Structuring Parallel Algorithms
Slides from:

- David Kirk/NVIDIA and Wen-mei W. Hwu,
- ECE 498AL, University of Illinois, Urbana-Champaign

- Podcasts of their lectures (recommended):
  - http://courses.ece.illinois.edu/ece498/al/Syllabus.html
Key Parallel Programming Steps

1) To find the concurrency in the problem
2) To structure the algorithm to translate concurrency into performance
3) To implement the algorithm in a suitable programming environment
4) To execute and tune the performance of the code on a parallel system

Unfortunately, these have not been separated into levels of abstractions that can be dealt with independently.
Algorithm

- A step by step procedure that is guaranteed to terminate, such that each step is precisely stated and can be carried out by a computer
  - Definiteness – the notion that each step is precisely stated
  - Effective computability – each step can be carried out by a computer
  - Finiteness – the procedure terminates

- Multiple algorithms can be used to solve the same problem
  - Some require fewer steps
  - Some exhibit more parallelism
  - Some have larger memory footprint than others
Choosing Algorithm Structure

Start

Task-centric
- Linear
  - Task Parallelism
- Recursive
  - Divide and Conquer

Data Flow centric
- Regular
  - Pipeline
- Irregular
  - Event Driven

Data-centric
- Linear
  - Geometric Decomposition
- Recursive
  - Recursive Data
A simple implementation

- Assume we have already loaded array into
  
  ```
  __shared__ float partialSum[]
  
  unsigned int t = threadIdx.x;
  for (unsigned int stride = 1;
      stride < blockDim.x; stride *= 2)
  {
      __syncthreads();
      if (t % (2*stride) == 0)
          partialSum[t] += partialSum[t+stride];
  }
  ```
Mapping a Divide and Conquer Algorithm

Thread 0  Thread 2  Thread 4  Thread 6  Thread 8  Thread 10

0  1  2  3  4  5  6  7  8  9  10  11

0+1  2+3  4+5  6+7  8+9  10+11

0...3  4..7  8..11

0..7  8..15

iterations

Array elements
Tiled (Stenciled) Algorithms are Important for Geometric Decomposition

- A framework for memory data sharing and reuse by increasing data access locality.
  - Tiled access patterns allow small cache/scartchpad memories to hold on to data for re-use.
  - For matrix multiplication, a 16X16 thread block perform 2*256 = 512 float loads from device memory for 256 * (2*16) = 8,192 mul/add operations.

- A convenient framework for organizing threads (tasks)
Increased Work per Thread for even more locality

- Each thread computes two elements of $P_{d_{sub}}$
- Reduced loads from global memory ($M_d$) to shared memory
- Reduced instruction overhead
  - More work done in each iteration
Double Buffering
- a frequently used algorithm pattern

- One could double buffer the computation, getting better instruction mix within each thread
  - This is classic software pipelining in ILP compilers

```c
Loop {
    Load current tile to shared memory
    syncthreads()
    Compute current tile
    syncthreads()
}
```

```c
Load next tile from global memory
Loop {
    Deposit current tile to shared memory
    syncthreads()
    Load next tile from global memory
    Compute current tile
    syncthreads()
}
```
Double Buffering

- Deposit blue tile from register into shared memory
- Syncthreads
- Load orange tile into register
- Compute Blue tile
- Deposit orange tile into shared memory
- ....
One can trade more work for increased parallelism

- Diamond search algorithm for motion estimation work efficient but sequential
  - Popular in traditional CPU implementations

- Exhaustive Search totally parallel but work inefficient
  - Popular in HW and parallel implementations
An MPEG Algorithm based on Data Parallelism

- Loops distributed – **DOALL** style
  - Replicates instructions and tables across accelerators
- If instructions and data are too large for local memory…
  - Large memory transfers required to preserve data
  - Saturation of and contention for communication resources can leave computation resources idle

**Memory bandwidth constrains performance**
Loop fusion & memory privatization

- Stage loops fused into single DOALL macroblock loop
  - Memory privatization reduces main memory access
- Replicates instructions and tables across processors
  - Local memory constraints may prevent this technique

Novel dimensions of parallelism reduce communication
Pipeline or “Spatial Computing” Model

- Each PE performs as one pipeline stage in macroblock processing
- Imbalanced stages result in idle resources
- Takes advantage of direct, accelerator to accelerator communication
- Not very effective in CUDA but can be effective for Cell

Efficient point-to-point communication can enable new models
Small decisions in algorithms can have major effect on parallelism.
(H.263 motion estimation example)

- Different algorithms may expose different levels of parallelism while achieving desired result.
- In motion estimation, can (a) Guess vectors are obtained from the previous macroblock.
  use previous vectors (either from space or time) as guess vectors.

(b) Guess vectors are obtained from the corresponding macroblock in the previous frame.