Performance Pitfalls

Vipin Vasu

College Of Engineering,Trivandrum

Overview

Ameliorating the impact of OpenMP worksharing constructs

- Whenever a parallel region is started or stopped or a parallel loop is initiated or ended, there is some nonnegligible overhead involved.
- Threads must be spawned or at least woken up from an idle state, the size of the work packages (chunks) for each thread must be determined, in the case of tasking or dynamic/guided scheduling schemes each thread that becomes available must be supplied with a new task to work on, and the default barrier at the end of worksharing constructs or parallel regions synchronizes all threads.
- If some simple guidelines are followed,the adverse effect of OpenMP overhead can be reduced.

Ameliorating the impact of OpenMP worksharing constructs

1. **Run serial code if parallelism does not pay off**
    - If the work sharing construct does not contain enough "work" per thread because, e.g., each iteration of a short loop executes in a short time, OpenMP overhead will lead to very bad performance.
    - It is then better to execute a serial version if the loop count is below some threshold.

Ameliorating the impact of OpenMP worksharing constructs

- Instead of disabling parallel execution altogether, it may also be an option to reduce the number of threads used on a particular parallel region by means of the NUM_THREADS clause
- Fewer threads mean less overhead, and the resulting performance may be better than with IF, at least for some loop lengths.

Ameliorating the impact of OpenMP worksharing constructs

2. **Avoid implicit barriers**
    - OpenMP worksharing constructs insert barriers at the end.This is a safe default so that all threads have completed their share of work before anything after the construct is executed.
    - In cases where this is not required, a NOWAIT clause removes the implicit barrier

Ameliorating the impact of OpenMP worksharing constructs

- There is also an implicit barrier at the end of a parallel region that cannot be removed.
- Implicit barriers add to synchronization overhead like critical regions, but they are often required to protect from race conditions

Ameliorating the impact of OpenMP worksharing constructs

3. **Try to minimize the number of parallel regions**
   - Parallelizing inner loop levels leads to increased OpenMP overhead because a team of threads is spawned or woken up multiple times

Ameliorating the impact of OpenMP worksharing constructs

- In this particular example, the team of threads is restarted N times, once in each iteration of the j loop. Pulling the complete parallel construct to the outer loop reduces the number of restarts to one

- The less often the team of threads needs to be forked or restarted, the less overhead is incurred

Ameliorating the impact of OpenMP worksharing constructs

- The SIN function call between the loops is performed by the master thread only. At the end of the first loop, all threads synchronize and are possibly even put to sleep,and they are started again when the second loop executes.

Ameliorating the impact of OpenMP worksharing constructs

- Now the SIN function in line 10 is computed by all threads, and consequently S must be privatized. It is safe to use the NOWAIT clause on the second loop in order to reduce barrier overhead.

Ameliorating the impact of OpenMP worksharing constructs

4. **Avoid "trivial" load imbalance**
   - The number of tasks, or the parallel loop trip count, should be large compared to the number of threads.
   - If the trip count is a small noninteger multiple of the number of threads, some threads will end up doing significantly less work than others, leading to load imbalance.

Ameliorating the impact of OpenMP worksharing constructs

- A typical situation where it may become important is the execution of deep loop nests on highly threaded architectures.Consider the program below:

Ameliorating the impact of OpenMP worksharing constructs

- The outer loop is the natural candidate for parallelization here, causing the minimal number of executed worksharing loops (and implicit barriers) and generating the least overhead.
- The COLLAPSE clause can collapse the two loop levels into one loop with a loop length of M x N and the resulting long loop will be executed in parallel by all threads.

Determining OpenMP overhead for short loops

- In general, the overhead consists of a constant part and a part that depends on the number of threads

Serialization

- **big fat lock**
- use a separate OpenMP lock variable
- privatization should be given priority over synchronization when possible.

False sharing

- In some cases cache coherence traffic can throttle performance to very low levels.
- standard technique is array padding
- data privatization, each thread gets its own local copy
- thereby won't occupy same cache line.