GPU Programming

Final Project Presentation

2017.06.26

Ming-Hsi Lee Yun-Chu Chen

Advisor : MoBagel

Outline

- Motivation
- Model Study
- Useful Packages
- Problems
- Results and Conclusions

Scikit-Learn Linear Model with GPU Support

- Motivation: sklearn has no GPU support

- LinearRegression(fit_intercept=True)
- Ridge(alpha=1.0, fit_intercept=True)

Train a model to…

- Predict the **Sales**, given some attributes

	A	B	C	D	E	F	G	H	I
1	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHolida	SchoolHoliday
2	1	5	2015/7/31	5263	555	1	1	0	1
3	2	5	2015/7/31	6064	625	1	1	0	1
4	3	5	2015/7/31	8314	821	1	1	0	1
5	4	5	2015/7/31	13995	1498	1	1	0	1
6	5	5	2015/7/31	4822	559	1	1	0	1
7	6	5	2015/7/31	5651	589	1	1	0	1
8	7	5	2015/7/31	15344	1414	1	1	0	1
9	8	5	2015/7/31	8492	833	1	1	0	1
10	9	5	2015/7/31	8565	687	1	1	0	1
11	10	5	2015/7/31	7185	681	1	1	0	1
12	11	5	2015/7/31	10457	1236	1	1	0	1
13	12	5	2015/7/31	8959	962	1	1	0	1
14	13	5	2015/7/31	8821	568	1	1	0	0
15	14	5	2015/7/31	6544	710	1	1	0	1
16	15	5	2015/7/31	9191	766	1	1	0	1
17	16	5	2015/7/31	10231	979	1	1	0	1
18	17	5	2015/7/31	8430	946	1	1	0	1
19	18	5	2015/7/31	10071	936	1	1	0	1
20	19	5	2015/7/31	8234	718	1	1	0	1

Brief Introduction to sklearn

```
class LinearRegression(LinearModel, RegressorMixin):
    """
    Ordinary least squares Linear Regression.
```

scipy.linalg.**lstsq**(*a, b, cond=None, overwrite_a=False, overwrite_b=False, check_finite=True, lapack_driver=None*)¶

Compute least-squares solution to equation Ax = b.

Compute a vector x such that the 2-norm `|b - A x|` is minimized.

```
def lstsq(a, b, cond=None, overwrite_a=False, overwrite_b=False,
          check_finite=True, lapack_driver=None):
    """
    Compute least-squares solution to equation Ax = b.

    Compute a vector x such that the 2-norm ``|b - A x|`` is minimized.
```

```
if sp.issparse(X):
    if y.ndim < 2:
        out = sparse_lsqr(X, y)
        self.coef_ = out[0]
        self._residues = out[3]
    else:
        # sparse_lstsq cannot handle y with shape (M, K)
        outs = Parallel(n_jobs=n_jobs_)(
            delayed(sparse_lsqr)(X, y[:, j].ravel())
            for j in range(y.shape[1]))
        self.coef_ = np.vstack(out[0] for out in outs)
        self._residues = np.vstack(out[3] for out in outs)
else:
    self.coef_, self._residues, self.rank_, self.singular_ = \
        linalg.lstsq(X, y)
    self.coef_ = self.coef_.T

if y.ndim == 1:
    self.coef_ = np.ravel(self.coef_)
self._set_intercept(X_offset, y_offset, X_scale)
return self
```

LAPACK (Linear Algebra PACKage)

- A library of Fortran 77 subroutines
- Designed to be efficient on numerical linear algebra
- For Linear Least Squares (LLS) Problems:
  - Driver routines:

Table 2.3: Driver routines for linear least squares problems

Operation	Single precision		Double precision	
	real	complex	real	complex
solve LLS using *QR* or *LQ* factorization	SGELS	CGELS	DGELS	ZGELS
solve LLS using complete orthogonal factorization	SGELSY	CGELSY	DGELSY	ZGELSY
solve LLS using SVD	SGELSS	CGELSS	DGELSS	ZGELSS
solve LLS using divide-and-conquer SVD	SGELSD	CGELSD	DGELSD	ZGELSD

Solve LLS ($Ax = b$)

- LinearRegression:
  - Directly:

$$x = (A^T A)^{-1} A^T b$$

  - Using QR:

$$A = QR \quad where\ Q^T Q = I$$

$$x = R^{-1} Q^T b$$

- Ridge:

$$x = (A^T A + \lambda\, I)^{-1} A^T b$$

PyCUDA

- Pythonic access to Nvidia's CUDA parallel computation API
- a_gpu = gpuarray.to_gpu(X)
- x = x_gpu.get()

- from pycuda.compiler import SourceModule

```
mod = SourceModule("""
  __global__ void add(float *a, float *b)
  {
    int idx = threadIdx.x + threadIdx.y * blockDim.x;
    a[idx] += b[idx];
  }
""")
```

Scikit-cuda Linear Algebra Routines

- Python interfaces to many of the functions in the CUDA
- linalg.transpose
- linalg.dot
- linalg.inv

Critical Problems

- CULA package is not accessible now
  - But most high-level routines are based on CULA library, ex. qr
  - Some functions have out-of-memory error with large data, ex. svd

- Inverse function (skcuda.linalg.inv) based on cusolver is slow

So finally…

- We solve the LLS problem directly
- And, let the inverse calculated on CPU

Function API

- cuModel = cu_model.cuLinearRegression()
- cuModel.fit(x_train, y_train)
- pred = cuModel.predict(x_test)
- cuModel.coef_ / cuModel.intercept_

Results and Comparisons (LinearRegression)

		1k	10k	10w	million
CPU	run time	0.001368	0.002355	0.01361	0.1656
	RMSE	1686.49	1571.41	1521.67	1478.04
GPU	run time	0.00388	0.01312	0.01347	0.04694
	RMSE	1686.69	1571.41	1521.67	1478.04

- Speed up when dataset is large enough, 3.5 times faster in this case

Results and Comparisons (Ridge)

		1k	10k	10w	million
CPU	run time	0.001275	0.002542	0.01346	0.1208
	RMSE	3108.52	2797.76	1690.2	1506.07
GPU	run time	0.024	0.034	0.02644	0.0712
	RMSE	3106.28	2756.80	1691.23	1506.1

- Speed up when dataset is large enough, 1.7 times faster in this case

Conclusions

- Most of the time in our cuda-model spent on data transfer… (CPU/GPU)
- More features, more speed-up gain
- Beneficial when dataset is large enough

References

- scikit-learn GitHub
  https://github.com/scikit-learn/scikit-learn
- scipy GitHub:
  https://github.com/scipy/scipy
- LAPACK—Linear Algebra PACKage:
  http://www.netlib.org/lapack/
- PyCUDA:
  https://documen.tician.de/pycuda/
- scikit-cuda:
  https://scikit-cuda.readthedocs.io/en/latest/