CS 475 Parallel Programming
Wavefront Parallelization

Sanjay Rajopadhye
Colorado State University

Outline

- Parallelizing programs with dependences
  - None of the loops in the program can be parallelized
- Dependence analysis
- Fine grain wavefronts
  - Determining the orientation of wavefronts
  - Transforming the program
Parallelization

Consider two statement instances x and y, where x executes after y in the sequential version of the program, that we want to parallelize.

x depends on y if x and y access (read or write) the same memory location, notation: y \leftarrow x

Three kinds:
- Y: write \leftarrow X: read RAW: read after write (true)
- Y: read \leftarrow X: write WAR: write after read (anti)
- Y: write \leftarrow X: write WAW: write after write (output)
- Y: read \leftarrow X: read RAR: write after read (input)

WHY?

Changing the order changes the outcome of the program.

Colorado State University
Wavefront Parallelization

- How to parallelize computations (e.g., loops in OpenMP) that have dependences:
  - None of the loop iterations are independent

Simple examples

```c
for (i=1; i<N; i++)
  for (j=1; j<M; j++)
    A[i,j] = foo(A[i,j-1], A[i-1,j])

for (i=1; i<N; i++)
  for (j=1; j<M; j++)
    B[j] = bar(B[j-1], B[j])

for (i=1; i<N; i++)
  for (j=1; j<M; j++)
    C[i] = baz(C[i-1], C[i])
```
Iteration Space & Data Space

- **Iteration Space**: set of values that the loop iterators can take
  - Rectangular region, with “corners” \([1,1]\) and \([N-1, M-1]\)

- **Data Space**: set of values of array indices accessed by the statements in the program
  - Ex 1: 2-D table, (nearly) identical to the iteration space
  - Ex 2: 1D array, bounded by \([0, M-1]\)
  - Ex 3: 1D array, bounded by \([0, N-1]\)

---

References and Dependences

- **Reference**: a occurrence of an array variable on either
  - left hand side (write reference)
  - right hand side (read)
  of a statement in the loop body

- **Dependences**: specify which iteration points depend on which others
  - can be refined if/when there are multiple statements in the program
Finding the dependences

- Very hard problem (undecidable in general) but we have simple cases
  - An iteration point \([i, j]\) reads a memory location
  - (Many) iterations (may) have written to that location
  - Find this set (as a function of \([i,j]\))
  - Find the “most recent writer” in this set (again, as a function of \([i,j]\))

Execution order

- [Diagrams showing execution order]
Solutions to examples

- Ex1 and Ex2 (same solution, even though the data space is very different). Iteration [i,j] depends on:
  - [i, j-1] and [i-1, j] neighbors on west and north
  - Ex2 has an additional (memory based dependence)
    - Iteration [i-1, j+1] reads a memory location that the iteration [i, j] is overwriting, that must also happen before [i, j] so it cannot be executed before its northeast neighbor
- Ex3 is more complicated
  - [i,j] depends on [i,j-1] and [i-1, M-1]

Dependence Graph (Ex 1)
(Finally) the parallelization

- Now that we know the dependences between iterations (the dependence graph)
  - Analyze to determine what can happen at what time (hopefully many things can happen at the same time)
  - Rewrite the program to represent this new order
Ex1 wavefront parallelization

Redraw the graph
Writing the (OpenMP) code

- Node $[i, j]$ is mapped to $[p, t]$
  - $(i, j \rightarrow p, t) = (i, j \rightarrow i, i+j-1)$
- Inverse of the transformation:
  - $(p, t \rightarrow i, j) = (p, t \rightarrow p, t-p+1)$
- Determine the transformed iteration space
- Write loops that traverse this
- Outer loop must be the time
- Inner loop is marked to be executed in parallel with
  #pragma omp parallel for
- Write the new loop body

Control structure

Source → Loop bounds → Iteration space (inequalities)

Target ← Loop bounds ← Transformed Iteration space (inequalities)
Control structure

\[
\text{for (i=1; i<N; i++)} \\
\text{for (j=1; j<M; j++)} \Rightarrow \{i, j | 1 \leq i \leq N-1; 1 \leq j \leq M-1\}
\]

\[
\{p, t | 1 \leq p \leq N-1; 1 \leq (t-p+1) \leq M-1\}
\]

\[
\{p, t | 1 \leq p \leq N-1; p \leq t \leq M+p-2\}
\]

\[
\text{for (t=1; t \leq N+M-3; t++)} \\
\text{for (p=\text{max}(1,t-M+2); p \leq \text{min}(t, N-1); p++) // this is parallel}
\]

From inequalities to loops

- Work “inside out,” i.e., generate bounds on the innermost dimension (say $z_n$) first.
- For each inequality, rearrange it into the form: $a_n z_n \geq \exp$, for some constant coefficient $a_n$.
  - If $a_n$ is positive, $\exp/a_n$ is a lower bound on $z_n$.
  - Otherwise, $-\exp/a_n$ is an upper bound.
- Let $l_1 \ldots l_m$ be the lower bound expressions and $u_1 \ldots u_m$ be the upper bound expressions.
- The innermost loop is (with one caveat):
  \[
  \text{for (}z_n = \text{max}(l_1 \ldots l_n); z_n < \text{min}(u_1 \ldots u_m); z_n++)
  \]
- Recurse on the outer $n-1$ dimensions.
Recursion: eliminate $z_n$

- For each pair, $u_i$, $l_j$, introduce an inequality, $u_i \geq l_j$
- Let the collection of these inequalities define the iteration space $I_{n-1}$
  - $I_{n-1}$ is an $(n-1)$-dimensional iteration space (doesn’t involve $z_n$)
  - The first $n-1$ coordinates of every point in the original iteration space, $I_n$ also satisfy $I_{n-1}$
  - $I_n$ is the intersection of $I_{n-1}$ and loop bounds

Example (on doc cam)
New loop body

- At each point \([t, p]\) in the new loop,
  - Determine the original iteration point that was mapped to \([t, p]\) (inverse of the rectangle-to-parallelogram transformation).
    Given \([t, p] = [i+j-1, j]\) solve for \([i, j]\) in terms of \(t\) and \(p\).
  - Add synchronization (optional)
  - Optionally, change memory

```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+2); t<=min(t,N-1); p++) {
    i = p;
    j = t-p+1;
    // insert old loop body (unchanged) here:
    // we chose \(t\) and \(p\) as brand new index names
    A[i,j] = foo(A[i,j-1], A[i-1,j]);
}
```
Ex2 wavefront parallelization

Example 2 (contd)

- Mapping is \( (i, j \rightarrow p, t) = (i, j \rightarrow i, 2i+j-2) \)
- Inverse of the transformation:
  - \( (p, t \rightarrow i, j) = (p, t \rightarrow p, t-2p+2) \)
- Transformed iteration space:
  - \( \{p, t \mid 1 \leq p \leq N-1; 1 \leq (t-2p+2) \leq M-1\} \)
- Rewrite as:
  - \( \{p, t \mid 1 \leq p \leq N-1; t-M+3 \leq 2p \leq t+1\} \)
- Write the new loop
Example 2 (contd)

for (t=3; t<= 2N+M-5; t++)
    for (p = max(1, \floor{\frac{t-M+3}{2}} ); p <= min( \floor{\frac{t+1}{2}}, N-1) ) {

        //NEW LOOP BODY:
        i = p; j = t-2p+2;
        // copy the old body
        B[j]=bar(B[j], B[j-1]);
    }

Better way

- Early preoccupation with memory:
  - Memory allocation of the original program is hurting us
- First parallelize the “full table version”
- Then make it use less memory
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[i%2, j] = bar(A[i%2, j-1], A[(i-1)%2, j]);
        if (p==N-1) B[j] = A[i%2, j];
    }
```

Conclusions

- Only simple (fine-grain wavefronts)
- Not dealing with memory
- Granularity of synchronization/fork-join overhead
- Just the beginning
  - Tiling
  - Tiling + parallelism
  - Memory (remapping)
- Advanced topics in CS 560/575