GPU Programming

Higher-level parallelism

Christian Lessig

Motivation

- Correctness / determinism
- Composability
- Scalability
- Portability
- Maintainability

Motivation

- Correctness / determinism
- Composability
- Scalability
- Portability
- Maintainability

Difficult to ensure using low level threads

Fork-join model

- Computations are decomposed into *tasks*.

Fork-join model

- Computations are decomposed into *tasks*.
  - Makes parallelism explicit but does not enforce it.
  - Classical programming languages encode serialization although it might not be necessary. Task avoid this unnecessary serialization.
  - "Smart" scheduler can ensure efficient execution (more or less) independent of hardware.

Fork-join model

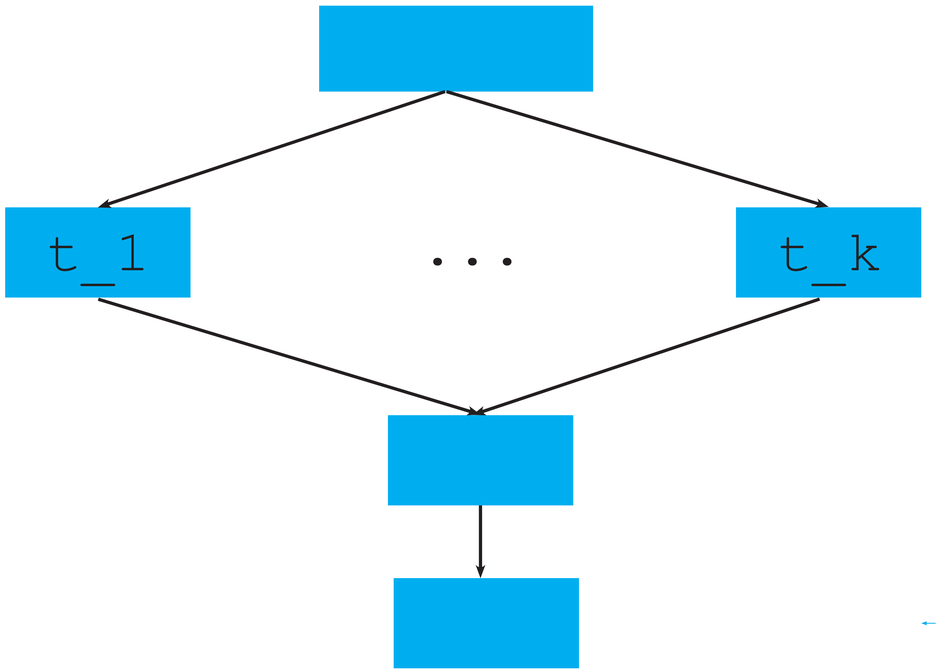
- Can be represented as (directed, acyclic) graph:

$$\|x\| = \left(\sum_{i=1}^{n} x_i \right)^{1/2}$$

Fork-join model

- Can be represented as (directed, acyclic) graph:

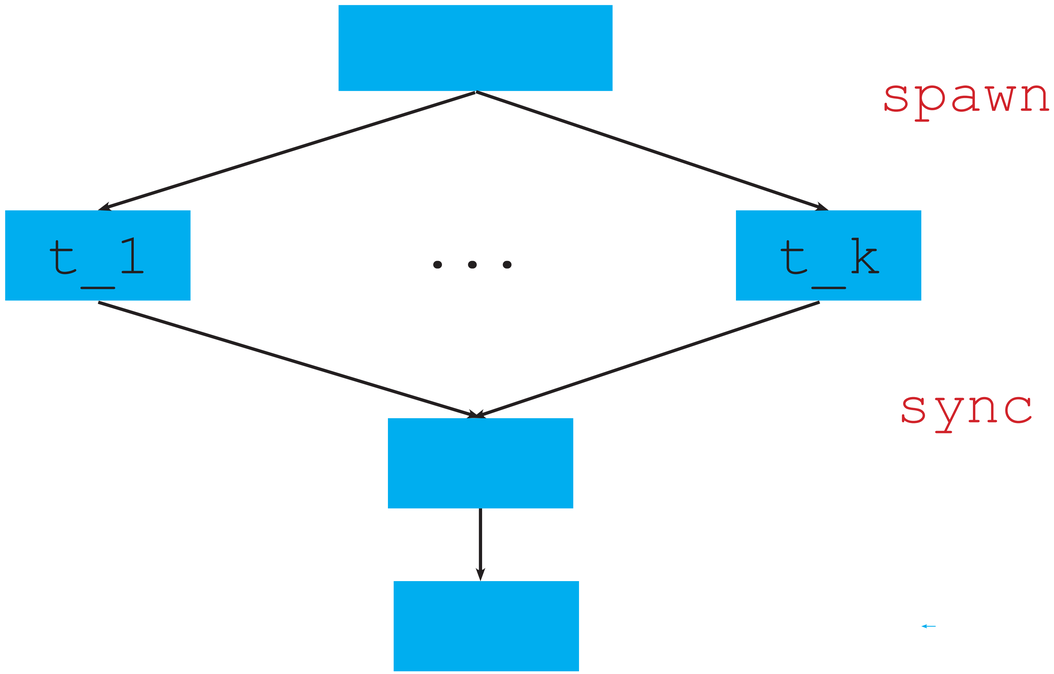
$$\|x\| = \left(\sum_{i=1}^{n} x_i \right)^{1/2}$$



Fork-join model

- Can be represented as (directed, acyclic) graph:

$$\|x\| = \left(\sum_{i=1}^{n} x_i \right)^{1/2}$$



Fork-join model

- Example:

```
int norm( T* a, int n) {

  int sum;
  for( tid = 0, tid < k, ++k) {
    spawn normWorker( tid, a, n, sum);
  }

  sync;
  normT = sqrt( sum);
}
```

Futures / promises

- Communication is explicit, parallel execution implicit

Futures / promises

- Communication is explicit, parallel execution implicit

```
std::future<int> p = std::async( std::launch::async,
                                 computeLargePrime);

// do some other work
...

res = encodeMessage( p.get());
```

Futures / promises

- Communication is explicit, parallel execution implicit

```
std::promise<int> channel;
std::future<int> fut = channel.get_future();

// t2: put value in channel
channel.set_value( comp);

// t1: pick up value
int val = channel.get();
```

OpenMP

- Cross-industry pragma extension of C/C++

MPI

- Standard API for message passing
- Example program:

Container-based parallelism

- Data-parallel processing of data in pre-defined containers
  - Functor applied to every data element with well defined side effects
- `parallel_for` is variation with similar concept

Container-based parallelism

- In next C++ standard:

```
std::sort( std::execution::par, v.begin(), v.end());

std::for_each( std::execution::par,
               v.begin(), v.end(), func);
```

- Intel thread building blocks
- Various other libraries

Higher-level parallelism

- Various approaches
- Not composable
- Library or specific language?

Further reading

- M. D. McCool, J. Reinders, and A. Robison, *Structured parallel programming: patterns for efficient computation*. Elsevier/Morgan Kaufmann, 2012.
- T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. The MIT Press, 2001.
- E. A. Lee, *The Problem with Threads*, Computer, vol. 39, no. 5, pp. 33–42, May 2006.