Formulas as Functional Fortran-Style Map-Based Programs

Franz Franchetti

January 6, 2021

1 Transforms and Breakdown Rules

Definition 1.1 (WHT definition)**.**

$$\mathrm{WHT}_{2^k} = \underbrace{\mathrm{WHT}_2 \otimes \ldots \otimes \mathrm{WHT}_2}_{k \text{ times}} \tag{1}$$

$$\mathrm{WHT}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{2}$$

Identity 1.1 (WHT breakdown rule)**.**

$$\mathrm{WHT}_{2^{m+n}} = (\mathrm{WHT}_{2^m} \otimes \mathrm{I}_{2^n})(\mathrm{I}_{2^m} \otimes \mathrm{WHT}_{2^n}) \tag{3}$$

Definition 1.2 (DFT definition)**.**

$$\mathrm{DFT}_n = [\omega_n^{k\ell}]_{0 \le k,\ell < n}, \quad \omega_n = \sqrt[n]{1} \tag{4}$$

Identity 1.2 (DFT breakdown rule)**.**

$$\mathrm{DFT}_{mn} = (\mathrm{DFT}_m \otimes \mathrm{I}_n)\, \mathrm{T}^{mn}_n (\mathrm{I}_m \otimes \mathrm{DFT}_n)\, \mathrm{L}^{mn}_m \tag{5}$$

$$\mathrm{T}^{mn}_n = \mathrm{diag}_{0 \le i < mn} \left(\omega_n^{\lfloor \frac{i}{n} \rfloor + (i \bmod n)} \right) \tag{6}$$

Definition 1.3 (MDDFT definition)**.**

$$\mathrm{DFT}_{m \times n} = \mathrm{DFT}_m \otimes \mathrm{DFT}_n \tag{7}$$

Identity 1.3 (MDDFT breakdown rule)**.**

$$\mathrm{DFT}_{m \times n} = (\mathrm{DFT}_m \otimes \mathrm{I}_n)(\mathrm{I}_m \otimes \mathrm{DFT}_n) \tag{8}$$

2 SPL and Σ-SPL

Here we derive the PLDI paper's idea [1] specialized to WHT and Cooley-Tukey FFT, restricted to what we need for developing the a simple Fortran-style interpretation. First we define the symbols with the same appearance as in [1] (but less mathematic obfuscation) as in the paper, and then introduce Yevgen's index-free "GT+xchains" as developed in his thesis [5] and the CGO paper [4]. His representation allows to do the PLDI analysis without symbolic computation (just using vectors of integers).

2.1 Simplified Σ-SPL

These definitions are equivalent to a subset of the PLDI paper, but not the same.

2.1.1 Basic Definitions

Definition 2.1 (Row Basis Vector)**.**

$$\mathrm{e}_i^{1\times N} = [0,\ldots,0,1,0,\ldots,0] \in \mathbb{R}^{1\times N} \tag{9}$$

Definition 2.2 (Column Basis Vector)**.**

$$\mathrm{e}_i^{N\times 1} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^{N\times 1} \tag{10}$$

2.1.2 Gather Matrices

Definition 2.3 (Basic Gather Matrices)**.**

$$\mathrm{G}_{\imath_n} = \mathrm{I}_n \tag{11}$$
$$\mathrm{G}_{(j)_m} = \mathrm{e}_j^{1\times m} \tag{12}$$

Identity 2.1 (Tensor Product of Gather Matrices)**.** *For $f,g,h \in \{\imath_n,(j)_m\}$ we define*

$$\mathrm{G}_{f\otimes g} = \mathrm{G}_f \otimes \mathrm{G}_g \tag{13}$$
$$\mathrm{G}_{f\otimes g\otimes h} = \mathrm{G}_f \otimes \mathrm{G}_g \otimes \mathrm{G}_h \tag{14}$$
$$\ldots \tag{15}$$
$$\mathrm{G}_{\imath_n\otimes(j)_m} = \mathrm{G}_{\imath_n} \otimes \mathrm{G}_{(j)_m}$$
$$\mathrm{G}_{(j)_m\otimes\imath_n} = \mathrm{G}_{(j)_m} \otimes \mathrm{G}_{\imath_n} \tag{16}$$
$$\ldots$$

2.1.3 Scatter Matrices

Definition 2.4 (Basic Scatter Matrices)**.**

$$\mathrm{S}_{\imath_n} = \mathrm{I}_n \tag{17}$$
$$\mathrm{S}_{(j)_m} = \mathrm{e}_j^{m\times 1} \tag{18}$$

Identity 2.2 (Tensor Product of Scatter Matrices)**.** *For $f,g,h \in \{\imath_n,(j)_m\}$ we define*

$$\mathrm{S}_{f\otimes g} = \mathrm{S}_f \otimes \mathrm{S}_g \tag{19}$$
$$\mathrm{S}_{f\otimes g\otimes h} = \mathrm{S}_f \otimes \mathrm{S}_g \otimes \mathrm{S}_h \tag{20}$$
$$\ldots \tag{21}$$
$$\mathrm{S}_{\imath_n\otimes(j)_m} = \mathrm{S}_{\imath_n} \otimes \mathrm{S}_{(j)_m}$$
$$\mathrm{S}_{(j)_m\otimes\imath_n} = \mathrm{S}_{(j)_m} \otimes \mathrm{S}_{\imath_n} \tag{22}$$
$$\ldots$$

2.2 Translating SPL Formulas Into Σ-SPL Formulas

Identity 2.3 (Basic Translation)**.**

$$\mathrm{I}_m \otimes A_n = \sum_{j=0}^{m-1} \mathrm{S}_{(j)_m \otimes \imath_n}\, A_n\, \mathrm{G}_{(j)_m \otimes \imath_n} \tag{23}$$

$$A_m \otimes \mathrm{I}_n = \sum_{j=0}^{n-1} \mathrm{S}_{\imath_m \otimes (j)_n}\, A_m\, \mathrm{G}_{\imath_m \otimes (j)_n} \tag{24}$$

Identity 2.4 (Some Optimization Identities)**.**

$$\left(\sum_{j=0}^{m-1} A_j\right) B = \sum_{j=0}^{m-1} (A_j B) \tag{25}$$

$$B \left(\sum_{j=0}^{m-1} A_j\right) = \sum_{j=0}^{m-1} (B A_j) \tag{26}$$

$$\mathrm{G}_{(j)_m \otimes \imath_n}\, \mathrm{L}^{mn}_m = \mathrm{G}_{\imath_n \otimes (j)_m} \tag{27}$$

$$\mathrm{G}_{\imath_m \otimes (j)_n}\, \mathrm{T}^{mn}_n = (\mathrm{G}_{\imath_m \otimes (j)_n}\, \mathrm{T}^{mn}_n)\, \mathrm{G}_{\imath_m \otimes (j)_n} \tag{28}$$

$$\mathrm{G}_{(i)_2 \otimes \imath_2}\, \mathrm{G}_{(j)_2 \otimes \imath_4} = \mathrm{G}_{(j)_2 \otimes (i)_2 \otimes \imath_2} \tag{29}$$

$$\mathrm{S}_{(j)_2 \otimes \imath_4}\, \mathrm{S}_{\imath_2 \otimes (i)_2} = \mathrm{S}_{(j)_2 \otimes \imath_2 \otimes (i)_2} \tag{30}$$

3 Simple Examples

3.1 Example: $y = \mathrm{WHT}_8\, x$

3.1.1 SPL Formula: $y = \mathrm{WHT}_8\, x$

$$\mathrm{WHT}_8 = (\mathrm{WHT}_2 \otimes \mathrm{I}_4)(\mathrm{I}_2 \otimes \mathrm{WHT}_4) = (\mathrm{WHT}_2 \otimes \mathrm{I}_4)\big(\mathrm{I}_2 \otimes (\mathrm{WHT}_2 \otimes \mathrm{I}_2)(\mathrm{I}_2 \otimes \mathrm{WHT}_2)\big) \tag{31}$$

3.1.2 Fortran-Style Code for SPL Formula: $y = \mathrm{WHT}_8\, x$

```
! FORTRAN-STYLE example code

real(1:2) function WHT2(real x(1:2))
   WHT2(1) = x(1) + x(2)
   WHT2(2) = x(1) - x(2)
end


real(1:4) function WHT4(real x(1:4))
   real t{1:2}(1:2)

   ! t = (I2 x WHT2)x
   ! map WHT2 to real(1:2) arrays, loop over {1:2} tiles
   t = map(WHT2, reshape(x, (1:4)->{1:2}(1:2)){:}(1:2))

   ! map WHT2 to the {1:2} tiles, loop over real(1:2) arrays
   WHT4 = reshape(map(WHT2, t{1:2}(:)), {1:2}(1:2)->(1:4))
end


real(1:8) function WHT8(real x(1:8))
   real t{1:2}(1:4)

   ! t = (I2 x WHT4)x
   ! map WHT4 to real(1:4) arrays, loop over {1:2} tiles
   t = map(WHT4, reshape(x, (1:8)->{1:2}(1:4)){:}(1:4))
```

```
   ! return (WHT2 x I4)t
   ! map WHT2 to the {1:2} tiles, loop over real(1:4) arrays
   WHT8 = reshape(map(WHT2, t{1:2}(:)), {1:2}(1:4)->(1:8))
end


function main()
   real x(1:8)
   real y(1:8)

   y = WHT8(x)
end
```

3.1.3 SPL to Σ-SPL: $y = \mathrm{WHT}_8\, x$

$$\mathrm{WHT}_8 = (\mathrm{WHT}_2 \otimes \mathrm{I}_4)\big(\mathrm{I}_2 \otimes (\mathrm{WHT}_2 \otimes \mathrm{I}_2)(\mathrm{I}_2 \otimes \mathrm{WHT}_2)\big) \tag{32}$$

$$= \left(\sum_{j=0}^{3} \mathrm{S}_{\imath_2 \otimes (j)_4}\, \mathrm{WHT}_2\, \mathrm{G}_{\imath_2 \otimes (j)_4}\right) \left(\sum_{j=0}^{1} \mathrm{S}_{(j)_2 \otimes \imath_4}\, \mathrm{WHT}_4\, \mathrm{G}_{(j)_2 \otimes \imath_4}\right) \tag{33}$$

$$= \left(\sum_{j=0}^{3} \mathrm{S}_{\imath_2 \otimes (j)_4}\, \mathrm{WHT}_2\, \mathrm{G}_{\imath_2 \otimes (j)_4}\right) \left(\sum_{j=0}^{1} \mathrm{S}_{(j)_2 \otimes \imath_4} \left(\sum_{i=0}^{1} \mathrm{S}_{\imath_2 \otimes (i)_2}\, \mathrm{WHT}_2\, \mathrm{G}_{\imath_2 \otimes (i)_2}\right) \left(\sum_{i=0}^{1} \mathrm{S}_{(i)_2 \otimes \imath_2}\, \mathrm{WHT}_2\, \mathrm{G}_{(i)_2 \otimes \imath_2}\right) \mathrm{G}_{(j)_2 \otimes \imath_4}\right) \tag{34}$$

$$= \left(\sum_{j=0}^{3} \mathrm{S}_{\imath_2 \otimes (j)_4}\, \mathrm{WHT}_2\, \mathrm{G}_{\imath_2 \otimes (j)_4}\right) \left(\sum_{j=0}^{1} \left(\sum_{i=0}^{1} \mathrm{S}_{(j)_2 \otimes \imath_2 \otimes (i)_2}\, \mathrm{WHT}_2\, \mathrm{G}_{\imath_2 \otimes (i)_2}\right) \left(\sum_{i=0}^{1} \mathrm{S}_{(i)_2 \otimes \imath_2}\, \mathrm{WHT}_2\, \mathrm{G}_{(j)_2 \otimes (i)_2 \otimes \imath_2}\right)\right) \tag{35}$$

3.1.4 Fortran-Style Program for Σ-SPL Formula: $y = \mathrm{WHT}_8\, x$

```
! FORTRAN-STYLE iterator program

real(1:8) WHT8(real x(1:8))
   int i, j
   real t1(1:2), t2(1:4), t3(1:2), t4(1:8), t5(1:2)
   real u(1:2), v(1:2), w(1:2)

   forall j in [0..1] do
      forall i in [0..1] do
         ! gather data
         t1 = x(4*j+2*i+1:1:4*j+2*i+2)
         ! butterfly
         u(1) = t1(1) + t1(2)
         u(2) = t1(1) - t1(2)
         ! scatter data
         t2(2*i+1:1:2*i+2) = u
      end do
      forall i in [0..1] do
         ! gather data
         t3 = t2(i+1:2:i+3)
         ! butterfly
         v(1) = t3(1) + t3(2)
         v(2) = t3(1) - t3(2)
         ! scatter data
         t4(4*j+i+1:2:4*j+i+3) = v
      end do
   end do
   forall j in [0..3] do
      ! gather data
      t5 = t4(j+1:4:j+5)
      ! butterfly
      w(1) = t5(1) + t5(2)
      w(2) = t5(1) - t5(2)
      ! scatter data
```

```
      WHT8(j+1:4:j+5) = w
   end do
end


function main()
   real x(1:8)
   real y(1:8)

   y = WHT8(x)
end
```

3.2 Example: $y = \mathrm{DFT}_8\, x$

3.2.1 SPL Formula: $y = \mathrm{DFT}_8\, x$

$$\mathrm{DFT}_8 = (\mathrm{DFT}_2 \otimes \mathrm{I}_4)\, \mathrm{T}^8_4 (\mathrm{I}_2 \otimes \mathrm{DFT}_4)\, \mathrm{L}^8_2 = (\mathrm{DFT}_2 \otimes \mathrm{I}_4)\, \mathrm{T}^8_4 \left(\mathrm{I}_2 \otimes (\mathrm{DFT}_2 \otimes \mathrm{I}_2)\, \mathrm{T}^4_2 (\mathrm{I}_2 \otimes \mathrm{DFT}_2)\, \mathrm{L}^4_2 \right) \mathrm{L}^8_2 \quad (36)$$

3.2.2 FORTRAN-STYLE Code for SPL Formula: $y = \mathrm{DFT}_8\, x$

```
! FORTRAN-STYLE example code

complex(1:2) function DFT2(complex x(1:2))
   DFT2(1) = x(1) + x(2)
   DFT2(2) = x(1) - x(2)
end


complex(1:4) function DFT4(complex x(1:4))
   complex t{1:2}(1:2), u{1:2}(1:2), v{1:2}(1:2)
   complex T_4_2(4) = (/(1,0), (1,0), (1,0), (0,1)/)

   ! u = L^4_2 x
   u = transpose(reshape(x, (1:4)->{1:2}(1:2)))

   ! t = (I2 x DFT2)u
   ! map DFT2 to complex(1:2) arrays, loop over {1:2} tiles
   t = map(DFT2, u{:}(1:2))

   ! v = T^4_2 t
   v = T_4_2 *. t

   ! return (DFT2 x I2)v
   ! map DFT2 to the {1:2} tiles, loop over complex(1:2) arrays
   DFT4 = reshape(map(DFT2, v{1:2}), {1:2}(1:2)->(1:4))
end


complex(1:8) function DFT8(complex x(1:8))
   complex t{1:2}(1:4), u{1:2}(1:4), v{1:2}(1:4)
   real S22 = 0.70710678118654757
   complex T_8_4(8) = (/(1,0), (1,0), (1,0), (1,0), (1,0), (S22,S22), (0,1),(-S22,S22)/)

   ! u = L^8_2 x
   u = transpose(reshape(x, (1:8)->{1:4}(1:2)))

   ! t = (I2 x DFT4)x
   ! map DFT4 to complex(1:4) arrays, loop over {1:2} tiles
   t = map(DFT4, u{:}(1:4))

   ! v = T^8_4 t
   v = T_8_4 *. t

   ! return (DFT2 x I4)v
   ! map DFT2 to the {1:2} tiles, loop over complex(1:4) arrays
   DFT8 = reshape(map(DFT2, v{1:2}), {1:2}(1:4)->(1:8))
end
```

```
function main()
   complex x(1:8)
   complex y(1:8)

   y = DFT8(x)
end
```

3.2.3 SPL to Σ-SPL: $y = \mathrm{DFT}_8\, x$

$$\mathrm{DFT}_8 = (\mathrm{DFT}_2 \otimes \mathrm{I}_4)\, \mathrm{T}^8_4 \left(\mathrm{I}_2 \otimes (\mathrm{DFT}_2 \otimes \mathrm{I}_2)\, \mathrm{T}^4_2 (\mathrm{I}_2 \otimes \mathrm{DFT}_2)\, \mathrm{L}^4_2\right) \mathrm{L}^8_2 \tag{37}$$

$$= \left(\sum_{j=0}^{3} \mathrm{S}_{\imath_2 \otimes (j)_4}\, \mathrm{DFT}_2\, \mathrm{G}_{\imath_2 \otimes (j)_4}\right) \mathrm{T}^8_4 \left(\sum_{j=0}^{1} \mathrm{S}_{(j)_2 \otimes \imath_4}\, \mathrm{DFT}_4\, \mathrm{G}_{(j)_2 \otimes \imath_4}\right) \mathrm{L}^8_2 \tag{38}$$

$$= \left(\sum_{j=0}^{3} \mathrm{S}_{\imath_2 \otimes (j)_4}\, \mathrm{DFT}_2 \left(\mathrm{G}_{\imath_2 \otimes (j)_4}\, \mathrm{T}^8_4\right) \mathrm{G}_{\imath_2 \otimes (j)_4}\right) \left(\sum_{j=0}^{1} \mathrm{S}_{(j)_2 \otimes \imath_4}\, \mathrm{DFT}_4\, \mathrm{G}_{\imath_4 \otimes (j)_2}\right) \tag{39}$$

$$= \left(\sum_{j=0}^{3} \mathrm{S}_{\imath_2 \otimes (j)_4}\, \mathrm{DFT}_2 \left(\mathrm{G}_{\imath_2 \otimes (j)_4}\, \mathrm{T}^8_4\right) \mathrm{G}_{\imath_2 \otimes (j)_4}\right) \left(\sum_{j=0}^{1} \mathrm{S}_{(j)_2 \otimes \imath_4} (\mathrm{DFT}_2 \otimes \mathrm{I}_2)\, \mathrm{T}^4_2 (\mathrm{I}_2 \otimes \mathrm{DFT}_2)\, \mathrm{L}^4_2\, \mathrm{G}_{\imath_4 \otimes (j)_2}\right) \tag{40}$$

$$\begin{aligned} = \quad & \left(\sum_{j=0}^{3} \mathrm{S}_{\imath_2 \otimes (j)_4}\, \mathrm{DFT}_2 \left(\mathrm{G}_{\imath_2 \otimes (j)_4}\, \mathrm{T}^8_4\right) \mathrm{G}_{\imath_2 \otimes (j)_4}\right) \\ & \left(\sum_{j=0}^{1} \mathrm{S}_{(j)_2 \otimes \imath_4} \left(\textstyle\sum_{i=0}^{1} \mathrm{S}_{\imath_2 \otimes (i)_2}\, \mathrm{DFT}_2\, \mathrm{G}_{\imath_2 \otimes (i)_2}\right) \mathrm{T}^4_2 \left(\sum_{i=0}^{1} \mathrm{S}_{(i)_2 \otimes \imath_2}\, \mathrm{DFT}_2\, \mathrm{G}_{(i)_2 \otimes \imath_2}\right) \mathrm{L}^4_2\, \mathrm{G}_{\imath_4 \otimes (j)_2}\right) \end{aligned} \tag{41}$$

$$\begin{aligned} = \quad & \left(\sum_{j=0}^{3} \mathrm{S}_{\imath_2 \otimes (j)_4}\, \mathrm{DFT}_2 \left(\mathrm{G}_{\imath_2 \otimes (j)_4}\, \mathrm{T}^8_4\right) \mathrm{G}_{\imath_2 \otimes (j)_4}\right) \\ & \left(\sum_{j=0}^{1} \mathrm{S}_{(j)_2 \otimes \imath_4} \left(\textstyle\sum_{i=0}^{1} \mathrm{S}_{\imath_2 \otimes (i)_2}\, \mathrm{DFT}_2 \left(\mathrm{G}_{\imath_2 \otimes (i)_2}\, \mathrm{T}^4_2\right) \mathrm{G}_{\imath_2 \otimes (i)_2}\right) \left(\sum_{i=0}^{1} \mathrm{S}_{(i)_2 \otimes \imath_2}\, \mathrm{DFT}_2\, \mathrm{G}_{\imath_2 \otimes (i)_2}\right) \mathrm{G}_{\imath_4 \otimes (j)_2}\right) \end{aligned} \tag{42}$$

$$\begin{aligned} = \quad & \left(\sum_{j=0}^{3} \mathrm{S}_{\imath_2 \otimes (j)_4}\, \mathrm{DFT}_2 \left(\mathrm{G}_{\imath_2 \otimes (j)_4}\, \mathrm{T}^8_4\right) \mathrm{G}_{\imath_2 \otimes (j)_4}\right) \\ & \left(\sum_{j=0}^{1} \left(\textstyle\sum_{i=0}^{1} \mathrm{S}_{(j)_2 \otimes \imath_2 \otimes (i)_2}\, \mathrm{DFT}_2 \left(\mathrm{G}_{\imath_2 \otimes (i)_2}\, \mathrm{T}^4_2\right) \mathrm{G}_{\imath_2 \otimes (i)_2}\right) \left(\sum_{i=0}^{1} \mathrm{S}_{(i)_2 \otimes \imath_2}\, \mathrm{DFT}_2\, \mathrm{G}_{\imath_2 \otimes (i)_2 \otimes (j)_2}\right)\right) \end{aligned} \tag{43}$$

3.2.4 Fortran-Style Program for Σ-SPL Formula: $y = \mathrm{DFT}_8\, x$

```
! FORTRAN-STYLE iterator program

complex(1:8) DFT8(complex x(1:8))
   int i, j
   real S22 = 0.70710678118654757
   complex t1(1:2), t2(1:4), t3(1:2), t4(1:8), t5(1:2)
   complex r(1:2), s(1:2), u(1:2), v(1:2), w(1:2)
   complex T_4_2(4) = (/(1,0), (1,0), (1,0), (0,1)/)
   complex T_8_4(8) = (/(1,0), (1,0), (1,0), (1,0), (1,0), (S22,S22), (0,1),(-S22,S22)/)

   forall j in [0..1] do
      forall i in [0..1] do
         ! gather data
         t1 = x(j+2*i+1:4:j+2*i+5)
         ! butterfly
         u(1) = t1(1) + t1(2)
         u(2) = t1(1) - t1(2)
         ! scatter data
         t2(2*i+1:1:2*i+2) = u
      end do
      forall i in [0..1] do
         ! gather data
         t3 = t2(i+1:2:i+3)
         ! twiddle scaling
```

```
        r = t3 *. T_4_2(i+1:2:i+3)
        ! butterfly
        v(1) = r(1) + r(2)
        v(2) = r(1) - r(2)
        ! scatter data
        t4(4*j+i+1:2:4*j+i+3) = v
      end do
   end do
   forall j in [0..3] do
      ! gather data
      t5 = t4(j+1:4:j+5)
      ! twiddle scaling
      s = t5 *. T_8_4(j+1:4:j+5)
      ! butterfly
      w(1) = s(1) + s(2)
      w(2) = s(1) - s(2)
      ! scatter data
      DFT8(j+1:4:j+5) = w
   end do
end


function main()
   complex x(1:8)
   complex y(1:8)

   y = DFT8(x)
end
```

4 Index-Free Representation

In this section we will show how to perform the above symbolical manipulations without symbolic computation. All these manipulations can be done with rewriting of integer lists. However, to understand the integer-only representation it is necessary to first understand the symbolic representation developed in the sections above. The names below have (non-obvious) historical reasons.

4.1 XChains

XChains are a numerical representation of gather and scatter functions. A tensor product of $\imath_n$'s and $(j)_m$' s is encoded as a list of integers. The range of loop variables is defined in the GT objects defined below. A XChain is a list of integers. "0" refers to $\imath_n$. Any integer larger than 0 refers to a loop variable; i refers to the ith outer loop; loops are counted inside-out. For instance

$$[2, 0, 1] = (j)_m \otimes \imath_n \otimes (i)_k$$

with a sum structure

$$\sum_{j=0}^{m-1} \sum_{i=0}^{k-1} \mathrm{S}_{(j)_m \otimes \imath_n \otimes (i)_k} A_n \, \mathrm{G}_{(j)_m \otimes \imath_n \otimes (i)_k} .$$

4.2 GT

GT is a representation for iterative sums. Since every sum derived from a tensor product has a scatter, a gather and a kernel, we use GT to make these components explicit.

$$\mathrm{GT}(4, \mathrm{DFT}_2, [0, 1], [1, 0]) = \sum_{j=0}^{3} \mathrm{S}_{\imath_2 \otimes (j)_4} \, \mathrm{DFT}_2 \, \mathrm{G}_{(j)_4 \otimes \imath_2}$$

Above, the first parameter (4) defines the range of the loop variable. The second parameter defines the kernel. The third parameter defines the scatter operation, using XChains. The forth parameter defines the gather operation using XChains.

The above definition by example show that the representation is index-free and only requires integers. All rewriting rules used in the symbolic derivation of the optimized Σ-SPL expression can be translated into respective GT+XChains rules.

4.3 WHT Example using GT+XChains

$$\mathrm{WHT}_8 = \left(\sum_{j=0}^{3} \mathrm{S}_{\imath_2 \otimes (j)_4}\, \mathrm{WHT}_2\, \mathrm{G}_{\imath_2 \otimes (j)_4}\right) \left(\sum_{j=0}^{1} \left(\sum_{i=0}^{1} \mathrm{S}_{(j)_2 \otimes \imath_2 \otimes (i)_2}\, \mathrm{WHT}_2\, \mathrm{G}_{\imath_2 \otimes (i)_2}\right) \left(\sum_{i=0}^{1} \mathrm{S}_{(i)_2 \otimes \imath_2}\, \mathrm{WHT}_2\, \mathrm{G}_{(j)_2 \otimes (i)_2 \otimes \imath_2}\right)\right) \tag{44}$$

$$= \mathrm{GT}\left(4, \mathrm{WHT}_2, [0,1], [0,1]\right) \mathrm{GT}\left(2, \mathrm{GT}\left(2, \mathrm{WHT}_2, [2,0,1], [0,1]\right) \mathrm{GT}\left(2, \mathrm{WHT}_2, [1,0], [2,1,0]\right), [], []\right) \tag{45}$$

4.4 DFT Example using GT+XChains

The twiddle factors in the DFT introduce some more complexity. We use

$$\mathrm{T}^8_4[0,1]$$

to denote the subset of twiddles to be used; $[0,1]$ is the XChain used for the access. In Spiral we use a constant reorganization step to normalize the access pattern into constant arrays.

$$\begin{aligned} \mathrm{DFT}_8 = \quad & \left(\sum_{j=0}^{3} \mathrm{S}_{\imath_2 \otimes (j)_4}\, \mathrm{DFT}_2\, \left(\mathrm{G}_{\imath_2 \otimes (j)_4}\, \mathrm{T}^8_4\right) \mathrm{G}_{\imath_2 \otimes (j)_4}\right) \\ & \left(\sum_{j=0}^{1} \left(\textstyle\sum_{i=0}^{1} \mathrm{S}_{(j)_2 \otimes \imath_2 \otimes (i)_2}\, \mathrm{DFT}_2\, \left(\mathrm{G}_{\imath_2 \otimes (i)_2}\, \mathrm{T}^4_2\right) \mathrm{G}_{\imath_2 \otimes (i)_2}\right) \left(\sum_{i=0}^{1} \mathrm{S}_{(i)_2 \otimes \imath_2}\, \mathrm{DFT}_2\, \mathrm{G}_{\imath_2 \otimes (i)_2 \otimes (j)_2}\right)\right) \end{aligned} \tag{46}$$

$$= \mathrm{GT}\left(4, \mathrm{DFT}_2\, \mathrm{T}^8_4[0,1], [0,1], [0,1]\right) \mathrm{GT}\left(2, \mathrm{GT}\left(2, \mathrm{DFT}_2\, \mathrm{T}^4_2[0,1], [2,0,1], [0,1]\right) \mathrm{GT}\left(2, \mathrm{DFT}_2, [1,0], [0,1,2]\right), [], []\right) \tag{47}$$

5 SIMD Vectorization

We extend the Fortran syntax to denote SIMD vectors by using `<1:4>` to denote 4-way vectors. For instance, a tiled array of SIMD vectors is written as

```
real a{1:4}[1:2]<1:4>
```

In addition, we define a higher-order function `vec(., <1:4>)` that translates scalar programs into vector programs by replacing all scalar operations by their respective vector operations. For instance, the expression `vec(WHT2,<1:4>)` with

```
real(1:2) function WHT2(real x(1:2))
   WHT2(1) = x(1) + x(2)
   WHT2(2) = x(1) - x(2)
end
```

yields a function that is equivalent to

```
real(1:2)<1:4> function vec(WHT2, <1:4>)(real x(1:2)<1:4>)
   WHT2(1) = x(1)<1:4> + x(2)<1:4>
   WHT2(2) = x(1)<1:4> - x(2)<1:4>
end
```

Using SSE3, we can pack 4 real or 2 complex numbers into a 128-bit 4-way float SSE register. We use the short vector Cooley-Tukey FFT [2, 3] and the vector WHT algorithm to ensure that all memory accesses are aligned vector loads or stores and all arithmetic is vectorized.

We use Intel C++ intrinsics within Fortran comments to show the actual implementation of more complicated vector operations.

5.1 SIMD Vectorization: $y = \mathrm{WHT}_{16}\, x$, 4-way SSE

Here we show a SSE 4-way vectorized WHT. we pack 4 real numbers into one 4-way float vector register.

5.1.1 SPL Vector Formula

$$\mathrm{WHT}_{16} = \overrightarrow{\mathrm{L}}^{16}_{4}\Big(((\mathrm{WHT}_2 \otimes \mathrm{I}_2)(\mathrm{I}_2 \otimes \mathrm{WHT}_2))\vec{\otimes}\,\mathrm{I}_4\Big)\overrightarrow{\mathrm{L}}^{16}_{4}\Big(((\mathrm{WHT}_2 \otimes \mathrm{I}_2)(\mathrm{I}_2 \otimes \mathrm{WHT}_2))\vec{\otimes}\,\mathrm{I}_4\Big) \tag{48}$$

5.1.2 FORTRAN-STYLE Program for SPL Vector Formula

```
! FORTRAN-STYLE example code

! matrix transposition kernel
real(1:4)<1:4> function L_16_4(real x(1:4)<1:4>)
   ! __m128 x[4], y[4];
   ! y[0] = x[0];
   ! y[1] = x[1];
   ! y[2] = x[2];
   ! y[3] = x[3];
   ! _MM_TRANSPOSE4_PS(y[0], y[1], y[2], y[3]);
   L_16_4(1:4)<1:4> = reshape(transpose(reshape(x, (1:4)<1:4>->{1:4}(1:4))), {1:4}(1:4)->(1:4)<1:4>)
end


real(1:2) function WHT2(real x(1:2))
   WHT2(1) = x(1) + x(2)
   WHT2(2) = x(1) - x(2)
end


real(1:4) function WHT4(real x(1:4))
   real t{1:2}(1:2)

   ! t = (I2 x WHT2)x
   ! map WHT2 to real(1:2) arrays, loop over {1:2} tiles
   t = map(WHT2, reshape(x, (1:4)->{1:2}(1:2)){:}(1:2))

   ! map WHT2 to the {1:2} tiles, loop over real(1:2) arrays
   WHT4 = reshape(map(WHT2, t{1:2}), {1:2}(1:2)->(1:4))
end


real(1:16) function WHT16(real x(1:16))
   real t(1:4)<1:4>, u(1:4)<1:4>, v(1:4)<1:4>

   ! t = (WHT4 <x> I4)x
   t = vec(WHT4, <1:4>)(reshape(x, (1:16)->(1:4)<1:4>))

   ! u = (L^16_4)t
   ! in-register transpose via shuffle
   u = L_16_4(t)

   ! v = (WHT4 <x> I4)u
   v = vec(WHT4, <1:4>)(reshape(u, (1:16)->(1:4)<1:4>))

   ! return (L^16_4)v
   ! in-register transpose via shuffle
   WHT16 = reshape(L_16_4(v), (1:4)<1:4>->(1:16))
end


function main()
   real x(1:16)
   real y(1:16)

   y = WHT16(x)
end
```

5.2 SIMD Vectorization: $y = \mathrm{DFT}_8\, x$, 4-way SSE3

Here we show a SSE3 4-way vectorized DFT. we pack 2 complex numbers into one 4-way float vector register.

5.2.1 SPL Vector Formula

$$\mathrm{DFT}_8 = \left((\mathrm{DFT}_2 \otimes \mathrm{I}_2)\vec{\otimes}\,\mathrm{I}_2\right)\overrightarrow{\mathrm{T}}^8_4\left(\mathrm{L}^4_2\,\vec{\otimes}\,\mathrm{I}_2\right)\left(\mathrm{I}_2 \otimes \overrightarrow{\mathrm{L}}^4_2\right)\left(\left((\mathrm{DFT}_2 \otimes \mathrm{I}_2)\,\mathrm{T}^4_2(\mathrm{I}_2 \otimes \mathrm{DFT}_2)\,\mathrm{L}^4_2\right)\vec{\otimes}\,\mathrm{I}_2\right) \tag{49}$$

5.2.2 FORTRAN-STYLE Program for SPL Vector Formula

```
! FORTRAN-STYLE example code

! matrix transposition kernel
complex(1:2)<1:2> function L_4_2(complex x(1:2)<1:2>)
   ! __m128 x[2], y[2];
   ! y[0] = _mm_shuffle_ps(x[0], x[1], _MM_SHUFFLE(1,0,1,0);
   ! y[0] = _mm_shuffle_ps(x[0], x[1], _MM_SHUFFLE(3,2,3,2);
   L_4_2(1:2)<1:2> = reshape(transpose(reshape(x, (1:2)<1:2>->{1:2}(1:2))), {1:2}(1:2)->(1:2)<1:2>)
end


! complex vector multiplication kernel
! operator overloading for <1:2> * <1:2> -> <1:2>
complex<1:2> operator(*)(complex x<1:2>, complex t<1:2>)
   ! __m128 x, y, t;
   ! y = _mm_addsub_ps(
   !    _mm_mul_ps(x, _mm_moveldup(t))),
   !    _mm_mul_ps(
   !       _mm_shuffle_ps(x, _MM_SHUFFLE(2,3,0,1)),
   !       _mm_movehdup(t))));
   operator(*) = reshape(reshape(x, <1:2>->(1:2)) *. reshape(t, <1:2>->(1:2)), (1:2)-><1:2>)
end


complex(1:2) function DFT2(complex x(1:2))
   DFT2(1) = x(1) + x(2)
   DFT2(2) = x(1) - x(2)
end


complex(1:4) function DFT4(complex x(1:4))
   complex t{1:2}(1:2), u{1:2}(1:2), v{1:2}(1:2)
   complex T_4_2(4) = (/(1,0), (1,0), (1,0), (0,1)/)

   ! u = L^4_2 x
   u = transpose(reshape(x, (1:4)->{1:2}(1:2)))

   ! t = (I2 x DFT2)u
   ! map DFT2 to complex(1:2) arrays, loop over {1:2} tiles
   t = map(DFT2, u{:}(1:2))

   ! v = T^4_2 t
   v = T_4_2 *. t

   ! return (DFT2 x I2)v
   ! map DFT2 to the {1:2} tiles, loop over complex(1:2) arrays
   DFT4 = reshape(map(DFT2, v{1:2}), {1:2}(1:2)->(1:4))
end


complex(1:8) function DFT8(complex x(1:8))
   complex t(1:4)<1:2>, u(1:4)<1:2>, v(1:4)<1:2>, w(1:4)<1:2>
   real S22 = 0.70710678118654757
   complex T_8_4(1:4)<1:2> = (/(1,0), (1,0), (1,0), (1,0), (1,0), (S22,S22), (0,1),(-S22,S22)/)

   ! t = (DFT4 <x> I2)x
   ! apply vectorized DFT4 to array(1:4) of SIMD vectors
```

```
   t = vec(DFT4, <1:2>)(reshape(x, (1:8)->(1:4)<1:2>))

   ! u = (I2 x L^4_2)t
   ! in-register transpose via shuffle
   u = reshape(map(L_4_2, reshape(t, (1:4)<1:2>->{1:2}(1:2)<1:2>){:}), {1:2}(1:2)<1:2>->(1:4)<1:2>)

   ! v = (L^4_2 <x> I2)u
   ! transpose of matrix of vectors
   v = reshape(vec(transpose, <1:2>)(reshape(u, (1:4)<1:2>->{1:2}(1:2)<1:2>)), {1:2}(1:2)<1:2>->(1:4)<1:2>)

   ! w = T^8_4 v
   ! uses complex<1:2> operator(*)
   w = v *. T_8_4

   ! return ((DFT2 x I2) <x> I2)v
   ! apply vectorized DFT2 to array{1:2} of SIMD vectors; loop over {1:2}
   DFT8 = reshape(map(vec(DFT2, <1:2>), reshape(w (1:4)<1:2>->{1:2}(1:2)<1:2>)(:)), {1:2}(1:2)<1:2>->(1:8))
end


function main()
   complex x(1:8)
   complex y(1:8)

   y = DFT8(x)
end
```

References

[1] F. Franchetti, Y. Voronenko, and M. Püschel. Loop merging for signal transforms. In *Proc. ACM PLDI*, pages 315–326, 2005.

[2] Franz Franchetti and Markus Püschel. Short vector code generation for the discrete Fourier transform. In *International Parallel and Distributed Processing Symposium (IPDPS)*, 2003.

[3] Franz Franchetti, Yevgen Voronenko, and Markus Püschel. A rewriting system for the vectorization of signal transforms. In *High Performance Computing for Computational Science (VECPAR)*, volume 4395 of *Lecture Notes in Computer Science*, pages 363–377. Springer, 2006.

[4] Y. Voronenko, F. de Mesmay, and M. Püschel. Computer generation of general size linear transform libraries. In *Proc. Code Generation and Optimization (CGO)*, 2009.

[5] Yevgen Voronenko. *Library Generation for Linear Transforms.* PhD thesis, Electrical and Computer Engineering, Carnegie Mellon University, 2008.