CS3350B
Computer Architecture
Winter 2015

Lecture 7.3: Multicore TLP (2)

Marc Moreno Maza

www.csd.uwo.ca/Courses/CS3350b

[Adapted from lectures on
Computer Organization and Design,
Patterson & Hennessy, 4th or 5th edition, 2011]

Plan

- Hardware multithreading
- CilkPlus and OpenMP: simple parallel extensions to C/C++ for high-level parallel programming on multicores
- Parallel performance metrics and profiling tool *cilkview*

Multithreading on A Chip

- Find a way to “hide” true data dependency stalls, cache miss stalls, and branch stalls by finding instructions (from other process threads) that are independent of those stalling instructions
- **Hardware multithreading** – increase the utilization of resources on a chip by allowing multiple processes (threads) to share the functional units of a single processor
  - Processor must duplicate the state hardware for each thread – a separate register file, PC, instruction buffer, and store buffer for each thread
  - The caches, TLBs, BHT, BTB, RUU can be shared (although the miss rates may increase if they are not sized accordingly)
  - The memory can be shared through virtual memory mechanisms
  - Hardware must support *efficient* thread context switching

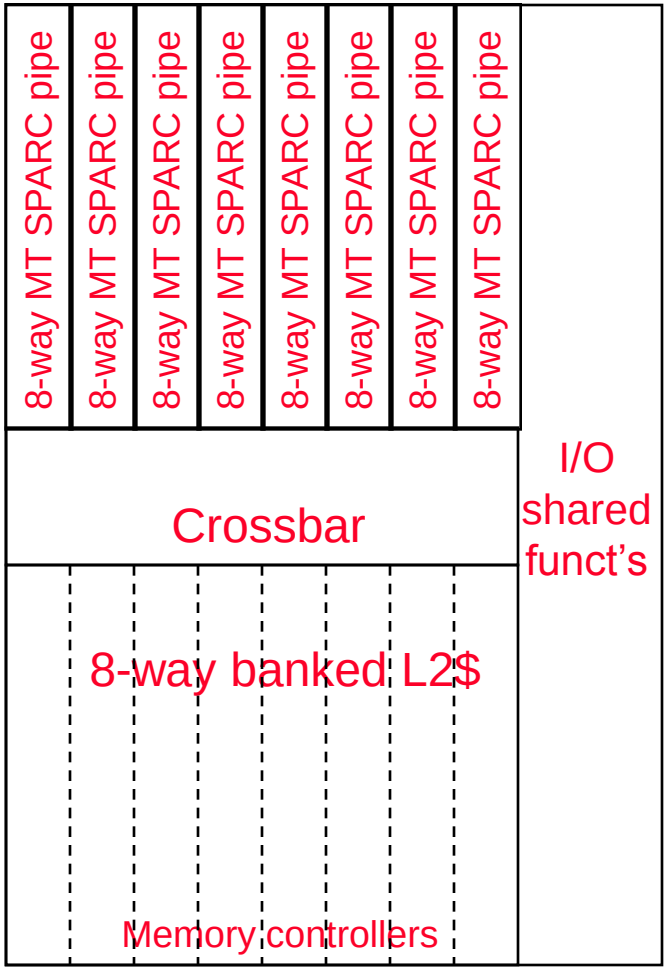
Types of Hardware Multithreading

- Fine-grain – switch threads on every instruction issue
  - Round-robin thread interleaving (skipping stalled threads)
  - Processor must be able to switch threads on every clock cycle
  - **Advantage** – can hide throughput losses that come from both short and long stalls
  - **Disadvantage** – slows down the execution of an individual thread since a thread that is ready to execute without stalls is delayed by instructions from other threads
- Coarse-grain – switches threads only on costly stalls (e.g., L2 cache misses)
  - **Advantage** – thread switching doesn't have to be essentially free and much less likely to slow down the execution of an individual thread
  - **Disadvantage** – limited, due to pipeline start-up costs, in its ability to overcome throughput loss
    - Pipeline must be flushed and refilled on thread switches

Multithreaded Example: Sun's Niagara (UltraSparc T2)

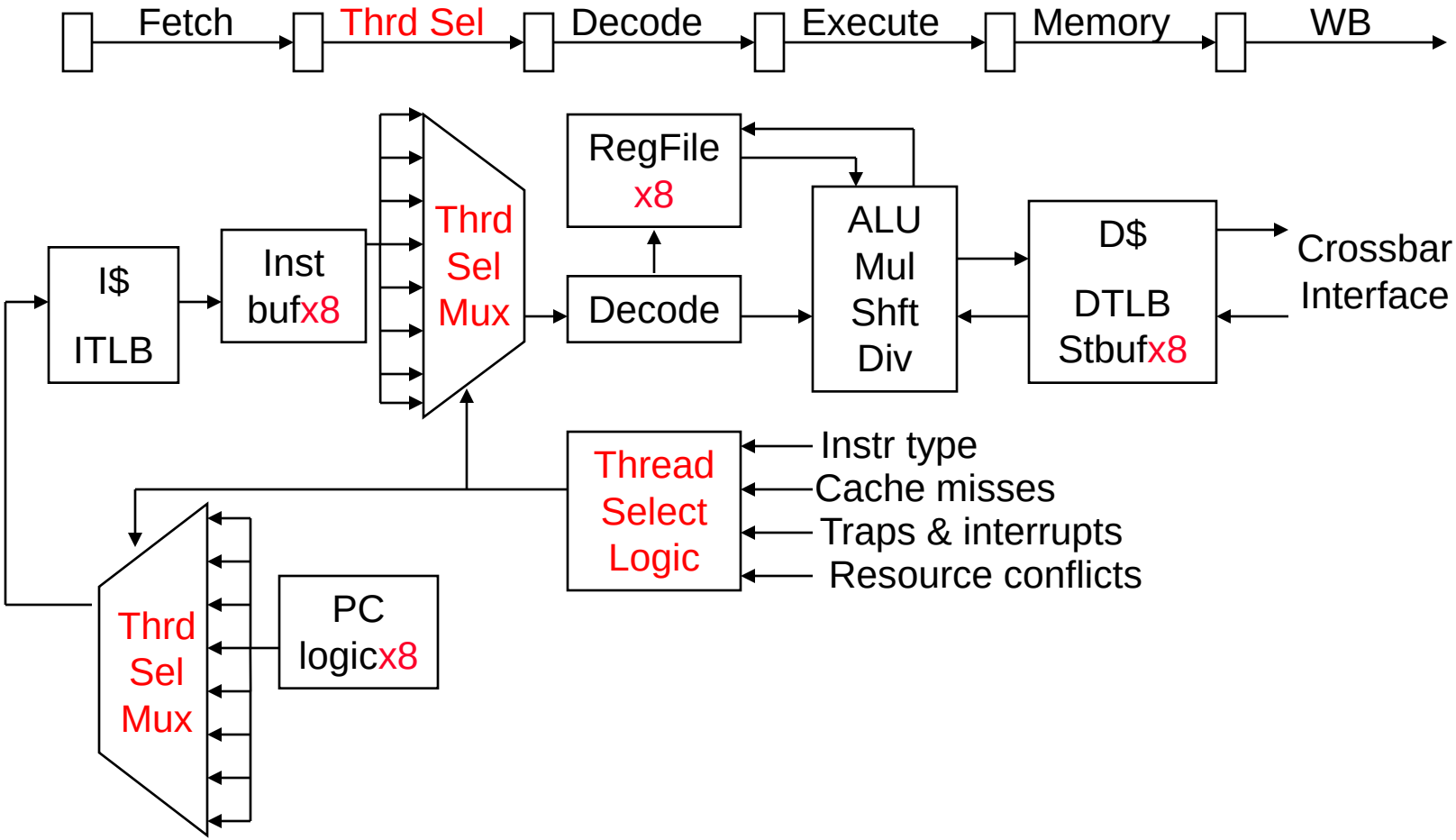
- **Eight fine grain** multithreaded single-issue, in-order cores (no speculation, no dynamic branch prediction)

	Niagara 2
Data width	64-b
Clock rate	1.4 GHz
Cache (I/D/L2)	16K/8K/4M
Issue rate	1 issue
Pipe stages	6 stages
BHT entries	None
TLB entries	64I/64D
Memory BW	60+ GB/s
Transistors	??? million
Power (max)	<95 W



Niagara Integer Pipeline

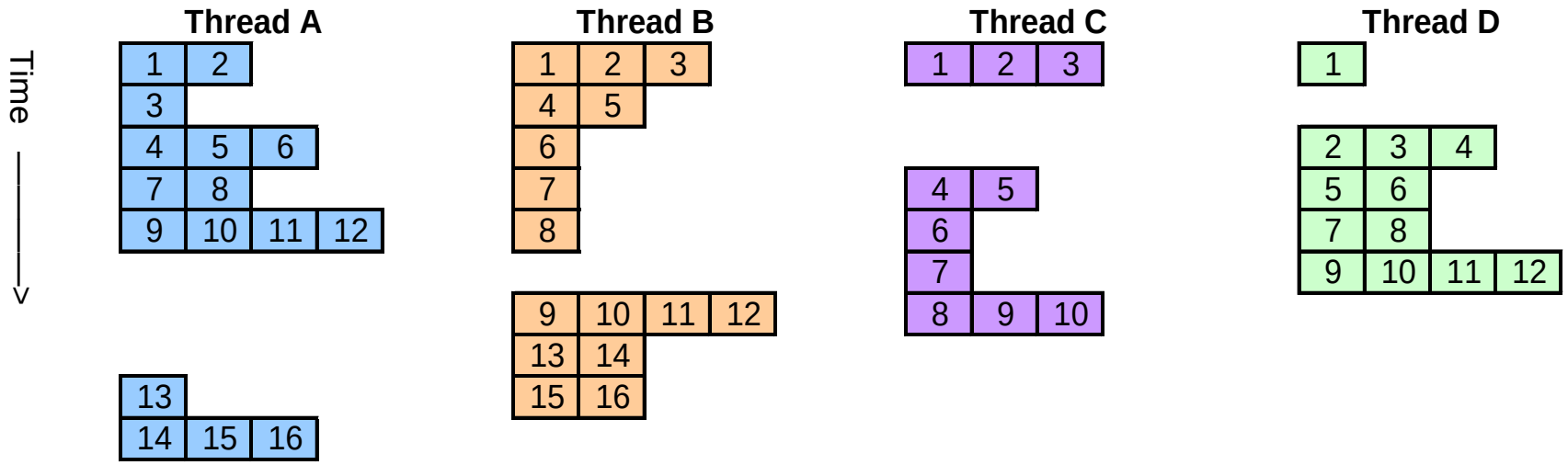
- Cores are simple (single-issue, 6 stages, no branch prediction), small, and power-efficient

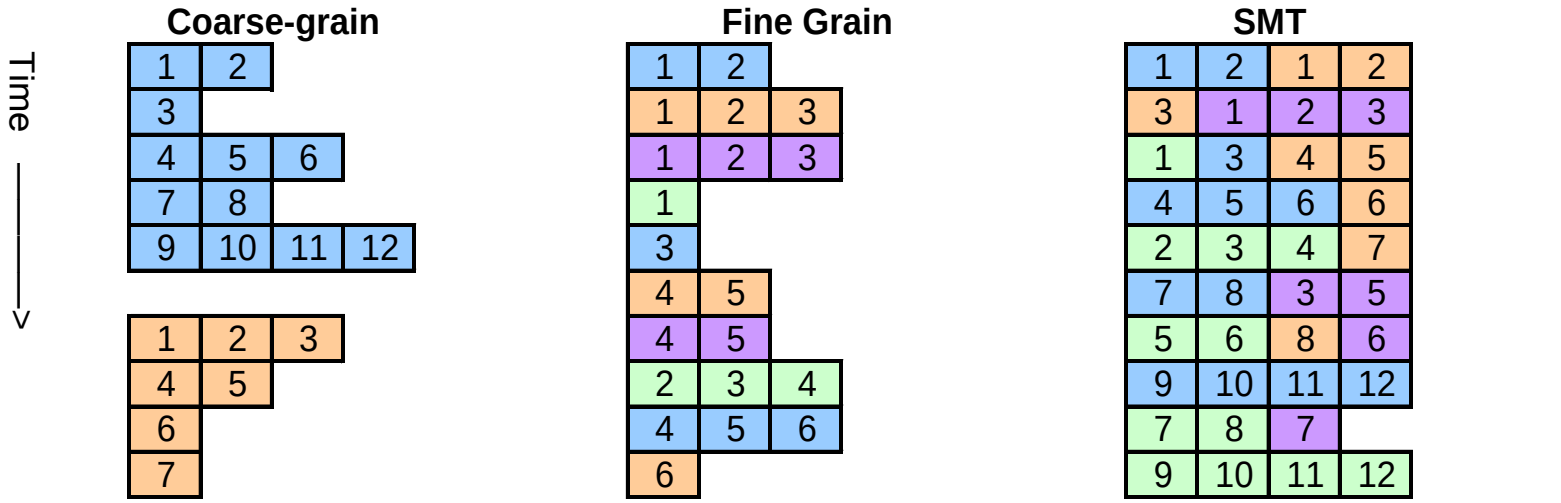


Simultaneous Multithreading (SMT)

- A variation on multithreading that uses the resources of a multiple-issue, dynamically scheduled processor (superscalar) to exploit **both ILP and TLP**
  - Most SS processors have more functional unit parallelism than a single thread can effectively use
  - With register renaming and dynamic scheduling, multiple instructions from independent threads can be issued without regard to dependencies among them
    - Need separate rename tables (RUUs) for each thread or need to be able to indicate which thread the entry belongs to
    - Need the capability to commit from multiple threads in one cycle
- Intel's Pentium 4 SMT is called hyperthreading
  - Supports just two threads (doubles the architecture state)

Threading on a 4-way SS Processor Example





Microprocessor Comparison

Processor	SUN T1	Opteron	Pentium D	IBM Power 5
Cores	**8**	2	2	2
Instruction issues / clock / core	1	3	3	4
Peak instr. issues / chip	**8**	6	6	**8**
Multithreading	Fine-grained	No	SMT	SMT
L1 I/D in KB per core	16/8	**64/64**	12K uops/16	64/32
L2 per core/shared	**3 MB** shared	1MB / core	1MB/ core	1.9 MB shared
Clock rate (GHz)	1.2	2.4	**3.2**	1.9
Transistor count (M)	**300**	233	230	276
Die size (mm^2)	379	199	206	**389**
Power (W)	**79**	110	130	125

Summary of Hardware Multithreading

- Benefit:
  - All multithreading techniques improve the utilisation of processor resources and, hence, the performance
  - If the different threads are accessing the same input data they may be using the same regions of memory
    - Cache efficiency improves in these cases
- Disadvantage:
  - The perceived performance may be degraded when comparing with a single-thread CPU
    - Multiple threads interfering with each other
  - The cache has to be shared among several threads so effectively they would use a smaller cache
  - Thread scheduling at hardware level adds high complexity to processor design
    - Thread state, managing priorities, OS-level information, …

Shared Memory Model with Explicit Thread-based Parallelism

- Shared memory process consists of **multiple threads**, **explicit programming model** with full programmer control over parallelization
- Pros:
  - Takes advantage of shared memory, programmer need not worry (that much) about **data placement**
  - Programming model is “serial-like” and thus conceptually simpler than alternatives
  - **Compiler directives** are generally simple and easy to use
  - Legacy serial code does not need to be rewritten
- Cons:
  - Codes can only be run in shared memory environments!
  - Compiler must support (e.g., **CilkPlus** and **OpenMP** in gcc 4.xx) (both are available on the machines in MC10)

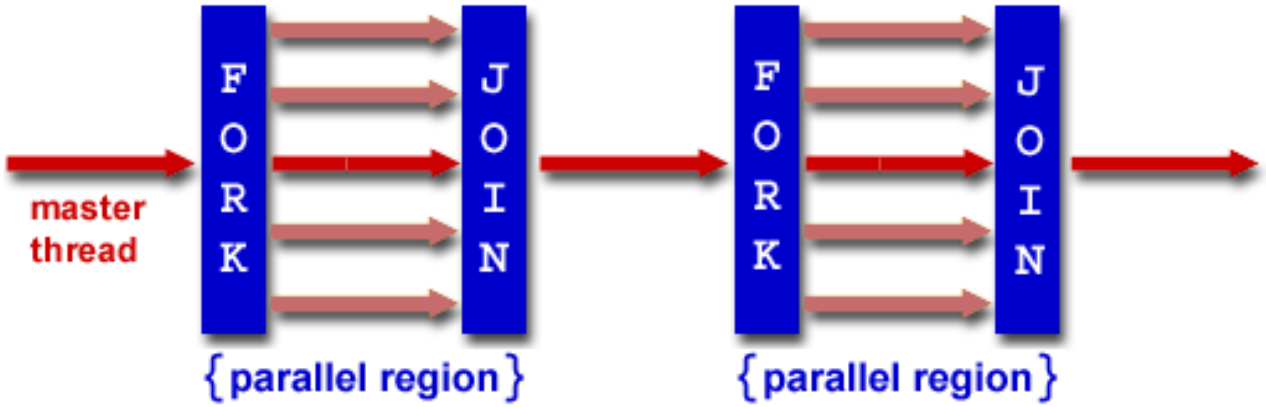
Introduction to CilkPlus

Introduction to OpenMP

- API used for multi-threaded, shared memory parallelism
  - Compiler Directives
  - Runtime Library Routines
  - Environment Variables
- Portable
- Standardized
- See
  http://www.openmp.org/mp documents/OpenMP4.0.0.pdf
  http://computing.llnl.gov/tutorials/openMP/

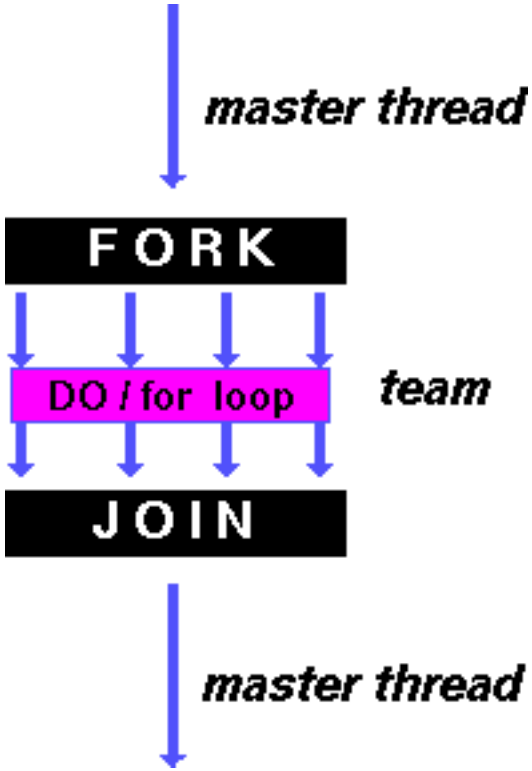
OpenMP Programming Model

- Fork - Join Model:

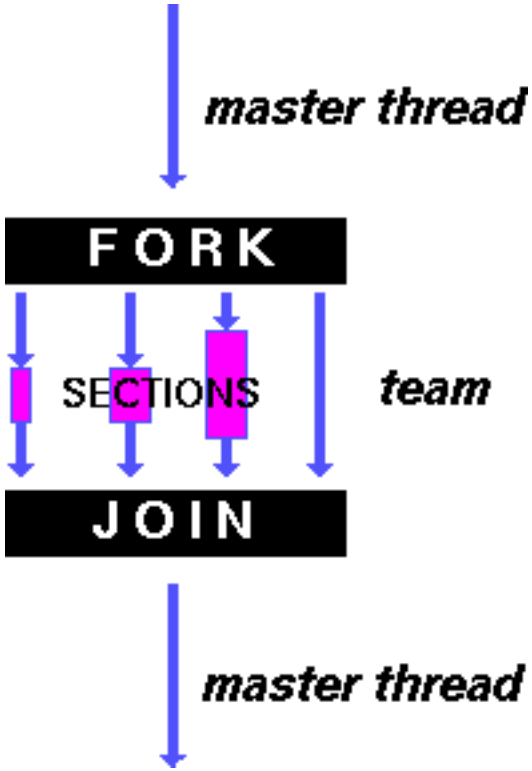


- OpenMP programs begin as single process: *master thread*; Executes sequentially until the first parallel region construct is encountered
  - **FORK**: the master thread then creates a team of parallel threads
  - Statements in program that are enclosed by the parallel region construct are executed in parallel among the various team threads
  - **JOIN**: When the team threads complete the statements in the parallel region construct, they synchronize and terminate, leaving only the master thread

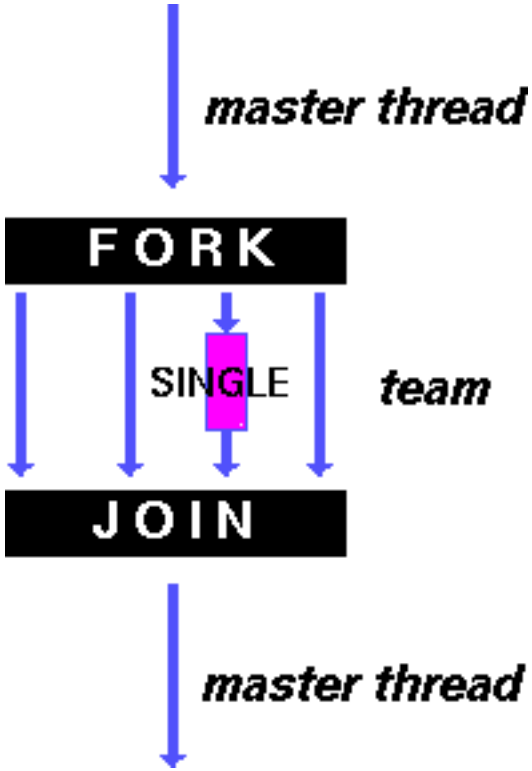
OpenMP Directives



shares iterations of a loop across the team

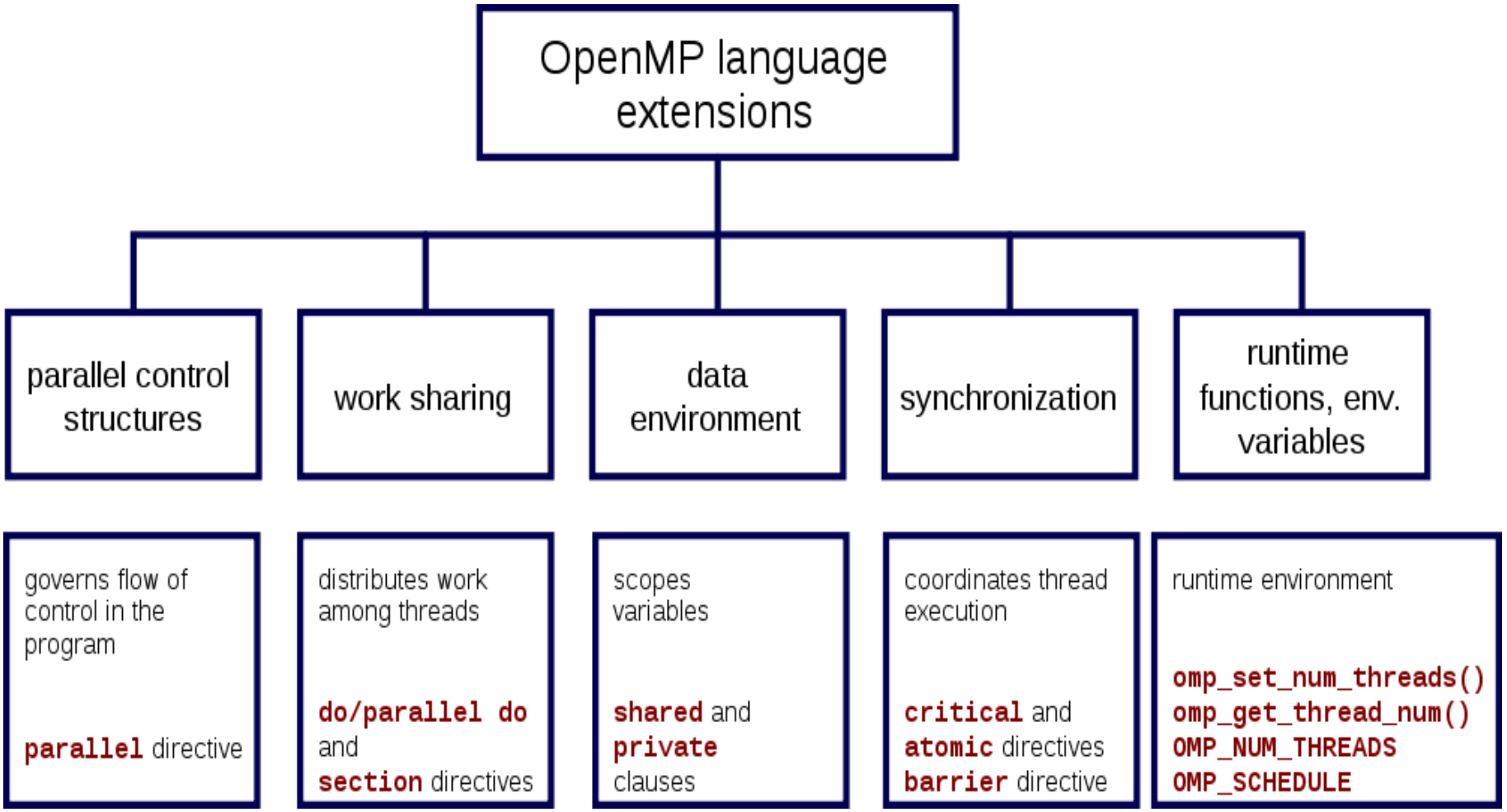


each section executed by a separate thread



serializes the execution of a thread

OpenMP Specification



OpenMP Extends C with Pragmas

- **Pragmas** are a mechanism C provides for language extensions
- Commonly implemented pragmas: structure packing, symbol aliasing, floating point exception modes
- Good mechanism for OpenMP because compilers that don't recognize a pragma are supposed to ignore them
  - Runs on sequential computer even with embedded pragmas

Matrix Multiply in OpenMP

```
#pragma omp parallel for private(tmp, i, j, k)
  for(i=0; i<Ndim; i++){
     for(j=0; j<Mdim; j++){
        tmp = 0.0;
           for( k=0; k<Pdim; k++){
              tmp += A[i*Ndim+k] * B[k*Pdim+j];
           }
           C[i*Ndim+j] = tmp;
     }
  }
```

Note: Outer loop spread across N threads; inner loops inside a thread

Amdahl's Law: theoretically how much speed up you can get by parallelization

$$\text{Speed up} = \frac{S + P}{S + (P/N)}$$

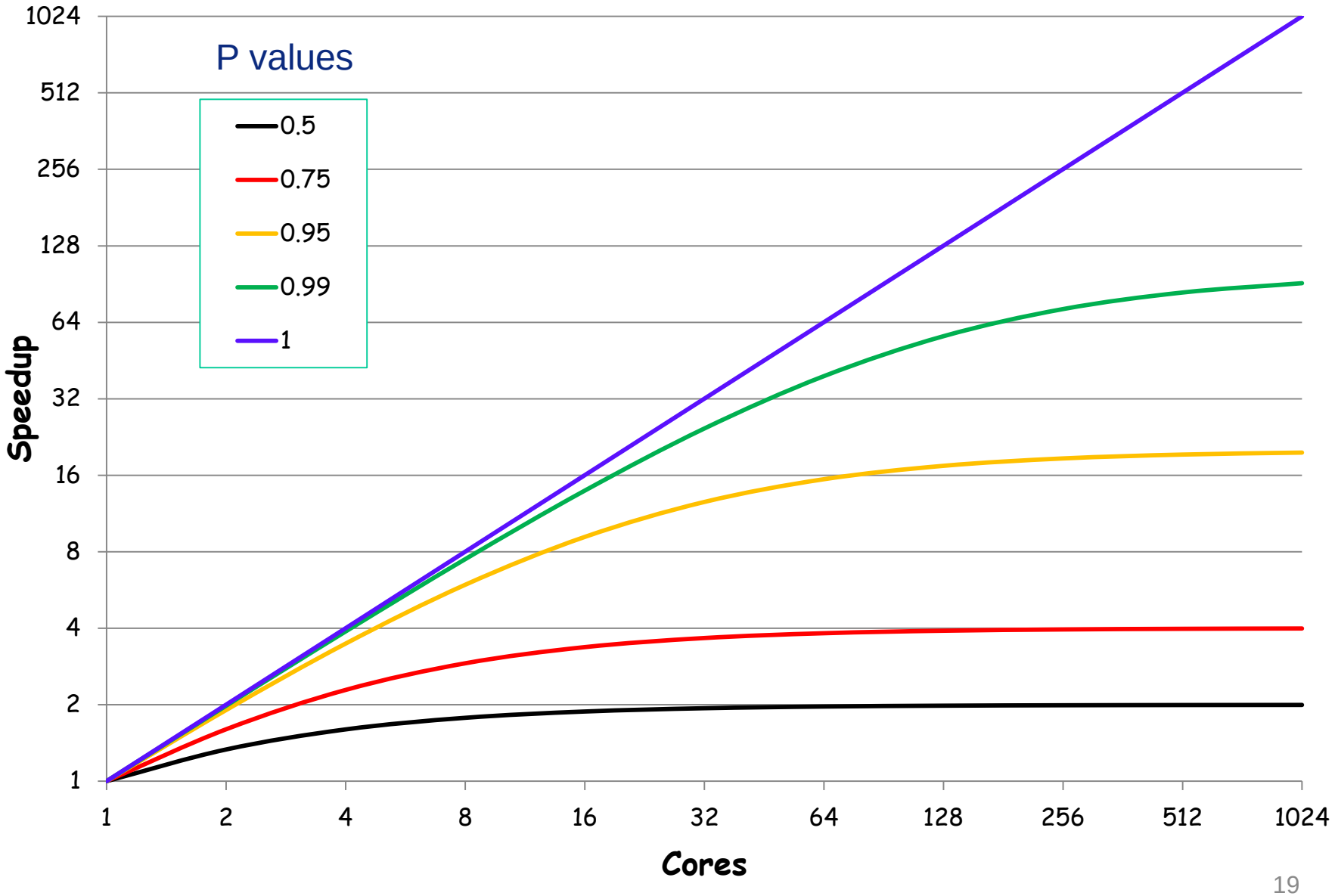
S = Fraction of the code which is serial

P = Fraction of the code which can be parallel

S + P = 1

N = Number of processor

Amdahl's Law


P values
0.5
0.75
0.95
0.99
1
Speedup
1024
512
256
128
64
32
16
8
4
2
1
1
2
4
8
16
32
64
128
256
512
1024
Cores

Exercise: Parallelize Sum of Squares

```
S = 0;
for (i=0; i<100; ++i)
     s += X[i]**2; //two instructions per loop
```

- Each iteration depends on the result of the iteration before.
- As written, unparalleizable
  - P = 0
- How would you create parallelism here?