Outline

- Compiler Goals
- Open64 Status
- Reservoir compiler Vs Open64
- “Leveraging” Reservoir compiler
Compiler Goals

- Memory allocation
- Synchronization
- Brook operators
- Stream scheduling
- Double buffering
- Kernel Conditionals
- Multi-node Support
- Liveness Analysis

Necessary passes:
- Strip mining
- Software Pipelining
- Kernel partitioning
- Kernel coalescing
- Operator combining
- Common flow analyses
- Record Partitioning
- Reorder Kernels
- Multi-node Synchronization
- Work partitioning
- Data Partitioning

Global optimizations:
- Double buffering
- Kernel Conditionals
- Multi-node Support
- Liveness Analysis

Native compiler
SVM printer
BRT printer
Merrimac/scalar printer
Kernel scheduler
Kernel printer

Brook Frontend
Brooktran Front-end
Common IR
Open64 Implementation Plan

- Implement brook and brooktran front ends
  - Modify gcc and fortran front ends of Open64
  - Generate High WHIRL representation

- Implement necessary/optimization passes in back end on the High WHIRL representation
  - Generates an optimized High WHIRL representation

- Implement BRT, SVM, and Merrimac Printers
  - Modify the tool “whirl2c” which works on High WHIRL

- Develop interfaces with kernel scheduler/low level compiler
Open64 Status: Brook Front End

- Parsing: Add keywords to lex tables
  - Variable qualifiers: stream, memstream, out, reduce
  - Function qualifier: kernel
  - UPC qualifiers: relaxed, shared, strict
- Propagate keywords into gcc tree
  - Set flags in gcc tree for the qualifiers
- Propagate keywords into WHIRL tables
  - Set flags in WHIRL symbol tables

- Main issues
  - Files absent in gcc front end distribution
  - No “space” in WHIRL type qualifier table
## Interface to Back End (1)

### Queries on streams

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>BOOL TY_is_stream (TY_IDX ty_idx)</code></td>
<td>Is this a stream?</td>
</tr>
<tr>
<td><code>BOOL TY_is_memstream (TY_IDX ty_idx)</code></td>
<td>Is this a memstream?</td>
</tr>
<tr>
<td><code>BOOL TY_is_outstream (TY_IDX ty_idx)</code></td>
<td>Is this an “out” stream?</td>
</tr>
<tr>
<td><code>BOOL TY_is_reduce (TY_IDX ty_idx)</code></td>
<td>Is this scalar being “reduced”?</td>
</tr>
<tr>
<td><code>BOOL TY_is_grouped_stream (TY_IDX ty_idx)</code></td>
<td>Is this a grouped stream?</td>
</tr>
<tr>
<td><code>BOOL TY_is Derived_stream (TY_IDX ty_idx)</code></td>
<td>Is this a stream derived from stencil/group/domain?</td>
</tr>
<tr>
<td><code>BOOL TY_is_stencil_stream (TY_IDX ty_idx)</code></td>
<td>Is this a stencil stream?</td>
</tr>
</tbody>
</table>

### Queries on functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>BOOL PU_is_kernel_function (const PU&amp; pu)</code></td>
<td>Is this a kernel function?</td>
</tr>
<tr>
<td><code>BOOL Intr_is_stream_operator (const INTRINSIC I)</code></td>
<td>Is this intrinsic function a brook stream operator?</td>
</tr>
</tbody>
</table>
### Stream properties in overloaded Array Table

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST_IDX ARB-derived_var (const ARB HANDLE stream)</td>
<td>Symbol Table index for stream from which the stencil/group stream is derived</td>
</tr>
<tr>
<td>INT64 ARB_stride_val (const ARB HANDLE stream)</td>
<td>Stride value for a strided derived stream</td>
</tr>
<tr>
<td>BOOL ARB_const_shape (const ARB HANDLE stream)</td>
<td>Does the stream have a constant shape?</td>
</tr>
<tr>
<td>BOOL ARB_dynamic_shape (const ARB HANDLE stream)</td>
<td>Does the stream have a dynamically changing shape?</td>
</tr>
<tr>
<td>STR_BND ARB_lbnd_boundary (const ARB HANDLE stream)</td>
<td>Stencil boundary condition for left side (in that dimension)</td>
</tr>
<tr>
<td>STR_BND ARB_ubnd_boundary (const ARB HANDLE stream)</td>
<td>Stencil boundary condition for right side (in that dimension)</td>
</tr>
<tr>
<td>INT64 ARB_lbnd_val (const ARB_HANDLE stream)</td>
<td>Value of lower bound of stencil/group/domain if constant</td>
</tr>
<tr>
<td>INT64 ARB_ubnd_val (const ARB_HANDLE stream)</td>
<td>Value of upper bound of stencil/group/domain if constant</td>
</tr>
</tbody>
</table>
Necessary Passes

- **Liveness Analysis implementation**
  - Live variables after/before every statement (implemented)

- **Brook Operators**
  - Treated as intrinsic function calls
  - Get handle into intrinsic table in front end
  - “Expand” the operators in back end (done in lno phase)

- **Memory allocation/deallocation of streams, scalar stream synchronization**
  - Designed algorithms to implement these passes
Ongoing work

- Final stages of Front end implementation
  - Some more flags to be set
  - Propagate qualifiers when “typedef” is used
- Brook Runtime Printer
  - Alan is working on whirl2brt
- Error checks
  - Check declarations
  - Match declarations of actual and formal kernel parameters
- Test cases
  - Write test cases to verify sanctity of the compiler
Open64 Vs Reservoir Compiler

- **Open64**
  - Gone through cold start period
  - Have complete control
  - Freely distributable
  - Fairly robust – used in several places
  - Has Fortran/C++ front end

- **Reservoir Compiler**
  - Effort reuse
  - Several people working on it
  - More time
  - More experience
Retargeting Reservoir Compiler to Merrimac

- Merrimac printer
- UPC multi-node support
- ScatterOp/GatherOp support
  - Reservoir may not support it
- Conditionals inside kernels
  - Predication Vs Conditional Streams
- Multidimensional stream optimizations
- Stream Scheduling (specific to Merrimac)
- Other Global Optimizations
Questions

- UPC?
  - What does Reservoir use for multi-node memory model?

- Compiler Optimizations/Passes
  - What optimizations does reservoir plan to implement?

- When will we get access to the Reservoir compiler?

- Open64 or Reservoir compiler?