Motivation: OpenMP needs a strategy for programming and exploiting current architectures
- Data-parallelism is hard to exploit on complex memory hierarchies
- Pipelining has a structuring effect on communication, which improves cache behaviour
- Scheduling fine-grained tasks is often less efficient than synchronizing persistent tasks

We compare the optimized fine-grained task scheduling in Cilk with persistent streaming tasks implemented with Erbium.

The synchronization algorithm is lock-free, uses no hardware atomic operations or fences and is optimized for minimizing cache traffic.

We use the exploration kernel with only one multiply-acc add per transaction to show the overhead incurred in the runtime.

Language extension:
- add input and output clauses for OpenMP3.0 task constructs
- make pipelined tasks persistent
- preserve the semantics
- improve performance

Detailed example: FFT streamization

Streaming runtime: Erbium
- Multi-Producer - Multi-Consumer streams
- Connect multiple read/write views to a stream using the >> connector

Detailed example: FFT streamization

Streaming runtime: Erbium
- Multi-Producer - Multi-Consumer streams
- Connect multiple read/write views to a stream using the >> connector

Taskgraph of the streamized FFT

FFT data-flow graph

Dynamic reorder pipeline

Dynamic DFT pipeline

- Pipelined FFT allows wavefront parallelization
- Data-parallelism is available in each stage (vertical slice)
- Granularity can be controlled by the number of times the data is split before applying the sequential algorithm

Target 1: 4 socket AMD quad-core Opteron 8380 (Shanghai) with 16 cores at 2.5GHz
- 64GB of memory, 64KB per core L1, 512KB per core L2, 6MB per chip L3

Target 2: 4 socket Intel hexa-core Xeon E7450 (Dunnington) with 24 cores at 2.4GHz
- 64GB of memory, 32KB per core L1, 3MB per two cores L2, 12MB per chip L3