Lab Guide 0

Support material for this session

Objectives:

- Write a basic matrix multiplication code in C
- Get accustomed with the lab session environment (`gcc` compiler, Linux, `perf` tool, cluster access)
- Understand code profiling basis (sampling-based approaches)

1. Simple Matrix Multiplication in C (see annex 1 if you need help)

Write a simple C code to implement a matrix multiplication with `double` data type. Assume that all three matrices are of equal size (size N) and square. Also include a simple function for matrix initialisation.

In order to get a more detailed profile (and a better program structure), write different C functions for each step (e.g., initialization, matrix multiplication, result validation, etc.).

2. Compile and run the program on the search cluster

Copy the C code to the search cluster using `scp`, login on the cluster with `ssh` and compile the code using `gcc` (see Annex 2, steps a) to d)).

To run the program on a compute node use the `srun` command (note: this will reserve one of the 20-core compute nodes and run the `a.out` binary of the current directory on that node):

```
srun --partition=cpar ./a.out
```

3. Code profiling and performance evaluation with the Linux perf tool

Measure the time required to execute the code. One option would be to place `time` before the previous `srun` command, but this is not accurate, why?

The `perf stat [-e XXX] ./a.out` command gets [low-level] performance metrics over the entire program (the `-e XXX` option can be used to specify a specific set of metrics, see `perf list` for a complete list of available metrics). Run the following command to get a set of "default" metrics:

```
srun --partition=cpar perf stat ./a.out
```

To obtain a detailed application profile with `perf`, run the application with `perf record ./a.out` to sample the program execution at fixed time intervals (4000 samples per second, by default). The profile data is written into a file named `perf.data`. (**Note**: `perf` requires privileged access to some machine resources, so it is only available on some compute nodes, e.g., `cpar` partition).

The `perf record` command should be placed on a script file and executed with the `sh` command:

```
[xxx@search7edu yyy]$ cat perf.sh
#!/bin/sh
perf record -g -o perf.data ./a.out
[xxx@search7edu yyy]$ srun --partition=cpar sh perf.sh
[ perf record: Woken up 11 times to write data ]
[ perf record: Captured and wrote 2,624 MB perf.data (34005 samples) ]
[xxx@search7edu yyy]$
```

Use `perf report` to generate and view a profile with the application hotspots.

To get a better profile and more accuracy, the code (C program) should be compiled with these two additional flags: `-g -fno-omit-frame-pointer`.

Annex 1 - Matrix Multiplication Algorithm

(see https://en.wikipedia.org/wiki/Matrix_multiplication for more information)

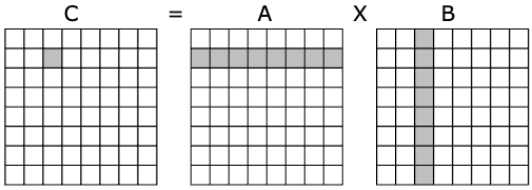
There are many alternatives to code a matrix multiplication. In these lab sessions we will use, as a base implementation, a variant with three nested cycles, stating with the most common implementation, that we call ijk variant (latter we will call it the DOT variant). The pseudo-code of square matrices A, B, C with size N is:

```
for(int i=0; i<N; i++)
  for(int j=0; j<N; j++)
    for(int k=0; k<N; k++)
      C[i][j] += A[i][k] * B[k][j]; // note: assumes that the matrix C was initialised with 0s
```

On a strict sequential execution, any order of the i j and k, in the previous code, produces a correct result, but there are noticeable performance differences among variants. This first variant (ijk) will be called DOT, since each element of the C matrix results from the ***dot*** *product* of one line of A (first vector) with a column of B (second vector), as it is illustrated by the following expression/algorithm and figure:

$$C_{ij} = DOT_{linha\_A_i, coluna\_B_j} = \sum_{k=0}^{n-1}(A_{ik} * B_{kj})$$

```
Foreach line of A
  Foreach column of B
    C line,column = DOT line of A,column of B
```



Annex 2: (simple) Instructions for using the SeARCH cluster

In this course we will use compute nodes from the University SeARCH cluster, more specifically two nodes with 20 cores, organised into a partition called "`cpar`". To execute the code of this lab session (and the following lab sessions) you should login into the cluster front-end using ssh (`ssh <<id…>>@s7edu.di.uminho.pt`) and provide the password received by email. The matrix multiplication code can be compiled on the cluster front-end, but must be executed on an available node by running the executable with `srun`.
A source file can be copied to the cluster front-end with `scp`. All editing and compilation can be performed on the front-end, but execution should be performed on a compute node (**please, never execute any code on the front-end**). Note: during the semester other front-ends might be provided (e.g., `s7edu2.di.uminho.pt`) to provide a better response time.

a) **Copy local file to remote machine (don't forget the two points at the end) :**

```
scp <local file name> <student_id>@s7edu.di.uminho.pt:
```

b) **Login:** `ssh <student_id>@s7edu.di.uminho.pt`

c) **Load the gcc environment:** `module load gcc/11.2.0`

d) **Compile:** `gcc …`

e) **Run perf profiling:** `srun --partition=cpar perf stat -e instructions,cycles <<full_path>>/a.out`

Annex 3. Perf installation on Ubuntu (follow similar steps for other Linux distributions)

Install these packages:

```
sudo apt install linux-tools-common
sudo apt install linux-tools-5.15.0-48-generic  # update for your kernel version
```