

CS4803DGC Design and Programming of Game Console

Spring 2011

Prof. Hyesoon Kim



**Georgia
Tech**



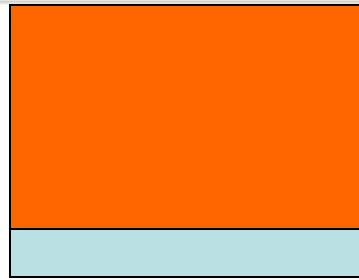
College of
Computing



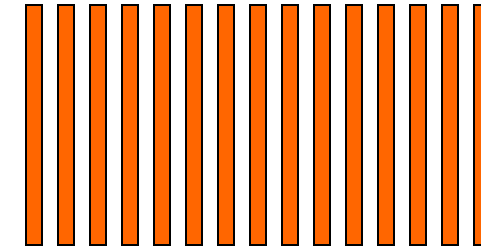
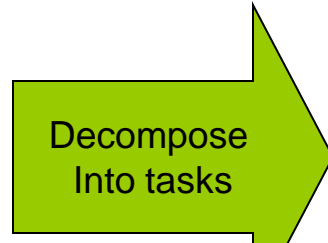
Today's Goal

- Today, we will study typical patterns of parallel programming
- This is just one of the ways.
- Materials are based on a book by Timothy.

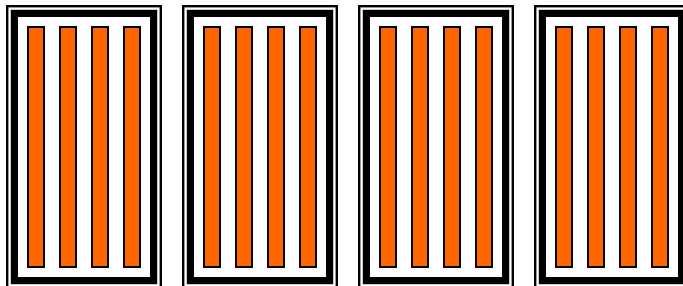
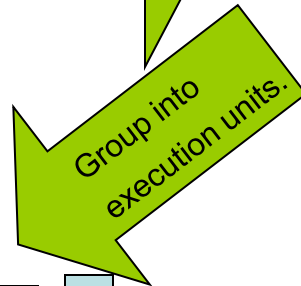
How to Create a Parallel Application:



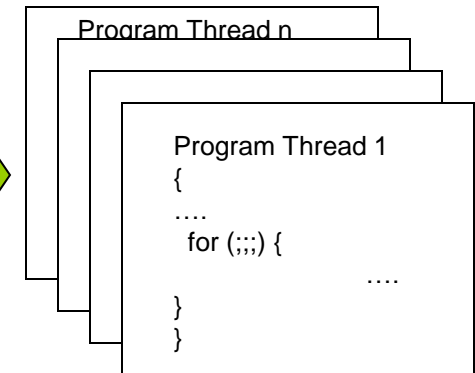
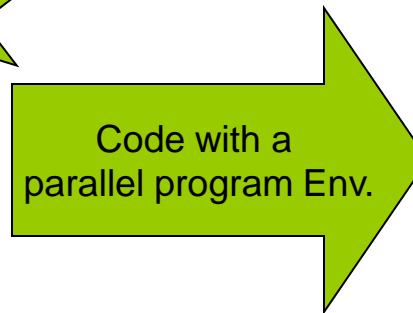
Original Problem



Tasks, shared and local data



Units of execution + new shared data for extracted dependencies



Corresponding source code

Learning to “Think Parallel”: Design Patterns

- High quality solution to frequently recurring problem in some domain
- Learning design patterns makes the programmer to quickly understand the solution and its context.



Before Writing Parallel Programs

- Parallel programs often start as sequential programs
 - Easy to write and debug
 - Already developed/tested
- Identify program hot spots
- Parallelization
 - Start with hot spots first
 - Make sequences of small changes, each followed by testing
 - Patterns provide guidance



Amdahl's Law

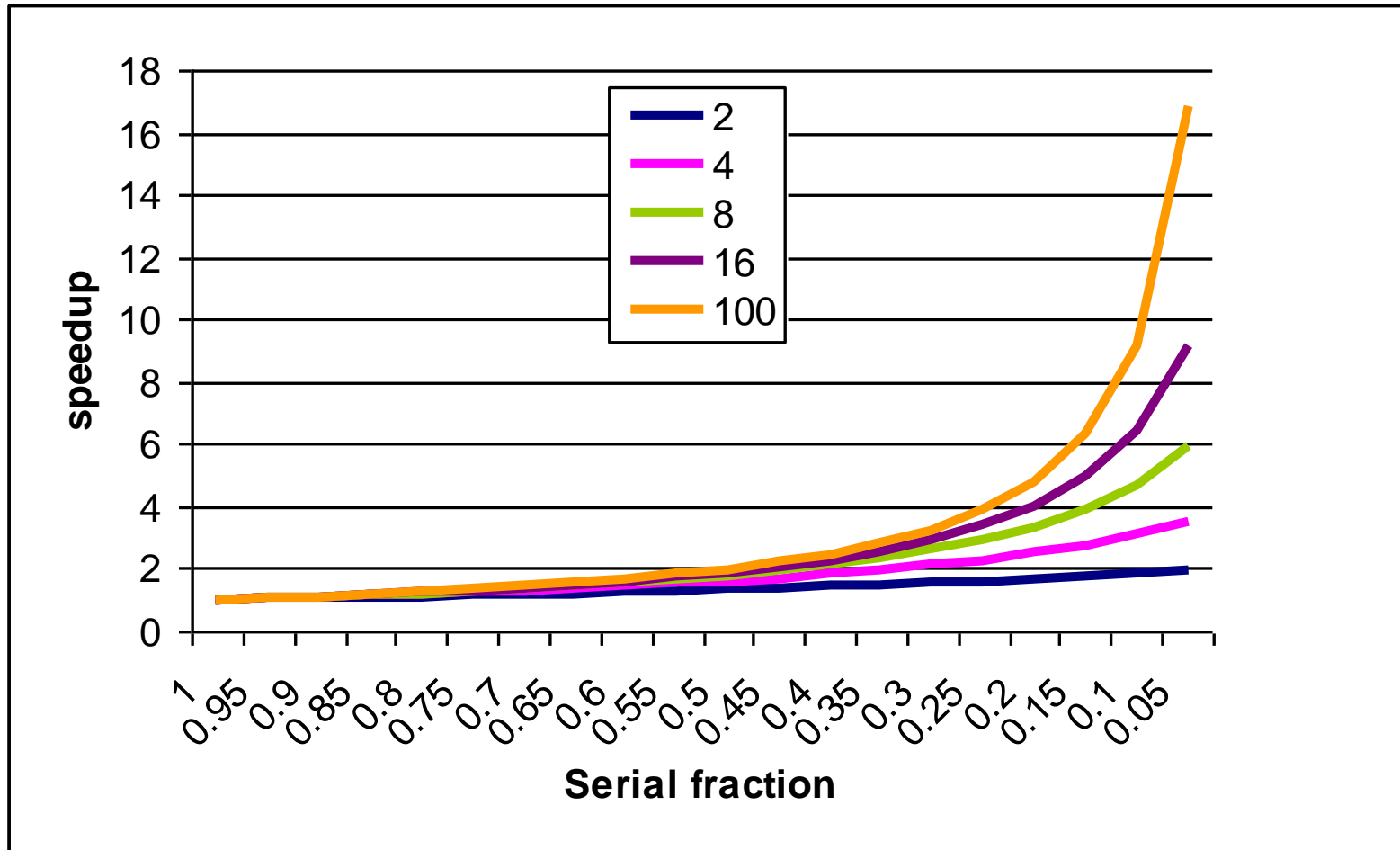
- Speedup =

Performance for entire task using the enhancement when possible

Performance for entire task without using the enhancement

- $\text{Speedup} = 1 / (P/N + S)$
- P = parallel fraction ($1 - S$)
- N = number of processors
- S = serial fraction

Amdahl's Law





Steps to Parallel Programming

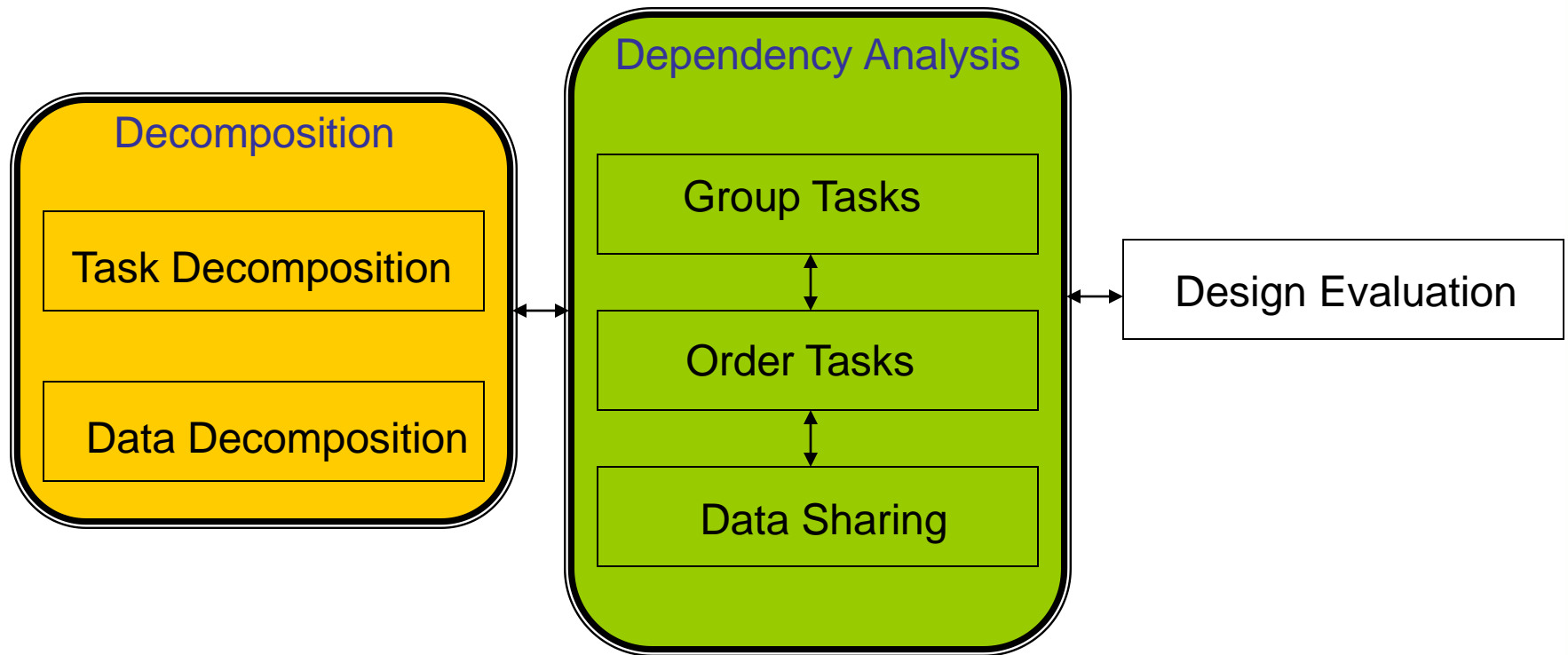
Step 1: Find concurrency

Step 2: Structure the algorithm so that concurrency can be exploited

Step 3 : Implement the algorithm in a suitable programming environment

Step 4: Execute and tune the performance of the code on a parallel system

Step 1: Finding Concurrency



Things to consider: Flexibility, Efficiency, Simplicity



Guidelines for Task Decomposition

- Flexibility
 - Program design should afford flexibility in the number and size of tasks generated
 - Tasks should not tie to a specific architecture
 - Fixed tasks vs. Parameterized tasks
- Efficiency
 - Tasks should have enough work to amortize the cost of creating and managing them
 - Tasks should be sufficiently independent so that managing dependencies doesn't become the bottleneck
- Simplicity
 - The code has to remain readable and easy to understand, and debug



Guidelines for Data Decomposition

- Data decomposition is often implied by task decomposition
- Programmers need to address task and data decomposition to create a parallel program
 - Which decomposition to start with?
- Data decomposition is a good starting point when
 - Main computation is organized around manipulation of a large data structure
 - Similar operations are applied to different parts of the data structure



Guidelines for Data Decomposition

- Flexibility
 - Size and number of data chunks should support a wide range of executions
- Efficiency
 - Data chunks should generate comparable amounts of work (for load balancing)
- Simplicity
 - Complex data compositions can get difficult to manage and debug



Common Data Decomposition

- Geometric data structures
 - Decomposition of arrays along rows, column, blocks
- Recursive data structures
 - Example: list, tree, graph



Step 2: Algorithm Structure

Organize by tasks

Task parallelism

Divide and conquer

Organize by data decomposition

Geometric decomposition

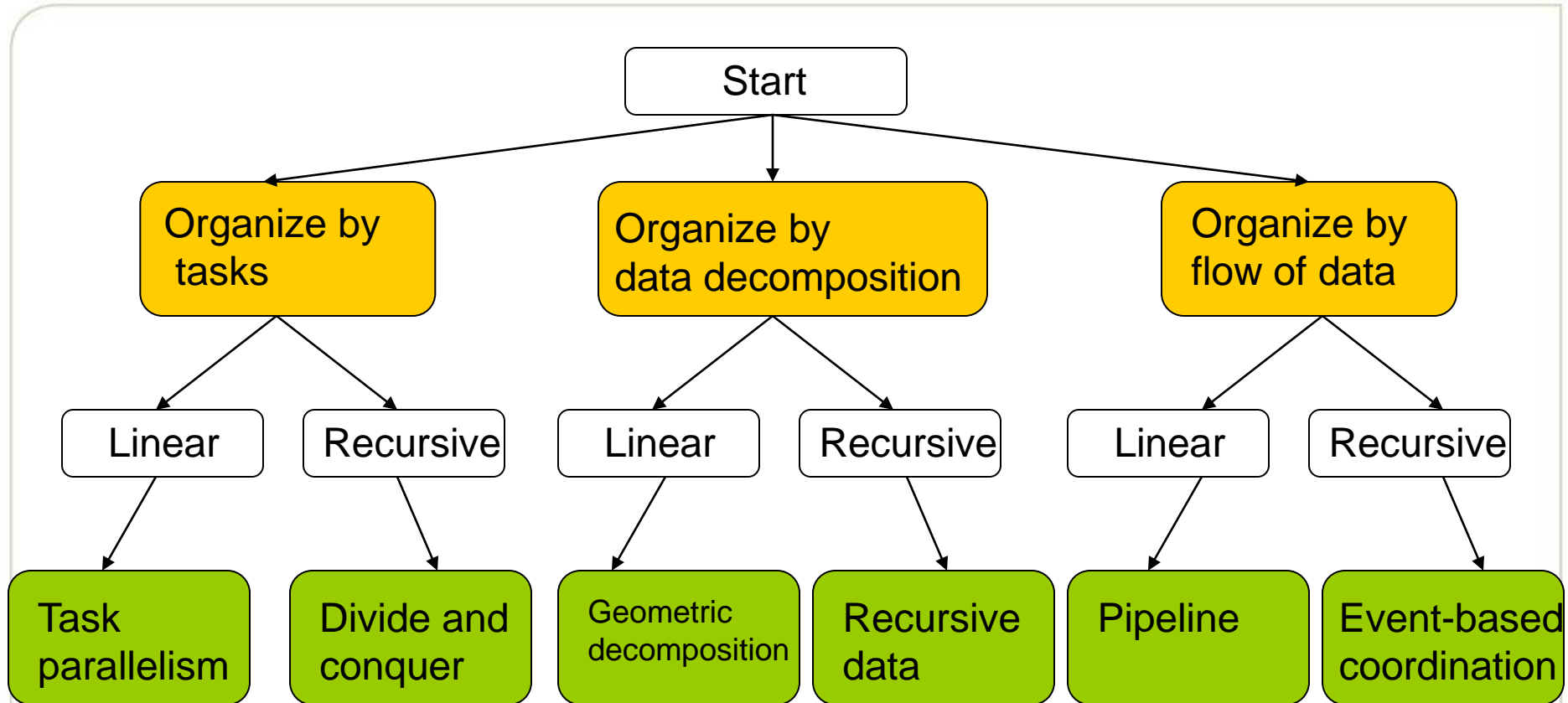
Recursive data

Organize by flow of data

Pipeline

Event-based coordinate

Algorithm Structure Design Space





Dependencies

- **Removable dependencies**

Temporary variable

```
int ii = 0, jj = 0;
for (int i = 0; i < N; i++) {
    ii = ii + 1;
    d[ii] = big_time_consuming_work (ii);
    jj = jj + i;
    a[jj] = other_big_calc(jj);
}
```

-> transformed code

```
For (int i = 0; i < N; i++) {
    d[i+1] = big_time_consuming_work(i+1);
    a[(i*i+i)/2] = other_big_calc((i*i+i)/2));
}
```

- **Separable dependencies**

```
for (int i = 0; i < N; i++) {
    sum = sum + f(i);
}
```



Supporting Structures

Program structures

SPMD

Master/Worker

Loop Parallelism

Fork/Join

Data structures

Shared Data

Shared Queue

Distributed Array



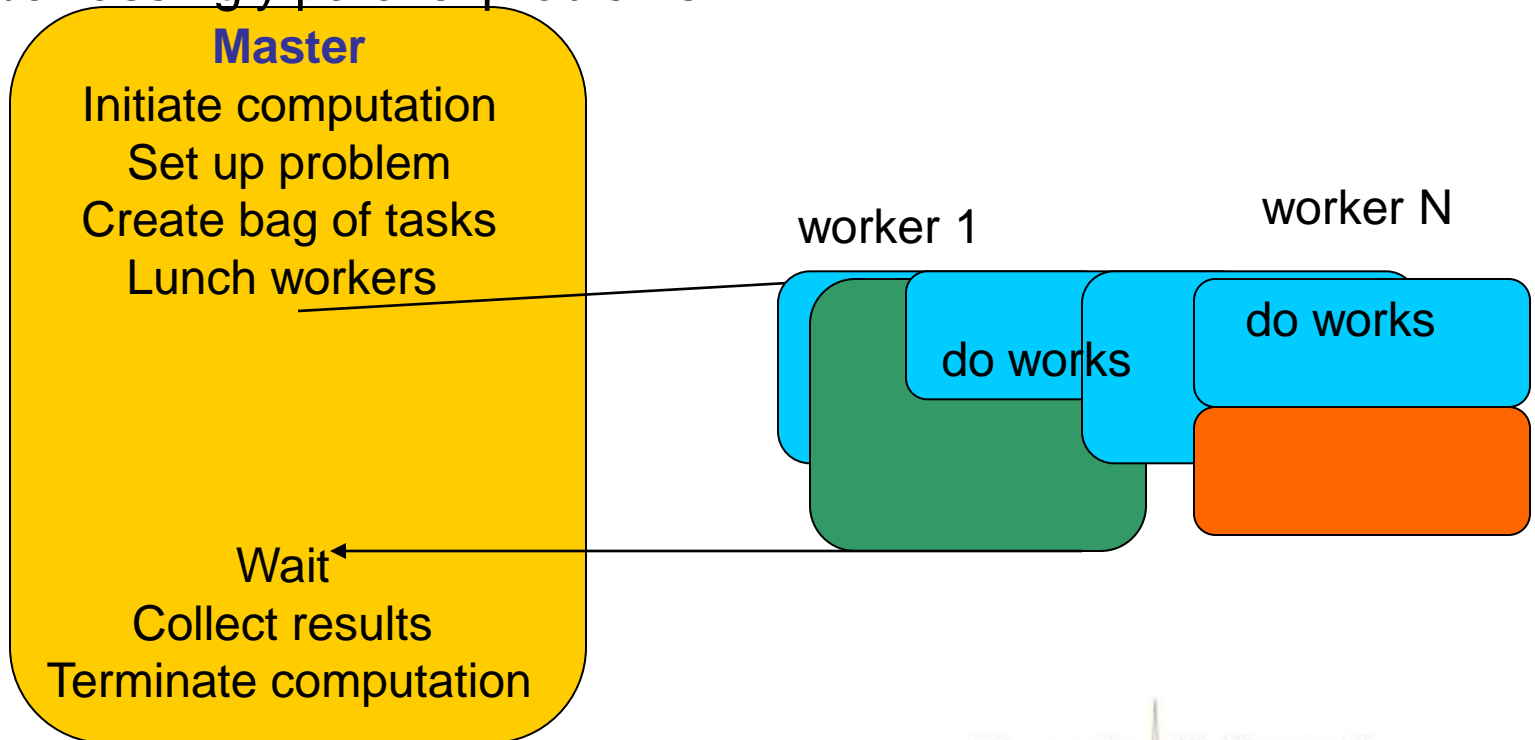
SPMD Pattern

- Single program, multiple data
- All UEs execute the same program in parallel, but each has its own set of data.
 - Initialize
 - Obtain a unique identifier
 - Run the same program each processor
 - Distributed data
 - Finalize
- CUDA



Master/Worker Pattern

- A master process or thread set up a pool of worker processes or threads and a bag of tasks.
- The workers execute concurrently, with each worker repeatedly removing a task from the bag of tasks.
- Embarrassingly parallel problems

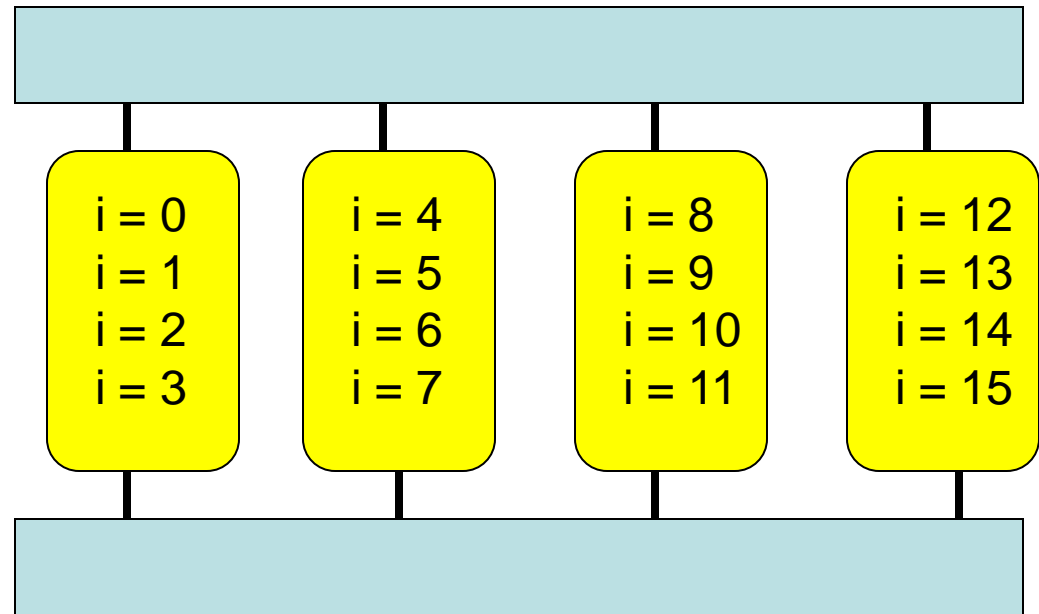




Loop Parallelism Pattern

- Many programs are expressed using iterative constructs
 - Programming models like OpenMP provide directives to automatically assign loop iteration to execution units
 - Especially good when code cannot be massively restructured

```
#pragma omp parallel for  
For (i = 0; i < 16; i++)  
    c[i] = A[i]+B[i];
```

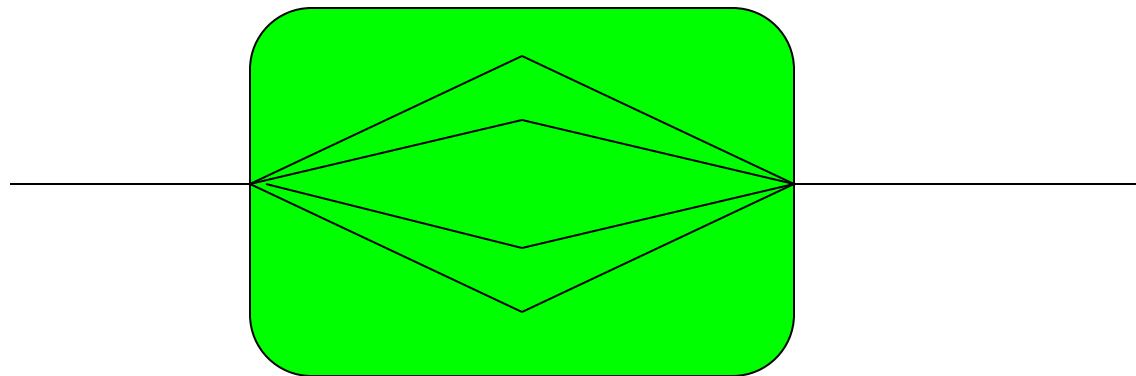




Fork/Join Pattern

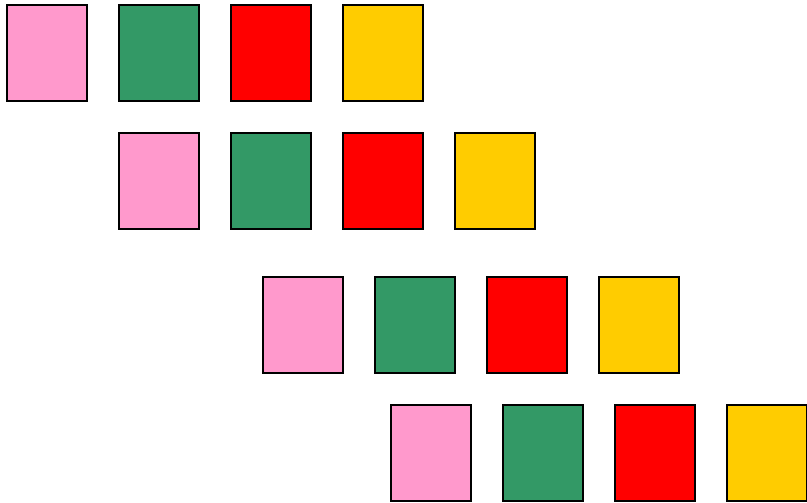
- A main UE forks off some number of other UEs that then continue in parallel to accomplish some portion of the overall work.
- Parent tasks creates new task (fork) then waits until all they complete (join) before continuing on with the computation

Parallel region





Pipeline Pattern



- Examples:
 - Instruction pipeline in modern CPUs
 - Algorithm level pipelining
 - Signal processing
 - Graphics
 - Shell programs
 - `Cat sampleFile | grep "word" | wc`



Choosing the Patterns

	Task Parallel.	Divide/ Conquer	Geometric Decomp.	Recursive Data	Pipeline	Event-based
SPMD	😊😊😊😊	😊😊😊	😊😊😊😊	😊😊	😊😊😊	😊😊
Loop Parallel	😊😊😊😊	😊😊	😊😊😊			
Master/Worker	😊😊😊😊	😊😊	😊	😊	😊	😊
Fork/Join	😊😊	😊😊😊😊	😊😊		😊😊😊😊	😊😊😊😊

Supporting Structures Pattern vs. Programming Environment



	OpenMP	MPI	CUDA
SPMD	☺ ☺ ☺	☺ ☺ ☺ ☺	☺ ☺ ☺ ☺ ☺
Loop Parallel	☺ ☺ ☺ ☺	☺	
Master/Slave	☺ ☺	☺ ☺ ☺	
Fork/Join	☺ ☺ ☺		

Step 3: The Implementations

Mechanisms Design Space



UE Management

Thread control

Process control

Synchronization

Memory sync/fences

Barriers

Mutual exclusion

Communications

Message passing

Collective communications

Other communications

- Program language
- Hardware



Jargon of Parallel Computing

- Task
- Unit of Execution (UE): process, thread
- Processing Element (PE)
- Load balance
- Synchronization
- Race conditions
- Dead locks
- Concurrency



Now Parallel programming!

- The lecture is just one guidelines.
- Most parallel programming is finding ways of avoiding data dependences, finding efficient data structures.
- Can compiler do it?
 - Automatic parallelization
 - Speculative parallelization?

Summer and Fall UROP/GRA (summer) Positions

- Paid/non-paid positions.
- Game developers for cloud game
 - Or any companies if you know, let me know
- Other research positions
 - Good programming skills.
 - Not related to games
- If you know other students, let me know.