Flavors of streams

- On-line or off-line
  - We have weak(er) linguistic power for on-line
- Functional or not
  - Fuctionality helps both optimization and mapping
- "Combinator" or not, *i.e.* allow memory reads or not
  - Differences in breadth of applicability, in practice
- Degree of generality allowed in memory references
  - The main determiner is interconnect bandwidth
  - Latency may also make some difference:
    - if parallelism is insufficient
    - if on-line latency is impacted
- Motivations differ
  - "Improved locality" is a frequent underlying theme

So what isn't streamable (well)?

- Computations with little or no parallelism
  - But even initial value problems for pretty darn stiff nonlinear ODEs can be streamed on the "right system"
- Computations with lots of "control flow"
  - But MIMD streaming is perfectly practical
- Corner turns, sparse matrix-vector multiply, and other computations with little or no temporal locality
- Computations that do fine-grain memory updates
  - The frame buffer update problem is one example
  - Functional languages can do it (slowly) with *monads*

Frame buffer operations

- These are parallelizable, and look like this:

```
buf[i][j] = blend(new,buf[i][j])
```

where the blending function is both associative and commutative.

- The typical case:

```
pixel blend(pixel p, pixel q){
    return (p.z < q.z)?
    p.infrontof(q): q.infrontof(p);}
```

- A way to parallelize this functionally is to collect all [i][j] sequences and then do a "blend reduction" on each
- Safe and live non-functional parallelization only requires that the updates of the buffer be *atomic*
  - Same work but less parallel time and space

Other interesting questions

- To what extent can an ordinary language and an extraordinary compiler result in good streaming?
- To what extent can a broad-spectrum language and an extraordinary compiler result in good streaming?
- What kinds of architectural features could augment a "conventional" architecture to help exploit streaming?
- How can we raise the level of on-line programming?
- How should we do latency *vs.* bandwidth optimization?
- How should hardware exploit stream concurrency?
  - VLIW, vectors, multithreading all can work
- How should compilers and programmers divide the work of optimization for stream computations?