Loop Schedules

Which Thread Executes Which Iterations
Assigning Iterations to Thread: Default

• The OpenMP runtime is free to assign any iteration to any thread

• A typical default is “static”
  • Assigns a block of contiguous iterations to each thread equally
  • E.g., if there are 800 iterations and 8 threads:
    • Thread 0 executes iteration 0-99, thread 1 executes iteration 100-199, and so on
  • Often, this helps with spatial locality because contiguous iterations tend to access contiguous memory locations

• However, as a programmer you can control this assignment
  • A particular way of assigning iterations to threads is called a “schedule”.
Assigning Iterations to Threads: Example

• How to balance the work per thread when the work per iteration is inherently imbalanced

```c
#pragma omp parallel for private(size)
for(int i=0; i<n; i++) {
    size = f(i);
    if (size < 10)
        smallwork(x[i]);
    else
        bigwork(x[i]);
}
```

Some Iterations are expensive, but we don’t know which because it depends on the function f
Dynamic Schedule

• We can have each thread pick the next iteration available
  • And when it finishes, go back and pick the next available one
• This can be implemented in many ways
  • E.g., having a shared variable that holds the value of the next available iteration
• Idea: every time a thread goes to ask for more work, it is given a chunk of iterations
  • We would like the chunk size to be controlled by the programmer
Schedule Clause and General Form

- **schedule** (kind[, chunk])
  - kind:
    - static
    - dynamic,
    - guided, (a variant of dynamic)
    - runtime, (programmer can set the schedule at runtime, using omp_set_schedule(..))
    - auto (let the compiler/runtime decide the schedule)
  - Optional chunk = scalar integer value
Schedule: Static, Dynamic or Guided

- **Static**: iterations are divided as evenly as possible among all threads with each thread getting a contiguous range of iterations
- **Static, chunk**: iterations are divided into chunks of size \( \text{chunk} \). Chunks are then assigned in round robin fashion to threads – *interleaved*
- **Dynamic, chunk**: iterations are divided into chunks of size \( \text{chunk} \) (1 if unspecified) and are assigned to threads dynamically after an initial round robin assignment – *simple dynamic*
- **Guided, chunk**: chunk size decreases exponentially from an implementation dependent value to \( \text{chunk} \) (1 if unspecified). Chunks are assigned dynamically – *guided self scheduling*
Tradeoffs in Static vs. Dynamic Scheduling

• Static has lower synchronization overhead, better spatial locality, but cannot deal with imbalances well

• Dynamic schedule will balance load well, but
  • Has a higher synchronization overhead, as threads have to coordinate frequently to decide assign iterations to threads
    • I.e., there will be overhead in picking the next available iteration
  • Spatial locality:
    • As iterations are assigned to threads when they ask for work, consecutive (chunk of) iteration may go to different threads, reducing or destroying spatial locality
Assigning Iterations to Threads

• Use dynamic schedule to balance the work per thread when the work per iteration is inherently unbalanced

```
#pragma omp parallel for schedule(dynamic,16)
for(int i=0; i<n; i++) {
    int size = f[i];
    if (size < 10)
        z[i] = smallwork(x[i]);
    else
        z[i] = bigwork(x[i]);
}
```