

LAB : OpenMP GNU Compiler Instructions



Kent Milfeld, Lars Koesterke

Texas Advanced Computing Center
The University of Texas at Austin

February 6th , 2012

Introduction

What you will learn

- How to compile Code (C and Fortran) with OpenMP
- How to parallelize code with OpenMP
 - Use the correct header declarations
 - Parallelize simple loops
- How to effectively hide OpenMP statements

What you will do

- Modify example code **READ the CODE COMMENTS**
- Compile and execute the example
- Compare the run-time of the serial codes and the OpenMP parallel codes with different scheduling methods

Accessing Lab Files

- If you do not have access to a TACC system, get and extract the following tar file on your own Linux system with these instructions:

- **From your browser download instructions and tar file:**

<http://www.tacc.utexas.edu/user-services/training/course-materials>

`instructions: openmp_lab_intel.pdf`

`source files: openmp_lab_intel.tar`

- Once you have downloaded the tar file, untar it.

```
tar -xvf ./openmp_lab_intel.tar
```

- Now change directory to the source files.

```
cd lab_openmp
```

Compiling

- All OpenMP statements are activated by the OpenMP flag:
 - Intel compiler: `icc/ifort -openmp source.<c,F90>`
 - PGI compiler: `pgcc/pgf90 -mp source.<c,F90>`
 - GNU compiler: `gcc/gfortran -fopenmp -lm source.<c,F90>`

• Compilation with the OpenMP flag (`-openmp/-mp/-fopenmp`):

Activates OpenMP comment directives (...):

Fortran: `!$OMP ...`

C: `#pragma omp ...`

Enables the macro named `_OPENMP`

`#ifdef _OPENMP` evaluates to true

(Fortraners: use F90 suffix for auto preprocessing)

Enables "hidden" statements (Fortran only!)

`!$...`

Exercises – Lab 1

- Exercise 1: Kernel check
f_kernel.f90/c_kernel.c
Kernel of the calculation (see exercise 2)
Parallelize one Loop
- Exercise 2: Calculation of π
f_pi.f90/c_pi.c
Parallelize one Loop with a reduction
- Exercise 3: daxpy ($a * x + b$)
f_daxpy.f90/c_daxpy.c
Parallelize one Loop

Exercise I: π Integration Kernel Check

- cd exercise_1
- Codes: f_kernel.f90/c_kernel.c
- Number of intervals is varied (Trial loop)

Kernel

Trial Loop: *itrial*
Calculation of *n* and *deltax*
Loop over *i*
make sure *area* >0.0

1 **Parallelize the code**

2 **Compile**

3 **Run with 1, 2, 4, 8 threads**

e.g. export OMP_NUM_THREADS=4
./a.out

4 **Compare the timings**

1 Parallelize the Loop over *i* :
Use **omp parallel do/for**
Set appropriate variables to private

2 Compile with:
ifort -openmp f_kernel.F90
icc -openmp c_kernel.c

(This loads up the c math lib for “sqrt” function.)

- ✓ Timings decrease with more threads.
- ✓ If you execute with more threads than cores, the timings will NOT decrease. Why?

Exercise II: π Integration

- cd exercise_2
- Codes: f_pi.90/c_pi.c
- Number of intervals is varied (Trial loop)

π calculation

Trial Loop: i trial
Calculation of n and deltax
Loop over i

- Parallelize the code

1 Complete OpenMP statements

- Initialization
- omp get max threads
- omp get thread num

1 Parallelize the Loop over i :
Use **omp parallel do/for**
with the default(none) clause

2 Compile with:
make f_pi
or
make c_pi

3 Run with 1, 2, 4, 8 threads

e.g. export OMP_NUM_THREADS=4
./c_pi or ./f_pi

4 Compare timings

- ✓ Timings decrease with more threads
- ✓ What is the scale up at 4 threads?.

Exercise III: daxpy

- cd exercise_3
- Codes: f_daxpy.f90/c_daxpy.c
- Number of intervals is varied (Trial loop)

daxpy

Trial Loop: *itrial*
Loop over *i*

1 Parallelize the Loop over ***i*** :
Use **omp parallel do/for**
with the default(none) clause

2 Compile with:
make f_daxpy
or
make c_daxpy

3 Run with 1 and 4

4 Compare timings

- Why is performance only doubled?

- Parallelize the code

1 complete OpenMP statements

- Initialization

- **omp get max threads**

✓ Hint: Parallel performance can be limited by memory bandwidth– what is happening for every daxpy operation? (Is there cache reuse?)