Code Constructor User Tool

supported by

- Canadian Foundation for Innovation
- Ontario Innovation Trust (?!)
- Apple Canada
- McMaster University

Christopher Anand
Wolfram Kahl

Computing and
Software

McMaster
University

Faculty of
Engineering

Coconut

Christopher Anand

3 years Medical Electronics Industry

- 4 patents + 2 pending
- real-time, parallel MRI,
  - rack of PPC604/750s
- best result: 1000X performance
  - mathematical transformation
  - efficient implementation

Wolfram Kahl

HOPS

- term-graph-based
  - program transformation
  - code generation
- derivation of correct programs
- functional programming
- relation-algebraic methods

Coconut

- targeting numerical code
  - signal processing
  - image processing
  - math libraries

- goals
  - more reliable
  - faster

Christopher Anand
Wolfram Kahl

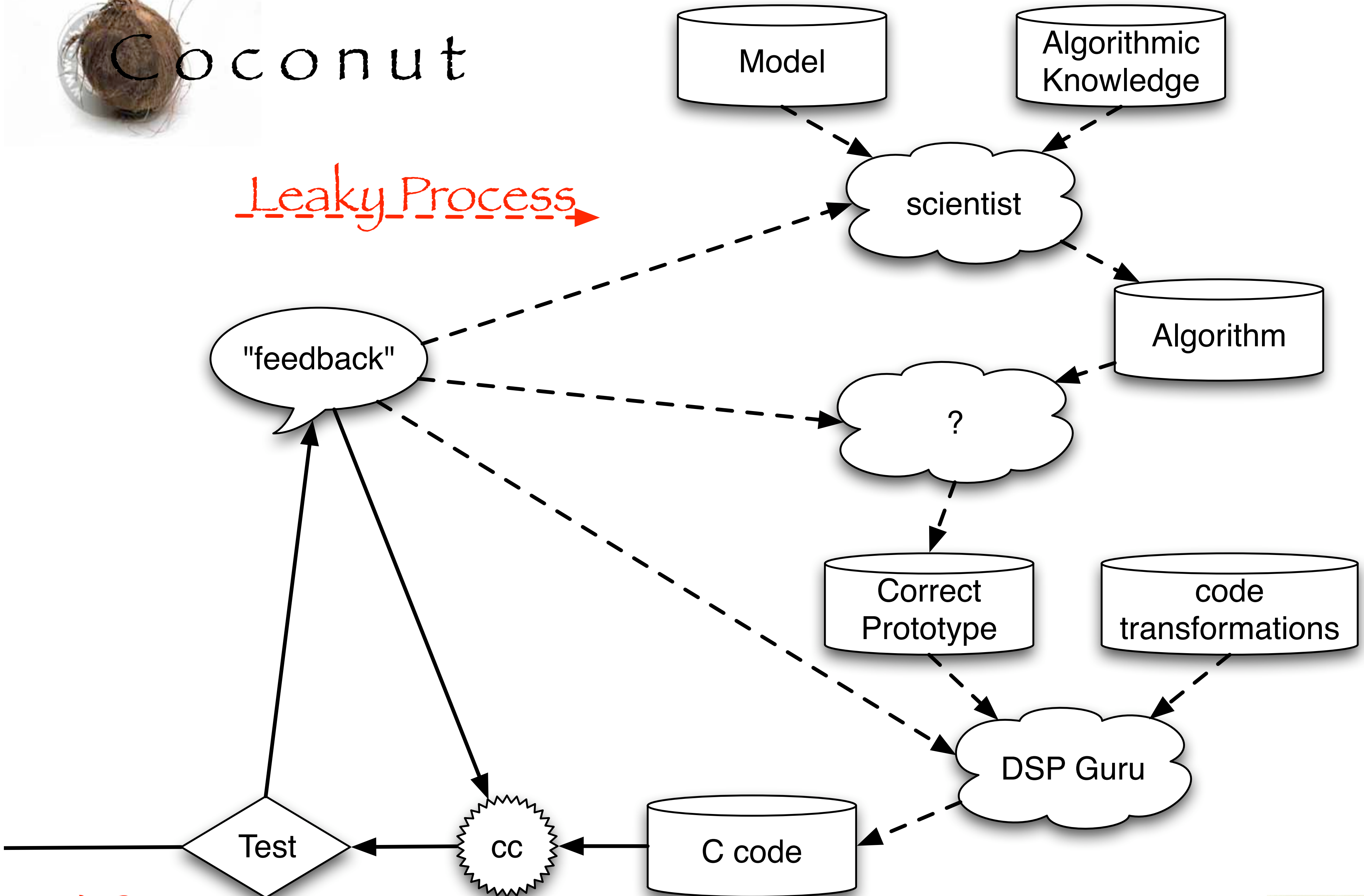
Computing and
Software

McMaster
University

Faculty of
Engineering

Coconut

Leaky Process



Christopher Anand
Wolfram Kahl

Computing and Software

McMaster University

Faculty of Engineering

Coconut

Plug leaks!

Christopher Anand
Wolfram Kahl

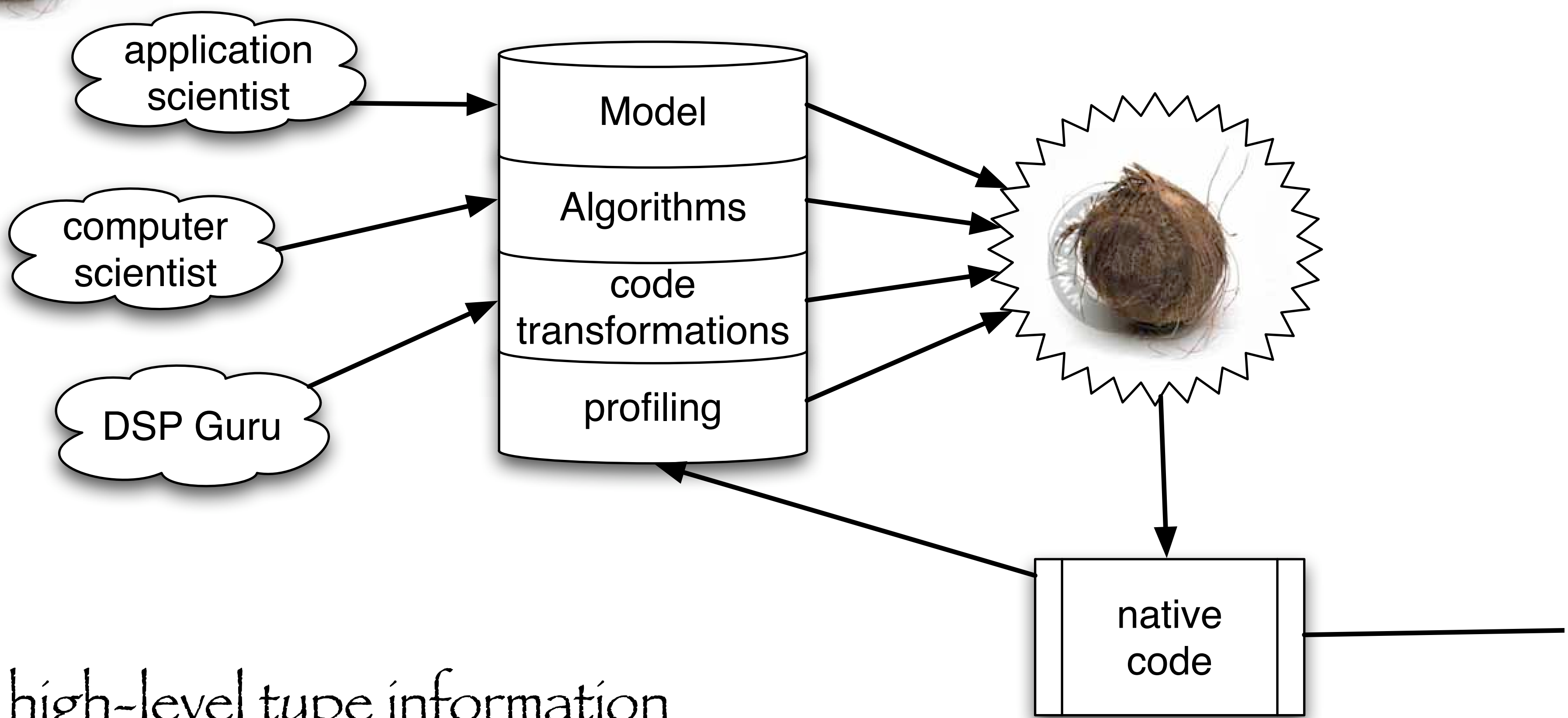
Computing and
Software

McMaster
University

Faculty of
Engineering

Coconut

no pipes ⇒ no leaks



high-level type information

+

machine specifics

Christopher Anand
Wolfram Kahl

Computing and Software

McMaster University

Faculty of Engineering

Coconut

- faster via
  - global data transformations
    - arrays of structs of arrays
    - alignment
    - virtual objects
  - leverage
    - SIMD (VMX)
    - cache hierarchy

Christopher Anand
Wolfram Kahl
Computing and Software
McMaster University
Faculty of Engineering

Coconut

- fast via
  - high-level code transformations
    - generic BLAS++
    - pipelining without cleanup
    - reason about precision
    - custom virtual machines
    - tricks !

Christopher Anand
Wolfram Kahl

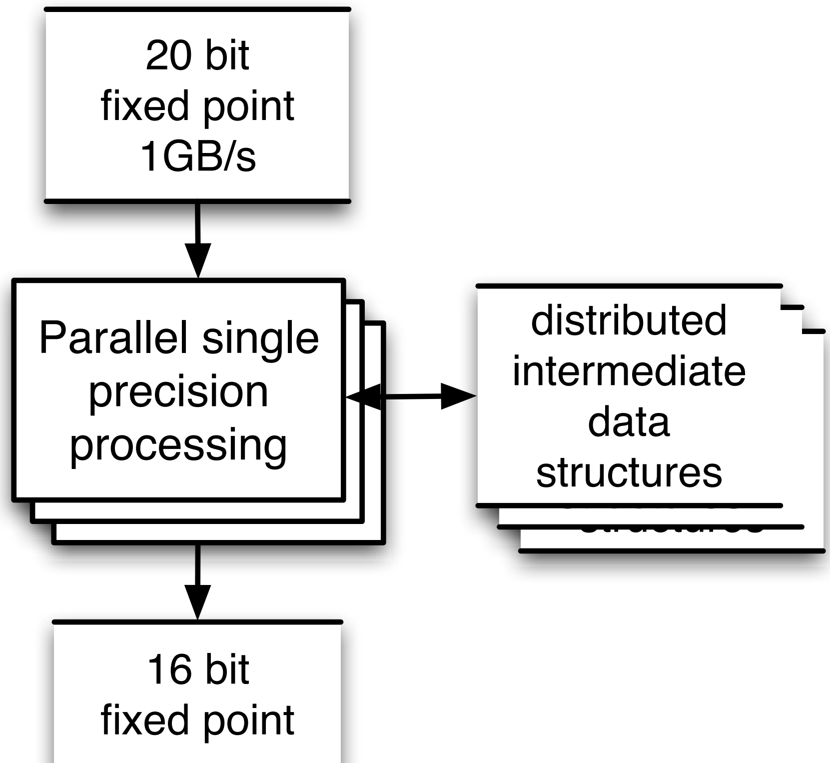
Computing and Software

McMaster University

Faculty of Engineering

Coconut

Magnetic Resonance Imaging



Christopher Anand
Wolfram Kahl

Computing and
Software

McMaster
University

Faculty of
Engineering

Coconut

fast imaging ⇒ nonuniform Fourier Transform ⇒

```
void    vexpif(     float *thetas, int quarterSize, float *sines, float *cosines) {
  int index;
  for (index = 0; index < 4*quarterSize; index++) {
    sines[index] = sin(thetas[index]);
    cosines[index] = cos(thetas[index]);
  }
}
```

want to

- inline
- eliminate storage for thetas
- use VMX/Altivec
- prefetch data

(use VMX/Altivec, prefetch data) } 12X acceleration

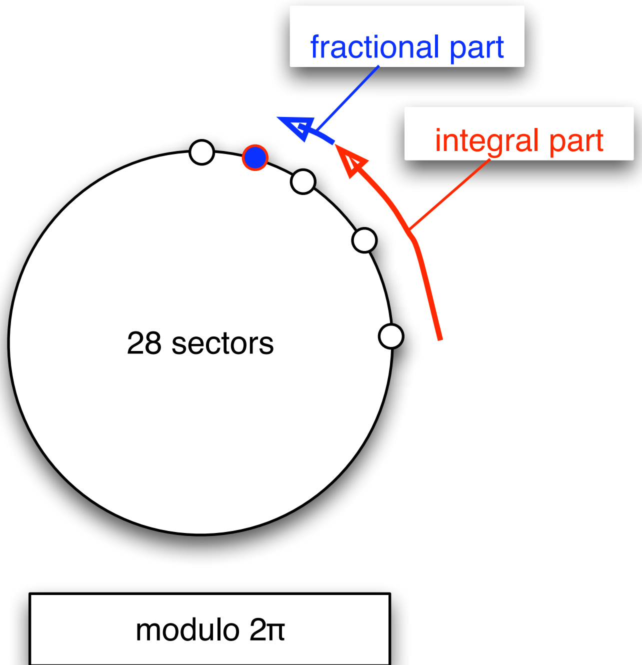
Christopher Anand
Wolfram Kahl

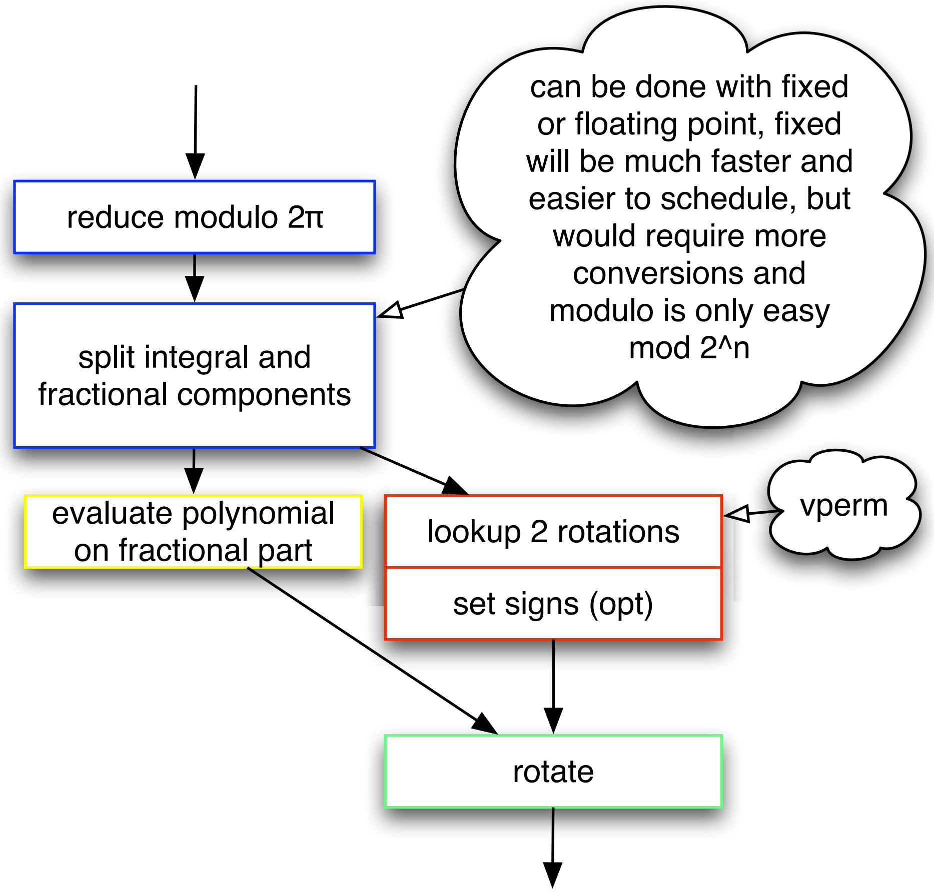
Computing and Software

McMaster University

Faculty of Engineering

Coconut





Christopher Anand
Wolfram Kahl

Computing and Software

McMaster University

Faculty of Engineering

Coconut

```
static unsigned char staticsinLookupLookup[] = {
  0*4,1*4,2*4,3*4,4*4,5*4,6*4,7*4,6*4,5*4,4*4,3*4,2*4,1*4,                        // positive lookups
  0*4,1*4+1,2*4+1,3*4+1,4*4+1,5*4+1,6*4+1,7*4+1,6*4+1,5*4+1,4*4+1,3*4+1,2*4+1,1*4+1, // negative lookups
  0,0,0,0  //unused
};
static unsigned char staticcosLookupLookup[] = {
  7*4,6*4,5*4,4*4,3*4,2*4,1*4,0*4,1*4+1,2*4+1,3*4+1,4*4+1,5*4+1,6*4+1,
  7*4+1,6*4+1,5*4+1,4*4+1,3*4+1,2*4+1,1*4+1,0*4,1*4,2*4,3*4,4*4,5*4,6*4,
  0,0,0,0  //unused
};
static unsigned char staticpartialSplat[] = {
  3,3,3,3,7,7,7,7,11,11,11,11,15,15,15,15
};
static unsigned int staticotherConstants[] = {
  0x3E22F983,//.1591549431,    // oneOverTwoPi
  0x3C924925,//.1785714286e-1, // oneOver56
  0x40C90FDB,//6.283185308,         // twoPi
  0x3E65C8F0,//.2243993266,  // sin1

  0xBAF6999E,//-.1881409241e-2, // sin3
  0x3F800000,//0.9999999996880272,  // cos0
  0xBCCE4126,//-.2517754893e-1, // cos2
  0x38DD614C,//.1055622430e-3  // cos4

  0x41E00000,//28.
  0x00010203,//0..3
  0x0000001C,//28
  0x3E65C8FA,//2PiOver28
};
```


Christopher Anand
Wolfram Kahl

Computing and Software

McMaster University

Faculty of Engineering

Coconut

```
for (count = quarterSize / refreshSize; count--; ) {
  vec_dst(pInput,GetPrefetchConstant(8, 2 * refreshSize * 2 / (8), 8*16),0);
  for (subcount = refreshSize; subcount--; ) {
    theta = vec_ld(0,pInput);
    vfp(theta);
    vOneOver56 = vec_splat(vOtherConstants1,1);
    vOneOver2Pi = vec_splat(vOtherConstants1,0);
    pInput++;
    theta = vec_madd(theta,vOneOver2Pi,vOneOver56);
    intPart = vec_floor(theta);
    v28f = vec_splat(vOtherConstants3,0);
    theta = vec_sub(theta,intPart);     // got (theta/2Pi + 1/2/28) mod 1, offset by 1/2 a segment length
    theta = vec_madd(theta,v28f,vZero);
    intPart = vec_floor(theta);
    theta = vec_sub(theta,intPart);
    (vector unsigned int)temp = vec_splat_u32(1);  // make .5f
    temp = vec_ctf((vector unsigned int)temp,1);
    theta = vec_sub(theta,temp);
    (vector unsigned int)intPart = vec_ctu(intPart,0);
    v28 = vec_splat((vector unsigned int)vOtherConstants3,2);
    vOneToFour = vec_splat((vector unsigned int)vOtherConstants3,1);

    asm("dcbz %0, %1" : : "O" (0), "b" (pCosines));
    asm("dcbz %0, %1" : : "O" (0), "b" (pSines));

    sinSign = vec_perm(vSinLookupLookup1,vSinLookupLookup2,(vector unsigned char)intPart);  // now we have 0x000000PosSign
    sinMap = vec_and(sinSign,(vector unsigned char)v28);
    sinMap = vec_perm(sinMap,sinMap,vPartialSplat);
    sinMap = vec_add(sinMap,vOneToFour);
    sinRot = vec_perm(lookup1,lookup2,sinMap);
    vOne = vec_splat_u32(31);
    (vector unsigned int)sinSign = vec_sl((vector unsigned int)sinSign,vOne);
    sinRot = vec_or((vector float)sinSign,sinRot);

    (vector unsigned char)cosSign = vec_perm(vCosLookupLookup1,vCosLookupLookup2,(vector unsigned char)intPart);  // now we have 0x000000PosSign
    cosMap = vec_and(cosSign,(vector unsigned char)v28);
    cosMap = vec_perm(cosMap,cosMap,vPartialSplat);
    cosMap = vec_add(cosMap,vOneToFour);
    cosRot = vec_perm(lookup1,lookup2,cosMap);
    (vector unsigned int)cosSign = vec_sl((vector unsigned int)cosSign,vOne);
    cosRot = vec_or((vector float)cosSign,cosRot);
    cos4 = vec_splat(vOtherConstants2,3);
    cos2 = vec_splat(vOtherConstants2,2);
    square = vec_madd(theta,theta,vZero);
    cos = vec_madd(square,cos4,cos2);
    cos0 = vec_splat(vOtherConstants2,1);
    cos = vec_madd(square,cos,cos0);
    sin1 = vec_splat(vOtherConstants1,3);
    sin3 = vec_splat(vOtherConstants2,0);
    sin = vec_madd(square,sin3,sin1);
    sin = vec_madd(sin,theta,vZero);
    cosOut = vec_madd(cos,cosRot,vZero);
    sinOut = vec_madd(cos,sinRot,vZero);
    cosOut = vec_nmsub(sin,sinRot,cosOut);
    sinOut = vec_madd(sin,cosRot,sinOut);
    vec_stl(cosOut,0,pCosines);
    vec_stl(sinOut,0,pSines);
    pCosines++;
    pSines++;
  }
```

now

- inline
- pipeline / unroll
- tune prefetch

Christopher Anand
Wolfram Kahl

Computing and Software

McMaster University

Faculty of Engineering

Coconut

Design Patterns for Signal Processing

"in functional languages Design Patterns are executable"

- less programmer intervention = fewer errors
- type checking = almost as good as model checking
- provably correct implementations
- inheritance replaced by Haskell type classes

Christopher Anand
Wolfram Kahl

Computing and Software

McMaster University

Faculty of Engineering

Coconut

Haskell

- functional: functions as first-class citizens
- pure: no side-effects
- non-strict: unneeded arguments may be undefined
  - implemented usually via lazy evaluation
  - enables separation of concerns, e.g., generate/prune
- strongly typed
- safe overloading (type classes)
- accessible for correctness proofs and program derivation
- compiled mostly by transformation
- large, active community; open definition and compilers

Haskell

static typing
no side-effects } $\Rightarrow$ verifiably correct code by construction

Christopher Anand
Wolfram Kahl

Computing and Software

McMaster University

Faculty of Engineering

Coconut

Domain-Specific Languages (DSLs) in Haskell

- Embedded DSLs
  - built from primitives and combinators defined in Haskell
  - compiled by Haskell compiler
- Wrapping DSLs
  - wrap other libraries via the foreign-function interface (FFI)
  - from immediate wrappers to high-level abstractions
- Arbitrary DSLs
  - represented by Haskell data structures
  - the compiler will be a special Haskell program
  - Haskell is an excellent language for writing compilers

Coconut

Domain Specific Language

inside Haskell ⇒ open, extensible

Christopher Anand
Wolfram Kahl

Computing and
Software

McMaster
University

Faculty of
Engineering

Coconut

High Level DSL

mathematical specification of algorithms

Low Level DSL

high-performance, parallel vector, target-specific implementation

Christopher Anand
Wolfram Kahl

Computing and Software

McMaster University

Faculty of Engineering

Coconut

High level DSL

- containers (matrices, structures)
- precision
- approximation
- derivatives
- special functions
- polynomials
- linear programming
- Newton's method

80% grammar
Oct 2003

Coconut

Low level DSL

- includes High Level DSL
- cache hierarchy
- cache hinting
- instruction scheduling
- storage classes
- page sizes
- inter-processor messaging
- SIMD

Christopher Anand
Wolfram Kahl

Computing and Software

McMaster University

Faculty of Engineering

Coconut

Example: Interior Point Method for LP

key step: $A^{*}\,\mathrm{diag}(x)\,\mathrm{diag}(s)^{-1}\,A$

sparsity $\notin$ BLAS

BLAS	custom
$n \times m \times m \times m \times n$	$n \times m \times n$
	cache s^{-1}

Christopher Anand
Wolfram Kahl

Computing and Software

McMaster University

Faculty of Engineering

Coconut

blr

Christopher Anand
Wolfram Kahl

Computing and Software

McMaster University

Faculty of Engineering