

Sparse neural networks

1 Problem description

We are interested in the parallelization of the inference phase of a neural network composed of multiple sparse layers of constant size. With each layer $l = 0 \dots L - 1$ of the network are associated n neurons (same number for all layers) each having a value nv and m_l synapses where a synapse (i, j, sv) updates neuron j of layer $l + 1$ with neuron i of layer l using a value sv through the update function.

The provided code contains an array called `layers` of size equal to the number of layers L .

- For each layer l , the element `layers[l]` contains
 - its neurons in an array `layers[l].neu` of size n ,
 - its synapses in an array `layers[l].syn` of size m_l
 - and the value of m_l in `layers[l].m`.
- For each neuron i in layer l its value is in `layers[l].neu[i].nv`.
- For each synapse s in layer l the (i, j, sv) values are in `layers[l].syn[s].i`, `layers[l].syn[s].j` and `layers[l].syn[s].sv`, respectively.

The inference is achieved through the following code in the `sequential_nn` routine

```
for(l=0; l<L-1; l++){
    for(s=0; s<layers[l].m; s++){
        i = layers[l].syn[s].i;
        j = layers[l].syn[s].j;
        layers[l+1].neu[j].nv += update(layers[l].neu[i].nv, layers[l].syn[s].sv);
    }
}
```

For all layers $l = 0, \dots, L - 1$ and for all synapses $s = 0, \dots, m_l - 1 = \text{layers}[l].m - 1$, the code updates neuron j of layer $l + 1$ with neuron i of layer l using the instruction

```
layers[l+1].neu[j].nv += update(layers[l].neu[i].nv, layers[l].syn[s].sv);
```

Please pay attention to the `+` in the `+=` operator.

2 Package content

In the `neural_network` directory you will find the following files:



- `main.c`: this file contains the main program which first calls the `init_data` routine which initializes with random values all the neurons and all the synapses for all the layers. The main program then calls the `sequential_nn` routine executes the above code, the `parallel_nn_loops` and `parallel_nn_tasks` routines which contain the parallelized code to be implemented. After these calls the main program checks that the output of the parallel routines is the same as that of the sequential one. **Only this file has to be modified for this exercise.**
- `aux.c`, `aux.h`: these two files contain auxiliary routines and **must not be modified.**

The code can be compiled with the `make` command: just type `make` inside the `sparse_nn` directory; this will generate a `main` program that can be run like this:

```
$ ./main n L
```

where `n` is the size (number of neurons) of the layers and `L` the number of layers in the network.

3 Assignment

-  At the beginning, the `parallel_nn_loops` and `parallel_nn_tasks` are a copy of `sequential_nn`. Modify these routines in order to parallelize them. The first **must** be parallelized using the `omp for` directive and the second **must** be parallelized using the `omp task` directive. Make sure that the computed result is correct.
-  Report the execution times for the implemented parallel versions with 1, 2 and 4 threads and compare them with the sequential routine. What speedup could you achieve? analyze and comment on your results. Report your answer in the form of comments at the bottom of the `main.c` file.

Advice

- When using the OpenMP `task` construct, always think about data scoping to make sure input data has the correct value upon execution of the task and returned results do not go out of scope when the task is finished.
- Note that the `depend` clause allows you to specify dependencies due to access to single coefficients of an array. For example:

```
#pragma omp task depend(inout:x[i])
    x[i] += 1;
```

- Choose small values for n , and L when developing (for example, $n = 4$, $L = 2$) and big ones when evaluating performance (for example, $n = 15$, $L = 10$).