

Oaz Nir
May 16, 2008
18.337J

Studying Morphological Variance of Single Cell Populations: A Parallel Computing Framework

Introduction

I applied techniques of parallel computing, specifically Star-P, to set up a parallel framework for doing a computation that is part of my thesis research. In this report, I first give some background information on my research, so that the computation can be understood in context. Next, I describe the computation itself and the data that is used. Subsequently, I provide details of the serial and parallel approaches to performing this computation, and provide results for each.

The overarching problem is to infer signaling relationships between proteins of interest. One way to approach this problem is to knock out one gene at a time and perform observations of the resulting cells. In most cases, the observations have been of transcriptional levels averaged over a population of cells, obtained using microarrays. This data is then used to infer signaling relationships.

Instead of using transcriptional data, we use morphological data. We acquire the data using image processing software that we developed and then measure, for each single cell, a slate of morphological traits. The goal then is to use this morphological data to backwards-infer the signaling relationships. (In practice, good inference can only be accomplished by using morphological data in conjunction with transcriptional data, but here we concentrate on the morphological data.)

Data

We compiled quantitative measures of $F = 150$ morphological features for single cells in $N = 250$ gene-knock-out treatment conditions (TCs) using the *Drosophila* BG-2 cell line. Each TC had about $C = 50$ single cells, on average.

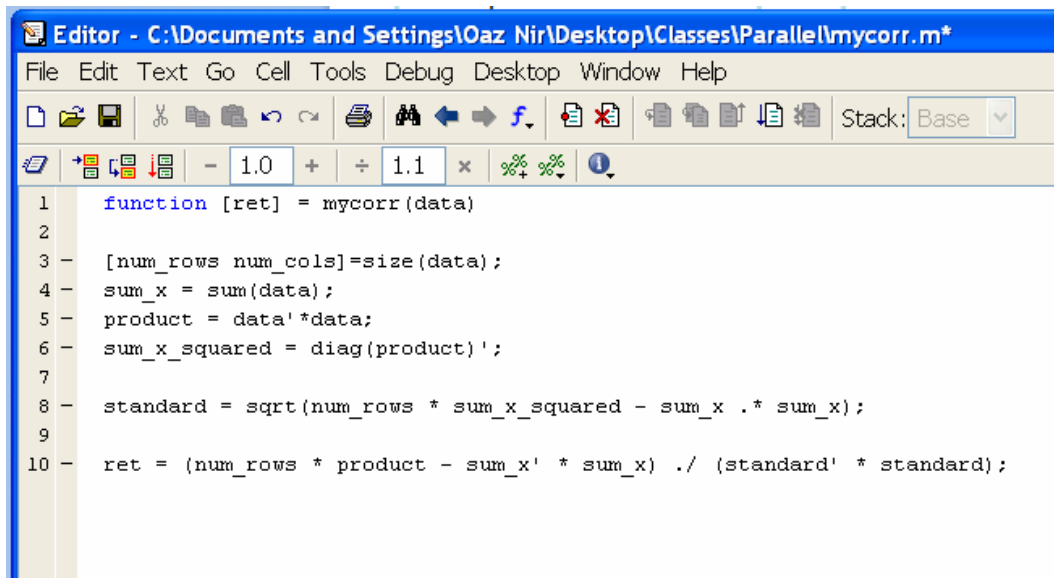
Computation

It is desirable to quantify the variability in the single cell populations that are obtained from each gene knock-out experiment. For each knock-out, we are interested in computing the linear correlations between each pair of morphological features taken across the population of single cells. This is a problem that naturally lends itself to a parallel formulation.

The linear correlation between points (x_i, y_i) , $i = 1, \dots, n$, is given by:

$$r_{xy} = \frac{\sum x_i y_i - n \bar{x} \bar{y}}{(n-1) s_x s_y} = \frac{n \sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n \sum x_i^2 - (\sum x_i)^2} \sqrt{n \sum y_i^2 - (\sum y_i)^2}}.$$

For a given TC, suppose we have data stored in the form of a CxF matrix (called “data”), where each row represents the morphological read-out for a single cell, and each column represents a feature. Then we are interested in computing the linear correlation between each pair of columns, i.e. a FxF matrix. The following function, called *my_corr*, accomplishes this.



The screenshot shows a MATLAB editor window titled "Editor - C:\Documents and Settings\Oaz Nir\Desktop\Classes\Parallel\mycorr.m*". The window has a menu bar (File, Edit, Text, Go, Cell, Tools, Debug, Desktop, Window, Help) and a toolbar with various icons. Below the toolbar is a stack of variables showing "Base". The main area contains the following MATLAB code:

```
1 function [ret] = mycorr(data)
2
3 [num_rows num_cols]=size(data);
4 sum_x = sum(data);
5 product = data'*data;
6 sum_x_squared = diag(product)';
7
8 standard = sqrt(num_rows * sum_x_squared - sum_x .* sum_x);
9
10 ret = (num_rows * product - sum_x' * sum_x) ./ (standard' * standard);
```

The dominant step comes in line 5, where multiplication of a FxC and a CxF matrix occurs.

We can then run this in serial on random data, i.e. a three dimensional array, `rand(50,150,250)`, for each 2D slice – or we can run it in parallel using Star-P. The results are as follows:

Serial

```
Command Window
1 To get started, select MATLAB Help or Demos from the Help menu.

>> clear ret
>> data=rand(50,150,250);
>> tic;for i=1:250
ret(i,1,i) = mycorr(data(i,1,i));
end;toc
Elapsed time is 13.98916 seconds.
>> data=rand(50,150,250);
>> tic;for i=1:250
ret(i,1,i) = mycorr(data(i,1,i));
end;toc
Elapsed time is 0.368716 seconds.
>> clear ret
>> data=rand(50,150,500);
>> tic;for i=1:500
ret(i,1,i) = mycorr(data(i,1,i));
end;toc
Elapsed time is 36.51039 seconds.
>> data=rand(50,150,500);
>> tic;for i=1:500
ret(i,1,i) = mycorr(data(i,1,i));
end;toc
Elapsed time is 0.817963 seconds.
>> clear ret
>> data=rand(100,150,250);
>> tic;for i=1:250
ret(i,1,i) = mycorr(data(i,1,i));
end;toc
Elapsed time is 13.877569 seconds.
>> data=rand(100,150,250);
>> tic;for i=1:250
ret(i,1,i) = mycorr(data(i,1,i));
end;toc
Elapsed time is 0.504176 seconds.
>>
```

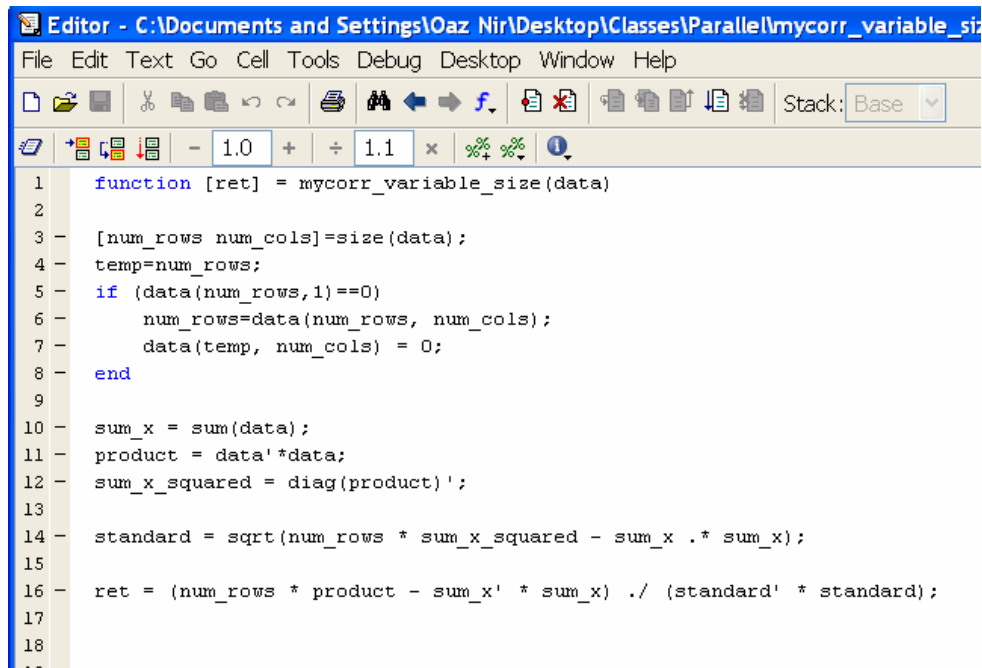
Parallel

```
Command Window
1 To get started, select MATLAB Help or Demos from the Help menu.

>> a=rand(50,150,250*p);
>> tic;ppeval('mycorr',a);toc
Elapsed time is 0.405599 seconds.
>> a=rand(50,150,250*p);
>> tic;ppeval('mycorr',a);toc
Elapsed time is 0.508790 seconds.
>>
>>
>> a=rand(50,150,500*p);
>> tic;ppeval('mycorr',a);toc
Elapsed time is 0.625101 seconds.
>> a=rand(50,150,500*p);
>> tic;ppeval('mycorr',a);toc
Elapsed time is 0.978677 seconds.
>>
>>
>> a=rand(100,150,250*p);
>> tic;ppeval('mycorr',a);toc
Elapsed time is 0.503163 seconds.
>> a=rand(100,150,250*p);
>> tic;ppeval('mycorr',a);toc
Elapsed time is 0.561599 seconds.
>>
```

One point to note is that the serial computation is significantly slower if the return variable, `ret`, is not pre-allocated. However, if it is pre-allocated, then the serial computation and parallel computation take roughly the same amount of time (perhaps the serial computation is slightly faster).

In practice, however, the data does not come in this nicely packaged form (i.e. a 3D matrix) but rather it comes in a 2D form that must be processed into this packaged form. In addition, not all TCs have the same number of single cells. For this reason, a simple parallel algorithm is not quite so easy to implement. To get around this, we encode the number of cells in each TC in the bottom-right corner of the data matrix in the following altered function, *mycorr_variable_size*.



```
1 function [ret] = mycorr_variable_size(data)
2
3 [num_rows num_cols]=size(data);
4 temp=num_rows;
5 if (data(num_rows,1)==0)
6     num_rows=data(num_rows, num_cols);
7     data(temp, num_cols) = 0;
8 end
9
10 sum_x = sum(data);
11 product = data'*data;
12 sum_x_squared = diag(product)';
13
14 standard = sqrt(num_rows * sum_x_squared - sum_x .* sum_x);
15
16 ret = (num_rows * product - sum_x' * sum_x) ./ (standard' * standard);
17
18
19
```

Furthermore, the data conversion from raw form to packaged form is accomplished by the following function, *input_data* (details of the conversion from raw data to the package form are not important for this discussion, but the function is included here for the sake of completeness).

```

Editor - C:\Documents and Settings\Oaz Nir\Desktop\Classes
File Edit Text Go Cell Tools Debug Desktop Window Help
[Icons]
1.0 1.1 x % %
1 function data = input_data(raw_data, type)
2
3 [num_rows num_cols]=size(raw_data);
4
5 num_classes = length(unique(type));
6
7 maximum_class_size = 0;
8 for i = 1:num_classes
9     k = sum(type == i);
10    if (k > maximum_class_size)
11        maximum_class_size = k;
12    end
13 end
14
15 for i = 1:num_classes
16     clear temp;
17     index = (type == i);
18     k = sum(index);
19     temp = raw_data(index,:);
20     temp((k+1):maximum_class_size,:)=0;
21     temp(maximum_class_size,num_cols) = k;
22     data(:, :, i) = temp;
23 end
24

```

Now, we can write serial and parallel functions that start with the raw data (i.e. the data as it is provided to us from the image processing software), package it, and then perform the correlation computation. In the case of the parallel computation, we also include *ppback* and *ppfront* commands to move the data to the server.

Serial

```

Editor - C:\Documents and Settings\Oaz Nir\Desktop\Classes\Parallel
File Edit Text Go Cell Tools Debug Desktop Window Help
[Icons]
1.0 1.1 x % %
1 function [ret] = ser(raw_data, type)
2 tic;
3 data = input_data(raw_data, type);
4 toc
5 [n m p] = size(data);
6 %parallel version
7 tic;
8 %data = ppback(data);
9 %toc
10 %tic;
11 %ret = ppeval('mycorr_variable_size', data);
12 %toc
13 %tic;
14 %ret = ppfront(ret);
15 %toc
16 %serial version
17 tic;
18 ret=zeros(m,n,p);
19 for i = 1:p
20     ret(:, :, i) = mycorr_variable_size(data(:, :, i));
21 end
22 toc
23
24

```

Parallel

```

Editor - C:\Documents and Settings\Oaz Nir\Desktop\Classes\Parallel
File Edit Text Go Cell Tools Debug Desktop Window Help
[Icons]
1.0 1.1 x % %
1 function [ret] = par(raw_data, type)
2 tic;
3 data = input_data(raw_data, type);
4 toc
5 [n m p] = size(data);
6 %parallel version
7 tic;
8 data = ppback(data);
9 toc
10 tic;
11 ret = ppeval('mycorr_variable_size', data);
12 toc
13 tic;
14 ret = ppfront(ret);
15 toc
16
17 %serial version
18 %tic;
19 %ret=zeros(m,n,p);
20 %for i = 1:p
21 %     ret(:, :, i) = mycorr_variable_size(data(:, :, i));
22 %end
23 %toc
24

```

Characteristic timings for these two functions are shown in two examples below.

```
Command Window
1 To get started, select MATLAB Help or Demos
>> raw_data=rand(20000,100);
>> type=floor(200*rand(20000,1))+1;
>>
>>
>> ser(raw_data,type);
Elapsed time is 5.218681 seconds.
Elapsed time is 0.213458 seconds.
>>
>>
>> par(raw_data,type);
Elapsed time is 5.267457 seconds.
Elapsed time is 8.800873 seconds.
Elapsed time is 0.631039 seconds.
Elapsed time is 6.179734 seconds.
>>

>> raw_data=rand(20000,100);
>> type=floor(200*rand(20000,1))+1;
>>
>>
>> ser(raw_data,type);
Elapsed time is 5.736294 seconds.
Elapsed time is 0.228404 seconds.
>>
>>
>> par(raw_data,type);
Elapsed time is 5.805299 seconds.
Elapsed time is 8.233302 seconds.
Elapsed time is 0.485387 seconds.
Elapsed time is 6.052744 seconds.
>> |
```

As before, note that the parallel computation time (.63 and .48 seconds in the two examples) is slower than the serial computation time (.21 and .22 seconds, respectively). Furthermore, the time to complete the *ppback* and *ppfront* commands is significant (in practice, we could mitigate this effect by ftp-ing the data to the server).

Conclusions

Overall, we have implemented a parallel framework for computing a matrix of linear correlations for all pairs of columns of morphological data, for multiple TCs. We would have hoped that the parallel version would have resulted in faster run-times than the serial version. As detailed above, however, this was not the case. (This effect was exacerbated when the time required to move data to/from the server was included as well).

Why weren't speed-ups realized? The reason is probably that the size of the data sets is too small for parallelization to achieve significant gains over serialization. In fact, speed-ups are realized for very large data sets (e.g. C, F, N each ~1000) because the server can handle larger data more smoothly than a typical PC. This is not necessarily due to parallelization, but rather due to hard drive usage (it could also be due to parallelization, but this cannot be immediately determined).

Overall, the project was a valuable excursion into the realm of parallelization. At the current scale, the computational problem is not large to require parallelization, but as we acquire data for larger populations of cells and for more TCs, or if we alter the computational algorithm so that it becomes more complex, parallelization will be useful. We now have the basic framework for this parallelization.