# PAR – lab2

Rosa M Badia

rosa.maria.badia@upc.edu

# Lab2

- OpenMP tutorial, 2 sessions
- cp /scratch/nas/1/par0/sessions/lab2.tar.gz .
- Useful slides in Atenea
- Deliverable is a questionnaire
  - Pay attention, the question about the overheads needs to be answered with content for two sections, one per each session
- For each code:
  - pi-vx-debug: prints iterations executed by each thread. Use script run-debug.sh to execute interactively (32 iterations by default)
  - pi-vx-omp: version used to do measure performance. Use submit-omp.sh script with queues (100000000 iterations)
  - You can generate traces with submit-extrae.sh (100000 iterations)
  - Run by default with 4 threads

# Versions implicit tasks

- V0 – sequential
  - omp_get_wtime : to measure time
  - make pi-v0-debug
  - ./run-debug.sh pi-v0
  - make pi-v0-omp
  - sbatch -p execution ./submit-omp.sh pi-v0
- V1 – introduces parallel construct
  - Parallel construct: Creates a set of threads
  - Each of the threads created in the parallel will run the so called **implicit task**
  - Incorrect version
  - Threads access at the same time variables i and x
- V2 - private classe
  - To protect variables
  - Incorrect version
  - All threads run all iterations

# Versions implicit tasks

- V3 – omp_get_num_threads()
  - Used to manually distribute the loop iterations to threads
  - omp_get_thread_num -> id of the thread
  - Still incorrect version -> check omp version
    - Races when accessing variable sum
    - sbatch -p execution ./submit-omp.sh pi-v3
- V4 – using synchronization to access sum
  - Critical construct
  - Correct version
  - Execution time???
- V5 – synchronization
  - Atomic construct
  - Correct version
  - Better execution time

# Versions implicit tasks

- V6 – using private copies of sum
  - Correct version
- V7 – reduction clause
  - The compiler creates a private copy of the reduction variable (sum)
  - It is initialized to the neutral value of the operation (0 for +)
  - Shared variable is updated at the end of the loop with the result of each thread
  - Correct version
  - Generate a tracefile
    - sbatch –p execution submit-extrae.sh pi-v7
- V8 – barrier
  - Point where all threads wait each other
  - Where is the reduction operation performed?

# Test your understanding

- Use files in directory lab2/openmp/Day1
- Answer questions in the questionnaire associate to it
- Follow the section “Observing overheads” also lab2/overheads

# Session 2

- New versions, now with the tasking model (“explicit tasks”)
- One of the threads (implicit task) is selected to generate the explicit tasks
- Tasks are executed by the other threads (and by the first thread whenever is idle)
- Synchronization with a task barrier when tasks are done
- Again, for each code:
  - pi-vx-debug: prints iterations executed by each thread. Use script run-debug.sh to execute interactively (32 iterations by default)
  - pi-vx-omp: version used to do measure performance. Use submit-omp.sh script with queues (100000000 iterations)
  - You can generate traces with submit-extrae.sh (100000 iterations)

# Versions with explicit tasks

- V9
  - Loop divided in two halves, each half computing half of the loop iterations
  - One task per half
  - Private copies of x and i
  - Each iteration is executed 4 times. Why?
- V10
  - Single construct -> a single thread executes the code, a single thread generated the tasks
  - Why the result is still not correct?

# Versions with explicit tasks

- V11
  - We need to wait for the tasks to finish
  - We need to produce correct values of sum in each task
    - Use of critical, atomic or reduction
  - Construct taskgroup – creates an implicit barrier at the end
  - Clause task_reduction – similar to the reduction for explicit tasks
  - Clause in_reduction – to indicate that the task contributes to the reduction
  - Run debug and omp version
  - Generate tracefile
- V12
  - Alternative version with atomic
  - Construct taskwait: waits for all tasks to finish
  - Observe the overhead in the trace
  - Run with omp version

# Versions with explicit tasks

- V13
  - Dependencies between tasks: depend clause
  - The third clause waits for the other two and adds their results
  - No atomic, no taskwait required

- V14
  - Taskloop construct
  - Splits the loop automatically
  - num_tasks or grainsize controls number of tasks
  - Task wait implicit
  - Which thread executes each task (with debug)

# Test your understanding

- Complete the questionnaire for the codes in lab2/openmp/Day2
- Answer observing overheads section
  - Comparison between thread and task creation and synchronization
  - Remember that you need to answer this section for the two sessions
- Deliverable:
  - Single pdf for the two members of the group, do not forget to add your names