P. Koumoutsakos
ETH Zentrum, CLT E 13
CH-8092 Zürich

Fall semester 2018

Set 1 - Roofline Model and Performance Measures

Issued: September 28, 2018
Hand in (optional): October 5, 2018 10:00am

Question 1: Setting up your local environment

This exercise will guide you through the setup of your local development environment. The procedure described here is only for Linux machines because the cluster also runs Linux. On the ETH computers, everything is already installed. Follow the following steps:

1. Open a terminal (press windows key and type `terminal`, then click the icon)
2. Clone the repository with the following command:
   `$ git clone https://gitlab.ethz.ch/hpcse18/lecture.git`
   Tip: You can still use copy paste in the terminal: Ctrl+Shift+C to copy from terminal, Ctrl+Shift+V to paste to the terminal.

This creates a directory named `lecture` inside your home folder. Its contents are those of the repository. You're done setting up your machine!

If the repository gets updated, you can get the latest changes with the following command from within the `lecture` directory:
`git pull`

Further information about git:
Git Handbook by Github (10min)
Git cheat sheet by Github

Question 2: Getting started on Euler

Euler is the new computing cluster of ETH Zurich, an evolution of the Brutus cluster. The cluster works with a queueing system: you submit your program with its parameters (called job) to a queue and wait for it to be finished. Your first task consists of the following steps:

1. **Creating an account**
   Euler accounts are created automatically when a user logs in for the first time. You will need to enter your nethz username and password. Euler is only accessible within the ETH Network.[1]

[1] You may use VPN `https://sslvpn.ethz.ch` to connect to the ETH Network from home.

2. **Login**
   Login from within the ETH network on Euler via ssh:
   `$ ssh username@euler.ethz.ch`
   and insert your password when asked to.

   Congratulations! You are now on a login node of the Euler cluster. In this environment you can write code, compile and run small tests. Keep in mind that there are other people working on the same nodes, so be mindful of how you use them!

3. **Modules**
   The Euler environment is organized in modules, which are conceptually software packages that can be loaded and unloaded as needed. The basic commands to use the module system are:
   `$ module load <modulename>`: this command sets the environment variables related to the specified module.
   `$ module unload <modulename>`: this commands unsets the environment variables related to `<modulename>`.
   `$ module list`: lists all the modules currently loaded.
   `$ module avail`: outputs a list of all the modules that can be loaded.

   If, for example, we need to compile a program with the GNU compiler (gcc or g++), we first load its module with
   `$ module load gcc`
   and then proceed with the compilation of our program:
   `$ g++ -O2 main.cpp -o program_name`

4. **Submitting a job**
   Performance measurements and long computations should not be performed on the login nodes but rather they should be submitted to the queue.
   To submit a simple job to the queue, you can use the following command from the folder where your program is stored:
   `$ bsub -n 24 -W 08:00 -o output_file ./program_name program_args`

   This command will submit a job for your executable `program_name` with arguments `program_args` by requesting 24 cores from a single node and a wall-clock time of 8 hours, after which, if the job is not already finished running, it will be terminated. The report of the job, along with the information that would usually appear on the terminal, will be appended in the file `output_file`, in the folder from where the job started.

   While your job is running you can always use the command:
   `$ bjobs`
   to get the status and IDs of your jobs.

   In order to terminate a job you can use the command:
   `$ bkill <jobid>`

Info: I/O performance and `$SCRATCH`:
Since your simulations might involve a lot of I/O (input/output), you must never run your simulations in your `$HOME` directory, but setup the runs in your `$SCRATCH` space. The disks associated with this space are designed for heavy loads[2]. Lastly, your quota in `$HOME` is much

smaller compared to `$SCRATCH`. However, please note that the memory in `$SCRATCH` is temporary and **any files older than 15 days are deleted automatically** (see `$SCRATCH/__USAGE_RULES__`). Follow the links below for more information on the Euler cluster.
Use the following command to change to your scratch directory:
`$ cd $SCRATCH`

Additional information on the Euler cluster, its instruments and on how to use it can be found at:

- https://scicomp.ethz.ch/wiki/Euler
- https://scicomp.ethz.ch/wiki/User_documentation

Question 3: Operational Intensity

The purpose of a compute architecture is to perform a certain operation on data. This requires a mechanism to send data along the memory hierarchy[3] to the execution units such that they can perform the operation(s).

The operational intensity is a measure that relates the amount of work W (operations) to the number of bytes Q (data) that need to be transferred and is defined as

$$I = \frac{W}{Q} \quad [\text{ops/byte}]. \tag{1}$$

The operational intensity I is used to characterize a code and reveals information about the expected utilization of the architecture's execution units and memory bandwidth.
In practice, operations are defined by floating point operations (FLOP) and memory transfer is defined by access from DRAM to the closest processor cache.
Hint: Remember that double precision numbers are typically stored in 8 bytes.

Your tasks

a) Determine the asymptotic bounds on the operational intensity $I(n)$ for the following matrix/vector operations, where n is the dimension of the vector. State your assumptions.

1. DAXPY: $y = \alpha x + y \quad \alpha \in \mathbb{R}; x, y \in \mathbb{R}^n$
2. SGEMV: $y = Ax + y \quad x, y \in \mathbb{R}^n; A \in \mathbb{R}^{n \times n}$
3. DGEMM: $C = AB + C \quad A, B, C \in \mathbb{R}^{n \times n}$
   Hint: Assume that A, B and C fit into the cache at the same time.

b) Consider the 1D diffusion equation in a periodic domain with length L

$$u_t(x,t) = \alpha u_{xx}(x,t), \tag{2}$$

$$u_0(x) = u(x,0) = \sin\left(\frac{2\pi}{L}x\right), \tag{3}$$

[2]However, `$SCRATCH` is not designed for frequent storing. If you are logging temporary results into a file, open the file once at the beginning, and do not flush e.g. more than one per second (or per minute). Related to it, note that `std::endl` not only prints the newline character `'\n'`, but also flushes the stream.

[3]https://en.wikipedia.org/wiki/Memory_hierarchy

where $0 \leq x < L$, $t > 0$ and $\alpha > 0$ is the diffusion coefficient. You are asked to solve this problem numerically by using a second order centered finite difference scheme and the explicit Euler method to advance in time. The domain is discretized with N uniformly spaced grid points. Discretization of Equation (2) leads to

$$u_i^{n+1} = u_i^n + \frac{\Delta t \alpha}{\Delta x^2} \left(u_{i-1}^n - 2u_i^n + u_{i+1}^n \right), \tag{4}$$

where $u_i^n \approx u(x_i, t^n)$ is the approximate solution with $t^{n+1} = t^n + \Delta t$ and $x_i = i\Delta x$. The constant time step size and uniform grid spacing are denoted by Δt and Δx, respectively. Determine the operational intensity $I(N)$ for the scheme given in Equation (4). Can you identify a possible hardware bottleneck for this scheme? State your assumptions.

Question 4: Roofline Model

The roofline model[4] is a simple visual tool that can be used to understand performance on a hardware architecture. It relates the nominal peak performance π and the nominal peak bandwidth β of a given hardware and introduces the concept of operational intensity, studied in the previous question.

Understanding the characteristics of the platform on which performance tests are executed is of fundamental importance since it gives a context in which to read performance results. The primary objective of this exercise is to learn how to characterize computing hardware performance.

Peak Performance and System Memory Bandwidth

The number of executed floating point operations (FLOP) is a measure used to characterize the costs of scientific software, e.g. computational fluid dynamics codes, structural mechanics, computational chemistry and computational biology packages.

The peak floating point performance, hereafter simply called peak performance, is a measure of the quantity of FLOP that a machine can execute in a given amount of time. Typically, the peak floating point performance π in FLOP/s can be computed as in the following:

$$\pi\ [FLOP/s] = f\ [HZ = cycle/s] \times c\ [FLOP/cycle/lane] \times v\ [lanes] \times n\ [-], \tag{5}$$

where f is the core frequency in CPU cycles per second (given in Hz), c the number of FLOP executed in each cycle, v the SIMD width (in number of elements that fit into the register size, also called SIMD *lanes*) and n the number of cores.

The exact values for the above features can be found at the hardware specifications or given by the system administrators. For example, for the Euler cluster we can find the specifications of the Intel Xeon E5-2680v3 here: https://ark.intel.com/products/81908/. Additional information is provided at the Euler wiki: https://scicomp.ethz.ch/wiki/Euler.

Typical floating point performance values are reported in GFLOP/s or TFLOP/s. Floating point performance is also used to assess the performance of a given algorithm implementation (http://en.wikipedia.org/wiki/FLOPS).

The bandwidth of the system memory is a measure of the speed of data movement from the system to caches and vice-versa. As the problems considered herein fit on a single node, we are mostly interested in quantifying the DRAM bandwidth.

[4] Williams et al, 2009: https://dl.acm.org/citation.cfm?id=1498785

Given the specifications of the DRAM memory, the theoretical memory bandwidth β in B/s can be computed as follows:

$$\beta\ [B/s] = f_{DDR}\ [Hz = cycle/s] \times c\ [channel] \times w\ [bit/channel/cycle] \times 0.125\ [B/bit], \quad (6)$$

where f_{DDR} is the DDR clock rate, c the number of memory channels and w the bits moved through a channel per cycle (typically 64 bits). Typical bandwidths are reported in MB/s, GB/s or TB/s (http://en.wikipedia.org/wiki/Memory_bandwidth).

Memories based on the DDR (Double Data Rate) technology, such as DDR-SDRAM, DDR2-SDRAM, and DDR3-SDRAM[5], transfer two units of data per clock cycle. As a result, they achieve double the transfer rate compared to traditional memory technologies (such as the original SDRAM) running at the same clock rate. Because of that, DDR-based memories are usually labeled with double their real clock rate. For example, DDR3-1866 memories actually work at 933 MHz transferring two units of data per clock cycle, and thus are labeled as being a "1866 MHz" device, even though the clock signal does not really work at 1.866 GHz.

Technological advances in the past dictate that the rate π at which operations can be executed doubles roughly every 18 months (Moore's Law), while the rate β at which data can be transported doubles roughly every 36 months.

Your task

a) According to the wiki of the Euler cluster, each of the two sockets on a node hosts an Intel Xeon E5-2680v3 12-core Haswell CPU capable of delivering 480 GFLOP/s each. In addition, the theoretical memory bandwidth for each of the two sockets can reach 68.3 GB/s. Try to justify the above numbers.

b)
1. What is the value of the operational intensity I_b for which the hardware is operated in balance, i. e. both the CPU and the memory bandwidth are optimally utilised? What is the value in numbers for a full node on Euler?
2. Give an expression for the maximum performance $P_{\text{peak}}(I)$(in FLOP/s) a piece of software with operational intensity I can reach. This expression should be a function of I that depends only on properties of the hardware.
3. The operational intensity I allows to classify codes into two groups: *memory bound* codes and *compute bound* codes. The classification depends on the used hardware! How can one determine from the roofline plot if a code is memory or compute bound? What implications does this have on the optimizations we will choose?

c) Draw the roofline for a full NUMA node of Euler.

d) Write a small C/C++ code that implements Equation (4). Choose a suitable number of grid points N, where $L = 1000$ and $\alpha = 10^{-4}$. For stability reasons, you must choose your time step such that

$$\Delta t < \frac{\Delta x^2}{2\alpha}. \quad (7)$$

Benchmark the diffusion kernel on one Euler node. Measure multiple kernel executions (each is on full iteration over the grid) to obtain an average time for your measurement. Use the

[5] http://en.wikipedia.org/wiki/DDR3_SDRAM

operational intensity calculated in Question 3b and indicate the measured performance in the roofline plot. Is your implementation memory bound or compute bound? What is the maximum performance that your code can reach according to the model? Explain issues in case you are not able to fully utilize the hardware.

Hint: Assume periodic boundary conditions and take care of the first and the last temporal iteration (indices...)!

Hint: Flush the cache between time measurements to ensure that you always read the data from DRAM! (We should do this because our benchmark size is small and will benefit from the cache otherwise.)

e) (Optional). The Swiss National Supercomputing Centre (CSCS) in Lugano is home to the Piz Daint supercomputer. The machine reaches a performance of 19.6 PFLOP/s based on the Linpack benchmark and is currently the sixth fastest computer in the world (https://www.top500.org/lists/2018/06/, as of June 2018). Piz Daint is composed of Cray XC50 compute nodes which are hybrid nodes with GPU accelerators. Go to http://www.cscs.ch/computers/piz_daint/index.html and find out the specifications of the GPUs. Based on the numbers you have researched, draw the roofline for the GPU. Compare this roofline with the one you have drawn in part 4c. Describe the differences you observe in terms of how it would impact you as application developer.