INTRODUCING KRONOS: A NOVEL APPROACH TO SIGNAL PROCESSING LANGUAGES

Vesa Norilo

Centre for Music & Technology
Sibelius–Academy

Linux Audio Conference, 2011
INTRODUCTION

MOTIVATION

The Current State of DSP Programming
Why Yet Another Programming Language?

KRONOS – AN OVERVIEW

A Language Specification
A Just-in-Time Compiler
Type Determinism

CASE STUDIES

Examples
INTRODUCTION

- Research background: *PWGL*
INTRODUCTION

• Research background: PWGL
  • Musical programming environment by Laurson, Kuuskankare, Norilo, Sprotte
  • High level visual interface to LISP programming
  • Synthesizer component in C++ written by the author: PWGLSynth
Introduction

- Research background: PWGL
  - Musical programming environment by Laurson, Kuuskankare, Norilo, Sprotte
  - High level visual interface to LISP programming
  - Synthesizer component in C++ written by the author: PWGLSynth
- Kronos began as a bunch of aspirations for PWGLSynth 2
Introduction

- Research background: PWGL
  - Musical programming environment by Laurson, Kuuskankare, Norilo, Sprotte
  - High level visual interface to LISP programming
  - Synthesizer component in C++ written by the author: PWGLSynth
- Kronos began as a bunch of aspirations for PWGLSynth 2
  - Generic computation engine
  - High level abstraction
  - Great performance
INTRODUCTION

- Research background: PWGL
  - Musical programming environment by Laurson, Kuuskankare, Norilo, Sprotte
  - High level visual interface to LISP programming
  - Synthesizer component in C++ written by the author: PWGLSynth
- Kronos began as a bunch of aspirations for PWGLSynth 2
  - Generic computation engine
  - High level abstraction
  - Great performance
- Since then, Kronos has morphed into a standalone compiler/language
INTRODUCTION

• Research background: PWGL
  • Musical programming environment by Laurson, Kuuskankare, Norilo, Sprotte
  • High level visual interface to LISP programming
  • Synthesizer component in C++ written by the author: PWGLSynth
• Kronos began as a bunch of aspirations for PWGLSynth 2
  • Generic computation engine
  • High level abstraction
  • Great performance
• Since then, Kronos has morphed into a standalone compiler/language
• Doctoral study project since 2010
MOTIVATION
THE CURRENT STATE OF DSP PROGRAMMING
The Current State of DSP Programming

- The industry standard for DSP is C
The Current State of DSP Programming

- The industry standard for DSP is C
- Plugins and DSP chips tend to have C toolchains
The Current State of DSP Programming

- The industry standard for DSP is C
- Plugins and DSP chips tend to have C toolchains
  - AU, VST, LADSPA
  - Motorola 56k
  - etc..


The Current State of DSP Programming

- The industry standard for DSP is C
- Plugins and DSP chips tend to have C toolchains
  - AU, VST, LADSPA
  - Motorola 56k
  - etc..
- C is relatively hostile to casual programmers
The Current State of DSP Programming

- The industry standard for DSP is C
- Plugins and DSP chips tend to have C toolchains
  - AU, VST, LADSPA
  - Motorola 56k
  - etc..
- C is relatively hostile to casual programmers
  - Casual programmers make a lot of musical applications!
  - Getting audio out of C is very difficult for learners
THE CURRENT STATE OF DSP PROGRAMMING

- The industry standard for DSP is C
- Plugins and DSP chips tend to have C toolchains
  - AU, VST, LADSPA
  - Motorola 56k
  - etc..
- C is relatively hostile to casual programmers
  - Casual programmers make a lot of musical applications!
  - Getting audio out of C is very difficult for learners
- High performance programs are low level
THE CURRENT STATE OF DSP PROGRAMMING

- The industry standard for DSP is C
- Plugins and DSP chips tend to have C toolchains
  - AU, VST, LADSPA
  - Motorola 56k
  - etc..
- C is relatively hostile to casual programmers
  - Casual programmers make a lot of musical applications!
  - Getting audio out of C is very difficult for learners
- High performance programs are low level
  - Many powerful abstractions have performance penalties
  - Tedious to write for professionals
Why Yet Another Programming Language?

or what if it would be possible to...
Why Yet Another Programming Language?

OR what if it would be possible to...

- easily learn an audio language
**Why Yet Another Programming Language?**

or *what if it would be possible to...*

- easily learn an audio language
- write abstract, reusable code that runs *fast*
Why Yet Another Programming Language?

OR what if it would be possible to...

- easily learn an audio language
- write abstract, reusable code that runs fast
- design all your algorithms down to the arithmetic primitive
Why Yet Another Programming Language?

Or *what if it would be possible to*...

- easily learn an audio language
- write abstract, reusable code that runs **fast**
- design all your algorithms down to the arithmetic primitive
- have a single filter for any combination of single or double precision, real or complex, mono or multichannel...
Why Yet Another Programming Language?

Or what if it would be possible to...

- easily learn an audio language
- write abstract, reusable code that runs fast
- design all your algorithms down to the arithmetic primitive
- have a single filter for any combination of single or double precision, real or complex, mono or multichannel...

Many of us here are working on a subset of these problems. The final solution is not yet here.
KRONOS
an Overview
A LANGUAGE SPECIFICATION

SYNTAX
A LANGUAGE SPECIFICATION

SYNTAX

• Simple syntax
A Language Specification

Syntax

- Simple syntax
  - Familiar function notation
    SomeFunction(param1 param2)

Infix functions for arithmetics
a + b * 3 / Sqrt(c)

Algebraic data structure yields pairs, lists and trees
(list = (a b c d))

Tie-in allows for partial decomposition too
(first-element other-elements) = list
A LANGUAGE SPECIFICATION

SYNTAX

• Simple syntax
  • Familiar function notation
    SomeFunction(param1 param2)
  • Infix functions for arithmetics
    a + b * 3 / Sqrt(c)
A LANGUAGE SPECIFICATION

SYNTAX

- Simple syntax
  - Familiar function notation
    SomeFunction(param1 param2)
  - Infix functions for arithmetics
    a + b * 3 / Sqrt(c)
  - Algebraic data structure yields pairs, lists and trees
    list = (a b c d)
A LANGUAGE SPECIFICATION

SYNTAX

- Simple syntax
  - Familiar function notation
    SomeFunction(param1 param2)
  - Infix functions for arithmetics
    a + b * 3 / Sqrt(c)
  - Algebraic data structure yields pairs, lists and trees
    list = (a b c d)
  - Tie-in allows for partial decomposition too
    (first-element other-elements) = list
A LANGUAGE SPECIFICATION

CLASSIFICATION
A LANGUAGE SPECIFICATION

CLASSIFICATION

- Functional Programming
A LANGUAGE SPECIFICATION

CLASSIFICATION

• Functional Programming
  • No state
A LANGUAGE SPECIFICATION

CLASSIFICATION

- Functional Programming
  - No state
  - No variables
A LANGUAGE SPECIFICATION

CLASSIFICATION

- Functional Programming
  - No state
  - No variables
  - Similar to audio signal routing
A LANGUAGE SPECIFICATION

CLASSIFICATION

• Functional Programming
  • No state
  • No variables
  • Similar to audio signal routing
  • Powerful abstraction

• Reactive Paradigm
  • Action is followed by reaction
  • Reactive graphs are used to optimize signal rates
  • Implicit inferral of control and audio signals
A LANGUAGE SPECIFICATION

CLASSIFICATION

- Functional Programming
  - No state
  - No variables
  - Similar to audio signal routing
  - Powerful abstraction
- Reactive Paradigm
A Language Specification

Classification

• Functional Programming
  • No state
  • No variables
  • Similar to audio signal routing
  • Powerful abstraction

• Reactive Paradigm
  • *Action* is followed by *reaction*
A LANGUAGE SPECIFICATION

CLASSIFICATION

- Functional Programming
  - No state
  - No variables
  - Similar to audio signal routing
  - Powerful abstraction

- Reactive Paradigm
  - *Action* is followed by *reaction*
  - Reactive graphs are used to optimize signal rates
A Language Specification

Classification

- Functional Programming
  - No state
  - No variables
  - Similar to audio signal routing
  - Powerful abstraction
- Reactive Paradigm
  - *Action* is followed by *reaction*
  - Reactive graphs are used to optimize signal rates
  - Implicit inferral of control and audio signals
Listing 1: Fold, a higher order function for reducing lists with example replies.

Fold(folding-function x)
{
    Fold = x
}

Fold(folding-function x xs)
{
    Fold = Eval(folding-function x Fold(folding-function xs))
}

/* Add several numbers */
Fold(Add 1 2 3 4) => 10
/* Multiply several numbers */
Fold(Mul 5 6 10) => 300
A JUST-IN-TIME COMPILER
A Just-in-Time Compiler

- *Kronos* compiles programs on the fly to native *x86*
- Some enhancements to *SoftWire*, a LGPL dynamic assembler written by *Nicolas Capens*
A JUST-IN-TIME COMPILER

- Kronos compiles programs on the fly to native x86
  - Some enhancements to SoftWire, a LGPL dynamic assembler written by Nicolas Capens
- Programs are configured on the fly for the present I/O configuration
A JUST-IN-TIME COMPILER

- **Kronos** compiles programs on the fly to native x86
  - Some enhancements to *SoftWire*, a LGPL dynamic assembler written by *Nicolas Capens*
- Programs are configured on the fly for the present I/O configuration
- Good runtime performance using SSE4.2
A JUST-IN-TIME COMPILER

- **Kronos** compiles programs on the fly to native x86
  - Some enhancements to *SoftWire*, a LGPL dynamic assembler written by *Nicolas Capens*
- Programs are configured on the fly for the present I/O configuration
- Good runtime performance using SSE4.2
  - Comparable and often superior to an optimizing C-compiler
TYPE DETERMINISM

...or The Catch

• ...too good to be true?
TYPE DETERMINISM

...or The Catch

- ...too good to be true?
- To achieve performance, a rigorous constraint;
Type Determinism

...or The Catch

- ...too good to be true?
- To achieve performance, a rigorous constraint;
- **Type Determinism**
TYPE DETERMINISM

...or The Catch

- ...too good to be true?
- To achieve performance, a rigorous constraint;
- TYPE DETERMINISM
  - The result type of an expression can only depend on argument type
TYPE DETERMINISM

...or The Catch

- ...too good to be true?
- To achieve performance, a rigorous constraint;
- **TYPE DETERMINISM**
  - The result type of an expression can only depend on argument type
  - Complete dataflow analysis of the entire program
Type Determinism

...or The Catch

- ...too good to be true?
- To achieve performance, a rigorous constraint;
- **Type Determinism**
  - The result type of an expression can only depend on argument type
  - Complete dataflow analysis of the entire program
  - *Result type can’t depend on data!*
**Type Determinism**

...or The Catch

- ...too good to be true?
- To achieve performance, a rigorous constraint;
- **Type Determinism**
  - The result type of an expression can only depend on argument type
  - Complete dataflow analysis of the entire program
  - Result type can’t depend on data!
  - No dynamic containers!
**Type Determinism**

...or The Catch

- ...too good to be true?
- To achieve performance, a rigorous constraint;

**Type Determinism**

- The result type of an expression can only depend on argument type
- Complete dataflow analysis of the entire program
- *Result type can’t depend on data!*
- No dynamic containers!
- Minimal branching!
TYPE DETERMINISM

...or The Catch

- ...too good to be true?
- To achieve performance, a rigorous constraint;
- **TYPE DETERMINISM**
  - The result type of an expression can only depend on argument type
  - Complete dataflow analysis of the entire program
  - *Result type can’t depend on data!*
  - No dynamic containers!
  - Minimal branching!
- No good for writing a word processor
TYPE DETERMINISM

...or The Catch

- ...too good to be true?
- To achieve performance, a rigorous constraint;
- **TYPE DETERMINISM**
  - The result type of an expression can only depend on argument type
  - Complete dataflow analysis of the entire program
  - *Result type can’t depend on data!*
  - No dynamic containers!
  - Minimal branching!
- No good for writing a word processor
- Designed for *DSP inner loops*
<table>
<thead>
<tr>
<th>Introduction</th>
<th>Motivation</th>
<th>Kronos – an Overview</th>
<th>Case Studies</th>
<th>Summary</th>
<th>Ending</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Case Studies</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Case Studies
Signal Generation

- Delay is a first-class unary operator
- Recursion permitted via delays
- Recursion can be turned into an osc by *clocking* a section of the loop
  - IO:Audio-Gen(sig) provides signal updates at the audio rate

Listing 2: A Simple Phasor Oscillator

```
Phasor(freq)
{
    next-phase = IO:Audio-Gen(z-1('0 phase + freq))
    phase = next-phase - Truncate(next-phase)
    Phasor = phase + phase - 1
}
```
**Fast $-cos(\pi x)$ for $x \in [-1, 1]$**

- Source is the math definition of series approximation!

**Listing 3: Taylor-series Cosine**

```plaintext
Sine-Coef(n)
{
  Sine-Coef = Crt:pow(Pi n * #2 - #1) * Crt:pow(#-1 n + #1) / Factorial(n * #2 - #1)
}

Fast-Cos(x order)
{
  Use Algorithm
  xm = Abs(x)
  xp = xm - 0.5
  coefs = Map(Sine-Coef Count-To(order))
  Fast-Cos = xp * Horner-Polynomial(xp * xp Reverse(coefs))
}
```
Combining the $-\cos(\pi x)$ mapper and the phasor, a sine oscillator is created.

With higher order functions, oscillator banks can be constructed from a list of frequencies!

Listing 4: Sinusoid oscillation by mapping the phasor

```c
FSin(freq)
{
    FSin = Fast-Cos(Phasor(freq) #8)
}

/* Example of using higher order functions */
Osc-Bank = Reduce(Add Map(FSin freq1 freq2 freq3 freq4))
```
**SIN-OSC APPLICATIONS**

- Additive and FM Synthesis are easily constructed

**Listing 5: Sinusoid Synthesis**

```plaintext
Suppress-Alias(f0 amp) {Suppress-Alias = (f0 < 0.4) & amp}

Additive(f0 num-harmonics harmonic-coef harmonic-spread)
{
    Use Algorithm
    freqs = \text{Map}(\text{Curry}(\text{Mul} f0) \text{ Expand}(\text{num-harmonics} \text{ Curry}(\text{Add harmonic-spread}) 1))
    amps = \text{Expand}(\text{num-harmonics} \text{ Curry}(\text{Mul} \text{ harmonic-coef}) 1)
    oscs = \text{Zip-With}(\text{Mul}(\text{Map}(\text{FSin} \text{ freqs}) \text{ Zip-With}(\text{\text{Suppress-Alias} freqs \text{ amps})}))
    Additive = \text{Reduce}(\text{Add oscs})
}

FM(f0 ratio mod feedback)
{
    modulator = \text{FSin}(f0 * ratio) * mod
    carrier = \text{FSin}((1 + \text{modulator} + \text{feedback} \ast \text{z-1('0 carrier)}) \ast f0)
    FM = carrier
}
```
INTERFACING WITH CONTROL

- Other *clocks* besides audio generators can be used
- Resonator coefficient computations are clocked from OSC
  - Only recomputes coefs when OSC signal arrives!

Listing 6: OSC-controlled Resonator Bank

```plaintext
Reson(x0 freq reson)
{
  x1 = z-1('0 x0) x2 = z-1('0 x1) y1 = z-1('0 y0) y2 = z-1('0 y1)
  r = Crt:pow(reson 0.125)
  y0 = x0 - x2 + y1 * 2 * r * Crt:cos(freq) - y2 * r * r
  Reson = y0 * 0.5 * (1 - r * r)
}

Reson-Bank()
{
  Use Algorithm
  params = ((IO:OSC-Input("cutoff1" Float) IO:OSC-Input("reson1" Float))
            (IO:OSC-Input("cutoff2" Float) IO:OSC-Input("reson2" Float))
            (IO:OSC-Input("cutoff3" Float) IO:OSC-Input("reson3" Float)))
  Reson-Bank = Reduce(Add Map(Curry(Reson Noise()) params))
}
```
Schroeder Reverberation

- Classic Schroeder reverberation can be concisely expressed

Listing 7: Classical Schroeder Reverb

```plaintext
Feedback-for-RT60(rt60 delay)
{
    Feedback-for-RT60 = Crt:pow(#0.001 delay / rt60)
}

Basic(sig rt60)
{
    Use Algorithm
    allpass-params = ((0.7 #221) (0.7 #75))
    delay-times = (#1310 #1636 #1813 #1927)
    feedbacks = Map(Curry(Feedback-for-RT60 rt60) delay-times)

    comb-section = Reduce(Add Zip-With(Curry(Delay sig) feedbacks delay-times))
    Basic = Cascade(Allpass-Comb comb-section allpass-params)
}
```
FDN Reverberation

- Some highlights from a FDN reverberator

Listing 8: Snippets of a 16th order FDN reverberator

```c
/* Orthogonal matrix multiply - Householder Algorithm */
Feedback-Mtx(input)
{
    Use Algorithm
    Feedback-Mtx = input
    (even odd) = Split(input)
    even-mtx = Recur(even) odd-mtx = Recur(odd)
    Feedback-Mtx = Append(Zip-With(Add even-mtx odd-mtx) Zip-With(Sub even-mtx
        odd-mtx))
}
/* 16-channel feedback signal recursively passed through a unit delay operator */
feedback-vector = z-1('0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
    Zip-With(Mul loss-coefs Zip-With(Filter:OnePole
        Feedback-Mtx(delay-vector) filter-coefs)))
```
SUMMARY

• Kronos combines high level audio programs with high performance – for both beginners and professionals
• The tradeoff of Type Determinism enables this unusual combination
• Kronos will be released as a C-callable library. Licensing options including dual-licensing with GPL are being investigated.
• Outlook
  • Development is in early stages. Debugging the compiler and designing the run time libraries are ongoing
  • A graphical user interface to the language should be created
  • The signal rate optimization should be further improved
SUMMARY

• Kronos combines high level audio programs with high performance – for both beginners and professionals
Summary

- Kronos combines high level audio programs with high performance – for both beginners and professionals
- The tradeoff of Type Determinism enables this unusual combination
Summary

- Kronos combines high level audio programs with high performance – for both beginners and professionals
- The tradeoff of Type Determinism enables this unusual combination
- Kronos will be released as a C-callable library. Licensing options including dual-licensing with GPL are being investigated.
SUMMARY

• Kronos combines high level audio programs with high performance – for both beginners and professionals
• The tradeoff of Type Determinism enables this unusual combination
• Kronos will be released as a C-callable library. Licensing options including dual-licensing with GPL are being investigated.

• Outlook
SUMMARY

- Kronos combines high level audio programs with high performance – for both beginners and professionals
- The tradeoff of Type Determinism enables this unusual combination
- Kronos will be released as a C-callable library. Licensing options including dual-licensing with GPL are being investigated.

Outlook
- Development is in early stages. Debugging the compiler and designing the run time libraries are ongoing
SUMMARY

- Kronos combines high level audio programs with high performance – for both beginners and professionals
- The tradeoff of Type Determinism enables this unusual combination
- Kronos will be released as a C-callable library. Licensing options including dual-licensing with GPL are being investigated.

Outlook
- Development is in early stages. Debugging the compiler and designing the run time libraries are ongoing
- A graphical user interface to the language should be created
SUMMARY

- Kronos combines high level audio programs with high performance – for both beginners and professionals
- The tradeoff of Type Determinism enables this unusual combination
- Kronos will be released as a C-callable library. Licensing options including dual-licensing with GPL are being investigated.

Outlook
- Development is in early stages. Debugging the compiler and designing the run time libraries are ongoing
- A graphical user interface to the language should be created
- The signal rate optimization should be further improved
Summary

- Kronos combines high level audio programs with high performance – for both beginners and professionals
- The tradeoff of Type Determinism enables this unusual combination
- Kronos will be released as a C-callable library. Licensing options including dual-licensing with GPL are being investigated.

Outlook
- Development is in early stages. Debugging the compiler and designing the run time libraries are ongoing
- A graphical user interface to the language should be created
- The signal rate optimization should be further improved
THANK YOU!

...questions?