Compiling Brook to StreamC

EE482C – Spring 2002 Class Project

Abhishek Das

Mattan Erez

Jayanth Gummaraju

Motivation

- **Brook is a high level stream language**
  - architecture independent
    - retargetable
    - no performance evaluation
  - easy to code
    - port existing applications
    - write new code
- **StreamC/KernelC**
  - fairly Imagine specific
  - hard to code
    - SIMD
    - strict syntax

Outline

- **Goals and accomplishments**
- **Specifics**
- **Stream Transformations**
- **Optimizations**
  - self-product operator
  - strip-mining
- **Conclusions**
- **Future work**

Goals and Accomplishments

- **Develop basic compiler framework**
  - used the Brook meta-compiler
    - generate StreamC code and files
- **Exercise the framework on StreamMD**
  - tested a toy program with similar structure
  - currently testing full StreamMD code
- **Perform some optimizations**
  - optimized self-product operator
  - general strip-mining

Specifics

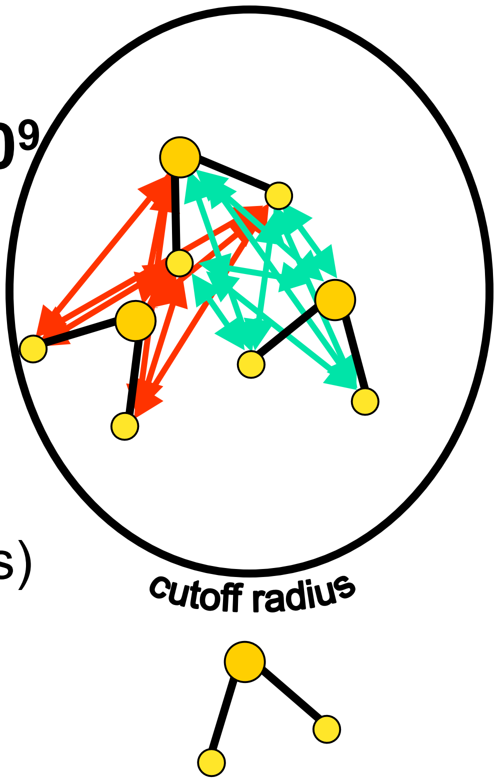
- **Stream Declarations**
- **Kernel Declarations**
  - prototypes and *_kc.cpp files
- **Kernel Calls**
  - simple kernels
  - Filestreams
  - reductions
- **Stripmining**
  - self-product operator
  - general

Transformations

- **File streams**
  - transform file manipulation kernels into functions
  - convert streams to vectors for I/O
- **Stream lengths**
  - Brook abstracts away stream length
  - currently only handle 1:1 ratio kernels
  - length is determined by the input files
- **Expanded streams**
  - just self-product in StreamMD
  - automatically strip-mine instead of expand→reduce

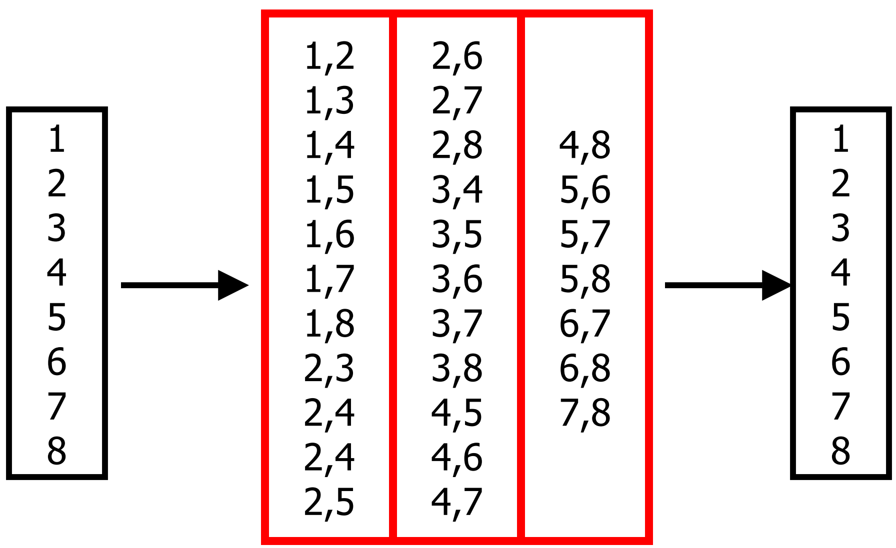
StreamMD

- **Molecular dynamics simulation of water**
  - an integral part of protein-folding simulation
  - simulation involves 10,000 – 100,000 molecules
- **For each time-step (repeated 10^8 – 10^9 times)**
  - calculate electro-static interactions
    - use cut-off to approximate the force
    - for all molecules within cut-off calculate the forces between all atoms
    - roughly 500 operations per pair (6 forces)
  - calculate spring-forces within a molecule
  - update velocity and position



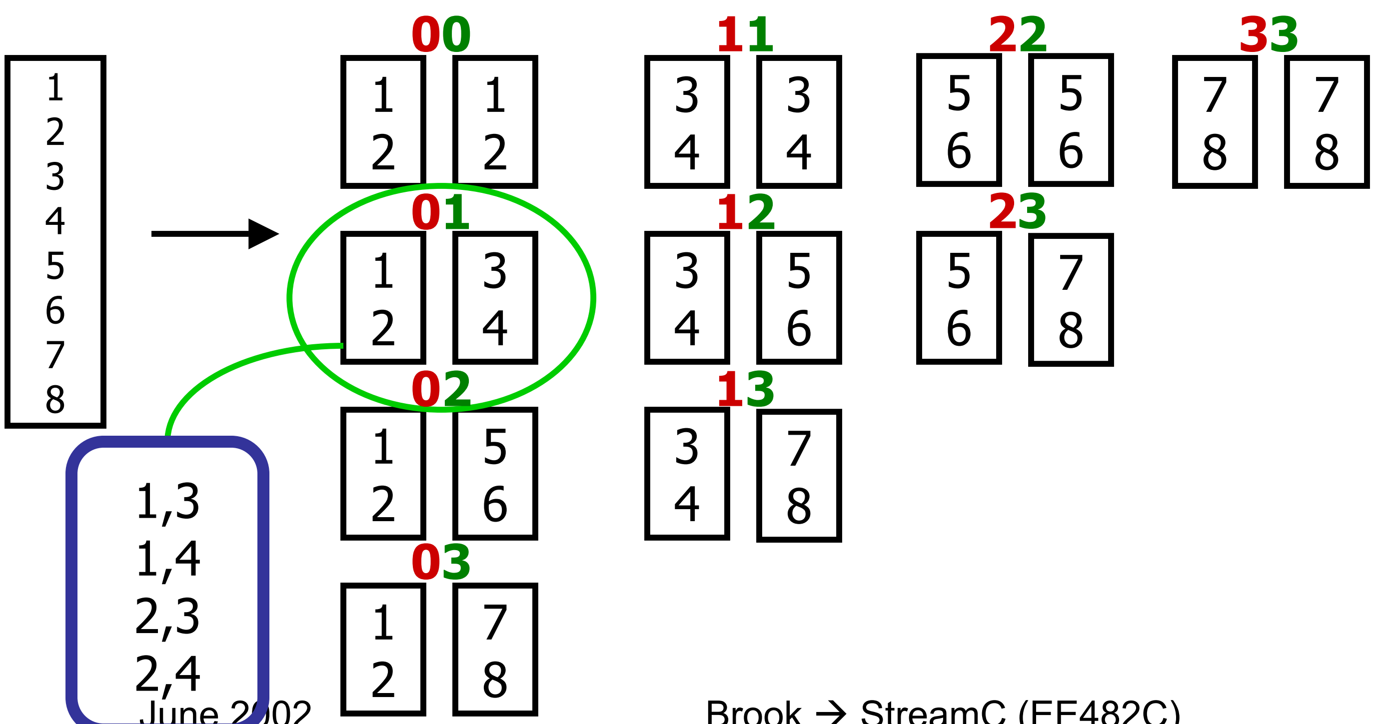
Optimizations – Self-Product (1)

- **Used to express the electro-static interactions between molecule pairs**
  - position_pairs = selfProduct(positions);
    Interact(position_pairs, reduce forces);
- **Expand to *O(n²)* records reduce back to *n***



Optimizations – Self-Product (2)

- **Optimization – never create *O(n²)***
  - combine the expansion and reduction in a strip-mined kernel and call sequence
  - for all *base_strips* ***i*** then for all *pair_strips* ***j > i***



Optimizations – Self-Product (3)

- **Need to do local expand on the bases**
  - loop over all bases for each pair record
  - all bases must fit into local storage
  - another level of strip-mining

```
for (big_base=0; big_base<N; big_base+=SRF_strip_size) {
  for (base=big_base; base<big_base+SRF_strip_size; base+=LRF_strip_size) {
    Interact(base, base);
    for (pair=base; pair<big_base+SRF_strip_size; pair+=LRF_strip_size) {
      Interact(base, pair);
    }
  }
  for (pair=big_base; pair<N; pair+=SRF_strip_size) {
    for (base=big_base; base<big_base+SRF_strip_size; base+=LRF_strip_size){
      Interact(base, pair);
    }
  }
}
```

Optimizations – Strip-mining

- **Brook**
  - `STRIP_BEGIN()`
  - `STRIP_END()`
- **Streamc**
  - take feedback from profiler
  - loop:

```
STREAMC_strip_size = STREAMC_STRIP_SIZE; //Set this to what profiler suggests

 for(int stripmine_start=0; ; ) {
          a_stripmine = a(stripmine_start, stripmine_start + STREAMC_strip_size);
          …          …
          // operate on a_stripmine
          …          …
          stripmine_start += STREAMC_strip_size;
          if_VARIABLE(stripmine_start == KERNEL_opstream_length) break;
}
```

Reduction Variables

- **Brook**
  - reduce to a variable
- **StreamC**
  - uc variables
  - ucRead after kernel calls
  - convert to Imagine types
  - reduction of complex types
    - structures
    - arrays

Conclusions

- **Stateless kernels are easy to convert**
- **Self-product is made efficient by exploiting each level of the bandwidth hierarchy**
- **Brook to streamC is possible**

Future work

- **Performance evaluation**
- **Profiling and software pipelining**
- **Stream operations**
  - Handle product the same way as self-product
  - Stencils and groups
- **Automate kernels**
  - Different I/O rates
  - mout (arbitrary number of outputs per input)
  - Stencils and groups
- **Split and merge kernels**
  - e.g. file-streams and computation together in brook kernels