Lab Report

Computer specification:

1. Processor : AMD Ryzen 5 4000series
2. Core Numbers: 6
3. RAM : 8 GB
4. Clock speed: Usual speed : (1397 MHz, 87 MHz, 3992 MHz)
   During work: (2420 MHz (not all cores), 87 MHz, 3992 MHz)
   For parallel work: (2420 MHz (5 cores), 86 Mhz (3 of them) 87 MHz(2 of them), 3992 MHz)

Code Performance:

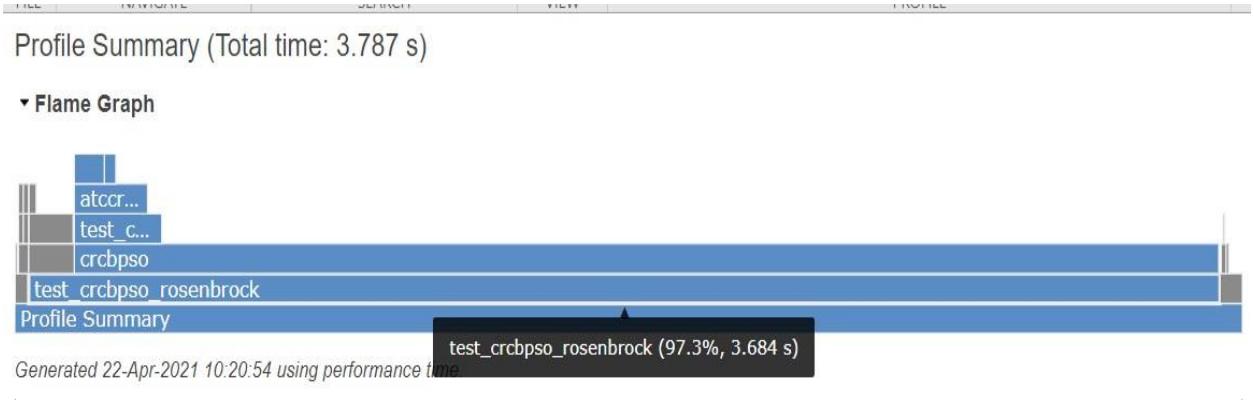
I have edited the function portion into code. The written function is generalized Rosenbrock. I have also written six hump camel function. There are two versions of codes: one with parallel computing and the other one is normal MatLab code i.e. without parallel computing. For finding out the elapsed time, profiling is used. Normal code takes less time on average than the code with parallel computing. Also, in both codes, 'crcbpso.m' takes most of the time for self-use. There are three versions of codes: the first one is by defining the variables, the second one is using one variable and vectorizing this variable in the function, and the last one is with counting variable for iterating numbers.

Normal Code performance:

First code, without using vectorization:

```
for lpc = 1:(nrows-1)
    if validPts(lpc)
    % Only the body of this block should be replaced for different fitness
    % functions
        x = xVec(lpc);
        xnext = xVec(lpc + 1);
        %fitVal(lpc) = sum(100 .* (x(2:end) - x(1:end-1).^2).^2 +(x(1:end-1) - 1).^2);
        fitVal(lpc) = sum(100 .* (xnext - x.^2).^2 +(x - 1).^2);
        %fitVal(lpc) =4*x(1)^2 - 2.1*x(1)^4 + (x(1)^6)/3 + x(1) * x(2) - 4*x(2)^2 + 4*x(2)^4;
    end
end
```

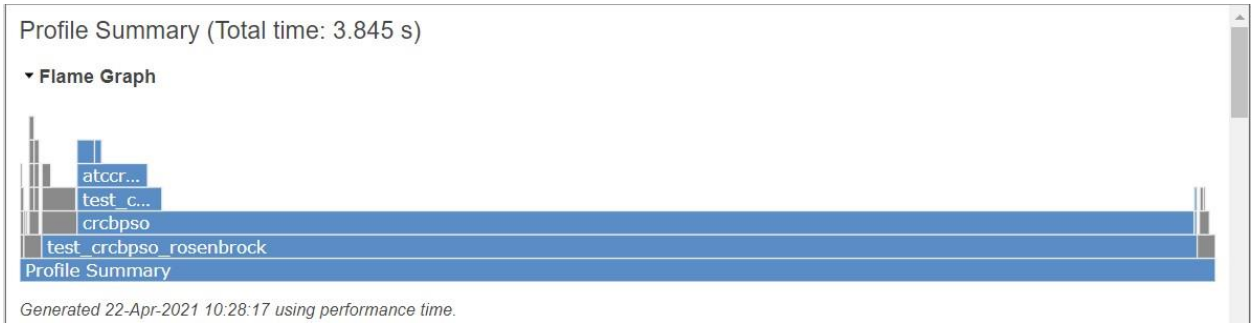
It takes nearly 3.787s.



Second code, using vectorization:

```
for lpc = 1:(nrows-1)
    if validPts(lpc)
    % Only the body of this block should be replaced for different fitness
    % functions
        x = xVec(lpc);
        %xnext = xVec(lpc + 1);
        fitVal(lpc) = sum(100 .* (x(2:end) - x(1:end-1).^2).^2 +(x(1:end-1) - 1).^2);
        %fitVal(lpc) = sum(100 .* (xnext - x.^2).^2 +(x - 1).^2);
        %fitVal(lpc) =4*x(1)^2 - 2.1*x(1)^4 + (x(1)^6)/3 + x(1) * x(2) - 4*x(2)^2 + 4*x(2)^4;
    end
end
```

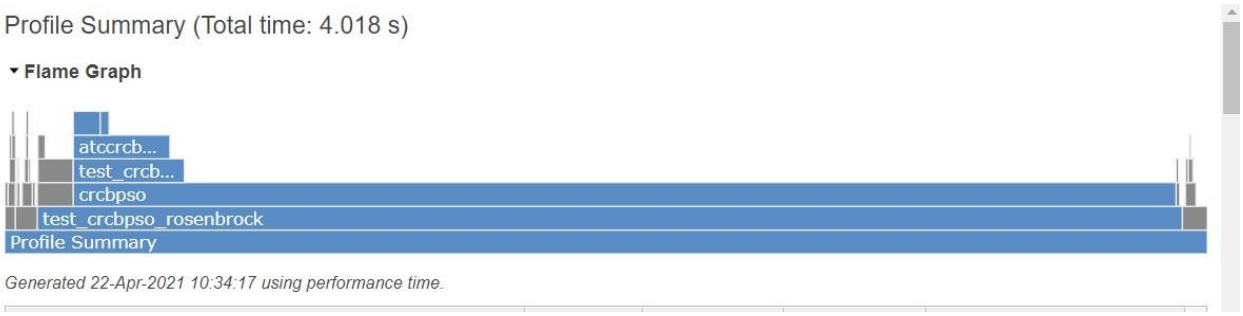
It takes nearly 3.845s.



I tried another code by numbering the x:

```
for lpc = 1:(nrows-1)
    if validPts(lpc)
    % Only the body of this block should be replaced for different fitness
    % functions
        x(lpc) = xVec(lpc);
        %xnext = xVec(lpc + 1);
        fitVal(lpc) = sum(100 .* (x(2:end) - x(1:end-1).^2).^2 +(x(1:end-1) - 1).^2);
        %fitVal(lpc) = sum(100 .* (xnext - x.^2).^2 +(x - 1).^2);
        %fitVal(lpc) =4*x(1)^2 - 2.1*x(1)^4 + (x(1)^6)/3 + x(1) * x(2) - 4*x(2)^2 + 4*x(2)^4;
    end
end
```

It takes longer time as expected. But the values of ‘realCoord’ and other values are exactly same.



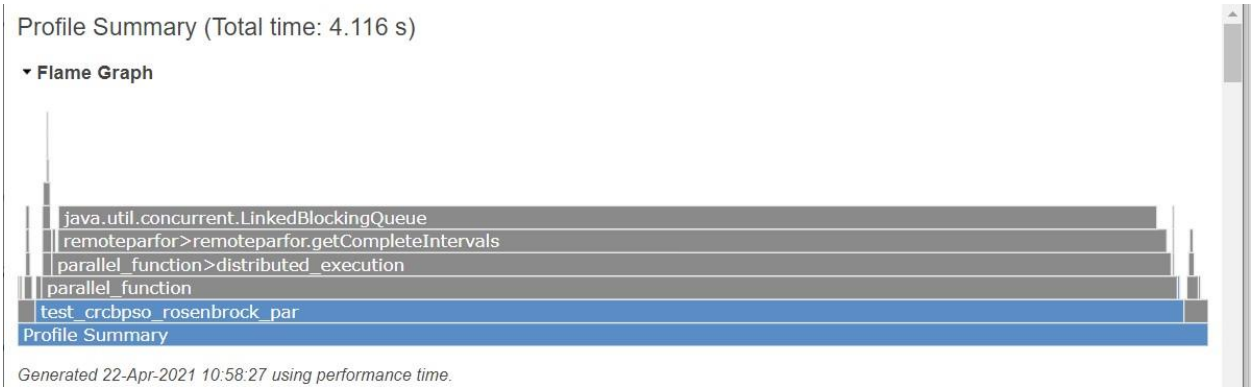
Parallel Code performance:

First code, without vectorization:

```
for lpc = 1:(nrows-1)
    if validPts(lpc)
    % Only the body of this block should be replaced for different fitness
    % functions
        x = xVec(lpc);
        xnext = xVec(lpc + 1);
        %fitVal(lpc) = sum(100 .* (x(2:end) - x(1:end-1).^2).^2 +(x(1:end-1) - 1).^2);
        fitVal(lpc) = sum(100 .* (xnext - x.^2).^2 +(x - 1).^2);
        %fitVal(lpc) =4*x(1)^2 - 2.1*x(1)^4 + (x(1)^6)/3 + x(1) * x(2) - 4*x(2)^2 + 4*x(2)^4;
    end
end

%Return real coordinates if requested
```

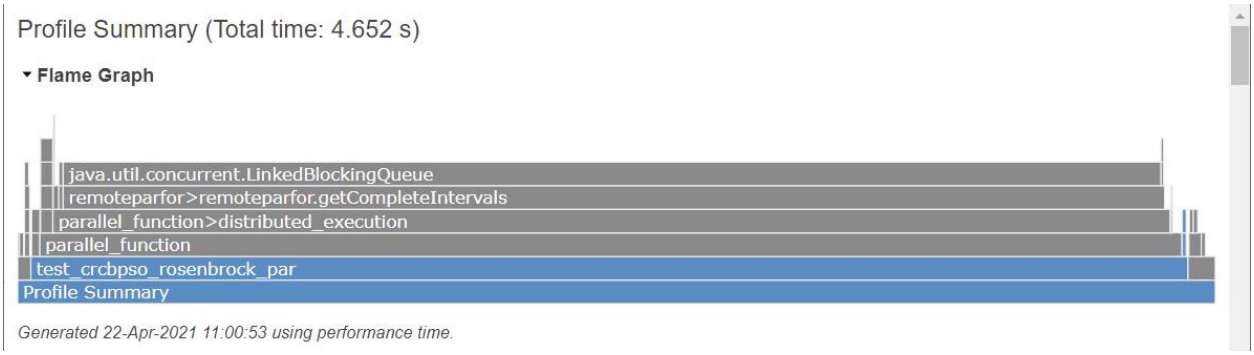
It takes 4.116s.



Second code, with vectorization:

```
for lpc = 1:(nrows-1)
    if validPts(lpc)
    % Only the body of this block should be replaced for different fitness
    % functions
        x = xVec(lpc);
        %xnext = xVec(lpc + 1);
        fitVal(lpc) = sum(100 .* (x(2:end) - x(1:end-1).^2).^2 +(x(1:end-1) - 1).^2);
        %fitVal(lpc) = sum(100 .* (xnext - x.^2).^2 +(x - 1).^2);
        %fitVal(lpc) =4*x(1)^2 - 2.1*x(1)^4 + (x(1)^6)/3 + x(1) * x(2) - 4*x(2)^2 + 4*x(2)^4;
    end
end
```

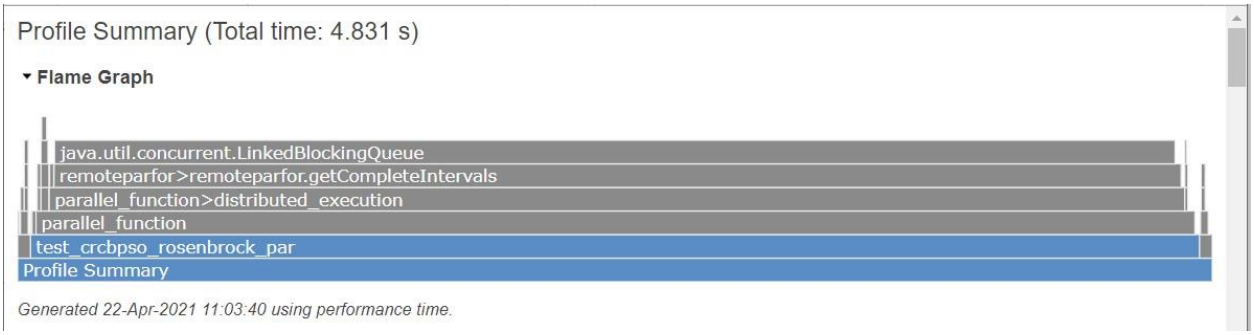
It takes 4.652s.



Third code:

```
for lpc = 1:(nrows-1)
    if validPts(lpc)
    % Only the body of this block should be replaced for different fitness
    % functions
        x(lpc) = xVec(lpc);
        %xnext = xVec(lpc + 1);
        fitVal(lpc) = sum(100 .* (x(2:end) - x(1:end-1).^2).^2 +(x(1:end-1) - 1).^2);
        %fitVal(lpc) = sum(100 .* (xnext - x.^2).^2 +(x - 1).^2);
        %fitVal(lpc) =4*x(1)^2 - 2.1*x(1)^4 + (x(1)^6)/3 + x(1) * x(2) - 4*x(2)^2 + 4*x(2)^4;
    end
end
```

It takes 4.831 s.



Calculation:

Rosenbrock function is

$$f(x) = 100(x_2 - x_1^2)^2 + (x_1 - 1)^2$$

Taking derivative of it, we get

$$f'(x) = \begin{pmatrix} \frac{\partial f(x)}{\partial x_1} \\ \\ \frac{\partial f(x)}{\partial x_2} \end{pmatrix} = \begin{pmatrix} 400(x_2 - x_1^2)x_1 + 2(x_1 - 1) \\ 200(x_2 - x_1^2) \end{pmatrix}$$

Taking second derivative or Hessian of the function, we get:

$$f''(x) = \begin{pmatrix} \frac{\partial^2 f(x)}{\partial x_1^2} & \frac{\partial^2 f(x)}{\partial x_2 \, \partial x_1} \\ \\ \frac{\partial^2 f(x)}{\partial x_1 \, \partial x_2} & \frac{\partial^2 f(x)}{\partial x_2^2} \end{pmatrix} = \begin{pmatrix} -400x_2 + 1200x_1^2 + 2 & -400x_1 \\ -400x_1 & 200 \end{pmatrix}$$

From first derivative, as f ’(x) = 0:

$$200(x_2 - \ x_1^2) = 0$$

And, using these into the first line of f ‘ (x),

$$x_1 - 1 = 0$$

$$x_1 = 1$$

Again, $x_2 = 1$, using x_1.

Thus, global minimizer is {1,1}.

Using these value into second derivative and taking determinent of the matrix, f’(x) greater than 0. Thus, this is the global minimizer. Using these values in function, it gives the result 0.

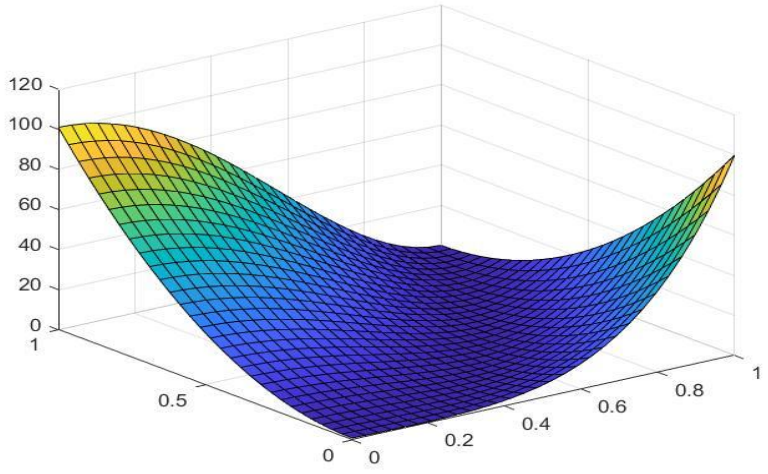


Figure: Rosenbrock function

Result:

For both normal and parallel code, all of the results are equivalent if the number of dimensions and parameters are the same. All these codes gave the same global minimum value ‘0’ and the minimizer values are the same. Mathematically, the global minimum value should be zero and the minimizer value {1,1} for two-dimension. But minimizer values are not same as codes.

In the case of the six hump camel function, all the minimizer and minimum values are equal in all cases.

Conclusion:

The most preferrable choise is the first code for both normal and parallel codes.