# Lazy reference counting for the Microgrid

Raphael 'kena' Poss
University of Amsterdam, the Netherlands

INTERACT Workshop, New Orleans, Feb 25th 2012
Co-located with HPCA 2012

# Automatic garbage collection

- Common in **productivity** languages
- **Deferred** garbage collection
  - e.g. mark-and-sweep
  - may be asynchronous and/or incremental
- **Non-deferred** garbage collection
  - i.e. **reference counting**
  - **needed to decide reuse of large, shared data**
  - on-chip RC usually based on **memory atomics**

# Example use of reference counting

Array comprehension in Single-Assignment C:

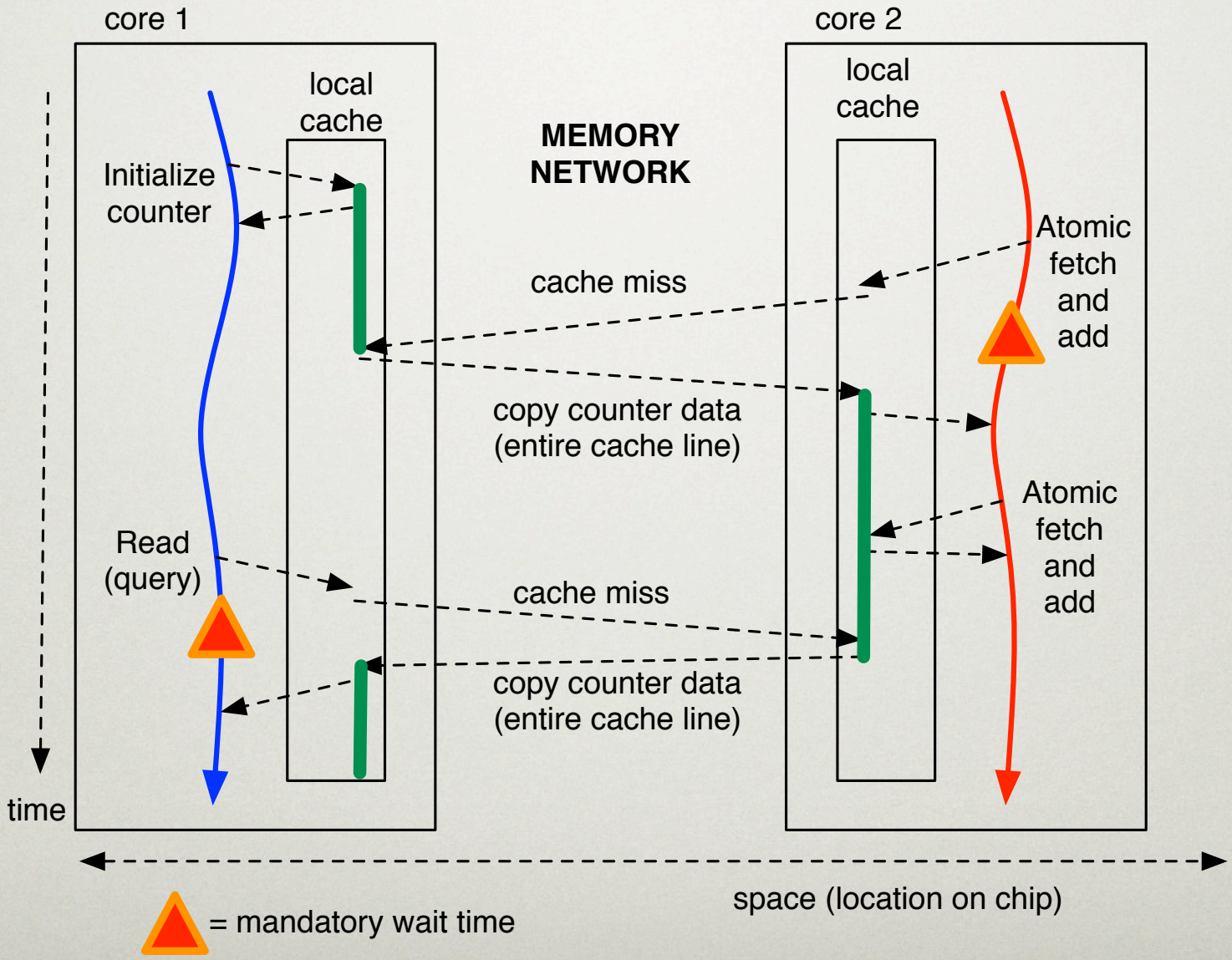
```
B = with {  ([3,0] <= iv < [5,4])  :  iv[0] + iv[1];
            ([0,4] <= iv < [5,6])  :  A[iv];    }: genarray( [5,6],  0)
```

$$B \leftarrow \begin{pmatrix} 0 & 0 & 0 & 0 & A[0,4] & A[0,5] \\ 0 & 0 & 0 & 0 & A[1,4] & A[1,5] \\ 0 & 0 & 0 & 0 & A[2,4] & A[2,5] \\ 3 & 4 & 5 & 6 & A[3,4] & A[3,5] \\ 4 & 5 & 6 & 7 & A[4,4] & A[4,5] \end{pmatrix}$$

If RC(A)>1, must allocate and copy space for B

If RC(A)=1, can reuse the space - saves 33% time

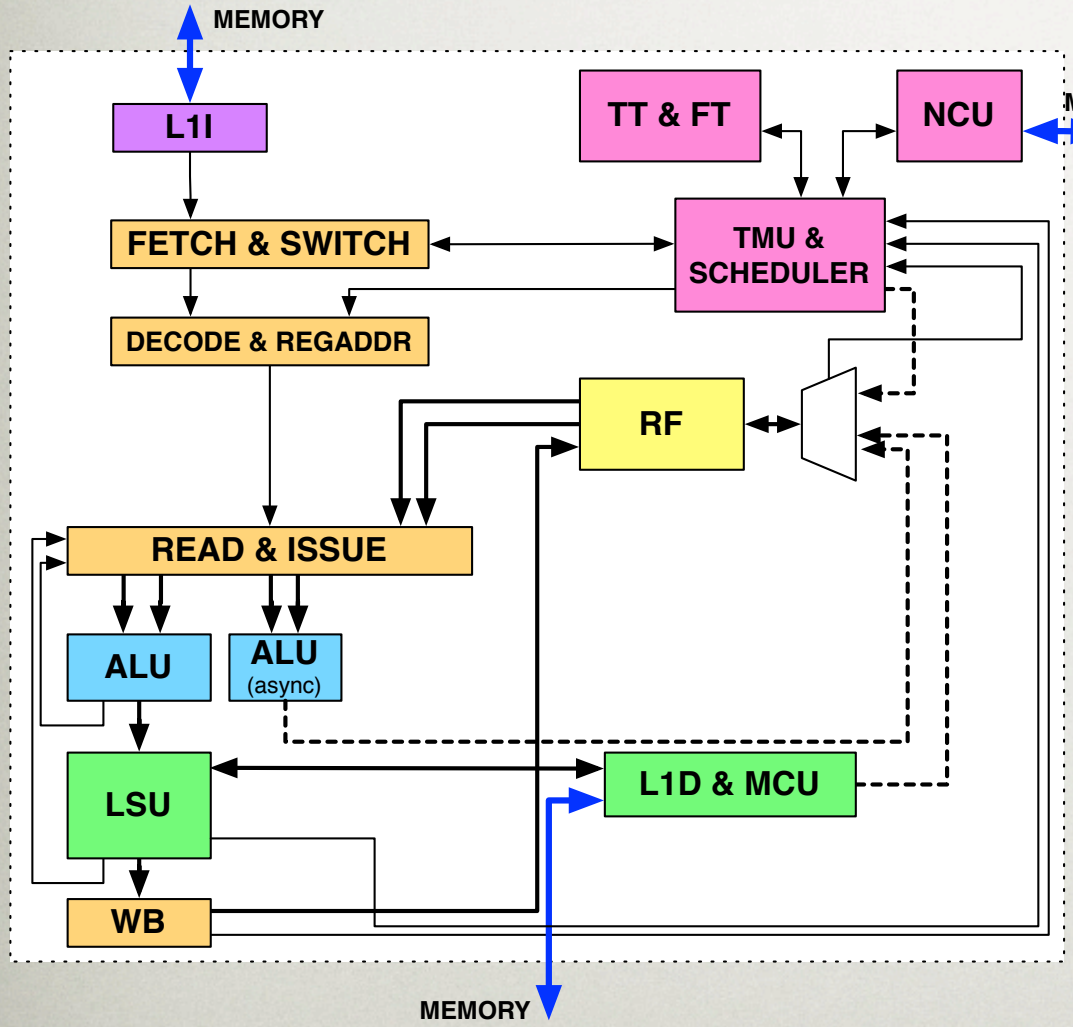
# Reference counting using memory atomics



# Many-core chips

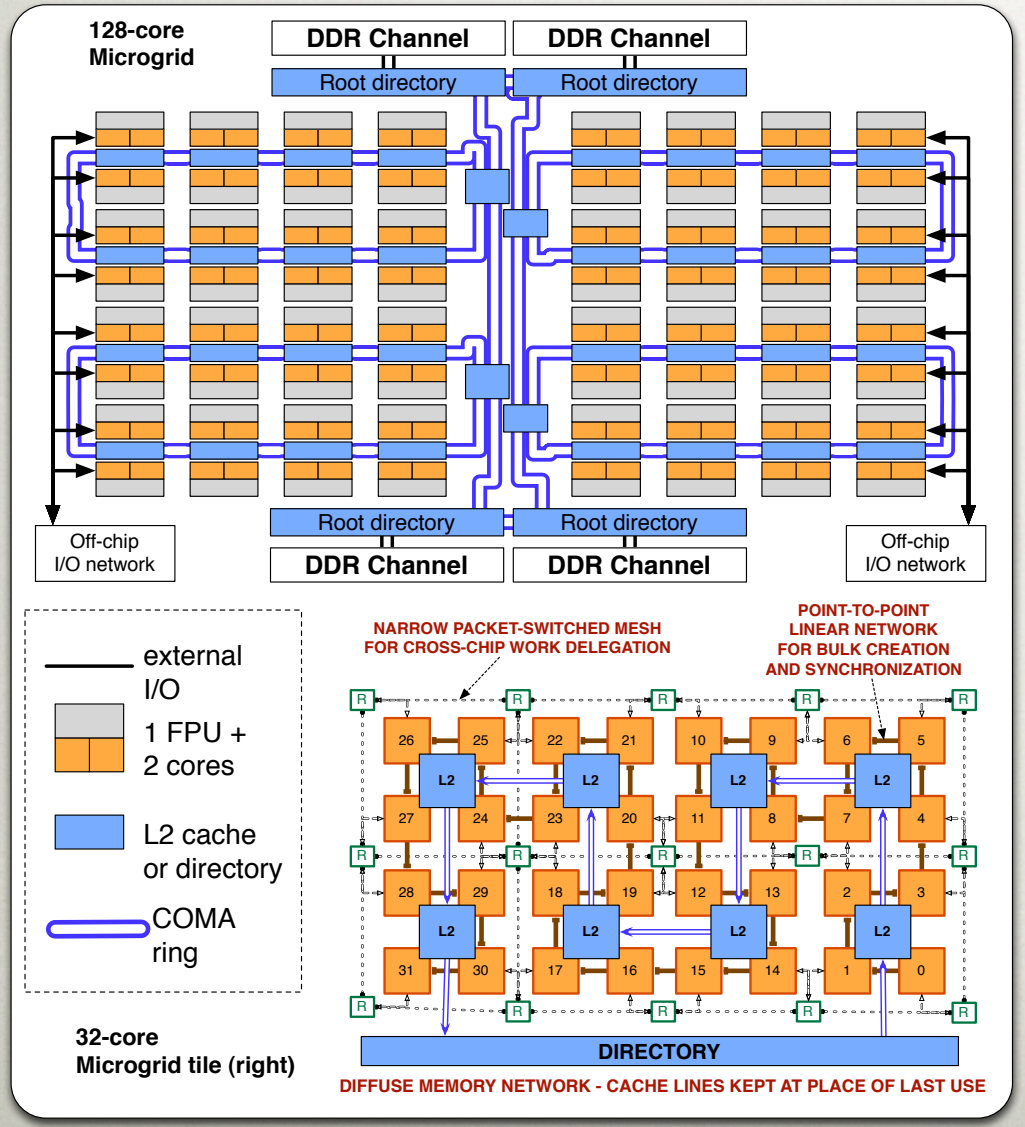
- Today: 10s of cores, tomorrow: 100+, 1000+?
- **Global cache coherency is expensive**
- Future chips will probably feature
  - Distributed memories
  - Explicit cache control
  - **Weakly coherent cache networks**
- Cross-chip memory atomics will be expensive, may not even be available!

# Microgrids of D-RISC cores



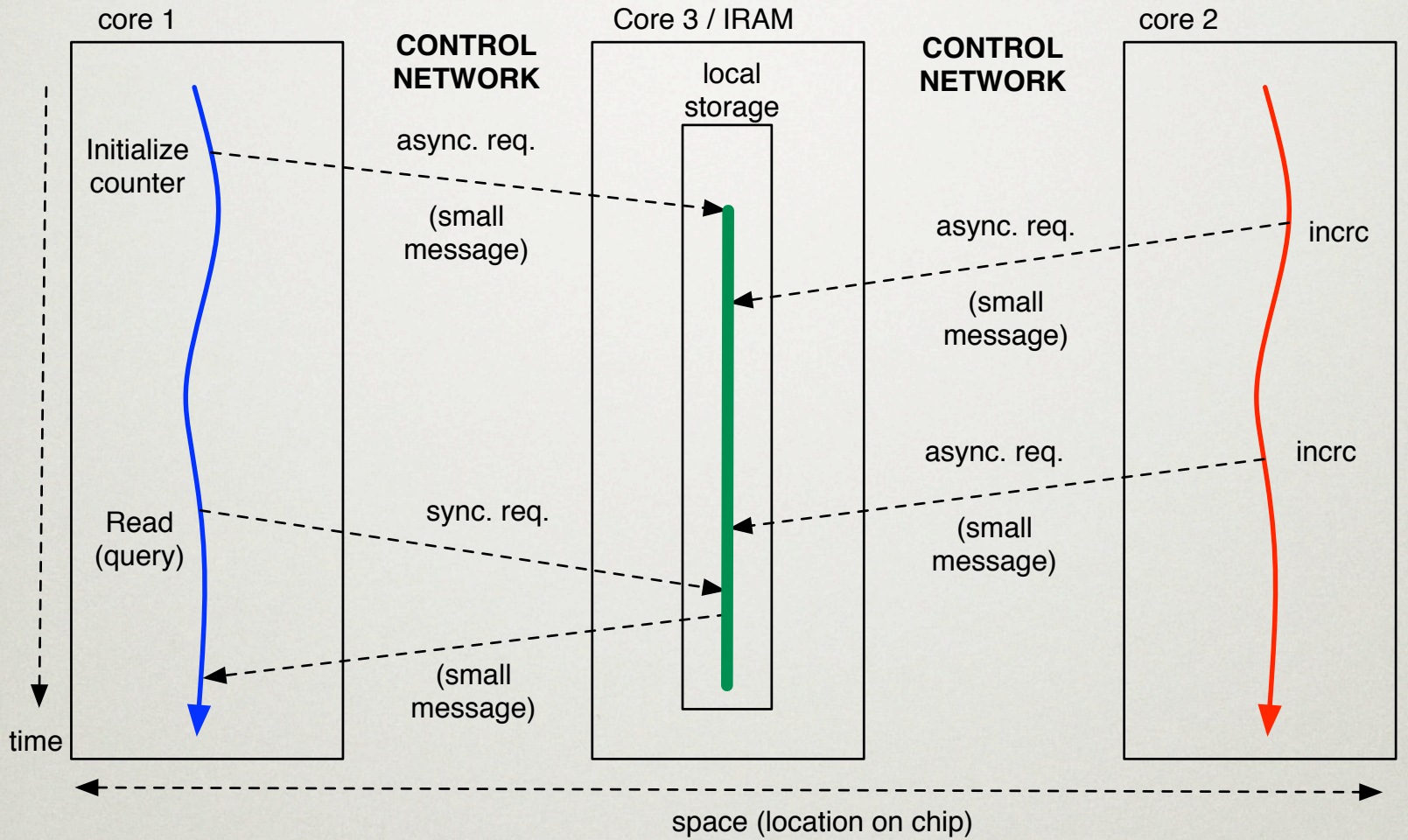

- D-RISC cores: hardware multithreading + dynamic dataflow scheduling
  - fine-grained threads: 0-cycle thread switching, <2 cycles creation overhead
  - ISA instructions for creation, synchronization and communication
  - dedicated hardware processes for asynchronous bulk creation and synchronization

# Example 128-core Microgrid




- Separate memory and control NoCs
- **Weak coherency**: cache updates only propagated upon bulk creation and termination of threads
- no support for atomics

# Lazy reference counting




**Asynchronous requests**: no wait time for incrc/decrc

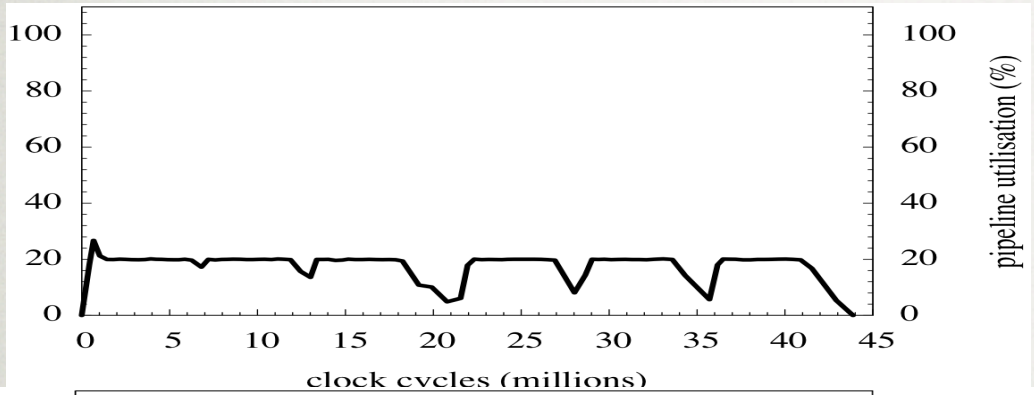
**Less bandwidth required**: requests smaller than cache line

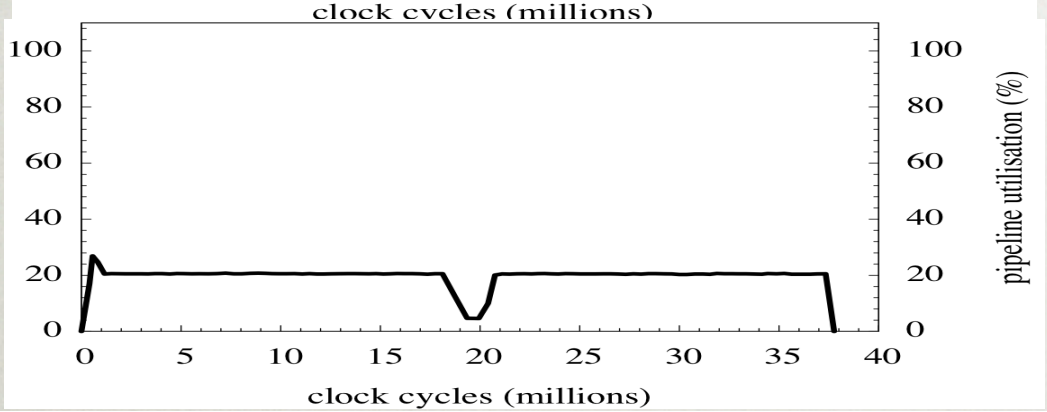
# 2D FFT (part of NAS FT)

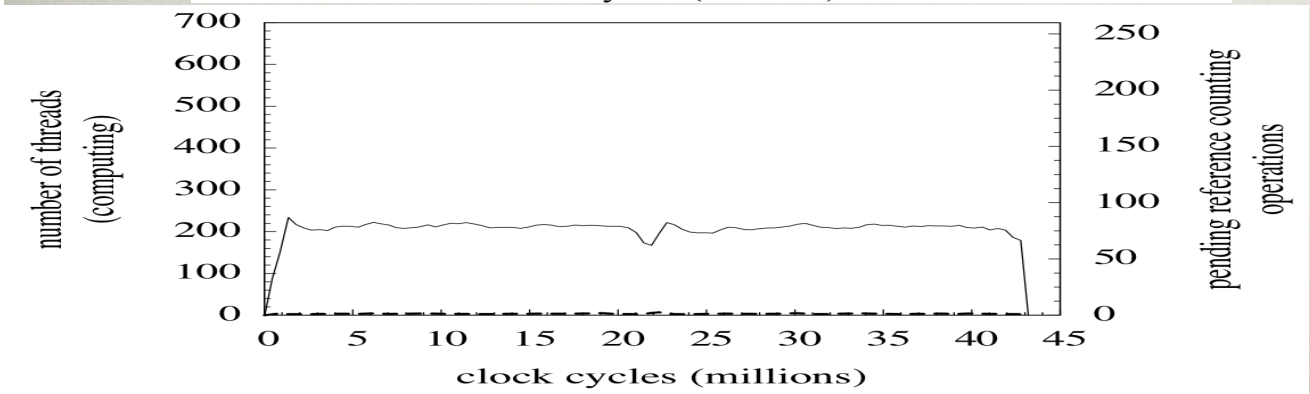
- Danielson-Lanczos algorithm: recursive decomposition of even- and odd-indexed elements.
- Implementation explained in

  Grelck, C., Scholz, S.B.: **Towards an Efficient Functional Implementation of the NAS Benchmark FT**. In: Proc. 7th International Conference on Parallel Computing Technologies (PaCT'03), Nizhni Novgorod, Russia. LNCS, vol. 2763, pp. 230–235. Springer (2003)
- Running on regions of a 128-core Microgrid, 1GHz D-RISC (64-bit) cores.

# 128 x 128 FFT
# Sequential execution



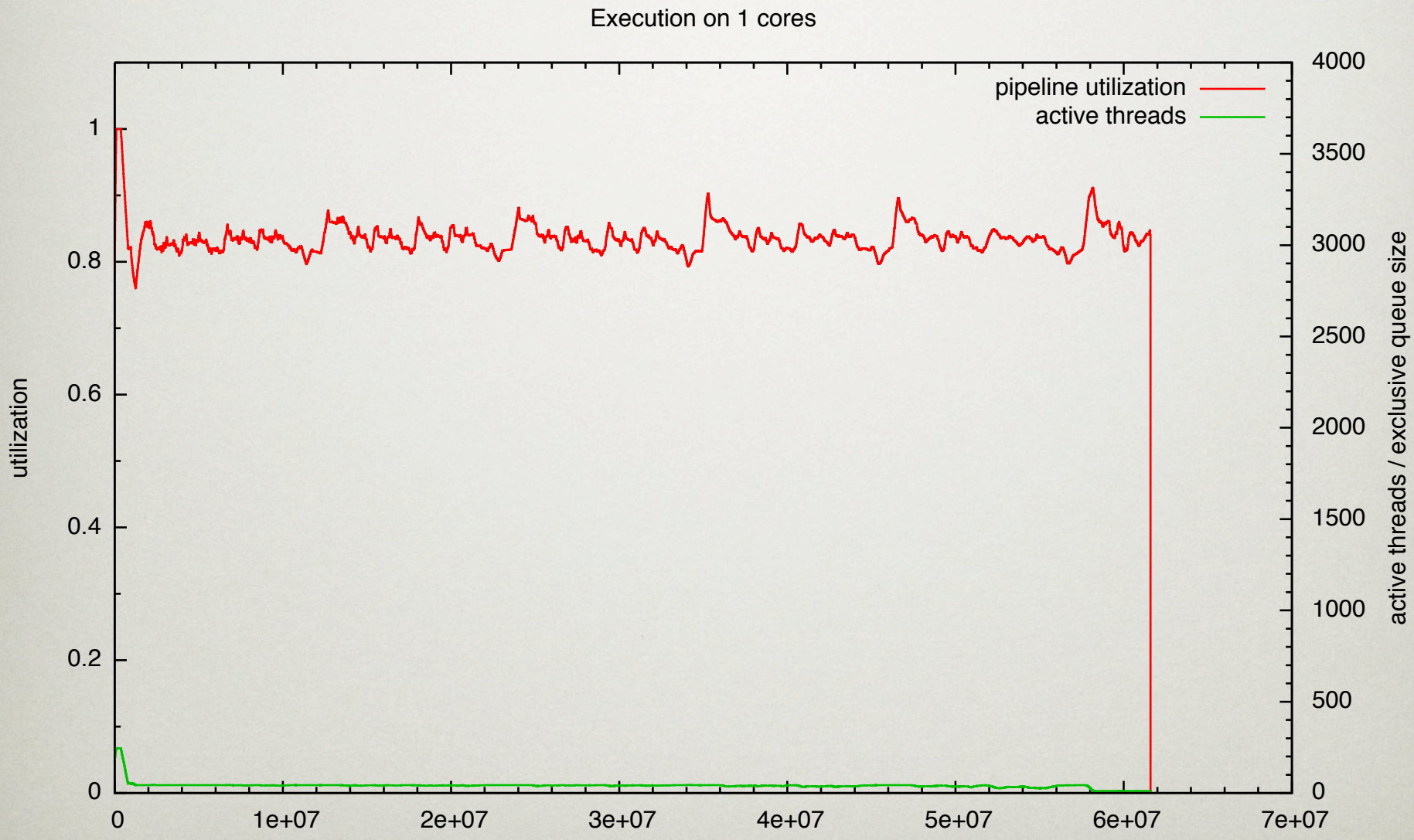

**Without** reference counting: less reuse opportunities, time wasted allocating and copying data

**With** reference counting: array data area reused when possible

**Multithreaded on 1 core**
16% overhead from thread management

# 256 x 256 FFT

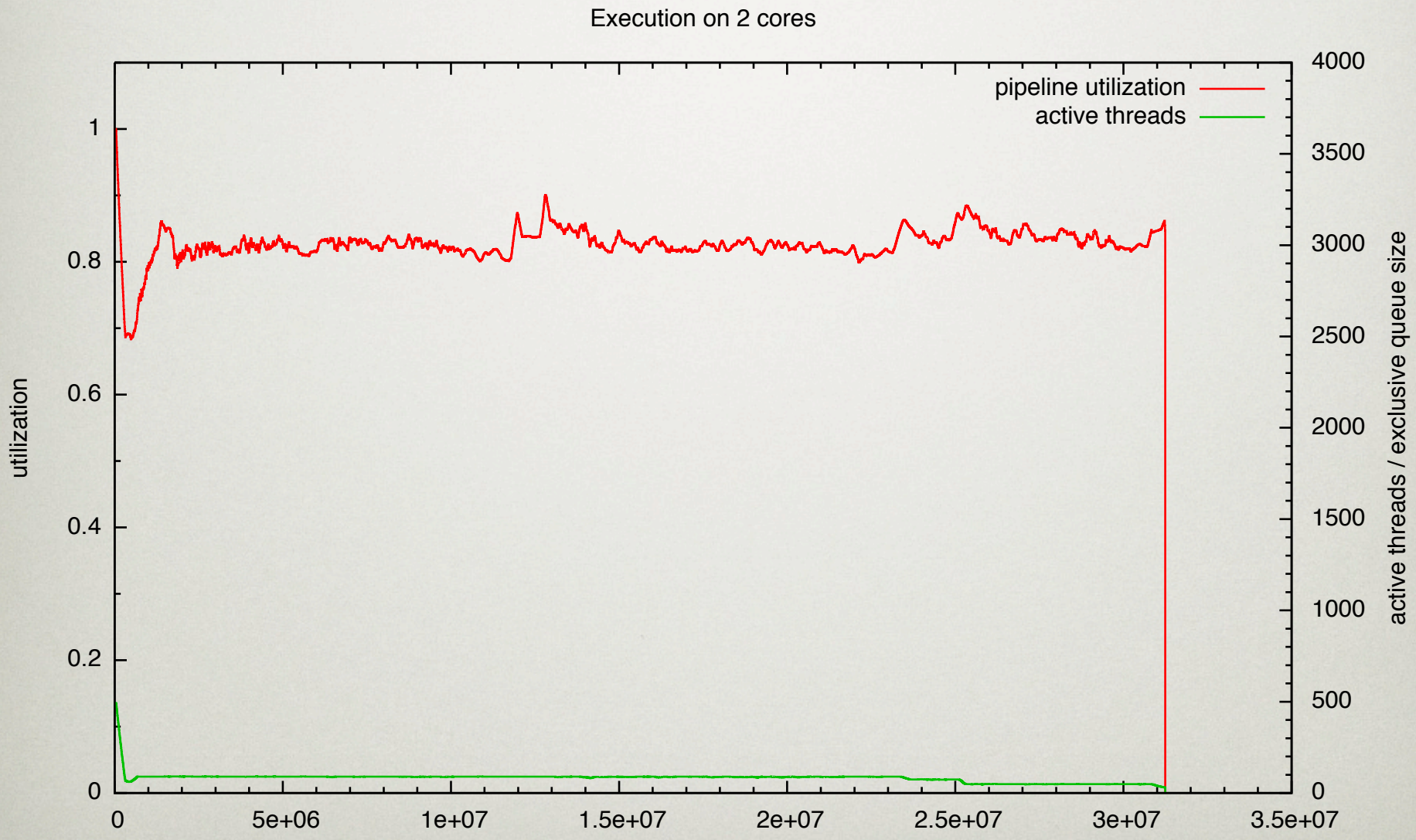
## Parallel execution




time (nanoseconds, or cycles @ 1GHz)

# 256 x 256 FFT

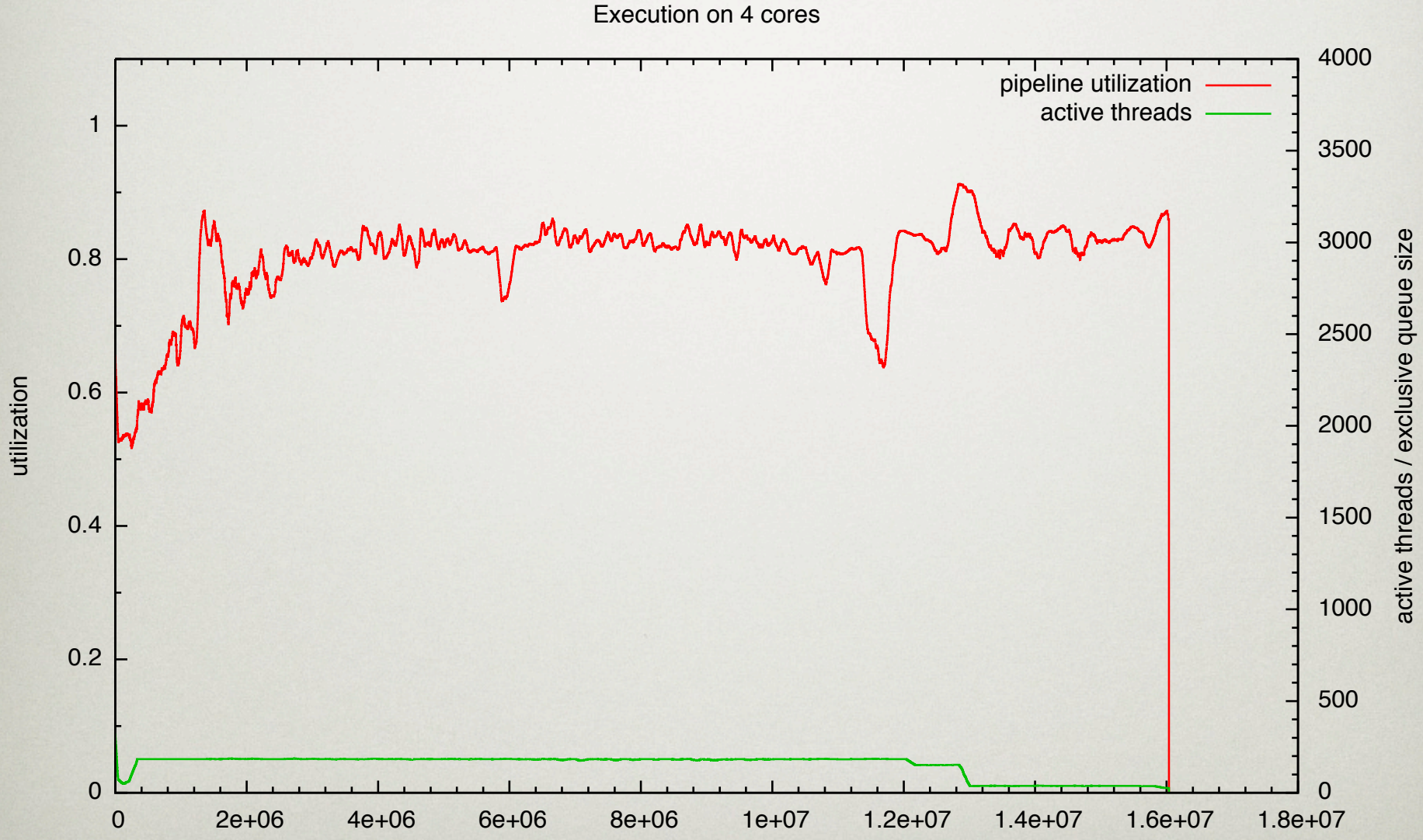
## Parallel execution




time (nanoseconds, or cycles @ 1GHz)

# 256 x 256 FFT

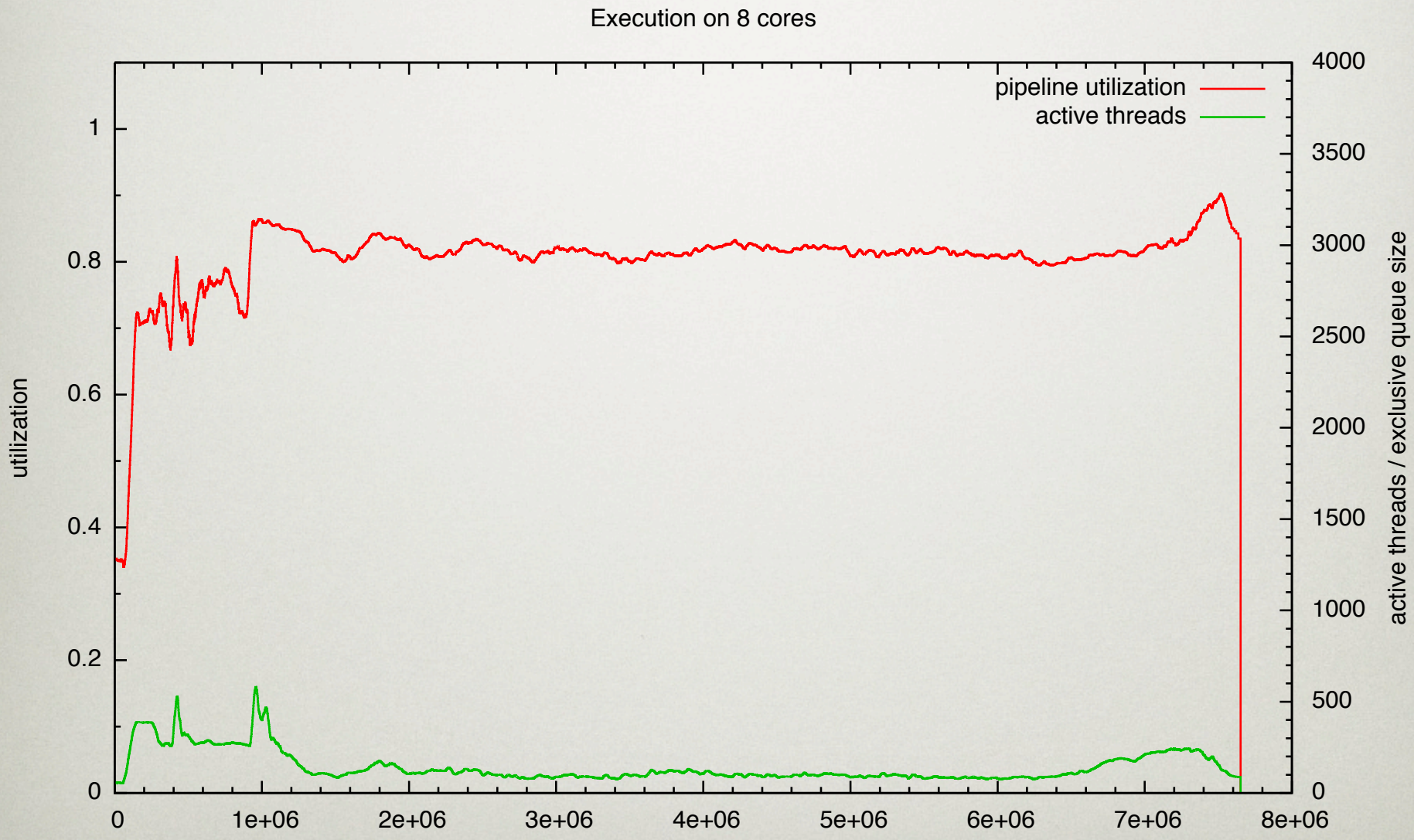
## Parallel execution




time (nanoseconds, or cycles @ 1GHz)

# 256 x 256 FFT

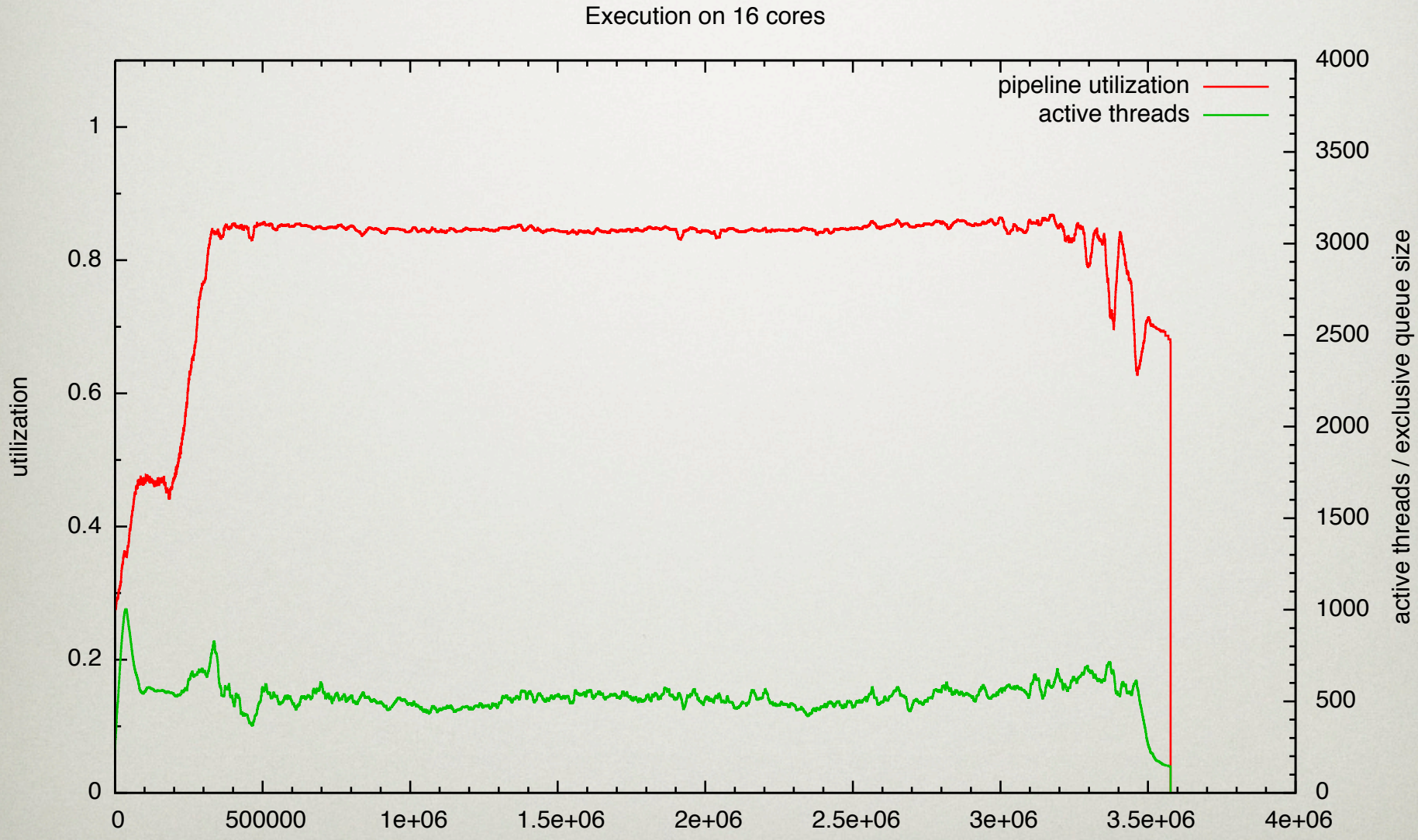
## Parallel execution




time (nanoseconds, or cycles @ 1GHz)

# 256 x 256 FFT

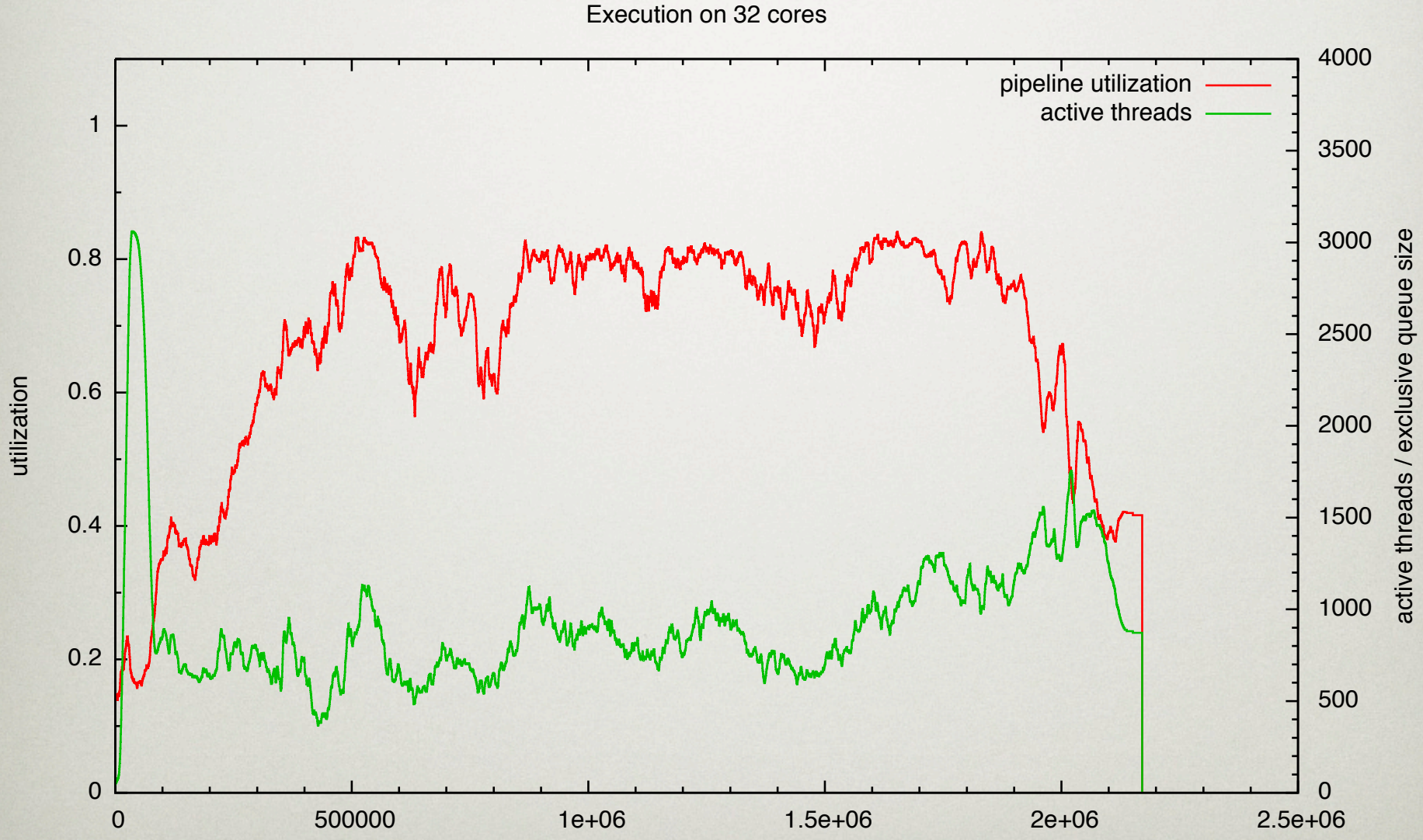
## Parallel execution




time (nanoseconds, or cycles @ 1GHz)

# 256 x 256 FFT

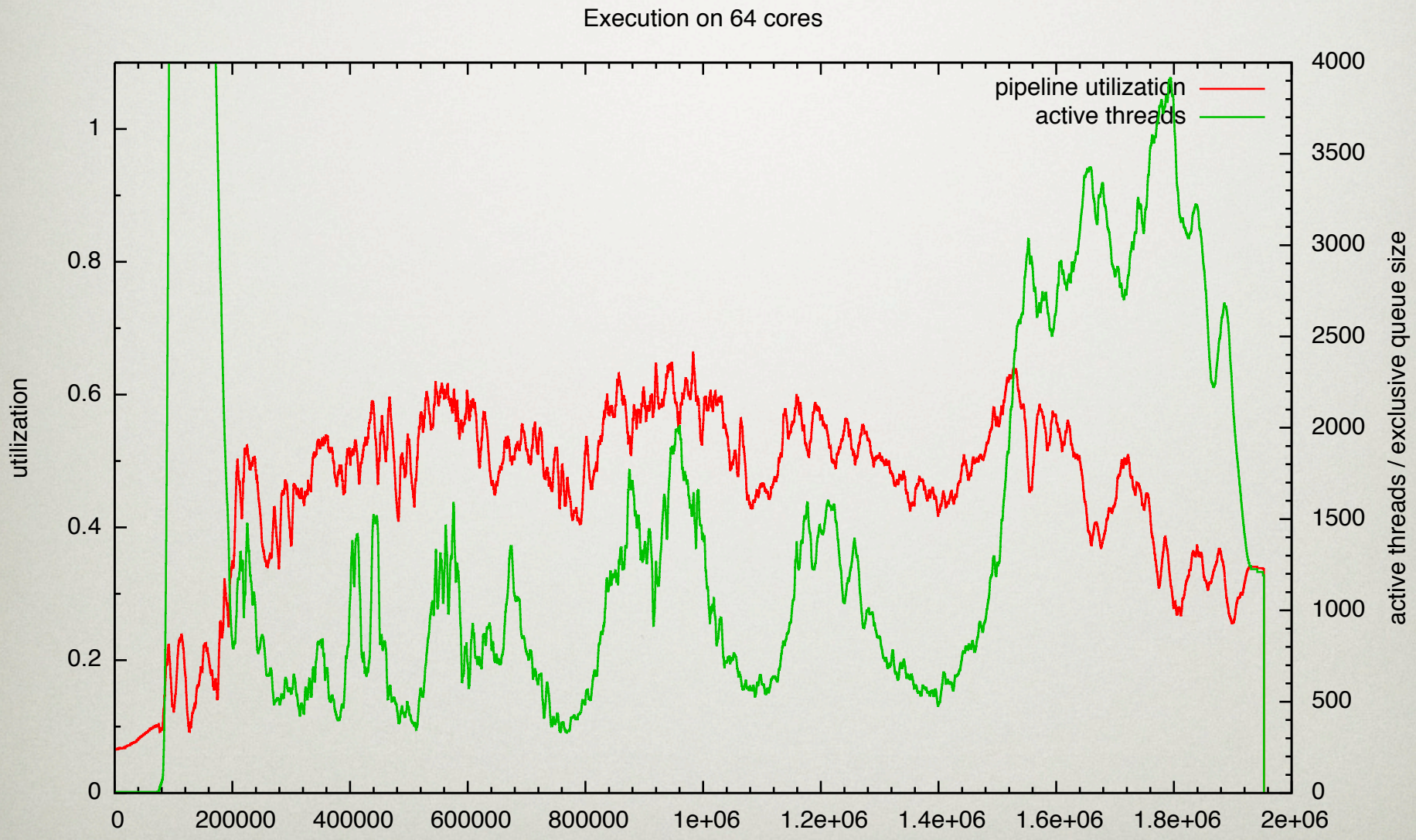
## Parallel execution




time (nanoseconds, or cycles @ 1GHz)

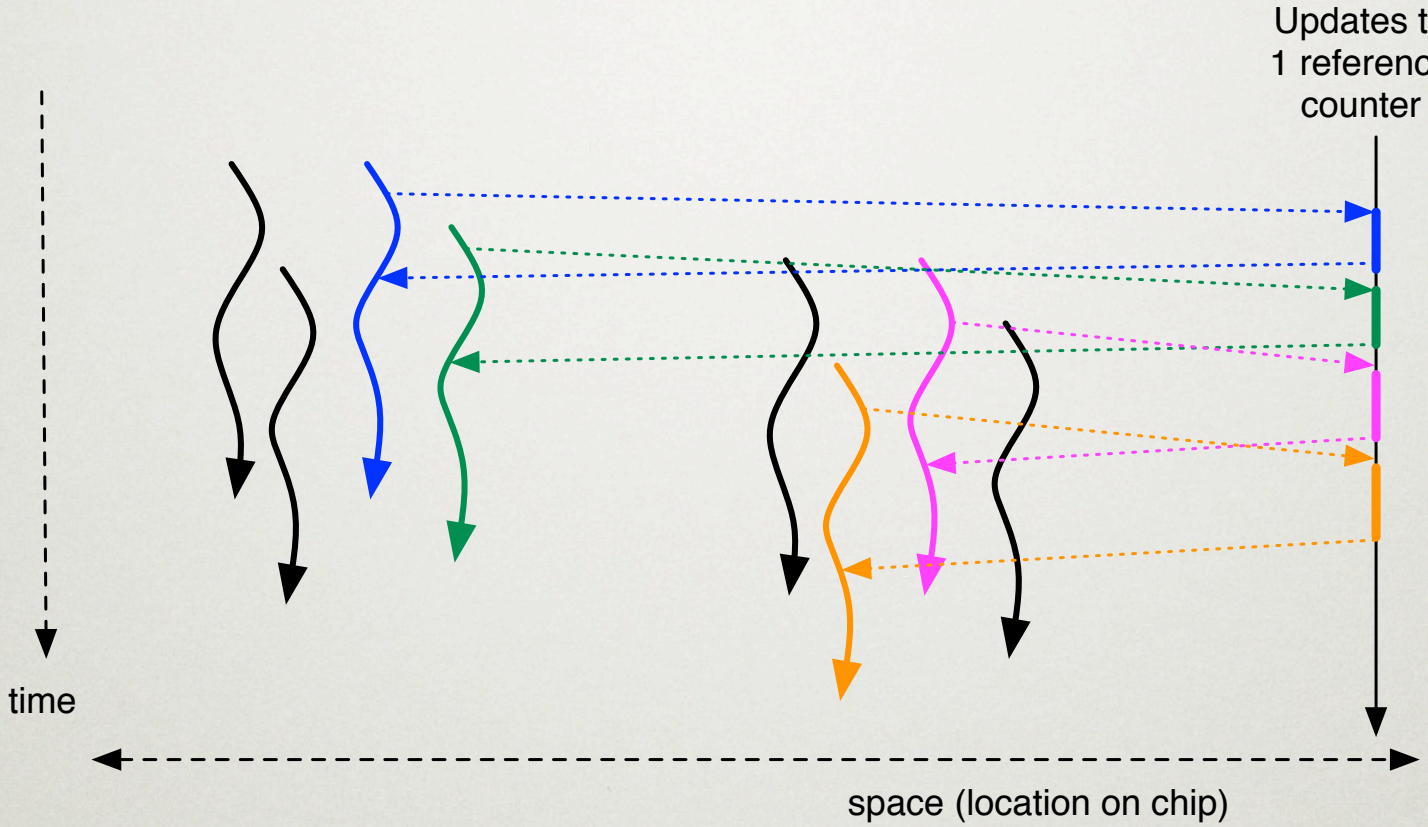
# 256 x 256 FFT

## Parallel execution




time (nanoseconds, or cycles @ 1GHz)

# Amdahl's law at the reference counters




Overall scalability constrained by
max throughput of RC operations at counter location

# Feedback to the hardware architect

- **Network ordering**: must preserve relative order from issue thread to target counter location
  - OK with X/Y routing
- **Latency**: performance advantage only if latency of an asynchronous creation ≤ latency of cache line migration
  - OK with separate memory/control NoCs

# Thank you!