Parallelizing Audio Feature Extraction

Using an Automatically-Partitioned Streaming Dataflow Language

Eric Battenberg, Mark Murphy

Introduction

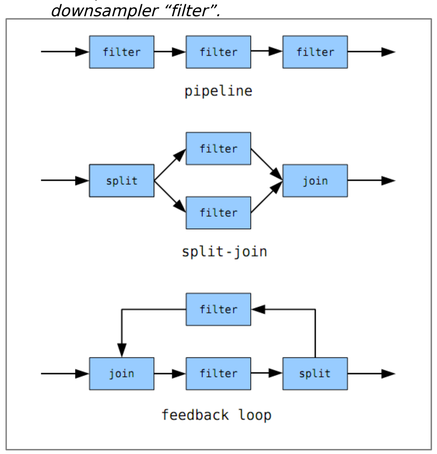
- Music information retrieval is becoming increasingly important as the size of digital music archives continues to grow.
- Dataflow languages can greatly improve programmer productivity for audio applications.
- The StreamIt compiler can automatically partition the work described by its dataflow code amongst multiple cores.

The StreamIt Language [1]

- Basic building blocks
  - Filter – like a function
  - Pipeline – cascade of filters
  - Split-Join – task-level parallelism
  - Feedback Loop
- C-like syntax
- Explicit input-output size definitions for buffering constraints

```
//filter construct
float->float filter Downsample(int N) {
    // downsample by a factor of N
    work pop N push 1 peek 1 {
        int i;
        push(pop()); //output a single input value
        //consume the remaining (N-1) input values
        for(i=0;i<(N-1);i++)
            pop();
    }
}
```

Example StreamIt code for a downsampler "filter".

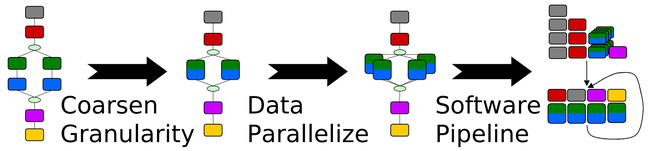


Basic StreamIt constructs

Eric Battenberg, eric@cnmat.berkeley.edu
Mark Murphy, mjmurphy@eecs.berkeley.edu

The StreamIt Compiler

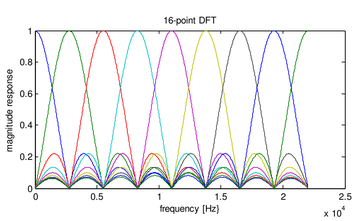
- 1. Fuse Stateless filters
  - Eliminates communication and buffer copies
- 2. Data-Parallelize
  - Allows for concurrent execution of future work
- 3. Pipeline
  - Optimal partitioning of the work using dynamic proprramming

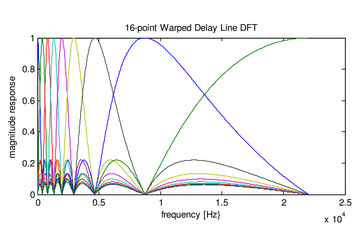


- This scheme achieves close to 14x speedup on the 16 cores of MIT's Raw architecture for common signal processing tasks [2]

Audio Feature Extraction

- **Frequency-warped spectrum** [3,4]
  - •Human ear has better spectral resolution at lower frequencies
  - •Sufficient spectral resolution achieved with 1024 point FFT (~23ms window).
  - •With frequency warping, only need 32 point window (~1ms window)
- **Allows finer time resolution**
  - •Improves analysis of rhythmic patterns and fast transients
- **<u>Comes with a significant performance cost</u>**
  - •Warping is achieved using an IIR all-pass chain
  - •Over 10x slower than MFCC extraction in Matlab





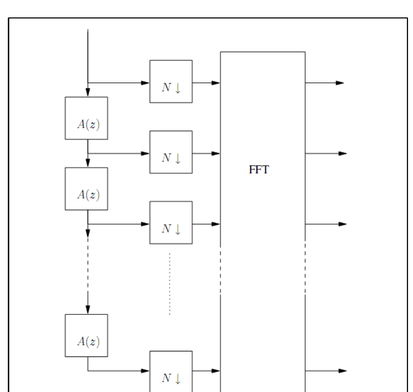


Figure 3: FFT filterbank using delay elements replaced with all-pass frequency-selective delays ($A(z)$). The boxes containing "$N\downarrow$" are Nth-order downsamplers.

$$A(z) = \frac{z^{-1} - \lambda}{1 - \lambda z^{-1}} \qquad \lambda \approx 0.7565$$

Results

- **Warped spectrum code tested on Clovertown, Opteron, Niagara2, Core 2 Duo Penryn**
  - Both fine-grained and coarse-grained implementations
- **Almost non-existent speedup beyond two cores on Clovertown and Opteron.** [Fig 3]
  - Portability problems with Java VM on Niagara2
- Moderate speedup for coarse-grained version on 2 Penryn cores. [Fig 1]
- No speedup for fine-grained version on Penryn.
- **Single core fine-grained performance was best considering all implementations and any number of cores.** [Fig 2]

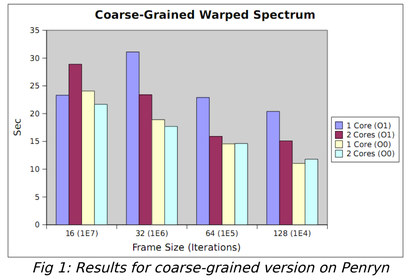


Fig 1: Results for coarse-grained version on Penryn

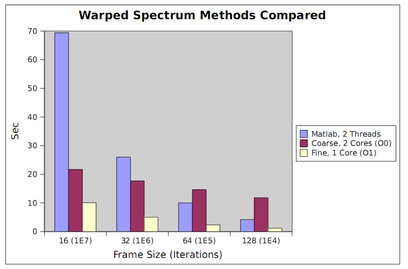


Fig 2: Results for various implementations on Penryn

Where's the Performance?

- Best performance seen in single core.
- Where is the speedup on x86 architectures?
- It seems that the StreamIt compiler is tailored to the Raw architecture
- Does it overlook:
  - cache hierarchy,
  - communication costs,
  - autotuning?

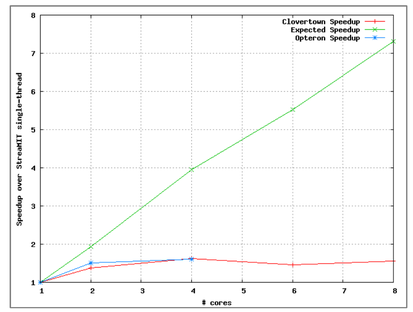


Fig 3: Speedup for coarse-grained version on Clovertown/Opteron

Conclusions

- Dataflow languages like StreamIt can significantly increase programmer productivity for audio applications.
- The StreamIt compiler achieves good uniprocessor performance with relatively little programmer effort.
- Multicore performance is severely lacking for x86 architectures.
- Different strategies need to be employed to bring StreamIt up to speed on more widespread architectures.

References

1) W. Thies, M. Karczmarek, and S. Amarasinghe. StreamIt: A Language for Streaming Applications. International Conference on Compiler Construction, 4, 2002.
2) M. Gordon, W. Thies, and S. Amarasinghe. Exploiting coarse-grained task, data, and pipeline parallelism in stream programs. Proceedings of the 12th international conference on Architectural support for programming languages and operating systems, pages 151–162, 2006
3) J. Smith III and J. Abel. Bark and ERB bilinear transforms. Speech and Audio Processing, IEEE Transactions on, 7(6):697–708, 1999.
4) C. Braccini and A. Oppenheim. Unequal bandwidth spectral analysis using digital frequency warping. Acoustics, Speech, and Signal Processing, IEEE Transactions on, 22(4):236–244, 1974.