Subject: Re: fast convolving question
Posted by Chris[5] on Fri, 30 May 2008 10:15:44 GMT
View Forum Message <> Reply to Message

On May 29, 9:44 pm, rog...@googlemail.com wrote:
> On May 29, 9:47 pm, Chris <cnb4s...@gmail.com> wrote:
>
>>> But, did you see the error, why:
>
>>> else (conv)[(i=indarray2[0:sm-1])]=kernel##mat(indsmall+ind(i))
>
>>> only works for the first ind(i=0) and then repeats it by using always
>>> ind(0) instead of ind(i)?
>
>>> It this a bug or error in code?
>
>> Both kernel and mat(indsmall+ind(i)) are vectors of length sm. When
>> you matrix multiply them, the answer is single number. So you are just
>> asking to fill in conv with the same number at every element given by
>> i.
>
> Ah, ok - THANKS. But how can I fix this?
>
> If I would:
>
> (conv)[(i=0)]=kernel##mat(indsmall+ind(i)) ->result right
> (conv)[(i=1)]=kernel##mat(indsmall+ind(i)) ->result right
> and so on..
>
> But if I would:
>
> (conv)[(i=indarray2[0:sm-1])]=kernel##mat(indsmall+ind(i))
>
> or
>
> (conv)[(i=indgen(sm))])]=kernel##mat(indsmall+ind(i))
>
> then it does what you said. That makes no sense to me. How can I do it
> parallel for all indices given by i?
>
> If the code would work, its 100 times faster than fft - and that is
> the aim.
>
> Please help again.
>
> Thanks and best regards
>
> Christian

Let me preface this by saying I don't fully understand your code, so I'm not sure if this is correct or helpful. But here goes:

1)matrix(indsmall)=10,000 element vector
2)matrix(indsmall+ind(0 or 1 or...)= 10,000 element vector, different from above and different for 0,1,etc
3)matrix(indsmall+ind)=10,000 element vector different from all of these

the loop fills in every pixel of convol by matrix multiplying by one of the many matrices in group 2. The non-loop fills in every pixel of convol by matrix multiplying by the SAME matrix in 3. Hence the difference.

How do you fix it? Hmmm not sure. If you want to do this by matrix multiplication, you need to matrix multiply kernel (10,000 elements) by a 10,000 by 10,000 matrix. Each row (or column, not sure) of the matrix needs to be matrix(indsmall+ind(i)), where i is different for each row/column. I wouldn't know how to construct this larger array without looping, though I'm sure there's a way.

Note, however, that you should NOT be thinking that a non-looping method is really going to beat the fft method based on your current program. The non-loop does the necessary amount of arithmetic for filling in one conolution pixel (multiplying 10,000 elements by 10,000 elements and adding). It then copies this 10,000 times. So while its 100x faster now, if you were to do 10,000 unique pixel calculations (which you'll have to eventually do), the method will be 100x slower.

Also, your fft and loop outputs are different. In fact, the outputs aren't even the same dimension. Are you sure that these other two are doing what you want? Using convol (and only convolving with kernel[0:98,0:98], since the kernel needs to be smaller than the image for some reason) gives yet a third answer.

Finally, I'm really skeptical that you will figure out a method significantly faster than blk_con or convol. Convol is 70x faster than your fft method with 400x400 arrays, and 90x faster with 600x600 arrays. Convolution is an expensive process that you may have to live with for 2 large arrays (though I haven't seen many applications where the kernel needs to be so large). Also remember that, if you are using an FFT method, the input arrays really need to be powers of 2 (64 or 128, not 100) for the fft speed benefit. Padding a 100x100 image with zeros up to a 128x128 image will make the fourier transform much more efficient.

Anyways, sorry this doesn't directly fix the bug in your code.

Hopefully, though, you can figure out a way to deal with an intrinsically expensive computation (or perhaps find a way to use a smaller kernel??)

Chris