Subject: efficient kernel or masking algorithm ?
Posted by Richard Tyc on Wed, 29 Nov 2000 08:00:00 GMT
View Forum Message <> Reply to Message

I need to apply a smoothing type kernel across an image, and calculate the standard deviation of the pixels masked by this kernel.

ie. lets say I have a 128x128 image. I apply a 3x3 kernel (or simply a mask) starting at [0:2,0:2] and use these pixels to find the standard deviation for the center pixel [1,1] based on its surrounding pixels, then advance the kernel etc deriving a std deviation image essentially.
I can see myself doing this 'C' like with for loops but does something exist for IDL to do it better or more efficiently ?

Rich

Subject: Re: efficient kernel or masking algorithm ?
Posted by John-David T. Smith on Thu, 30 Nov 2000 08:00:00 GMT
View Forum Message <> Reply to Message

Craig Markwardt wrote:
>
> "J.D. Smith" <jdsmith@astro.cornell.edu> writes:
> ...
>> While I'm on the gripe train, why shouldn't we be able to consistently
>> perform operations along any dimension of an array we like with relevant
>> IDL routines. E.g., we can total along a single dimension. All due
>> respect to Craig's CMAPPLY function, but some of these things should be
>> much faster. Resorting to summed logarithms for multiplication is not
>> entirely dubious, but why shouldn't we be able to say:
> ...
>
> Agree! Agree! Agree! For once we are griping in synchrony :-)
>
> These are exactly the kinds of operations that would be enhanced by
> vectorization, but they can't as IDL stands now.
>
> By the way, CMAPPLY doesn't use summed logarithms any more. It uses
> my bestest algorithm that came out of the recent newsgroup discussion.
>

Ahh yes, multiplication by decimation. Must have missed that one. I simply read the comment in your code without looking at the details:

;; *** Multiplication
(newop EQ '*'): begin ;; Multiplication (by summation of logarithms)

Did you do some time testing and find all that shifted indexing was really faster than the logarithm? This I suspect will be very architecture dependent. Looks neat though.

Maybe I'll write up the 100 lines of C it would take for a shared library to do dimensional multiply, sum, add, median, min, max, and, or, mode, variance, etc., and send it to RSI. The problem with all of this specific "vector-aware" coding, is that it reveals a dirty secret of IDL's. It was built to do some vector operations fast, but was never a truly generic vector language like APL or J.

But sinceDavid hasn't written a book on either of those, we'll just have to slog through with what we have. <insert disremembered sarcasm code>

JD

Subject: Re: efficient kernel or masking algorithm ?
Posted by Craig Markwardt on Thu, 30 Nov 2000 08:00:00 GMT
View Forum Message <> Reply to Message

"J.D. Smith" <jdsmith@astro.cornell.edu> writes:
...
> While I'm on the gripe train, why shouldn't we be able to consistently
> perform operations along any dimension of an array we like with relevant
> IDL routines. E.g., we can total along a single dimension. All due
> respect to Craig's CMAPPLY function, but some of these things should be
> much faster. Resorting to summed logarithms for multiplication is not
> entirely dubious, but why shouldn't we be able to say:
...

Agree! Agree! Agree! For once we are griping in synchrony :-)

These are exactly the kinds of operations that would be enhanced by vectorization, but they can't as IDL stands now.

By the way, CMAPPLY doesn't use summed logarithms any more. It uses my bestest algorithm that came out of the recent newsgroup discussion.

Craig

--

Craig B. Markwardt, Ph.D. EMAIL: craigmnet@cow.physics.wisc.edu
Astrophysics, IDL, Finance, Derivatives | Remove "net" for better response
----

Subject: Re: efficient kernel or masking algorithm ?
Posted by Craig Markwardt on Fri, 01 Dec 2000 08:00:00 GMT
View Forum Message <> Reply to Message

"J.D. Smith" <jdsmith@astro.cornell.edu> writes:
> Ahh yes, multiplication by decimation. Must have missed that one. I
> simply read the comment in your code without looking at the details:
>
>
> ;; *** Multiplication
> (newop EQ '*'): begin ;; Multiplication (by summation of logarithms)
>
> Did you do some time testing and find all that shifted indexing was
> really faster than the logarithm? This I suspect will be very
> architecture dependent. Looks neat though.

The summation of logarithms was never very satisfactory. It never handled zeroes very well. Since there can be multiplication by negative numbers you really had to do a complex logarithm. All of these conversions made it quite slow.

> Maybe I'll write up the 100 lines of C it would take for a shared
> library to do dimensional multiply, sum, add, median, min, max, and, or,
> mode, variance, etc., and send it to RSI. The problem with all of this
> specific "vector-aware" coding, is that it reveals a dirty secret of
> IDL's. It was built to do some vector operations fast, but was never a
> truly generic vector language like APL or J.

Yorick is very similar to IDL, but "better" in a lot of ways. One thing it has is the ability to write array indices which are functions. So, if you had a 2 dimensional array, you could get the MIN along one dimension or the other by doing this:

array[min,*] or array[*,min] [syntax not totally correct]

This is a very compact and meaningful notation. It has all sorts of functions that can go in there, like cumulative sum, first difference, etc.

Now, people who want to do complicated sliding variances and medians, that's a much harder thing to accomplish with a vector language. We would need a fairly sophisticated "convolution" routine, which might be hard to optimize.

Craig

--
Craig B. Markwardt, Ph.D. EMAIL: craigmnet@cow.physics.wisc.edu