Guide: OpenMP on AWS

Ignacio M. Llorente

v3.0 - 1 Mar 2021

Abstract

This is a guideline document to show the necessary actions to set up and use `gcc` to evaluate its OpenMP support on Ubuntu (18.04).

Requirements

- **First you should have followed the Guide "First Access to AWS"**. It is assumed you already have an AWS account and a key pair, and you are familiar with the AWS EC2 environment.
- The files needed to do the exercises are available for download from **Canvas**.

Acknowledgments

The author is grateful for constructive comments and suggestions from David Sondak, Charles Liu, Matthew Holman, Keshavamurthy Indireshkumar, Kar Tong Tan, Zudi Lin, Nick Stern and Hayoun Oh.

1. Spin up EC2 instance

- We strongly recommend an instance with at least 4 vCPUs to be able to evaluate parallel implementation. The results in this guide have been obtained on a t2.2xlarge instance with 8 vCPUs, which is the instance type recommended in the homework assignment.
- Use Ubuntu 18.04

2. Install gcc

- Install `gcc` via the toolchain PPA

```
$ sudo apt-get install software-properties-common
$ sudo add-apt-repository ppa:ubuntu-toolchain-r/test
$ sudo apt-get update
$ sudo apt-get install gcc
```

- To check the `gcc` installation is successful run following command in the terminal

```
$ gcc -v
```

3. Verify OpenMP Support

This section includes a simple session aimed at verifying the OpenMP support provided by the gcc compiler.

- Use `lscpu` to visualize the number of CPUs and cores of the system.

```
$ lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                8
On-line CPU(s) list:   0-7
Thread(s) per core:    1
Core(s) per socket:    8
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 63
Model name:            Intel(R) Xeon(R) CPU E5-2676 v3 @ 2.40GHz
Stepping:              2
CPU MHz:               2400.072
…
```

- Upload to the VM the `omp_sc.c`, compile it with the `-fopenmp` flag, and run the code with different numbers of cores.

```
$ gcc -fopenmp omp_sc.c -o omp_sc
$ export OMP_NUM_THREADS=8
$ time ./omp_sc
```

- Take a quick look at `omp_sc.c` to understand why adding threads seems to increase runtime.
- Upload to the VM the `omp_mm.c` code with the OpenMP parallelization of `seq_mm.c`, compile it with the `-fopenmp` flag, and run the code with a growing number of cores.

```
$ ulimit -s 64000

$ gcc -O3 -fopenmp omp_mm.c -o omp_mm_O3
$ export OMP_NUM_THREADS=1
$ time ./omp_mm_O3 > output

real   0m9.406s
user   0m9.326s
sys    0m0.080s

$ export OMP_NUM_THREADS=2
$ time ./omp_mm_O3 > output

real   0m5.970s
user   0m8.991s
sys    0m0.057s

$ export OMP_NUM_THREADS=4
$ time ./omp_mm_O3 > output

real   0m4.877s
user   0m11.987s
sys    0m0.057s

$ export OMP_NUM_THREADS=8
$ time ./omp_mm_O3 > output

real   0m3.260s
user   0m9.324s
sys    0m0.068s
```

There are two important considerations from previous results:

- An OpenMP program in one thread runs slower than its sequential version, because the parallelized version introduces an overhead associated with the setup of the runtime environment and the creation of the thread. Moreover the compiler may not be able to as aggressively optimise the parallel code as the serial code.
- In order to measure times we must use real time and not cpu time, which adds the time consumed

by the process in all CPUs. See that CPU times are the same for any number of threads.
- This code ends with a write to file part that limits the speedup (Amdahl law). In our particular case this sequential part takes 1.8 seconds approximately. If we only consider the parallel part we achieve a linear speedup.

4. Automatic Parallelization

gcc brings a simple automatic parallelization:

- Upload `seq_mm.c`, `timing.c`, `timing.h` to the VM
- Use the automatic parallelization flag `-ftree-parallelize-loops=8` to generate a parallel version of `seq_mm.c`

```
$ gcc -O3 -DUSE_CLOCK -ftree-parallelize-loops=8 seq_mm.c  timing.c -o seq_mm_ap
$ time ./seq_mm_ap > output
```

```
real   0m3.608s
user   0m12.431s
sys    0m0.082s
```

Stop your instances when are done for the day to avoid incurring charges
Terminate them when you are sure you are done with your instance