Stanford Streaming Supercomputer

Brook Applications

Ian Buck
Anand Ramalingam

Stanford University

January 7, 2004
Convolution

• Let $s$ be the input and $w$ be the weights and let $slen$, $wlen$ be their lengths.
  – The convolved output $t$ is given by
    $$t[n] += w[m] \times s[n-m]$$
    where,
    $$0 \leq n < (slen+wlen)$$
    $$\max(0, n-slen+1) \leq m \leq \min(wlen, n)$$

• References
Observations

• Parallelism.
  – The outputs $t[0] \ldots t[wlen+slen-1]$ don’t depend on each other. They can be computed independently.

• We can give each processor a part of summation to work on.
  – One processor can do $t[0]$, other can do $t[1] \ldots$
  – More localization and less communication

• Result, make *convolve* a *kernel* function
Brook Implementation

• Brook functions used
  - LoadStream()
  - streamGetLength()
  - streamSetLength()
  - streamZero()
  - streamDomain()

  • streamDomain() gets back a specific range of the stream which we need to compute the convolved output

  - kernel void DoConvolve(floats *t, floats *s, float w) {

      *t += *s * w;

  }
Fast Fourier Transform

• Two Approaches
  – Decimation in Time
    • Input bit-reversed
    • Output is in the natural order
  – Decimation in Frequency
    • Input is in the natural order
    • Output bit-reversed

• Reference
Decimation in Time
Decimation in Frequency
Observations

• In place computation
  – Suited for streams are views than streams are data.

• Memory access in strides

• We are implementing Decimation in Frequency
  – Inputs are in the natural order
  – Keep the bit-reversal at the very last

• Caveat: The algorithm implemented works only for the powers of 2 sequences.
  – The PCA document gives the lengths as powers of 2 and doesn’t mention any FFT algorithm to use.
FFT (First Cut)

```c
stream struct complex s[1024];
stream struct complex t[512];
stream struct complex *f;
FileStream fp ("data.txt", "rt");
for (;;) {
    // Read in 1K values
    LoadData (fp, s);
    if (streamGetLength (s) != 1024) break;
    // Loop 10 times.  10 = log base 2 of 1024.
    for (i=0; i<10; i++) {
        t = steamDomain(s, 512, 1024);
        DoDFT ( s, t, s );
    }
    s = streamBitReverse(s);
    f = streamCat(f, s);
}
```
Matrix Multiplication (First Cut)

streamcmplx a[10][5], b[5][10];
cmplx c[10][10];

streamcmplx *rowView;
streamcmplx *colView;
for(nRowIndex=0; nRowIndex < 10; ++nRowIndex){
    rowView = a[nRowIndex];
    for(int nColIndex = 0; nColIndex<10; ++nColIndex){
        // b_t is a transposed version of b
        colView = b_t[nColIndex];
        int temp;
        Matmult(rowView, colView,&temp);
        c[nRowIndex][nColIndex] = temp;
    }
}

// MATRIX MULTIPLY KERNEL
kernel void MatMult(streamcmplx *row, streamcmplx *col,
                    out cmplx reduce *res){
    *res += (*row) * (*col);
}
Suggestions(1)

• Standard Library for Brook.
  – Similar to C Standard Library

• Dr. Fatica requested Fortran90 support.
  – I request OO support specifically for a subset of C++, Brook++!
    • Pro
      – Consider Matrix Multiplication,
        • Easy to construct a Matrix class
        • Bundle all the operations like multiplication into the class
        • Reusability
      – More natural to say
        • Matrix c = a*b than c = Matmult(a,b)
    • Con
      – Difficult to define what to leave out
Suggestions(2)

• The following is a wish-list for Brook++!
  – Needs
    • Templates, STL
    • Strict type safety
    • Array references to be bounded
    • Brook makes the Memory Management transparent
      – Do we need pointers?
      – Can we stick with references?
    – Please Avoid them
      • No C style casting
      • No MI in the C++ way.

• References
  – Markku Sakkinen, *The darker side of C++ revisited*, 1993
Suggestions(3)

- More documentation
  - `Stream<type>` was mistaken as a template!
    - This has been fixed.
  - `mat2f`, `mat2i` are bit misleading
Finally …

• Thank you
  – Prof. Rosenblum, for an opportunity to work in this project
  – Ian,
    • for creating Brook
    • helping me out develop these stuff.