

## GPU Microarchitecture Note Set 1a—Parallelism

### Parallel Computation Terminology

### Parallel Computation Idea:

One computer takes  $t$  seconds to run a program, which is not fast enough ...  
... so try to use  $c$  computers to get the program to run in  $t/c$  seconds ...  
... choose  $c$  to fit your performance goal and budget.

**Easier said than done.**

### Example:

Suppose 1 computer takes 1 hour to run program  $A$ .

Convert  $A$  to a parallel program,  $A_p$ .

For  $c = 2$ , we hope that  $A_p$  will run in  $\frac{1}{2}$  hour on a system that costs twice as much.

For  $c = 60$ , we hope that  $A_p$  will run in 1 minute on a system that costs 60 times as much.

For  $c = 60 \times 10^9$ , we hope that  $A_p$  will run in 1 nanosecond on a system that costs  $c$  times as much.

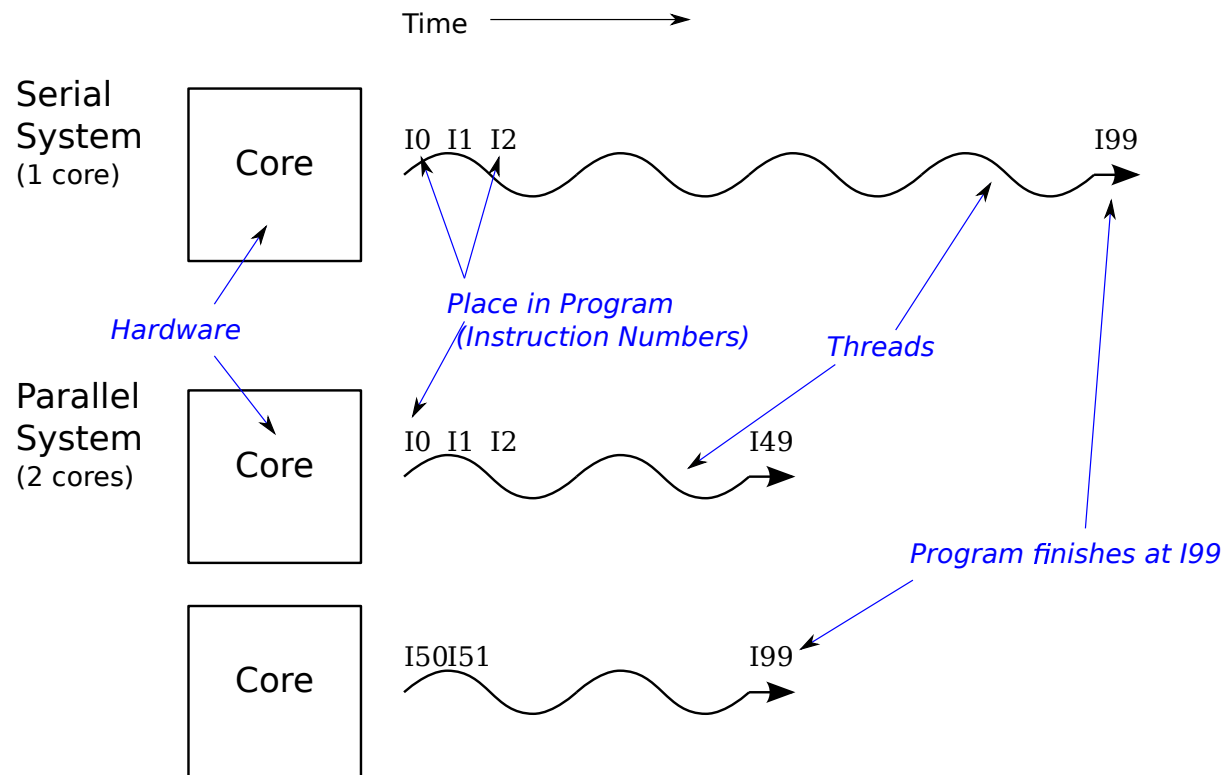
## Parallel Computation:

The use of multiple processor cores to speed the execution of a program.

A *parallel program* consists of multiple *threads* that will execute on a *parallel system* consisting multiple *cores*.

The goal is to lower execution time by using multiple cores.

Realizing this goal is often *frustrated* by the *difficulty* of parallel programming.



## Definitions

### *Thread:*

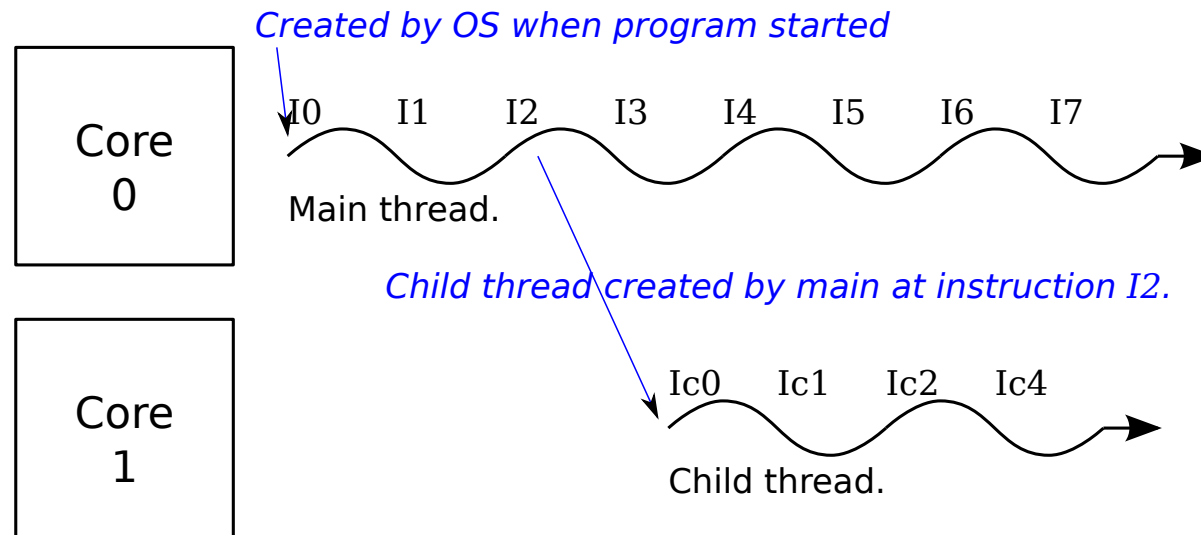
A path through the program defined by the programmer, compiler, or some piece of support software.

The first program you wrote probably consisted of a single thread.

Programs start with a single thread ...

... and can create additional threads as needed.

A program with multiple threads is a parallel program.



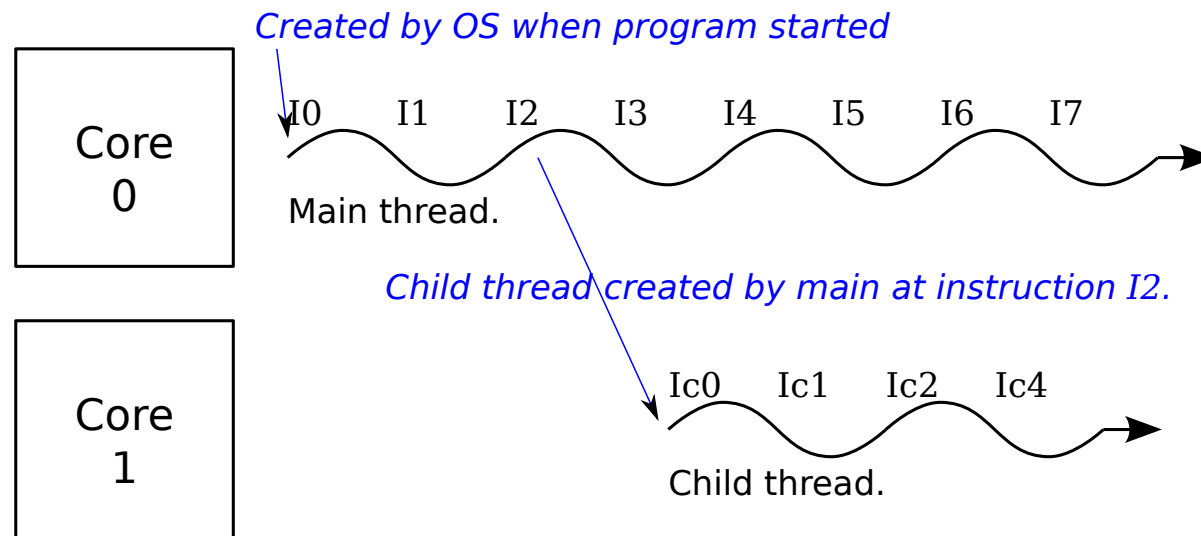
## Thread Spawn Example

Pseudocode with *ad-hoc* instruction labels:

```
void main()
  I0: a = 1;
  I1: b = 9;
  I2: thread_create(my_child);
  I3: c = a + b;
  ...
```

```
void my_child()
  Ic0: x = 7;
  Ic1: y = 9;
  Ic2: z = x + y;
  ...
```

Execution timing:



*Core:*

Hardware needed to execute a thread.

Sometimes called a CPU (central processing unit).

A core has:

Hardware to *fetch* instructions.

*Functional units* to perform arithmetic operations.

*Register files* to hold intermediate (working, temporary) data values.

Hardware to *decode* and orchestrate instruction execution.

## Execution of Multithreaded Programs

Consider a system with  $c$  cores and a program with  $r$  threads.

Typically the OS will distribute the  $r$  threads evenly over the  $c$  cores.

If  $c < r$  then  $c - r$  cores will sit idle.

If  $c > r$  then a core may have more than one thread assigned.

## Computation of Speedup Parallel System

Consider

A **parallel program** that can spawn any number of threads, as needed.

A computer consisting of  **$c$  cores**.

Let  $t(1)$  denote **the execution time on 1 core**.

Its value is determined by the single-thread performance of the core.

Let  $t(c)$  denote **the execution time on  $c$  cores**.

Its value is determined by the parallel program and by  $t(1)$ .

*Speedup:*

[of a parallel program on parallel system]. The ratio of execution time on one core to the time on the entire system.

Using the notation above:

$$S(c) = \frac{t(1)}{t(c)}.$$

For example:

*A program runs in 10 s on one core and 3 s on 5 cores.*

The speedup is then  $S(5) = \frac{10\text{s}}{3\text{s}} = 3.33$ .

## Speedup Special Cases

Speedup Case: *Linear Speedup*—  $S(c) = c$ .

This occurs when  $t(c) = t(1)/c$ .

This indicates no duplication of effort by threads, no time lost to communication.

There are some programs with linear speedup...  
... but for many others the speedup is lower.

### Example:

*A program runs in 10 s on one core and is to be run on 5 cores. What would its run time be if it achieves linear speedup?*

To achieve linear speedup it would need to run in  $10 \text{ s} / 5 = 2 \text{ s}$ .

## Speedup Special Cases

Speedup Case: *No Speedup*—  $S(c) = 1$ .

This occurs when  $t(c) = t(1)$ .

This might be the programmer's fault ...  
... or an inherent property of the problem.

## Speedup Special Cases

Speedup Case: *Serial Limiter*—  $S(c) = \frac{1}{f + (1-f)/c}$

This is sometimes referred to as *Amdahl's Law*.

Cannot parallelize  $f \in [0, 1]$  of program,  $f$  is *serial* portion.

*E.g.*, for  $f = 0.2$ , can't parallelize 20% of program.

This applies to a program that can be split into two parts...

... a part with linear speedup...

... and a part with no speedup (the *serial* portion).

$1 - f$  is the fraction of the program with linear speedup.

When  $f = 0$ , all of the program enjoys linear speedup;...

... when  $f = 1$ , no part of the program can be parallelized.

## Limit of Preceding Speedup Analysis

Preceding analysis assumed only one kind of core.

In this class we will compare **different kinds of cores**.