

Forum:[Hack Forums](#)

Thread: [Help Optimizing Code](#)

n00b.ca - July 20, 2020, 2:22 pm

I have the following code below. I'm having trouble moving away from the rigid for loops and optimizing my code for a greater performance benefit. I've tried using list comprehension but did not experience any more success.

Could anyone help potentially vectorize this code or suggest improvements?

Maxie - July 20, 2020, 2:32 pm

I lack insight both in Python and what the algorithm is supposed to do. I suppose it's a ML algorithm.

I would do some measurements to see which parts of the code is taking the most time. I would guess that the matrix inversion is expensive.

If you can walk us through the code line for line, we might be able to figure out which can be moved around as well.

You might also want to use a compiler that gets this to a lower level language, I think CPython would do the trick?

n00b.ca - July 20, 2020, 3:09 pm

(Jul 20, 2020 02:32 PM)Maxie Wrote: [\[CITE\]](#) I lack insight both in Python and what the algorithm is supposed to do. I suppose it's a ML algorithm.

I would do some measurements to see which parts of the code is taking the most time. I would guess that the matrix inversion is expensive.

If you can walk us through the code line for line, we might be able to figure out which can be moved around as well.

You might also want to use a compiler that gets this to a lower level language, I think CPython would do the trick?

Yeah it's part of a ML algorithm.

I'm currently using numba which helps boost performance but I'm trying to vectorize the for loops so I can use a drop in GPU accelerated library like cupy.

You would be right in saying that the 576x576 matrix inversion is expensive.

The basics of this code is essentially determining if the minimum tolerance value is greater than the current tau value to help establish fault status. The reason why it's taking so long is the number of times it has to iterate through. We have tau which goes from 1 to 121 and within that we have Ptrain which iterates to 3800 and so on forth.

Maxie - July 20, 2020, 3:48 pm

(Jul 20, 2020 03:09 PM)n00b.ca Wrote: [\[CITE\]](#) Yeah it's part of a ML algorithm.

I'm currently using numba which helps boost performance but I'm trying to vectorize the for loops so I can use a drop in GPU accelerated library like cupy.

You would be right in saying that the 576x576 matrix inversion is expensive.

The basics of this code is essentially determining if the minimum tolerance value is greater than the current tau value to help establish fault status. The reason why it's taking so long is the number of times it has to iterate through. We have tau which goes from 1 to 121 and within that we have Ptrain which iterates to 3800 and so on forth.

If it doesn't mess up the math, moving the i-loop out should at least provide some cache memory optimization. The K-pop should already be vectorized.

I suspect that numpy already is pretty optimized.

I'd look into optimizations of the algorithm perhaps or see if there's any options of compiling the python code instead of interpreting it.

There is also compilers which perform the vectorization for you, at least in C - and considering that python is mainly used for ML, I would guess that similar exist for python.

n00b.ca - July 20, 2020, 3:58 pm

(Jul 20, 2020 03:48 PM)Maxie Wrote: [\[CITE\]](#) If it doesn't mess up the math, moving the i-loop out should at least provide some cache memory optimization. The K-pop should already be vectorized.

I suspect that numpy already is pretty optimized.

I'd look into optimizations of the algorithm perhaps or see if there's any options of compiling the python code instead of interpreting it.

There is also compilers which perform the vectorization for you, at least in C - and considering that python is mainly used for ML, I would guess that similar exist for python.

How do you mean by moving the i-loop out? I only need the i loop variable to compute the position of the Traindata for the f_sensor_n.

Maxie - July 20, 2020, 4:03 pm

(Jul 20, 2020 03:58 PM)n00b.ca Wrote: [\[CITE\]](#) How do you mean by moving the i-loop out? I only need the i loop variable to compute the position of the Traindata for the f_sensor_n.

Change the position of the loop with the i-variable and the n-variable.

If you access an array, in memory the place $a[n][i]$ and $a[n][i+1]$ is closer than $a[n][i]$ and $a[n+1][i]$ which should in theory result in less cache misses if you access i then i+1 in the array.

Might help, might not. As far as I could see, they should be mathematically the same.

flatline - July 20, 2020, 4:16 pm

(Jul 20, 2020 03:48 PM)Maxie Wrote: [\[CITE\]](#) I'd look into optimizations of the algorithm perhaps or see if there's any options of compiling the python code instead of interpreting it.

There is also compilers which perform the vectorization for you, at least in C - and considering that python is mainly used for ML, I would guess that similar exist for python.

These two (one really) steps will make it much faster. Plenty of python compilers include auto-vectorization and some are made with SIMD support in mind.

Case in point:

<http://serge-sans-paille.github.io/pythran-stories/costless-abstraction-with-pythran-broadcasting.html>

From article:

Quote:numpy 100 loops, best of 3: 1.86 ms per loop
cython+simd 1000 loops, best of 3: 207 µs per loop
pythran+simd 1000 loops, best of 3: 246 µs per loop
numpy 10 loops, best of 3: 80.7 ms per loop
cython+simd 100 loops, best of 3: 5.36 ms per loop
pythran+simd 100 loops, best of 3: 5.96 ms per loop
numpy 1 loop, best of 3: 250 ms per loop
cython+simd 10 loops, best of 3: 21.4 ms per loop
pythran+simd 10 loops, best of 3: 21.5 ms per loop
Use Pythran or Cython.

n00b.ca - July 20, 2020, 5:26 pm

(Jul 20, 2020 04:03 PM)Maxie Wrote: [\[CITE\]](#) Change the position of the loop with the i-variable and the n-variable.

If you access an array, in memory the place $a[n][i]$ and $a[n][i+1]$ is closer than $a[n][i]$ and $a[n+1][i]$ which should in theory result in less cache misses if you access i then $i+1$ in the array.

Might help, might not. As far as I could see, they should be mathematically the same.
This didn't help speedup the code unfortunately.

(Jul 20, 2020 04:16 PM)flatline Wrote: [\[CITE\]](#) These two (one really) steps will make it much faster. Plenty of python compilers include auto-vectorization and some are made with SIMD support in mind.

Case in point:

<http://serge-sans-paille.github.io/pythran-stories/costless-abstraction-with-pythran-broadcasting.html>

From article:

Use Pythran or Cython.

Interesting, I've tried implementing this but I'm having trouble getting the syntax right for the code above. Think you can take a crack at it?

flatline - July 20, 2020, 6:31 pm

(Jul 20, 2020 05:26 PM)n00b.ca Wrote: [\[CITE\]](#) This didn't help speedup the code unfortunately.

Interesting, I've tried implementing this but I'm having trouble getting the syntax right for the code above. Think you can take a crack at it?

So I'm not sure exactly what the issue you're having is exactly but I'll have a crack at it once I get back to my PC. Also, I checked them out some more and pythran is probably better to use for this project since the script is already written and only one function.

For pythran you need to add a comment above the function definition with the variable types since C doesn't know what they are. Example:

Code #pythran tauLoop(int, int list, int list ... etc)

Then run Code pythran tauLoop.py to compile.

n00b.ca - July 20, 2020, 6:42 pm

(Jul 20, 2020 06:31 PM)flatline Wrote: [\[CITE\]](#) So I'm not sure exactly what the issue you're having is exactly but I'll have a crack at it once I get back to my PC. Also, I checked them out some more and pythran is probably better to use for this project since the script is already written and only one function.

For pythran you need to add a comment above the function definition with the variable types since C doesn't know what they are. Example:

Code #pythran tauLoop(int, int list, int list ... etc)

Then run

Code pythran tauLoop.py to compile.

So the trouble I'm having has to do with the comment above the function definition with the variable types.

This is what I have right now: Code #pythran export tauLoop(int, float64[:, :], float64[:, :], float64[:, :], int, float64[:, :, :], int)

but I get the error that the input file does not match Pythran's constraints and that there is an invalid spec at this function definition.

flatline - July 20, 2020, 8:06 pm

(Jul 20, 2020 06:42 PM)n00b.ca Wrote: [\[CITE\]](#) So the trouble I'm having has to do with the comment above the function definition with the variable types.

This is what I have right now:

Code #pythran export tauLoop(int, float64[:, :], float64[:, :], float64[:, :], int, float64[:, :, :], int)

but I get the error that the input file does not match Pythran's constraints and that there is an invalid spec at this function definition.

Just tried it out on my machine.

For me, the comment was fine. Maybe you're missing a dependency or something?

Unfortunately, numpy.linalg.inv doesn't seem to be supported by Pythran ?

<https://pythran.readthedocs.io/en/latest/SUPPORT.html>

Workaround would be to basically do this without numpy.linalg.inv. Did some digging on stack overflow and found this:

<https://stackoverflow.com/questions/32114054/matrix-inversion-without-numpy>

Which looks like it would work, and may be faster since apparently numpy.linalg.inv recreates the identity matrix a couple times while executing.

n00b.ca - July 20, 2020, 8:27 pm

(Jul 20, 2020 08:06 PM)flatline Wrote: [\[CITE\]](#) Just tried it out on my machine.

For me, the comment was fine. Maybe you're missing a dependency or something?

Unfortunately, numpy.linalg.inv doesn't seem to be supported by Pythran ?

<https://pythran.readthedocs.io/en/latest/SUPPORT.html>

Workaround would be to basically do this without numpy.linalg.inv. Did some digging on stack overflow and found this:

<https://stackoverflow.com/questions/32114054/matrix-inversion-without-numpy>

Which looks like it would work, and may be faster since apparently numpy.linalg.inv recreates the identity matrix a couple times while executing.

No luck. I'm unable to use multiple functions when using Pythran. On top of this, those solutions

error out due to some shape mismatch. Is there anyway we can use a GPU to accelerate this program? I've been looking into cupy but haven't figured out a way to make it efficient on the GPU just yet.