Brook+ Release Notes

Stream Computing SDK 1.4.1

1. Resolved Issues

- Scatter is now supported for 8- and 16-bit vectors of length 3 or 4.
- Using domain on structure streams gives correct result now.
- The `binomial_option` sample is modified to use correct algorithm.

.2. Known Issues

2.1. Brook+ Compiler (brcc)

- brcc generates incorrect code if the index expression has an array element. For example:

  ```
  A[B[i]]
  ```

- Implicit type conversion of double vectors may cause brcc to crash. Always use explicit type casting when casting double vectors to another data type.
- brcc treats “line” and “transpose” as keywords. It is necessary to choose alternative names for functions and variables in Brook+.
- Warning levels can be set using the brcc `-w` option. See `brcc -h` for a list of all command-line options.
- brcc does not perform implicit type promotion; and type mismatches are now errors.
- Passing scatter streams to sub-kernels are not allowed.
- Declaring structure inside br file requires data type to be specified for each struct element. For example:

  ```
  typedef struct { float x, y; } XY; // compilation error.
  typedef struct { float x; float y; } XY; // Compilation successful.
  ```

- Operator ?: and && are not supported for double data type.
- Any structure containing pointers should not be declared inside br file, even if it is not used inside kernel.

2.2. Brook+ Runtime

- Brook+ supports automatic input stream resizing when the size of the input stream size does not match the output stream size; however, automatic resizing may affect performance. To disable automatic input stream resizing, set the `BRT_DISABLE_STREAM_RESIZE` environment variable.
- The CPU backend does not have an error handling and reporting mechanism. It can crash if passed invalid inputs.
- Brook+ kernels with non-void return types cannot be called from non-kernel code. Ensure that all kernels to be called from non-kernel code have `void` as the return type.
- If your Brook+ application under Windows cannot find DLLs that appear to be in the path, ensure that the Active Solution Platform is set correctly for your development machine (32-bit vs 64-bit).
- The runtime enforces the rule given in the Brook+ language specification that makes it illegal to bind the same stream for both reading and writing. For compatibility with existing code, this type of aliasing is permitted if the environment variable `BRT_PERMIT_READ_WRITE_ALIASING` is defined.
- We do not recommend this for new code because unpredictable results can occur without a detailed understanding of the underlying hardware architecture.
  - The maximum size of a stream is 2^{26} elements.

    - The underlying hardware only supports a very small (32-deep) call stack. This does not directly translate into a fixed limit for user code when the various levels of compilation add and remove levels of hierarchy as part of their processing. However, users must be aware that kernels exhibiting significant recursion are likely to fail. Use iterative forms where possible.

2.3. Brook+ Kernel Language

- A parameter of a kernel cannot have the same name as the kernel.
- Switch statements are not supported in the Brook+ kernel language.
- Scatter is supported on cards that have the necessary hardware support.
- Only a single scatter stream is supported per kernel.
- All non-stream parameters passed to a kernel are treated as constants. To modify their values, explicitly copy them into a local temporary variable.
- Neither unions nor arrays are permitted as stream elements.
- C-style indexing of gather/scatter arrays has been introduced. brcc now issues a warning if non-C-style indexing is used for gather/scatter arrays.

  Although non-c-style indexing still functions, it is strongly recommended that you modify your code to use the new C-style indexing method instead.

- All gather array arguments must be declared with a constant integer for all dimensions. To allow unspecified dimensions for gather arrays, you must disable cached gather arrays by specifying `-c` when invoking brcc.

2.4. Thread Data Sharing

- When using thread data share in Brook+, the Brook+ runtime must switch to compute shader mode. When this happens, the Brook+ runtime issues the following warning.

```
CALRuntime uses a compute shader mode to execute “brookKernel” kernelMultitheaded
code generated for cpu backend code. This is normal and expected behavior.
```

- Thread data sharing does not work with double data types.

2.5. brcc Preprocessor

- Only comments ending on the same line as the preprocessor directive are supported on the same line as the preprocessor directive.
- brcc does not run its preprocessor stage by default. If your program requires reprocessing, invoke brcc with the `-pp` flag. See the *Brook+ Programming Guide* for more information about supported brcc preprocessor directives.

2.6. Vectors

- Arithmetic operations are not supported with vector constructors in non-kernel code.
- Relational operators on short vectors in conditional expressions assume an x component as the conditional expression. When using the output of a relational operator as the input to a conditional expression, only the x component of the value is considered. If your application requires full component-wise conditional expressions, you must operate on each component individually.

  When you perform an operation on short vectors, the expected behavior is that:

```
float4 a,b;
float4 c;
c = a + b;
```

is the same as:

```
c.x = a.x + b.x;
c.y = a.y + b.y;
c.z = a.z + b.z;
c.w = a.w + b.w;
```

However, for relational operators, such as `a < b`, the following code illustrates the difference.

```
d = a < b ? a : b
```

is the same as:

```
bool4 c;
c.x = a.x < b.x;
c.y = a.y < b.y;
c.z = a.z < b.z;
c.w = a.w < b.w;
d.x = c.x ? a.x : b.x;
d.y = c.x ? a.y : b.y;
d.z = c.x ? a.z : b.z;
```

2.7. Floating Point

- Double precision is supported on cards that have the necessary hardware support.
- brcc does not automatically promote or downcast between float and double. You must add explicit casts.

2.8. Compilation Issues

- Microsoft Visual Studio 2002 is known to be incompatible due to template support issues.
- Defining `HIDE_BROOK_TYPES` hides all new Brook+ types (fixed, float4, int3, etc.), so they do not conflict with user identifiers.
- It is possible to create 32-bit Brook+ applications on 64-bit systems using the following procedure:
    - Install the 32-bit version of Brook+.
    - Copy the libraries from the 32-bit version into a separate directory.
    - Uninstall the 32-bit version of Brook+.
    - Install the 64-bit version of Brook+.
    - Create a lib32 directory, and copy the 32-bit-version libraries there.
- When building Brook+ on windows using Cygwin, use version 3.79 of the `make` utility. Currently, building with version 3.81 of `make` is not supported.

2.9. Compatibility with ATI Stream SDK Versions Prior to 1.3

- The behavior of the `domain` and `execDomain` operators has changed from SDK versions prior to 1.3. For backward compatibility, define the macro `USE_OLD_API`.
- Brook+ applications written for SDK versions prior to 1.3 must be recompiled with 1.4 to work with the current version of Brook+.

- A change in the directory structure might make the Brook+ applications written with SDK versions prior to 1.3 incompatible. The header files that were in the `$(BROOKROOT)\sdk\include\brook` folder and that have been moved to `$(BROOKROOT)\sdk\include\brook\CPU` are:

  ```
  brt.hpp
  brtarray.hpp
  brtintrinsic.hpp
  brtscatter.hpp
  brtscatterintrinsic.hpp
  brtvector.hpp
  janitorial.hpp
  type_promotion.hpp
  ```

  To eliminate this problem, we have placed a dummy header file, `brt.hpp`, inside `$(BROOKROOT)\sdk\include\brook`. This file includes the real `brt.hpp`.

- The Brook+ compiler now enforces semantic checking and supplies useful warnings and error messages. Some samples prior to SDK 1.3 may not build without warnings or errors. You can suppress these warnings by adding one or both of the flags `-a` and `-c` to the brcc command line.

In Visual Studio:

- Right click the `.br` file, and click Properties.
- Go to Custom Build step.
- In the Command Line field, add one or both of the `-a` and `-c` flags to the brcc command right after `$(BROOKROOT)\sdk\bin\brcc_d.exe`.

In Makefiles, open the appropriate Makefile, and modify the brcc command line to add `-a` and `-c`.

- Support for variable output kernels using the `vout` and `push` keywords has been removed. Use in-kernel gather and scatter instead.

4. Deprecated Interfaces

- The following three interfaces have been deprecated:

`streamScatterOp –`

```
streamScatterOp( dst_stream, index_stream, input_stream, reduce void
*(scatter_OP)(...))
```

This can be implemented (naive implementation) by writing a kernel to gather an element.

```
kernel void gatherElement(<type> dst_stream[], <type> value_of_index_i,
                          out <type> tempOut_stream<>)
{
    tempOut_stream = dst_stream[value_of_index_i]; // tempOut_stream is
                                                   // a temporary stream.
}

kernel void scatterElement(<type> tempOut_stream<>, <type> value_of_index_i,
                           <type> dst_stream[])
{
    dst_stream[value_of_index_i] = tempOut_stream;
}

for( i = 0; i < index_length; i++)
{
    gatherElement(dst_stream, index[i], tempOut_stream);
    scatterElement(tempOut_stream, index[i], dst_stream);
}
```

streamGatherOp –

```
streamGatherOp( dst_stream, index_stream, src_stream, reduce void
*(fetch_OP)(...))
```

This can be implemented (naive implementation) by writing a kernel to gather input elements with the values of the `index_stream`.

```
kernel void gatherElement(<type> src_stream[], <type> value_of_index_i,
                          out <type> tempOut_stream<>)
{
    tempOut_stream = src_stream[value_of_index_i];
}

kernel void copy(out <type> dst_stream[], <type> value_of_index_i,
                 <type> tempOut_stream<>)
{
    dst_stream[value_of_index_i] = tempOut_stream;
}

for(i = 0; i < index_length; i++)
{
    gatherElement(src_stream, index[i], tempOut_stream);
    copy(dst_stream, index[i], tempOut_stream);
}
```

streamSwap –

```
streamSwap(src1_stream,src2_stream)
```

This is required only when you must invoke the same kernel again with interchanged stream values. In this case, it is not necessary to write an extra kernel. You can unroll the loop with a factor of 2 and invoke consecutive kernels by swapping stream names. This implementation foregoes unnecessary stream object creations, which `streamSwap` does.

If you are using the C++ API, `streamSwap()` is easy to implement. This looks like:

```
::brook::Stream<type> temp = src1;  src1 = src2;  src2 = temp;
```

This invokes the copy constructor of `Stream<T>` class, which copies `src1` to `temp` at the time of its creation. This technique is used for the `samples/CPP/PrefixSum` sample.

And `sample/legacy/prefix_sum` does this:

```
<type> temp; // Temporary stream variable which has the same dimensions
             // as of src1/src2.temp = src1;src1 = src2;src2 = temp;
```

5. Browser

- Firefox is the preferred browser.

6. C++ API vs Legacy C API

- The recommended programming technique is to use the C++ API. It is recommended that you separate the kernel from the non-kernel code by placing them in different files. The legacy Brook+ style is not recommended for new code because it does not expose the full feature set of Brook+.