Exercise 1.5.9

Let R be an $n \times n$ upper-triangular matrix with semiband width s. Show that the system $Rx = y$ can be solved by back substitution in about $2ns$ flops. An analogous result holds for lower-triangular systems.

$$\begin{bmatrix} a_{11} & a_{12} & ... & a_{1s} & 0 & ... & 0 \\ 0 & a_{22} & a_{23} & ... & a_{2s} & ... & 0 \\ 0 & 0 & a_{33} & a_{34} & ... & ... & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{bmatrix}$$

Proof:

First step:

$$x_n = b_n / a_{nn}$$

1 - flop, 2 - reads, 1 - write

Next step would be:

$$x_i = (b_i - \sum_{j=i+1}^{n} (a_{ij} * x_j))/a_{ii}$$

but, we now that from $a_{i,j+s+1}$ to a_{in}, all entries are zero because the matrix A is banded with a semi-band of s. This will reflect in the sumation, so, in this case:

$$\begin{aligned} \sum_{j=i+1}^{n} (a_{ij} * x_j) &= \sum_{j=i+1}^{i+s} (a_{ij} * x_j) + \sum_{j=i+s+1}^{n} (0 * x_j) \\ &= \sum_{j=i+1}^{i+s} (a_{ij} * x_j) \end{aligned}$$

This will generate $(i+s) - (i+1) + 1 = i + s - i - 1 + 1 = s$ iterations per line. Each iteration of the summation will perform:
1 flop, 3 reads (we will read the accumulator), 1 write (updating the accumulator). Totalling: s flops, $3s$ reads, s writes.

Plugging back in the original expression for line i we have: (+1)Read b_i, (+1)flop $b_i -$ acc, (+1)read a_{ii}, (+1)flop $/a_{ii}$, (+1)write x_i. In total we have: $s+2$ flops, $3s+2$ reads, $s+1$ writes for each line (worst case). Now we can calculate the cost for the whole matrix, just assuming (incorrectly) that each line takes the worst case and arrive at:
$n * (s+2) = ns + 2n$ flops,
$n * (3s+2) = 3ns + 2n$ reads,
$n * (s+1) = ns + n$ writes.
In this banded matrix scenario s ¡¡ n and it is not an absurd to approximate s 2 that would gives us the $2ns$ flops of the exercise.

□

But to better understand the cost of the algorithm we will need to reference Agner4

```
Latency:
This is the delay that the instruction generates in a dependency chain.
The numbers are  minimum values. Cache misses, misalignment, and
exceptions may increase the clock counts considerably. Where
hyperthreading is enabled, the use of the same execution units in the
other thread leads to inferior performance. Denormal numbers, NAN's
and infinity do not increase the latency. The time unit used is core
clock cycles, not the reference clock cycles given by the time stamp
counter.

Throughput:
The average number of core clock cycles per instruction for a series of
independent instructions of the same kind in the same thread.

Instruction Operand  Latency Throughput
FADD        Register    3       1
FSUB        Register    3       1
FADD        Memory      5       1
FSUB        Memory      5       1
FMUL        Register    5       1
FDIV        Register    16      5
FDIV        Memory      16      5

http://www.agner.org/optimize/instruction_tables.pdf
```

What we must analyse is that these are best-case scenarios. As long we have cache misses these values can,and will increase enormously. We can see the cost of cache misses, for example, at:

Level	Fastest Latency
First Level Data	4 cycles
Second Level	12 cycles
Third Level	44 cycles

https://www.intel.co.uk/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf

Tehese, again, are the best scenario. And we are not event touching the main system meory yet.
Memory access is very difficult to stimate, but we will use the value used at:

Data Source	Latency
L3 CACHE hit, line unshared	~40 cycles
L3 CACHE hit, shared line in another core	~65 cycles
L3 CACHE hit, modified in another core	~75 cycles
remote L3 CACHE	~100-300 cycles
Local Dram	~60 ns
Remote Dram	~100 ns

https://software.intel.com/sites/products/collateral/hpc/vtune/performance_analysis_guide.pdf

So, I think it is safe to say that today, the algorithm will be dominated by its memory access pattern, that is why we are also calculating read, and write patterns.