Subject: Re: Accelerating a one-line program doing matrix multiplication
Posted by natha on Mon, 27 Sep 2010 13:31:58 GMT
View Forum Message <> Reply to Message

You can use the TEMPORARY function if you can set the input to
undefined...
When you do [[v1],[v2],[v3]] you are duplicating data. v1, v2 and v3
are copied and you are not conserving memory.

You could try:
RETURN, [[TEMPORARY(v1)],[TEMPORARY(v2)],[TEMPORARY(v3)]] # vc +
REBIN(v0, SIZE(vc, /DIMENSIONS))

Cheers,
nata

Subject: Re: Accelerating a one-line program doing matrix multiplication
Posted by on Tue, 28 Sep 2010 06:17:43 GMT
View Forum Message <> Reply to Message

On 27 Sep., 15:31, nata <bernat.puigdomen...@gmail.com> wrote:
> You can use the TEMPORARY function if you can set the input to
> undefined...
> When you do [[v1],[v2],[v3]] you are duplicating data. v1, v2 and v3
> are copied and you are not conserving memory.
>
> You could try:
> RETURN, [[TEMPORARY(v1)],[TEMPORARY(v2)],[TEMPORARY(v3)]] # vc +
> REBIN(v0, SIZE(vc, /DIMENSIONS))
>
> Cheers,
> nata

Thanks nata.

v0, v1, v2 and v3 are each of them 3-element vectors. I can add that
but, as I understand it, it will only save the place of 12 floating
values in memory (48 bytes?).

But I am happy that you did not see any other obvious thing. I started
feeling depressed seeing that I am not being able to improve this
single line of code... maybe it is ok, and the whole thing is just
slow...? ahh.

Subject: Re: Accelerating a one-line program doing matrix multiplication

Posted by natha on Tue, 28 Sep 2010 13:10:12 GMT
View Forum Message <> Reply to Message

Concatenation is a very slow action in IDL and, if you are copying memory, the time of computation increases...
If v0, v1, v2 and v3 are each of them 3-element vectors then you will not see the difference. TEMPORARY function is great when you are copying large arrays. I think you can not improve your code because the problem is the matrix multiplication and you can not change that.
Try putting timers to see what's the time to compute each instruction.

```
tt=SYSTIME(/SEC)
aux=[[v1],[v2],[v3]]
PRINT, SYSTIME(/SEc)-tt

tt=SYSTIME(/SEC)
aux=aux # vc
PRINT, SYSTIME(/SEC)-tt
```

etc.

Cheers,
nata

On Sep 28, 2:17 am, Axel M <axe...@gmail.com> wrote:
> On 27 Sep., 15:31, nata <bernat.puigdomen...@gmail.com> wrote:
>
>> You can use the TEMPORARY function if you can set the input to
>> undefined...
>> When you do [[v1],[v2],[v3]] you are duplicating data. v1, v2 and v3
>> are copied and you are not conserving memory.
>
>> You could try:
>> RETURN, [[TEMPORARY(v1)],[TEMPORARY(v2)],[TEMPORARY(v3)]] # vc +
>> REBIN(v0, SIZE(vc, /DIMENSIONS))
>
>> Cheers,
>> nata
>
> Thanks nata.
>
> v0, v1, v2 and v3 are each of them 3-element vectors. I can add that
> but, as I understand it, it will only save the place of 12 floating
> values in memory (48 bytes?).
>
> But I am happy that you did not see any other obvious thing. I started
> feeling depressed seeing that I am not being able to improve this

> single line of code... maybe it is ok, and the whole thing is just
> slow...? ahh.

Subject: Re: Accelerating a one-line program doing matrix multiplication
Posted by Jeremy Bailin on Tue, 28 Sep 2010 15:31:07 GMT
View Forum Message <> Reply to Message

On Sep 27, 5:18 am, Axel M <axe...@gmail.com> wrote:
> Hi all,
>
> I wrote a one-line function to convert a list of points from "voxel
> coordinates" (image coordinates) to "real coordinates" (physical
> coordinates):
>
> ;input: the points "vc", the spatial origin of an image v0 and its x,
> y, and z orientation vectors (v1,v2,v3).
> FUNCTION vc2rc, v0,v1,v2,v3,vc
> RETURN, [[v1],[v2],[v3]] # vc + REBIN(v0, SIZE(vc, /DIMENSIONS))
> END
>
> For example, I give the image coordinate [8,1,0] and I want as output
> something like [34.25, 4.12, 0], indicating the location of this voxel
> in space. And the same thing but, instead of having one input point,
> having several millions.
>
> The function looks simple to me and it works great. BUT, for large
> images (e.g. 500x500x200 voxels), it is terribly slow and uses way too
> much memory... Am I doing something wrong, could I save speed
> somewhere? I guess there should be some way to accelerate it, but I am
> not able to see how...
>
> I also have the opposite function, in my opinion also too slow (though
> faster than the other)...
>
> FUNCTION rc2vc_round, v0,v1,v2,v3,rc
> RETURN, ROUND((rc - REBIN(v0, SIZE(rc, /DIMENSIONS))) ## INVERT([[v1],
> [v2],[v3]]))
> END
>
> I would be really grateful for any clue!

You can try adding /SAMPLE to the REBIN call, but I suspect that it's
not the bottleneck.

-Jeremy.

Subject: Re: Accelerating a one-line program doing matrix multiplication
Posted by rogass on Wed, 29 Sep 2010 06:57:27 GMT
View Forum Message <> Reply to Message

On 28 Sep., 15:10, nata <bernat.puigdomen...@gmail.com> wrote:
> Concatenation is a very slow action in IDL and, if you are copying
> memory, the time of computation increases...
> If v0, v1, v2 and v3 are each of them 3-element vectors then you will
> not see the difference. TEMPORARY function is great when you are
> copying large arrays. I think you can not improve your code because
> the problem is the matrix multiplication and you can not change that.
> Try putting timers to see what's the time to compute each instruction.
>
> tt=SYSTIME(/SEC)
> aux=[[v1],[v2],[v3]]
> PRINT, SYSTIME(/SEc)-tt
>
> tt=SYSTIME(/SEC)
> aux=aux # vc
> PRINT, SYSTIME(/SEC)-tt
>
> etc.
>
> Cheers,
> nata
>
> On Sep 28, 2:17 am, Axel M <axe...@gmail.com> wrote:
>
>
>
>> On 27 Sep., 15:31, nata <bernat.puigdomen...@gmail.com> wrote:
>
>>> You can use the TEMPORARY function if you can set the input to
>>> undefined...
>>> When you do [[v1],[v2],[v3]] you are duplicating data. v1, v2 and v3
>>> are copied and you are not conserving memory.
>
>>> You could try:
>>> RETURN, [[TEMPORARY(v1)],[TEMPORARY(v2)],[TEMPORARY(v3)]] # vc +
>>> REBIN(v0, SIZE(vc, /DIMENSIONS))
>
>>> Cheers,
>>> nata
>
>> Thanks nata.
>
>> v0, v1, v2 and v3 are each of them 3-element vectors. I can add that
>> but, as I understand it, it will only save the place of 12 floating
>> values in memory (48 bytes?).

>
>> But I am happy that you did not see any other obvious thing. I started
>> feeling depressed seeing that I am not being able to improve this
>> single line of code... maybe it is ok, and the whole thing is just
>> slow...? ahh.

Yes, and you can substitute some of your calls:

aux #= vc (makes no copy of aux as far as i know)
invert -> la_invert (much much speedier)
sometimes (replicate({temp:input},newsize)).(0) is faster then rebin
exchange ## with matrix_multiply

Cheers

CR

Subject: Re: Accelerating a one-line program doing matrix multiplication
Posted by on Wed, 29 Sep 2010 10:22:54 GMT
View Forum Message <> Reply to Message

On Sep 29, 8:57 am, chris <rog...@googlemail.com> wrote:
> On 28 Sep., 15:10, nata <bernat.puigdomen...@gmail.com> wrote:
>
>
>
>> Concatenation is a very slow action in IDL and, if you are copying
>> memory, the time of computation increases...
>> If v0, v1, v2 and v3 are each of them 3-element vectors then you will
>> not see the difference. TEMPORARY function is great when you are
>> copying large arrays. I think you can not improve your code because
>> the problem is the matrix multiplication and you can not change that.
>> Try putting timers to see what's the time to compute each instruction.
>
>> tt=SYSTIME(/SEC)
>> aux=[[v1],[v2],[v3]]
>> PRINT, SYSTIME(/SEc)-tt
>
>> tt=SYSTIME(/SEC)
>> aux=aux # vc
>> PRINT, SYSTIME(/SEC)-tt
>
>> etc.
>
>> Cheers,
>> nata
>

>> On Sep 28, 2:17 am, Axel M <axe...@gmail.com> wrote:
>
>>> On 27 Sep., 15:31, nata <bernat.puigdomen...@gmail.com> wrote:
>
>>>> You can use the TEMPORARY function if you can set the input to
>>>> undefined...
>>>> When you do [[v1],[v2],[v3]] you are duplicating data. v1, v2 and v3
>>>> are copied and you are not conserving memory.
>
>>>> You could try:
>>>> RETURN, [[TEMPORARY(v1)],[TEMPORARY(v2)],[TEMPORARY(v3)]] # vc +
>>>> REBIN(v0, SIZE(vc, /DIMENSIONS))
>
>>>> Cheers,
>>>> nata
>
>>> Thanks nata.
>
>>> v0, v1, v2 and v3 are each of them 3-element vectors. I can add that
>>> but, as I understand it, it will only save the place of 12 floating
>>> values in memory (48 bytes?).
>
>>> But I am happy that you did not see any other obvious thing. I started
>>> feeling depressed seeing that I am not being able to improve this
>>> single line of code... maybe it is ok, and the whole thing is just
>>> slow...? ahh.
>
> Yes, and you can substitute some of your calls:
>
> aux #= vc (makes no copy of aux as far as i know)
> invert -> la_invert (much much speedier)
> sometimes (replicate({temp:input},newsize)).(0) is faster then rebin
> exchange ## with matrix_multiply
>
> Cheers
>
> CR

Thanks a lot for all suggestions!

I have tried the different ideas. Unfortunately no improvements. Not
even the /SAMPLE in the rebin call helped, which I thought could make
a difference.

So I guess the matrix multiplication is just too computationally
intensive to get fast results.

What I also thought is that the method is fast enough when only a few

points are given, but very time-consuming when "vc" contains all coordinates of the images, and they are sorted like [[0,0,0], [0,0,1], [0, 0, 2], [0, 1, 0], ... [2, 2, 1], [2, 2, 2]]. Maybe there is some way to use this property of vc to accelerate it. Like multiplying the v1, v2, v3 by INDGEN and then somehow adding them smartly... I will give it a try.

Subject: Re: Accelerating a one-line program doing matrix multiplication
Posted by on Wed, 29 Sep 2010 10:48:51 GMT
View Forum Message <> Reply to Message

I have to admit that I did not understand this proposed use of REPLICATE:

>> sometimes (replicate({temp:input},newsize)).(0) is faster then rebin

But it brought me a related question in mind: does IDL have a "REPLICATE" function for vectors instead of scalar values? I am using REBIN, but REBIN is thought for more advanced uses and probably suboptimal for a "replicate-like" use... right?

Subject: Re: Accelerating a one-line program doing matrix multiplication
Posted by rogass on Wed, 29 Sep 2010 14:49:57 GMT
View Forum Message <> Reply to Message

On 29 Sep., 12:48, Axel M <axe...@gmail.com> wrote:
> I have to admit that I did not understand this proposed use of
> REPLICATE:
>
>>> sometimes (replicate({temp:input},newsize)).(0) is faster then rebin
>
> But it brought me a related question in mind: does IDL have a
> "REPLICATE" function for vectors instead of scalar values? I am using
> REBIN, but REBIN is thought for more advanced uses and probably
> suboptimal for a "replicate-like" use... right?

Yes, it has as I mentioned uncommented above:

sometimes (replicate({temp:input},newsize)).(0) is faster then rebin

-> this means:

```
IDL> a=findgen(3)
IDL> print,a
    0.000000      1.00000      2.00000
```

```
IDL> print, rebin(a,3,5)
      0.000000      1.00000      2.00000
      0.000000      1.00000      2.00000
      0.000000      1.00000      2.00000
      0.000000      1.00000      2.00000
      0.000000      1.00000      2.00000
IDL> print, (replicate({temp:a},5)).(0)
      0.000000      1.00000      2.00000
      0.000000      1.00000      2.00000
      0.000000      1.00000      2.00000
      0.000000      1.00000      2.00000
      0.000000      1.00000      2.00000
```

Regards

CR
