Outline

- Can we do the same thing for CUDA?
- Yes but …
- Coarse grain wavefront parallelism
Ex1 revisited = Ex 2

```c
int i, j;
for (t=1; t<=N+M-3; t++)
    #pragma omp parallel for private i, j
    for (p=max(1,t-M+1); t<=min(t,N-1); p++) {
        i = p;
        j = t-p+1;
        // A[i,j] = foo(A[i,j-1], A[i-1,j]);
        A[t%2, p] = bar(A[(t-1)%2, p-1], A[(t-1)%2, p]);
        if (p==N-1) B[j] = A[t%2, j];
    }
```

So what about CUDA

- Two levels of parallelism
  - Focus for now only on fine grain parallelism
- CUDA threads are not the same as OpenMP
  - Work done by each thread is explicitly described (parametric function of tid)
  - Implicit outer parallel loops, iterating over
    - Block coordinates, and
    - Thread coordinates
- Memory may be
  - private (registers), or
  - shared (shared memory)
- Synchronization may have to be explicit
- Tweak what we did for OpenMP to do CUDA
Make p the outer loop?

```
for (i=1; i<N; i++)
    for (j=1; j<M; j++)
        {p, t | 1<=p<N; 1<=(t-p+1)<M}
```

```
#pragma omp parallel for // make the outer loop parallel
for (p=1; p<=N-1; p++)
    for (t=p; t<=M+p-1; t++) {
        // body
        #pragma omp barrier (what if we added this line)
    }
```

Assume numthreads = N-1

```
#pragma omp parallel for // make the outer loop parallel
for (p=1; p<=N-1; p++)
    for (t=0; t<p; t++) {
        #pragma omp barrier
    }
    for (t=p; t<=M+p-1; t++) { // body
        #pragma omp barrier
    }
    for (t=M+p; t<tmax; t++) {
        #pragma omp barrier
    }
```
Dummies & Synchronization

Feedback Questions & Recap

- Why is outer p not correct
- Why need the synchronizations at the end
  - OpenMP semantics – deadlock
- Should have done this stuff first before doing the models
- Quiz on more isoefficiency analysis of dense matrix computations
Why is outer p incorrect?

```c
#define omp parallel for 
for (p=1; p<N-1; p++)
    for (t=p; t<=M+p-1; t++) {
        // body
    }
```

- OpenMP semantics: all iterations of the p loop should be independent of each other
  - But iteration p
    - reads data produced by iteration p-1, and
    - produces data to be consumed by p+1
- Solution – add explicit synchronization

---

So what about CUDA (recap)

- Two levels of parallelism
  - Focus for now only on fine grain parallelism
- CUDA threads are not the same as OpenMP
  - Work done by each thread is explicitly described (parametric function of tid)
  - Implicit outer parallel loops, iterating over
    - Block coordinates, and
    - Thread coordinates
- Memory may be
  - private (registers), or
  - shared (shared memory)
- Synchronization may have to be explicit
- Tweak what we did for OpenMP to do CUDA
Dummy (nop) nodes

Synchronization in CUDA

- Waruna’s toy kernel (array increment)
  - Each thread increments all the elements in an array (in shared memory)
  - Avoid races by taking turns
  - Thread 0 starts first and updates the first block (of width SUBTILE_WIDTH)
  - Next it moves on to the next block while thread 1 updates the one to which thread 0 just wrote
  - ...

Colorado State University
What if numthreads < N-1

- This is the more realistic case:
  - N is typically large (limited only by shared memory capacity)
  - numthreads is no more than a machine dependent upper bound (e.g., 512), usually much less than N
  - Solution: divide the rows into strips of height \( h = (N-1)/\text{numthreads} \) and have each thread responsible for the whole strip
    - Avoid serialization by splitting columns into strips too
    - Basic unit is a rectangular, hxw tile

Dependence Graph (Ex 1)
Knapsack DP parallelization

Lessons from HW0
- The code (i.e., function `fill-table` in HW3) was memory bound
- `omp` overhead of thread spawn/sync in each of the outer n iterations

HW3: Coarse grain parallelism
- What is the speedup? Is it good?
- Why?
- What happens if fine-grain parallelism is added?

Dependences of the DPKP
“Better” Parallelization

Pipeline fill-flush vs “concurrent start” of all threadblocks

Load balance
- During fill/flush
- In steady state: adjust tile sizes such that # threadblocks is multiple of #SMs – can be parameterized
  - Easy with “concurrent start wavefronts”
  - Complicated with [1,1] wavefront

So do concurrent start with tiling to improve the balance
Tiled DPKP Dependences

Main Drawback

- Tiling loses the concurrent start property
- Unless
  - Tile height is 1
- Challenge – how to retain/improve balance (arithmetic intensity) with tiles of height 1