**Design of Parallel and High Performance Computing**
*HS 2019*
*Tal Ben-Nun, Markus Püschel*
*Department of Computer Science*
*ETH Zurich*

**Homework 5**

# Performance Modeling

## Little's Law

Imagine you want to board a train which leaves in 20 minutes. But before you have to buy the train ticket at a counter. You see that there are about 50 people in line before you. Serving a customer takes 40 seconds on average.

What property has to hold for this system to be stable? Will you miss your train?

## Solution

A system is stable if the arrival rate is equal to the departure rate. Then the equality $\alpha\beta = N$ holds, where $N$ is the number of things in the system, $\alpha$ is the amount of time between entering and leaving the system (latency), and $\beta$ is the arrival rate.

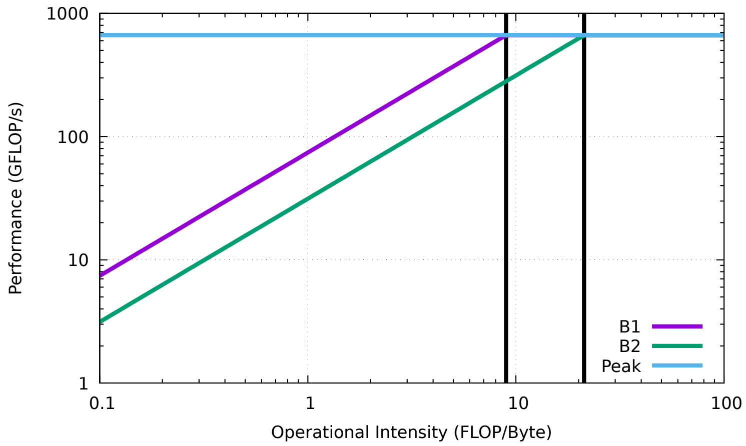
In our example $N = 50$, $\beta = \frac{1}{40s}$. Therefore $\alpha = 2000s \approx 33min$.

## Roofline Model

Assume a NUMA architecture with 2 nodes and a peak memory bandwidth of $B_1 = 74.2$ GB/s. Every node has 4 cores and can carry out up to $P = 332.8$ GFLOPs/sec. Each core runs at 2.6 GHz. If the memory accesses are not well balanced, the peak memory bandwidth becomes $B_2 = 31.32$ GB/s. Draw a roofline plot for this processor. If a program and input combination land on the lower left of the plot, what does this tell you about the program?

Will all program executions yield points which lie either on the diagonal or on the "roof" of the roofline plot?

## Solution




If a measurement result lies on the left side of the diagram and on the roofline (drawn in black) it is bound by memory bandwidth. If it lies on the right side of the roofline it is bound by computational bandwidth. If it lies below the roofline, it is bound by something else i.e., integer performance, which is not represented in the (simple) roofline model.

## Sparse Matrix Vector Multiplication SpMV

The following code compute a Sparse Matrix Vector Multiplication $\overrightarrow{y} = A \cdot \overrightarrow{x}$ between a matrix A (sparse) and a vector $\overrightarrow{x}$ (dense). The matrix is stored in the Compressed Row Storage format.

```
<fill data structures: blockptr, values, col_idx, row_start>

#pragma omp parallel private(i,j,is,ie,j0,y0,thread,bs,be)

thread = omp_get_thread_num()

//Compute the block boundaries
bstart = blockptr[thread]
bend = blockptr[thread+1]

for (i=bstart; i<bend; i++){
  y0=0
  row_start = row_start[i]
  next_row_start = row_start[i+1]

  for (j=row_start; j<next_row_start; j++){
    j0 = col_idx[j]
    y0 += value[j] * x[j0]
  }

  y[i] = y0
}
```

Assume that $\overrightarrow{x}$ and $\overrightarrow{y}$ are kept in cache. The CSR format uses 4byte `integers` to store column indexes. Values are stored using 8byte `doubles`. Compute the operational intensity and check if the code is memory- or compute-bound w.r.t. the previously described architecture (consider only the innermost loop).

You run this code, observing that it reaches a performance up to $5.22$ GFLOPS, and you notice that this is mostly due to how the array `value` is stored. Describe an optmization that you can apply to improve the performance.

## Solution

In the innermost loop we have 2 flops every 12bytes of data, hence the operational intensity is $O = 1/6$ Flop/byte. The code is memory bound.

The array elements that are initialized by a thread running on a core of a specific NUMA node are allocated in the memory of that node. We need that all the cores "touch" the memory that they are going to access before it is initialized by the master thread. This can be done by performing the initialization in the following way:

```
#pragma omp parallel for
for (int i=0; i<nnz; i++){
  value[i] = 0.0
}
```