auto-parallelism

April 9, 2019

1 Automatic Parallelism

```
In [1]: import d2l
        import mxnet as mx
        from mxnet import nd
```

1.1 Parallel Computation using CPUs and GPUs

```
In [2]: def run(x):
            return [nd.dot(x, x) for _ in range(10)]

        x_cpu = nd.random.uniform(shape=(2000, 2000))
        x_gpu = nd.random.uniform(shape=(6000, 6000), ctx=mx.gpu(0))
```

Then, use the two NDArrays to run the `run` function on both the CPU and GPU and print the time required.

```
In [4]: run(x_cpu)  # Warm-up begins.
        run(x_gpu)
        nd.waitall()  # Warm-up ends.

        with d2l.Benchmark('Run on CPU.'):
            run(x_cpu)
            nd.waitall()

        with d2l.Benchmark('Then run on GPU.'):
            run(x_gpu)
            nd.waitall()
```

```
Run on CPU. time: 0.3592 sec
Then run on GPU. time: 1.2187 sec
```

We remove `nd.waitall()` between the two computing tasks `run(x_cpu)` and `run(x_gpu)` and hope the system can automatically parallel these two tasks.

```
In [5]: with d2l.Benchmark('Run on both CPU and GPU in parallel.'):
            run(x_cpu)
            run(x_gpu)
            nd.waitall()
```

```
Run on both CPU and GPU in parallel. time: 1.2227 sec
```

As we can see, when two computing tasks are executed together, the total execution time is less than the sum of their separate execution times. This means that MXNet can effectively automate parallel computation on CPUs and GPUs.

1.2 Parallel Computation of Computing and Communication

In computations that use both the CPU and GPU, we often need to copy data between the CPU and GPU, resulting in data communication. In the example below, we compute on the GPU and then copy the results back to the CPU. We print the GPU computation time and the communication time from the GPU to CPU.

```
In [6]: def copy_to_cpu(x):
            return [y.copyto(mx.cpu()) for y in x]

        with d2l.Benchmark('Run on GPU.'):
            y = run(x_gpu)
            nd.waitall()

        with d2l.Benchmark('Then copy to CPU.'):
            copy_to_cpu(y)
            nd.waitall()
```

```
Run on GPU. time: 1.2269 sec
Then copy to CPU. time: 0.5156 sec
```

We remove the `waitall` function between computation and communication and print the total time need to complete both tasks.

```
In [7]: with d2l.Benchmark('Run and copy in parallel.'):
            y = run(x_gpu)
            copy_to_cpu(y)
            nd.waitall()
```

```
Run and copy in parallel. time: 1.2807 sec
```

As we can see, the total time required to perform computation and communication is less than the sum of their separate execution times. It should be noted that this computation and communication task is different from the parallel computation task that simultaneously used the CPU and GPU described earlier in this section. Here, there is a dependency between execution and communication: `y[i]` must be computed before it can be copied to the CPU. Fortunately, the system can copy `y[i-1]` when computing `y[i]` to reduce the total running time of computation and communication.

1.3 Summary

- MXNet can improve computing performance through automatic parallel computation, such as parallel computation using the CPU and GPU and the parallelization of computation and communication.

1.4 Problems

- 10 operations were performed in the `run` function defined in this section. There are no dependencies between them. Design an experiment to see if MXNet will automatically execute them in parallel.
- Designing computation tasks that include more complex data dependencies, and run experiments to see if MXNet can obtain the correct results and improve computing performance.
- When the computational load of an operator is small enough, parallel computation on only the CPU or a single GPU may also improve the computing performance. Design an experiment to verify this.

1.5 Discuss on our Forum