter Entwurfsziele wie

- Geschwindigkeit,
- effiziente Speicher-Nutzung,
- Genauigkeit [18],
- geringe Verlustleistung [19-22],
- Fehler-Toleranz [23],

zu bewerten. Beispiele für verschiedene CORDIC-Architekturen mögen dies illustrieren.

- Building Blocks:
  - [SHIFT] switches, barrel shifters;
  - [ADD/SUB] ripple-carry-, carry-save-, carry-look-ahead-, conditional sum-, limited-size carry-look-ahead-, … adders;
  - [scaling] pre-, post-scaling;
  - [Architecture] fix $\longrightarrow$ float, sign digit, pipeline, processor array, systolic array.
- Basic Circuit (Abbildung 4);
- Pipeline (Abbildung 5);
- Tree (DFT) (Abbildung 6).

Weiterentwicklungen weisen in verschiedene Richtungen:

- Verbessertes pipelining durch *schnelleres Vorzeichen* (sign-digit und adder);

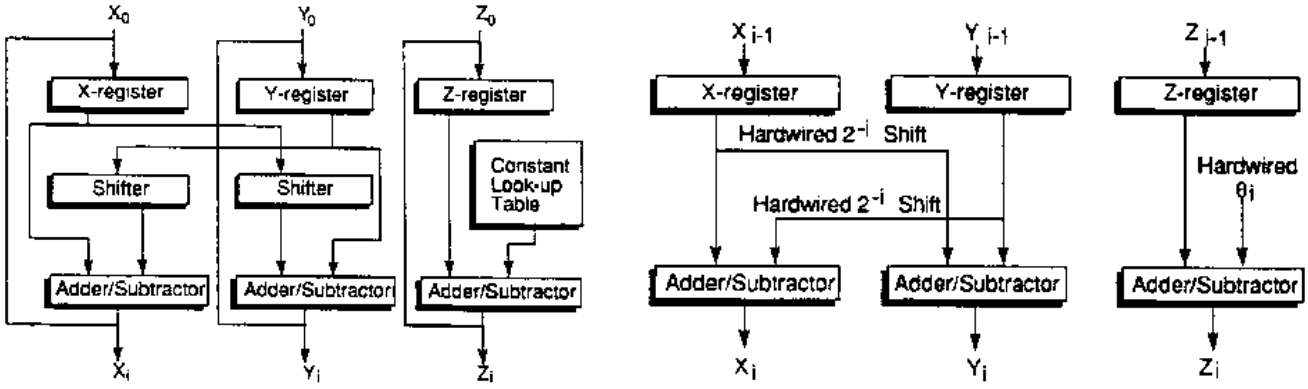

Abbildung 4: Basic Circuit.

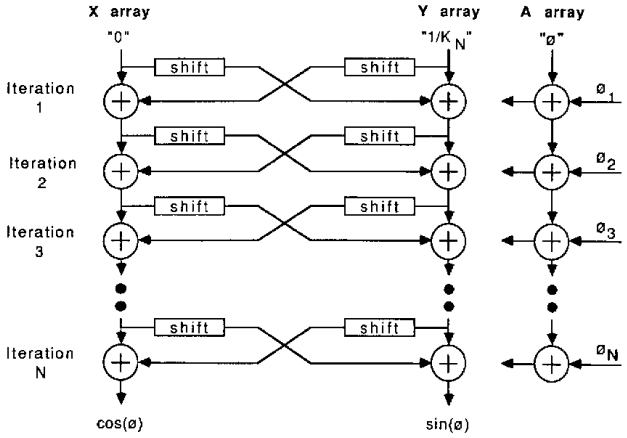

Abbildung 5: Pipeline.

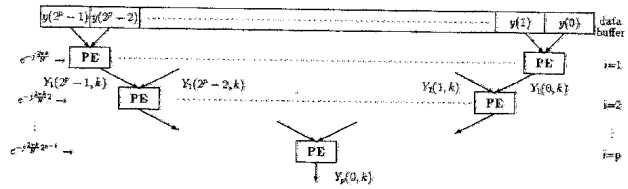

Abbildung 6: Tree (DFT).

- Weniger Iterationen, d.h. schnellere Konvergenz durch adaptives Inkrementieren;
- Verringerter Chip-Fläche, z.B. durch hierarchische barrel shifter;
- Verallgemeinerung durch Integration der drei Modi: circular, linear, hyperbolisch;
- Flexibilität durch Programmierbarkeit/ISA [24];
- Weitere Erhöhung der Geschwindigkeit durch Parallelisierung, z.B. MMX [25] oder SSE [26].

ANWENDUNGSBEISPIELE

Unter den unzähligen Anwendungen von CORDIC-Algorithmen seien einige exemplarische aufgelistet.

- Roboter-Steuerung: (inverse) Kinematik und Dynamik [27];
- Transformationen in der generative Computer-Graphik, z.B. 2-D, 3-D Rotation, Normalisierung, 3D-Graphik und Animationen [28];
- spherical vector interpolation für Phong shading bei Berücksichtigung des level of detail/HVS, gesteuert durch die Anzahl der CORDIC-Interationen [29];
- $\{X_n: 0 \leq n < N\} \longrightarrow \{Y_k = X_0 e^{-j2\pi\, 0\, k/N} + X_1 e^{-j2\pi\, 1\, k/N} + X_2 e^{-j2\pi 2k/N} + \ldots + X_n e^{-j2\pi\, n\, k/N}: 0 \leq k < N\}$, DFT per pipelined CORDIC, DCT, Wavelets [16][30-32];
- Raster-Bilder in real time drehen (parallele Rotation von 8x8 pixel-Blöcken) [33];
- Preprocessing, d.h. Normalisieren per Rotation, Skalierung für Handschriften-Erkennung etwa auf PDAs [34];
- Motion Estimation statt per block matching per matching in DCT domain [35];
- Motion Estimation per Hough Transformation (Strecken der polygonalen Kontour in Hesse'scher Normalform) [36], für Video-Coding wie MPEG, H.261, H.263 [37].

EMPFEHLUNGEN FÜR DIE TECHNISCHE INFORMATIK

Das Gesagte legt nahe, CORDIC-Algorithmen – ihrer Bedeutung entsprechend – in

- Mathematik (Additionstheoreme, Geometrie, Konvergenz, Algorithmik, Komplexität, Fehler-Analyse),
- Rechner-Strukturen/Computer-Architektur (Festkomma- vs Gleitkomma-Arithmetik, sign-digit-representation, Speicher-Effizienz, pipelining, Parallelisierung),
- generative Computer Graphik, imaging, Signal-verarbeitung, etc,

zu berücksichtigen, um so Anwendungsbezogenes, Fächer-übergreifendes Lehren und Lernen zu stärken.

Naturgemäß sind die angestellten Überlegungen in

erster Linie einschlägig nur für die Technische Informatik. Andere ingenieurwissenschaftliche Studiengänge haben sicher ihre eigenen Algorithmen, die in exemplarischer Weise die Pole Theorie und Praxis, Grundlagen und Anwendung verbinden.

Tendenziell schwächere Vorkenntnisse der Studienanfänger gerade in Mathematik, tendenziell verkürzte Studienzeiten und tendenziell weiter modularisierte Studiengänge, die integrierendes, Fächer-übergreifendes, interdisziplinäres Lehren und Lernen erschweren, stehen solchen Wünschen, diese Pole zu verbinden, allerdings grundsätzlich entgegen.

REFERENZEN

Im Folgenden sind einige ausgewählte Referenzen zu CORDIC-Algorithmen aus [38] chronologisch aufgelistet, u.a.

- die beiden seminal papers [12][13],
- Überblicksartikel, z.B. [15],
- Genauigkeit von CORDIC-Algorithmen, z.B. [18],
- Implementierungen, z.B. pipeline, processor array, systolic arrays,
- viele Anwendungen,
- Reduktion der Verlustleistung, z.B. [19-22],
- Einsatz in Gewährleistungsarchitekturen, etwa für Fehler-Toleranz, z.B. [23],

gefolgt von Lehrbüchern der Mathematik und einigen wenigen Referenzen auf papers zu Aspekten der Computer-Architektur.

1. Hartmann, P., *Mathematik für Informatiker*. Wiesbaden: Vieweg (2004).
2. Papula, L., *Mathematik für Ingenieure*. Mehrbändig, Wiesbaden: Vieweg (2001).
3. Stingl, P., *Mathematik für Fachhochschulen – Technik und Informatik*. München: Hanser (2003).
4. Stöcker, H., *Mathematik – Der Grundkurs*. Mehrbändig, Frankfurt am Main: Harri Deutsch (1995-1999).
5. Brauch, W., Dreyer, H-J. und Haacke, W., *Mathematik für Ingenieure*. Leipzig: Teubner (2003).
6. Burg, K., Haf, H., Wille, F., *Höhere Mathematik für Ingenieure*. Mehrbändig, Leipzig: Teubner (1993-2002).
7. Lillis, K.M., MASM and Intel Documentation: Reference Guide, Chapter 5: Coprocessors. St. Ambrose University (2000), http://web.sau.edu/LillisKevinM/csci240/masmdocs/ bzw. http://web.sau.edu/LillisKevinM/csci240/masmdocs/reference/09LMARFC05.pdf
8. Microsoft Corp., Notes on Intel® Pentium™ Processor (1993), http://www.singlix.org/trdos/pentium.txt
9. Dudacek, K., 80x87 Instruction Set (x87 – Pentium). University of West Bohemia (2002), http://home.zcu.cz/~dudacek/SOJ/manualy/80x87set.pdf
10. Intel Architecture Software Developer's Manual, Vol.2: Instruction Set Reference. (1997), http://www.mit.edu/afs/sipb/contrib/doc/specs/ic/cpu/x86/iapx-v2.pdf
11. Story, S. et al, Highly Optimized Mathematical Functions for the Itanium Processor. Presentation at the Intel Developer Forum (2001), http://cnscenter.future.co.kr/resource/rsc-center/presentation/intel/spring2001/soss049.pdf
12. Volder, J.E., The CORDIC trigonometric computing technique. *IRE Trans. Electronic Computers*, 8, September, 330-334 (1959).
13. Walther, J.S., A unified algorithm for elementary functions. *Spring Joint Computer Conf.*, Atlantic City, USA, 379-385 (1971).
14. Risse, T., Numerik – Interaktiv (2004), http://www.weblearn.hs-bremen.de/risse/MAI/docs/numerik.pdf bzw. http://www.weblearn.hs-bremen.de/risse/MAI/docs/numerics.pdf
15. Hu, Y.H., CORDIC based vlsi-architectures for digital signal processing. *IEEE Signal Processing Magazine*, July, 16-35 (1992).
16. Yu, S. und Swartzlander, E.E.Jr., A scaled DCT architecture with the CORDIC algorithm. *IEEE Transactions on Signal Processing*, 50, **1** 160-167 (2002), http://www.ece.utexas.edu/\~sungwook/zPDF/sc.pdf
17. Patterson, D.A. und Hennessy, J.L., *Computer Organisation and Design* (2nd Ausg.). San Francisco: Morgan Kaufmann (1998).
18. Hu, Y.H., The quantization effects of the CORDIC algorithm. *IEEE Trans. Signal Processing*, 40, **4**, April, 834-844 (1992).
19. Schimpfle, C.V., Simon, S. und Nossek, J.A., Low power CORDIC implementation using redundant number representation. *IEEE Inter. Conf. on Application-Specific Systems, Architectures and Processors (ASAP 1997)*, Zürich, Schweiz, 154-161 (1997).
20. Juang, T.B. und Hsiao, S.F., A low power and fast CORDIC processor for vector rotation. *Proc. 42th IEEE Midwest Symp. on Circuits and Systems*, Las Cruces, USA, 81-83 (1999).
21. Costello, J. und Khalili, D-A., Behavioural synthesis of low power floating point CORDIC processors. *Proc. Inter. Conf. on Electronics,*

Circuit and Systems, ICECS 2000, Beirut, Libanon, 506-509 (2000).
22. Krstic, M., Troya, A., Maharatna, K. und Grass, E., Optimized low power synchronizer design for the IEEE 802.11a Standard. *Proc. Inter. Conf. on Acoustics, Speech, and Signal Processing*, Hong Kong, China, II-333–II-336 (2003).
23. Kwak, J-H., Piuri, V. und Schwartzlander, E.E.Jr., Fault-tolerant high-performance CORDIC processor. *Proc. Inter. Symp. on Defect and Fault Tolerance in VLSI Systems*, Yamanashi, Japan, 164-172 (2000).
24. Simon, S., Müller, M., Gryska, H., Wortmann, A. und Buch, S., An instruction set for the efficient implementation of the CORDIC algorithm. *Proc. IEEE Inter. Symp. on Circuits and Systems (ISCAS 2004)*, Vancouver, Canada (2004).
25. Villalba, J., Hormigo, J., González, M.A. und Zapata, E.L., MMX-like architecture extension to support the rotation operation. *Proc. IEEE Inter. Conf. on Multimedia and Expo. (ICME 2000)*, 3, New York, USA, 1383-1386 (2000).
26. Nyland, L. und Snyder, M., Fast Trigonometric Functions Using Intel's SSE2 Instructions. Technical Report TR03-041, University of North Carolina at Chapel Hill (2003), ftp://ftp.cs.unc.edu/pub/publications/techreports/03-041.pdf
27. Lee, C.S.G. und Chang, P.R., A maximum pipelined CORDIC architecture for inverse kinematic position computation. *IEEE J. of Robotics and Automation*, 3, **5**, 445-458 (1987).
28. Lang, T. und Antelo, E., High-throughput 3D rotations and normalizations. *Proc. 35th Asilomar Conf. on Signals, Systems and Computers*, Pacific Grove, USA, 846-851 (2001).
29. Euh, J., Chittamuru, J. und Burleson, W., CORDIC vector interpolator for power aware 3D computer graphics. *IEEE Workshop on Signal Processing Systems (SIPS2002)*, San Diego, USA, 240-245 (2002).
30. Simon, S., Rieder, P., Schimpfle, C.V. und Nossek, J.A., CORDIC based architectures for the efficient implementation of discrete wavelet transforms. *Proc. IEEE Inter. Symp. on Circuits and Systems (ISCAS 1996)*, 4, Atlanta, USA, 77-80 (1964).
31. Rieder, P., Simon, S. und Schimpfle, C.V., Application specific efficient VLSI architectures for orthogonal single- and multiwavelet transform. *J. VLSI Signal Processing – Systems for Signal, Image, and Video Technology*, 21, **2**, 77-90 (1999).
32. Wu, Z., Ren, G. und Zhao, Y., A study on implementing wavelet transform and FFT with FPGA. *Proc. 4th Inter. Conf. (ASIC 2001)*, Shanghai, China, 486-489 (2001).
33. Gosh, I., Bandana majumdar: design of an application specific VLSI chip for image rotation. *Proc. 7th Inter. Conf. on VLSI Design*, Kalkutta, Indien, 275-278 (1994).
34. McInerney, S. und Reilly, R.B., Hybrid multiplier CORDIC unit for online handwriting recognition. *Proc. Inter. Conf. on Acoustics, Speech, and Signal Processing*, 4, Phoenix, USA, 1909-1912 (1999).
35. Chen, J. und Liu, K.J.R., A complete pipelined parallel CORDIC architecture for motion estimation. *IEEE Trans. on Circuits and Systems – II: Analog and Digital Signal Processing*, 45, **6**, 653-660 (1998).
36. Li, H-L. und Chakrabarti, C., Hardware design of a 2-D motion estimation system based on the Hough-transform. *IEEE Trans. on Circuits and Systems – II: Analog and Digital Signal Processing*, 45, **1**, 80-95 (1998).
37. Chen, J. und Liu, K.J.R., Efficient architecture and design of an embedded video coding engine. *IEEE Trans. on Multimedia*, 3, **3**, 285-297 (2001).
38. Risse, T., References CORDIC & Signal Processing (2004), http://www.weblearn.hs-bremen.de/risse/papers/CORDIC.doc/CORDICreferences.pdf

BIOGRAPHIE

Thomas Risse ist Diplom-Mathematiker und promovierter Physiker. Derzeit arbeitet er als Professor für Rechnerstrukturen, Computer-Graphik und Mathematik im Fachbereich Elektrotechnik & Informatik der Hochschule Bremen.

In der Forschung war er z.B. als visiting scientist am T.J. Watson Research Center, Yorktown Heigths der IBM u.a. zu den Themen: Fehlertolerante Systeme und Computer-Graphik tätig. Er konnte Erfahrungen in seinen praktischen Tätigkeiten u.a. in den Bereichen Datenschutz und Datensicherheit sowie Medizin-Informatik sammeln.