Streams Library Description

5 November 2013

Address:	Gedae, Inc. 1247 N. Church Street, Suite 5 Moorestown, NJ 08057
Telephone:	(856) 231-4458
FAX:	(856) 231-1403
Internet:	www.gedae.com

Streams Library Description

The document describes the kernels found in the Gedae streams library. It also lets the user know how to find various kernels in the streams library and provides a convention that users should use for creating their own libraries. The first section defines the naming convention and style of the kernels.

1 Kernel Name Format

The typical kernel name will be one of the following form:

<Token Type><Data Type>_[prefixes]<RootName>[<ParamType>]

<Token Type1>_<Token Type2> /* token type conversion operator */

<Token Type><Datat Type1>_<Token Type>_<Data Type2> /* casting operator */

Examples:

mf_rsum has the equation:

out[r] += in[r][c];

mf_ gives the input token type of matrix and the data type of float. The prefix r is used for matrix collapsing operators to indicate the collapsing is along the row dimension. The root name sum indicates that the += collapsing operator is used.

vxf_multVX has the equation:

out[n] = in[n]*VX[n];

vxf_ give the input token type, mult says the operation is an element wise multiplication and VX indicates that one of the operands is a complex parameter vector

sf_si has the equation:

int out = in;

the sf indicates the input type is a scalar float and the si indicates the output type is a scalar int.

m_v has the equation

out[c](r) = in[r][c];

2 Data Type Names and Token Type Names:

This following table lists the data type names.

Type	<Data Type>	Directory Name	Comment
float	f	float	
int	i	int	
char	c	char	
short	s	short	
unsigned int	ui	uint	
unsigned char	uc	uchar	
unsigned short	us	ushort	
double	d	double	
complex	xf	complex	Single precision complex
dcomplex	xd	dcomplex	Double precision complex
void	(no prefix)		

The next table lists the token type names.

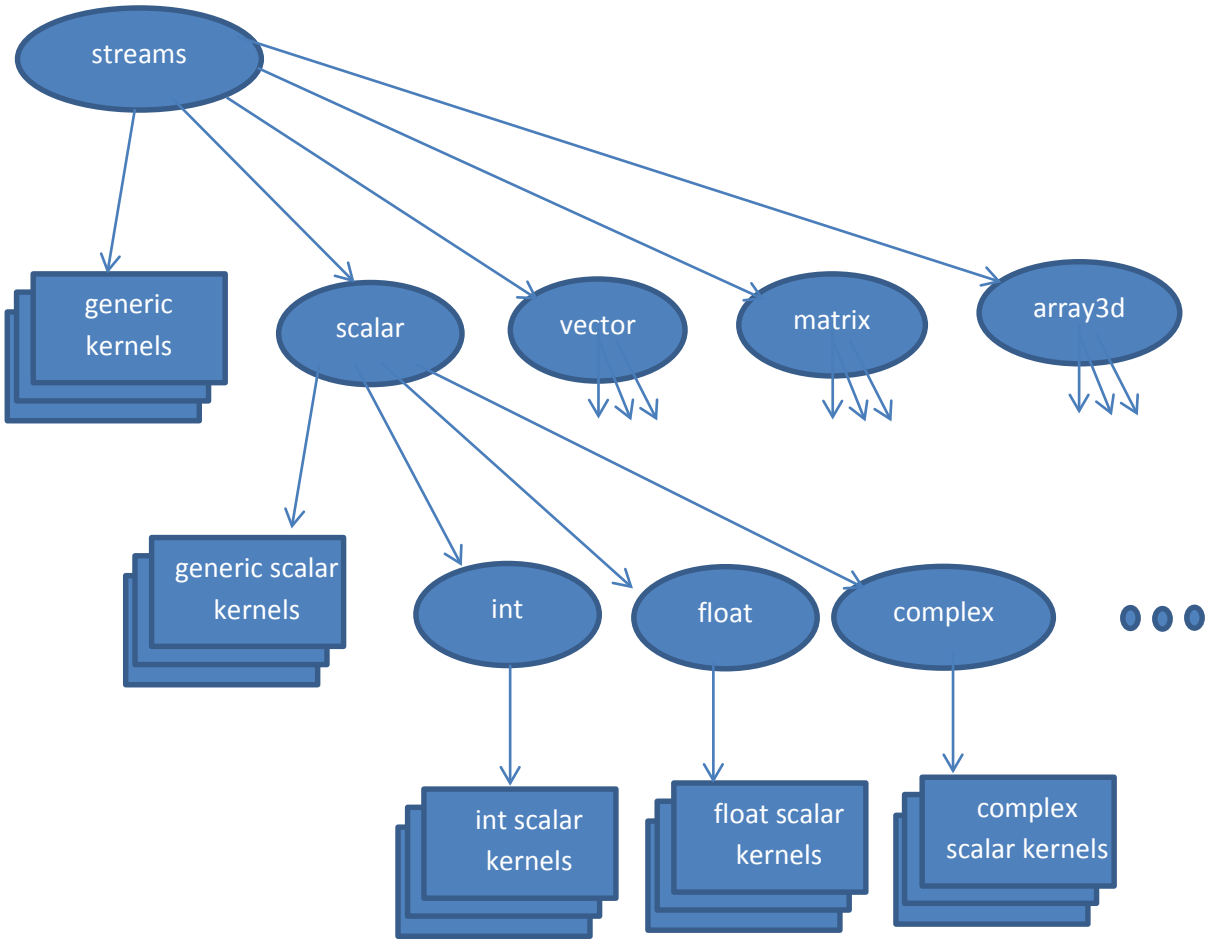
Object	<Token Type>	Directory Name
Scalar	s	scalar
Vector	v	vector
Matrix	m	matrix
3d array	a3	array3d

Examples: vxf_add.k is a single precision complex vector add. v_m is a void type vector to matrix conversion. Void type kernels can be connected to any data type. a3f_abs.k is an element-wise absolute value of a floating pointer 3d array. mi_mmult.k is an integer matrix multiply. sf_fir.k is a scalar float fir filter.

3 Streams kernel directory path

Kernels in the streams directory are put in directories based on their token type and data type. Kernels that input/output void tokens with no dimensions specified will take on the type and dimensionality of the data ports they are connected to. These are called generic kernels as they apply to any datatype. An example of such a kernel is the streams/copy.k that will copy the value of any input stream to the output. These kernels reside in the toplevel streams directory. If the highest dimensional token type of a kernel input/output is 0 dimensional the kernel reside in the streams/scalar directory. Vectors are in the streams/vector directory, matricies in the streams/matrix directory and 3d tokens in the streams/array3d directory. If the token type is specified but the data type is not (data types set to void) then the kernel appears in these directories. Otherwise the kernel appears in a subdirectory with the name of the data type. So for example a complex vector fft is found in streams/vector/complex/vxf_fft.k and a generic matrix family row part function is found in streams/matrix/m_rpart_fam.k.

The streams directory structure is illustrated in the following diagram. Only directories for types int, float and complex are shown but additional directory typenames include char, short, uchar (unsigned char), ushort, uint, double and dcomplex (double complex).



4 Port Names

Kernels with just one stream input and one stream output (or a family of either) will name the inputs in and the output out. Kernels that have two or more interchangeable inputs, such as the sf_add.k kernel, will name their inputs a, b, c, ... up to the number of inputs. A single float/double parameter input to a binary operator will be named K. Complex/dcomplex parameters will be named X. Vector parameter inputs of type float/double or complex/dcomplex will be named VK and VX respectively. And matrix parameter inputs of type float/double or complex/dcomplex will be named MK and MX respectively. When a binary operator parameter input is the first input of a kernel the parameter name is added as a prefix to the root name. For example the kernel that has scalar float inputs and the equation out = K/in has the name sf_Kdiv. If a parameter is the second input of a kernel the parameter name becomes a suffix to the root name. For example a kernel with float input in that produces a complex vector ouptut out is and implements the equation out[i] = in*VX[i] has the name sf_multVX.

A parameter that specifies a vector dimension output will have the name N or begin with N followed by lower case letters to further describe the parameter. Parameters that specify the row size of a matrix will be named R or begin with R and a parameter specifying the column size of a matrix will be named or begin with C. Three-d arrays will have dimension parameters names beginning with X, Y and Z respectively.

5 Kernels Descriptions

As previously stated, a Gedae kernel can be described by its token type, data type and root name and various prefixes and suffixes. The root name of the kernel describes the basic function the kernel performs. Examples of root names are add, fft, mmult, norm. This section list different catagories of functions and the root names associated with each. Th Idea expression, function or operator that the root name corresponds to is described as are the data types and token types to which it applies.

5.1 C operators

The table below lists the root names for all binary and unary C operators. The binary operators generally have both operands of the same type. The * operators can in addition have one operand of type float and one of type complex.

Operator	Type	Root Name	Data Types
+	Binary	add	All
-	Binary	sub	All
*	Binary	mult	All
/	Binary	div	All
/ with zero check	Binary	divz	All
1.0f /	Unary	recip	f,d,xf,xd
==	Binary	eq	All
!=	Binary	ne	All
<=	Binary	le	c,s,i,f,d
<	Binary	lt	c,s,i,f,d
>=	Binary	ge	c,s,i,f,d

>	Binary	gt	c,s,i,f,d
&&	Binary	and	c,s,i
||	Binary	or	c,s,i
&	Binary	bitand	c,s,i
|	Binary	bitor	c,s,i
!	Unary	not	c,s,i
~	Unary	bitnot	c,s,i
-	Unary	neg	All
? :	Tertiary	select	All
* +	Tertiary	multadd	f,d
* -	Tertiary	multsub	f,d
- *	Tertiary	submult	f,d

All binary operators have versions with a parameter as the second input (add K or X suffix to name). The sub and div operators have versions with a parameter as the first input (add K or X prefix to name). Operators mult and add can in addition take a parameter of a higher dimensionality. For example sf_addVK.k adds a scalar stream to a vector parameter.

Multiply-add and multiply-subtract kernels are also provided to link to E functions that best utilize multiply-add ALUs. Similar combinations of arrays and types are possible, however complex tokens are not used as the connection to ALU performance is indirect. Also not used are combinations where there are more adds than multiplies, such as the hypothetical kernel sf_sf_vf_multadd (out[i] = a*b + c[i]).

Examples:

A list of all the binary add kernels of type float for scalars, vectors and matrices are:

Kernel	Equation	Description
sf_add	out = a+b	scalar addition of streams
sf_addK	out = in+K	addition of scalar stream to parameter
sf_addVK	out[n] = in+VK[n]	addition of scalar stream to vector parameter
sf_addMK	out[r][c] = in+MK[r][c]	addition of scalar stream to matrix parameter
vf_add	out[n] = a[n]+b[n]	vector addition
vf_addK	out[n] = in[n]+K	addition of vector stream to scalar parameter
vf_addVK	out[n] = in[n]+VK[n]	addition of vector stream to vector parameter
vf_raddMK	out[r][c] = in[c]+MK[r][c]	add vector in to every row of MK
vf_caddMK	out[r][c] = in[r]+MK[r][c]	add vector in to every column of MK
vf_sf_add	out[n] = in[n]+k	add scalar k to every element of vector in
mf_add	out[r][c] = a[r][c]+b[r][c]	add matrix a to b
mf_addK	out[r][c] = in[r][c]+K	add scalar parameter K to matrix in
mf_raddVK	out[r][c] = in[r][c]+VK[c]	add vector parameter VK to every row of in
mf_caddVK	out[r][c] = in[r][c]+VK[r]	add vector parameter VK to every column of in
mf_addMK	out[r][c] = in[r][c]+MK[r][c]	add stream matrix in to parameter matrix MK
mf_sf_add	out[r][c] = in[r][c]+k	add matrix stream in to scalar stream k
mf_vf_radd	out[r][c] = a[r][c]+b[c]	add vector b to every row of a

mf_vf_cadd	out[r][c] = a[r][c]+b[r]	add vector b to every column of a

A list of all the binary multiply kernels of type float, for scalars, vectors and matrices are:

sf_mult, sf_multK, sf_multX, sf_multVK, sf_multVX, sf_multMK, sf_multMX, vf_mult, vf_multK, vf_multX, vf_multVK, vf_multVX, vf_rmultMK, vf_cmultMK, vf_rmultMX, vf_cmultMX, vf_sf_mult, vf_sxf_mult, mf_mult, mf_multK, mf_multX, mf_rmultVK, mf_multVX, mf_cmultVK, mf_multMK, mf_cmultVX, mf_rmultVX, mf_multMX, mf_sf_mult, mf_vf_rmult, mf_vf_cmult, mf_sxf_mult, mf_vxf_rmult, mf_vxf_cmult.

A list of all the multiply-add kernels of type float, for scalars, vectors and matrices are:

sf_multadd, sf_sf_K_multadd, sf_K_sf_multadd, sf_{MK,VK}_sf_multadd, sf_{MK,VK}_K_multadd, sf_VK_VK_multadd, sf_MK_MK_multadd

vf_multadd, vf_vf_sf_multadd, vf_vf_K_multadd, vf_vf_VK_multadd, vf_VK_vf_multadd, vf_VK_VK_multadd, vf_VK_sf_multadd, vf_VK_K_multadd,

mf_multadd, mf_mf_sf_multadd, mf_mf_K_multadd, mf_mf_MK_multadd, mf_sf_mf_multadd, mf_sf_MK_multadd, mf_sf_sf_multadd, mf_sf_K_multadd, mf_MK_mf_multadd, mf_MK_MK_multadd, mf_MK_sf_multadd, mf_MK_K_multadd, mf_K_mf_multadd, mf_K_MK_multadd, mf_K_sf_multadd, mf_K_K_multadd

vf_{sf,K}_{sf,K,vf,VK}_multadd, sf_VK_vf_multadd,

mf_{vf, VK}_{sf,K,vf,VK,mf,MK}_{r,c}multadd, vf_MK_{sf,K,vf,VK,mf,MK}_{r,c}multadd

5.2 Standard math library functions:

RootName	Datatypes
cos,sin,tan	f,d
cosh,sinh,tanh	f,d
acos,asin,atan,atan2	f,d
hypot	f,d
sqrt	f,d,xf,xd
exp	f,d,xf,xd
exp2, exp10	f,d
log,log10	f,d
log2	i,f,d
floor,ceil	f,d
round,trunc	f,d
abs	i,f,d,xf,xd
min,max*	i,f,d
minabs,maxabs	All
pow+	f,d
mod	i,f,d

conj	xf,xd
pol2rec	xf,xd
rec2pol	xd,xf
sqr	All
signsqr	i,f,d

* K suffix version also

+ K prefix and suffix versions also

Kernels for all the functions exist for all the token types (s, v, m, and a3)

5.3 Filtering Functions

The following filter functions are provided for scalar token types.

Root Name	Function	Data Types	Comment
acc	out=out(-1)+in;	f,d,xf,xd	Integration
macc	out=out(-1)+a*b;	f,d,xf,xd	Multiply Accumulate
lpf	out=A*in+(1-A)*out(-1); out=lpf(in,A);	f,d,xf,xd	Single Pole Low Pass Filter
fir	out+=C[i]*in(-i); out=fir(in,C);	f,d,xf,xd	FIR Filter
firD	out=firD(in,C,D);	f,d,xf,xd	Decimating FIR Filter

5.4 Special integer functions

The following functions that operate only on integers are useful for calculating parameters

bf(a,b)	biggest factor in a that is a power of b
gcd(a,b)	greatest common denominator of a and b
lcm(a,b)	least common multiple of a and b

For example functions si_bf, si_gcd and si_lcm are provided.

5.5 Source functions

The following functions provide generic stream sources. All inputs to these functions are parameters.

Root Name	Data Types	Comment
random(Seed)	i	random number between 0 and 0x7fffffff with initial seed of Seed
uniform(Seed)	f	uniform distribution between 0 and 1 with initial seed of Seed
normal(Seed)	f,d,xf,xd	normal distribution with mean 0 and stdv 1 and initial seed of Seed
poisson(P,Seed)	i	output is 1 with a probability of P and 0 with a probability of (1-P).
ramp(K)	f,d,xf,xd	ramp with initial value of 0 and step size of K
osc(F,A,P)	f,d,xf,xd	oscillator with radian frequency F, amplitude A and starting phase P
constant(K)	void type	stream with constant value K

5.6 Collapsing library functions

The collapsing opertators are supported with kernels having with root names given in the following table.

Operator	RootName
+=	sum
*=	product
>?=	max
<?=	min
||=	any
&&=	all
|=	bitany
&=	bitall

The Collapsing kernels can have the following prefixes on the root name:

Prefix	Operation
	Collapse all the dimensions
f	Collapse family index
N	Collapse time index
r	Collapse row
c	Collapse column
x	Collapse x
y	Collapse y
z	Collapse z
xy	Collapse xy
yz	Collapse yz
xz	Collapse xz

Example float sum box:

The collapsing sum operation the most important of the collapsing operators. All of the following kernels are provided in the library.

Kernel Name	Equation
scalar/float/sf_fsum.k	out+=[f]in
scalar/float/sf_Nsum.k	out+=in(n)
vector/float/vf_sum.k	out+=in[n]
vector/float/vf_fsum.k	out[n]+=[f]in[n]
vector/float/vf_Nsum.k	out[n]+=in[n](m)
matrix/float/mf_sum.k	out+=in[r][c]
matrix/float/mf_fsum.k	out[r][c]+=[f]in[r][c]
matrix/float/mf_csum.k	out[c]+=in[r][c]
matrix/float/mf_rsum.k	out[r]+=in[r][c]
matrix/float/mf_Nsum.k	out[r][c]+=in[r][c](n)
array3d/float/a3f_sum.k	out+=in[x][y][z]
array3d/float/a3f_fsum.k	out[x][y][z]+=[f]in[x][y][z]
array3d/float/a3f_xsum.k	out[x]+=in[x][y][z]
array3d/float/a3f_ysum.k	out[y]+=in[x][y][z]

array3d/float/a3f_zsum.k	out[z]+=in[x][y][z]
array3d/float/a3f_xysum.k	out[x][y]+=in[x][y][z]
array3d/float/a3f_yzsum.k	out[y][z]+=in[x][y][z]
array3d/float/a3f_xzsum.k	out[x][z]+=in[x][y][z]
array3d/float/a3f_Nsum.k	out[x][y][z]+=in[x][y][z](n)

5.7 Threshold operations

clip: out = a<b ? b : a > c ? c : a;

5.8 Data reorg operations

Data reorg operations do not change the values of individual word but instead reogranize them in time and space. For example, matrix transpose, partitioning, concatenation and gather operations are all data reorg operations.

5.8.1 get - extract elements from a token

The get operations extract scalars, subvector or submatrix from a token based on integer parameter inputs that control the offset and/or size of the output token. The input token can be either a stream or a parameter. A get function that has no suffix outputs a subtoken of the same dimension as the input token. Otherwise the get function outputs a token of the type of the suffix. When a nonscalar but lower dimension type of token is output from a get function a prefix indicating the direction in which the output token is taken from the input is given. If it gets a subtoken out of that dimension the root name is changed to getsub.

v_get.k has equation out[i] = in[i+N0]; Gets a subvector from input vector in beginning at offset Noff.

m_rget_v.k has equation out[c] = in[R0][c];

m_rgetsub_v.k has equation out[c] = in[R0][c+C0];

streams/vector/v_resize.k	out[i]=in[i]; range i = N;
streams/vector/v_resize_clr.k	out[i]=i<dimof(in)?in[i]:0; range i = N;
streams/vector/v_get.k	out[i]=in[i+N0]; range i = N;
streams/vector/v_get_s.k	out=in[N];
streams/matrix/m_get.k	out[r][c]=in[r+R0][c+C0]; range r=R; range c=C;
streams/matrix/m_get_s.k	out=in[R0][C0]
streams/matrix/m_rgetK_v.k	out[c]=in[R0][c];
streams/matrix/m_rgetsubK_v.k	out[c]=in[R0][c+C0]; range c=C;
streams/matrix/m_cgetK_v.fg	out[r]=in[r][C0];
streams/matrix/m_cgetsubK_v.fg	out[r]=in[r+R0][C0]; range r=R;

5.8.2 set - insert elements into a token

The set and setsub functions inserts elements into a token. All set functions take an input token that is the same size as the output token to which the output token is initially set. Then a part of the output token is set to a smaller token. For example given input streams in[n] and v[nj] and integer offset stream k, the v_set.k kernel implements the idea expression

out[n] = set(in,v,k);

Which is equivalent to:

out[n] = in[n];
out[nj+k] = v[nj];

All set functions begin by setting the output to the input and only differ in how they then adjust the output based on the other paramters. The following table assumes the initial setting of the input to the output and only describes how the set functions then modify the output further. Capital letter names are parameters. Currently not all possible varients of set functions have been implemented but since streams can be connected to parameters this is not a serious limitation. Missing set functions may be supplied on request.

Kernel Name	Idea Equation	Algebraic Expression
v_set.k	out[n] = set(in,v,k)	out[nj+k] = v[nj]
v_setK.k	out[n] = set(in,v,K)	out[nj+K] = v[nj]
v_si_setK.k	out[i] = set(in,K,si);	out[si] = K
v_VK_set.k	out[i] = set(in,VK,k)	out[nj+k] = Vk[nj]
v_s_set.k	out[i] = set(in,s,k);	out[k] = s
v_VK_setK.k	out[i] = set(in,VK,K)	out[nj+K] = Vk[nj]
v_s_setK.k	out[i] = set(in,s,K);	out[K] = s
m_setK.k	out[r][c] = set(in,m,R0,C0)	out[rj+R0][cj+C0] = m[rj][cj]
m_s_setK.k	out[r][c] = set(in,s,R0,C0)	out[R0][C0] = s
m_v_csetK.fg		out[r][C0] = v[r]
m_v_csetsubK.fg		out[rj+R0][C0] = v[rj]
m_v_rsetK.k	out[r][c] = rset(in,v,R0)	out[R0][c] = v[c]
m_v_rsetsubK.k	out[r][c] = rsetsub(in,v,R0,C0)	out[R0][cj+C0] = v[cj]
a3_setK.k	out[x][y][z] = set(in,a,X0,Y0,Z0)	out[xj+X0][yj+Y0][zj+Z0] = a[xj][yj][zj]

5.8.3 Time partitioning and concatenation

Partitioning tokens in time allows large tokens to be broken into smaller tokens that can entirely fit in cache. Typically the large token will be partitioned – many operations will be done on the token – and

then it will be concatenated back into a full sized token. Because the operations are done on the smaller partitioned token operations can be strung together and done without a cache miss. The functions that are provided are:

part_strm - partition a token to a stream of tokens covering the input token
concat_strm – concatenate a stream of tokens into a single token – inverse of part_strm
parteq_strm – partition a token into N equal sized tokens that may not cover the input
concateq_strm – concatenate N equal sized tokens into an output token – inverse of parteq_strm

These functions have void data input so appear in the stream/vector, stream/matrix or stream/array3d libraries.

For example the following vector and matrix kernels are provided:
v_part{,eq}_strm.k
m_{r,c}part{,eq}_strm.k
a3_{x,y,z}part{,eq}_strm.k

Partitioning of matrices can be across row dimension or column dimension, so the r and c prefixes are used to specify this information. Similarly, for 3-d arrays, x, y, z are used. Partitioning a matrix into tiles (partitioning in both the r and c direction) can be achieved by first doing an m_rpart_strm and following it with an m_cpart_strm.

Because the goal of the part_strm functions is to break the token into subtokens of a given size that fit in cache the parameters to the part_strm function specify the maximum dimension size. Tokens of the maximum size are partitioned out of the token with the last token partitioned out handling the remainder. For example if a v_part_strm kernel is applied to an input token of size N and specifies a maximum size of Nmax then the output will consist of floor(N/Nmax) tokens of size Nmax and – if Nmax does not divide evenly into N – one additional token of size N%Nmax. m_rpart_strm takes Rmax as a parameter and m_cpart_strm takes Cmax as a parameter. Similarly the a3_<x,y,z>_strm functions take Xmax,Ymax or Zmax as paramters.

The concat_strm concatenation function takes the max value passed to the partitioning function and the size of the input token to the partitioning function as parameters. So v_concat_strm will take Nmax and N as parameters. With this information the v_concat_strm function can calculate the number of tokens needed to rebuild the output vector of size N.

The parteq_strm kernels are a bit simpler. The input parameter N specifies how many equal sized tokens to break the input token into. If the input token is of size N1 then the N output tokens are of size floor(N1/N). Because the tokens have equal size they may not cover the input token if N does not divide evenly into N1. The concateq_strm kernels take the same parameter N and just concatenate N equally sized input tokens on the input stream into an output vector that is N times as big as the input vector.

5.8.4 Spatial partitioning and concatenation

Spatial partitioning boxes break an input token into a family of output tokens so the tokens can be processed in parallel. As such these kernels are some of the most important functions in the library.

Root Name	Description.
part_fam	partition to a family tokens that completely cover the input.
partovl_fam	partition into a family of overlapped tokens
parteq_fam	partition into a family of equal sized tokens possibly not covering the input
parteqovl_fam	partition into family of equal sized overlapping tokens
concat_fam	concatenate from a family of streams
concateq_fam	concatenate using equal sized tokens

These functions are provided for vectors, matricies and array3d objects. All take any data type inputs and can be found in the streams/vector, streams/matrix and streams/array3d libraries. Matrix parts and concats require the prefix r or c and array3d parts and concats require the prefix x, y or z. For example kernels m_rpart_fam.k, v_partovl_fam.k and a3_xparteq_fam.k are all boxes in the libraries.

5.8.5 Find

The find function finds all the indices of a token that are non-zero. Find functions are provided for void types and can be found in the streams/vector, streams/matrix of streams/array3d libraries. There are several variants of the find function depending described below. In the table we show the equation for a vector find. Each find function outputs an array of indices n[i]. Equivalent matrix and array3d finds are also provided. Matrix finds output indices r[i],c[i] and arra3d finds output indices x[i],y[i],z[i].

Root Name	Equation	Descritpion
find	n[i] = find(in);	find all the indices of the non-zero elements.
findval	n[i],value[i] = find(in);	Find the indices of the non-zero elements and the value of those elements. Equivalent to converting a vector, matrix or array3d element to its sparse representation.
findN	n[i] = find(in,N)	find the indices of the first N non-zero elements
findvalN	n[i],value[i] = find(in,N);	find the indices first N non-zero elements and the values of those elements
findLastN	n[i] = findLast(in,N);	find the indices of the last N non-zero elements.
findvalLastN	n[i], v[i] = findLast(in,N);	find the indices of the last N non-zero elements and the value of those elements.

For example functions v_findval.k implements Idea equation n[i],v[i] = find(in) and m_findLastN.k implements Idea equation r[i],c[i]=findLast(in).

5.8.6 Gather

The gather function takes an array in and an array of indices and returns the values at those indices. Vector, matrix and array3d gather functions. The root name is gather. The v_gather.k function implements the equation out[i] = in[indx[i]]. The m_gather.k function implements out[r][c] =

in[rindx[r]][cindx[c]]. And the a3_gather.k function implements out[x][y][z] = in[xindx[i]][yindx[i]][zindx[i]].

Note that

n[i],v[i] = find(in);

Is equivalent to

n[i] = find(in);
v[i] = gather(in,n);

5.8.7 Scatter

The scatter function scatters the values of a vector into a vector, matrix or array3d object. For example the v_scatter.k function implements the following to scatter the values of v into the input vector in and produce an output vector out:

out[i] = in[i];
out[indx[j]] = v[j];

The above is not a legitimate idea expression (out is assigned twice and the second assignment doesn't take a simple range variable as its input). Therefore the equation for the v_scatter function is

out[i] = scatter(in,v,indx);

And the equation for the m_scatter function is

out[r][c] = scatter(in,v,rindx,cindx)

and for an array3d function is

out[x][y][z] = scatter(in,v,xindx,yindx,zindx);

Note that if we have

n[i],v[i] = find(in);

We can reconstruct the input matrix in[n] as:

y[n] = 0;
z[n] = scatter(y,v,n);

This allows moving back and forth between full and sparse matrix representations of a vector, matrix or array3d object.

5.8.8 Mux/demux – family to time index conversion

The following generic functions found in the streams library can be used to multiplex an input family onto an output stream or demultplex an input family

fmux: out(i) = [i]in;

demuxf: [i]out = in(i);

mux2: Two input mux

demux2: Two output demux

5.8.9 Collecting family elements into single token – family to dimension conversion

<Input>_fam_<Output>: Where Input has 1 more dimension than Output, or Output has 1 more dimension than Input. For example generic functions v_fam_m.k takes a family of vectors and creates a matrix. It implements the Idea equation: out[r][c] = [r]in[c]; The v_fam_s.k kernel takes a vector and outputs a family of scalars. It has the idea equation: [n]out = in[n];

5.8.10 Reverse Vector or Matrix

Reverse functions reverse the values in a vector, matrix or array3d token along the specified direction.

The reverse functions are generic functions and are v_reverse.k, m_{r,c}reverse.k, a3_{x,y,z}reverse.k. So for example the m_rreverse.k kernel implements the equation: out[i][j] = in[#i-i-1][j] and v_reverse.k implements the equation out[i] = in[#i-i-1].

5.9 Delay

The generic streams/delay.k kernel produces an output that is the same as the input delayed by D. The first D values produced by the delay.k kernel have a value of 0 after which the tokens on the input are copied to the tokens on the output. The delay kernel can be used in a feedback loop to initialize the loop execution and maintain the state variable of the loop.

5.10 Vector Operations

Some functions take an entire vector as an input as opposed to appling the function to the elements of the vector. The following table lists functions of this kind provided in the streams library. The root name and data type of each function is given.

Root Name	Equation	Data Type	Description
fft	out[i] = dft(in)	xf,xd	power of 2 fft
ifft	out[i] = idft(in)	xf,xd	power of 2 inverse fft
ifftnd	out[i] = idftnd(in)	xf,xd	power of 2 inverse fft without final divide by N
rfft	out[i] = rdft(in)	f,d	power of 2 real fft – input is float – output is complex. output vector is half the size of input vector with nyquist point stored in out[0].im
rifft	out[i] = ridft(in)	f,d	power of 2 read fft – input is complex – output is real. Inverts results from rfft
rifftnd	out[i] = ridftnd(in)	f,d	Like rifft without final divide by N
norm	out = norm(in)	f,d,xf,xd	Root of sum of vector elements squared (2-norm)
norminf	out >?=abs(in[n]) out = norminf(in)	f	Infinity norm

norm1	out += abs(in[n]) out = norm1(in)	f	1-norm
dot	out += a[n]*b[n]	f,d,xf,xd	dot product
dotc	out += a[n]*conj(b[n])	xf,xd	conjugate dot product
dotVK	out += in[n]*VK[n]	f,d,xf,xd	dot product of stream with real (float,double) parameter
dotVX	out += in[n]*VX[n]	xf,xd	dot product of stream with complex parameter
dotcVX	out += in[n]*conj(VX[n])	xf,xd	conjugate dot product of stream with complex parameter
VXdotc	out += VX[n]*conj(in)	xf,xd	conjugate dot product of input parameter with stream
mean	out += in[n]/#n out = mean(in)	f	mean of vector
meansq	out += in[n]`2/#n out = meansq(in)	f	mean square of vector
meanabs	out += abs(in)/#n out = meanabs(in)	f	mean of absolute value of vector elements
stddev	out = stddev(in)	f	standard deviation of vector
var	out = var(in)	f	variance of vector
sort	out[n] = sort(in)	f	out is input vector sorted in ascending order (by default) Set parameter Up to 0 to sort in descending order
median	out = median(in)	f	median value of vector

5.11 Matrix operators

Operations that are peculiar to matrices.

5.11.1 Matrix Transpose

Root Name	Equation	Data Type	Description
trans	out[c][r] = in[r][c]	f,d,xf,xd	Matrix transpose
trans_ip	out[c][r] = in[r][c]	f,d,xf,xd	Inplace matrix transpose – same as matrix transpose but uses same memory for input and output. Conserves memory but can be much slower

5.11.2 Matrix multiply

Matrix and matrix-vector multiply kernels are provided. Variations are provided for transposing “T” and not-transposing “N” each matrix input.

mf_mmult: out[i][j] += a[i][k] * b[k][j];

mxf_mf_mmult: complex times real

mf_mxf_mmult

mf_mmultTT: out[i][j] += a[k][i] * b[j][k];

mf_mmultTN: out[i][j] += a[k][i] * b[k][j];

mf_mmultNT: out[i][j] += a[i][k] * b[j][k];

{mxf_mf,mf_mxf}_mmultTT: out[i][j] += a[k][i] * b[j][k];

{mxf_mf,mf_mxf}_mmultTN: out[i][j] += a[k][i] * b[k][j];

{mxf_mf,mf_mxf}_mmultNT: out[i][j] += a[i][k] * b[j][k];

mf_vf_mmult: out[i] += a[i][j] * b[j];

mf_vf_mmultT: out[j] += a[i][j] * b[i];

{mxf_vf_,mf_vxf_}_mmult{,T}

Parameter input boxes are also provided

mf_ mmult{,TT,TN,NT}MK, mf_MKmmult{,TT,TN,NT}, mxf_mmult{,TT,TN,NT}MK, mxf_mmult{,TT,TN,NT}MX, mxf_MKmmult{,TT,TN,NT}, mxf_MXmmult{,TT,TN,NT}, mf_ mmult{,T}VK, vf_MKmmult{,T}

5.11.3 Outer product

The outer product of two vectors forms a matrix has the root name outer and support f,d,xf and xd data types

outer: out[r][c] = a[r]*b[c]

5.11.4 Matrix decomposition functions

mf_qr: QR decomposition

mf_eig: Eigenvalue Decomposition (along with mf_eigsym2tri and mf_eigtri2di subcomponents)

mf_svd: Singular Value Decomposition

mf_lup: LU factorization with partial pivoting

mf_chol: Cholesky

mf_solve_lup: Solve using existing LU factorization from mf_lup

mf_solve_ut: Solve upper triangular system

mf_solve_lt: Solve lower triangular system

mf_solve_diag: Solve diagonal system

mf_solve: Solve least squares A x = b for x

5.11.5 Matrix Norms

mf_norm1: maximum of the 1-norm of the columns

mf_normfro: Frobenius norm (square root of sum of squares of all elements)

mf_norminf: maximum of the 1-norm of the rows

5.12 Stream to parameter conversion and constraints

Streams can be used to dynamically control the size of vectors and matricies. For example a stream can be used to control the N parameter of an s_v.k kernel that sets the size of the output vector. When using a stream in this way its value must be constrained so the Gedae compiler knows how much memory should be used to allocate the maximum size of the token. These constraints can be added using the streams/integer/si_constrain.k kernel which forwards the input to the output while adding a constraint to the output to indicate its maximum size to the compiler. The si_constrain.k kernel implements the Idea equation: out = constrain(in,Max). Two integer streams can be simultaneously constrained and at the same time their product can be constrained using the si_constrain2.k kernel. This kernel implements the Idea equation out1,out2 = constrain(in1,in2,Max1,Max2,Max12) which guarantees that out1<=Max1, out2<=Max2 and out1*out2<=Max12. Using such a joint constraint allows matrices whose sizes are controlled by out1 and out2 to only require Max12 elements to be allocated instead of the potentially larger Max1*Max2.

Because constrained streams can be used to control data token sizes of streams that are running at different rates it is often convenient to convert the stream to a parameter using the si_param.k kernel. This kernel will run at the rate of the stream input parameters that it controls can run at higher rates. The si_param.k kernel implements the Idea equation out = param(in);

5.13 Padding

It is often desirable to create a padded version of a matrix or vector where the padded edges are filled in a specified manner. Padding a matrix may be the first step before applying a neighborhood operator to the matrix. For an example a 128x128 matrix might be padded with 16 rows and columns on all sides of the matrix and the padded area is cleared to 0. Or we might just want to add a pad of 8 rows to at the beginning of the matrix but not at the end and no padding of the columns. And then fill the padded area with a mirror image of the data that follows the padded area. Also it might be desirable to remove an area from the the edges of the vector or token. We call this operation unpadding the vector or matrix. This section presents functions for padding, unpadding and filling padded areas.

To do vector padding use the kernels in streams/vector:

v_pad{,Begin,End}.k If the Begin or End suffix is not used both the beginning and end of the vector are padded. If the Begin suffix is used only the beginning of the vector is padded and if End is used only the end of the vector is padded. Matrix paddingis done with the kernels in streams/matrix. The kernels v_unpad{,Begin,End}.k does the inverse padding.

m_{r,c,rc}pad{,Begin,End}.k. The prefix r, c, or rc control whether padding is applied to the rows, columns or rows and columns of the matrix. The kernels m_{r,c,rc}unapd{,Begin,End}.k peforms the inverse padding.

There are three types of filling operations to fill a pad, clear, fill or mirror. The clear operation sets the padded area to 0 and leaves the unpadded part alone. The fill operation fills the padded area with the value at the edge of the pad. The mirror operation fills the padded area with a mirror image of the Pad elements of data at the edge of the pad. The following vector and matrix operations are provided

v_{clear,fill,mirror}pad{,Begin,End}.k

m_{r,c,rc}{clear,fill,mirror}Pad{,Begin,End}.k

Typically if a vector or matrix is padded then one of the three corresponding filling operations should be done. So for example an m_rcpadBegin.k could be followed by an m_rcmirroPadBegin.k. A v_pad.k could be followed by a v_clearPad.k. If a filling operation does not follow the pad the area in the pad remains uninitialized.

5.14 Schedule control

gettime – The streams/gettime.k function copies its input to its output (inplace and at no cost) and outputs on its time output the current wallclock time (where the wallclock time is a double and gives the number of seconds since the application started). This function can be used to measure the time that data passed along a given arc of the graph and can be used for reporting execution times or to supply times needed by real time control kernels.

The streams/gate.k kernel takes an input of any type or dimensionality in on its main input and copies it (at no cost) to its output. A second input t is of type int and is the trigger to the gate kernel. The t input must arrive before the gate executes. The gate function is useful for controlling the order of operations in a graph or for tying together parts of the application that have no other connection and are therefore not otherwise dataflow synchronized.

To easily generate a trigger to the gate from any token the streams/trigger.k kernel is provided. This kernel takes a token of any type or dimensionality on its input and outputs an uninitialized int on its output.

Both the trigger and gate functions have no cost in that they do not actually execute at runtime but are only in the graph to synchronize and control the order of functions.

6 Type conversion boxes:

A type conversion boxes exists for all token type (s, v, m, a3) and data types pairs (c, s, i, uc, us, ui, f, d xf, xd, zf, zd) x'd in the table below.

<token type><data type 1>_<token type><data type 2>

	c	s	i	uc	us	ui	f	d	xf	xd	zf	zd
c			x	x			x					
s			x		x		x					
i	x	x				x	x	x				
uc	x		x			x	x					
us		x	x			x	x					
ui			x				x					
f	x	x	x	x	x	x		x	x		x	
d			x				x			x		x
xf										x	x	
xd									x			x
zf									x			x
zd										x	x	

Examples:

scalar/char/sc_si: scalar char to int conversion
vector/int/vi_vf: vector int to float coversion
matrix/complex/mxf_mxd: matrix single precision complex to double precision complex.
scalar/float/sf_sxd: scalar float to double precision complex.
array3d/char/a3xf_a3zd: 3d array char to double precision split complex

6.1 Complex-to-real

In addition sxf_complex creates a complex output from scalar float real and imaginary stream inputs. sxd_complex creates a dcomplex output from scalar double real and imaginary stream inputs. The kernels sxf_split and sxd_split create real and imaginary outputs from a complex input, and the kernels sxf_real, sxd_real, sxf_imag, and sxd_imag create real or imaginary outputs.

7 Token conversion boxes

Token conversion kernels convert spatial dimensions to a time index and the reverse. Token conversion boxes are provided between very different token type: {s,v,m,a3}. Kernels are:

Kernel Path	Equation
streams/vector/v_s.k	out(n) = in[n];
streams/vector/s_v.k	out[n] = in(n);
streams/matrix/m_v.k	out[c](r) = in[r][c];
streams/matrix/v_m.k	out[r][c] = in[c](r);
streams/matrix/m_s.k	out(r,c) = in[r][c];
streams/matrix/s_m.k	out[r][c] = in(r,c);
streams/array3d/a3_m.k	out[y][z](x) = in[x][y][z];
streams/array3d/m_a3.k	out[x][y][z] = in[y][z](x);
streams/array3d/a3_v.k	out[z](x,y) = in[x][y][z];
streams/array3d/v_a3.k	out[x][y][z] = in[z](x,y);
streams/array3d/a3_s.k	out(x,y,z) = in[x][y][z];

streams/array3d/s_a3.k	out[x][y][z] = in(x,y,z);

A kernel whose input token has more input dimension than output dimensions has an input parameter named after the new dimensions to be added (N,C,R,X,Y,Z). A kernel whose input token has fewer input dimension than output dimensions has an output parameter named after the dimension or dimensions that was converted to a time index (N,C,R,X,Y,Z).

The above kernels are all inplace and are merely a reinterpretation of the input data. Streaming versions that can be efficiently used to read vectors out of a sub matrix tile are also available. These kernels are named <token_type>_stream_<token_type>. For example m_stream_v.k and v_stream_m.k are functionally equivalent to m_v.k and v_m.k.

7.1 Overlap

A common function is to convert a stream of scalars into an overlapping stream of vectors. The generic streams/vector/s_ovrl_v.k kernel does this and implements the equation:

out[i] = in(i-Ovrl);

Where the N parameter input determines the size of range of i.

8 E Function Completeness

Each E function should be referenced in at least one kernel. The following table lists the root that corresponds to the E function name. Any E function not referenced is should be placed in the attic and be subject to removal from the library. (In the table below "convert" refers to the full set of type conversion kernels which do not have a rootname.)

e_*dotc.c	dotc	e_*dotpr.c	Dot	e_*fftwts.c	N/A
e_*fird.c	fir	e_*maxmgv.c	maxabs	e_*maxv.c	max
e_*maxv0.c	max	e_*meamgv.c	meanabs	e_*meanv.c	mean
e_*measqv.c	meansqr	e_*minmgv.c	Minabs	e_*minv.c	min
e_*minv0.c	min	e_*mmul.c	Mmult	e_*mmulacc.c	mmultacc
e_*mtran.c	trans	e_*mtran_ip.c	trans_ip	e_*mtran2.c	subtrans
e_*polar.c	rec2pol	e_*rect.c	pol2rec	e_*rfftwts.c	N/A
e_*rvadd.c	add	e_*rvdiv.c	div	e_*rvmul.c	mult
e_*rvsub.c	sub	e_*svdiv.c	div	e_*sve.c	sum
e_*svemg.c	sumabs	e_*svemgs.c	sumabssqr	e_*svesq.c	sumsqr
e_*svessq.c	sumsignsqr	e_*svmul.c	mult	e_*svsub.c	sub
e_*temp.c	N/A	e_*vaa.c	DELETE	e_*vaam.c	DELETE
e_*vabs.c	abs	e_*vacos.c	acos	e_*vadd.c	add
e_*vafix.c	*convert*	e_*vam.c	DELETE	e_*vasbm.c	DELETE
e_*vasin.c	asin	e_*vasm.c	DELETE	e_*vasub.c	DELETE
e_*vatan.c	atan	e_*vatan2.c	atan2	e_*vceil.c	ceil
e_*vclip.c	clip	e_*vclr.c	pad	e_*vcomb.c	*convert*

e_*vconj.c	conj	e_*vcos.c	cos	e_*vcosh.c	cosh
e_*vcub.c	DELETE	e_*vdct.c	dct	e_*vdiv.c	div
e_*vdump.c	N/A	e_*veq.c	eq	e_*veql.c	eq
e_*vexp.c	Exp	e_*vexp10.c	exp10	e_*vexp2.c	exp2
e_*vfftb.c	Fft	e_*vfftbcols.c	cfft	e_*vfill.c	constant
e_*vfirst.c	Find	e_*vfloat.c	*convert*	e_*vfloor.c	floor
e_*vgathr.c	Gath	e_*vge.c	ge	e_*vgt.c	gt
e_*vhypot.c	hypot	e_*vifftbnd.c	Ifftnd	e_*vifix.c	*convert*
e_*vklip.c	DELETE	e_*vlast.c	find	e_*vle.c	le
e_*vlim.c	DELETE	e_*vlint.c	Interp2	e_*vlnot.c	not
e_*vlog.c	log	e_*vlog10.c	log10	e_*vlog2.c	log2
e_*vlt.c	lt	e_*vma.c	multadd	e_*vmags.c	abssqr
e_*vmax.c	max	e_*vmaxmg.c	maxmg	e_*vmin.c	min
e_*vminmg.c	minmg	e_*vmma.c	DELETE	e_*vmmsb.c	DELETE
e_*vmov.c	copy	e_*vmrg.c	merge	e_*vmsa.c	multadd
e_*vmsb.c	multsub	e_*vmsn.c	submult	e_*vmul.c	mult
e_*vnabs.c	DELETE	e_*vne.c	ne	e_*vneg.c	neg
e_*vneql.c	Ne	e_*vphas.c	DELETE	e_*vpolre.c	DELETE
e_*vpow.c	Pow	e_*vprog.c	acc	e_*vramp.c	ramp
e_*vrcip.c	DELETE	e_*vrecip.c	recip	e_*vrectp.c	DELETE
e_*vrfft.c	Rfft	e_*vrfftnd.c	rfftnd	e_*vround.c	round
e_*vrsmul.c	mult	e_*vrvrs.c	reverse	e_*vsadd.c	add
e_*vsam.c	DELETE	e_*vsbm.c	DELETE	e_*vsbsbm.c	DELETE
e_*vsbsm.c	DELETE	e_*vscale.c	DELETE	e_*vscatr.c	scat
e_*vsdiv.c	div	e_*vselect.c	select	e_*vsfix.c	*convert*
e_*vshrink.c	DELETE	e_*vsin.c	sin	e_*vsinh.c	sinh
e_*vsma.c	multadd	e_*vsmsa.c	multadd	e_*vsmsb.c	multsub
e_*vsmul.c	mult	e_*vsort.c	sort	e_*vsplit.c	*convert*
e_*vsq.c	sqr	e_*vsqrt.c	sqrt	e_*vss.c	DELETE
e_*vssq.c	signsqr	e_*vsub.c	sub	e_*vsubsq.c	DELETE
e_*vswap.c	N/A	e_*vtan.c	tan	e_*vtanh.c	tanh
e_*vthr.c	thresmax	e_*vthres.c	DELETE	e_*vtrunc.c	trunc
e_*vuafix.c	*convert*	e_*vuifix.c	*convert*	e_*vusfix.c	*convert*
e_a3mov.c	getcopy	e_conv2d.c	conv	e_dctwts.c	N/A
e_fftwtscols.c	N/A	e_ivand.c	bitand	e_ivmod.c	mod
e_ivor.c	bitor	e_minsrt.c	setcopy	e_mmov.c	getcopy
e_rmsqv.c	norm	e_sacorr.c	acorr	e_sccoh.c	ccoh
e_sccorr.c	ccorr	e_scorr.c	corr	e_sfact.c	fact
e_slsq.c	DELETE	e_smeandev.c	DELETE	e_srange.c	DELETE
e_sstddev.c	stddev	e_subsqs.c	DELETE	e_svar.c	var
e_swmean.c	DELETE				